# Quantum Algorithms for Cryptanalysis and Quantum-safe Symmetric Cryptography

André Schrottenloher

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS
DEPUIS 1257

Inria

# THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

**Informatique**

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

## André Schrottenloher

Pour obtenir le grade de

**DOCTEUR de SORBONNE UNIVERSITÉ**

# Quantum Algorithms for Cryptanalysis and Quantum-safe Symmetric Cryptography

soutenue publiquement le 8 février 2021

devant le jury composé de :

| | | |
|---|---|---|
| María Naya-Plasencia | Inria de Paris | Directrice |
| André Chailloux | Inria de Paris | Co-directeur |
| Gregor Leander | Ruhr-Universität Bochum | Rapporteur |
| Bart Preneel | Katholieke Universiteit Leuven | Rapporteur |
| Henri Gilbert | ANSSI | Examinateur |
| Stacey Jeffery | QuSoft et CWI | Examinatrice |
| Antoine Joux | CISPA Helmholtz Center | Examinateur |
| Damien Vergnaud | Sorbonne Université | Examinateur |

*Wir müssen unser Dasein so* weit*, als es irgend geht, annehmen; alles, auch das Unerhörte, muß darin möglich sein. Das ist im Grunde der einzige Mut, den man von uns verlangt: mutig zu sein zu dem Seltsamsten, Wunderlichsten und Unaufklärbarsten, das uns begegnen kann.*

*Nous devons accepter notre existence aussi* largement *qu'il se peut ; tout, même l'inoui, doit y être possible. C'est au fond le seul courage que l'on exige de nous : être courageux envers ce qui, venant à nous, est le plus bizarre, le plus étonnant, le moins éclaircissable.*

Rainer Maria Rilke, *Briefe an einen jungen Dichter* [Ril89]

# Remerciements

Comment décrire ces trois ans[1] de recherche au sein de l'équipe SECRET[2] ? N'ayons pas peur des mots, et puisque les concerné(e)s sauront m'excuser d'un tel lyrisme, je lance : ce fut un privilège. Aussi trois pages de remerciements ne donneront-elles qu'une approximation éhontée de ce que je dois à toutes celles et ceux qui m'ont aidé, soutenu, encouragé ou que j'ai tout simplement croisés sur ce parcours. Je vais en citer une partie ; il ne manquera pas seulement une personne, mais des dizaines, et je leur présente d'avance mes excuses : vous êtes aussi ici, entre les lignes.

En premier lieu, merci à María pour avoir dirigé cette thèse avec une implication, une disponibilité et un enthousiasme qui rendirent ma vie de thésard somme toute assez facile. Merci à André pour être parvenu à m'inculquer un peu de calcul et de preuves quantiques[3]. Merci à Henri et Ludovic d'en avoir effectué le suivi.

Thanks to Henri Gilbert, Stacey Jeffery, Antoine Joux, Gregor Leander, Bart Preneel and Damien Vergnaud for accepting to be part of the jury, and special thanks to Gregor and Bart for reviewing the manuscript[4].

Merci à tous les membres de l'équipe SECRET[5] que j'ai rencontrés entre mars 2017 et février 2021[6]: Anaïs, André, Andrea, Anne, Anthony, Antoine, Antonio, Augustin, Christelle, Christina, Christophe, Clara, Clémence, Daniel, Étienne, Ferdinand, Florian, Gaëtan, Ivan, Jean-Pierre, Johanna, Kaushik, Kévin, Léo, Lucien, Magali, María, Mariem, Mathilde, Matthieu, Nicolas D., Nicolas S., Pascale, Paul, Pierre, Rémi, Ritam, Rocco, Rodolfo, Sébastien, Shibam, Shizhu, Simon, Simona, Shouvik, Thomas, Valentin, Vivien, Xavier, Yann B., Yann R.[7]

Thanks to Ronald Cramer and Marc Stevens at the Cryptology group at CWI for accepting my post-doc application: I'm looking forward to more quantum cryptanalysis in the Netherlands! Special thanks to Marc Stevens for DBLPBibTex,which was used in generating the bibliography of this document.

[1] Officiellement, et en pratique quatre.

[2] Devenue COSMIQ.

[3] Et désolé d'avoir encombré vos bureaux de crocodiles, ornithorynques, crabes, koalas, Pokémons et autres pandas.

[4] Despite its weight.

[5] Devenue COSMIQ.

[6] Vous êtes nombreux, et j'espère n'avoir oublié personne, mais si tel est le cas, les réclamations sont à adresser, comme toujours, à `jean-pierre.tillich@inria.fr`.

[7] Mais aussi aux futurs membres, n'est-ce pas Albin ;) ?

Merci à Xavier pour avoir supporté avec bienveillance tant de questions et d'idées qui ne marchaient pas[8], du début de mon stage jusqu'à aujourd'hui. J'espère que notre collaboration scientifique se poursuivra avec la même complémentarité que le pilotage de la Toyota sur les autoroutes californiennes[9].

Merci aux thésards avec qui j'ai partagé le "bureau quantique" avant ma migration : Antoine, Vivien, Kaushik ; je souhaite que vos successeurs continuent de faire vivre l'esprit si particulier des contreforts de l'empire[10].

Merci à Jean-Pierre pour son enthousiasme lors de ces parties de babyfoot auxquelles je me suis quelquefois égaré[11]. Merci aussi à toi pour ce projet de codes correcteurs d'erreurs quantiques dans ton cours de théorie de l'information à l'École polytechnique, auquel je dois ma première rencontre officielle avec l'informatique quantique[12] ; dans le même contexte, merci à Laura de m'avoir prêté son Nielsen & Chuang.

Merci à Léo, compagnon de voyage[13] de Hong Kong à Kyoto en passant par cette plaine enneigée d'Amsterdam, dont nous avions cru, dans notre touchante naïveté, qu'elle se nommait l'aéroport de Schiphol et servait à faire décoller des avions. Merci à la peluche de Pui Pui, la célèbre[14] crocodile hongkongaise, dont le regard pétillant d'intelligence fut un soutien indéfectible lors de ce formidable périple en train de Schiphol à Paris, dont l'histoire mériterait une autre thèse à elle seule[15].

Merci à Thomas et Kévin pour m'avoir embarqué dans une journée de leur *road-trip* australien[16]. Merci à Yixin pour avoir traduit la carte de tous les restaurants de Shenzhen, et pour m'en avoir fait découvrir au moins la moitié.

Mais aucun de ces voyages n'aurait eu lieu sans l'aide de Christelle, sans qui je serais encore en train d'essayer de remplir mon formulaire de demande de visa chinois. Il faut une dose remarquable de patience et d'abnégation pour s'occuper d'autant de chercheurs et d'étudiants qui se trompent dans leurs demandes de mission (moi y compris), et sans toi cette équipe aurait pris feu depuis assez longtemps.

Merci à toutes celles et ceux qui se sont occupé(e)s de Bérengère la fougère : Daniel, Clara, et bien sûr Rémi durant de longs mois de confinement !

Merci à Étienne Chazal, Thierry Larsan et Bernard Philippet, dont les grilles de

[8] Morceau choisi : l'algorithme de recherche quantique en temps constant.

[9] J'espère aussi que mes talents de conducteur t'ont appris à relativiser, et à ne plus jamais oublier ton permis.

[10] Autrement dit, le bout de l'étage.

[11] Traduction : j'y ai été traîné contre mon gré. À tous ceux à qui cela pourrait arriver, et qui voudraient y échapper : cachez-vous sous votre bureau et ne bougez pas. Sa vision est basée sur le mouvement.

[12] Pour être honnête, je n'ai pas vraiment compris ce que tu essayais de m'expliquer lors de l'oral, concernant ce code de Hamming que j'étais bien en peine de reconnaître.

[13] Maintenant que tu as ton bureau pour toi tout seul, j'espère que tu ne t'y ennuies pas trop.

[14] Contrairement à nombre de cryptographes, elle a sa page Wikipédia.

[15] Afin d'être détaillée dans les remerciements, vous l'aurez compris.

[16] Même sur cette toute petite partie qui s'est faite du mauvais côté de la route.

mots fléchés sont le ciment de l'esprit d'équipe[17] SECRET[18].

Durant cette thèse, je n'ai pas tant appris en ouvrant des livres qu'en travaillant avec des chercheuses et des chercheurs de tous horizons, liées par la même passion de la cryptographie. Je voudrais donc d'abord remercier tous mes coauteur(ice)s : André, Anne, Antonio, Akinori, Gaëtan, Ferdinand, Jean-François, Léo, Lorenzo, María, Nicolas, Patrick, Paul, Paul, Rémi, Ritam, Sam, Sébastien, Thomas, Virginie, Xavier, Yannick, Yixin, Yu, et plus généralement, toutes les personnes avec qui j'ai eu la chance de collaborer, et toutes celles et ceux avec qui j'ai pu échanger durant ces années[19], y compris lors de nos désaccords[20], car c'est peut-être là que j'ai le plus appris.

Cette thèse fut riche d'enseignements, tant sur le plan scientifique que sur le plan humain, et j'espère en avoir tiré toutes les conséquences[21]. Merci à Anne, Christophe, Emmanuel, Gaëtan, Léo, Lucca, Matthieu, Pierrick, Serge, Stéphanie, Steve, Véronique, Xavier et toutes celles et ceux qui m'ont appris à quel point le monde est complexe hors de la tour d'ivoire, et qu'il est parfois nécessaire de s'y confronter.

Et en parlant d'enseignement, merci à Fabien pour le jus de carottes, et désolé pour la soupe de Python.

Puisque je suis arrivé jusqu'ici, il ne me reste plus qu'à remonter le temps et remercier toutes celles et ceux qui m'ont guidé sur cette voie – ou qui m'y ont accompagné. Merci à Adina, Bogdan, Eunice, Zeinab et toute l'équipe du MPRI avec laquelle nous mangions les pizzas aux légumes[22] du CROUS de Paris 7. Merci à Manuel Bodirsky et toute l'équipe de l'Institut Für Algebra de la TU Dresden pour une première expérience de recherche passionnante. Merci à François Morain et Aurore Guillevic pour avoir encadré mon PRL de 3A. Merci à Antonin et toute l'équipe du PSC de 2A qui m'offrit des nuits blanches parmi les plus mémorables[23]. Merci aux génies, aux gentils fous et aux artistes du parcours Maths-Info, de la Section Escrime, de la 4$^{\text{e}}$ compagnie, de la promo 2012, 2013 et de la 2014. Merci à Bruno, sans doute le premier avec qui j'ai parlé de thèse ! Merci à mes collègues, amis et futurs docteurs[24] du Lycée Hoche. Merci à cette grande famille d'enseignants de mathématiques et d'informatique qui coupent des fromages[25], tapent sur les équations linéaires[26] et font avancer des tortues[27]. Je mesure ma chance

[17] Certains s'offusqueront de cette généralité et je les entends déjà grommeler : "*gnagnagna* babyfoot *gnagnagna*". Mais à l'heure où j'écris ces lignes, le babyfoot est encore confiné. Malgré l'épidémie, malgré le réchauffement climatique, malgré les menaces de l'ordinateur quantique, malgré notre avenir incertain, les mots fléchés, eux, n'ont pas failli à leur devoir.

[18] Devenue COSMIQ, mais je suis nostalgique.

[19] J'avais préparé une liste, mais elle est trop longue pour figurer dans cette note de bas de page.

[20] Celle-là aussi.

[21] Quelques enseignements utiles : il faut fuir Twitter et se méfier des blogs.[30]

[30] Merci aux instructeurs et aux membres de la section "Krav Maga" du CSADL pour m'avoir offert l'occasion, dans certaines de ces circonstances, de me détendre un peu.

[22] Oui, c'étaient les meilleures.

[23] Ainsi qu'un enseignement essentiel : la recherche, ça ne marche pas toujours.

[24] Dédicace spéciale à Agathe et Nicolas :)

[25] Illustration de la section d'un parallélépipède par un plan. Quelquefois, il vaut mieux ne pas être assis au premier rang.

[26] Un système linéaire devient beaucoup plus simple une fois qu'on a dessiné un marteau.

[27] Ce qui n'est pas toujours facile, car les tortues tournent souvent dans le mauvais sens.

de vous avoir connus.

Merci à tous les membres de ma famille qui se sont retrouvés concernés, de gré ou de force, par cette thèse[28]: Nathalie, Franck, Abel, Jean-Baptiste, Véronique et aussi André, dont j'imagine que ce manuscrit aurait trouvé une bonne place dans sa bibliothèque.

Pour finir sur une note légère[29], merci à Beethoven, Chopin et Fleshgod Apocalypse, qui se sont relayés pour me fournir la motivation nécessaire à l'écriture de cette thèse. Si vous souhaitez venir à bout de cette lecture, n'hésitez pas à faire aussi appel à eux.

[28] Même si malgré tous mes efforts, la "cryptanalyse symétrique quantique" continue, et continuera de jeter un froid dans tous les repas de famille.

[29] Jeu de mots intentionnel.

# Abstract

Modern cryptography relies on the notion of computational security. The level of security given by a cryptosystem is expressed as an amount of computational resources required to break it. The goal of cryptanalysis is to find attacks, that is, algorithms with lower complexities than the conjectural bounds. With the advent of quantum computing devices, these levels of security have to be updated to take a whole new notion of algorithms into account. At the same time, cryptography is becoming widely used in small devices (smart cards, sensors), with new cost constraints.

In this thesis, we study the security of secret-key cryptosystems against quantum adversaries.

We first build new quantum algorithms for k-list (k-XOR or k-SUM) problems, by composing exhaustive search procedures.

Next, we present dedicated cryptanalysis results, starting with a new quantum cryptanalysis tool, the offline Simon's algorithm. We describe new attacks against the lightweight algorithms Spook and Gimli and we perform the first quantum security analysis of the standard cipher AES.

Finally, we specify Saturnin, a family of lightweight cryptosystems oriented towards post-quantum security. Thanks to a very similar structure, its security relies largely on the analysis of AES.

**Keywords: Symmetric cryptography, cryptanalysis, post-quantum security, quantum algorithms, merging algorithms, lightweight cryptography.**

# Contents

# List of Figures

# List of Tables

# Introduction

Cryptography aims at protecting information, transmitted through insecure channels, against adversaries trying to access or to control it. This science has certainly accompanied all types of human communication, from written letters to radio messages, and its history features a gallery of prominent historical figures, from Julius Caesar to Alan Turing. But it did not settle down as a branch of mathematics until the course of the 20th century, and more precisely, as a branch of *complexity theory*, which studies the hardness of computational problems.

**Computational security.** In our era of digital communications, modern cryptographic tools are mathematical objects which provide some security guarantees up to *computational assumptions*. These assumptions say how hard it should be, for the adversary, to solve a well-defined mathematical problem. For example, the RSA public-key scheme [RSA78], arguably the most widely known cryptosystem of modern times, relies on the hardness of factoring natural integers. Factoring algorithms have been well-studied, and the best known algorithm to date is the General Number Field Sieve [LL+93]. Thus, when two users *Alice* and *Bob* set up their communication protocol, they parameterize it so that the eavesdropper *Eve*, using the GNFS, cannot recover the prime factorization within a reasonable time.

In a secure communication protocol, RSA will often be used only once per session, in order to set up a shared secret between the participants, denoted a *symmetric key*. Afterwards, Alice and Bob will rely on a secret-key cryptosytem, such as AES-GCM encryption. Again, the security of AES-GCM is a computational assumption: it is conjectured that, in order to decrypt the exchanged messages, Eve has no other choice but to try all possible keys, which is infeasible if the number of keys is sufficiently large.

Our confidence in these assumptions relies on a continuous, public and independent re-evaluation of them. New algorithms may be designed, and new attacks may be found, that may challenge our initial security assumptions. Since deprecating and updating cryptographic standards takes a long time, cryptography must always be (more than) one step ahead.

**Making cryptography lightweight.** At the dawn of the digital era, Alice and Bob were two humans using personal computers. Today, the number of devices connected to the Internet exceeds by far the world population, and the *Internet of Things* (IoT) experiences a steady growth. Alice and Bob are now more likely to be two *smart* devices, sensors, vehicles, routers, communicating and reporting without any human interaction or supervision. The smaller and ubiquitous these devices become, the stronger the constraints in terms of computing power, bandwidth or latency. Early cryptographic algorithms were not designed with these constraints in mind, and can perform badly in these new environments. Therefore, the last decade has seen the rise of new algorithms, aiming at lower implementation costs, energy consumption, and a better balanced between security and efficiency: this is *lightweight cryptography* (a nice overview of these algorithms can be found in [BP17]). However, being *lightweight* should not come at the expense of new weaknesses, and these new designs must undergo the same public scrutiny.

In 2019, the American National Institute of Standards and Technology (NIST) launched a standardization process for new lightweight cryptographic primitives [NIS18]. Out of 56 first-round candidates, 32 of them passed to the second round on August 30, 2019 [NIS19].

**The quantum paradigm shift.** The concept of a *universal quantum simulator* was proposed in the early 1980s by visionary physicists. Simulating the evolution of a complex quantum mechanical system is a hard problem for today's computers. But if the simulator was a *controlled quantum system* itself, it would be possible to reproduce any other system with fidelity. The first physicists working in this direction rapidly understood that this quantum simulator would be a *quantum computer*, running *quantum algorithms*, with a new notion of complexity [Deu85]. Thus, *quantum complexity theory* was born. But the first major achievements of quantum computing appeared later in 1994, with Simon's [Sim94] and Shor's [Sho94] algorithms. The former showed the first exponential speedup of a black-box problem. The latter posed a direct threat to some classical cryptosystems, which unluckily happened to be some of the most widely used, including RSA.

Guessing when a scalable quantum computing architecture will be available to Eve, if it happens to be technically (and economically) feasible, is a perilous exercise that we shall not attempt here. While technologies make constant progress, major engineering problems still need to be solved. In 2019, a team of researchers supported by Google announced having performed a quantum computation (sampling the distribution of random circuits) that outperformed the best classical simulator available [Aru+19], a milestone inconveniently named *quantum supremacy.* This means that today's quantum architectures are becoming faster than their classical elders at some very *specific* tasks, although they haven't seen practical applications yet.

**Cryptography in the post-quantum future.** Future quantum computers can have an impact on today's secrets. This fact was summarized by Mosca [Mos15] in a statement known today as *Mosca's theorem.* Let $X$ be the amount of time for which Alice and Bob expect their communication to remain secure. Let $Y$ be the time necessary to transition to quantum-secure cryptography. Let $Z$ be the time to build a large-scale quantum computer with applications to cryptography. Then if $X + Y > Z$, the communications that were exchanged at time $Y$, before the transition, could be revealed to the malicious Eve earlier than their expiration date (Figure 1).

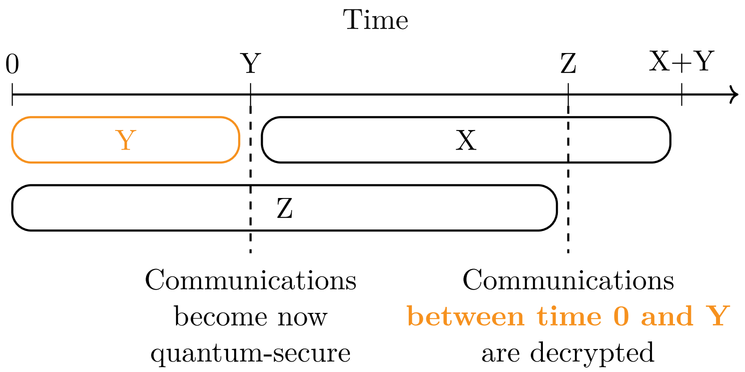

Time
0
Y
Z
X+Y
Y
X
Z
Communications become now quantum-secure
Communications **between time 0 and Y** are decrypted

**Figure 1:** Illustration of “Mosca’s theorem”.

This is why the mere *possibility of a quantum Eve in an undefined future* is a sufficient motivation to seek and explore *post-quantum* designs today. This view has been notably embraced by the NSA, which made in 2015 a public statement recommending a transition to post-quantum systems. In 2016, the NIST launched a standardization process for quantum-secure *public-key* cryptosystems [NIS16], in replacement to those broken by Shor’s algorithm. Out of 69 submissions, 26 second-round candidates were announced in 2019, including 17 public-key encryption and key-establishment systems, and 9 digital signatures. In July 2020, NIST announced the third-round “finalist” candidates (4 for public-key encryption and key-establishment and 3 for digital signatures) alongside 8 “alternate” algorithms advancing to the third round, but unlikely to be standardized at the end of the third round.

But these public-key systems represent only a subset of the cryptographic tools used in today’s digital infrastructures.

**Post-quantum symmetric cryptography.** In today’s protocols, public-key cryptosystems intervene usually in the architecture that enables Alice and Bob to agree on a secret (certificates, public-key infrastructures, key-exchange mechanisms). But in order to have confidence in the protocol, we ask for security guarantees of *all* its components; and most of the transiting data is encrypted under a secret key $k$, using a standard symmetric cipher such as [AES].

Symmetric ciphers are usually regarded as less structured, as they do not need to embed any trapdoor. Quantum algorithms with an exponential speedup, such as Shor’s, seem inapplicable. Despite this fact, it was known very early on that Grover’s generic search algorithm [Gro96] would speed up the exhaustive search of the key by a quadratic factor. In order to bring this computational effort back to the same level as classical exhaustive search, a simple solution seems to *double* the length of the keys [Nat18]. Fortunately, a standard such as [AES] allows keys of 128 as well as 256 bits.

This *status quo* was challenged by pioneering work of Kuwakado and Morii [KM10; KM12] who showed that a quantum attacker of a new type, using the model of *superposition access*, could turn some classically secure constructions into quantumly insecure ones. Although the meaningfulness of these attacks is still debated, these results triggered a wide interest in post-quantum symmetric cryptography.

**Quantum cryptanalysis.** Both in public-key (asymmetric) and secret-key (symmetric) cryptography, we gain knowledge and confidence in the security of our designs by challenging their conjectural security claims. Cryptanalysts try to create algorithms that either solve generic problems more efficiently than before, or disprove a security assumption, thereby breaking the scheme or lowering its promised security.

To ensure security against a quantum Eve, we must consider *quantum attacks.* Upon its three decades of existence, quantum information science has created a sound notion of *quantum computation*, and there exists universal models of quantum computers. Thus, we can define *quantum* security levels, we can make *quantum* security claims and we can design *quantum* attacks that challenge them, up to additional assumptions on the adversary's power (query models, memory models, and so on). As security evidence in the classical world was obtained through a continuous cryptanalysis effort, so will it be in the quantum world.

## Contributions

The algorithms that will be presented in this document can be roughly divided into two categories.

**Algorithms for generic problems.** In this first category, we aim at solving generic problems. In our case, these problems will be defined with some kind of oracle access.

**Example 1** (Collision Search Problem). Given query access to a random function $h : \{0,1\}^n \to \{0,1\}^n$, find a *collision* pair: $x \neq y$ such that $h(x) = h(y)$.

In this example, the input of the problem is a function drawn uniformly at random from all functions $\{0,1\}^n \to \{0,1\}^n$. The obtained algorithms, with complexities depending on $n$, will provide us with generic bounds and with suitable tools for cryptanalysis.

**Algorithms dedicated to cryptanalysis.** In this second category, we study specific constructions, with a particular design.

**Example 2** (AES key-recovery). Given query access to an AES block cipher encrypting under some secret key $k$, find $k$.

In this example, we want to recover the key of a specific construction (the AES). To date, there does not exist any algorithm that performs this task faster than the generic exhaustive search of the key. Reduced-round (weakened) versions of AES have been *broken*, in the sense that better dedicated algorithms have been designed.

**Organization.** This thesis is organized in four main parts: • an introduction to quantum computing, cryptanalysis and known results (Chapters 1 to 4), • new generic quantum algorithms for merging problems (Chapters 5 and 6), • new quantum (and classical) cryptanalysis results (Chapters 7, 8 and 9), • the design of SATURNIN (Chapter 10).

**Chapter 1 (Introduction to Quantum Computing).** We introduce quantum computing tools. We first define the quantum circuit model, following textbooks such as [NC00]. We define quantum memory models and qRAM. We flesh out some folklore properties of qRAM, such as its emulation with sequential circuits. We end this chapter by introducing Simon's algorithm, one of the simplest examples of the power of quantum computations, and an invaluable tool for quantum symmetric cryptanalysis.

**Chapter 2 (Classical and Quantum Search).** We review the frameworks of classical and quantum exhaustive search. We present Grover's algorithm, Amplitude Amplification, and other well-known results. We study variants of Amplitude Amplification when the success probability is unknown. We introduce the notion of an *approximate test* that makes errors, and show that Amplitude Amplification can still run appropriately. We introduce a framework for *nested searches*, with a natural classical-quantum correspondence: this is a new contribution from this thesis [BNS19b]. Finally, we compare classical and quantum algorithms for collision search known prior to our work.

**Chapter 3 (Introduction to Cryptography).** We give a broad introduction to cryptography, with a particular focus on symmetric designs and the corresponding generic attacks. We introduce the Q1 and Q2 settings for quantum attacks, that correspond respectively to quantum adversaries accessing only classical data (e.g., the communications exchanged between time 0 and time Y in Figure 1) and adversaries making *superposition queries*.

**Chapter 4 (Quantum Symmetric Cryptanalysis).** We recall previous works in quantum cryptanalysis, notably the impact of Simon's algorithm on ciphers with a strong algebraic structure, in the Q2 query model. We detail the Grover-meets-Simon algorithm of Leander and May [LM17], which uses Simon's algorithm on top of an exhaustive search procedure.

**Chapter 5 (Quantum Merging Algorithms for the k-XOR Problem), Chapter 6 (Solving the k-XOR Problem with a Single Solution).** These chapters are based on a joint work with André Chailloux and María Naya-Plasencia [CNS17] published at ASIACRYPT 2017, a joint work with Lorenzo Grassi and María Naya-Plasencia [GNS18] published at ASIACRYPT 2018 and a joint work with María Naya-Plasencia [NS20] published at EUROCRYPT 2020.

We study *quantum merging algorithms* for $k$-list problems (also known as $k$-SUM or $k$-XOR). We start with a new quantum algorithm for collision search, which corresponds to the case $k = 2$. Our algorithm achieves a quantum speedup *without qRAM* and realizes the lowest quantum operation count for collision search known to date. Next, we introduce the *quantum merging* framework, by analogy with classical merging algorithms. We define a class of *merging trees*, which represent merging strategies. We describe the best known quantum algorithms for $k$-XOR and $k$-SUM problems, including the single-solution variants, and a variety of applications in secret- and public-key cryptography. We prove the optimality of these new algorithms in the class of merging trees. The presentation of these chapters is simpler than [NS20] and includes some new improved results.

**Chapter 7 (The Offline Simon's Algorithm).** This chapter is based on a joint work with Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia and Yu Sasaki [Bon+19], which was published at ASIACRYPT 2019 and presented at QIP 2020.

We introduce the *offline Simon's algorithm* and present its implications in quantum cryptanalysis. Two main ideas are involved. First of all, we analyze further the Grover-meets-Simon algorithm of Leander and May. We remark that in most applications, the algorithm queries the secret-key primitive with fresh uniform superpositions only, in order to run independent instances of Simon's algorithm. Thus, these queries can be done once and *reused*. This reduces drastically the number of Q2 queries. Next, we make the algorithm *offline*: we modify it to allow a precomputation step, in which the few necessary Q2 queries are replaced by an accumulation of classical data inside a small quantum register. This strategy leads to surprisingly efficient attacks in the Q1 setting. Indeed, we are now able to obtain square-root speedups over some classical attack scenarios, while simultaneously reducing the memory used from exponential to polynomial. This answers positively the (previously open) questions whether Simon's algorithm could be exploited in Q1 quantum cryptanalysis, and whether the algebraic structure of schemes broken in the Q2 scenario could increase their Q1 vulnerability.

**Chapter 8 (Cryptanalysis Based on Symmetries).** This chapter is based on a joint work with Patrick Derbez, Paul Huynh, Virginie Lallemand, María Naya-Plasencia and Léo Perrin [Der+20] which was published at CRYPTO 2020; and a joint work with Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin and Ferdinand Sibleyras [Fló+20] which was published at ASIACRYPT 2020 and received a "best paper" award.

In the context of this thesis, a classical adversary is all but a quantum one restricted to classical computations. Thus, security in the quantum world also implies security against classical adversaries. Classical cryptanalysis, and estimating the classical security margin of primitives, will remain as important as it is today.

In this chapter, we present classical cryptanalysis results, based on *internal symmetries*, on two second-round candidates of the NIST lightweight process [NIS19], Spook and Gimli. Both are respectively based on the efficient permutations Shadow and

Gimli, in which the internal state is divided into medium-sized blocks that are processed in a similar way (*bundles* in Shadow and *columns* in Gimli). In Spook, we observe that some round constants used to break the symmetry between bundles in Shadow can cancel each other out, which negates their effect. This allows to propagate symmetric states inside the permutation. In combination with an efficient truncated differential pattern, we exhibit a practical limited-birthday distinguisher on the full Shadow. Next, we present a practical forgery attack on the candidate Spook using Shadow reduced to two thirds of its original rounds. These results won the "Mathematical cryptanalysis 1" challenge proposed by the designers[1] and motivated a tweak in their submission [Bel+20].

In Gimli, we also target first the permutation from [Ber+17a]. We remark that the simple linear layer used in Gimli does not diffuse enough between the columns, and that symmetric columns can be propagated backwards and forwards on many rounds. Using a guess-and-determine approach, we design an *internal symmetry* distinguisher on the full Gimli, with a practical variant on 23 of its 24 rounds. Next, we use the same design pattern in collision attacks on the hash function of the Gimli submission package, reaching 12 rounds (8 practical) and 18 rounds for semi-free-start collisions. Finally, we extend these collision attacks in the quantum setting.

**Chapter 9 (Quantum Security Analysis of AES).** This chapter is based on a joint work with Xavier Bonnetain and María Naya-Plasencia [BNS19b] which was published in the second issue of *Transactions in Symmetric Cryptology* (ToSC) 2019. This work also introduced the nested search framework presented in Chapter 2.

One of the most well-known block ciphers to date is the current standard [AES]. In this chapter, we perform the first quantum security analysis of AES, in the secret-key model. Indeed, despite a massive amount of classical cryptanalysis, only generic exhaustive search of the keys had been studied so far in the quantum setting. The new quantum key-recovery attacks presented here help us to draw a first estimation of the security margin of AES in the quantum setting. They rely on classical design patterns. However, *quantizing* classical attacks is a non-trivial task, since by definition, a quantum key-recovery must compete against *quantum* exhaustive search of the keys, that is, Grover's algorithm. After discussing the attack strategies that could be applied, we present in detail the two families that give new results. First, we quantize the *square attacks* on 6-round and 7-round AES variants, using Q1 queries only and quantum memory with quantum random access. Next, we present our best attack, which reaches 8 rounds out of 14 for AES-256, the variant which is usually recommended for post-quantum security. This attack uses a Demirci-Selçuk Meet-in-the-Middle design pattern, but differs significantly from its classical counterparts. We show that it can use Q1 queries, classical memory and a small amount of qubits only (with no quantum random access). Surprisingly, our adaptation to the quantum setting gives also new ideas for classical attacks of this type.

[1] The challenges are detailed on their webpage: www.spook.dev/challenges

**Chapter 10 (Saturnin).** This chapter is based on a joint work with Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin and Thomas Pornin, which was submitted to the NIST LWC process [Can+19] and published in a special issue of ToSC [Can+20].

We introduce the SATURNIN suite, a candidate of the NIST lightweight standardization project, now in the second round. SATURNIN is a block cipher with 256-bit keys and 256-bit blocks. SATURNIN-CTR-Cascade is an authenticated cipher with associated data (AEAD), and SATURNIN-Hash is a 256-bit hash function. The guiding principle of SATURNIN is that post-quantum security should not be sacrificed for lightness, and it should be possible to obtain both with an appropriate design. The increased block size of SATURNIN overcomes some limitations of AES. The cipher is designed as to have an efficient bitsliced implementation, while admitting an abstract representation that appears to be close to an AES with a bigger state, and a simpler key-schedule. Thus, it inherits most of our knowledge on the security of AES, including the analysis of Chapter 9 in the quantum setting. The AEAD and Hash constructions use quantum-secure block cipher modes of operation.

We detail below other results that will not be presented in this document.

**Quantum slide attacks.** Classical *slide attacks* are powerful key-recovery attacks that rely on self-similarities in iterated ciphers such as Substitution-Permutation Networks or Feistel schemes. These attacks have the unique property of performing independently from the number of iterates of the construction. Introduced in [Kap+16a], *quantum slide attacks* translate the self-similarity into a promise problem. This problem is solved using a quantum shift-finding algorithm with an exponential acceleration, such as Simon's algorithm. With Xavier Bonnetain and María Naya-Plasencia, we performed a comprehensive study of slide attacks in the classical literature and of quantum variants. These results were published at SAC 2019 [BNS19a].

**Generic algorithms for golden collision search.** We studied algorithms for collision search when a single collision (the *golden collision*) has to be found. The best attack on the NIST post-quantum candidate SIKE [Jao+17] is a generic golden collision search. With Samuel Jaques, we improved the known quantum algorithms without quantum random-access. These results did not threaten the parameter choices of [Jao+17], but showed the potential of quantum walk algorithms even without qRAM, under the assumption that errors in the quantum architecture are self-corrected. These results were published at SAC 2020 [JS20].

**Quantum algorithms for the subset-sum problem.** The random subset-sum problem aims to find, given $n$ integers of $\mathbb{Z}_M$ ($M \simeq 2^n$), a subset of them that sums to a given target. Although no cryptosystem based on subset sums is considered for standardization by the NIST post-quantum project, this problem borders on generic decoding, and has cryptanalytic applications such as quantum key-recoveries of CSIDH [BS20]. The best classical algorithms to date are merging algorithms using the *representation technique* of Howgrave-Graham and Joux [HJ10]. With Xavier Bonnetain, Rémi Bricout and Yixin Shen, we improved both the classical and quantum algorithms for subset-sums. In the classical setting, we used extended representations and loosened constraints to reach a better time complexity than previous work [BCJ11]. In the quantum setting, we designed the first quantum subset-sum algorithm using quantum-accessible *classical* memory, improved the time complexity in all models and worked around a heuristic on quantum walk algorithms that previous works were dependent on. These results were published at ASIACRYPT 2020 [Bon+20].

**Security analysis of CSIDH.** CSIDH [Cas+18] is a proposal for a non-interactive key-exchange based on an Abelian group law, using elliptic curves at its core. Although it is not a candidate of the NIST post-quantum project, as it was proposed slightly later at ASIACRYPT 2018, CSIDH triggered a massive amount of research. Contrarily to most proposals, the best quantum attack has more than a quadratic advantage with respect to the best classical attack, as it uses Kuperberg's Abelian hidden shift algorithm [Kup05] or its variants. These algorithms have a subexponential complexity, whereas the best known classical algorithm is exponential. The security levels of the proposed parameters in [Cas+18] were only conjectured. Due to the gap between classical and quantum complexities, a precise analysis of the quantum attack is paramount, since this attack alone defines the security level. With Jean-François Biasse, Xavier Bonnetain, Benjamin Pring and William Youmans, we studied asymptotic trade-offs in a *hybrid* (mixed quantum and classical) attack against CSIDH based on a variant of Kuperberg's algorithm. This work was published in the *Journal of Mathematical Cryptology* [Bia+19]. With Xavier Bonnetain, we performed a non-asymptotic study, both of the hidden shift algorithms and the other factors in the attack. We showed that the initial parameters of [Cas+18] offered a lower security than conjectured, not comparable with the security levels of the NIST call that they intended to match. This work was published at EUROCRYPT 2020 [BS20].

# Publications

[CNS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. "An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography". In: *ASIACRYPT (2)*. Vol. 10625. LNCS. Springer, 2017, pp. 211–240 (cit. on pp. xxiii, 75, 77, 79, 80).

[GNS18] Lorenzo Grassi, María Naya-Plasencia, and André Schrottenloher. "Quantum Algorithms for the k-xor Problem". In: *ASIACRYPT 2018*. Vol. 11272. LNCS. Springer, 2018, pp. 527–559 (cit. on pp. xxiii, 75, 89, 97, 99).

[Bia+19] Jean-François Biasse, Xavier Bonnetain, Benjamin Pring, André Schrottenloher, and William Youmans. "A Trade-off Between Classical and Quantum Circuit Size for an Attack Against CSIDH". In: *Journal of Mathematical Cryptology* (2019), pp. 1–16 (cit. on pp. xxvii, 11).

[Bon+19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. "Quantum Attacks Without Superposition Queries: The Offline Simon's Algorithm". In: *ASIACRYPT (1)*. Vol. 11921. LNCS. Springer, 2019, pp. 552–583 (cit. on pp. xxiv, 145).

[BNS19a] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "On Quantum Slide Attacks". In: *SAC*. Vol. 11959. LNCS. Springer, 2019, pp. 492–519 (cit. on pp. xxvi, 67, 70, 155).

[BNS19b] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Trans. Symmetric Cryptol.* 2019.2 (2019), pp. 55–93 (cit. on pp. xxiii, xxv, 27, 28, 34, 201, 205, 216, 217, 229).

[Bon+20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. "Improved Classical and Quantum Algorithms for Subset-Sum". In: *ASIACRYPT (2)*. Vol. 12492. LNCS. Springer, 2020, pp. 633–666 (cit. on pp. xxvii, 141, 144, 274).

[BS20] Xavier Bonnetain and André Schrottenloher. "Quantum Security Analysis of CSIDH". In: *EUROCRYPT (2)*. Vol. 12106. LNCS. Springer, 2020, pp. 493–522 (cit. on pp. xxvi, xxvii, 11, 69, 274).

[Can+20] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. “Saturnin: a Suite of Lightweight Symmetric Algorithms for Post-quantum Security”. In: *IACR Trans. Symmetric Cryptol.* (2020) (cit. on pp. xxvi, 54, 235, 238).

[Der+20] Patrick Derbez, Paul Huynh, Virginie Lallemand, María Naya-Plasencia, Léo Perrin, and André Schrottenloher. “Cryptanalysis Results on Spook - Bringing Full-Round Shadow-512 to the Light”. In: *CRYPTO (3)*. Vol. 12172. LNCS. Springer, 2020, pp. 359–388 (cit. on pp. xxiv, 157, 173).

[Fló+20] Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sibleyras. “New Results on Gimli: Full-Permutation Distinguishers and Improved Collisions”. In: *ASIACRYPT (1)*. Vol. 12491. LNCS. Springer, 2020, pp. 33–63 (cit. on pp. xxiv, 157, 184, 185, 199).

[JS20] Samuel Jaques and André Schrottenloher. “Low-gate Quantum Golden Collision Finding”. In: *SAC*. LNCS. Springer, 2020 (cit. on pp. xxvi, 42, 43, 80, 113).

[NS20] María Naya-Plasencia and André Schrottenloher. “Optimal Merging in Quantum k-XOR and k-SUM Algorithms”. In: *EUROCRYPT (2)*. LNCS Vol. 12106. Springer, 2020, pp. 311–340 (cit. on pp. xxiii, xxiv, 75, 89, 99, 107, 109, 118, 121, 126, 144).

# Preprints

[Bha+20] Ritam Bhaumik, Xavier Bonnetain, André Chailloux, Gaëtan Leurent, María Naya-Plasencia, André Schrottenloher, and Yannick Seurin. “QCB: Efficient Quantum-secure Authenticated Encryption”. In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1304 (cit. on pp. 271, 275).

# Notation and Acronyms

We recapitulate here the main notations and acronyms used throughout this thesis. Usually, $n$ will be an integer parameter and we will compare asymptotic complexities in $n$ using the Bachmann-Landau notations of complexity theory: *big O, small o, big Omega, big Theta*. We use $\mathsf{poly}(n)$ to denote a polynomial of $n$.

$$\left.\begin{array}{r} f(n) = \mathcal{O}(g(n)) \\ f(n) = o(g(n)) \\ f(n) = \Omega(g(n)) \\ f(n) = \Theta(g(n)) \end{array}\right\} \iff \left\{\begin{array}{c} \exists C > 0, \exists n_0, \forall n > n_0, |f(n)| \leqslant C \cdot g(n) \\ \forall C > 0, \exists n_0, \forall n > n_0, |f(n)| < C \cdot g(n) \\ \exists C > 0, \exists n_0, \forall n > n_0, |f(n)| \geqslant C \cdot g(n) \\ f(n) = \Omega(g(n)) \text{ and } f(n) = \mathcal{O}(g(n)) \end{array}\right.$$

## Acronyms

| | |
|---:|---|
| AES | Advanced Encryption Standard |
| AEAD | Authenticated encryption with associated data |
| CCA | Chosen-ciphertext attack |
| CPA | chosen-plaintext attack |
| CSAM | Classical memory with sequential access |
| DES | Data Encryption Standard |
| DLP | Discrete logarithm problem |
| DS-MITM | Demirci-Selçuk Meet-in-the-Middle |
| LWC | Lightweight cryptography (NIST project) |
| NISQ | Noisy intermediate-scale quantum (devices) |
| NIST | National Institute of Standards and Technology |
| MAC | Message authentication code |
| PQC | Post-quantum cryptography (NIST project) |
| Q1 | Quantum attacks with classical queries |
| Q2 | Quantum attacks with quantum queries |
| QRACM | Classical memory with quantum random-access |
| QRAQM | Quantum memory with quantum random-access |
| qRAM | Quantum random-access memory |
| RAM | (Classical) random-access memory |
| RSA | Rivest-Shamir-Adleman cryptosystem |
| SPN | Substitution-Permutation Network |
| ZRR | Zone à régime restrictif (Restricted Area) |

## All chapters

| | |
|---|---|
| $\oplus$ | bitwise addition (XOR) |
| $+$ | integer addition (modular addition) |
| $\cdot$ | scalar product between vectors or bit-strings |
| $\cdot$ | product (example: $T \cdot M = 2^n$) |
| $\cdot$ | placeholder (example: $f(\cdot)$) |
| $\Vert$ | concatenation of bit-strings |
| $\delta_{ij}$ | Kronecker symbol |
| $\lvert X \rvert$ | cardinality of a set $X$ |
| $\overline{a_{n-1}\dots a_0}^b$ | integer written as $a_{n-1}\dots a_0$ in basis $b$<br>When $b = 2$, this corresponds to big-endian (the most significant bit is written first) |
| $\neg x$ | negation of the Boolean $x$ |
| $\cdot \wedge \cdot$ | Boolean AND |
| $\cdot \vee \cdot$ | Boolean OR |
| $E_k$ | block cipher with a secret key $k$ |
| $\Pi$ | permutation |
| $f, g, h$ | functions |

## Chapter 1

| | |
|---|---|
| $\mathcal{H}$ | finite-dimensional Hilbert space |
| $\langle \cdot \vert \cdot \rangle$ | inner product in $\mathcal{H}$ |
| $\Vert \cdot \Vert$ | inner norm in $\mathcal{H}$ |
| $\lvert \cdot \rvert$ | modulus of a complex number |
| $\lvert x \rangle$ | basis vector in $\mathcal{H}$ ($x$ is an integer or a bit-string) |
| $\langle x \rvert$ | linear form (basis vector in the dual space $\mathcal{H}^*$) |
| $\lvert \phi \rangle \otimes \lvert \psi \rangle$ | tensor product of states |
| $\mathsf{U}$ | unitary operator |
| $\mathsf{U}^\dagger$ | adjoint of $\mathsf{U}$ (*uncomputation*) |
| $\mathsf{U}_1 \otimes \mathsf{U}_2$ | tensor product of operators |
| $\mathsf{H}$ | Hadamard gate |
| $\mathsf{H}^{\otimes n}$ | Hadamard transform |
| $\mathsf{X}$ or $\mathsf{NOT}$ | bitflip gate |
| $\mathsf{Z}$ | phaseflip gate |
| $\mathsf{T}$ | T-gate |
| $\mathsf{CNOT}$ | CNOT gate |
| $\mathsf{Tof}$ | Toffoli gate |
| $\mathsf{nTof}$ | $n$-qubit Toffoli gate |
| $\mathsf{I}$ | identity gate or operator |
| $O_f$ | standard *or* phase oracle for $f$ |

## Chapter 2

| | |
|---|---|
| $X$ | exhaustive search space |
| $G$ | “good” subspace $G \subseteq X$ |
| $B$ | “bad” subspace $B = X \backslash G$ |
| $f$ | test function ($f\ : X \rightarrow \{0,1\}$) |
| $X_{\vert f}$ | subset of elements of $X$ that satisfy $f$ |
| $\mathcal{A}$ | classical or quantum algorithm |
| $\mathsf{T}_{\mathsf{c}}\left(\mathcal{A}\right)$ | average classical time complexity of $\mathcal{A}$ |
| $\mathsf{T}_{\mathsf{q}}\left(\mathcal{A}\right)$ | quantum time complexity of $\mathcal{A}$ |
| $\mathsf{M}\left(\mathcal{A}\right)$ | memory complexity (classical or quantum) of $\mathcal{A}$ |
| $h$ | random $n$-bit to $n$-bit function |
| $u\vert *$ | bit-string starting with the prefix $u$ |

## Chapter 5 and Chapter 6

| | |
|---|---|
| $k$ | number of lists to merge |
| $\kappa$ | $\lfloor \log_2(k) \rfloor$ |
| $\mathcal{L}_i$ | a list of bit-strings (of exponential size) |
| $L_i$ | size of $\mathcal{L}_i$ |
| $\ell_i$ | $\frac{1}{n} \log_2 L_i$ |
| $\mathcal{L}_1 \bowtie_u \mathcal{L}_2$ | “join” operator |
| $0_m$ | zero bit-string of length $m$ |
| $\mathcal{T}$ | merging tree, subtree *or* root node |
| $T_k$ | family of optimal trees for quantum merging |

## Chapter 8

| | |
|---|---|
| $0 * * * 01 * *$ | truncated differential pattern |
| $S$ | Shadow 4-bit S-Box |
| $\mathsf{L}'$ | Shadow 64-bit L-Box |
| $\mathsf{D}$ | Shadow diffusion layer |
| $\mathsf{RC}_i$ | round constant of step $i$ |
| $\sigma_0, \sigma_1, \sigma_2, \sigma_3$ | Shadow Super S-Boxes |
| $S$ | Gimli state |
| $A, B, C, D$ | Gimli columns |
| $x, y, z$ | 32-bit words in a column |
| $\ll, \gg$ | shift left, shift right (32-bit words) |
| $\lll, \ggg$ | rotate left, rotate right (32-bit words) |
| $SP$ | Gimli Substitution-Permutation function |
| $\mathsf{RC}_i$ | round constant of round $i$ |

## Chapter 9

| | |
|---:|---|
| ARK | AES AddRoundKey operation |
| SB | AES SubBytes operation |
| SR | AES ShiftRows operation |
| MC | AES MixColumns operation |
| $k, k_i$ | master key, round key at round $i$ |
| $S$ | AES 8-bit S-Box |
| $x_i, y_i, z_i, w_i$ | AES state after ARK, SB, SR, MC |

## Chapter 10

| | |
|---:|---|
| $D$ | domain separator |
| $R$ | number of super-rounds |
| $\mathcal{T}$, $\mathcal{D}$, $\mathcal{M}$ | time, data, memory used by the attacker |
| $\mathsf{RC}_0, \mathsf{RC}_1$ | pair of SATURNIN round constants |
| $\mathsf{S}$ | S-Box layer |
| $\sigma_0, \sigma_1$ | SATURNIN S-Boxes |
| $\mathsf{SR}_r, \mathsf{SR}_{\text{slice}}, \mathsf{SR}_{\text{sheet}}$ | nibble permutations |
| $\mathsf{MC}$ | SATURNIN MixColumns |
| $\mathsf{S}_{16}$ | super S-Box |

By convention, *tuples* of "small" size (typically polynomial in $n$) are numbered starting from 1 and *lists* of "big" size (typically exponential in $n$) are numbered starting from 0. Note that some of these notations overlap (for example, $S$ will denote the AES S-Box, the Shadow S-Box, a GIMLI state) but should nevertheless remain clear from context.

# Chapter 1
# Introduction to Quantum Computing

We open this chapter with an overview of the main principles of *quantum computing*, for the purpose of presenting the main quantum algorithms of Section 1.4 and Chapter 2. More details can be found in the celebrated book of Nielsen and Chuang [NC00], as well as in the lecture notes of de Wolf [dW19]. We then go into the details of *quantum memory models* and define quantum RAM. Among the algorithms that we will present later, some quantum speedups can only be obtained using powerful memory models, which represent optimistic assumptions on the capacities of future quantum hardware. Thus, giving such power to quantum adversaries is a conservative assumption. However, there exists connections between these models, that we will explicit, which will justify to take all of them in consideration in our security analyses. Finally, we present Simon's algorithm [Sim94], arguably the simplest example of an exponential quantum speedup, and cover some of its technicalities in order to prepare Chapter 4. Note that the contents of this chapter may provide a more profound understanding for the curious reader, but most of them are not needed to understand the main results of this thesis. Although we require some technical results to ensure their correctness and success probability, all our quantum algorithms can be phrased as combinations of Simon's algorithm (in this chapter) and quantum search (in Chapter 2), that can be used as black boxes.

## Contents

## 1.1 History

The field of quantum computing originated in the 1980s as a subfield of quantum physics. Some physicists, including notably Richard Feynman [Fey82], had remarked that the task of simulating quantum mechanical systems (via their *wave function*, which describes completely their state) would pose a challenge to classical computers.

Being able to simulate efficiently the evolution of any wave function would enable us to study complex quantum systems such as molecules. This problem, known today as *Hamiltonian simulation*, is the earliest motivation of quantum information science. In order to solve it efficiently, the solution was to implement the simulator as a quantum system itself. Some proposals were given, but the quantum computing model was not completely fleshed out until the definition by Deutsch of the *universal quantum Turing machine* [Deu85], a universal simulator for quantum physics. We shall not go into the details of this definition here; it is equivalent to the *quantum circuits* that we will define in Section 1.2.

From the point of view of *computability*, the universal quantum Turing machine is not more powerful than the universal Turing machine, defined by Turing in his seminal work [Tur36]. Thus, the quantum world does not challenge the Church-Turing thesis:[1]

> *Every function which should naturally be regarded as computable can be computed by the universal Turing machine.*

However, for the same reason that made quantum computers desirable in the first place, the quantum computing model challenges the *strong* Church-Turing thesis, which could be formulated as follows:

> *Any computable function can be computed by the universal Turing machine, with at most polynomial overhead.*

It is not known today whether quantum computers bring a strict computational advantage over classical computers, in terms of complexity classes.[2] In other words, it is not known whether the inclusion: $\mathsf{BPP} \subseteq \mathsf{BQP}$ is strict, where $\mathsf{BPP}$ is the class of problems solvable in probabilistic polynomial-time and $\mathsf{BQP}$ the class of problems solvable by quantum Turing machines in polynomial time.

[1] These formulations of the Church-Turing thesis and strong thesis are taken from [Deu85].

[2] However, this is true for *oracle problems*, where an oracle satisfying certain properties is given as input.

After the definition of the quantum Turing machine, a few early algorithms were developed, such as Deutsch-Jozsa's [DJ92], and the theory of *quantum complexity* appeared [BV93]. But the classical computer science community really became aware of the quantum paradigm shift with Shor's algorithm [Sho94], which solves in polynomial time the factoring and discrete logarithm problems, for which only classical exponential or subexponential algorithms are known. This breakthrough did not only challenge the strong Church-Turing thesis, but also the widely used cryptosystems based on these (classically) difficult problems.

Therefore, in a few years, quantum computers had turned from a physicist's dream into a theoretical framework in complexity theory; another few years later, and they had become the bogeyman of cryptographic schemes. However, despite their success at solving *some* computational problems, it was known very early on that quantum computers would not be all-powerful [Ben+97]. *Post-quantum cryptography* was born with the prospect of finding the strengths and weaknesses of this new attackers, thereby finding new secure designs for an era of quantum computations. This is the main motivation of our work. We defer a brief history of cryptography and the post-quantum perspective to Chapter 3 and, for now, focus only on quantum computing.

## 1.2 Quantum Computations

In this section, we describe *quantum systems* and define the *quantum circuit model*, which is today the most standard way of describing quantum computations.

### 1.2.1 Quantum Systems

We name *quantum system* any physical object, e.g., one or more atoms, of which we can measure some physical property, e.g., the momentum. We assume that observing this property can yield $N$ possible values, which are *classical states*. These measurement outcomes are denoted $|\psi_0\rangle, \dots, |\psi_{N-1}\rangle$.

**Definition 1.1** (Hilbert space)**.** A Hilbert space $\mathcal{H}$ is a vector space over $\mathbb{C}$ equipped with an inner product $\langle \cdot | \cdot \rangle$, which is:

- Positive definite: for all $x \in \mathcal{H}$, $\langle x|x\rangle \geqslant 0$ and $\langle x|x\rangle = 0 \implies x = 0$ ,
- Linear in its first argument,
- Conjugate symmetric: for all $x, y \in \mathcal{H}$, $\langle x|y\rangle = \overline{\langle y|x\rangle}$ .

The notation $|\cdot\rangle$ ("ket") is used for basis vectors; the notation $\langle\cdot|$ ("bra") is used for linear forms (basis vectors of the dual space $\mathcal{H}^*$). These notations are convenient to work with, as we have $\langle\psi|\,|x\rangle = \langle\psi|x\rangle$ ("bracket").

**Definition 1.2** (Quantum state)**.** The *state* of the quantum system is an element of norm 1 of the $N$-dimensional Hilbert space with orthonormal basis $(|\psi_i\rangle)_{0 \leqslant i \leqslant N-1}$, isomorphic to $\mathbb{C}^N$. It is sometimes referred to as as a *pure state*.

Thus, we represent the state as a *superposition* of all these measurement outcomes:

$$|\psi\rangle = \alpha_0 |\psi_0\rangle + \ldots + \alpha_{N-1} |\psi_{N-1}\rangle \ ,$$

where the $\alpha_i$ are complex numbers, coined the *amplitudes*, such that $\sum_i |\alpha_i|^2 = 1$ (normalization condition). A *mixed state* is a probability distribution of pure states. The normalization condition allows to omit global amplitude factors for ease of notation. In the physical world, amplitudes give the probability distribution of measurement outcomes.

**Proposition 1.1** (Measurement)**.** Measuring *the system yields the value* $\psi_i$ *with probability* $|\alpha_i|^2$*. This makes the state of the system* collapse *to* $|\psi_i\rangle$*.*

Note that global *phase* factors can also be omitted, and that states $|\psi\rangle$ and $e^{i\alpha} |\psi\rangle$ are indistinguishable.

Between two measurements, the system can *evolve* through unitary operators of $\mathcal{H}$.

**Definition 1.3** (Unitary operators)**.** A matrix $U \in \mathcal{M}^{N,N}(\mathbb{C})$ describes a unitary operator of $\mathcal{H} \simeq \mathbb{C}^N$ if $U^\dagger U = I$, where $U^\dagger$ is the conjugate transpose of $U$. The amplitude vector $(\alpha_0 \alpha_2 \ldots \alpha_{N-1})^t$ of the state is transformed into $U(\alpha_0 \alpha_2 \ldots \alpha_{N-1})^t$.

The fact that $U$ must be unitary follows from the normalization condition. Since $U$ is unitary, it always admits $U^\dagger$ as an inverse. Thus, the unitary evolution of a quantum system is always reversible. This was soon remarked by physicists [Fey82; Fey85]. The only non-reversible transformation of a quantum state is its measurement. It is also the only way of obtaining information about this state.

### 1.2.2 Quantum Circuits

The quantum circuit model can be seen as a universal, yet simple, way of describing quantum systems and their unitary evolution. We will now define its fundamental building blocks, qubits, named by analogy with classical bits, and quantum gates, analogous to classical logical gates.

**Definition 1.4** (Qubit)**.** A *qubit* is a two-dimensional quantum system with two basis states $|0\rangle$ and $|1\rangle$ (referred to as the *computational basis*). Its state is an element of the two-dimensional Hilbert space $\mathcal{H} \simeq \mathbb{C}^2$:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \text{ where } |\alpha|^2 + |\beta|^2 = 1 \ .$$

A quantum circuit uses a prescribed number of qubits, say $n$, that are initialized to the default state $|0\rangle$.

**Dimensionality.** A system of $n$ qubits, although it contains only $n$ basic components, is described by a quantum state of dimension $2^n$, in the space $\mathcal{H}^{\otimes n}$, where $\otimes$ is the *tensor product*. The canonical basis of $\mathcal{H}^{\otimes n}$ is $|i\rangle, 0 \leqslant i \leqslant 2^n - 1$. Indeed, each basis element $i$, written as an $n$-bit string, represents a possible measurement for the $n$-qubit system, and the quantum state of the system is a superposition over these $2^n$ strings:

$$|\psi\rangle = \alpha_0 |0\cdots 0\rangle + \alpha_1 |0\cdots 01\rangle + \ldots + \alpha_{2^n-1} |1\cdots 1\rangle \enspace .$$

From two states $|\psi_1\rangle$ and $|\psi_2\rangle$, it is naturally possible to consider a joint state, which is generally written $|\psi_1\rangle \otimes |\psi_2\rangle$ or simply $|\psi_1\rangle |\psi_2\rangle$. Because the measurement outcomes of both individual systems are independent, they are said *disentangled*. But in general, the $n$ qubits in a circuit are *entangled*. Entanglement is one of the powerful features of quantum computation, as it allows to work in a space of dimension $2^n$ ($\mathcal{H}^{\otimes n}$) instead of $2n$ ($\mathcal{H}^n$).

**Example 1.1** (Measuring a subsystem)**.** Entanglement between two quantum subsystems is a correlation of their measurement outcomes. As an example, consider an entangled state of two qubits:

$$|\psi\rangle = |+\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle \enspace .$$

Measuring only the first qubit projects the state $|\psi\rangle$ on the compatible subspace. If we obtain 1, then $|\psi\rangle$ is projected to $|11\rangle$, which means that the state of the second qubit is modified as well.

On the contrary, if $|\psi\rangle$ is of the form:

$$|\psi\rangle = \frac{1}{2} |00\rangle + \frac{1}{2} |01\rangle + \frac{1}{2} |10\rangle + \frac{1}{2} |11\rangle \enspace ,$$

then measuring the first qubit projects the second to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ in all cases. They are disentangled, and we can write:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \enspace .$$

**Operations.** We have now set up the system in its basis state $|0\cdots 0\rangle$. Notice that all qubits are disentangled at the beginning. A quantum circuit then applies a sequence of unitary operators known as *quantum gates*. These gates are drawn from some given *universal gate set*, which will contain simple operators acting on one or two qubits at once, with the power of implementing any unitary up to an arbitrary precision. In a quantum circuit, the prescribed quantum gates always form a series of sequential instructions, in contrast to classical circuits, where the gates can also be created in application-specific hardware.

**Single-qubit Gates.** Some well-known single-qubit gates are:

- the *bitflip* gate $\mathsf{X}$, analogous to the classical $\mathsf{NOT}$ gate: it swaps the states $|0\rangle$ and $|1\rangle$ (we will often write it $\mathsf{NOT}$).
- the *phaseflip gate* $\mathsf{Z}$: it does nothing on $|0\rangle$ and changes the complex amplitude of $|1\rangle$ to its opposite.
- the *Hadamard gate* $\mathsf{H}$: it maps $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Notice that $\mathsf{H}$ is involutory (self-adjoint), that is, $\mathsf{H}^\dagger = \mathsf{H}$.
- the *T-gate* $\mathsf{T}$: it rotates the phase of the state $|1\rangle$ by a fixed angle $\pi/4$; that is, it maps $|0\rangle$ to $|0\rangle$ and $|1\rangle$ to $\exp(i\pi/4)|1\rangle$.

As unitary matrices:

$$\mathsf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \mathsf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \mathsf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \mathsf{T} = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{pmatrix} \quad .$$

**CNOT and Toffoli gates.** The standard two-qubit gate is the $\mathsf{CNOT}$ (*Controlled NOT*), which realizes the operation $|a\rangle|b\rangle \mapsto |a\rangle|b \oplus a\rangle$. In other words, it negates the second qubit depending on the first one. Since it acts on two qubits, it is a unitary matrix of dimension $2^2 = 4$:

$$\mathsf{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad .$$

Next, the *Toffoli* gate $\mathsf{Tof}$ [Tof80] acts on three qubits and maps $|abc\rangle$ to $|abc \oplus (a \wedge b)\rangle$. We will also use the identity operator $\mathsf{I}$ to denote that the state remains unchanged. When two operators are applied on two different subsystems, we use a tensor product notation.

**Definition 1.5** (Tensor product of operators)**.** We define:

$$(U_1 \otimes U_2)(|\psi_1\rangle \otimes |\psi_2\rangle) = U_1 |\psi_1\rangle \otimes U_2 |\psi_2\rangle \quad ,$$

and extend this definition to entangled states by linearity.

When applying $n$ copies of an operator, we write $U^{\otimes n}$. A simple example of $n$-qubit unitary operator built from simple gates is the Hadamard transform $\mathsf{H}^{\otimes n}$. It consists in applying a Hadamard gate to each qubit in parallel:

$$\mathsf{H}^{\otimes m} |y\rangle = \sum_{z \in \{0,1\}^m} (-1)^{y \cdot z} |z\rangle \quad .$$

The Hadamard transform is a simple example of the Quantum Fourier Transform. In the one-dimensional version, we replace $\mathbb{Z}_2$ by any $\mathbb{Z}_p$:

$$\mathrm{QFT}_p \left| x \right\rangle = \frac{1}{\sqrt{p}} \sum_{y \in \mathbb{Z}_p} \omega^{yx} \left| x \right\rangle \quad ,$$

where $\omega = e^{2i\pi/p}$, and the $n$-dimensional version is simply a tensor product. One can easily remark that the reverse is another Quantum Fourier Transform replacing $\omega$ by $\overline{\omega}$.

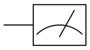
**The Clifford+T gate set.** The universal gate set that we will consider in the following chapters is the *Clifford+T* set, that is the most frequent in the post-quantum cryptographic literature nowadays.

**Definition 1.6** (Clifford set [Got98b])**.** The *Clifford* set is generated by $\mathsf{Z}$, $\mathsf{H}$, $\mathsf{CNOT}$.

The rationale for the Clifford+T set is that quantum circuits generated from Clifford gates can be simulated efficiently: this is the Gottesman-Knill theorem [Got98a]. Adding the T-gate to this set enables one to approximate efficiently *any* gate on one or two qubits, hence any unitary: this is the Solovay-Kitaev theorem [DN06].

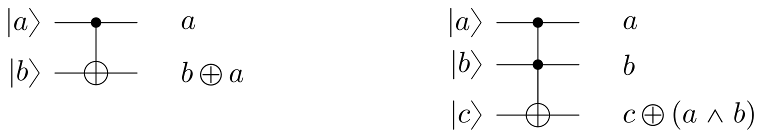
**Theorem 1.1** (Solovay-Kitaev ([NC00], Appendix 3))**.** *One can approximate any one- or two-qubit gate up to error $\epsilon$ using an amount at most* $\mathsf{polylog}\left(\frac{1}{\epsilon}\right)$ *of* $\mathsf{CNOT}, \mathsf{H}$ *and* $\mathsf{T}$ *gates.*

The Toffoli gate, which is convenient to work with in reversible computing, can be implemented using 7 T-gates and 8 Clifford gates.

**Circuit drawings.** In a quantum circuit, individual qubits are represented by wires. We often regroup them in *qubit registers* and use —/— to denote that there are $n$ qubits. Generic operators applied to subsystems are simply represented by boxes. Some elementary gates, such as $\mathsf{CNOT}$ and $\mathsf{Tof}$, have a special representation (Circuit 1.1, Circuit 1.2). The symbol  denotes measurement (the wire stops because the state is destroyed).

**Quantum Circuit 1.1:** Elementary gates $\mathsf{X}$, $\mathsf{Z}$, $\mathsf{H}$, $\mathsf{T}$.

**Physical realizations of quantum circuits.** The quantum circuit model represents the “logical layer” of quantum computations, which is supposed to work perfectly. Throughout this thesis, the words “qubits” and “gates” will refer exclusively to *logical* qubits and gates, not to the physical components that shall lie behind these concepts.



**Quantum Circuit 1.2:** CNOT and Toffoli gates.

As such, the circuit model can be seen as a programming interface, the implementation details being the experimentalists' work. At the physical level, quantum systems are susceptible to *decoherence*, the spontaneous destruction of their state due to unwanted interactions, and the gates induce considerable errors. In order to make a quantum computation fault-tolerant, one must work with bigger systems: any logical qubit would then be implemented with hundreds or thousands of physical qubits, using *quantum error-correcting codes*, so that the errors at the physical level do not disrupt the computation at the logical level. The quantum threshold theorem [Got10] states that if the error rates of physical qubits are small enough, then the errors can be corrected faster than they are created, enabling to run indefinitely a fault-tolerant computation.

Current quantum computing devices belong to the category of *noisy intermediate scale* (NISQ) machines [Pre18], and a fault-tolerant qubit has not yet been built. The recent results of [Aru+19], performing a quantum computation that cannot be simulated on today's classical computers, were obtained in this setting. Meanwhile, most envisioned cryptanalytic applications, including all algorithms presented in the next chapters, require fault-tolerance. But this does not rule out a use of NISQ computations in cryptography.

### 1.2.3 Quantum Algorithms

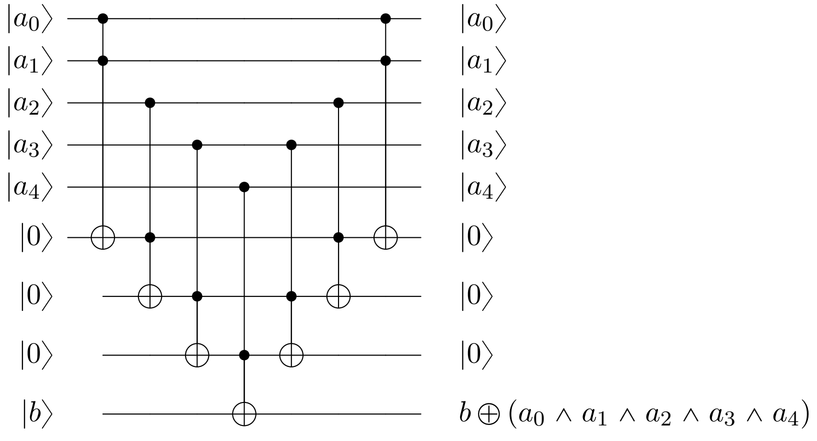
We will describe quantum algorithms using quantum circuits, in the same way as classical algorithms can be described with families of classical circuits.

**Example: the $n$-bit Toffoli gate.** As an example, consider the following operation:

$$\mathsf{nTof} : (a_0, \dots, a_{n-1}, b) \mapsto (a_0, \dots, a_{n-1}, b \oplus (a_0 \wedge \dots \wedge a_{n-1}) \enspace .$$

**Lemma 1.1.** *There exists a quantum circuit for* nTof *using $2n-3$ Toffoli gates and $n-2$ ancilla qubits.*

Here, an *ancilla qubit* is a work qubit that is initialized in the state $|0\rangle$, used during the computation and returned to $|0\rangle$ afterwards; we will often omit them from more complicated quantum circuits. We give an example for $n = 5$ which is adapted from [NC00, Figure 4.10]. Lemma 1.1 follows from a trivial induction. The doubling in cost follows from the necessity of returning the ancilla qubits to the state $|0\rangle$. Other trade-offs are possible, using more Toffolis and less ancilla qubits, as shown in [NC00, Exercises 4.27–4.30]



**Quantum Circuit 1.3:** Simple circuit for a 5-qubit Toffoli gate $\mathsf{5Tof}$.

**Oracles.** Most of the algorithms described in the next chapters access an *oracle* $O_f$ implementing a function $f$. A quantum algorithm is then a family of circuits describing unitary operators interleaved with oracle calls. We will use two equivalent ways to realize a reversible oracle call to a possibly non-invertible function $f$.

**Definition 1.7** (Quantum oracles)**.** Let $f : \{0,1\}^n \to \{0,1\}^m$. The *standard oracle* $\mathsf{SO}_f$ and the *phase oracle* $\mathsf{PO}_f$ act on $n+m$ qubits as:

$$\mathsf{SO}_f \left|x\right\rangle\left|y\right\rangle \mapsto \left|x\right\rangle\left|y \oplus f(x)\right\rangle \quad \text{and} \quad \mathsf{PO}_f \left|x\right\rangle\left|y\right\rangle \mapsto (-1)^{f(x)\cdot y}\left|x\right\rangle\left|y\right\rangle \ .$$

Both are involutory operators.

**Proposition 1.2** (Equivalence of oracles)**.** *The standard oracle and the phase oracle are equivalent under a Hadamard transform on the output register:*

$$\mathsf{PO}_f = (\mathsf{I} \otimes \mathsf{H}^{\otimes m})\mathsf{SO}_f(\mathsf{I} \otimes \mathsf{H}^{\otimes m}) \textit{ and equivalently } \mathsf{SO}_f = (\mathsf{I} \otimes \mathsf{H}^{\otimes m})\mathsf{PO}_f(\mathsf{I} \otimes \mathsf{H}^{\otimes m}) \ .$$

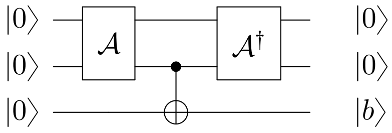
*Proof.* We have:

$$\begin{aligned}
\left|x\right\rangle\left|y\right\rangle &\xmapsto{(\mathsf{I}\otimes\mathsf{H}^{\otimes m})} \left|x\right\rangle\sum_z(-1)^{y\cdot z}\left|z\right\rangle \\
&\xmapsto{\mathsf{SO}_f} \left|x\right\rangle\sum_z(-1)^{y\cdot z}\left|z\oplus f(x)\right\rangle \\
&= \left|x\right\rangle\sum_{z'}(-1)^{y\cdot(z'\oplus f(x))}\left|z'\right\rangle && \text{(Writing } z' = z \oplus f(x)) \\
&= (-1)^{y\cdot f(x)}\left|x\right\rangle\sum_z(-1)^{y\cdot z}\left|z\right\rangle && \xmapsto{(\mathsf{I}\otimes\mathsf{H}^{\otimes m})} (-1)^{y\cdot f(x)}\left|x\right\rangle\left|y\right\rangle \ .
\end{aligned}$$

Once we have obtained the first equality, it suffices to compose by $(\mathsf{I}\otimes\mathsf{H}^{\otimes m})$ on the left and right, since $\mathsf{H}$ is an involution. □

Thanks to Proposition 1.2, we will refer to either oracle by the notation $O_f$, which shall be clear from context.

**Computations and uncomputations.** Since a quantum circuit is no more than the specification of a unitary operator, and a unitary operator is invertible, a quantum circuit always admits an inverse. Furthermore, given the sequence of gates of the circuit, the inverse operator is simply the sequence of their inverses, written backwards. Oracle calls are involutory. Computing $\mathcal{A}^\dagger$ is called *uncomputing* $\mathcal{A}$. It is especially useful if $\mathcal{A}$ produces a meaningful result, e.g., a single bit $b$ that we want to retrieve, but modifies the state in a way that we wish to forget.



**Quantum Circuit 1.4:** Uncomputing a quantum circuit while keeping its result.

**Embedding classical computations.** If $f$ is specified as a classical deterministic algorithm, then the oracle $O_f$ can be implemented with about the same number of operations as $f$. However, since this computation must be reversible, the intermediate memory storage cannot be erased, which can lead to an explosion of memory. Fortunately, it is possible to uncompute the intermediate steps as done in Circuit 1.4. There is an optimal strategy to decide which steps to uncompute at which time, sometimes known as a "pebble game" [Kni95], and it leads to Bennett's time-space trade-off [Ben89; LS90].

**Theorem 1.2** (Theorem 1 in [Ben89])**.** *For any $\varepsilon > 0$, a classical algorithm running in time $T$ and using space $S$ can be simulated by a quantum circuit using $\mathcal{O}\big(T^{1+\varepsilon}\big)$ quantum gates and $\mathcal{O}(S \cdot \ln T)$ qubits (the constants depend on $\varepsilon$).*

More generally, any classical randomized algorithm admits a *quantum embedding*, that is, a quantum algorithm that returns the same distribution of results upon measurement.

**Definition 1.8.** Let $\mathcal{A}$ be a randomized algorithm with no input. A *quantum embedding* for $\mathcal{A}$ is a quantum algorithm $\mathcal{A}'$ that has no input, and the distribution over the possible outcomes of $\mathcal{A}'$ (after measurement) is the same as the distribution over possible outcomes of $\mathcal{A}$.

This quantum embedding admits similar time and space complexities, where classical elementary operations (logic gates) are replaced by quantum gates and classical bits by qubits. Any selection of random bits is replaced by Hadamard gates. Ratios between time complexities are approximately preserved when embedding classical algorithms into quantum algorithms.

**In-place and out-of-place computations.** Most of the classical computations that we will embed in quantum circuits will be *out of place*, meaning that they write their result on a separate register than the input. This is the case with the standard oracle $\mathsf{SO}_f$. However, if $f$ is a permutation, then the transformation $|x\rangle \mapsto |f(x)\rangle$ is unitary. In that case, we say that $f$ is computed *in place*. This is not harder to do if we have the ability to compute $f$ *and* $f^{-1}$ out of place, since we can perform:

$$|x\rangle |0\rangle \mapsto |x\rangle |f(x)\rangle \mapsto |x \oplus f^{-1}(f(x))\rangle |f(x)\rangle = |0\rangle |f(x)\rangle \quad .$$

But in general, when given oracle access to $f$ only, we do not consider the in-place variant of $O_f$.

**The deferred measurement principle.** Only a measurement can extract information from the system. However, during a quantum computation, intermediate measurements are not necessary. The circuit will return the same final result if they are *not* performed. Despite this fact, these partial measurements have some advantages. By turning pure states into mixed states, they can help to simplify the presentation of an algorithm. They can also help to reduce the amount of ancilla qubits, by measuring them and reusing them, as in the variant of Shor's algorithm by Mosca and Ekert [ME98].

### 1.2.4 Quantum Complexity Notions

In order to estimate their advantage over classical algorithms, we need asymptotic and non-asymptotic notions of complexities for quantum algorithms, described as quantum circuits.

**Query complexity.** Given access to the oracle $O_f$, we count the number of queries made to $O_f$.

**Time complexity.** We count the number of quantum gates performed. When using $\mathcal{O}$ notations, we do not need to specify a universal gate set, as all good universal sets are equivalent up to a constant factor. For performing detailed quantum gate counts, we use the Clifford+T gate set, as it has become the most widely used for benchmarking cryptanalytic algorithms [Amy+16; Gra+16; Hän+20; Jaq+20].

**Comparing quantum and classical complexities.** There does not exist a clear common benchmark for comparing quantum and classical operations. Common sense dictates us that, in the foreseeable future, quantum gates are likely to remain orders of magnitude slower than classical gates, and qubits more costly than classical bits. This justifies to look for unbalanced trade-offs between quantum and classical resources. This is done for example in [Bia+19; BS20] in the study of some quantum algorithms with cryptanalytic applications. However, it is sometimes easier to think of this gap as a mere constant and equalize asymptotic classical and quantum times.

**Notations.** We will denote by $\mathsf{T_c}(\mathcal{A})$ the *classical* time complexity of a classical algorithm $\mathcal{A}$, $\mathsf{T_q}(\mathcal{A})$ the *quantum* time complexity of a quantum algorithm $\mathcal{A}$ and $\mathsf{M}(\mathcal{A})$ the *memory* complexity of an algorithm, for a given model of memory.

## 1.3 Quantum Memory Models

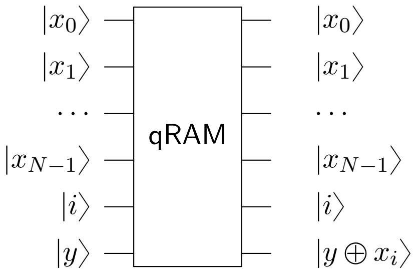
Quantum computing machines from a near future will likely have only a few qubits available, which is why the topic of this section is mostly theoretical. But the study of quantum algorithms using massive amounts of qubits, or even more hypothetical hardware capabilities, can be a first step towards more constrained resources and quantum circuits of smaller area.

In particular, many classical algorithms relevant for cryptanalysis require read and write access to a random-access memory. In the analysis of these algorithms, we often suppose that these operations take a constant time, though this is an assumption that does not hold at large scales, as discussed in [Wie04].

When trying to make quantum versions of these algorithms, we often fall upon the need for memory operations of the same efficiency, although they are now handling quantum data. This is where we need to define *quantum Random-Access Memory* (qRAM) and all its variants. qRAM is ubiquitous in the quantum computing literature, but often implicit. In this section, definitions are borrowed from [Amb07; Kup13] and technical results are from [Jef14] or folklore.

### 1.3.1 Variants of qRAM

As its core, the quantum random-access model relies on a new quantum gate, the $\mathsf{qRAM}$ gate (see, for example, [Amb07, Section 6.1]), which is assumed to have unit cost regardless of the number of qubits it spans. Assume that we have $N$ data registers $|x_0\rangle, \dots, |x_{N-1}\rangle$ and an index register $|i\rangle$. Then the gate performs the unitary of Circuit 1.5.



**Quantum Circuit 1.5:** The $\mathsf{qRAM}$ gate.

The data at index $i$ is accessed and put into the output register $y$ (which is always quantum). Depending on whether the data in $|x_0\rangle, \dots, |x_{N-1}\rangle$, and the index register, are classical or not, we obtain the four models of Table 1.1.

**Table 1.1:** qRAM variants.

| Memory ($x_j$)<br>Index ($i$) | Classical | Superposition |
|---|---|---|
| Classical | Classical RAM | Quantum circuit model |
| Superposition | QRACM | QRAQM |

In this thesis, we will use the terms "QRACM" and "QRAQM" in a technical context, and "qRAM" in a broader sense, for any of them.

### 1.3.2 Quantum Data Structures using qRAM

Using qRAM gates, it is possible to obtain quantum data structures with fast insertion and lookups. If we want only to implement dichotomy search in a sorted list, without inserting, this is doable without any errors using a logarithmic number of qRAM gates. If we want both insertion and search in a logarithmic number of gates, this is allowed by the *radix tree* data structure presented in [Jef14]. Many algorithms that we will consider in this thesis, for example in Chapter 5 and Chapter 6, can make use of a simpler data structure for membership testing. We only need to assume that the list accessed contains random bit-strings with distinct prefixes.

We define a *Unique prefix list* as follows (we refer to Algorithm 2.10 for a use case). This is a list $\mathcal{L}$ holding $n$-bit strings indexed by $m$-bit prefixes ($m \leqslant n$), which are also their addresses. Memory cells are $n+1$-bit or qubit registers initialized to a special state $\perp$. Thus, the list requires exactly $(n+1)2^m$ bits (for QRACM) or qubits (QRAQM) of storage. It can only contain a single element of a given $m$-bit prefix, and is of maximal size $2^m$ (alternatively, we may authorize a constant number of elements of a given prefix, which we would put in a "bucket" of fixed size). It supports the following operations, which use $\mathcal{O}(1)$ qRAM gates and $\mathcal{O}(n)$ basic gates each:

- Search($s$), where $s$ is a bit-string of size $m$: returns the element of prefix $s$ if there is one, and $\perp$ otherwise. This requires simply to we fetch the value of the register at index $s$.

- Membership($x$), where $x$ is a bit-string of size $n$: returns 1 if $x \in \mathcal{L}$ and 0 otherwise. This is done by calling Search($\text{trunc}_m(x)$) with the $m$-bit prefix of $x$ and checking whether the result is $\perp$.

- SearchMany($s$), where $s$ is a bit-string of size smaller than $m$: returns the uniform superposition of all $x \in \mathcal{L}$ that have prefix $s$, or $\perp$ if there is no such $x$.

If all memory cells with prefix $s$ are occupied, SearchMany can easily be implemented with a few Hadamard gates and a qRAM gate, and it is exact. If this is not the case, then among the cells that we query, there are unwanted $\perp$ values. However, if we know the size of $s$ in advance, it is possible to make SearchMany exact using an exact Amplitude Amplification (Theorem 2.3).

### 1.3.3 Emulating qRAM Gates

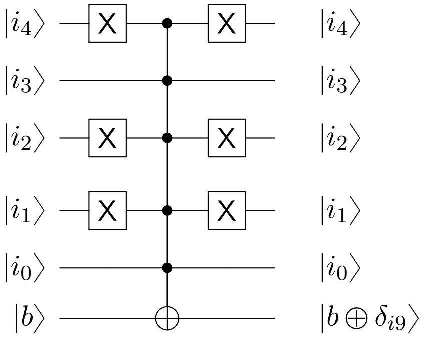
It is well known, and will be of particular importance for us (in Section 5.1 for example), that QRACM can be emulated with *purely classical storage*.

**Lemma 1.2** (QRACM emulation)**.** *A qRAM gate accessing* $2^n$ *classical memory cells of* $m$ *bits each can be replaced by a quantum circuit using* $(2n-3)2^n$ *Toffoli gates (none if* $n \leqslant 1$*),* $n2^n$ $\mathsf{NOT}$ *gates,* at most $m2^n$ $\mathsf{CNOT}$ *gates and* $n-1$ *ancillas qubits (none if* $n \leqslant 1$*).*

*Proof.* Let $\delta_{ij}$ be the Kronecker symbol: $\delta_{ij} = 1 \iff i = j$ and $\delta_{ij} = 0$ otherwise. For a given $j$, it is easy to implement a circuit that computes $i \mapsto \delta_{ij}$. Let us take 5-bit integers as an example, and $j = 9 = \overline{01001}^2$. Then if we write $i = \overline{i_4 i_3 i_2 i_1 i_0}^2$:

$$\delta_{ij} = \neg i_4 \wedge i_3 \wedge \neg i_2 \wedge \neg i_1 \wedge i_0 \ \ ,$$

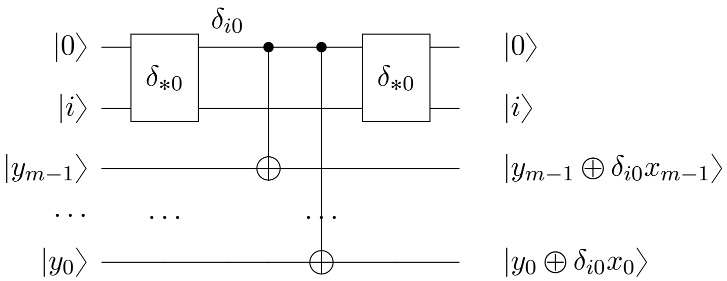
which leads to Circuit 1.6 using the 5-bit Toffoli gate of Circuit 1.3 and $\mathsf{NOT}$ gates $\mathsf{X}$.



**Quantum Circuit 1.6:** Quantum circuit for $\delta_{*9}$.

In general, we only need a single $n$-bit Toffoli gate and at most $2n$ $\mathsf{NOT}$ gates for indices of $n$ bits. Next, assume without loss of generality that we access the memory cell at index 0, containing a classical value $x = \overline{x_{m-1} \dots x_0}^2$. We apply the unitary $|i\rangle |y\rangle \mapsto |i\rangle |y \oplus \delta_{i0} x\rangle$, using our circuit for $\delta_{*0}$ and at most $m$ more $\mathsf{CNOT}$ gates: if $x_i = 1$ then we apply a $\mathsf{CNOT}$ on $y_i$, controlled by $\delta_{i0}$ (i.e., we compute $y_i \oplus \delta_{i0}$). Otherwise we do nothing.

Next, remark that between the two circuits for $\delta_{*0}$ in Circuit 1.7, we can remove the redundant $\mathsf{X}$ gates and all the uncomputation phase of the $n$-bit Toffoli circuits. That



**Quantum Circuit 1.7:** Quantum circuit for accessing a classical memory cell.

is, we compute only a single $n$-bit Toffoli, and in the middle, apply the CNOT gates. So Circuit 1.7 requires only $2n-3$ Toffolis and $n-2$ ancillas by Lemma 1.1. We obtain a circuit of *at most* $2n-3$ Toffolis, $m$ CNOTs and $n-1$ ancillas for computing $y \oplus \delta_{i0} x$ bit by bit. We do that $2^n$ times. We also remark that since $j$ spans all memory indices, the total number of NOTs used is $n2^n$. □

It is also easy to use such a circuit in the case where the data accessed is part of the quantum circuit. Instead of using a specific sequence of CNOT gates in Circuit 1.7, we replace them by Toffoli gates controlled by $\delta_{i0}$ and by the qubits that we are accessing.

**Corollary 1.1.** *A qRAM gate spanning $2^n$ data registers of size $m$ each can be emulated with $n2^n$ NOT gates, $(\max(2n-3, 0) + m)2^n$ Toffoli gates and $n-1$ ancilla registers.*

Thus, the separation between memory models becomes significant only when massive amounts of memory are used. If the memory size is small (say, polynomial in the dimension $n$ of the problem), then swapping memory models does not modify the gate complexity by more than a factor $\mathsf{poly}(n)$.

## 1.3.4 Membership Queries

Another version of Lemma 1.2 allows us to bypass completely a quantum data structure if we want simply to test membership to a list $\mathcal{L}$ of $N$ elements. This will be used later in Chapter 5 (see Algorithm 5.1), but also in Chapter 9.

**Lemma 1.3** (Membership query). *Let $\mathcal{L} = x_0, \ldots, x_{N-1}$ be a list of distinct $m$-bit strings of size $N$, with $m = 2m'$ and $m - 3 \geqslant 1$. There exists a quantum circuit $\mathsf{Membership}_{\mathcal{L}}$ that performs:*

$$|y\rangle |b\rangle \xmapsto{\mathsf{Membership}_{\mathcal{L}}} |y\rangle |b \oplus (y \in \mathcal{L})\rangle \ ,$$

*using less than $2m$ NOTs, $mN$ CNOTs and exactly $(m-3)N + m$ Toffolis.*

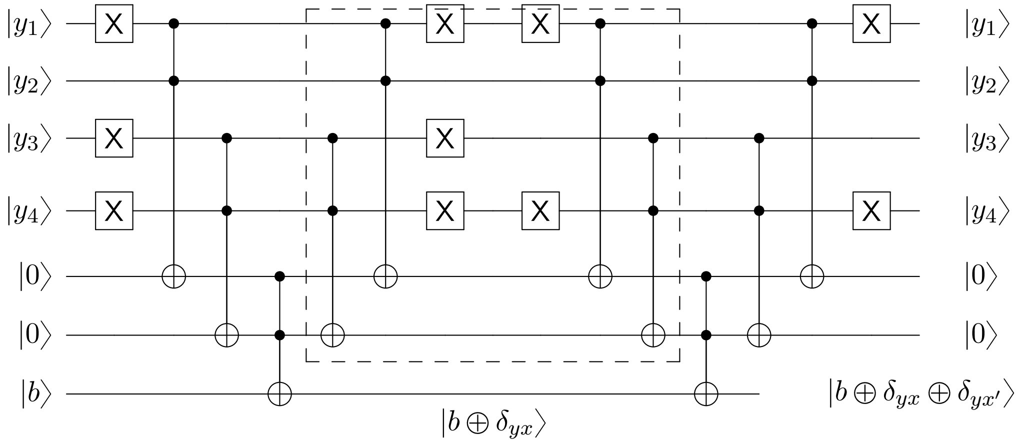
*Proof.* Notice that $\mathcal{L}$ does not need to be sorted, but sorting it beforehand can help to ensure that all its elements are distinct. We can simply reuse the circuit for the

Kronecker symbol (Circuit 1.6), which requires NOTs and a single $m$-bit Toffoli gate. For each (classical) value $x \in \mathcal{L}$, we apply the unitary:

$$|y\rangle |b\rangle \mapsto |y\rangle |b \oplus \delta_{yx}\rangle \quad .$$

Applying all unitaries in sequence is enough, since $\mathcal{L}$ contains distinct elements, as we can write:

$$(y \in \mathcal{L}) = \delta_{yx_0} \oplus \delta_{yx_1} \oplus \ldots \oplus \delta_{yx_N} \quad .$$



**Quantum Circuit 1.8:** Cancellation of Toffoli gates in the membership circuit.

Next, we remark that half the gates in the circuit can be removed. Since we assumed for simplicity an even $m$, in the Kronecker symbol computations, we can replace the $m$-bit Toffoli gates by $m'$ Toffolis, an $m'$-bit Toffoli, and $m'$ Toffolis again. Then half of the work consists in computing a Toffoli gate, then applying zero, one or two NOTs on the control wires, then applying a Toffoli again. If there are no NOTs, the Toffoli cancel out. If there is only one, we are computing:

$$|a\rangle |b\rangle |c\rangle \mapsto |a\rangle |b\rangle |(c \oplus (a \wedge b) \oplus (\bar{a} \wedge b)\rangle = |a\rangle |b\rangle |c \oplus b\rangle \quad ,$$

and if there are two:

$$|a\rangle |b\rangle |c\rangle \mapsto |a\rangle |b\rangle |(c \oplus (a \wedge b) \oplus (\bar{a} \wedge \bar{b})\rangle = |a\rangle |b\rangle |c \oplus a \oplus b\rangle \quad .$$

Thus, we can replace them by CNOTs in all cases, removing these Toffolis. □

The upper bound of Lemma 1.3 is valid for all lists. For a given one, it is likely that the circuit can be optimized further, for example by computing the Kronecker symbols in a sequence that maximizes the overlap between consecutive elements of $\mathcal{L}$. But a simple counting argument shows that we cannot do asymptotically much better.

**Proposition 1.3.** *Fix a universal set of $c$ gates of bounded arity $r$. Let $R_{q,N,m}$ be the smallest number of quantum gates such that, for all lists $\mathcal{L}$ of $m$-bit strings of size $N$, there exists a circuit using $q$ qubits (including ancillas), with at most $R_{q,N,m}$ gates computing* $\mathsf{Membership}_{\mathcal{L}}$*. Then $R_{q,N,m} \geqslant \frac{mN}{\log_2 c + r \log_2 q}$ .*

*Proof.* There are less than $\left(c\binom{q}{r}\right)^R$ such quantum circuits with less than $R = R_{q,N,m}$ gates, since at each new gate, we select its $r$ input qubits among $q$. There are $2^{mN}$ different lists $\mathcal{L}$, which need different circuits. Hence:

$$\begin{aligned}\left(c\binom{q}{r}\right)^R \simeq c^R q^{rR} \geqslant 2^{mN} &\implies R(\log_2 c + r\log_2 q) \geqslant mN \\ &\implies R \geqslant \frac{mN}{\log_2 c + r\log_2 q} \ .\end{aligned}$$

Thus, with $r$ and $c$ constants, and if the circuit uses $q = \mathcal{O}(m)$ qubits, we cannot hope to go below $\mathcal{O}(mN/\log m)$. □

**Replacing QRACM with classical sequential-access memory.** A direct consequence of Lemmas 1.2 and 1.3 is that QRACMs, even of exponential size, can be removed from quantum algorithms making only few membership queries (or performing few qRAM gates).

**Corollary 1.2.** *Let $\mathcal{A}$ be a quantum algorithm making $Q$ membership queries to a classical list $\mathcal{L}$ of size $L$, using $m$ qubits and $T$ other gates. Then there exists an equivalent algorithm $\mathcal{A}'$ using $m$ qubits, running in time $\widetilde{\mathcal{O}}(QL + T)$, that uses a* classical memory with sequential access *(CSAM) of size $L$.*

Each time $\mathcal{A}$ accesses $\mathcal{L}$, we use the circuit $\mathsf{Membership}_{\mathcal{L}}$. The gates that we apply depend on the elements of $\mathcal{L}$, which is why it must still be stored. But this storage is purely classical, and with *sequential access*. Indeed, while each call of $\mathsf{Membership}_{\mathcal{L}}$ requires to go through the whole list $\mathcal{L}$, the elements form a sequence of instructions to the quantum processing unit, that only needs to be read in fixed order. Physically, we can imagine that $\mathcal{L}$ is stored in a hard drive, with a single magnetic head. This type of memory is cheaper and does not suffer from scaling energy consumption.

### 1.3.5 Discussion and Comparisons

The qRAM gate is obviously powerful, as it allows to span $M$ components in a single time step. Physical realizations of qRAM would have many applications in different areas of quantum computing, not limited to quantum cryptanalysis.

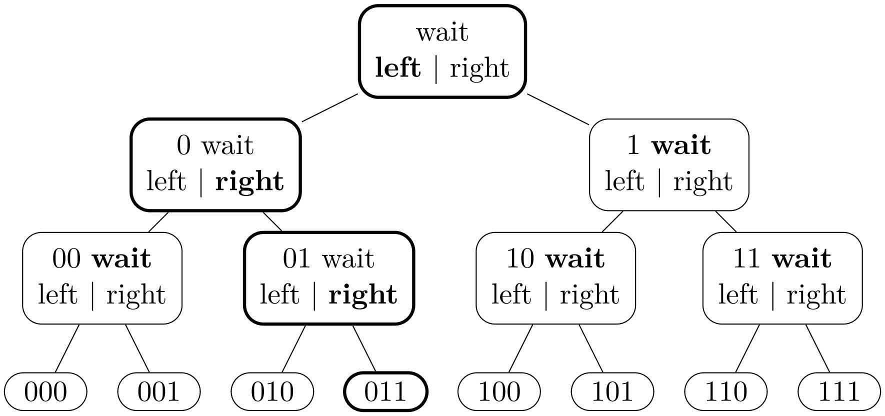


**Figure 1.1:** State of a bucket-brigade qRAM with 8 registers and 7 switches, after reading the address 011 (0 = left, 1 = right).

**Circuits vs. QRACM.** The specificity of quantum memory models is that the quantum circuit model and the QRACM model are competitors: we do not know, and we cannot predict whether a QRACM of size $M$ would cost more to maintain than a quantum circuit with $M$ qubits. In fact, some authors have argued that specific architectures for QRACM may turn out to be cheaper than plain qubits [Kup13]; others have considered the opposite [GR04].

**From QRAQM to plain circuits.** Even if we consider it as a blurry horizon in a distant future, the most powerful model of QRAQM remains a meaningful setting, not only because we may be interested in worst-case scenarios. Throughout the following chapters, we encounter some algorithms that *seem* to require QRAQM but admit QRACM-only variants, and some others that *seem* to require QRACM but admit CSAM-only variants. Hence, time-efficient quantum algorithms in the QRAQM model may be a "first step" towards versions with lower hardware requirements. This is why, in the following, we will encounter and consider all three settings: plain circuits (CSAM or RAM), QRACM, QRAQM.

**An Example of qRAM architecture.** In [GLM08], Giovannetti, Lloyd, and Maccone proposed the *bucket-brigade* architecture.

They start with the standard arrangement of the $2^n$ memory cells in leaves of a tree of depth $n$, with $2^n - 1$ switches that decide whether to go to left (0) or to right (1) depending on the requested address. In the bucket-brigade qRAM (Figure 1.1), each switch is a *trit* and can be either in the state *left*, *right* or *wait*. All switches start in the state *wait*. Address bits are read one by one. When a bit $b$ arrives at a switch, if the switch is in the state *wait*, then it is changed to the state *left* or *right* depending on $b$. If the switch is in the *left* or *right* state, then it routes $b$ accordingly to the next switch.

Translating a bucket-brigade call into a quantum circuit gives a complexity similar to Lemma 1.2, because all paths are explored in superposition, and all switches modified. However, this circuit will be massively parallel, with a depth $\mathcal{O}(n)$ and a width of $\mathcal{O}(n2^n)$ qubits. The other main feature of this architecture is its error-resilience, which stems from the fact that most of the switches remain in the state *wait*. In any basis vector of the superposition, only $\mathcal{O}(n)$ switches are in the state *left* or *right*. Thus, a practical advantage of qRAM may come from a reduced cost of error-correction with respect to plain quantum circuits. However, in [Aru+15], the authors questioned the robustness of this model with respect to errors. In particular, the error per gate would need to be inversely proportional to the number of queries performed to the qRAM.

The current research seems to focus on qRAM as components of hybrid quantum architectures [Ble10; Hon+12].

## 1.4 Simon's Algorithm

Simon's algorithm [Sim94] is the simplest member of the family of *Hidden subgroup algorithms*, and the precursor of Shor's algorithm [Sho94]. It has many use cases in quantum symmetric cryptanalysis and will be one of the most used tools in the following chapters. In this section, we introduce the problem and the algorithm.

### 1.4.1 Simon's Problem

The problem to solve is the following.

**Problem 1.1** (Boolean hidden period)**.** *Let $X$ be a set. Suppose given access to a function $f : \{0,1\}^n \to X$ such that there exists $s \in \{0,1\}^n$ with:*

$$\forall x, f(x) = f(y) \iff y = x \text{ or } y = x \oplus s \ ,$$

*then find $s$.*

In other words, we must determine if a given function is periodic ($s \neq 0$) or injective ($s = 0$) given the *promise* that this is one of the cases, and we must find its period. From now on, we will assume that $X = \{0,1\}^n$ for simplicity. Simon's problem can be seen as a hidden period problem, but also as a hidden subgroup problem for a subgroup of $(\mathbb{Z}_2)^n$, and last but not least, it is equivalent to the Boolean *hidden shift* problem.

**Problem 1.2** (Boolean hidden shift)**.** *Given access to two functions $f, g : \{0,1\}^n \to X$ that either have no image in common, or satisfy $f(\cdot) = g(\cdot \oplus s)$ for some $s$, find $s$.*

*Proof of equivalence.* An algorithm for the Boolean hidden shift problem solves the hidden period problem by setting $f = g$. Next, given an instance $f, g$ for the hidden shift problem, we define:

$$F : \{0,1\} \times \{0,1\}^n \to X$$
$$(b, x) \mapsto \begin{cases} f(x) \text{ if } b = 0 \\ g(x) \text{ if } b = 1 \end{cases} .$$

Then $F$ admits the period $(1, s)$. □

Solving this problem with classical oracle access to $f$ requires $\Omega\left(2^{n/2}\right)$ queries, as this is the expected number of queries before a *collision pair* $x, y, f(x) = f(y)$ occurs in the periodic case. If we don't find such a pair, then we can conclude that the function is injective. Simon [Sim94] gives a probabilistic quantum polynomial-time algorithm which only requires $\mathcal{O}(n)$ queries to $O_f$. An exact version was later given by Brassard and Høyer [BH97], who also extended the algorithm to the case where $f$ admits multiple periods and the whole subspace of periods must be retrieved.

### 1.4.2 Description of the Algorithm

Simon's algorithm relies on a subroutine FindVector, which is depicted in Circuit 1.9. In short, it first creates a superposition of the two preimages of a random element of the codomain of $f$. By making these two elements interfere (constructively or destructively), it allows to sample a random vector orthogonal to the period. In Algorithm 1.1, we follow the evolution of the quantum state through the operations applied.

We choose a number $m$ constant or linear in $n$. Simon's algorithm (Algorithm 1.2) consists in running $n + m$ times the subroutine FindVector (Algorithm 1.1), creating an $(n+m) \times n$ matrix $Y$ out of the $y$ obtained, and computing its rank $r$. If $r < n-1$, then the algorithm has failed. If $r = n$, then we know for sure that the function is injective; if $r = n - 1$, then the linear system $YS = 0$ has a single solution, which should be $s$.

The probability of failure is governed by the value of $m$, due to Proposition 1.4.

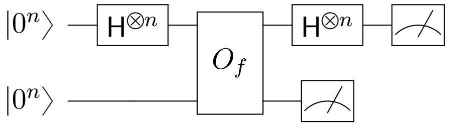
**Proposition 1.4** (Failure probability in Simon's algorithm)**.** *Algorithm 1.2 returns an incorrect result with probability* $\leqslant 2^{-m}$.

*Proof.* The two cases can be brought down to the same computation: given $r+m$ vectors $y_1, \ldots, y_{r+m}$ drawn uniformly at random from an $\mathbb{F}_2$-vector space $V$ of dimension $r$, we lower bound the probability that these vectors contain a basis of $V$. When $s = 0$, $V = \mathbb{F}_2^n$ and when $s \neq 0$, $V$ is the subspace orthogonal to $s$.

We use a remark from [SS17]. A family of $r$ binary vectors of length $r + m$, drawn uniformly at random, has full rank with probability:

$$\prod_{i=0}^{r-1} \left(1 - \frac{1}{2^{r+m-i}}\right) \geqslant 1 - 2^m \ ,$$

since the first vector is zero with probability $2^{-(r+m)}$, and then, each new column must be drawn outside the subspace of size $2^i$ generated by the previous $i$ vectors. Then, by equality of row and column rank, we obtain the result. □



**Quantum Circuit 1.9:** Quantum circuit of FindVector.

**Algorithm 1.1** FindVector subroutine.

```
    Input: oracle access to $O_f$
    Output: $y$ (selected uniformly at random) such that $y \cdot s = 0$
1:  Start in the all-zero state.   ▷ $|0^n\rangle |0^n\rangle$
2:  Apply a Hadamard transform.
                                   ▷ $\sum_{x \in \{0,1\}^n} |x\rangle |0\rangle$
3:  Apply $O_f$.                   ▷ $\sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$
4:  Rewrite the current state depending on the value in the second register.
                                   ▷ $\sum_{a \in X} \left( \sum_{x \in f^{-1}(a)} |x\rangle \right) |a\rangle$
5:  Measure the second register and obtain a value $a \in \{0,1\}^n$, drawn uniformly at
    random. The first register collapses to all preimages of $a$.
                                   ▷ $\sum_{x \in f^{-1}(a)} |x\rangle$
6:  Apply a Hadamard transform $\mathsf{H}^{\otimes n}$.
                                   ▷ $\sum_{y \in \{0,1\}^n} \left( \sum_{x \in f^{-1}(a)} (-1)^{x \cdot y} \right) |y\rangle$
7:  Measure the second register.
      • If $s = 0$ then $f$ is injective. The amplitude of each $y \in \{0,1\}^n$ is $\pm 1$ (up to a
        global normalization factor), hence each $y$ has the same probability of being
        measured.
      • If $s \neq 0$ then $f$ is periodic. The value $a$ measured in Step 5 has exactly two
        preimages $x_0$ and $x_0 \oplus s$ for some $x_0$. The amplitude of $y$ is:
            $\sum_{x \in f^{-1}(a)} (-1)^{x \cdot y} = (-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus s) \cdot y}$ ,
        which is zero if $y \cdot s = 1$ and non-zero otherwise.
    In all cases, we obtain a uniformly random $y$ such that $y \cdot s = 0$.
```

**Algorithm 1.2** Simon's algorithm.

```
Input: oracle access to $O_f$ (injective or periodic), $t \in \mathbb{N}$
Output: the value $s$
Set $Y := \varnothing$
Repeat $n + m$ times
    $y \leftarrow \mathsf{FindVector}(O_f)$
    $Y \leftarrow Y \cup \{y\}$
EndRepeat
$r \leftarrow \mathsf{rank}(Y)$
if $r = n$ then
    return 0
else if $r = n - 1$ then
    return the single $s$ orthogonal to $Y$
else
    return Failure
```

Thus, Simon's algorithm runs in time $\mathcal{O}(n^3)$, using $\mathcal{O}(n)$ queries, and achieves an exponential success probability.

### 1.4.3 Weakening the Promise

In a cryptographic context, the functions on which we apply Simon's algorithm are often not exactly injective, and random collisions may occur. Let $\{0,1\}^n/(s)$ be the equivalence classes of $\{0,1\}^n$ under the relation: $x \equiv y \iff x = y$ or $x = y \oplus s$.

**Definition 1.9** (Random periodic function ([Bon19], Definition 4.1))**.** A function $f : \{0,1\}^n \to X$ of period $s$ is *random periodic* if $f$ restricted to $\{0,1\}^n/(s)$ is a random function.

**Problem 1.3** (Simon's problem with random functions)**.** *Suppose given access to a function $f : \{0,1\}^n \to \{0,1\}^n$ which is either a random function, or a random periodic function of period $s$. Determine the case and find $s$.*

The soundness of Algorithm 1.2 applied to random functions has already been investigated in the literature. In [Kap+16a], the authors relate the probability of failure to the quantity:

$$\epsilon(f,s) = \max_{t \in \{0,1\}^n \setminus \{0,s\}} \Pr_x \left( f(x) = f(x \oplus t) \right) \quad . \tag{1.1}$$

This notation naturally extends to both the periodic case ($s \neq 0$) and the aperiodic case ($s = 0$), and we write $\epsilon(f) = \epsilon(f,s)$.

**Theorem 1.3** ([Kap+16a], Theorem 1)**.** *If $\epsilon = \epsilon(f) < 1$, then after $n + r$ queries, Simon's algorithm returns $s$ with a probability greater than $1 - 2^n \left(\frac{1+\epsilon}{2}\right)^{n+r}$.*

In short, $\epsilon(f)$ controls the weight of unwanted periods of $f$. If $\epsilon(f) = 0$, then we fall back in the perfect case. On the contrary, if $\epsilon(f) = 1$, then $f$ is constant and $s$ cannot be recovered at all.

*Proof of Theorem 1.3.* After having measured $n + r$ vectors $y_1, \ldots, y_{n+r}$, the algorithm fails if all the vectors are orthogonal to a given $t \neq s$.

In the subroutine FindVector, if the $x_0$ in the last step is such that $f(x_0 \oplus t) = f(x_0)$, then the vector $y_i$ measured will be orthogonal to $t$ with probability 1. Otherwise, it will be orthogonal to $t$ with probability $\frac{1}{2}$. The first case occurs with probability $\epsilon$, the second with probability $1 - \epsilon$. All in all, we have $\Pr(y_i \perp t) = \frac{1+\epsilon}{2}$.

We do a union bound over all $t$ and obtain:

$$\Pr(\dim(\mathrm{Span}\,(y_1, \ldots, y_{n+r})) \leqslant n - 2) \leqslant 2^n \left(\frac{1+\epsilon}{2}\right)^{n+r} .$$

The bound is the same in the case where there is no period. □

**Bound of $\epsilon(f)$ for random functions.** In quantum cryptanalysis, Simon's algorithm is often applied to functions of which it is safe to assume that they are drawn uniformly at random. Any other behavior may denote a weakness that should be cryptographically exploited. For a random function, $\epsilon(f)$ is negligibly small [DR07]. We give below an upper bound taken from [Bon20].

**Lemma 1.4.** *For a function $f$ drawn uniformly at random from $(\{0,1\}^n \to \{0,1\}^n)$, for any $a > 0$:*

$$\Pr_f\left(\epsilon(f) \geqslant \frac{(a+1)n}{2^n}\right) \leqslant 2^{-an} . \tag{1.2}$$

*Proof.* Let $m$ be a bound to choose later. Consider a fixed value of $t \in \{0,1\}^n$. We count the number of functions such that *at least* $m$ elements $x$ are sent to the same image as $x \oplus t$. Since we just have to choose these $x$ elements, their images, and complete the function, there are $\binom{2^n}{m} 2^{nm} 2^{n(2^n - 2m)} \leqslant \frac{2^{nm}}{m!} 2^{n(2^n - m)}$, hence a proportion $\frac{1}{m!}$ of them at most. Hence, for a given $t$:

$$\Pr_f\left(\Pr_x\left(f(x) = f(x \oplus t)\right) \geqslant \frac{m}{2^n}\right) \leqslant \frac{1}{m!} .$$

Next, we do a union bound on all values of $t$:

$$\Pr_f\left(\exists t, \Pr_x\left(f(x) = f(x \oplus t)\right) \geqslant \frac{m}{2^n}\right) \leqslant \frac{2^n}{m!} ,$$

and replace by $\epsilon$:

$$\Pr_f\left(\epsilon(f) \geqslant \frac{m}{2^n}\right) \leqslant \frac{2^n}{m!} .$$

It remains to choose $m$ appropriately: $m = (a+1)n$ is enough for our result, and for $m \geqslant 4$, we have $m! \geqslant 2^m$. □

Thus, even if we consider families of random functions of exponential size, we can assume that $\epsilon(f)$ is less than $\frac{an}{2^n}$ *for all functions simultaneously*, for an accordingly chosen constant $a$, by Lemma 1.4 and a union bound. Next, we show that:

**Lemma 1.5.** *Assume that $\epsilon(f) \leqslant \frac{an}{2^n}$ for some constant $a$. Then for sufficiently big $n$, Simon's algorithm fails with probability lower than $2^{-r}e$ after $n+r$ queries.*

*Proof.* By Theorem 1.3, the probability of error is lower than:

$$\begin{aligned} 2^n \left(\frac{1+\epsilon}{2}\right)^{n+r} &= (1+\epsilon)^{n+r} 2^{-r} \\ &\leqslant e^{(n+r)\epsilon} 2^{-r} \\ &\leqslant e^{a(n+r)n/2^n} 2^{-r} \leqslant e 2^{-r} \quad , \end{aligned}$$

where the last inequality holds if $n$ is not too small (cryptographically relevant values of $n \geqslant 80$ are all but enough). □

All in all, Simon's algorithm applies very well to cryptographic functions or permutations exhibiting a hidden shift, as soon as they do not contain another unwanted structure.

### 1.4.4 Simon's Algorithm as a Quantum Circuit

We have presented Simon's algorithm as a quantum *procedure*, which contains many measurements. For some applications, we will need a fully reversible implementation of this algorithm.

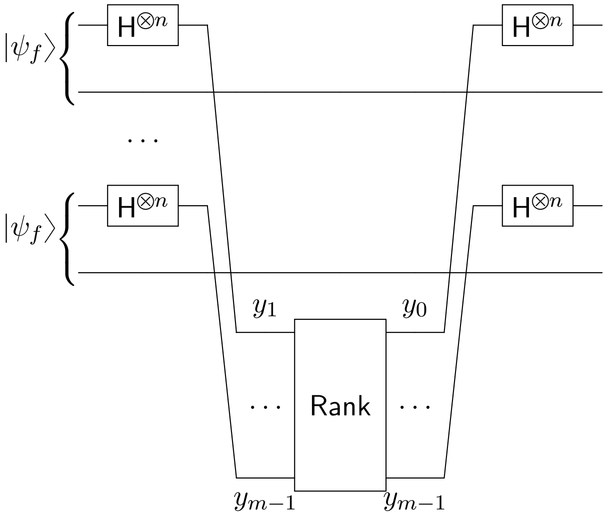
In total, Simon's algorithm makes $n+r$ queries to $O_f$ where $r$ allows to control its probability of failure. The inputs of these queries are disentangled copies of $\sum_x |x\rangle |0\rangle$. Hence, we can consider instead that all the queries are done beforehand, and given to a quantum circuit $\mathsf{QSimon}$. For our purposes, $\mathsf{QSimon}$ only needs to perform a phase shift of the state if the function is periodic, and nothing otherwise.

We will show that, while an exact $\mathsf{QSimon}$ seems hard to realize, it is as easy to bound the induced error as to bound the error in Simon's algorithm. In the following, let us denote $|\psi_f\rangle = \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$ a quantum state that contains one of the queries to $O_f$.

**Theorem 1.4** (Quantum circuit for Simon's algorithm). *Assuming that $\epsilon(f) \leqslant \frac{an}{2^n}$ for some $a$, where $\epsilon$ is defined as in Equation (1.1), and setting $m = n+r = \mathcal{O}(n)$, there exists a quantum circuit $\mathsf{QSimon}$ using $\mathcal{O}(n^3)$ elementary gates that realizes:*

$$\left(\bigotimes_{i=0}^{m-1} |\psi_f\rangle\right) \xmapsto{\mathsf{QSimon}} (-1)^b \left(\bigotimes_{i=0}^{m-1} |\psi_f\rangle\right) + |\delta\rangle \quad ,$$

*where $b = 1$ if $f$ is periodic and $0$ otherwise, and $|\delta\rangle$ is an error vector of amplitude: $\| |\delta\rangle \| \leqslant e2^{-r/2+1}$.*



**Quantum Circuit 1.10:** Quantum circuit for Simon's algorithm.

*Proof.* Circuit 1.10 gives a representation of QSimon, where Rank is a unitary operator that given $n$-bit vectors $y_1, \dots, y_{n+r}$, flips the phase if the rank of the family is less than $n-1$ (periodic case). Rank will perform Gaussian elimination, and may require additional ancilla qubits that are not represented in Circuit 1.10.

- If $f$ is periodic, then the rank of $y_1, \dots, y_{n+r}$ is always less than $n-1$, and we have $\| \, |\delta\rangle \, \| = 0$.
- If $f$ is simply random, then the families $y_1, \dots, y_{n+r}$ of rank less than $n$ incur errors on the output bit $b$ *and* on the product of states $|\psi_f\rangle$ that we wish to leave unchanged.

Before the final Hadamard transform, the error vector $|\delta'\rangle$ contains twice the bad families (with amplitude $-1$ on the right $|b\rangle$, and $+1$ on the wrong $|b\rangle$). Thus its squared norm is equal to 4 times the probability of measuring such a family, and the analysis of Theorem 1.3 and Lemma 1.5 applies. After the Hadamard transform, the norm of $|\delta\rangle$ satisfies: $\||\delta\rangle\| \leqslant \||\delta'\rangle\| \leqslant e2^{-r/2+1}$.

□

# Chapter 2
# Classical and Quantum Search

In this chapter, we focus on classical and quantum algorithms for unstructured search problems, including Grover's algorithm, its generalization to Amplitude Amplification, and its exact variant. We give some well-known technical results covering our use of quantum search in the following chapters. We cover the *sampling* framework, a bridge that we will use to walk from classical to quantum search. Note that this framework is equivalent to the *filter* framework that we introduced in [BNS19b]. Finally, we describe the collision, multicollision and preimage search problems for random functions and their previous best classical and quantum algorithms.

## Contents

## 2.1 Unordered Search

In this section, we will present Grover's algorithm [Gro96] and its generalization to *Amplitude Amplification* [Bra+02]. In the next chapters, we will refer to either of them by the term *quantum search*, but most often we will use the latter. These algorithms solve variants of the *unordered search* problem that we define as follows.

**Problem 2.1** (Unordered search)**.** *Let $X$ be a set. Let $G \subseteq X$ be a subset of $X$ ("good elements") and $f : X \to \{0,1\}$ a function ("test") such that $f(x) = 1 \iff x \in G$. Find $x \in G$.*

### 2.1.1 Classical Sampling

If no assumption is made on the function $f$, then any classical randomized algorithm solving Problem 2.1 requires $\Omega\left(|X|/|G|\right)$ queries to $f$, which is the average number of queries with random inputs $x$ before one of them is "good".

In order to emphasize the similarities between classical and quantum unordered search procedures, we will think of classical search as a *sampling* procedure that returns a random $x \in G$. In [BNS19b], the term *filter* is used, where a filter is thought of as a lazy sampling procedure. The presentation here will be equivalent.

**Definition 2.1** (Classical sampling)**.** Let $X$ be a set. A classical *sampling* procedure for $X$ is a randomized algorithm that requires no input and returns a random $x \in X$ that is uniformly distributed.

Classical unordered search can be seen as a function that turns a sampling $\mathsf{Sample}_X$ for $X$ into a sampling $\mathsf{Sample}_G$ for $G \subseteq X$, using the information given by the test $f$.

**Algorithm 2.1** Implementation of $\mathsf{Sample}_G$

```
repeat
    $x \leftarrow \mathsf{Sample}_X()$
until $f(x) = 1$
return $x$
```

In the following, we let $\mathsf{T_c}\left(\mathcal{A}\right)$ denote the classical average time complexity of a randomized algorithm $\mathcal{A}$, and by abuse of notation, we use also $f$ to denote an algorithm that computes $f$. We have:

**Lemma 2.1.**

$$\mathsf{T_c}\left(\mathsf{Sample}_G\right) = \frac{|X|}{|G|}\left(\mathsf{T_c}\left(\mathsf{Sample}_X\right) + \mathsf{T_c}\left(f\right)\right) \quad . \tag{2.1}$$

*Proof.* A proof can be done by a direct computation. Each step of $\mathsf{Sample}_G$ invokes $\mathsf{Sample}_X$ and $f$ once. Let $t$ be the number of steps before we find $x \in G$:

$$\mathbb{E}(t) = \sum_{t=1}^{+\infty} t \left(1 - \frac{|G|}{|X|}\right)^{t-1} \frac{|G|}{|X|} \quad .$$

If we let $f(x) = \sum_{t=0}^{+\infty}(1-x)^t = \frac{1}{x}$, then $\mathbb{E}(t) = -\frac{|G|}{|X|} f'\left(\frac{|G|}{|X|}\right)$. We conclude:

$$\mathbb{E}(t) = \left(\frac{|X|}{|G|}\right)^2 \frac{|G|}{|X|} = \frac{|X|}{|G|} \quad . \qquad \square$$

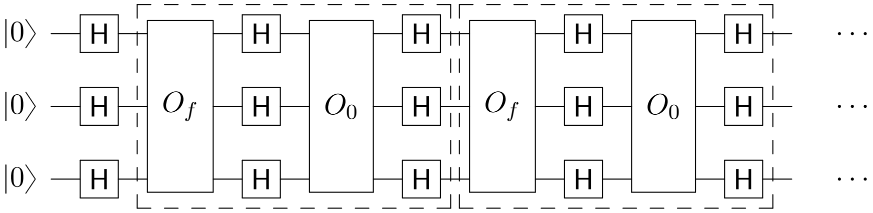
### 2.1.2 Grover's Algorithm

In the quantum setting, Grover's algorithm [Gro96] provides an elegant, efficient and optimal solution to Problem 2.1. In the following, we identify $X$ with a set of bit-strings $\{0,1\}^n$ or $n$-bit integers. Hence, the test is a function $f : \{0,1\}^n \to \{0,1\}$.

**Algorithm description.** Let $|X\rangle = \sum_{x\in X} |x\rangle$, $|G\rangle = \sum_{x\in G} |x\rangle$ and $|B\rangle = \sum_{x\in B} |x\rangle$ be the uniform superpositions over $X$, $G$ ("good" elements) and $B$ ("bad" elements). Grover's algorithm (Circuit 2.1) applies a first Hadamard transform $\mathsf{H}^{\otimes n}$, after which the state is $|X\rangle$. Then it iterates the operator:

$$\mathcal{G} = \mathsf{H}^{\otimes n} O_0 \mathsf{H}^{\otimes n} O_f \ \ ,$$

where $O_0$ is an "inversion around 0" which flips the phase of all basis vectors except $|0\rangle$ and $O_f$ is the phase oracle for $f$.



**Quantum Circuit 2.1:** Quantum circuit of Grover's algorithm.

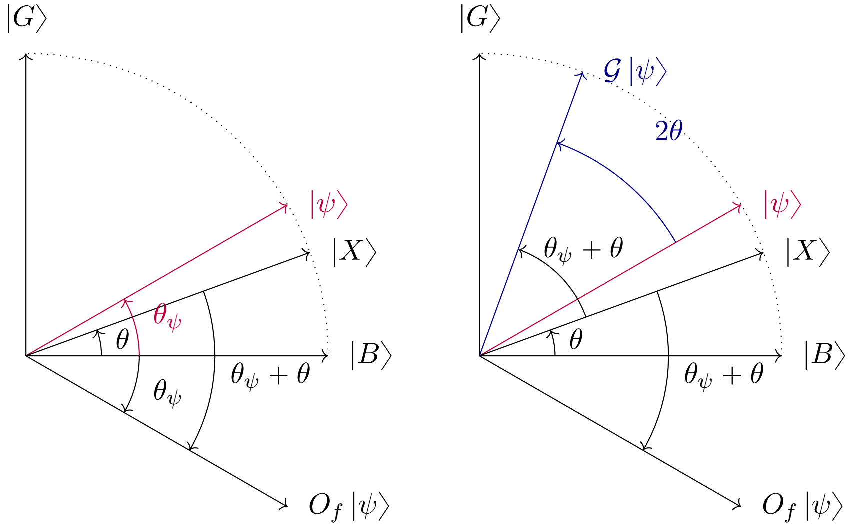


**Figure 2.1:** Rotation realized by $\mathcal{G}$ in the plane Span $(|B\rangle, |G\rangle)$. The initial state is $|\psi\rangle$.

**Action of $\mathcal{G}$.** Grover's algorithm is usually presented with a geometrical point of view, as in Figure 2.1. The initial state can be written as:

$$|X\rangle = \sin\theta\,|G\rangle + \cos\theta\,|B\rangle \quad \text{for} \quad \theta = \arcsin\left(\sqrt{\frac{|G|}{|X|}}\right) \enspace .$$

After each iterate $\mathcal{G}$, the current state remains a linear combination of $|G\rangle$ and $|B\rangle$, and the angle increases by $2\theta$.

**Lemma 2.2.** *Grover's iterate $\mathcal{G}$ is a rotation of angle $2\theta$ in the plane spanned by $|G\rangle$ and $|B\rangle$.*

*Proof.* We show that $\mathcal{G}$ is a composition of two reflections in $\text{Span}\,(|G\rangle, |B\rangle)$. First, $O_f$ is a reflection around $|B\rangle$, since it flips the phase of all good vectors:

$$O_f = 2\,|B\rangle\langle B| - \mathsf{I} \enspace .$$

Next, the operator $\mathsf{H}^{\otimes n} O_0 \mathsf{H}^{\otimes n}$, which can be implemented using $\mathcal{O}(n)$ quantum gates, realizes an *inversion around the average.* On input $\sum_{x\in X} \alpha_x\,|x\rangle$, let $A = \frac{1}{2^n}\sum \alpha_x$ be the average amplitude, then:

$$\mathsf{H}^{\otimes n} O_0 \mathsf{H}^{\otimes n}\left(\sum_x \alpha_x\,|x\rangle\right) = \sum_x (2A - \alpha_x)\,|x\rangle \enspace .$$

Hence, it realizes a reflection around $|X\rangle$:

$$\mathsf{H}^{\otimes n} O_0 \mathsf{H}^{\otimes n} = \mathsf{H}^{\otimes n}\left(2\,|0^n\rangle\langle 0^n| - \mathsf{I}\right)\mathsf{H}^{\otimes n} = 2\,|X\rangle\langle X| - \mathsf{I} \enspace .$$

The composition of these two reflections is a rotation of angle $2\theta$, as drawn in Figure 2.1. □

**Number of iterates.** Let $|\psi_t\rangle$ be the state after $t$ iterates. As we have seen:

$$|\psi_0\rangle = \sin\theta\,|G\rangle + \cos\theta\,|B\rangle \enspace ,$$

with $\theta = \arcsin\left(\sqrt{\frac{|G|}{|X|}}\right)$, and each iterate increases this angle by $2\theta$. What remains is to compute the number of iterates after which the current state will be closest to $|G\rangle$. We have:

$$|\psi_t\rangle = \sin((2t+1)\theta)\,|G\rangle + \cos((2t+1)\theta)\,|B\rangle \enspace ,$$

which means that we must choose $t$ such that $\cos^2((2t+1)\theta)$ is at its smallest. If we iterate too much, we start moving away from $|G\rangle$, what is metaphorically called the *soufflé* property of Grover's algorithm (we say that the state is *overcooked*).

**Theorem 2.1.** *By choosing $t = \left\lfloor \frac{\pi}{4\theta} \right\rfloor$, Grover's algorithm produces:*

$$\sqrt{(1-\alpha^2)}\,|G\rangle + \alpha\,|B\rangle \enspace ,$$

*where $|\alpha| \leqslant \sqrt{\frac{|G|}{|X|}}$.*

*Proof.* With this choice of $t$, we have: $\frac{\pi}{4\theta} - 1 < t \leqslant \frac{\pi}{4\theta}$ hence $\frac{\pi}{2} - \theta < (2t+1)\theta \leqslant \frac{\pi}{2} + \theta$, hence $|\cos((2t+1)\theta)| \leqslant \cos\left(\frac{\pi}{2} - \theta\right) = \sin\theta = \sqrt{\frac{|G|}{|X|}}$. □

If we immediately measure the output state, then with probability at least $1 - \frac{|G|}{|X|}$, we obtain an element of $G$. Conditioned on being in $G$, this element is sampled uniformly at random. In general, we have $\frac{|G|}{|X|} \ll 1$ and the approximation $\theta \simeq \sqrt{\frac{|G|}{|X|}}$ is valid. Hence, Grover's algorithm requires $\mathcal{O}\left(\sqrt{\frac{|G|}{|X|}}\right)$ iterates for a constant (and often negligible) probability of error, offering a well-known square-root speedup on classical unordered search.

**Optimality.** A first optimality result was given in [Ben+97], before Grover's algorithm was discovered. They show that when $|G| = 1$, a quantum algorithm for Problem 2.1 requires $\Omega(\sqrt{|X|})$ queries to the oracle $O_f$. This bound was improved in [Boy+98]; later on, Zalka showed that Grover's algorithm was optimal [Zal99].

### 2.1.3 Discussion on the Algorithm

Before we give the technical extensions of the algorithm that appeared later in the literature, let us discuss briefly its applicability.

**Search space.** In our presentation, we have assumed that the search space $X$ could be identified with bit-strings $\{0,1\}^n$. That is, there exists an injective function, computable in polynomial time that maps an index $i \in \{0,1\}^n$ to $x_i \in X$. This will always be the case in the following chapters, and it has multiple practical consequences: we can consider $X$ as an ordered set (as the indices have a canonical ordering) and we can perform a search in *subsets* of $X$ (by giving arbitrary values to some of the index bits).

**Grover's algorithm as a database search.** The original paper by Grover was entitled *A fast quantum mechanical algorithm for database search*. However, if $X$ represents a classical database, mapping indices $i$ to $x_i$ is a memory access. Without QRACM, performing this operation in superposition over $i$ cannot be done in $\mathsf{poly}(n)$.

However, it applies without difficulty to many problems with a trivial search space, e.g., SAT solving or exhaustive key search in symmetric ciphers, making it one of the most well-known applications of quantum computers in symmetric cryptography.

**Parallelization.** The classical exhaustive search procedure (Algorithm 2.1) that solves Problem 2.1 can be perfectly distributed. If we run $S$ instances of the algorithm in parallel, then $G$ can be sampled in expected wall-clock time $\frac{|X|}{|G|S}$ and after a total number of $\frac{|X|}{|G|}$ iterations.

In comparison, Grover's algorithm parallelizes badly. By running $S$ independent instances and stopping after $\mathcal{O}\left(\sqrt{\frac{|X|}{|G|S}}\right)$ iterates, the measurement results will contain expectedly an element in $G$. The wall-clock time is reduced only by a factor $\sqrt{S}$ and the total number of iterates increases by the same factor. This is proven optimal in [Zal99].

**Limiting the number of queries.** Grover's algorithm performs badly if we enforce a limit on the number of queries to $O_f$, or equivalently, on the time. While stopping a *classical* exhaustive search in $X$ after $t$ iterations yields a solution with success probability $\mathcal{O}\left(\frac{t|G|}{|X|}\right)$, stopping Grover's algorithm after $t$ iterations yields a state that, if measured, gives a solution with probability $\mathcal{O}\left(\frac{t^2|G|}{|X|}\right)$ (this follows from an estimation of $\sin^2((2t+1)\theta)$). This is actually optimal (the proof follows from [Zal99]).

**Lemma 2.3.** *If $f$ is a random oracle, then any quantum algorithm that makes $q$ queries to $O_f$ can only output $x \in G$ with probability $\mathcal{O}\left(\frac{q^2|G|}{|X|}\right)$.*

### 2.1.4 Amplitude Amplification

Amplitude Amplification, proposed by Brassard, Hoyer, Mosca, and Tapp, is a powerful generalization of Grover's algorithm. It allows to speed up the search for a "good" output of *any* probabilistic algorithm, including all kinds of exhaustive search problems.

**Theorem 2.2** ([Bra+02], Theorem 2)**.** *Let $\mathcal{A}$ be a quantum algorithm that uses no measurements, let $f : X \to \{0,1\}$ be a Boolean function that tests if an output of $\mathcal{A}$ is "good". Let $a$ be the success probability of $\mathcal{A}$. Let $\mathcal{Q}$ be the operator:*

$$\mathcal{Q} = \mathcal{A} O_0 \mathcal{A}^\dagger O_f \ ,$$

*let $\theta_a = \arcsin\sqrt{a}$ and $0 < \theta_a \leqslant \frac{\pi}{2}$. Let $t = \left\lfloor \frac{\pi}{4\theta_a} \right\rfloor$. Then by measuring $\mathcal{Q}^t \mathcal{A} |0\rangle$, we obtain a good result with success probability greater than* $\max(1-a, a)$.

In Grover's algorithm, the Hadamard transform $\mathsf{H}^{\otimes n}$ plays the role of $\mathcal{A}$. Indeed, immediately measuring $\mathsf{H}^{\otimes n} |0^n\rangle$ produces a good result with probability $a = \frac{|G|}{|X|}$. The idea behind Amplitude Amplification is that we can now run a quantum search *on the outputs of another quantum algorithm.* But it goes even further. If the exact probability of success is known, then the search can be made exact: it will have a pre-determined number of iterations and a success probability 1.

**Theorem 2.3** ([Bra+02], Theorem 4)**.** *Let $a$ be the success probability of $\mathcal{A}$. Then there is a quantum algorithm* $\mathsf{QExact}(\mathcal{A}, f, a)$ *that finds a correct solution with probability 1 using $\Theta\left(\frac{1}{\sqrt{a}}\right)$ evaluations of $\mathcal{A}$.*

If we note $\mathsf{T_q}(\mathcal{A})$ the quantum time complexity of $\mathcal{A}$, then we have more precisely (for small $a$):

$$\mathsf{T_q}(\mathsf{QExact}(\mathcal{A}, f, a)) = \overbrace{\mathsf{T_q}(\mathcal{A})}^{\text{Initial state}} + \underbrace{\left(\left\lfloor \frac{\pi}{4\theta_a} \right\rfloor + 1\right)}_{\substack{\text{Includes the last}\\ \text{modified}\\ \text{iteration}}} \left( \underbrace{2\mathsf{T_q}(\mathcal{A})}_{\substack{\text{Run } \mathcal{A} \text{ and}\\ \mathcal{A}^\dagger}} + \mathsf{T_q}(O_f) + \mathsf{T_q}(O_0) \right) ,$$

where this complexity corresponds to the algorithm of Theorem 2.2 with a small modification. The idea is to go back to the iterates in Grover's algorithm. The probability of error comes from the fact that the iterates move the current state by a fixed angle $2\theta$ in the plane $\mathrm{Span}(|B\rangle, |G\rangle)$, thus reaching only a vector that forms an angle less than $\theta$ with $|G\rangle$. However, it is possible to modify the angle in the last rotation in order to meet perfectly the target $|G\rangle$. This requires to implement rotation gates of arbitrary precision which is, in turn, doable with a limited universal gate set thanks to the Solovay-Kitaev theorem (Theorem 1.1).

**With unknown success probability.** If the success probability is unknown, we can rely on the following theorem.

**Theorem 2.4** ([Bra+02], Theorem 3)**.** *Let $\mathcal{A}$ be a quantum algorithm without measurements of unknown success probability $a > 0$, $f$ a function that tests the outputs of $\mathcal{A}$. Then there is an algorithm* $\mathsf{QSearch}(\mathcal{A}, f)$ *that finds a correct solution using an expected $\Theta\left(\frac{1}{\sqrt{a}}\right)$ evaluations of $\mathcal{A}$ and $f$.*

The idea of Algorithm 2.2 is to run the Amplitude Amplification procedure with an exponentially increasing number of iterations, until the solution is found. At the beginning, the end state is only negligibly rotated in $\mathrm{Span}(|B\rangle, |G\rangle)$ and it is close to the uniform state $|X\rangle$ (which is close to $|B\rangle$ if there are only few solutions). However, once the number of iterations reaches and exceeds the optimal value $\left\lfloor \frac{\pi}{4 \arcsin(\sqrt{a})} \right\rfloor$, the end state roughly becomes a random unit vector in $\mathrm{Span}(|B\rangle, |G\rangle)$, hence $\sin\phi |G\rangle + \cos\phi |B\rangle$ for a random angle $\phi$. Hence, measuring projects in $G$ with probability $\frac{1}{2}$.

If $a = 0$, the algorithm runs indefinitely. Usually we know that either $a = 0$, either is it lower bounded by $b$. In that case, we use $\mathsf{QSearch}(\mathcal{A}, f)$ and stop it when $M\sqrt{b}$ exceeds some fixed constant value. The algorithm has one-sided errors: it can return $\perp$ even with a solution exists, in the case where all Steps 6 with $M > \left\lfloor \frac{\pi}{4 \arcsin(\sqrt{a})} \right\rfloor$ fail. Since each of them has a constant probability of failure, the constant in the $\mathcal{O}$ will be proportional to $\epsilon$ in order to have a failure probability less than $\frac{1}{2^\epsilon}$.

**Algorithm 2.2** Quantum search $\mathsf{QSearch}(\mathcal{A}, f)$

```
Input: $\mathcal{A}, f$
Set $\ell = 0$ and $c$ a constant such that $1 < c < 2$
repeat
    $\ell \leftarrow \ell + 1$, $M \leftarrow \lceil c^\ell \rceil$
    Compute $\mathcal{A}|0\rangle$, measure and obtain $z$. If $f(z) = 1$, return $z$
    Pick $j \overset{\$}{\leftarrow} [1; M]$
    Compute $\mathcal{Q}^j \mathcal{A}|0\rangle$ where $\mathcal{Q}$ is the Amplitude Amplification iterate, measure and obtain $z$.
until $f(z) = 1$
return $z$ and stop
```

**Corollary 2.1.** *If $a = 0$ or $a \geqslant b$, there is an algorithm $\mathsf{QFind}(\mathcal{A}, f, b)$ that returns a correct solution or $\perp$ using an expected $\mathcal{O}\left(\frac{1}{\sqrt{a}}\right)$ number of iterations if $a > 0$ or $\mathcal{O}\left(\frac{1}{\sqrt{b}}\right)$ otherwise.*

**With a good interval.** We can wish for more efficiency if an interval $[a(1-\epsilon), a(1+\epsilon)]$ on the success probability of $\mathcal{A}$ is given. In that case, we do not need to use the full power of Theorem 2.4. A single run of quantum search by assuming a success probability $a$ ensures a high success probability, as long as the relative error $\epsilon$ is small enough.

**Theorem 2.5** (Lemma 5 in [BNS19b])**.** *Assume that $\mathcal{A}$ has a success probability $a' \in [a(1-\epsilon), a(1+\epsilon)]$ for $\epsilon \leqslant \frac{1}{2}$. Then after running an Exact Amplitude Amplification $\mathsf{QExact}(\mathcal{A}, a)$ that assumes a success probability exactly equal to $a$, we measure a good state with probability greater than $1 - \epsilon^2$.*

*Proof.* An exact amplitude amplification with parameter $a$ results in a rotation of angle $\theta = \frac{\pi}{2}\sqrt{1 \pm \epsilon}$ instead of $\frac{\pi}{2}$ in the plane spanned by good and bad results for $\mathcal{A}$. If we measure immediately, the probability of failure is given by $\cos^2\theta$. We bound $|\cos\theta|$ by:

$$|\cos\theta| = \left|\sin\left(\theta - \frac{\pi}{2}\right)\right| \leqslant \left|\theta - \frac{\pi}{2}\right| \leqslant \frac{\pi}{2}(\sqrt{1+\epsilon} - 1) \ .$$

Then, we use that $\sqrt{1+\epsilon} \leqslant 1 + \frac{\epsilon}{2}$ for $\epsilon \leqslant \frac{1}{2}$, and $\frac{\pi}{4} \leqslant 1$. The result follows. □

### 2.1.5 Finding Many Solutions

Classically, since Algorithm 2.1 samples elements at random from $G$, finding $M \ll \sqrt{|G|}$ solutions costs time $M \cdot \frac{|X|}{|G|}$, as all outputs are distinct with high probability. However, learning all of $G$ may be slightly more difficult, as this is an instance of the *coupon collector problem.*

**Problem 2.2** (Coupon collector)**.** *Given uniformly random samples from a set $G$, learn $G$.*

It is well-known that $\Theta(|G| \log |G|)$ queries are necessary and sufficient to solve this problem with overwhelming probability ([MR10, Theorem 3.8]). In our case, we assumed that the elements of $X$ are uniquely indexed by bit-strings, which means that $X$ is ordered. Thus, we can adapt the "sampling" framework from Section 2.1.1 to iterate on the elements of $X$ without duplicates. That is, Algorithm 2.1 will maintain an internal iterator, ensuring that it never looks twice at the same $x \in X$. As a consequence, the same property follows for the elements sampled from $G$, which have no duplicates.

Thus, if we wish to obtain a particular number of solutions, we should just set up the size of the search space $X$ so that $G$ is of sufficient cardinality.

**With quantum search.** Quantumly, it is known that $\Theta(\sqrt{|X||G|})$ queries are necessary (as shown in [KSW04]). If we consider that exact copies of the state $|G\rangle$ are given, and that we have to learn $G$ from it, the problem is similar [Aru+20].

**Lemma 2.4.** *Given $|G|$, there is a quantum algorithm* QFindAll$(X, G)$ *that finds all elements of $G \subseteq X$, using $\mathcal{O}\left(\sqrt{|X||G|}\right)$ queries to $O_f$ and $\mathcal{O}\left(\sqrt{|X||G|}\right)$ time* in the QRACM model.

*Proof.* The correctness of Algorithm 2.3 is trivial. We simply prove its time complexity:

$$\sum_{i=0}^{|G|-1} \mathcal{O}\left(\sqrt{\frac{|X|}{|G|-i}}\right) = \mathcal{O}\left(\sqrt{|X|}\right) \sum_{i=1}^{|G|} \frac{1}{\sqrt{i}} = \mathcal{O}\left(\sqrt{|G||X|}\right) \quad . \qquad \square$$

**Algorithm 2.3** Algorithm QFindAll$(X, G)$

```
Input: $X$, oracle access to $O_f$
Returns: the complete set $G$
Initialize the result $R = \varnothing$
while $|R| < |G|$ do
    Let $f'$ be the oracle that accepts only $x \in G \backslash R$        ▷ Use QRACM
    Run exact quantum search QExact(Sample$_X$, $f'$, $|G| - |R|$) on $X$
    Measure the result and obtain $z$
    $R \leftarrow R \cup \{z\}$
return $R$
```

Lemma 2.4 allows us to remove the logarithmic factor of the coupon collector, but it relies heavily on the knowledge of $|G|$ and on the QRACM. Without QRACM, we will instead re-run $\mathcal{O}(|G| \log |G|)$ independent searches and dismiss duplicate results.

### 2.1.6 Approximate Test Functions

In the following chapters, we will encounter some cases where the test $f$ is not exact, encompassed by the following definition.

**Definition 2.2** (Approximate test)**.** An approximate test oracle $O_f$ with error *at most* $\epsilon$ is a unitary operator that maps:

$$|x\rangle\,|b\rangle \xmapsto{O_f} |x\rangle\,|b \oplus f(x)\rangle + |x\rangle\,|\delta_x\rangle \ \ ,$$

where $\|\,|\delta_x\rangle\,\| \leqslant \epsilon$ for all $x$, and some $\epsilon > 0$.

Note that this definition can be used as well with a phase oracle. We can also say that upon measurement, the probability of error is less than $\epsilon$. The definition for a phase oracle is similar. In the following, we also consider that the approximate test uses an ancillary state $|\psi\rangle$.

**Theorem 2.6** (Amplitude amplification with approximate test)**.** *Let $\mathcal{A}, a, \theta_a$, be as in Theorem 2.2 a quantum algorithm, its success probability and corresponding angle. Let $O'_f$ be an approximate test of error $\epsilon$. On input $|\psi\rangle \otimes (\mathcal{A}\,|0\rangle)$, run $t = \left\lfloor \frac{\pi}{4\theta_a} \right\rfloor$ iterations of:*

$$\mathcal{Q}' = (\mathsf{I} \otimes \mathcal{A})(\mathsf{I} \otimes O_0)(\mathsf{I} \otimes \mathcal{A}^\dagger)O'_f \ \ ,$$

*where $\mathsf{I}$ is the identity operator applied to the ancillary state $|\psi\rangle$. Then measuring the output yields a good result with probability greater than $(1-a)\,(1-t\epsilon)^2$.*

*Proof.* The proof uses a "hybrid argument" as in [Ben+97] or [Amb07, Lemma 5]. The idea is that if the error vector is sufficiently small, errors will go unnoticed during the search. Let $|\psi'_k\rangle$ be the state after $k$ iterations of $\mathcal{Q}'$, and let $|\psi_k\rangle$ be the state after $k$ iterations of the "ideal" operator $\mathcal{Q}$ where the test $O_f$ is exact.

Let $U = (\mathsf{I} \otimes \mathcal{A})O_0(\mathsf{I} \otimes \mathcal{A}^\dagger)$. We deduce from the triangle inequality:

$$\begin{aligned} \left\| |\psi'_{k+1}\rangle - |\psi_{k+1}\rangle \right\| &= \left\| UO'_f\,|\psi'_k\rangle - UO_f\,|\psi_k\rangle \right\| \\ &= \left\| UO_f\,|\psi'_k\rangle + U\delta - UO_f\,|\psi_k\rangle \right\| \\ &\leqslant \left\| UO_f(|\psi'_k\rangle - |\psi_k\rangle) \right\| + \|U\delta\| \\ &\leqslant \left\| |\psi'_k\rangle - |\psi_k\rangle \right\| + \epsilon \ \ . \end{aligned}$$

Hence we have $|\psi'_t\rangle = |\psi_t\rangle + |\psi_{\text{err}}\rangle$ where $\||\psi_{\text{err}}\rangle\| \leqslant t\epsilon$. By the Cauchy-Schwarz inequality, we have:

$$|\langle\psi_t|\psi_{\text{err}}\rangle| \leqslant \||\psi_t\rangle\|\,\||\psi_{\text{err}}\rangle\| \leqslant t\epsilon \ \ .$$

Measuring $|\psi'_t\rangle$, we project on $|\psi_t\rangle$ with a probability greater than:

$$(1 - |\langle\psi_t|\psi_{\text{err}}\rangle|^2) \geqslant (1-t\epsilon)^2 \ \ ,$$

which we combine with Theorem 2.2 to obtain the result. □

## 2.2 Nested Search

Many problems admit a significant quantum speedup when using a single Grover search instead of a classical exhaustive search, e.g., in the search for a good permutation in Information Set Decoding [Ber10]. However, the variety of search problems that can be solved efficiently with a *combination* of searches is even greater. In this section, we define a class of *sampling* algorithms which both admit a classical and a quantum variant, with a square-root speedup between them. They are especially useful to analyze complex procedures based on exhaustive search, by using classical procedures as prototypes, as we will do in Chapter 9.

### 2.2.1 Quantum Search as a Sampling Procedure

Recall that we let $\mathsf{T_q}\left(\mathcal{A}\right)$ denote the quantum time complexity of a quantum algorithm $\mathcal{A}$.

In Section 2.1.1, we defined a *classical sampling* $\mathsf{Sample}_X$ as a function producing a uniformly random $x \in X$. We showed that classical unordered search turned a sampling $\mathsf{Sample}_X$ into a sampling $\mathsf{Sample}_G$ for the good subspace. We can now proceed by analogy in the quantum setting.

**Definition 2.3** (Quantum sampling)**.** Let $X$ be a set. A *quantum sampling* procedure for $X$ is a quantum algorithm that requires no specific input (e.g., $|0\rangle$) and returns the uniform superposition $\sum_{x \in X} |x\rangle$.

In Grover's algorithm, as $X = \{0,1\}^n$, this procedure is simply the Hadamard transform $\mathsf{H}^{\otimes n}$.

Then, *quantum search* can be seen as a mapping that turns a quantum sampling $\mathsf{QSample}_X$ for $X$ into a sampling $\mathsf{QSample}_G$ for $G = X|_f = \{x \in X, f(x) = 1\}$, using evaluations of the quantum oracle $O_f$. We use exact Amplitude Amplification (Theorem 2.3).

**Lemma 2.5.** *Let* $\theta = \arcsin\sqrt{\frac{|G|}{|X|}}$, *then:*

$$\mathsf{T_q}\left(\mathsf{QSample}_G\right) = \left(\left\lfloor \frac{\pi}{4\theta} \right\rfloor + 1\right)\left(\mathsf{T_q}\left(\mathsf{QSample}_X\right) + \mathsf{T_q}\left(O_f\right)\right) \quad . \tag{2.2}$$

If $|G|$ and $|X|$ are big and if we are interested only in asymptotic complexities, the formulas:

$$\mathsf{T_c}\left(\mathsf{Sample}_G\right) = \frac{|X|}{|G|}\left(\mathsf{T_c}\left(\mathsf{Sample}_X\right) + \mathsf{T_c}\left(f\right)\right)$$

and

$$\mathsf{T_q}\left(\mathsf{QSample}_G\right) = \sqrt{\frac{|X|}{|G|}}\left(\mathsf{T_q}\left(\mathsf{QSample}_X\right) + \mathsf{T_q}\left(O_f\right)\right)$$

are similar up to the square-root factor in the number of iterations.

### 2.2.2 Nesting

In the following, if $X$ if the search space and $G$ the "good" subspace, we let $\mathsf{qiter}\left(X, G\right)$ denote the number of iterations of quantum search, $\mathsf{iter}\left(X, G\right)$ for classical search, and also $G = X|_f$ the subset of elements that satisfy the condition $f$.

**When the test is a product.** Suppose that $f = f_1 \wedge f_2$. For example, $f_1$ could be evaluating if a key $k$ encrypts a given message $m_1$ to a given ciphertext $c_1$, $f_2$ if $m_2$ encrypts to $c_2$, and their combination would ensure that $k$ is the good key. A classical sampling $\mathsf{Sample}_{X|_{f_1 \wedge f_2}}$ can be implemented in different ways.

**Algorithm 2.4** Direct implementation of $\mathsf{Sample}_{X|_{f_1 \wedge f_2}}$

```
repeat
    $x \leftarrow \mathsf{Sample}_X$
until $f_1 \wedge f_2(x)$
return $x$
```

**Algorithm 2.5** Nested implementation of $\mathsf{Sample}_{X|_{f_1 \wedge f_2}}$

```
repeat
    $x \leftarrow \mathsf{Sample}_{X|_{f_1}}$
until $f_2(x)$
return $x$
```

Depending on the cardinalities of $X$, $X|_{f_1}$, $X|_{f_2}$, and on the costs of $f_1$ and $f_2$, a combination of samples might perform better than a mere exhaustive search in $X$. Algorithm 2.4 has a complexity:

$$\mathsf{T_c}\left(\mathsf{Sample}_{X|_{f_1 \wedge f_2}}\right) = \mathsf{iter}\left(X, X|_{f_1 \wedge f_2}\right)\left(\mathsf{Sample}_X + \mathsf{T_c}\left(f_1\right) + \mathsf{T_c}\left(f_2\right)\right)$$

and Algorithm 2.5 gives:

$$\begin{aligned}\mathsf{T_c}\left(\mathsf{Sample}_{X|_{f_1 \wedge f_2}}\right) &= \mathsf{iter}\left(X|_{f_1}, X|_{f_1 \wedge f_2}\right)\left(\mathsf{Sample}_{X|_{f_1}} + \mathsf{T_c}\left(f_2\right)\right) \\ &= \mathsf{iter}\left(X|_{f_1}, X|_{f_1 \wedge f_2}\right)\left(\mathsf{iter}\left(X, X|_{f_1}\right)\left(\mathsf{T_c}\left(\mathsf{Sample}_X\right) + \mathsf{T_c}\left(f_1\right)\right) + \mathsf{T_c}\left(f_2\right)\right)\end{aligned}$$

where we can remark that $\mathsf{iter}\left(X|_{f_1}, X|_{f_1 \wedge f_2}\right)\mathsf{iter}\left(X, X|_{f_1}\right) = \mathsf{iter}\left(X, X|_{f_1 \wedge f_2}\right)$. In other words, the *total* number of iterations remains the same, but we only test for $f_2$ if $f_1$ is already satisfied (or the converse).

**When the search space is a product.** Suppose that $X = X_1 \times X_2$, with a test $f : X_1 \times X_2 \rightarrow \{0, 1\}$. Again, there are different ways of implementing this search. We see that Algorithm 2.6 repeats samples of $X_1$ that could be simply reused. For example, if there is only a single solution in $X_1 \times X_2$, then Algorithm 2.7 corresponds to searching in $X_1$ for $x_1$ such that *there exists* $x_2 \in X_2$ *with* $f(x_1, x_2)$. Thus, while before the sampling contained another search, now it is the test, and we go from:

$$\mathsf{T_c}\left(\mathsf{Sample}_{X_1 \times X_2|_f}\right) = |X_1 \times X_2|\left(\mathsf{T_c}\left(\mathsf{Sample}_{X_1}\right) + \mathsf{T_c}\left(\mathsf{Sample}_{X_2}\right) + \mathsf{T_c}\left(f\right)\right)$$

to:

$$\mathsf{T_c}\left(\mathsf{Sample}_{X_1 \times X_2|_f}\right) = |X_1|\Big(\mathsf{T_c}\left(\mathsf{Sample}_{X_1}\right) + |X_2|\left(\mathsf{T_c}\left(\mathsf{Sample}_{X_2}\right) + \mathsf{T_c}\left(f\right)\right)\Big) \ .$$

**Algorithm 2.6** Direct implementation of $\mathsf{Sample}_{X_1 \times X_2|_f}$

```
repeat
    $x_1 \leftarrow \mathsf{Sample}_{X_1}$
    $x_2 \leftarrow \mathsf{Sample}_{X_2}$
until $f(x_1, x_2)$
return $x_1, x_2$
```

**Algorithm 2.7** Nested implementation of $\mathsf{Sample}_{X_1 \times X_2|_f}$

```
repeat
    $x_1 \leftarrow \mathsf{Sample}_{X_1}$
    repeat
        $x_2 \leftarrow \mathsf{Sample}_{X_2}$
    until $f(x_1, x_2)$ or $X_2$ is exhausted
until $f(x_1, x_2)$
return $(x_1, x_2)$
```

### 2.2.3 Quantum-Classical Correspondence

The above discussion justifies to define **Sample** programs as follows, encompassing early-abort strategies and nested searches.

**Definition 2.4** (Sample programs)**.** A **Sample** program samples an element of some space $X$. It is defined recursively by: • a randomized algorithm $\mathsf{Sample}_X$ that simply samples uniformly at random from $X$, • another sample (or a sequence of a fixed number of samples), and a test, both of which can be **Sample** programs.

If we write down a classical **Sample** program, we can use Lemma 2.5 recursively and replace all of its subprocedures by quantum samples, obtaining the following (informal) result.

**Theorem 2.7.** *Let $\mathcal{A}$ be a classical algorithm written as a combination of **Sample**s, with a constant amount of nesting. Let us write down the classical complexity of $\mathcal{A}$ as a function of the number of iterations of each **Sample**:*

$$\mathsf{T_c}\left(\mathcal{A}\right) = T(I_1, \dots, I_t) \ ,$$

*and assume that all non-sample operations of $\mathcal{A}$ are written in a reversible way (in order to avoid trade-offs for reversible computing). Then there exists a quantum algorithm $\mathcal{A}'$, that admits the same distribution of outputs as $\mathcal{A}$, and has a time complexity:*

$$\mathsf{T_q}\left(\mathcal{A}'\right) = \mathcal{O}\Big(T\left(\sqrt{I_1}, \dots, \sqrt{I_t}\right)\Big) \ .$$

*Proof.* The proof is a simple induction using Lemma 2.5 and the definition of a **Sample**. □

**Example 2.1** (Product test)**.** The classical **Sample** of Algorithm 2.5 translates into a combination of two quantum searches: $\mathsf{QSearch}(\mathsf{QSearch}(\mathsf{Sample}_X, f_1), f_2)$. It has the following complexity, up to a constant:

$$\mathsf{T_q}\left(\mathsf{QSample}_{X|_{f_1 \wedge f_2}}\right) = \sqrt{\mathsf{iter}\left(X|_{f_1}, X|_{f_1 \wedge f_2}\right)}\left(\sqrt{\mathsf{iter}\left(X, X|_{f_1}\right)}\left(\mathsf{T_q}\left(\mathsf{Sample}_X\right) + \mathsf{T_q}\left(f_1\right)\right) + \mathsf{T_q}\left(f_2\right)\right) \enspace .$$

Note that we cannot actually use Algorithm 2.2 here, and we must resort to an estimate of the success probability with Theorem 2.5.

**Example 2.2** (Product search space)**.** With a single solution, the classical **Sample** or Algorithm 2.7 has a quantum equivalent with the following complexity, up to a constant:

$$\mathsf{T_q}\left(\mathsf{QSample}_{X_1 \times X_2|_f}\right) = \sqrt{|X_1|}\left(\mathsf{T_c}\left(\mathsf{Sample}_{X_1}\right) + \sqrt{|X_2|}\left(\mathsf{T_c}\left(\mathsf{Sample}_{X_2}\right) + \mathsf{T_c}\left(f\right)\right)\right) \enspace .$$

Theorem 2.7 allows us to design quantum sampling procedures in two stages: **1.** we specify a classical **Sample** program, determine its complexity and use the theorem to estimate the complexity of a corresponding quantum sampling. **2.** we use more precise results on quantum search to determine the success probability of the quantum sampling and its complexity. This separates the classical and quantum technicalities, and allows to delay the latter as much as possible. We will use this design principle in Chapter 9.

### 2.2.4 Notations

In the following chapters, we will write many algorithms as **Sample** programs, in order to use the classical-quantum correspondence of Theorem 2.7. As we saw above, these programs are easily translated as combinations of classical **repeat-until** loops, *or* quantum searches, with a quadratic correspondence in the number of search iterates. Inside a **Sample** block, we define a variable $x$ sampled from some search space $X$, and we evaluate some intricate Boolean condition on $X$. This requires further computations, and possibly invoking other **Sample**s. By abuse of notation, we will add keywords such as **if** and **abort** to emphasize where this Boolean condition is evaluated lazily. After the **Sample** block, $x$ contains either a value that satisfies the condition, or $\perp$, if there is no such value. An example is given in Algorithm 2.8.

## 2.3 Collisions

We now review generic algorithms for finding *collisions*, which give upper bounds in the study of hash functions and modes of operation, but also building blocks for cryptanalytic algorithms. Throughout this section, $h : \{0,1\}^n \to \{0,1\}^n$ is a random function to which we have quantum oracle access.

**Algorithm 2.8** Example of **Sample** program. $f_1, f_2, f_3, f_4$ are arbitrary functions, $X$ and $Z$ are sets which can be sampled efficiently.

```
Input: no input.
Output: $x \in X$ such that $f_2(x, f_1(x))$ and $\exists z, f_3(x,z) \wedge f_4(x, f_1(x), z)$, or $\perp$ if it doesn't exist.
Sample $x \in X$ such that
                                  ▷ Inside this block, the variable $x$ holds a fixed value
    $y \leftarrow f_1(x)$
    if not $f_2(x,y)$ then abort
    Sample $z \in Z$ such that
        if not $f_3(x,z)$ then abort
    EndSample
                                  ▷ Here $z$ holds a value such that $f_3(x,z)$, or $\perp$ if there is none
    if not $f_4(x,y,z)$ then abort
EndSample
return $x$
```

**Problem 2.3** (Collision search)**.** *Given access to $h : \{0,1\}^n \to \{0,1\}^n$, find a* collision *of $h$: $x \neq y$ such that $h(x) = h(y)$.*

If the function $h$ picked at random does not have collisions, then all algorithms for Problem 2.3 will fail. More generally, the problems that we define here may not have any solutions, but the probability of such an event goes into the probability of failure of the corresponding algorithms.

A problem similar to collision search is *claw-finding*, where we want a collision between the outputs of two independent random functions $f, g$.

**Problem 2.4** (Claw-finding)**.** *Given access to $f, g : \{0,1\}^n \to \{0,1\}^n$, find a* claw *between $f$ and $g$: a pair $x, y$ such that $f(x) = g(y)$.*

These problems are equivalent. On the one hand, we can define the function $h : \{0,1\} \times \{0,1\}^n \to \{0,1\}^n$ by $h(0,x) = f(x)$ and $h(1,x) = g(x)$, then a collision pair of $h$ yields a claw of $f, g$ with probability $\frac{1}{2}$.

*Multicollisions* extend this to an $r$-tuple of colliding elements.

**Problem 2.5** (Multicollision search)**.** *Let $r \geqslant 2$ be an integer. Find an $r$-collision: a tuple $x_1, \ldots, x_r$ of distinct inputs such that $h(x_1) = \ldots = h(x_r)$.*

Finally, we also consider preimage search problems.

**Problem 2.6** (Preimage Search)**.** *Given access to $h$, and a target $y$, find $x \in \{0,1\}^n$ such that $h(x) = y$ (if it exists).*

**Problem 2.7** (Multi-target Preimage Search)**.** *Given access to $h$, and a list of targets $\mathcal{L} = \{y_0, \ldots, y_{L-1}\}$, find $x \in \{0,1\}^n$ such that $\exists i \in [0;\ L-1]$, $h(x) = y_i$.*

Note that the case where $h$ admits a single collision, or where such a function has to be distinguished from injective, is the well-studied *element distinctness* problem, which will appear in Chapter 6. It admits classical and quantum algorithms whose time complexities are the square of those of this chapter. Ambainis gave an optimal algorithm in the QRAQM model [Amb07] and we gave the best algorithms to date in the plain circuit model (without qRAM gates) in [JS20].

### 2.3.1 Classical Algorithms

For a random function, collision search requires provably $\Omega(2^{n/2})$ queries. By the *birthday paradox*, among $2^{n/2}$ queries to $h$, a collision will occur with constant probability (since $2^n$ pairs are formed). Pollard's rho method, introduced in [Pol] with the purpose of factoring integers, gives an elegant solution meeting this bound with $\mathsf{poly}(n)$ memory. It relies on iterating the function $h$, starting with a random $x_0 \in \{0,1\}^n$ and defining the sequence $x_{i+1} = h(x_i)$.

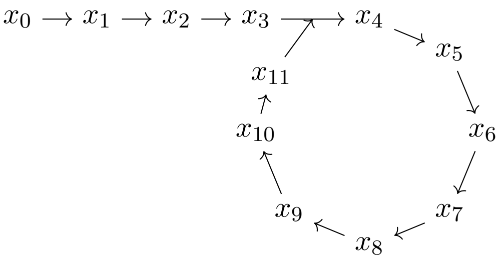


**Figure 2.2:** Illustration of Pollard's rho

After an expected number of $\mathcal{O}\left(2^{n/2}\right)$ iterations, a collision between the current $x_i$ and one of the previous $x_j$ will eventually occur, and the sequence will start to loop, as shown in Figure 2.2. Thus, the shape of the graph is reminiscent of the Greek letter $\rho$. The collision is found with a cycle-finding algorithm, for example Algorithm 2.9 (which is credited to Floyd by Knuth [Knu14]).

At Step 4, we have $x = y = h^i(x_0) = h^{2i}(x_0)$ for some $i$ and the value lies somewhere along the circle of the $\rho$. Let $i_0$ be the first index such that $h^{i_0+\ell}(x_0) = h^{i_0}(x_0)$, where $\ell$ is the length of the cycle. We know that $i$ is a multiple of $\ell$. At Step 9 we have $x = h^j(x_0) = h^{2i+j}(x_0)$ for some $j$, which implies that $j = i_0$ is the top of the $\rho$. Thus, $x' = h^{i_0-1}(x_0)$ and $y' = h^{2i+i_0}(x_0)$ are indeed distinct values. The algorithm will perform another loop if we want to retrieve the cycle length.

**There is no quantum Pollard's rho.** The idea of iterating the function has found many other applications in related problems, e.g., in van Oorschot-Wiener distributed collision search [vOW99] or Nikolic and Sasaki's time-memory trade-offs for the Generalized Birthday Problem [NS15]. However, this technique has found almost no use in quantum algorithms.

**Algorithm 2.9** Floyd's cycle-finding algorithm

```
Input: starting point $x_0$
Output: a collision of $h$
Initialize: $x \leftarrow x_0, y \leftarrow x_0$
repeat
    $x \leftarrow h(x)$, $y \leftarrow h^2(y)$
until $x = y$
Restart: $x \leftarrow x_0$
repeat
    $x' \leftarrow x, y' \leftarrow y'$
    $x \leftarrow h(x)$, $y \leftarrow h(y)$
until $x = y$
return $x', y'$
```

Iterating a function is done in [BB17] in a distributed multi-target preimage search algorithm. It is also done in [JS20], but without any advantage over other methods.

Intuitively, the probability of finding a collision of $h^i$ increases quadratically with $i$. This does not impact the time complexity, as $h$ must be evaluated the same number of times, but the memory can be reduced by storing chain-ends $h^i(x)$ of exponentially long chains instead of the whole chain. In the quantum setting, iterations must be performed in sequence, and iterating $h$ costs the same as classically. Meanwhile, quantum search achieves a quadratic advantage over exhaustive search. A direct Grover search for collision pairs $x, y$ among $(\{0,1\}^n)^2$ gives a time $\mathcal{O}\left(2^{n/2}\right)$, which competes with Pollard's rho. Hence, the potential of iterating is weakened by the relative efficiency of quantum search. This is why none of the quantum algorithms presented in this thesis use iterations of $h$.

**Multi-target preimage search.** For a random function, the best and optimal classical method for preimage search with $L$ targets is exhaustive search in time $\mathcal{O}\left(\frac{2^n}{L}\right)$ [And+08], using a random-access memory of size $L$. No classical single-processor algorithm with time-memory product smaller than $2^n$ is known. Even though, it is possible to divide efficiently the work among many processors, and / or to batch many attacks, by computing chain-ends as in Hellman's time-memory trade-off [Hel80; Oec03].

## 2.3.2 Quantum Collision Search

The first quantum collision search algorithm (Algorithm 2.10) with a query speedup was proposed by Brassard, Høyer and Tapp [BHT98]. Although they considered two-to-one functions, the algorithm works similarly with random functions, and we analyze it in this setting.

Using an appropriate data structure, the complexity of the algorithm is tight.

**Algorithm 2.10** Brassard, Høyer, and Tapp's (BHT) quantum collision search algorithm.

```
Input:  quantum query access to $h : \{0,1\}^n \rightarrow \{0,1\}^n$, a random function
Output: a collision of $h$
1: Build a list $\mathcal{L}$ of $L$ pairs $(x, h(x))$ for $L$ arbitrary values of $x$
2: Use Grover's algorithm to find $x \in \{0,1\}^n$ such that $\exists y \neq x, (y, h(x)) \in \mathcal{L}$
3: return $x, y$
```

**Theorem 2.8.** *Used with a memory of size $L$, Algorithm 2.10 outputs a collision using an expected number of $\mathcal{O}\left(L + \sqrt{\frac{2^n}{L}}\right)$ qRAM gates, queries and $n$-qubit register operations.*

*Proof.* We store $\mathcal{L}$ in a simple *unique prefix list* (Section 1.3.2), i.e., QRACM memory cells where each element is indexed by a prefix of $\log_2 L$ bits. After making $L$ queries, the number of cells that contain an element is on average $(1 - e^{-1})L$, since we have drawn $\log_2 L$ bits a number of $L$ times, and this is a random mapping from $\log_2 L$ bits to $\log_2 L$ bits [FO89].

Using a unique prefix list instead of a full-fledged set data structure is not a strong restriction if the function is random, and this enables us to perform membership queries using $\mathcal{O}(1)$ qRAM gates only.

Hence, Step 1 gives us $(1 - e^{-1})L$ distinct images. Since the function is random, the average probability, for an element among the $2^n - L$ remaining inputs, to collide on this list, is $\frac{(1-e^{-1})L}{2^n}$. The exact probability for a given instance is unknown, but we can use Amplitude Amplification. Thus, the number of iterates in Step 2 is on average $\mathcal{O}\left(\sqrt{\frac{2^n}{L}}\right)$. Each iterate contains only $\mathcal{O}(1)$ qRAM gates and $\mathcal{O}(n)$ other gates.

The time complexities between Step 1 and Step 2 are balanced when $L = \mathcal{O}\left(2^{n/3}\right)$, with $\mathcal{O}\left(2^{n/3}\right)$ time and queries. □

Algorithm 2.10 can be seen as a quantum version of the "naïve" classical collision search, which would build a table of size $2^{n/2}$. It can also be seen as a quantum algorithm for multi-target preimage search (Problem 2.7) specialized at finding collisions. As of today, no quantum collision search algorithm goes below the time-memory trade-off curve $T^2 \cdot M = 2^n$.

*Remark* 2.1. The unique prefix list of Section 1.3.2, where the memory addresses of the stored elements are given by their prefixes, is the most appropriate data structure for this kind of problem. Reading and writing require only $\mathcal{O}(1)$ qRAM gates and the list is populated with random bit-strings. This remains true for the merging algorithms that we will study in Chapter 5.

*Remark* 2.2. Grover and Rudolph [GR04] argued that the QRACM requirement of Algorithm 2.10 could make it more expensive than classical collision search, unless the QRACM can be realized as a physical component without any active error correction. Indeed, if maintaining $L$ memory cells costs $\mathcal{O}(L)$ error correction operations per time step, then the trade-off curve of the BHT algorithm is above the classical one. Later, Bernstein [Ber09] drew similar arguments. In fact, without QRACM, no quantum speedup was known previous to the algorithm that will be presented at the beginning of Chapter 5.

**Lower bound.** After this result, the query lower bound for the collision problem was improved in successive works [Aar02; AS04; Amb05; Kut05] until reaching the bound $\Omega(2^{n/3})$ for 2-to-1 functions, which was shown to hold as well for random functions [Zha15].

**Limiting the number of queries.** In Chapter 2, we have seen how the success probability of quantum exhaustive search scales with the number of queries allowed: if we search for a single solution in a space of size $2^n$, then $q$ quantum checking queries allow only $\mathcal{O}\left(\frac{q^2}{2^n}\right)$ probability of success. The scaling is quadratic in $q$ and not linear as in classical search. The same goes for collision search, where the classical quadratic scaling $\mathcal{O}\left(\frac{q^2}{2^n}\right)$ is replaced by a cubic one.

**Lemma 2.6** (From [Zha15])**.** *If $h$ is a random function, any quantum algorithm that makes $q$ queries to $O_h$ cannot output a collision with more than $\mathcal{O}\left(\frac{q^3}{2^n}\right)$ success probability.*

A matching upper bound consists in applying a truncated version of Algorithm 2.10. We first query $\frac{q}{2}$ elements, store them in QRACM, then run $\frac{q}{2}$ iterates of quantum search.

**Preimage search.** The generic algorithm for quantum multi-target preimage search is analogous to the classical one: it uses $\mathcal{O}\left(\sqrt{\frac{2^n}{L}}\right)$ iterations of Grover search, each of which queries a QRACM of size $L$ that holds the targets. Banegas and Bernstein [BB17] gave a distributed quantum algorithm for this problem, but it does not perform better for a single processor. The best time-memory trade-off for classical algorithms remains $T \cdot M = 2^n$, and the quantum trade-off is $T^2 \cdot M = 2^n$, with $M \leqslant L$ deciding how many targets we consider for the search.

### 2.3.3 Quantum Multicollision Search

In [HSX17], Hosoyamada, Sasaki, and Xagawa presented a quantum algorithm for multicollision search, which was later improved in [Hos+19]. For a constant $r$, it runs in time $\widetilde{\mathcal{O}}\left(2^{\frac{n}{2}\left(1-\frac{1}{2^{r-1}-1}\right)}\right)$, using the same amount of queries and QRACM. This query complexity was proven optimal for $r > 2$ by Liu and Zhandry [LZ19].

Optimizing Algorithm 2.11 with a classical exhaustive search at line 12 gives the classical complexity, up to a constant factor in $r$. Optimizing with quantum search instead gives:

$$\forall 1 \leqslant i \leqslant r, \ell_i = \left(1 - \frac{1}{2^{r-1}}\right)\left(1 - \frac{1}{2^i}\right) - \left(1 - \frac{1}{2^{i-1}}\right) \quad .$$

**Algorithm 2.11** Quantum multicollision-finding algorithm (version of [Hos+19]). Taking $r = 2$ gives Algorithm 2.10.

```
Input: $r$, query access to $O_h$
Output: an $r$-collision of $h$

Choose wisely $\ell_1, \ldots, \ell_r = 0$ depending only on $r$
$i \leftarrow 1$
Initialize $\mathcal{L}_1 \leftarrow \varnothing$
repeat
    Sample $x \in \{0,1\}^n$
    Add $(\{x\}, h(x))$ to $\mathcal{L}_1$
until $\mathcal{L}_1$ is of size $2^{n\ell_1}$
while $i < r$ do
    $i \leftarrow i + 1$
    Initialize $\mathcal{L}_i \leftarrow \varnothing$                ▷ $\mathcal{L}_i$ will contain $i$-collisions
    repeat
        Sample $x \in \{0,1\}^n$ such that $\exists (t, y) \in \mathcal{L}_{i-1}, h(x) = y$ and $x \notin t$
        Add $t \cup \{x\}, y$ to $\mathcal{L}_i$
    until $\mathcal{L}_i$ is of size $2^{n\ell_i}$
return the single element in $\mathcal{L}_r$
```

Since the total complexity is (up to a constant):

$$2^{\ell_1 n} + \underbrace{2^{\ell_2 n} 2^{\frac{1}{2}(1-\ell_1)n}}_{\text{Creating } \mathcal{L}_2} + \ldots + \underbrace{2^{\ell_i n} 2^{\frac{1}{2}(1-\ell_{i-1})n}}_{\text{Creating } \mathcal{L}_i} + \ldots + \underbrace{2^{\frac{1}{2}(1-\ell_{r-1})n}}_{\substack{\text{Finding the}\\ r\text{-collision}}} ,$$

and all terms should be of the same order. The result follows.

**Hidden constants.** The classical "hidden constant" of $r$-collision search is of the order $(r!)^{1/r}$: we are ensured that, given a $\{0,1\}^n \to \{0,1\}^n$ random function, the probability to find an $r$-collision after $(r!)^{1/r} 2^{n(r-1)/r}$ queries is higher than $1 - 1/e$. The situation is different in Algorithm 2.11.

From [FO89, 3.3, Theorem 4], the average probability that a given $i$-collision can be extended to an $i+1$-collision is $\frac{1}{i}$. Thus, there is a constraint on the sizes of the intermediate lists $\mathcal{L}_i$: $2^{n\ell_i} \geqslant \frac{(r-1)!}{(i-1)!}$. This suggests that the multiplicative factor may contain a factorial in $r$ (contrary to the classical setting).

# Chapter 3 Introduction to Cryptography

In this chapter, we set up some standard notions and definitions in cryptography. After a brief historical detour, we recall in Section 3.1 notions of security, of security margin, the role of cryptanalysis, and how quantum computers may put today's computational security guarantees in peril. We separate generic and dedicated attacks, both being studied in the following chapters. In Section 3.4, we define attack settings, notably "Q1" and "Q2", and common symmetric primitives such as ciphers, hash functions, MACs. We give a list of previous known generic attacks on ideal primitives, classical and quantum.

**Contents**

## 3.1 Overview

Cryptography is the science of protecting information, transmitted by insecure means, against adversaries which are either passively listening to the communications (eavesdroppers) or actively tampering with them. Its history dates back as early as the antiquity. One of the most prominent adopters of cryptographic techniques is certainly Julius Caesar. The roman historian Suetonius reports that he encrypted secret messages by shifting the letters by three positions in the Latin alphabet (the well-known *Caesar* *cipher*). In his account of the Gallic wars [Cae69, Book 5, Chapter 48], Caesar records using another trick to send a letter to the surrounded camp of his general Cicero.

> *Tum cuidam ex equitibus Gallis magnis praemiis persuadet uti ad Ciceronem epistolam deferat. Hanc Graecis conscriptam litteris mittit, ne intercepta epistola nostra ab hostibus consilia cognoscantur. Si adire non possit, monet ut tragulam cum epistola ad amentum deligata intra munitionem castrorum abiciat.*
>
> Then with great rewards he induces a certain man of the Gallic horse to convey a letter to Cicero. This he sends written in Greek characters, lest the letter being intercepted, our measures should be discovered by the enemy. He directs him, if he should be unable to enter, to throw his spear with the letter fastened to the thong, inside the fortifications of the camp.

As can be inferred from this excerpt, early cryptographers used a variety of *ad hoc* linguistic tricks. The paradigm shift that put cryptography in the hands of mathematicians seems to have occurred in modern times, notably with the foundational work of Auguste Kerckhoffs [Ker83a; Ker83b], who edicted six rules known today as *Kerckhoffs' principles.* These rules were intended to guide the design of ciphers for military use. Nowadays, computers have replaced electromechanical ciphering machines and cryptosystems, which were a tool for diplomatic and military personnel, have become essential building blocks of the Information Era. Yet, Kerckhoffs' principles have remained valid.

**Computational secrecy.** Kerckhoff's first principle dictates: *The cryptosystem should be materially or mathematically impossible to decrypt.* Modern cryptography focuses on *computational security*, introduced by Shannon in [Sha49], where the problem of recovering the secret information ought to be computationally intractable.

*Remark* 3.1. This situation was translated in terms of complexity classes by Diffie and Hellman [DH76]: if encryption and decryption under a given secret are polynomial-time operations, then the problem of finding the secret belongs to the complexity class $\mathsf{NP}$, but it should be sufficiently hard in this class.

**Public algorithms.** Kerckhoff's second principle reads: *The system should not require secrecy. It should not be a problem if it falls into enemy hands.* Nowadays, common sense dictates that the specification of a cryptographic algorithm (or function, or *primitive*) should be public.

**Portability and lightness.** Kerckhoff's four other principles focus on the usability of the cryptosystem. He required it to be compatible with telegraph transmissions and to require only few human resources to handle and operate. Today, cryptographic algorithms are not computed by hand, but equivalent necessities still prevail. As more and more devices are being connected to the Internet, cryptographic algorithms are challenged by very constrained environments, with limits either in latency, number of operations, or circuit size. Thus, a need for secure *lightweight* primitives has arisen, embodied today by the ongoing standardization process launched by the NIST (see Section 3.3).

**Cryptanalysis.** While cryptography is the science of protecting information, cryptanalysis is the science of overcoming the defenses that cryptographers tried to set up. The oldest known written work about cryptanalysis is by Al-Kindi, an Arab scientist and philosopher born around 801 A.D. In his book *A Manuscrit on Deciphering Cryptographic Messages*, he introduced notably the statistical cryptanalysis of mono-alphabetic ciphers such as Caesar's, and the use of probable words [Sin99; Al-92]. Nowadays, cryptography and cryptanalysis are deeply intertwined. In order to have good working cryptography, both lines of work are necessary, and they must communicate deeply with each other: a public effort of cryptanalysis incurs confidence in the designs that have remained unbroken. Meanwhile, our knowledge of cryptanalysis techniques, which is constantly renewed, helps in creating designs that reach a better balance between cost and security.

**Branches.** Cryptography is nowadays divided into two main subfields: symmetric (Section 3.1.1) and asymmetric (Section 3.1.2).

### 3.1.1 Symmetric Cryptography and its Security

The elderly branch of cryptology, that dates back to Caesar and Al-Kindi, is *symmetric* or *secret-key* cryptography. In this setting, the two users, conveniently named Alice and Bob, are assumed to possess a shared *secret key* $K$ that enables them to communicate securely on an insecure channel (e.g., a Gallic horseman, or the Internet).

**Generic bounds.** Symmetric designs start from *primitives*, small algorithms (for example, block ciphers or permutations) that handle a fixed amount of data, e.g., 128 bits, and offer only little functionality.

In full generality, a primitive is considered *secure* if it behaves like an *ideal* primitive, where the definition of *ideal* depends on the context. Writing a cryptanalytic attack is, in a sense, writing a proof that the primitive is not ideal.

There are *generic bounds* that apply to all primitives of a certain type, e.g., for a cipher, exhaustive search of the secret key $k$. Hence, an ideal cipher only needs to dimension the parameter $k$ according to the security level wanted. For example, while keys of 56 bits seemed fine at the time it was designed, the former cryptographic standard [DES] is today an easy target for this attack.

**Dedicated attacks.** The goal of cryptanalysis is to ensure that a given primitive does not admit better *dedicated attacks* than the generic ones. If an attack exists, regardless of its practicality, the primitive can be declared *broken.* An attack reveals a weakness that cannot be fixed without changing the algorithm, and that can only get worse. Thus, a continued study of both generic and dedicated attacks is necessary to set the generic bound, check that it remains impractical and verify that it is met.

**Notion of security margin.** The security of a primitive is not a one-bit information. It is commonly inferred by the success of attacks on weakened versions. As an example, the best key-recoveries against the standard AES-256 [AES] manage to break a version reduced to 9 rounds out of 14. Thus, AES-256 still benefits of a 5/14 = 36% margin.

**Constructions and generic attacks.** Primitives are used as building blocks in *constructions* such as modes of encryption, MACs, AEAD schemes, *etc.* If the primitive behaves as ideal, then only some *generic* attacks can be applied to the construction. But these attacks also need a careful study. Figure 3.1 summarizes these distinctions.

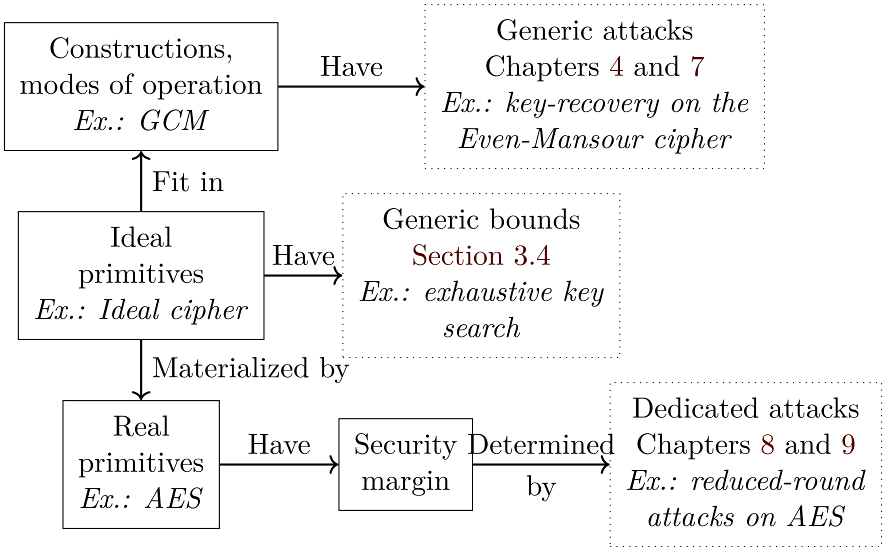


**Figure 3.1:** Landscape of symmetric attacks.

## 3.1.2 Asymmetric Cryptography and Hard Problems

Introduced by Diffie and Hellman in a seminal work [DH76], *asymmetric* (or *public-key*) cryptography considers a setting where Alice and Bob do not have any pre-established secret. Still, they want to communicate securely. Diffie and Hellman proposed to use a *public-key cryptosystem*, in which the encryption key $K_e$ is different from the decryption key $K_d$. Further, they introduced the Diffie-Hellman key exchange (Algorithm 3.1).

Later on the RSA cryptosystem, based on the hardness of integer factorization, was designed by Rivest, Shamir and Adleman [RSA78]. It became the most widely used public-key scheme.

**Algorithm 3.1** Diffie-Hellman key-exchange between Alice and Bob.

```
**Setup:** Alice and Bob select an Abelian group, e.g., $\mathbb{Z}_p$ for a prime $p$, and a random generator $g$ of $\mathbb{Z}_p^*$
Alice does: $a \xleftarrow{\$} \mathbb{Z}_p^*$
Bob does: $b \xleftarrow{\$} \mathbb{Z}_p^*$
Alice sends $g^a$; Bob sends $g^b$
Both Alice and Bob compute: $(g^a)^b = (g^b)^a$
```

**Security notions.** The security of public-key schemes follows a slightly different approach from symmetric schemes. In public-key cryptography, the cryptosystem's security is reduced to one or more *hard problems*, believed to be intractable. Many of these problems are not proven to be NP-complete, and admit better algorithms than simple exhaustive search. For example, the Discrete Logarithm Problem (DLP) in $\mathbb{Z}_p$ (and so, a hard problem behind the Diffie-Hellman key-exchange in $\mathbb{Z}_p$) is solved in subexponential time with index calculus methods.

**Problem 3.1** (Discrete Logarithm Problem)**.** *Let $g$ be a generator of $\mathbb{Z}_p^*$. On input $g^a$, where $a$ is chosen uniformly at random, compute $a$.*

A public-key cryptosystem is said *broken* if the underlying hard problem turns out *not* to be hard, that is, solvable in polynomial time in the system's parameters. While the symmetric cryptanalysis effort focuses in breaking specific primitives or constructions, public-key cryptography requires in addition to improve the algorithms available for these problems. Then, the parameter sizes (e.g., the bit-size of $p$ in Algorithm 3.1) are dimensioned as to meet some given security levels. Public-key schemes are much heavier than secret-key schemes, whether in time, memory or bandwidth, and determining their security level with precision is necessary, if only to compare them fairly.

## 3.2 Post-Quantum Cryptography

In 1994, Shor [Sho94] discovered a quantum algorithm that solves the Abelian hidden period problem in polynomial time.

**Problem 3.2** (Abelian hidden period, generalized)**.** *Let $(G, +)$ be an Abelian group, $X$ a set. Let $f : G^k \to X$ be a function such that there exists $s_1, \ldots, s_k$ with:*

$$\forall x_1, \ldots, x_k, f(s_1 + x_1, \ldots, s_k + x_k) = f(x_1, \ldots, x_k) \enspace ,$$

*then find $(s_1, \ldots, s_k)$.*

In particular, it can solve the Discrete Logarithm Problem in Abelian groups and factor large integers in polynomial time, problems on which most public-key cryptography in use until today was based on, notably RSA and ECC (using the group law on elliptic curves). In light of this breakthrough, the NIST launched in 2016 a standardization process [NIS16] for *post-quantum* primitives, which reached its third round in July 2020.

As they should resist an attacker equipped with a quantum computer, the security of these primitives is based on problems which should be quantumly intractable, such as decoding linear codes, finding paths in supersingular isogeny graphs, finding short vectors in integer lattices. Many works have studied and improved the generic quantum algorithms for these problems, a full list of which would be outside the scope of this chapter. We can cite [Ber10; KT17] for generic decoding, [LMP13] for lattice sieving, [ANS18] for lattice enumeration.

However, to date, all the schemes broken during the process were broken using *classical* attacks, which suggests that more dedicated *quantum cryptanalysis* is yet necessary.

**Post-quantum symmetric cryptography.** Symmetric constructions seem easy to protect at first sight. Most problems that they are based on are naturally unstructured, and do not seem to benefit from any exponential speedup. Although Grover's algorithm can be used to accelerate exhaustive key search, it suffices to double the size of the parameters, in particular the key size, in order to provide an equivalent ideal security. Furthermore, Grover's algorithm does not benefit from a good parallelization, and quantum operations will likely remain orders of magnitude more costly than classical ones.

However, the generic attacks given by Grover's algorithm only reveal an upper bound on the security, not a lower bound. In 2010, Kuwakado and Morii [KM10] introduced the first attack in symmetric cryptography in the *superposition query model*, that we will define in Section 3.4.2. This started a series of quantum polynomial-time attacks on classically secure symmetric cryptosystems [KM12; Kap+16a]. Their common point is that they exploit an algebraic structure as a promise problem, and use Simon's algorithm to recover the secret.

The practical implications of these powerful attacks are yet unclear, but security in this stronger model has appeared more and more desirable over the time, if only to ensure compatibility between the different definitions of quantum security and proof frameworks that have flourished [GHS16; Unr17; SY17; BZ13a; Zha12; BZ13b; Zha19].

Besides, *doubling the key size* is a non trivial requirement. If the primitive does not allow it, we must change the primitive, use a new design and restart cryptanalysis. Furthermore, it leaves behind many other attack vectors, for example, generically exploiting a small block size. To date, most ciphers that allow 256-bit keys have a block size limited to 128 bits, for example the standard AES. Finally, quantum cryptanalysis is our only way to gain an understanding of the potential dedicated quantum attacks.

**On the "quantum world".** The quantum world is not very different from the classical world: we try to make primitives secure against adversaries with bounded resources, either computational or in queries. In asymmetric cryptanalysis, we try to find better algorithms for solving the underlying hard problems, but we must as well look into concrete constructions. In symmetric cryptanalysis, we find the best dedicated attacks performing better than the generic one, for example Grover's algorithm. The security margin is determined with respect to this attack. Although our perspective now includes quantum computations and queries, the life cycle of cryptographic primitives has remained the same.

In a post-quantum future, we can imagine that the *quantum* surname (quantum world, quantum cryptanalysis, quantum security margin, quantum generic attack. . . ) will disappear. We will just care about the "best attack", reaching the highest number of rounds, whether classical or quantum. Thus, we won't stop to study classical attacks: a primitive can only be secure if it is classically secure.

This is why, despite the emphasis on the quantum setting that is put in the title of this thesis, most of the dedicated cryptanalysis performed in Chapter 8 will be classical only. The analysis of GIMLI will provide examples of attacks that we study first in the classical setting, then modify and extend when switching to the quantum setting.

## 3.3 Lighweight Cryptography

With the unstoppable growth of the *Internet of Things* (IoT), cryptographic algorithms are now ubiquitous and required to run not only on personal computers or servers, but also low-power micro-controllers, low-area circuits, RFID chips, and so on. These new constraints motivated a wave of new *lightweight* designs.

As there are different constraints involved, this term actually regroups algorithms with very different design goals: some of them target the energy consumption, the code size, circuit area, latency, throughput, RAM consumption, *etc.* Some algorithms can also be optimized for side-channel resistance or masking.

In their survey [BP17], Biryukov and Perrin argued that the field could actually be split into two subfields, *ultra-lightweight cryptography* and *ubiquitous cryptography.* A cipher with a low area implementation, aiming at Application Specific Integrated Circuits (ASICs) but with worse software performances, would be included in the former category, whereas the latter would designate versatile algorithms with fairly efficient implementations on micro-controllers, ASICs and even software at the same time.

**Standardizing lightweight cryptography.** In 2019, the American National Institute of Standards and Technology (NIST) launched a standardization process for new *lightweight* cryptographic primitives [NIS17; NIS18]. Submission packages were required to define at least an authenticated encryption algorithm and, optionally, a hash function. The NIST plans to wait until the final rounds to determine whether they will select a single algorithm (which would likely be an *ubiquitous* one in the definition of Biryukov and Perrin) or a portfolio (which would likely contain specialized *ultra-lightweight* ciphers).

57 candidate algorithms were submitted in February 2019. 56 of them were announced as first-round candidates in April 2019. On August 30, 2019, 32 or them passed to the second round [NIS19], among which Spook [Bel+20], GIMLI [Ber+19] and

Saturnin [Can+20], which will appear in Chapters 8 and 10 respectively. We shall consider only the attack settings of Section 3.4.2, and in particular, no leakage scenarios.

# 3.4 Symmetric Primitives

In this section, we recall some standard attack scenarios in symmetric cryptanalysis, we define some ideal primitives and give classical and quantum generic bounds.

## 3.4.1 Primitives

We recall the main families of primitives in symmetric cryptography. They are functions acting on bit-strings of fixed or variable length. Each definition gives an *interface* offering some functionality, and expected security guarantees such as: • *confidentiality*: the secret message should not be recoverable by the attacker; • *integrity*: the attacker should not be able to tamper with the message; • *authenticity*: the attacker should not be able to impersonate the sender.

We give only a few examples of specific constructions, deferring most of them to the chapters in which they will be needed.

### 3.4.1.1 Block Ciphers

A *block cipher* is a family of permutations $E_k$ of a message space $\{0,1\}^n$, indexed by a key space $\{0,1\}^m$, where $n$ is the *block size* and $m = \mathcal{O}(n)$ is the *key size*:

$$\begin{cases} E: & \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n \\ D: & \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n \\ \forall k \in \{0,1\}^m, x \in \{0,1\}^n, & D_k(E_k(x)) = E_k(D_k(x)) = x \end{cases} .$$

The strongest ideal model of a block cipher is the *ideal cipher model* introduced by Shannon [Sha49], in which a block cipher is selected uniformly at random from all families of permutations $\Pi_k$ indexed by $k$. However, it is possible to construct primitives secure with an ideal cipher and insecure with any practical instantiation [Bla06]. Many proofs require instead the cipher to be a *pseudorandom permutation* PRP: when $k$ is selected uniformly at random from $\{0,1\}^m$, the permutation $E_k$ should be indistinguishable from a randomly chosen permutation $\Pi$.

**Generic key-recovery.** We will consider chosen-plaintext or -ciphertext key-recovery attacks. The classical generic bound, in evaluations of $E$., is $\mathcal{O}\left(\left\lceil \frac{m}{n} \right\rceil 2^m\right)$ using $\left\lceil \frac{m}{n} \right\rceil$ chosen-plaintext queries. The adversary tries all possible keys and, for each key, re-encrypts the given plaintexts, trying to match the given corresponding ciphertexts. The quantum generic bound is $\mathcal{O}\left(\left\lceil \frac{m}{n} \right\rceil 2^{m/2}\right)$ using *the same classical data.*

*Remark* 3.2. Determining precise bounds for Grover search requires to dive into the details of a quantum circuit evaluating $E_k(x)$ for a given $k, x$. The case of [AES] will be studied in Chapter 9.

There are two main families of block ciphers: Feistel networks and Substitution-Permutation Networks (SPNs). The former includes notably the former standard [DES], and the latter the current standard [AES]. In this thesis, we mainly study SPNs.

**SPNs.** They consist in iterating a round function which applies a key addition, followed by a *substitution layer* and a *permutation layer*. The substitution layer applies small nonlinear functions, or S-Boxes to the state. The permutation layer mixes the whole state using an efficient linear function. The key of each round is derived from the *master key* $k$ through a key-derivation algorithm, which may be as simple as the addition of a round-dependent constant.

**Feistel networks.** The Feistel network is a generic construction for turning functions into permutations. The state of the cipher, of $n$ bits, is divided into two branches $L_0, R_0$ of $n/2$ bits each. Then, $r$ rounds are applied, using round functions $f_0, \dots, f_{r-1}$:

$$L_{i+1}, R_{i+1} = R_i, L_i \oplus f_i(R_i) \ .$$

This operation is invertible. The full $\mathsf{Feistel}(f_0, \dots, f_{r-1})$ is thus invertible for any choice of functions.

As a block cipher proposes a very limited functionality (it only encrypts a single block and does not ensure authenticity), it is necessary to use it inside a *mode of operation*.

### 3.4.1.2 Stream Ciphers

A *stream cipher* encrypts messages of variable length (up to some limit). The key space remains $\{0,1\}^m$ and we introduce a *nonce* space $\{0,1\}^\nu$. The nonce (contraction of *number used once*), sometimes *initial value* or IV, is a public value used to avoid re-encryption.

$$\begin{cases} E : & \{0,1\}^m \times \{0,1\}^\nu \times \{0,1\}^* \to \{0,1\}^* \\ D : & \{0,1\}^m \times \{0,1\}^\nu \times \{0,1\}^* \to \{0,1\}^* \\ \forall k, x, N \in \{0,1\}^\nu, & D_k(E_k(x;N);N) = E_k(D_k(x;N);N) = x \end{cases} .$$

In a stream cipher such as Grain-128a (the basis of the LWC candidate [HJM] Grain-128AEAD), each combination of key and nonce generates a different sequence called the *keystream*, which is typically XORed to the message to generate the ciphertext.

A stream cipher can be built from a block cipher using an encryption mode, for example the *counter mode* (CTR) [Nat01] which is used in SATURNIN [Can+19] (see Figure 10.10 in Section 10.2.3). Note that the use of any block cipher mode of operation requires first a *padding scheme*, to cut the message $m$ into a sequence of blocks $m_0, \dots, m_1, m_\ell, m_*$ where $m_*$ is the last (padded) block.

### 3.4.1.3 Hash Functions

A *hash function* is a function that is hard to invert, and produces an output of fixed length $n$ (the hash) from an input message of any length:

$$\left\{ h : \{0,1\}^* \to \{0,1\}^n \right. \quad .$$

It is mainly used to provide integrity (and helps with authentication, e.g., when storing passwords). An *ideal* hash function cannot be distinguished from a random function.

**Attacks.** The main attacks on hash functions, that lead to exploitable weaknesses, consist in finding collisions (see Section 2.3), preimages, and *second preimages*: given $x$, find another input $y \neq x$ such that $h(y) = h(x)$.

There are also different ways to build hash functions from smaller components. The Merkle-Damgårg construction [Mer89; Dam89], that is used in Saturnin-Hash in Chapter 10, uses a compression function of fixed length. Another example is the Sponge construction [Ber+07; Ber+11b], which is used in several algorithms considered throughout this thesis. It is based on a public permutation.

The basic layout is displayed in Figure 3.2. The state of the permutation $\Pi$ is divided into two parts, the *rate* $r$ and the *capacity* $c$. The rate determines in part how fast the sponge will be computed, whereas the capacity mainly determines its security level. The function is computed in two phases. In the *absorbing phase*, the padded message blocks $m_1, \ldots, m_t$ are XORed in the rate part, before a call to the permutation $\Pi$. After all message blocks have been processed, we enter the *squeezing phase*, in which the rate part of the sponge is extracted. These concatenated blocks form the output of the function. The current hash function standard [SHA3] is a Sponge.

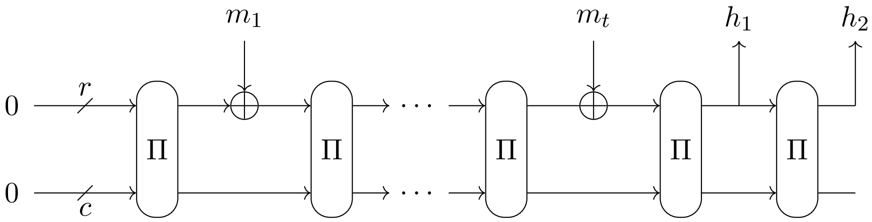


**Figure 3.2:** Example of Sponge with two output blocks.

Sponge functions benefit from strong security guarantees. If $\Pi$ is a random permutation, then a Sponge built with $\Pi$ is indistinguishable from a random function up to $2^{c/2}$ queries. This corresponds to finding a collision on the capacity part, which yields a collision when inserting the right messages. In the quantum setting, sponges were proven collision-resistant up to $2^{c/3}$ queries [Cza+18], which is the quantum collision lower bound on $c$ bits, and thus, is also essentially optimal. Indistinguishability was shown in [CHS19].

#### 3.4.1.4 MACs

A *message authentication code* is a function MAC of a key space $\{0,1\}^m$ and message space $\{0,1\}^*$, producing a *tag* of length $t$:

$$\left\{ \mathrm{MAC} : \quad \{0,1\}^m \times \{0,1\}^* \to \{0,1\}^t \right. \quad .$$

It is intended to provide integrity, similarly to a hash function, but also authenticity, since the knowledge of the secret key $k$ is required to compute the tag.

**Attacks.** Standard attacks on a MAC either try to recover the key, or to build *forgeries*: creating a valid (message, tag) pair for a message that has not been previously queried. Thus, we can either search exhaustively for $k$ or try to guess the value of the tag, in time $\mathcal{O}\left(2^t\right)$. In the quantum setting, we will accelerate this forgery if we have Q2 access to a verification oracle, using Grover search.

The definition that we just gave is that of a *deterministic MAC*. For iterated deterministic MACs, there exists a generic forgery attack [PO95] of complexity $2^{n/2}$ if $n$ is the *internal state size*. Thus, there are three alternatives to obtain better security: either use a *nonce* or a randomly chosen *salt*, that is transmitted alongside the message, or design a MAC with a bigger internal state. There have been many proposals for the latter, studied from a cryptanalytic standpoint in [LNS18].

#### 3.4.1.5 Authenticated Encryption

An *authenticated encryption scheme* (AE) offers both the functionalities of a stream cipher and a MAC. With *associated data* (AEAD), it accepts an additional input which is not encrypted, but contributes to the tag. As before, it is possible to define a standalone AEAD (such as Grain-128AEAD [HJM]), or to build it from smaller components.

**Example: Encrypt-then-MAC.** In the Encrypt-then-MAC paradigm, Alice will first compute the ciphertext $c$ using a mode of encryption, then $\mathrm{MAC}_k(c)$. Upon reception, Bob will recompute the MAC and check that the tags match. This is used in SATURNIN (Chapter 10).

**Example: Duplex Mode.** The Duplex Mode [Ber+11a] allows to build an authenticated cipher with a Sponge. Its main idea is that, after XORing each message block $m_i$, the current value of the rate is outputted as a ciphertext block $c_i$. Then after absorbing the whole message, the squeezing phase returns an authentication tag. This provides an AE functionality with a single pass. This design pattern has been used by many candidates in the LWC process, including the second-round candidates [Ber+19; Bel+19] studied in Chapter 8.

### 3.4.2 Attack Settings

We consider *classical* and *quantum* adversaries (or attackers), which are algorithms (respectively classical and quantum) that can have black-box query access to a primitive holding a secret $k$ (this is the secret-key setting), e.g., a cipher $E_k$. In general we consider that there is a single secret, although the *multi-user setting*, where many secrets $k_i$ are used on the same communication channel, has also practical meaning. Classical attack scenarios include:

**Known-ciphertext:** the attacker can only obtain encrypted ciphertexts $E_k(m_i)$

**Known-plaintext:** the attacker obtains plaintext-ciphertext pairs $(m_i, E_k(m_i))$ with random or arbitrary $m_i$

**Chosen-plaintext:** the attacker can obtain plaintext-ciphertext pairs $(x_i, E_k(x_i))$ with chosen $x_i$

**Chosen-ciphertext:** the attacker can obtain pairs $(x_i, E_k(x_i))$ with chosen $x_i$ *or* chosen $E_k(x_i)$ (encrypting and decrypting at will)

In the quantum setting, the adversary can perform quantum computations, but the way she[1] accesses the data plays an important role. We adopt the *Q1/Q2* terminology used in [Kap+16a; Kap+16b; HS18a; HS18b].

**Q1.** The Q1 setting is called QS1 in [Gag17] and is sometimes called the *standard query model* in the literature on quantum provable security (*standard IND-CPA* in [Ana+16]). Although the adversary can perform *offline* quantum computations, she can only access the primitive through *classical* queries, whether it be known, chosen-plaintext or ciphertext. This attack setting is certainly the most meaningful. It will be applicable to all communications once a scalable quantum architecture is built, but it also concerns today's encrypted traffic, if an attacker can record it and wait patiently for a quantum computer to appear.

To date, the best Q1 quantum attacks known reach a quadratic speedup, based on quantum search, and most of them also imply classical attacks. The only known example of quantum Q1 attacks with no classical equivalent comes from the relative inefficiency of quantum collision search against quantum search [HS20] (see Section 4.2.2).

**Q2.** Here, the adversary performs quantum computations and has quantum oracle access. Different types of oracles can be considered, but we will focus on the standard oracle $O_{E_k}$:

$$|x\rangle |0\rangle \xmapsto{O_{E_k}} |x\rangle |E_k(x)\rangle \ .$$

*Remark* 3.3. The word "standard" in "standard oracle" is not to be confused with "standard" in "standard query model". We will prevent this by using the Q1 / Q2 terminology.

[1] We consider that Eve is the mastermind behind this quantum attack.

Note that Q2 corresponds to the *quantum query model* in the literature on provable quantum security, for instance in [Dam+13; BZ13b; Zha12; GHS16; Bon+11].

The impact of the attacks using the Q2 model is unclear, but it has led to powerful breaks of classically secure constructions, some of which will be discussed in Section 4.1. There is though no question about the desirability of Q2 security. • The model is simple and inclusive of any theoretical intermediates, for example, a *quantum side-channel* attack in which the adversary could trick a device into performing quantum computations. • The model is non-trivial: although there have been breaks, the Q2 attacks generally target constructions with a strong algebraic structure, and many designs retain their security. • The model seems easier to work with in quantum provable security. • The model includes the scenarios of white-box encryption or obfuscation, in which the adversary has access to a full description of the oracle with a secret key and can implement a quantum embedding of it.

**Stronger models.** Rötteler and Steinwandt [RS15] showed that a very strong model of *superposition related-key attacks* allowed to break all block ciphers. In this model, a secret $k$ is fixed and we are given oracle access to:

$$|x\rangle\,|\ell\rangle\,|0\rangle \xmapsto{O_{E_{k\oplus\cdot}}} |x\rangle\,|\ell\rangle\,|E_{k\oplus\ell}\,(x)\rangle \quad .$$

But then, since the specification of $E_\cdot$ is known, the adversary can implement a quantum oracle for the function $f(\ell) = E_{k\oplus\ell}\,(0) \oplus E_\ell\,(0)$, which is a random periodic function of period $k$, and use Simon's algorithm (Algorithm 1.1) to recover $k$. This shows that the model is intrinsically too strong, and it is not considered meaningful today.

**Q2 setting in public-key cryptography.** As we have seen, the assertion of security in public-key cryptography comes from designing algorithms for generic problems, such as factorization or discrete logarithms. The input will always be classical. Grilo, Kerenidis, and Zijlstra [GKZ19] showed that the learning-with-errors (LWE) problem, which underlies multiple candidates of the NIST call [NIS16], could be broken in polynomial time with superposition access, using the Bernstein-Vazirani algorithm [BV93]. Their attack was also improved by Alagic, Jeffery, Ozols, and Poremba [Ala+20a].

### 3.4.3 Summary of Main Generic Attacks

We give in Table 3.1 a non-exhaustive list of generic attacks considered in this thesis, for each type of primitive, and bounds to compare our cryptanalysis to. Time complexities are given up to some constant, for a constant probability of success, in number of evaluations of the primitive. We omit $\mathcal{O}$ notations for simplicity. Query complexities are given in number of black-box queries to the primitive. Memory is given in blocks of queried data.

Note that for block cipher modes of operation, we consider the number of blocks queried. Thus, a quantum chosen-plaintext query with $r$ blocks corresponds to $r$ data.

A hash function computation with $r$ blocks will cost a time $r$. Generic attacks usually use inputs of a single block.

**Table 3.1:** Main classical and quantum generic attacks on block ciphers, hash functions and MACs. Polynomial factors are omitted from the time complexity.

| Primitive | Attacks | Time | Data | Memory |
|---|---|---|---|---|
| | | Classical | | |
| Block cipher<br>$n$ block size<br>$m$ key size | Key-recovery | $2^m$ | $\lceil \frac{m}{n} \rceil$ KP | $\lceil \frac{m}{n} \rceil$ |
| | $2^t$-user key-recov | $2^{m-t}$ | $\lceil \frac{m}{n} \rceil 2^t$ KP | $\lceil \frac{m}{n} \rceil 2^t$ RAM |
| Hash function<br>$n$ output size | Collision | $2^{n/2}$ | None | $\mathsf{poly}(n)$ |
| | Preimage | $2^n$ | None | $n$ |
| | $2^t$-target preimage | $2^{n-t}$ | None | $n2^t$ RAM |
| MAC<br>$t, m$ | Forgery | $2^t$ | $2^t$ verif. queries | $t$ |
| | Key-recovery | $2^m$ | $\lceil \frac{m}{t} \rceil$ KP | $\lceil \frac{m}{t} \rceil$ |
| | | Quantum | | |
| Block cipher | Key-recovery | $2^{m/2}$ | $\lceil \frac{m}{n} \rceil$ KP (Q1) | $\lceil \frac{m}{n} \rceil$ |
| | $2^t$-user key-recov | $2^{(m-t)/2}$ | $\lceil \frac{m}{n} \rceil 2^t$ KP (Q1) | $\lceil \frac{m}{n} \rceil 2^t$ QRACM |
| Hash function | Collision | $2^{n/2-L/2}$ | None | $n2^L$ QRACM |
| | Preimage | $2^{n/2}$ | None | $n$ |
| | $2^t$-target preimage | $2^{(n-t)/2}$ | None | $n2^t$ QRACM |
| MAC | Forgery | $2^{t/2}$ | $2^{t/2}$ Q2 verif. queries | $t$ |
| | Key-recovery | $2^{m/2}$ | $\lceil \frac{m}{t} \rceil$ KP | $\lceil \frac{m}{t} \rceil$ |

## 3.5 Preliminaries of Cryptanalysis

In this section, we briefly introduce some cryptanalysis notions used in the next chapters.

### 3.5.1 Differential Cryptanalysis

*Differential cryptanalysis* is one of the most fruitful families of cryptanalytic algorithms. It was introduced by Biham and Shamir [BS91] in order to attack the [DES]. It studies the propagation of differences through multiple rounds of a cipher. A pair of differences

$\Delta_{in}, \Delta_{out}$ such that, with relatively high probability $p$ over $m, m'$:

$$m \oplus m' = \Delta_{in} \implies \Pi(m) \oplus \Pi(m') = \Delta_{out} \ ,$$

is denoted a *differential*. In differential cryptanalysis, one typically finds a differential pattern covering a certain number of rounds of the cipher, and appends one or more rounds before and after this pattern, performing an exhaustive search of some key bits.

*Truncated differential cryptanalysis* is an extension by Knudsen [Knu94] in which the condition $m \oplus m' = \Delta$ becomes $m \oplus m' \in E$, where $E$ is a vector space of higher dimension. It is common to encounter such a situation, in which the difference between $m$ and $m'$ (and their images) is only partially determined. Instead of defining $E$ formally, we note for example: $m = 0 * * * 01 * *$, where $*$ denotes a part of the state where the difference is not fixed.

The extension of differential and truncated differential cryptanalysis to the quantum setting, using Grover's algorithm to speedup such key-recovery attacks, was studied in [Kap+16b].

### 3.5.2 Distinguishers on Permutations

Cryptographic algorithms based on public permutations have been widely adopted, and they actually represent the majority of second-round candidates in the NIST LWC process. But the security of many of these algorithms relies on the absence of a *distinguisher* for the permutation used.

In the case of the Sponge construction, this is the *hermetic sponge strategy*: a *strong* permutation $\Pi$ is used, that is, it should not have any "exploitable properties" [Ber+11b], or exhibit any property that distinguishes is from a random permutation, outside the existence of a compact implementation. Such a property of $\Pi$ implies the security of the Sponge. Of course, the converse is not necessarily true, and if $\Pi$ admits a distinguisher, this does not necessarily yield an attack on any construction using it. But depending on the type of distinguisher used, it may raise some concerns.

### 3.5.3 Limited-birthday Distinguishers

The limited-birthday problem was introduced by Gilbert and Peyrin [GP10] in the known-key cryptanalysis of the [AES]. It requires to find a pair of inputs satisfying a truncated differential.

**Definition 3.1** (Limited-birthday problem)**.** Given access to a permutation $\Pi : \mathbb{F}_2^n \to \mathbb{F}_2^n$, given two vector spaces $E, F$ of $\mathbb{F}_2^n$ of respective dimensions $d_{in}$ and $d_{out}$, find a pair of inputs $(x, x') \in (\mathbb{F}_2^n)^2$ such that $x \oplus x' \in E$ and $\Pi(x) \oplus \Pi(x') \in F$.

Since access to both $\Pi$ and $\Pi^{-1}$ is allowed, the generic algorithm chooses depending on the values of $d_{in}$ and $d_{out}$.

**Theorem 3.1** ([GP10])**.** *There exists an algorithm that generates a limited-birthday pair in time:*

$$\max\left(\min_{d\in\{d_{in},d_{out}\}}\sqrt{2^{n+1-d}}, 2^{n+1-(d_{in}+d_{out})}\right) \quad ,$$

*counted in number of queries to* $\Pi$ *and* $\Pi^{-1}$*, and using*

$$\min\left(2^{d_{in}}, 2^{d_{out}}\right)$$

*classical memory with random access.*

The optimality of this method is proven in [IPS13] for a black-box function, and in [HNS20] for a black-box permutation.

**Definition 3.2** (Limited-birthday distinguisher)**.** Let $\Pi : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be a permutation, e.g., Shadow-512. A *limited-birthday distinguisher* against $\Pi$ is given by a pair of vector spaces $E, F$ of $\mathbb{F}_2^n$ of respective dimensions $d_{in}$ and $d_{out}$, and an algorithm that solves the limited birthday problem in less time than the formula of Theorem 3.1.

Many of these distinguishers borrow the techniques of block cipher cryptanalysis. The existence of fast algorithms to find such constrained pairs of inputs is an example of a "non-random" property of the permutation.

# Chapter 4
# Quantum Symmetric Cryptanalysis

In Chapters 1 and 3, we have introduced quantum computing and symmetric cryptography. We now bring these two worlds together and review previous works in *quantum symmetric cryptanalysis*. We separate these quantum attacks into the Q2 setting and the Q1 setting.

**Contents**

## 4.1 Q2 Attacks on Symmetric Constructions

We will now review Q2 attacks from the literature, applying to popular generic constructions in symmetric cryptography. Most of these attacks rely on quantum hidden shift algorithms, except the last one that we will present, which differs significantly.

### 4.1.1 Distinguishers on the Luby-Rackoff Constructions

Luby and Rackoff [LR88] studied a three-round Feistel network as shown in Figure 4.1. They showed that if the functions $f_0, f_1, f_2$ are pseudorandom functions, then $\mathcal{O}\left(2^{n/4}\right)$ queries to $\mathsf{Feistel}(f_0, f_1, f_2)$ are necessary to distinguish it from a random permutation. Adding another round, they found that the same holds for $\mathsf{Feistel}(f_0, f_1, f_2, f_3)$ even with

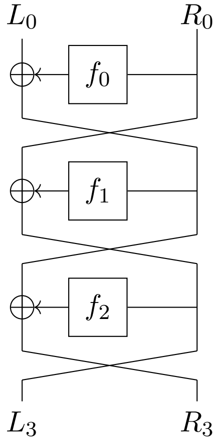


**Figure 4.1:** Three-round Feistel network.

inverse queries, which makes it a *strong* pseudorandom permutation. Matching attacks were found by Patarin [Pat91].

**The quantum 3-round distinguisher.** In [KM10], Kuwakado and Morii showed that in the Q2 setting, the three-round Feistel could be distinguished from a random permutation in polynomial time. This was the first exponential quantum speedup of a cryptanalytic algorithm in symmetric cryptography.

Most polynomial-time quantum attacks on symmetric cryptographic algorithms to date follow the same design pattern as Kuwakado and Morii's distinguisher. The problem is reframed as a hidden Boolean shift problem, then solved using Simon's algorithm.

In this case, let $E$ be the primitive, which is either $E = \mathsf{Feistel}(f_0, f_1, f_2)$ or $E = \Pi$ a random permutation. Kuwakado and Morii considered the case of permutations for $f_0, f_1, f_2$ in order to have precisely Simon's promise, but we have seen how to extend it to an approximate promise (Section 1.4.3). Let $\mathsf{LFeistel}(f_0, f_1, f_2)$ be the truncation to the left branch of the three-round Feistel. They remark that:

$$\begin{aligned}\mathsf{LFeistel}(f_0, f_1, f_2)(L_0, R_0) &= R_0 \oplus f_1(L_0 \oplus f_0(R_0)) \\ &\implies \mathsf{LFeistel}(f_0, f_1, f_2)(L_0, R_0) \oplus R_0 = f_1(L_0 \oplus f_0(R_0)) \ ,\end{aligned}$$

so they take two random $\alpha_0, \alpha_1 \in \{0,1\}^n$ and define the function:

$$f(b, x) = \alpha_b \oplus E_{\mathrm{Trunc}_L}(x, \alpha_b) \ ,$$

which, when $E$ is the Feistel scheme, satisfies the hidden period equation:

$$f(b, x) = f(b \oplus 1, x \oplus f_0(\alpha_0) \oplus f_0(\alpha_1)) \ ,$$

and is random otherwise.

**Further works.** Recently, Hosoyamada and Iwata showed that the 4-round construction remained secure up to $\mathcal{O}(2^{n/12})$ Q2 forward queries, with a distinguishing attack using $\mathcal{O}(2^{n/6})$ queries [HI19a]. The proof was subsequently improved to $\mathcal{O}(2^{n/6})$ in [HI19b], making it tight. However, if inverse queries are authorized, the distinguisher based on Simon's algorithm was extended to 4 rounds by Ito et al. [Ito+19].

### 4.1.2 The Even-Mansour Cipher

The Even-Mansour construction [EM97] is a simple way to build a block cipher from a public $n$-bit permutation $\Pi$, and two $n$-bit keys, as shown in Figure 4.2:

$$E_{k_1,k_2} = \Pi(x \oplus k_1) \oplus k_2 \ .$$

Even and Mansour show that if $\Pi$ is a random permutation, the construction is classically secure up to $2^{n/2}$ queries and $2^{n/2}$ offline computations. More precisely, there exists a time-data trade-off $T \cdot D = 2^n$. Daemen [Dae91] showed a matching chosen-plaintext attack. Next, Dunkelman, Keller, and Shamir [DKS12] changed the attack setting to known-plaintext only and also obtained an attack in time $2^{n/2}$ using $\mathsf{poly}(n)$ memory.

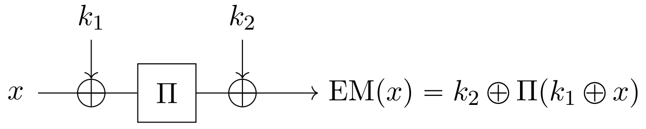


**Figure 4.2:** The Even-Mansour construction.

**Quantum attack.** In another seminal work, Kuwakado and Morii [KM12] showed how to break the Even-Mansour cipher in quantum time $\mathcal{O}(n^3)$, using $\mathcal{O}(n)$ superposition queries to EM, thanks to Simon's algorithm. The attack targets $k_1$. Once it has been obtained, the cipher becomes trivially broken. Let us define:

$$f(x) = \mathrm{EM}(x) \oplus \Pi(x) = k_2 \oplus \Pi(k_1 \oplus x) \oplus \Pi(x)$$

and observe that $\forall x, f(x \oplus k_1) = f(x)$. Thus, assuming that $f$ behaves like a random periodic function, Simon's algorithm can be applied to recover $k_1$.

### 4.1.3 Attacks on MACs

Subsequent works presented a variety of attacks with exponential speedups on MAC constructions, either by doing forgeries or recovering completely the key. In [Kap+16a], attacks are given on CBC-MAC [BR00b], PMAC [BR02], CMAC [Dwo05], GCM [MV04], OMAC [IK03], OCB [KR11]. A generalization on the CBC-MAC attack is given in [SS17]. Other works targeted the AEZ lightweight authenticated cipher [Bon17], the classically secure MAC Poly1305 [BN18].

We present the attack of [Kap+16a, Section 5] on the CBC-MAC variant of [BR00b] (Figure 4.3). This is a deterministic MAC based on a block cipher $E$ that uses two keys $k, k'$.

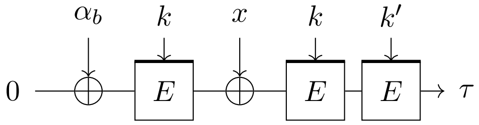


**Figure 4.3:** CBC-MAC with two blocks $\alpha_b \parallel x$ and two keys $k, k'$, producing a tag $\tau$.

The attack of [Kap+16a, Section 5] queries the MAC with two blocks, but it can accommodate any number of blocks if we simply replace the 0 in Figure 4.3 by the previous chain of inputs (this is done in [SS17]). Let $\alpha_0, \alpha_1$ be fixed values for the first message block, then:

$$\mathsf{CBC{-}MAC}(\alpha_b \parallel x) = E_{k'}\left(E_k\left(x \oplus E_k\left(\alpha_b\right)\right)\right) \ .$$

Hence, there is a hidden Boolean shift between the functions $\mathsf{CBC} - \mathsf{MAC}(\alpha_0 \parallel \cdot)$ and $\mathsf{CBC} - \mathsf{MAC}(\alpha_1 \parallel \cdot)$, which is $E_k\left(\alpha_0\right) \oplus E_k\left(\alpha_1\right)$. We obtain this value using Simon's algorithm.

Next, we remark that for any message block $m$, the tags of $\alpha_0 \parallel m$ and of $\alpha_1 \parallel m \oplus E_k\left(\alpha_0\right) \oplus E_k\left(\alpha_1\right)$ have the same values. Thus, we can build a forgery.

### 4.1.4 Quantum Slide Attacks

Classical *slide attacks* [BW99] draw their power from structural self-similarities in ciphers. Let us take the simple example of a *one-round self-similar* cipher $E_k : \{0,1\}^n \to \{0,1\}^n$, built by iterating a round function $F(x,k) : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ for a total of $r$ rounds, using the same round key $k$. This structure entails a very simple equality, the *slide property*:

$$E_k(F(x,k)) = F(E_k(x),k) \ .$$

**Basic slide attack.** The goal of the attacker is to find two pairs $x, y$ satisfying $F(x,k) = y$. By the *birthday paradox*, among $\mathcal{O}\left(2^{n/2}\right)$ plaintext-ciphertext couples

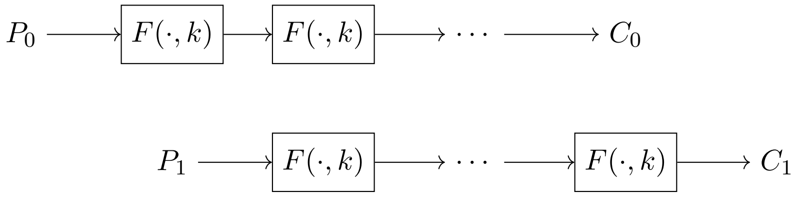


**Figure 4.4:** Illustration of the basic slide property.

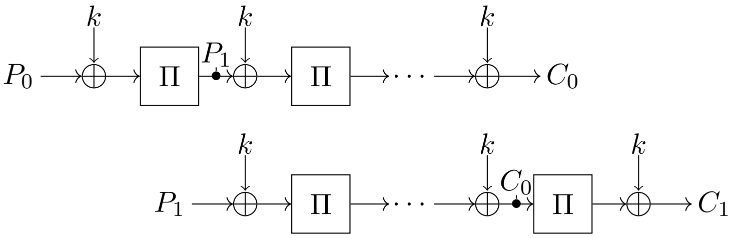


**Figure 4.5:** Slide attack on the key-alternating cipher.

$P, C$, there exists a *slide pair*: $P_0, C_0$ and $P_1, C_1$ such that $F(P_0, k) = P_1$. In that case, we also have: $F(C_0, k) = C_1$. If $F$ is sufficiently *weak*, then $k$ can be retrieved from these two equations. Hence, the simplest attack setting consists in performing $\mathcal{O}(2^{n/2})$ queries, then checking for each pair $P_0, C_0$ and $P_1, C_1$ if it is a slide pair; and in that case returning the key $k$.

**Quantum version.** In [Kap+16a], Kaplan, Leurent, Leverrier, and Naya-Plasencia showed that, in some cases, the slide property can be turned into a promise problem, and this problem can be solved with Simon's algorithm. We will just present the attack of [Kap+16a] on the key-alternating cipher of Figure 4.5, which is an iterated Even-Mansour construction using a permutation $\Pi$ and a single key $k$.

In this example, slide pairs have the property: $P_1 = \Pi(k \oplus P_0)$, or equivalently, $C_1 = k \oplus \Pi(C_0)$. Thus, $C_1 \oplus \Pi^{-1}(P_1) = P_0 \oplus \Pi(C_0)$ allows to detect a slide pair, and $k$ can be recovered in time $\mathcal{O}(2^{n/2})$ and $\mathsf{poly}(n)$ memory, using Pollard's rho method and Algorithm 2.9.

Kaplan, Leurent, Leverrier, and Naya-Plasencia observe that the self-similarity can be rewritten as:

$$\forall x, \Pi(E_k(x)) \oplus k = E_k(\Pi(x \oplus k)) \quad .$$

Thus, if we define the two random functions $f(x) = \Pi(E_k(x)) \oplus x$ and $g(x) = E_k(\Pi(x)) \oplus x$, we have $f(x) = g(x \oplus k)$. With Simon's algorithm, $k$ can be recovered in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n)$ Q2 queries. Various other attacks have appeared in subsequent works (see [BNS19a] for an exhaustive list). Most of them consist in rewriting the classical slide property as a *slide-shift equation*.

### 4.1.5 The Grover-meets-Simon Attack

The primary target of the *Grover-meets-Simon* algorithm [LM17] is the FX construction, which consists in transforming a block cipher $E_k$ with *whitening* keys $k_{in}$ and $k_{out}$ (Figure 4.6). This construction is used to increase the key length of the cipher. Let $n$ be the block size and $m$ be the length of $k$. Since the cipher can be seen as an Even-Mansour with a keyed permutation instead of the keyless $\Pi$, the classical attacks are similar [Din15]. In particular, one obtains a time-data trade-off $T \cdot D = 2^{m+n}$, and

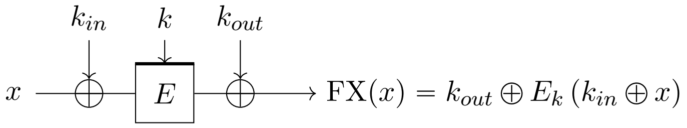


**Figure 4.6:** The FX construction. A thick line represents the input of the key.

the construction is proven to be secure up to $\mathcal{O}\big(2^{(n+m)/2}\big)$ queries and computations if the underlying block cipher is secure [KR96].

The FX design has been used for many block ciphers, such as DESX, proposed by Rivest in 1984 and analyzed in [KR96], PRINCE [Bor+12], and PRIDE [Alb+14].

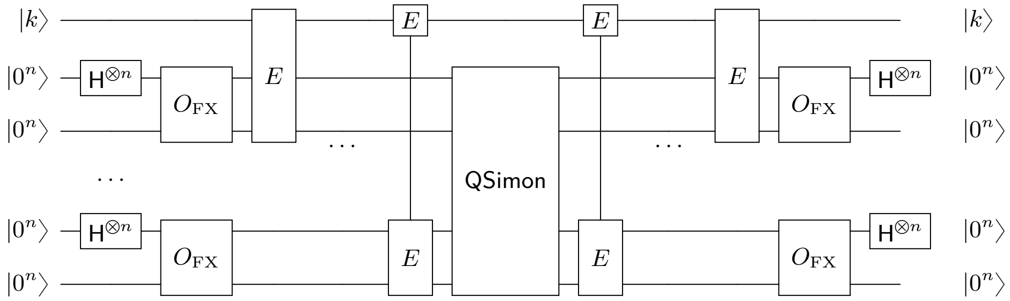
If $k$ was known, Kuwakado and Morii's attack would recover $k_{in}$ and break the cipher. Leander and May [LM17] proposed to use a Grover search for $k$. In order to test if a guess of $k$ is the good one, they run Simon's algorithm with the function:

$$f_z(x) = \mathrm{FX}(x) \oplus E_z(x) \quad ,$$

which is periodic of period $k_{in}$ if and only if $z = k$, and random otherwise. Thus, they solve Problem 4.1.

**Problem 4.1** (Finding a periodic function)**.** *Let $f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^\ell$ be a family of functions indexed by $\{0,1\}^m$ and write $f(i,\cdot) = f_i(\cdot)$. Assume that there exists exactly one $i_0 \in \{0,1\}^m$ such that $f_{i_0}$ is a random periodic function, and that all others are random functions. Then find $i_0$.*

It is easy to implement an oracle (Circuit 4.1) that flips the phase of the good key guess only, up to some error, using the quantum circuit for Simon's algorithm QSimon (Circuit 1.9), $\mathcal{O}(n)$ queries to $O_{\mathrm{FX}}$ and $\mathcal{O}(n)$ circuits $E$ computing the unitary $|z\rangle |x\rangle |y\rangle \xmapsto{E} |z\rangle |x\rangle |y \oplus E_z(x)\rangle$. Thus, the keys can be recovered in time $\mathcal{O}\big(n^3 2^{m/2}\big)$, using $\mathcal{O}\big(n 2^{m/2}\big)$ queries to $O_{\mathrm{FX}}$, assuming that $m = \mathcal{O}(n)$.



**Quantum Circuit 4.1:** Quantum test for the Grover-meets-Simon algorithm applied to FX.

**Theorem 4.1** (Grover-meets-Simon (adapted))**.** *Consider the setting of Problem 4.1, and assume that the family $(f_i)$ satisfies the condition:*

$$\exists a > 0, \max_{\substack{i \in \{0,1\}^m \setminus \{i_0\} \\ t \in \{0,1\}^n \setminus \{0^n\}}} \Pr_{x \leftarrow \{0,1\}^n} [f_i(x \oplus t) = f_i(x)] \leqslant \frac{an}{2^n} \quad . \tag{4.1}$$

*Then there exists a quantum algorithm that finds $i_0$ in time $\mathcal{O}\big((n+m)^3 2^{m/2}\big)$, with constant probability. It makes $\mathcal{O}\big((n+m)2^{m/2}\big)$ queries to $F$.*

*Proof.* The condition (4.1) is a generalization of the definition of $\epsilon(f, s)$ in Equation (1.1). In short, all functions $f_i$ must be sufficiently far from periodic, so that the probability of raising false positive remains low. By Equation (1.2), this is true if the functions $f_i$ are all independent random functions.

By condition (4.1) and Theorem 1.4, the circuit QSimon adds an error vector of amplitude less than $e2^{-r/2+1}$ if we use $n + r$ queries.

Thus, QSimon qualifies as an approximate test by Definition 2.2, and furthermore, it does not yield false negatives. We can use an Amplitude Amplification with approximate test, where the test consists in making the appropriate queries and running QSimon. By Theorem 2.6, it suffices to have an error smaller than $2^{-m/2}$ in the circuit to ensure a constant success probability, so $r = m + 3$ is enough (and $r = 2m$ will ensure a probability of success exponentially close to 1). □

This attack is particularly relevant in the context of post-quantum symmetric cryptography, where we look for generic ways of extending the key length of block ciphers (in order to counter Grover's exhaustive search). In the Q2 setting, wrapping the cipher with whitening keys brings practically no advantage.

### 4.1.6 Attacks with Modular Additions

In all the attacks above, the primitive has a structure with bitwise XORs $\oplus$. Alagic and Russell [AR17] proposed to replace $\oplus$ by another group operation, to thwart Simon's algorithm and recover the lost security. Their most practical proposal, already in use in many designs, is the modular addition $+$. In that case, the Boolean hidden shift problem becomes an *Abelian* hidden shift in the group $\mathbb{Z}_{2^n}$.

**Problem 4.2** (Abelian Hidden shift problem)**.** *Let $(\mathbb{G}, +)$ an Abelian group, $g_0, g_1 : \mathbb{G} \to \mathbb{G}$ two permutations such that there exists $s \in \mathbb{G}$ such that, for all $x$, $g_0(x) = g_1(x + s)$. Find $s$.*

No quantum polynomial-time algorithm for Problem 4.2 is known. However, Kuperberg [Kup05; Kup13] gave an algorithm in time $2^{\mathcal{O}(\sqrt{n})}$, of which many variants have been studied, both asymptotically [Reg04; CJS14] and non-asymptotically [BN18; BS20].

Notably, Bonnetain and Naya-Plasencia computed explicit costs for Kuperberg's algorithm and found $2^{\sqrt{2\log_2(3)n}} \simeq 2^{1.8\sqrt{n}}$ for the group $\mathbb{Z}_{2^n}$. Thus, if one tries to retain significant security levels in face of such an attack, the state size $n$ must be increased to a level that goes far outside the standards of symmetric cryptography. In general, replacing $\oplus$ by $+$ in insecure constructions often also produces an attack, though its cost may increase by a significant factor (polynomial to subexponential). Kuperberg's algorithm can be also be written as a quantum circuit without measurements, although with a herd of technical difficulties.

Some examples of attacks that don't work anymore with modular additions can be found in [BNS19a]. These are slide attacks that use the involutory nature of the bitwise XOR.

### 4.1.7 Distinguisher on the One-Time Pad

The distinguisher on the *one-time pad* is a folklore result that shows the power of quantum adversaries and, at the same time, the difficulty in assessing the meaningfulness of a quantum attack. Let us consider an encryption scheme of the form: $E_k(N, m) = m \oplus f(k, N)$ where $f$ is any function and $N$ is a nonce that changes at each encryption query. Then it is possible to distinguish $E_k(N, m)$ from a random permutation $\Pi$ with overwhelming probability, using a single superposition query to $E_k$ for a randomly chosen nonce:

$$|m\rangle |y\rangle \xmapsto{O_{E_k(N,\cdot)}} |m\rangle |y \oplus m \oplus f(k, N)\rangle \enspace .$$

In particular, if $f$ is a random oracle, i.e., if $f(k, N)$ is randomly chosen, this construction is the well-known *one-time pad*[1]. It was proven by Shannon [Sha49] to achieve perfect secrecy, that is, the message cannot be recovered from the sole knowledge of the ciphertext. There exists no classical distinguisher between the one-time pad and true randomness.

**Proposition 4.1** (Quantum distinguisher on the One-time Pad, informal)**.** *Assume that for some function $f$, e.g., a random oracle, $E_k(N, m) = m \oplus f(k, N)$. Then there exists a quantum algorithm that, given a standard oracle $O$ that is $O_{E_k(N,\cdot)}$ or $O_\Pi$, makes a single query and distinguishes $O_{E_k(N,\cdot)}$ from $O_\Pi$, where $\Pi$ is a random permutation, with overwhelming success.*

*Proof.* The distinguisher (Algorithm 4.1) uses the Deutsch-Jozsa algorithm [DJ92] which, in a single query, is capable of distinguishing a random function from a constant one. For a mode of the given form, $E_k(m) \oplus m = f(k, N)$ is a constant function of $m$, while $\Pi(m) \oplus m$ is random. Thus, in the former case, at Step 4 in Algorithm 4.1, the state obtained is $\sum_m |m\rangle |f(k, N)\rangle$ and the Hadamard transform turns it to $|0\rangle$, with a probability 1 of measuring 0. In the latter, the probability to measure 0 is:

$$\sum_a \left( \frac{1}{2^n} \sum_{m, m \oplus \Pi(m) = a} (-1)^{m \cdot 0} \right)^2 = \sum_a \left( \frac{|\{m, m \oplus \Pi(m) = a\}|}{2^n} \right)^2$$

[1] In french, *Le masque jetable* (the disposable mask), a name that was ahead of its time.

which depends on the distribution of the number of preimages of $\Pi(m) \oplus m$, but is negligible in general for a random permutation. □

**Algorithm 4.1** Distinguisher on the one-time pad based on Deutsch-Jozsa's algorithm.

```
Input: oracle access to $O_g$
Output: 1 if $g = \mathsf{Encrypt}_k$ for some $k$ and 0 if $g = F$.
Initialize a register for $n$-bit messages and another for the outputs
        ▷ $|0^n\rangle |0^n\rangle$
Apply a Hadamard transform $\mathsf{H}^{\otimes n}$ and CNOTs
        ▷ $\sum_{m \in \{0,1\}^n} |m\rangle |m\rangle$
Apply $O_g$
        ▷ $\sum_{m \in \{0,1\}^n} |m\rangle |g(m) \oplus m\rangle$
Apply a Hadamard transform on the first register to produce:
        ▷ $\sum_{y \in \{0,1\}^n} \sum_{m \in \{0,1\}^n} (-1)^{y \cdot m} |y\rangle |g(m) \oplus m\rangle$
Measure the first register
if The result is 0 then
    return 1
else
    return 0
```

Classically, such a distinguisher would need more than one query to obtain a significant success probability, but this is prevented by the change of nonces: in the case of the one-time pad, $f$ is a random oracle and a new, entirely independent keystream is produced at each query. However, the quantum attack uses the same nonce for all messages in the superposition. Thus it can be seen as a kind of *nonce misuse*.

It seems difficult to assert whether this distinguisher is a meaningful attack. On the one hand, it is non-trivial, and may be countered by other modes of encryption. On the other hand, it does not reveal any information on the key, and does not help to distinguish between *classical* messages, as we will see in Chapter 10, Section 10.3.1. This contrasts with all key-recovery or forgery attacks of this section, which result in a classical break.

## 4.2 Q1 Attacks on Symmetric Constructions

From the different attacks detailed in Section 4.1, we can infer a prevailing design pattern: these attacks exploit the algebraic structure of the primitive and use a quantum hidden shift algorithm to extract a secret from the oracle. We now turn ourselves towards Q1 attacks. For now, most of them reproduce a classical setup and use quantum search to obtain *at best* a quadratic acceleration. The first example of Q1 attacks reaching more rounds than the classical attacks appeared in the work of Hosoyamada and Sasaki [HS20] (Section 4.2.2).

### 4.2.1 Q1 Attack on Even-Mansour

Kuwakado and Morii [KM12] give an attack in the Q1 setting, that is analogous to the classical time-data trade-off [Dae91].

**Classical trade-off curve.** Let us define:

$$f(y) = \Pi(y) \oplus \Pi(y \oplus 1) \ , \qquad g(x) = \mathrm{EM}(x) \oplus \mathrm{EM}(x \oplus 1) \ ,$$

then $\forall x, f(x \oplus k_1) = g(x)$. By finding a claw between $f$ and $g$, we recover $k_1$ with high probability. The classical and quantum attacks (Algorithms 4.2 and 4.3) solve this problem in analogous ways.

**Algorithm 4.2** Classical trade-off on Even-Mansour.

```
Input: query access to $g$
            (limited to $D$ queries)
Output: $k_1$
            (with constant probability)
1: Query $D$ values $x, g(x)$ and store them in a list $\mathcal{L}$
2: repeat
3:     Sample $y \xleftarrow{\$} \{0,1\}^n$
4: until $\exists x, (x, f(y)) \in \mathcal{L}$
5: return $y \oplus x$
```

Steps 2-4 cost $T = \frac{2^n}{D}$, hence up to $D = 2^{n/2}$, the trade-off curve is:

$$T \cdot D = 2^n \ .$$

**Algorithm 4.3** Q1 trade-off on Even-Mansour.

```
Input: query access to $g$
            (limited to $D$ queries)
Output: $k_1$
            (with constant probability)
1: Query $D$ values $x, g(x)$ and store them in a list $\mathcal{L}$
2: Find $y \in \{0,1\}^n$ such that:
            $\exists x, (x, f(y)) \in \mathcal{L}$
            using quantum search
3: return $y \oplus x$
```

Step 2 costs $T = \sqrt{\frac{2^n}{D}}$, hence up to $D = 2^{n/3}$, the trade-off curve is:

$$T^2 \cdot D = 2^n \ .$$

Notice that, since there is no quantum Pollard's rho method, the balanced point $T = D = 2^{n/3}$ still requires $2^{n/3}$, memory, whereas classically it needs $\mathsf{poly}(n)$ only. The memory used here is QRACM.

**Trade-off without QRACM.** Hosoyamada and Sasaki [HS18a] proposed to replace QRACM by CSAM, using the multi-target preimage search algorithm that will be presented in Section 5.1. The trade-off curve becomes $D \cdot T^6 = 2^{3n}$, with $D^{1/3}$ memory, and the balanced point is $D = T = 2^{3n/7}$. Similar results hold for the FX construction.

### 4.2.2 Quantum Collision Attacks more Efficient than Classical

Hosoyamada and Sasaki [HS20] observed that, since the quantum birthday bound on hash functions dropped from a generic $\mathcal{O}\big(2^{n/2}\big)$ to a generic $\mathcal{O}\big(2^{n/3}\big)$, hence less than a quadratic speedup, some procedures less efficient than classical collision search may qualify as attacks in the quantum setting, if they benefit from a sufficient quantum speedup. For example, any **Sample** procedure benefits from a quadratic speedup, as we saw in Section 2.2.

This enabled Hosoyamada and Sasaki to show that some hash functions (AES-MMO and Whirlpool) had a smaller quantum security margin than classical. We will draw a similar result in Chapter 8 in collision attacks against GIMLI-HASH.

### 4.2.3 Other Methods

Quantum cryptanalysis is a rapidly evolving field and new methods are still appearing and being investigated. An interesting example to mention is the algorithm for solving Boolean equations of Chen and Gao [CG18]. This algorithm is based on the HHL algorithm for linear systems [HHL09], which has an exponential speedup. The authors propose to set up quantum algebraic attacks in which the problem of recovering the key of a block cipher, or finding a preimage of a hash function, is simply translated into a system of Boolean equations, which is solved using their new algorithm. At the time of writing, this attack is the only candidate for a more than quadratic speedup *in the Q1 setting*. However, its status remains unclear, because its complexity depends on a parameter (the condition number of a large sparse linear system) which has not yet been duly estimated.

# Chapter 5
# Quantum Merging Algorithms for the k-XOR Problem

This chapter and the next one focus on *generic* quantum algorithms that solve the $k$-XOR problem. It can be seen as a generalization of the collision search problem introduced in Section 2.3, where, instead of two colliding elements, we wish to find a $k$-tuple for some given $k$. Furthermore, there may be many solutions as in Section 2.3, but there may also be only a single expected one, which will be the case in Chapter 6. This versatile problem encompasses a variety of problems arising in symmetric and asymmetric cryptography. Some of these applications will be reviewed both in this chapter and Chapter 6.

This chapter merges results published in [CNS17; GNS18; NS20] and adds new details. We introduce a new quantum algorithm for collision search and its applications. Next, we review the $k$-XOR problem, classical methods to solve it and introduce the framework of *quantum merging* and of *merging trees*. We describe the optimal merging strategies for quantum $k$-XOR *with many solutions* and prove this optimality among the set of merging trees.

## Contents

## 5.1 Quantum Collision and Multi-Target Preimages without QRACM

Let us restate the *collision search problem*:

**Problem 2.3** (collision search). *Given access to* $h : \{0,1\}^n \to \{0,1\}^n$*, find a* collision *pair:* $x \neq y$ *such that* $h(x) = h(y)$*.*

We have seen in Section 2.3 that Pollard's rho algorithm solves this problem for a random function in time and queries $\mathcal{O}\big(2^{n/2}\big)$, its lower bound, using a negligible memory, and that Brassard, Høyer and Tapp's algorithm (Algorithm 2.10) solved it in time and queries $\mathcal{O}\big(2^{n/3}\big)$, also the quantum lower bound. But the BHT algorithm uses a QRACM of exponential size.

In this section, we study quantum speedups in the plain quantum circuit model. We show that it is possible to obtain a quantum gate count below $\mathcal{O}\big(2^{n/2}\big)$.

### 5.1.1 Quantum Collision Search

The algorithm presented here (Algorithm 5.1) relies on the definition of a set of *distinguished points*, an idea that is reminiscent of the parallel collision search technique of [vOW99].

**Definition 5.1** (Distinguished points)**.** Take an arbitrary prefix $u$ of $2\log_2 L$ bits. We say that $x \in \{0,1\}^n$ is *distinguished* if $h(x)$ has prefix $u$: $h(x) = u|*$. Let $S_u \subseteq \{0,1\}^n$ be the set of distinguished points.

With this definition, there are on average $\frac{2^n}{L^2}$ distinguished points. Since $h$ is simply assumed to be a random function, the precise number of distinguished points is not known. However, its deviation from the average can be quantified with Chernoff-Hoeffding bounds [Hoe94]. Let $X_i$, $0 \leqslant i \leqslant 2^n$ be the independent random variables taking the value 1 if $i$ is distinguished and 0 otherwise, then for any constant $0 \leqslant \delta \leqslant 1$:

$$\Pr\left(\left|\sum_i X_i - \frac{2^n}{L^2}\right| \geqslant \delta \frac{2^n}{L^2}\right) \leqslant 2e^{-\frac{\delta^2 2^n}{3L^2}} \quad . \tag{5.1}$$

The exact value of the prefix does not matter, only its size and the fact that two distinguished points have a higher probability of forming a collision pair than two random points.

Then, the algorithm is divided into two main steps, which adapt the two steps of BHT collision search: **1.** create a list of values $(y, h(y))$ for many $y$, where $h(y) \in S_u$; **2.** search for $x \in \{0,1\}^n$ such that $\exists y \neq x, (y, h(x)) \in \mathcal{L}$.

**Algorithm 5.1** Quantum collision search without QRACM.

```
Input:  query access to $O_h$, memory bound $L \leqslant 2^{n/5}$
Output:  a collision of $h$

Initialize an empty list $\mathcal{L}$ of size $L$
Select a prefix $u$ of length $2 \log_2 L$
Repeat $L$ times
    Sample $y \in \{0,1\}^n$ such that
        $y \in S_u$ ($h(y)$ has the right prefix)
    EndSample
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{(y, h(y))\}$
EndRepeat
If $\mathcal{L}$ contains a collision, return it
Sample $x \in S_u$ such that
    $\exists y \neq x, (y, h(x)) \in \mathcal{L}$
EndSample
return $x, y$
```

We now prove its correctness and time complexity.

**Theorem 5.1** (QRACM-free quantum collision search [CNS17]). *For any memory limit $L \leqslant 2^{n/5}$, there exists a quantum algorithm solving the collision search problem for random functions, running in time $\mathcal{O}\left(L^2 + \frac{2^{n/2}}{\sqrt{L}}\right)$ (counted in $n$-bit register operations and evaluations of $h$), making $\mathcal{O}\left(L^2 + \frac{2^{n/2}}{\sqrt{L}}\right)$ queries to $O_h$.*

*Proof.* The average time complexity in Algorithm 5.1 is easy to compute. In the first step, there are $\frac{2^n}{L^2}$ distinguished points, hence $\mathcal{O}(L)$ iterations to find one, and we repeat this $\mathcal{O}(L)$ times. The final Amplitude Amplification has (on average) $L$ solutions among $|S_u|$, hence a quantum time complexity of order:

$$\sqrt{\frac{|S_u|}{L}}\left(\underbrace{L}_{\text{Sample}} + \underbrace{L}_{\text{Test}}\right) ,$$

where the sample consists in running Grover's algorithm again, and the test contains a sequential membership circuit. The choice of distinguished points ensures that the time complexities of the sample and the test are balanced.

We now show that with probability exponentially close to 1, we stay close to the average. In the first step, it suffices to set the number of iterations of Grover search to $\left\lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{L^2}} \right\rfloor$, depending on the *expected* number of solutions. Taking $\delta = \left(\frac{L^2}{2^n}\right)^{1/3}$ in Equation (5.1), we obtain that with overwhelming probability, the number of distinguished points is between: $\frac{2^n}{L^2} - \left(\frac{2^n}{L^2}\right)^{2/3}$ and $\frac{2^n}{L^2} + \left(\frac{2^n}{L^2}\right)^{2/3}$. So the probability of success of the classical sampling (taking a point at random and checking if it is distinguished) lies in the interval:

$$\frac{\frac{2^n}{L^2} - \left(\frac{2^n}{L^2}\right)^{2/3}}{2^n} \leqslant \frac{|S_u|}{2^n} \leqslant \frac{\frac{2^n}{L^2} + \left(\frac{2^n}{L^2}\right)^{2/3}}{2^n}$$

$$\implies \frac{1}{L^2}\left(1 - \frac{L^{2/3}}{2^{n/3}}\right) \leqslant \frac{|S_u|}{2^n} \leqslant \frac{1}{L^2}\left(1 + \frac{L^{2/3}}{2^{n/3}}\right) .$$

Thus, the relative error vanishes and by Theorem 2.5, using Exact amplitude amplification, we can sample from $S_u$ with error probability at most:

$$\left(\frac{L^{2/3}}{2^{n/3}}\right)^2 = \frac{L^{4/3}}{2^{2n/3}} , \tag{5.2}$$

so only $L(1 + o(1))$ samples are necessary to obtain $L$ different results with high probability.

In the second step, the list $\mathcal{L}$ contains exactly $L$ fixed distinct elements $x_0, \dots, x_{L-1}$ and $L$ distinct images $h(x_0), \dots, h(x_{L-1})$. There are on average $(2^n - L)\frac{L}{2^n} = L\left(1 - \frac{L}{2^n}\right)$ elements $x \in \{0,1\}^n \backslash \{x_0, \dots, x_{L-1}\}$ such that $h(x) \in \{h(x_0), \dots, h(x_{L-1})\}$. Using the same inequality as (5.1), we can bound the probability $p$ that a random distinguished point is a solution:

$$\frac{L\left(1 - \frac{L}{2^n}\right) - L^{2/3}\left(1 - \frac{L}{2^n}\right)^{2/3}}{|S_u|} \leqslant p \leqslant \frac{L\left(1 - \frac{L}{2^n}\right) + L^{2/3}\left(1 - \frac{L}{2^n}\right)^{2/3}}{|S_u|}$$

$$\implies \frac{L^2}{2^n}\left(1 - \frac{L^{2/3}}{2^{n/3}}\right)(L - L^{2/3}) \leqslant p \leqslant \frac{L^2}{2^n}\left(1 + \frac{L^{2/3}}{2^{n/3}}\right)(L + L^{2/3}) \tag{5.3}$$

(with an approximation: $\frac{L}{2^n} \ll \frac{L^{1/3}}{2^{n/3}}$). We combine this with Equation (5.2) to bound the total error probability. The uncertainties from the sample and the number of solutions pile up additively. Applying Theorem 2.5 again, we find that the probability of success of this final Amplitude Amplification is exponentially close to 1. □

### 5.1.2 Quantum Multi-Target Preimage Search

Algorithm 5.1 applies as well to multi-target preimage search.

**Problem 2.7** (Multi-target Preimage Search). *Given access to $h$, and a list of targets $\mathcal{L} = \{y_0, \dots, y_{L-1}\}$ (selected uniformly at random), find $x \in \{0,1\}^n$ such that $\exists 0 \leqslant i \leqslant L-1$, $h(x) = y_i$.*

The idea of Algorithm 5.2 below is to go back to the setting of Algorithm 5.1, with a search for a collision *among distinguished points*. Since the targets are random, we need to drop most of them and keep only the distinguished ones. Thus, the solution found will actually be a preimage of a distinguished target.

**Algorithm 5.2** Quantum multi-target preimage search without QRACM.

```
Input:  query access to $O_h$, list $\mathcal{L} = \{y_0, \dots, y_{L-1}\}$ of $L$ targets
Output:  a preimage of one of these targets: $x$ such that $\exists y \in \mathcal{L}, h(x) = y$
Initialize an empty list $\mathcal{L}'$ of size $\min(L^{1/3}, 2^{n/7})$
Select a prefix $u$ of length $\frac{2}{3}\log_2 L$
for $y_i \in \mathcal{L}$ do
    if $y_i$ has the prefix $u$ then
        Add $y_i$ to $\mathcal{L}'$
If $\mathcal{L}'$ contains a collision, return it
Sample $x \in S_u$ such that
    $\exists y \neq x, (y, h(x)) \in \mathcal{L}'$
EndSample
return $x, y$
```

**Theorem 5.2** (QRACM-free quantum multi-target preimage search [CNS17])**.** *There exists a quantum algorithm solving the multi-target preimage search problem for random functions, for $L$ targets, running in time $\mathcal{O}\left(\min\left(L + \frac{2^{n/2}}{L^{1/6}}, 2^{3n/7}\right)\right)$ (counted in $n$-bit register operations and evaluations of $h$), making the same number of queries to $O_h$, and using $L^{1/3}$ memory.*

*Proof.* The runtime of Algorithm 5.2 is very similar to what we computed for Algorithm 5.1. In the first step, we only read the list of targets and stockpile the distinguished ones. In the second step, we search in the space of distinguished points, of size $\frac{2^n}{L^{2/3}}$, for one among $L^{1/3}$ solutions.

Our choice of prefix size ensures that the sampling and test in the second step are balanced. For $L \leqslant 2^{3n/7}$:

$$L + \underbrace{\sqrt{\frac{|S_u|}{L^{1/3}}}}_{\text{Iterations}} \Bigg( \underbrace{L^{1/3}}_{\text{Sample}} + \underbrace{L^{1/3}}_{\text{Test}} \Bigg) = L + \frac{2^{n/2}}{L^{1/6}} \ . \tag{5.4}$$

Even if $\mathcal{L}$ is not stored, we assume that reading from it costs time 1. The trade-off remains valid only until both terms are balanced, which happens at $L = 2^{3n/7}$. Thus, with an unlimited number of targets, the time complexity can decrease down to $\mathcal{O}(2^{3n/7})$. □

### 5.1.3 Discussion

Algorithm 5.1 shows that QRACM is not necessary in quantum collision search, *up to the memory bound* $2^{n/5}$. While the time-memory trade-off curve remains $T^2 \cdot M = 2^n$, a large part of this curve becomes achievable with classical memory only (and more precisely CSAM, as we have seen in Section 1.3.4).

Three natural questions come to mind.

**Open Problem 5.1.** *Is it possible to move further on this curve, for example to bring the time complexity down to $\mathcal{O}\left(2^{n/3}\right)$, using $\mathcal{O}\left(2^{n/3}\right)$ classical memory?*

We conjecture that this is unlikely, as a quantum circuit requiring $\mathcal{O}\left(2^{n/3}\right)$ time and classical memory would be accessing each classical cell at most $\mathcal{O}(1)$ times.

**Open Problem 5.2.** *Is it possible to move* below *the curve $T = \sqrt{\frac{2^n}{M}}$?*

We conjecture that this is unlikely.

**Open Problem 5.3.** *Does there exist a quantum algorithm for collision search with $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$ time (but using any amount of qubits), in the plain quantum circuit model?*

Leaving aside Pollard's rho, classical collision search can be performed with $2^{n/2}$ memory and $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ time, *without random access*, by simply storing the $2^{n/2}$ elements and performing a sorting network (e.g., the AKS network [AKS83]). This question asks whether a corresponding quantum method exist, that would then be free of qRAM gates.

**Parallelization.** A distributed variant of Algorithm 5.1 was done in [CNS17]. This algorithm was improved subsequently in [JS20]. The main idea is that the first step can be easily distributed, since the distinguished points can be searched for by independent processors. Then, the second step can be distributed as a Grover search [CNS17], or using more complex unitary operators to accelerate the membership circuit [JS20]. The efficiency of the second variant depends on the connectivity of the quantum processors [Bea+13].

**Further works.** The idea to replace quantum by classical memory was subsequently reused by Helm and May [HM20] in subset-sum algorithms. However, the restriction on the time-memory trade-offs available is quite strong, and their results do not outperform the best classical algorithms for this problem from the single metric of time complexity. Instead, they achieve better time-memory trade-offs quantumly than classically. Algorithm 5.2 was used in the attacks against Even-Mansour and FX of [HS18a].

## 5.2 Classical Merging Algorithms

In this section, we define the $k$-XOR Problem *with many solutions* and detail Wagner's algorithm [Wag02], that efficiently solves it.

### 5.2.1 k-XOR Problems

The *Generalized Birthday Problem* was introduced by Wagner [Wag02], who solved it by generalizing a method credited to Camion and Patarin [CP91]. It has many applications in cryptography.

We will consider several variants of this problem. The input data can be accessed via *input lists* or via an *oracle*. Classically, this does not make a significant difference. Quantumly, we need to take it into account, as it determines whether the data is accessed by a quantum or a classical oracle.

**Problem 5.1** ($k$-XOR with lists)**.** *Given $\mathcal{L}_1, \ldots, \mathcal{L}_k$ lists of uniformly random $n$-bit strings, find $x_1, \ldots, x_k \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_k$ such that $x_1 \oplus \ldots \oplus x_k = 0$ in minimal time.*

Problem 5.1 is the original problem solved by Wagner in [Wag02], where the lists can be extended at will. We are in a situation where *many solutions exist* and we want to find only one. In that case, we will see that there exists an optimal list size, which is exponential in $n$ (otherwise we wouldn't expect a solution) and the same for all lists (otherwise we could increase the size of the non-maximal lists and simply drop the additional elements). The *oracle* version of this problem is as follows.

**Problem 5.2** ($k$-XOR with an oracle)**.** *Given oracle access to a random $n$-bit to $n$-bit function $h$, find distinct inputs $x_1, \ldots, x_k$ such that $h(x_1) \oplus \ldots \oplus h(x_k) = 0$.*

When $k = 2$, Problem 5.2 collapses to the collision search problem. Alternatively, one can define Problem 5.2 with $k$ different random functions, or Problem 5.1 with a single input list. For random objects, these formulations are equivalent up to a constant factor in $k$.

**Extension to other operations.** We choose to focus on the bitwise XOR operation $\oplus$ for simplicity, but, as Wagner already remarks [Wag02], the problem can be defined with modular additions instead ($k$-SUM). All the algorithms of this chapter and the next one can also be applied in the $k$-SUM setting, which is also of cryptographic interest.

**Query complexity of k-XOR.** In all variants, we consider $k$ to be a constant. Intuitively, increasing $k$ can only make the problem easier on average, since new degrees of freedom are available. The following folklore result is well-known.

**Proposition 5.1.** *The classical query complexity of $k$-XOR (Problem 5.1 or Problem 5.2) is $\Omega(2^{n/k})$.*

*Proof.* We make $k2^{n/k}$ queries to generate $2^n$ random independent sums $x_1 \oplus \ldots \oplus x_k$. Then the probability that none of them is 0 is less than $(1 - 2^{-n})^{2^n} \leqslant e^{-1}$. □

In the quantum setting, Zhandry proved a similar result, that includes the quantum collision lower bound as a special case.

**Proposition 5.2** ([Zha19], Corollary 3)**.** *Any algorithm that makes $q$ quantum queries to $h$ can only output a $k$-tuple $x_1, \dots, x_k$ such that $h(x_1) \oplus \dots \oplus h(x_k) = 0$ with probability $\mathcal{O}\big(q^{k+1}/2^n\big)$. Consequently, any algorithm succeeding with constant probability must make at least $\Omega\left(2^{n/(k+1)}\right)$ such queries.*

Thus, algorithms for $k$-XOR have exponential query and time complexities in $n$: $\widetilde{\mathcal{O}}(2^{\alpha_k n})$ for some $\alpha_k$ depending *only* on $k$. We will be focused on the algorithms achieving the best time complexities. Despite the favorable case of collisions, they generally require memories of exponential size.

### 5.2.2 Classical Merging

We will now detail the fundamental building block of Wagner's algorithm, and the whole class of *merging algorithms* that we explore in this chapter and the next one.

From now on, we adopt the following conventions: lists named $\mathcal{L}_i$ have corresponding sizes $L_i = 2^{\ell_i n}$ (up to a constant). We write for simplicity that $\mathcal{L}_i$ "has size $\ell_i$". All these $\ell_i$ are constants. Since all time complexities are exponential in $n$, we write them as $\mathcal{O}(2^{\alpha n})$ and focus on the exponent $\alpha$, which is also a constant. In other words, we often remove $n$ from our computations as a mere common factor.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two lists of $n$-bit strings selected uniformly and independently at random, of respective sizes $L_1 \simeq 2^{\ell_1 n}$ and $L_2 \simeq 2^{\ell_2 n}$. We need the lists to be sorted to ensure the efficiency of the procedure. We select a prefix $t$ of $un$ bits ($u < 1$), where $un$ is approximated to an integer.

**Definition 5.2** (Join operator)**.** The "join" $\mathcal{L}_1 \bowtie_u \mathcal{L}_2$ is the list of pairs $x_1 \in \mathcal{L}_1, x_2 \in \mathcal{L}_2$ such that $x_1 \oplus x_2 = t|*$. We say that such $x_1$ and $x_2$ *partially collide* on $un$ bits.

In the following, we will favor the term "merging" over "join". Instead of *computing the join*, we say that we *merge $\mathcal{L}_1$ and $\mathcal{L}_2$ on un bits.* By construction, if both input lists are sorted, the join list will be as well.

*Remark* 5.1. Computationally, the join list will contain rather three-tuples $x_1, x_2, x_1 \oplus x_2$, so that we keep the knowledge of $x_1$ and $x_2$ and do not recompute the sum.

If the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ contain elements drawn uniformly at random, $\mathcal{L}_u = \mathcal{L}_1 \bowtie_u \mathcal{L}_2$ is of average size $\frac{L_1 L_2}{2^{un}}$. Indeed, when $x_1 \in \mathcal{L}_1$ and $x_2 \in \mathcal{L}_2$ are selected uniformly at random, then $\Pr(x_1 \oplus x_2 = t|*) = 2^{-un}$. This average follows by linearity of the expectation. Thus, the average time complexity of algorithms based on merging is easy to compute; this is what Wagner does [Wag02]. However, in practice, and in order to simplify the analysis, we want to know how much we can deviate from this average.

If the elements of $\mathcal{L}_u$ were all independent, this deviation would be small thanks to Chernoff bounds. But this is usually not the case, and assuming a small variance without further justification is a heuristic assumption. Minder and Sinclair bounded the variance in list sizes for Wagner's algorithm and more generally, for merging algorithms that start from lists of uniformly distributed elements [MS12, Section 4]. They used Chebyshev's inequality and bound the covariance between random variables that indicate whether a tuple of initial elements form a solution. Instead of going into the details of their proof, we will give alternative arguments sufficient for the algorithms presented in this chapter and the next one.

**Lemma 5.1** (Classical merging)**.** *Let $t$ be an arbitrary prefix of $un$ bits. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two lists of respective sizes $2^{\ell_1 n}$ and $2^{\ell_2 n}$, of $n$-bit strings drawn uniformly at random.*

*Then the join list $\mathcal{L}_u = \mathcal{L}_1 \bowtie_u \mathcal{L}_2$ can be computed in average time:*

$$\max\left(\frac{L_1 L_2}{2^{un}}, \min(L_1, L_2)\right) \quad , \tag{5.5}$$

*that is, in $\log_2$, $\max(\ell_1 + \ell_2 - u, \min(\ell_1, \ell_2))$. This list is of size $L_u$ such that $\mathbb{E}(L_u) = \frac{L_1 L_2}{2^{un}}$.*

*Furthermore, if $\ell_u \leqslant \max(\ell_1, \ell_2)$, there exists two constants $a, b > 0$ such that with probability $1 - e^{-an}$, a proportion $b$ of the elements of $\mathcal{L}_u$ are drawn uniformly at random from all $n$-bit strings with prefix $t$.*

*Proof.* The merging procedure is given in Algorithm 5.3. If $L_1 > L_2$, then we will roughly iterate on all elements of $\mathcal{L}_1$ while trying to find matching elements in $\mathcal{L}_2$. If $L_1 < L_2$, the converse happens. The total time taken is at least the time to write the pairs; this gives Equation (5.5).

As an example of non-independence between the output pairs, let us consider $x_1, y_1 \in \mathcal{L}_1$ and $x_2, y_2 \in \mathcal{L}_2$, then the events $x_1 \oplus x_2 = t|*$, $y_1 \oplus x_2 = t|*$, $x_1 \oplus y_2 = t|*$ and $y_1 \oplus y_2 = t|*$ are not independent. In order to recover independence when $\ell_u \leqslant \max(\ell_1, \ell_2)$, we will use an argument similar to [Lyu05]. We first need a technical result, which is a property of random mappings.

**Lemma 5.2.** *Let $h : \{0,1\}^n \to \{0,1\}^n$ be a random function. Let $Y(h)$ be the number of elements in $\{0,1\}^n$ without a preimage. Then:*

$$\Pr(Y(h) > 0.4 \cdot 2^n) \leqslant 0.9987^{2^n} \quad . \tag{5.6}$$

*Proof.* We write $Y(h) = \sum_{i \in \{0,1\}^n} Y_i(h)$. The $Y_i(h)$ are not independent, but they are negatively correlated: knowing that $x$ has no preimage only decreases the probability that this is the case for $x' \neq x$. In that case, a Chernoff bound applies [PS97], and for any $\delta > 0$:

$$\Pr\left(Y(h) \geqslant \frac{(1+\delta)2^n}{e}\right) \leqslant \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\frac{2^n}{e}} \quad ,$$

where $\frac{2^n}{e}$ is the average of $Y(h)$, which is a standard result of random mappings [FO89]. We then choose $\delta = 0.4e - 1 \simeq 0.087$ and obtain the claimed bound by rounding the term on the left side. □

Assume without loss of generality that $\ell_1 \leqslant \ell_2$. The idea is that since $\ell_u \leqslant \ell_2 \implies \ell_1 \leqslant u$, a given element of $\mathcal{L}_1$ intervenes in one pair on average, which ensures independence between the pairs. We assume that $\ell_2 - u > 0$.

First of all, we cut $\mathcal{L}_2$ into sublists $\mathcal{L}_2^x$ depending on $un$-bit prefixes $x$. Then any element in $\mathcal{L}_u$ is the sum of an element $x_1$ in $\mathcal{L}_1$ of prefix $x$, and an element of $x_2$ of $\mathcal{L}_2^x$. We use Chernoff bounds to show that the individual sizes of the $\mathcal{L}_2^x$ do not deviate much from their expectation $\mathbb{E}\left(L_2^x\right) = 2^{n(\ell_2 - u)}$. We have:

$$\forall x, \forall \delta \leqslant 1, \Pr(|L_2^x - \mathbb{E}\left(L_2^x\right)| \geqslant \delta\, \mathbb{E}\left(L_2^x\right)) \leqslant 2e^{-\frac{\delta^2\, \mathbb{E}\left(L_2^x\right)}{3}}, \tag{5.7}$$

and by taking $\delta = (\mathbb{E}\left(L_2^x\right))^{1/3}$, since $\mathbb{E}\left(L_2^x\right)$ is exponential in $n$, taking a union bound over all prefixes $x$ does not change that the probability to deviate is vanishingly small:

$$\Pr\Big(\exists x, |L_2^x - \mathbb{E}\left(L_2^x\right)| \geqslant \mathbb{E}\left(L_2^x\right)^{2/3}\Big) \leqslant 2^{un+1} e^{-\frac{\mathbb{E}\left(L_2^x\right)^{1/3}}{3}} \quad . \tag{5.8}$$

Focusing on $\mathcal{L}_1$, we extract a sublist $\mathcal{L}_1'$ of its elements having *distinct* prefixes of $un$ bits, and we show that the size of $\mathcal{L}_1'$ is only smaller by a constant. This comes from Lemma 5.2. If we index elements of $\mathcal{L}_1$ by their $\ell_1 n$-bit prefix, then at least a constant proportion of these prefixes are occupied, with probability exponentially close to 1. In other words, we are free to store $\mathcal{L}_1$ in a *unique prefix list.* Combining this with Equation (5.8), we bound the deviation of the merged list size from its expectation.

It remains to observe that $\mathcal{L}_1' \bowtie \mathcal{L}_2 \subseteq \mathcal{L}_u$ contains independent sums, since each element of $\mathcal{L}_2$ appears at most once. In the case where $\ell_1 = \ell_2 = u$, taking unique prefixes for both lists $\mathcal{L}_1$ and $\mathcal{L}_2$ gives the same result. □

**Algorithm 5.3** Classical merging.

```
Input: $\mathcal{L}_1$, $\mathcal{L}_2$ sorted lists of $n$-bit strings of sizes $L_1, L_2$, prefix $t$ of $un$ bits. Elements
of these lists are denoted $x_{1,i_1}$ and $x_{2,i_2}$ respectively.
Output: join list $\mathcal{L}_u = \mathcal{L}_1 \bowtie_u \mathcal{L}_2$
function MERGE($\mathcal{L}_1$, $\mathcal{L}_2$, $t$)
    Set $b, s = 1, 2$ if $L_1 > L_2$ and $2, 1$ otherwise
    If $t$ is not provided, set $t = 0_{un}$
    $\mathcal{L}_s \leftarrow \{x \oplus (t\|0), x \in \mathcal{L}_s\}$
    $i_1 \leftarrow 0, i_2 \leftarrow 0$
    $\mathcal{L}_u \leftarrow \varnothing$
    while $i_1 \leqslant L_1$ and $i_2 \leqslant L_2$ do
        if $x_{1,i_1}|_{un} > x_{2,i_2}|_{un}$ then
            Advance $i_2$ until $x_{1,i_1}|_{un} \leqslant x_{2,i_2}|_{un}$
        else if $x_{1,i_1}|_{un} < x_{2,i_2}|_{un}$ then
            Advance $i_1$ until $x_{1,i_1}|_{un} \geqslant x_{2,i_2}|_{un}$
        else
            $\mathcal{L}_u \leftarrow \mathcal{L}_u \cup \{(x_{b,i_b}, x_{s,i_s} \oplus t)\}$
return $\mathcal{L}_u$
```

A consequence of Lemma 5.1 is that it is sound to consider the merging operation to be exact, as long as the successive list sizes decrease. The situation is different if the list sizes increase, as in Minder and Sinclair's generalization [MS12] and in Chapter 6. For example, if we take the cross-product of two lists $\mathcal{L}_1 \times \mathcal{L}_2 = \{x_1 \oplus x_2, x_1 \in \mathcal{L}_1, x_2 \in \mathcal{L}_2\}$, it is not statistically close to a list of random bit-strings of size $L_1 L_2$.

By abuse of notation, let us now denote by $\mathcal{L}_i$ lists of *random variables* over $\{0,1\}^n$ following the uniform distribution. When we merge two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ on a prefix of $un$ bits, we are actually selecting a subset of $\mathcal{L}_1 \times \mathcal{L}_2$; a subset of all sums $X_1 \oplus X_2, X_1 \in \mathcal{L}_1, X_2 \in \mathcal{L}_2$, depending on the $un$ first bits of $X_1$ and $X_2$. But in the variables $X_i$, the $un$-bit prefix and the $(1-u)n$ remaining bits are independent. Thus, $\mathcal{L}_u = \mathcal{L}_1 \bowtie_u \mathcal{L}_2$ will contain sums of the form $t | X'_1 \oplus X'_2$, where $X'$ is a subset of $(1-u)n$ bits of $X$.

At one limit case, all these sums are independent, this is the proof of Lemma 5.1. At the other limit, the list is a complete product. In general, we are somewhere in-between. We remark that such a list satisfies all the statistical properties that we need. In particular, when selecting a sublist based on a prefix, the sublist has a size close to its expectation, by a global constant, thanks to Chernoff bounds. Thus, subsequent merging operations will produce lists of adequate size, which will also contain sums of the initial $X_i$, and so on.

**Lemma 5.3** (Classical merging (informal))**.** *Consider a sequence of merging operations (with a constant number of mergings), starting with lists of uniformly random bit-strings, of exponential size. Then there exists two constants* $a, b > 0$ *such that all subsequent lists* $\mathcal{L}$ *have a size between* $\frac{1}{b}\mathbb{E}(L)$ *and* $b\,\mathbb{E}(L)$*, with probability* $1 - e^{an}$*.*

### 5.2.3 Wagner's Algorithm

Wagner's algorithm [Wag02] solves Problem 5.1 using a recursive *merging strategy*, with a constant (in $k$) number of calls to the procedure Merge of Algorithm 5.3. It builds a sequence of lists of *partial* $\ell$-XOR on $un$ bits, for increasing values of $u < 1$ and $\ell < k$, culminating into a single $k$-XOR (on expectation). In the following, we take a constant $k$ and a random $h : \{0,1\}^n \to \{0,1\}^n$.

**Example: 4-XOR.** Wagner's algorithm for 4-XOR is detailed in Algorithm 5.4. Its strategy can be depicted by the binary tree of Figure 5.1.

**Algorithm 5.4** Merging-based 4-XOR algorithm (*breadth-first*).

```
Input:  oracle access to $h$
Output: $x_1, x_2, x_3, x_4$ such that $h(x_1) \oplus h(x_2) \oplus h(x_3) \oplus h(x_4) = 0$
Create 4 lists $\mathcal{L}_i, 1 \leqslant i \leqslant 4$ of pairs $x, h(x)$, of size $2^{n/3}$
$\mathcal{L}_{12} \leftarrow \textsc{Merge}(\mathcal{L}_1, \mathcal{L}_2, 0_{n/3})$        ▷ $\mathcal{L}_{12}$ is of average size $\frac{2^{n/3} \times 2^{n/3}}{2^{n/3}} = 2^{n/3}$
$\mathcal{L}_{34} \leftarrow \textsc{Merge}(\mathcal{L}_3, \mathcal{L}_4, 0_{n/3})$
Find a collision between $\mathcal{L}_{12}$ and $\mathcal{L}_{34}$.
                ▷ There are $2^{n/3} \times 2^{n/3}$ pairs in $\mathcal{L}_{12} \times \mathcal{L}_{34}$ and $\frac{2n}{3}$ bits to put to zero.
```

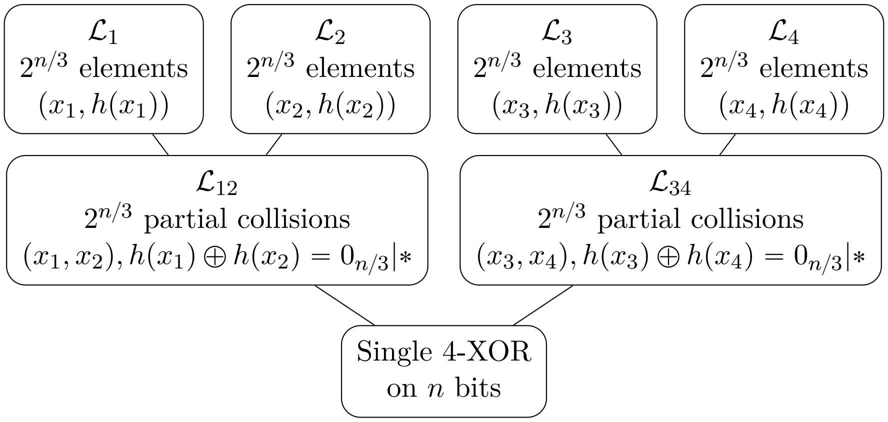

$\mathcal{L}_1$ $2^{n/3}$ elements $(x_1, h(x_1))$
$\mathcal{L}_2$ $2^{n/3}$ elements $(x_2, h(x_2))$
$\mathcal{L}_3$ $2^{n/3}$ elements $(x_3, h(x_3))$
$\mathcal{L}_4$ $2^{n/3}$ elements $(x_4, h(x_4))$
$\mathcal{L}_{12}$ $2^{n/3}$ partial collisions $(x_1, x_2), h(x_1) \oplus h(x_2) = 0_{n/3}|*$
$\mathcal{L}_{34}$ $2^{n/3}$ partial collisions $(x_3, x_4), h(x_3) \oplus h(x_4) = 0_{n/3}|*$
Single 4-XOR on $n$ bits

**Figure 5.1:** Structure of Wagner's 4-XOR tree.

**General k.** We write $\kappa = \lfloor \log_2(k) \rfloor$ for ease of notation. In the context of Wagner's algorithm, if $k$ is not a power of 2, we first take $k - 2^\kappa$ arbitrary elements $z_1, \dots, z_{k-2^\kappa}$ and then find a $2^\kappa$-XOR on their sum. So we assume without loss of generality that $k = 2^\kappa$. All the lists in the tree will have size $2^{\frac{n}{\kappa+1}}$.

- At the leaves of the tree (level $\kappa$), we build $2^\kappa$ lists of $2^{\frac{n}{\kappa+1}}$ single elements, making random queries to $H$.
- At level $\kappa - 1$, we merge the lists pairwise, obtaining $2^{\kappa-1}$ lists, each one containing $2^{\frac{n}{\kappa+1}}$ collisions on $\frac{n}{\kappa+1}$ bits.
- At level $i$ ($1 \leqslant i \leqslant \kappa - 1$), we have $2^i$ lists of $2^i$-tuples which XOR to zero on $\frac{in}{\kappa+1}$ bits: each level puts $\frac{n}{\kappa+1}$ new bits to zero. Notice that all these bit-positions are arbitrary and fixed, for example prefixes of increasing size.
- Finally, we merge two lists of $2^{\kappa-1}$-tuples XORing to zero on $\frac{(\kappa-1)n}{\kappa+1}$ bits, both lists having size $2^{\frac{n}{\kappa+1}}$. We expect on average one $2^\kappa$-tuple to entirely XOR to zero.

**Theorem 5.3** (Correctness of Wagner's algorithm)**.** *When $h : \{0,1\}^n \to \{0,1\}^n$ is a random function, Wagner's algorithm runs in time: $\mathcal{O}\left(2^{\frac{n}{\kappa+1}}\right)$, and succeeds with constant probability. It uses $\mathcal{O}\left(2^{\frac{n}{\kappa+1}}\right)$ queries to h and classical RAM. The constant in $\mathcal{O}$ depends on $k$.*

*Proof.* The only queries to $h$ are made at the first level. The algorithm calls the procedure Merge a constant (in $k$) number of times; for each merge, we apply Lemma 5.1. Note that we can store the lists in *unique-prefix* data structure, thus removing the need to sort if we have RAM access. Taking slightly bigger lists (but no more than by a constant in $k$) ensures that immediately before the root, we obtain two lists of size $2^{\frac{n}{\kappa+1}}$. The deviation of the list sizes is negligibly small. A collision occurs then with constant probability. □

**Further works.** Wagner's algorithm provides currently the best classical time complexity exponent of $\frac{1}{[\log_2(k)]+1}$ for all $k$ (there are logarithmic improvements for non-powers of 2). Many subsequent works have improved the memory consumption and given new trade-offs [Ber+09; NS15; Din19], but we shall not study them in detail. Minder and Sinclair have studied a variation of Wagner's algorithm where the input list sizes are bounded. Their optimal algorithms roughly run in two steps: in the first levels of the binary tree, all pairs of elements are produced, increasing the list sizes; after this expansion step, classical merging is used.

### 5.2.4 Wagner's Binary Tree in a Breadth-first Order

To build a node of the tree, it suffices to have built its children; not necessarily all nodes of bigger depth. Wagner [Wag02] already remarks that this allows to reduce the memory requirement of his algorithm from $2^\kappa$ lists to $\kappa$.

In Figure 5.2, we highlight the difference between these two strategies, by considering the 4-XOR tree of Figure 5.1. In a *breadth-first* manner, we build one level after another, and four lists need to be stored (the whole lowest level). In a depth-first manner, only two lists need to be stored.

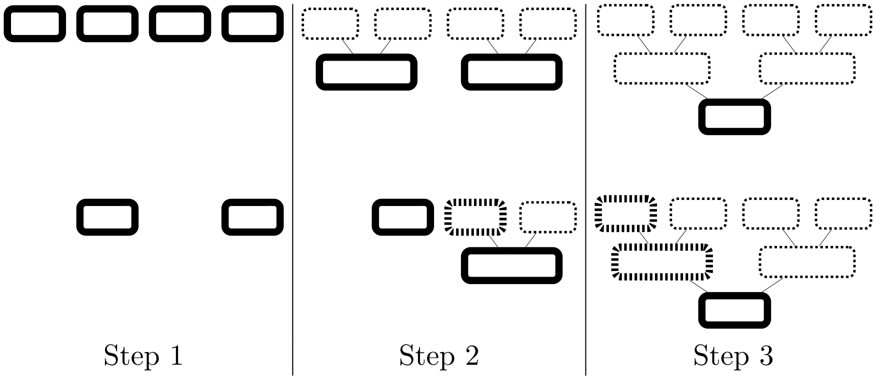


**Figure 5.2:** Building the 4-XOR tree of Figure 5.1 in a breadth-first (above) or depth-first manner (below). The new nodes are put in bold. Between two steps, only the lists in bold are stored. Dotted lists are either discarded at this step, or do not need to be stored at all.

This change of perspective has almost no incidence on Theorem 5.3, and the success of the merging strategy is still guaranteed. We illustrate this with a 4-XOR "breadth-first" merging algorithm, that we translate as a combination of **Sample** procedures. The strategy is the same, only the algorithm is rephrased. Instead of merging lists pairwise, we **Sample** from new lists, using the previously computed ones to eliminate some bits. There is *a priori* an issue with the variations in the number of partial solutions found. For example, in the loop that builds $\mathcal{L}_{34}$, we should add *all* partial collisions found and not only the first one. However, by storing all lists in *unique-prefix* data structures, we will ensure *a single partial solution at most* at each time. We have seen in Section 5.2.2 that only a constant fraction of the lists is lost in this process.

**Algorithm 5.5** Sample-based 4-XOR algorithm (*depth-first*).

```
Input: oracle access to $h$
Output: $x_1, x_2, x_3, x_4$ such that $h(x_1) \oplus h(x_2) \oplus h(x_3) \oplus h(x_4) = 0$

Initialize $\mathcal{L}_2 = \varnothing, \mathcal{L}_4 = \varnothing$
Repeat $2^{n/3}$ times                                  ▷ We build $\mathcal{L}_2$ and $\mathcal{L}_4$
    Sample $x_2 \in \{0,1\}^n$
    $\mathcal{L}_2 \leftarrow \mathcal{L}_2 \cup \{(x_2, h(x_2))\}$
    Sample $x_4 \in \{0,1\}^n$
    $\mathcal{L}_4 \leftarrow \mathcal{L}_4 \cup \{(x_4, h(x_4))\}$
EndRepeat
Initialize $\mathcal{L}_{34} = \varnothing$
Repeat $2^{n/3}$ times                                  ▷ We build $\mathcal{L}_{34}$
    Sample $x_3 \in \{0,1\}^n$
    if $\exists (x_4, h(x_4)) \in \mathcal{L}_4, h(x_3) \oplus h(x_4) = 0_{n/3}|*$ then
        $\mathcal{L}_{34} \leftarrow \mathcal{L}_{34} \cup \{(x_3, x_4, h(x_3) \oplus h(x_4)\}$
EndRepeat
Drop $\mathcal{L}_4$
Sample $x_1 \in \{0,1\}^n$ such that                    ▷ We build the root node
    Find $(x_2, h(x_2)) \in \mathcal{L}_2$ such that $h(x_1) \oplus h(x_2) = 0_{n/3}|*$
(if there is no solution, abort)
    if $\exists (x_3, x_4, h(x_3) \oplus h(x_4)) \in \mathcal{L}_{34}, h(x_1) \oplus h(x_2) \oplus h(x_3) \oplus h(x_4)$ then
        exit and return $x_1, x_2, x_3, x_4$
EndSample
```

### 5.2.5 Towards Quantum Merging

Quantum search has been a very useful tool in quantum collision search, for $k = 2$. It seems natural to try to use it to speed-up classical merging algorithms for $k$-XOR. Unfortunately, the Merge function defined in Algorithm 5.3 cannot be accelerated: the bottleneck of the complexity is reading the input lists and writing the output list.

However, we have seen that the depth-first variant is a sequence of **Sample** procedures. Each procedure samples an element in a new list by partially matching (not merging) against the previous ones. We also know since Chapter 2 that **Sample** programs translate naturally into quantum algorithms. Together, these two ideas pave the way for *quantum merging algorithms* analogous to classical merging.

## 5.3 Merging Trees for the k-XOR Problem

In this section, we describe a framework of classical and quantum merging algorithms, that we name *merging trees*. Its goal is to describe merging strategies for $k$-XOR that directly translate into **Sample** programs. Merging trees can then be optimized for classical or quantum computations. We will, naturally, focus on the quantum case.

The presentation of *merging trees* in this section will differ from [NS20], but the trees will represent the same family of algorithms.

### 5.3.1 Examples of Quantum Merging

Let us start with some examples of optimal quantum merging trees for concrete values of $k$.

#### 5.3.1.1 4-XOR

Translating Algorithm 5.5 into a quantum algorithm, we would obtain the following (heuristic) complexity:

$$\underbrace{2^{n/3}}_{\mathcal{L}_2} + \underbrace{2^{n/3}}_{\mathcal{L}_4} + \underbrace{2^{n/3}}_{\mathcal{L}_{34}} + \underbrace{\mathcal{O}\left(\sqrt{2^{n/3}}\right)}_{\text{Final sample}} = \mathcal{O}\left(2^{n/3}\right) \quad , \tag{5.9}$$

where the final **Sample** procedure requires QRACM access to $\mathcal{L}_2$ and $\mathcal{L}_{34}$, and superposition queries to $h$. As said before for merging, writing the lists $\mathcal{L}_2, \mathcal{L}_4, \mathcal{L}_{34}$ is a step that we do not expect to accelerate, and this is the bottleneck of the algorithm.

Though it is not necessary to change the structure of the tree, this situation calls for an *inherently quantum* re-optimization of the parameters. This is displayed in Figure 5.3. We will decrease the size of the lists $\mathcal{L}_2, \mathcal{L}_4, \mathcal{L}_{34}$ and increase the size of $\mathcal{L}_1$ and $\mathcal{L}_{12}$. The complexity becomes:

$$\underbrace{2^{n/4}}_{\mathcal{L}_2} + \underbrace{2^{n/4}}_{\mathcal{L}_4} + \underbrace{2^{n/4}}_{\mathcal{L}_{34}} + \underbrace{\mathcal{O}\left(\sqrt{2^{n/2}}\right)}_{\text{Final sample}} = \mathcal{O}\left(2^{n/4}\right) \quad . \tag{5.10}$$

Thus, we have obtained an exponential quantum speedup for 4-XOR *in the QRACM model*. The same time complexity was obtained in [GNS18] in the QRAQM model, with a quantum walk, a more advanced algorithmic framework.

#### 5.3.1.2 3-XOR

Classically, the 3-XOR problem is almost as hard as collision search, as only a logarithmic improvement is known (see [BDF18] for a practical study). In the quantum setting, we will use a merging strategy that is inspired by classical rebound attacks [Nay11].

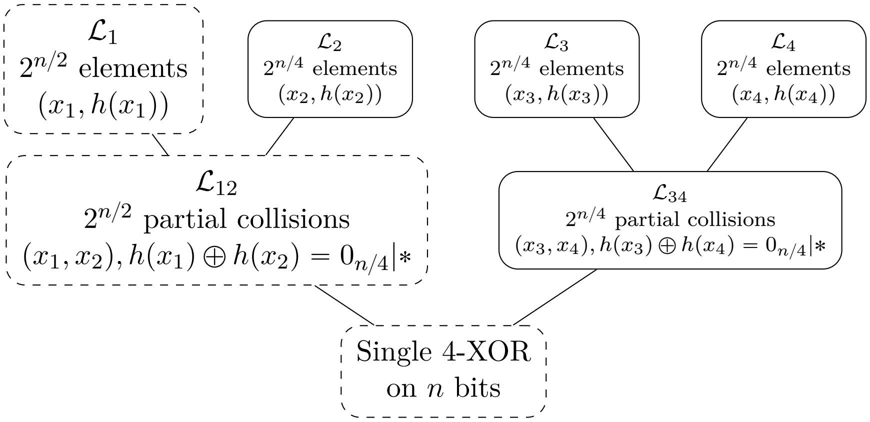


**Figure 5.3:** Re-optimization of 4-XOR merging. Dashed lists correspond to the final **Sample** procedure.

**Algorithm 5.6** Quantum 3-XOR Algorithm with QRACM.

```
Input: oracle access to $h$
Output: $x_1, x_2, x_3$ such that $h(x_1) \oplus h(x_2) \oplus h(x_3) = 0$
Initialize $\mathcal{L}_2 = \mathcal{L}_3 = \varnothing$
Repeat $2^{2n/7}$ times                                         ▷ Building $\mathcal{L}_2$
    Sample $x_2 \in \{0,1\}^n$
    $\mathcal{L}_2 \leftarrow \mathcal{L}_2 \cup \{(x_2, h(x_2))\}$
EndRepeat
Repeat $2^{n/7}$ times                                          ▷ Building $\mathcal{L}_3$
    Sample $x_3 \in \{0,1\}^n$ such that $h(x_3) = 0_{2n/7}|*$
    $\mathcal{L}_3 \leftarrow \mathcal{L}_3 \cup \{(x_3, h(x_3))\}$
EndRepeat
Sample $x_1 \in \{0,1\}^n$ such that
    Find $(x_2, h(x_2)) \in \mathcal{L}_2$ such that $h(x_1) \oplus h(x_2) = 0_{2n/7}|*$
    if $\exists (x_3, h(x_3)) \in \mathcal{L}_3, h(x_1) \oplus h(x_2) \oplus h(x_3)$ then
        exit and return $x_1, x_2, x_3$
EndSample
```

The heuristic complexity of Algorithm 5.6 is:

$$\underbrace{2^{2n/7}}_{\mathcal{L}_2} + \underbrace{2^{n/7} \cdot \sqrt{2^{2n/7}}}_{\mathcal{L}_3} + \sqrt{2^{4n/7}} \ , \tag{5.11}$$

as intuitively, the lists $\mathcal{L}_2$ and $\mathcal{L}_3$ allow to eliminate $\frac{3n}{7}$ bits, and $\frac{4n}{7}$ remain to be put to zero. Trying to optimize this strategy classically will not decrease the time complexity exponent below $\frac{1}{2}$. In the quantum setting, collision search does not benefit from a quadratic speedup, and using quantum search shifts the balance in our favor.

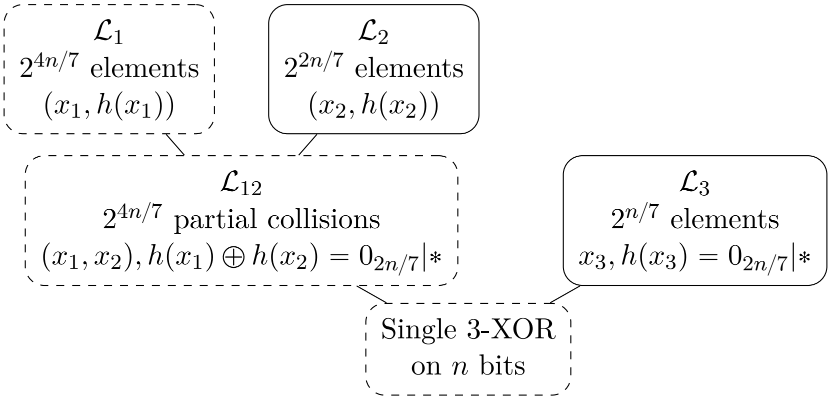


**Figure 5.4:** Quantum 3-XOR merging strategy (with QRACM).

#### 5.3.1.3 Collisions

The BHT algorithm is a 2-XOR algorithm in the QRACM model, and the collision search algorithm of Section 5.1 is a 2-XOR algorithm in the circuit model. Both would be represented by trees of depth 1.

### 5.3.2 Definition of Merging Trees

The goal of *merging trees* is to represent quantum merging strategies for $k$-XOR in a purely syntactical way.

**Definition 5.3** (Merging trees)**.** A $k$-merging tree $\mathcal{T}_k$ is a binary tree defined recursively as follows:

- A node is either labeled "Sample" (S-node) or "List" (L-node);
- If $k = 1$, this is a leaf node $\mathcal{T}_1$;
- If $k > 1$, $\mathcal{T}_k$ has two children: an S-node $\mathcal{T}_{k_l}$ and an L-node $\mathcal{T}_{k_r}$, where $k_l + k_r = k$.

It follows inductively that a $k$-merging tree has $k$ leaf nodes. Intuitively, an S-node represents a **Sample** from a given list and an L-node represents a list stored in memory, constructed with exponentially many samples.

By convention, we draw Sample nodes (dashed) on the left and List nodes (plain) on the right. To each node $\mathcal{T}$ corresponds a list $\mathcal{L}$ which is either *built* or *sampled.* Since the trees are binary, we adopt a simple numbering of lists $\mathcal{L}_i^j$. The root node, at level 0 in the tree, is $\mathcal{L}_0^0$, and the two children of $\mathcal{L}_i^j$ are numbered respectively $\mathcal{L}_{2i}^{j+1}$ for the sampled one and $\mathcal{L}_{2i+1}^{j+1}$ for the list one. We label a node with the following variables representing the characteristics of $\mathcal{L}_i^j$:

- The *width* $k_i^j$;

- The number $u_i^j$ of bits set to zero (relatively to $n$);
- The size $\ell_i^j$ of this list: following our conventions, $\ell_i^j$ represents a size of $2^{\ell_i^j n}$.

Thus, $\mathcal{L}_i^j$ is a list of $k_i^j$-tuples $x_1, \ldots, x_{k_i^j}$ such that $x_1 \oplus \ldots \oplus x_{k_i^j} = 0_{u_i^j n}|*$, of size $2^{\ell_i^j n}$, which may or may not be stored in memory.

**Merging strategy and constraints.** We constraint the variables $\ell_i^j$ and $u_i^j$ in order to represent a valid merging strategy. First, the goal is to output a single $k$-XOR solution.

**Constraint 5.1** (Root node)**.** *At the root node:* $u_0^0 = 1$ *and* $\ell_0^0 = 0$.

As each node results from the merging of its two children, the number of zeros increases. Furthermore, two siblings shall have the same number of zeros: $u_{2i}^j = u_{2i+1}^j$. Otherwise, we could reduce this proportion to the minimum between them, obtaining an easier strategy.

**Constraint 5.2** (Zero-prefixes)**.**

$$\forall i, j \geqslant 1, u_{2i}^j = u_{2i+1}^j \textit{ and } u_i^{j-1} \geqslant u_{2i}^j$$

Finally, the size of a list is constrained by the sizes of its predecessors and the new constraint ($(u_i^{j-1} - u_{2i+1}^j)n$ more bits to put to zero).

**Constraint 5.3** (Size of a list)**.**

$$\forall i, j \geqslant 1, \ell_i^{j-1} = \ell_{2i}^j + \ell_{2i+1}^j - (u_i^{j-1} - u_{2i+1}^j)$$

An example of merging tree for $k = 7$ is given in Figure 5.5.

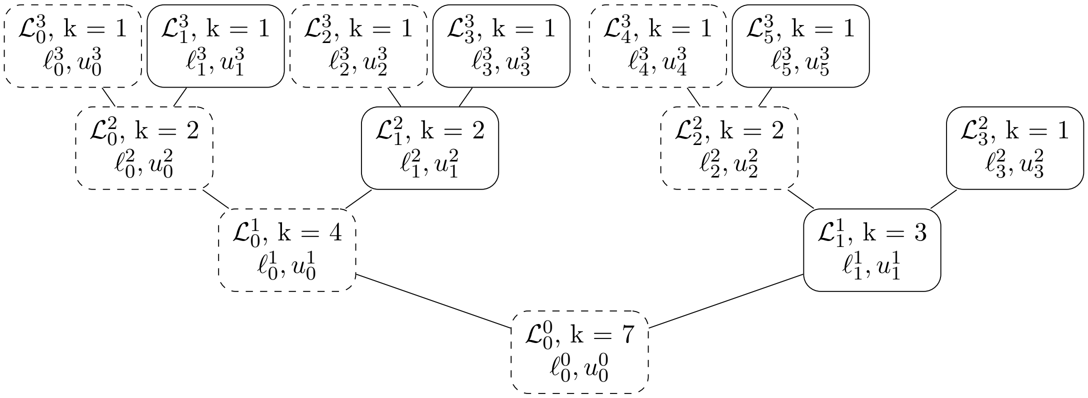

$\mathcal{L}_0^3$, k = 1 $\ell_0^3, u_0^3$
$\mathcal{L}_1^3$, k = 1 $\ell_1^3, u_1^3$
$\mathcal{L}_2^3$, k = 1 $\ell_2^3, u_2^3$
$\mathcal{L}_3^3$, k = 1 $\ell_3^3, u_3^3$
$\mathcal{L}_4^3$, k = 1 $\ell_4^3, u_4^3$
$\mathcal{L}_5^3$, k = 1 $\ell_5^3, u_5^3$
$\mathcal{L}_0^2$, k = 2 $\ell_0^2, u_0^2$
$\mathcal{L}_1^2$, k = 2 $\ell_1^2, u_1^2$
$\mathcal{L}_2^2$, k = 2 $\ell_2^2, u_2^2$
$\mathcal{L}_3^2$, k = 1 $\ell_3^2, u_3^2$
$\mathcal{L}_0^1$, k = 4 $\ell_0^1, u_0^1$
$\mathcal{L}_1^1$, k = 3 $\ell_1^1, u_1^1$
$\mathcal{L}_0^0$, k = 7 $\ell_0^0, u_0^0$

**Figure 5.5:** Merging tree for $k = 7$.

### 5.3.3 From Trees to Algorithms

Next, we show that a given merging tree always corresponds to a quantum algorithm for $k$-XOR, and we compute its time complexity depending on the tree parameters. We use a result analogous to Lemma 5.1.

**Lemma 5.4** (Quantum merging)**.** *Let $t$ be an arbitrary prefix of $un$ bits. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two lists of respective sizes $2^{\ell_1 n}$ and $2^{\ell_2 n}$. Assume that $\mathcal{L}_2$ is stored either in QRACM* or *in CSAM, with a* Unique prefix list *data structure defined in Section 1.3.2.*

*Assume that we are given a quantum sample for $\mathcal{L}_1$: $\mathsf{QSample}_{\mathcal{L}_1}$ that returns (after measurement) an element of $\mathcal{L}_1$ selected uniformly at random, with probability $1 - e^{-an}$ for some constant $a > 0$.*

*Then there exists a function $\mathsf{QSample}_{\mathcal{L}_u}$ that returns (after measurement) an element of $\mathcal{L}_u$, selected uniformly at random, with probability $1 - e^{-bn}$ for some other constant $b > 0$, where $\mathcal{L}_u = \mathcal{L}_1 \bowtie_u \mathcal{L}_2$. Up to a constant, the quantum time complexity of $\mathsf{QSample}_{\mathcal{L}_u}$ is:*

$$\mathsf{T_q}\left(\mathsf{QSample}_{\mathcal{L}_u}\right) = \begin{cases} \mathsf{T_q}\left(\mathsf{QSample}_{\mathcal{L}_1}\right) \cdot \max(2^{\frac{(u-\ell_2)}{2}n}, 1) \text{ \textit{in the QRACM model}} \\ \left(\mathsf{T_q}\left(\mathsf{QSample}_{\mathcal{L}_1}\right) + 2^{\ell_1 n}\right) \cdot \max(2^{\frac{(u-\ell_2)}{2}n}, 1) \text{ \textit{with CSAM}} \end{cases} \tag{5.12}$$

*in qRAM gates and $n$-qubit register operations.*

*Proof.* Using a *unique prefix list* avoids the issue of multiple matching solutions, and we can quickly see that it will bring no harm in a more general merging tree. The size of all lists is guaranteed by Lemma 5.1, and we have seen that the constraint of unique prefixes leads only to the loss of a constant fraction of the lists.

We use an Amplitude Amplification, where the amplified algorithm consists in running $\mathsf{QSample}_{\mathcal{L}_1}$, finding whether there is a match of the given prefix in $\mathcal{L}_2$, and returning the pair if it exists. The probability of success depends on the size of the merged list, which is controlled by the Chernoff bound of Lemma 5.1. Thus, we can use Theorem 2.5. The error of the full procedure will remain exponentially low.

To obtain the time complexity, we separate two cases: if $u > \ell_2$, then the amplification really needs to take place, and it has $2^{(u-\ell_2)n}$ iterations up to a constant. Each iteration calls $\mathsf{QSample}_{\mathcal{L}_1}$ (setup) and queries the unique-prefix list. Without the QRACM, we use an emulation circuit.

If $u < \ell_2$, then a given element $x_1 \in \mathcal{L}_1$ will have on average exponentially many $x_2 \in \mathcal{L}_2$ such that $x_1 \oplus x_2 = t|*$. As we want the uniform superposition of them, we use the ability of the *unique-prefix* list to return a superposition of elements of a given prefix. This also works with the emulation circuit. □

**Time.** We attach to each node another parameter $t$, which represents the *sample time*. Our intuition is that the time to sample from the list $\mathcal{L}_i^j$ represented by this node should be, in practice, $\mathcal{O}\big(2^{nt}\big)$.

**Constraint 5.4** (Sampling). *Let $\mathcal{T}_i^j$ be a node in the tree, either an S-node or an L-node.*

- *if $\mathcal{T}_i^j$ is a leaf, $t_i^j = \frac{u_i^j}{2}$.*
- *otherwise, $\mathcal{T}_i^j$ has an S-child $\mathcal{T}_{2i}^{j+1}$ and an L-child $\mathcal{T}_{2i+1}^{j+1}$, and:*

$$t_i^j = \begin{cases} t_{2i}^{j+1} + \frac{1}{2}\max(u_i^j - u_{2i}^{j+1} - \ell_{2i+1}^{j+1}, 0) \text{ in the QRACM model} \\ \max(t_{2i}^{j+1}, \ell_{2i+1}^{j+1}) + \frac{1}{2}\max((u_i^j - u_{2i}^{j+1} - \ell_{2i+1}^{j+1}, 0) \text{ with CSAM} \\ t_{2i}^{j+1} + \max(u_i^j - u_{2i}^{j+1} - \ell_{2i+1}^{j+1}, 0) \text{ classically} \end{cases} \tag{5.13}$$

If the node is a leaf, then we simply run Grover's algorithm multiple times. Equation (5.13) is simply a translation of (5.12) in the case of a specific node. The third option is added only in the circuit model, in the case where quantum samplings are too costly (due to the membership queries) with respect to classical samplings. Notice that in that case, since the node can be sampled only classically, so will be its parent. Next, from the individual sampling times of each node, we can compute what should be the time complexity exponent of a tree.

**Definition 5.4.** Let $\mathcal{T}_k$ be a $k$-merging tree. We define $\mathsf{T_q}(\mathcal{T}_k)$ and $\mathsf{M}(\mathcal{T}_k)$ as:

$$\mathsf{T_q}(\mathcal{T}_k) = \max\left(\max_{\text{List nodes}}\left(t_i^j + \ell_i^j\right), t_0^0\right)$$
$$\mathsf{M}(\mathcal{T}_k) = \max_{\text{List nodes}}\left(\ell_i^j\right)$$

It should be noted that the list size of sample nodes plays only a role in the structural constraints, not in the time complexity. They should simply have a size sufficient to ensure the existence of a solution in the tree.

With these definitions, and with the help of Lemma 5.4, we are now ready to make merging trees correspond to merging algorithms.

**Theorem 5.4** (Quantum merging strategies). *Let $\mathcal{T}_k$ be a $k$-merging tree and $\mathsf{T_q}(\mathcal{T}_k)$ computed as in Definition 5.4. Then there exists a* quantum merging algorithm *that, given access to a quantum oracle $O_h$, finds a k-XOR.*

*This algorithm succeeds with probability more than $1 - e^{-an}$ for some constant $a > 0$. It runs in time $\mathcal{O}\left(2^{\mathsf{T_q}(\mathcal{T}_k)n}\right)$, counted in $n$-qubit register operations and qRAM gates, makes the same number of queries to $O_h$. It requires only $\mathcal{O}(n)$ computing qubits. It uses a memory $\mathcal{O}\left(2^{\mathsf{M}(\mathcal{T}_k)n}\right)$, counted in $n$-bit registers (either CSAM, RAM or QRACM). The constants in the $\mathcal{O}$ depend on $k$.*

*Proof.* We define recursively the correspondence $\mathcal{T}_k \overset{\mathcal{A}}{\longmapsto} \mathcal{A}(\mathcal{T}_k)$ from a merging tree $\mathcal{T}_k$ to an algorithm $\mathcal{A}(\mathcal{T}_k)$ solving the $k$-XOR problem, with the wanted time and memory complexities.

Let $N(k, u, \ell)$ be the root node of $\mathcal{T}_k$ and $S(k', u', \ell')$ and $L(k'', u'', \ell'')$ its two children, if they exist.

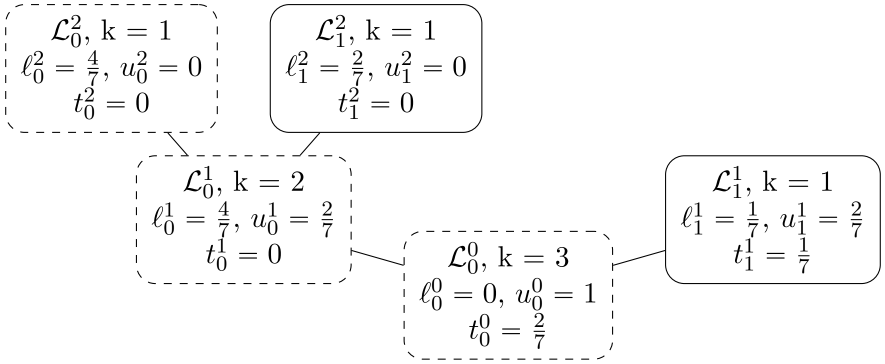

$\mathcal{L}_0^2$, k = 1
$\ell_0^2 = \frac{4}{7}$, $u_0^2 = 0$
$t_0^2 = 0$
$\mathcal{L}_1^2$, k = 1
$\ell_1^2 = \frac{2}{7}$, $u_1^2 = 0$
$t_1^2 = 0$
$\mathcal{L}_0^1$, k = 2
$\ell_0^1 = \frac{4}{7}$, $u_0^1 = \frac{2}{7}$
$t_0^1 = 0$
$\mathcal{L}_1^1$, k = 1
$\ell_1^1 = \frac{1}{7}$, $u_1^1 = \frac{2}{7}$
$t_1^1 = \frac{1}{7}$
$\mathcal{L}_0^0$, k = 3
$\ell_0^0 = 0$, $u_0^0 = 1$
$t_0^0 = \frac{2}{7}$

**Figure 5.6:** 3-XOR (optimal) merging tree with QRACM.

- If it is a Sample leaf, then $\mathcal{A}(\mathcal{T}_k)$ simply consists in running Grover's algorithm in time $\mathcal{O}\left(2^{un/2}\right)$.

- Otherwise, if it is a Sample:

    1. we use $\mathcal{A}(L)$ to sample repeatedly from the child $L$: we build the list. Each call costs time $\mathcal{O}\left(2^{\mathsf{T_q}(L)n}\right)$ and we need to make $\mathcal{O}\left(2^{\ell'' n}\right)$ of them before we obtain a list of the wanted size, with high probability (there is some negligible variation).
    2. then we apply Lemma 5.4, since we have a sample for the child $S$: $\mathcal{A}(S)$

- If it is a List, the situation is the same, except that we repeat the operation exponentially many times.

When taking sums of complexities exponential in $n$, we write $\mathcal{O}(2^{\alpha n}) + \mathcal{O}\left(2^{\beta n}\right) = \mathcal{O}\left(2^{\max(\alpha,\beta)n}\right)$, which remains sound since we do that only a constant number of times. □

Thus, merging trees offer a compact and sound way to represent quantum merging algorithms for the $k$-XOR problem. As an example, we represent Algorithm 5.6 as a merging tree in Figure 5.6.

### 5.3.4 Finding Optimal Trees

Now that we have defined the set of merging trees, we can explore this space to search for the trees $\mathcal{T}_k$ that minimize $\mathsf{T_q}\left(\mathcal{T}_k\right)$.

Given a tree $\mathcal{T}_k$, its time and memory complexity exponents $\mathsf{T_q}\left(\mathcal{T}_k\right)$ and $\mathsf{M}\left(\mathcal{T}_k\right)$ are defined as the maximums of linear combinations of the parameters $\ell_i^j, u_i^j$. Thus, there exists a choice of these parameters that minimizes $\mathsf{T_q}\left(\mathcal{T}_k\right)$, under Constraints 5.1, 5.2, 5.3 and 5.4. Finding this minimum is a linear problem that we can solve with Mixed Integer Linear Programming (MILP). Given $k$, we try all possible tree structures (all binary trees with $k$ leaves) and find the optimal one. As an example, the optimization problem for Figure 5.6 would be:

$$\begin{aligned}
\text{Optimize: } & \max(t_0^0, \ell_1^1 + t_1^1, \ell_1^2 + t_1^2) \\
\text{Under the constraints: } & t_1^1 = \frac{u_1^1}{2}, t_1^2 = \frac{u_1^2}{2}, t_0^2 = \frac{u_0^2}{2}, \\
& t_0^1 = t_0^2 + \frac{1}{2}\max\left(u_0^1 - u_0^2 - \ell_1^2, 0\right), \\
& t_0^0 = t_0^1 + \frac{1}{2}\max\left(1 - u_0^1 - \ell_1^1, 0\right), u_0^2 = u_1^2, u_0^1 = u_1^1
\end{aligned}$$

There always exist an optimal tree $\mathcal{T}_k$ that achieves the best time complexity exponent $\mathsf{T}_{\mathsf{q}}(\mathcal{T}_k)$, but there may also be more than one. It is also easy to see that the exponent obtained will be a rational number.

### 5.3.5 Extensions

The classical literature on merging algorithms contains many extensions to Wagner's algorithm [Din19; Bai+19; Din+12; NS15; MS12]. We tried to extend merging trees with some of these designs, but they did not seem to give an actual quantum improvement, whether in time or memory.

**Clamping.** The *clamping* technique [Ber+09] corresponds to taking non-empty prefixes of zeroes for leaf nodes, allowing for a time-memory trade-off. This is already accessible in the space of merging trees.

**Chain-ends.** Nikolic and Sasaki [NS15] use Hellman tables to replace leaf lists of pairs $(x, h(x))$ by lists of chain-ends $(x, h^{2^s}(x))$. This allows to reduce the memory used in Wagner's algorithm. However, as we saw before for collision search, in the quantum setting, finding elements with arbitrary prefixes is quadratically faster than iterating the function (which is not the case classically). Thus, chain-ends are not better than the clamping technique.

**Solving the k-SUM problem.** Bitwise XORs can be replaced by modular additions without affecting the success probabilities and runtime of the algorithms. Indeed, classical merging algorithms are easily adapted to the $k$-SUM problem [Wag02]. Suppose that all the elements lie in an interval $[-N; N]$ where $N \simeq 2^n$ is some modulus, and we are looking for a $k$-SUM to zero. Instead of using lists of partially colliding elements, we look for sums falling in successive reduced intervals $\left[-\frac{N}{2^{un}}; \frac{N}{2^{un}}\right]$. If $N$ is a power of 2, it is easy to adapt the algorithms. Otherwise, we may take the smallest power of 2 that is bigger than $N$, and do as if the values $h(x)$ fell uniformly at random in $[-2^n; 2^n]$. The statistical difference will be negligible.

## 5.4 Optimal Merging Trees for k-XOR

In this section, we give new quantum algorithms for the $k$-XOR problem with many solutions. We describe them using merging trees and prove their optimality among all merging trees. We distinguish two memory models: the QRACM model and the quantum circuit model (CSAM or classical RAM). To date, none of the best algorithms known for Problem 5.2 requires the QRAQM model.

**Relation with previous results.** With $k = 2$, by optimizing a merging tree with three nodes, we obtain the BHT collision algorithm for random functions if QRACM is used, and Algorithm 5.1 without. They also give the best achievable memory complexity with quantum merging.

The results of [GNS18] are subsumed by the merging tree framework. Their algorithms without QRACM can be seen as special cases of merging trees, and their algorithms with QRAQM can be replaced by merging trees of the same time complexity. The results presented in this section improve on [GNS18] for almost all $k$.

### 5.4.1 Description of the Optimal Trees

Although more than one merging tree can achieve the optimal time exponent for a given $k$, we find that it is reached by a family of balanced trees $T_k$.

**Definition 5.5** (Trees $T_k$)**.** If $k = 1$, then $T_k$ is simply a leaf node. If $k = 2k'$, then the "Sample" child of $T_k$ is $T_{k'}$ and the "List" child is $T_{k'}$. If $k = 2k' + 1$, then the "Sample" child of $T_k$ is $T_{k'+1}$ and the "List" child is $T_{k'}$.

In particular, when $k = 2^\kappa$ is a power of 2, $T_k$ is Wagner's balanced binary tree.

**Proposition 5.3.** *Optimizing $T_k$ under* classical *constraints gives the time complexity exponent of Wagner's algorithm:* $\mathsf{T_c}(T_k) = \frac{1}{\kappa+1}$ .

Quantumly, when $k$ is not a power of 2, $T_k$ has the opportunity of a better quantum time exponent than $T_{k-1}$, for the same reason as in the example of 3-XOR: building a leaf list of elements with a non-empty zero-prefix costs relatively less than in the classical optimization.

### 5.4.2 Results

For our experiments, we used the MILP solver of the SCIP suite [Gle+18a; Gle+18b]. After trying with $k \leqslant 20$, we could can gain an intuition of the optimal tree shape $T_k$ and conjecture closed formulas for the time complexity exponents, that we will prove in Sections 5.4.3 and 5.4.4.

#### 5.4.2.1 QRACM Model

For powers of 2, we obtain $\alpha_k = \frac{1}{\kappa+2}$, to compare with Wagner's algorithm: $\frac{1}{\kappa+1}$. But the exponent also decreases strictly when $k$ increases.

**Theorem 5.5** (Quantum merging complexity for $k$-XOR, with QRACM)**.** *Let $k \geqslant 2$ be an integer and $\kappa = \lfloor \log_2(k) \rfloor$. The best quantum merging tree finds a $k$-XOR on $n$ bits in quantum time (and memory) $\widetilde{\mathcal{O}}(2^{\alpha_k n})$ where $\alpha_k = \frac{2^\kappa}{(1+\kappa)2^\kappa+k}$. To find $2^{cn}$ $k$-XORs for $c > 0$, the complexity exponent goes to* $\max(\alpha_k(1+2c), c)$. *Furthermore, for every $k$, the optimum is realized by $T_k$: $\alpha_k = \mathsf{T_q}(T_k)$.*

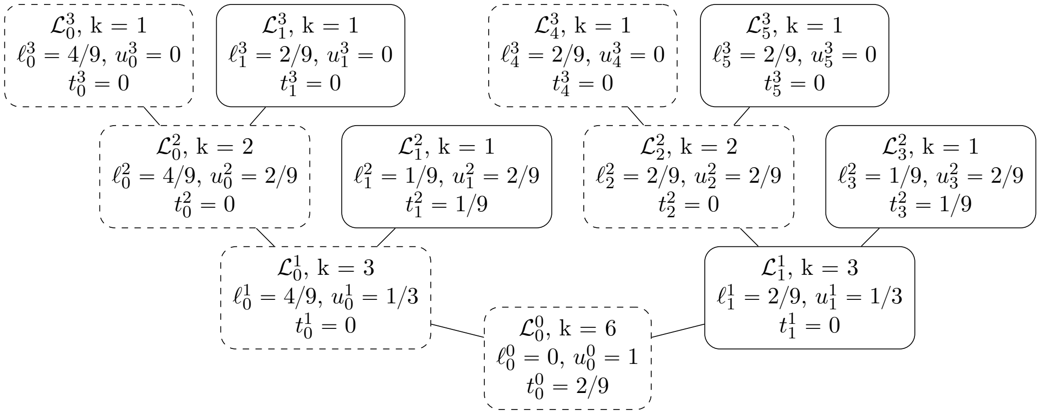


**Figure 5.7:** Optimal merging tree for 6-XOR, with QRACM.

When $k$ is a power of 2, the tree is a complete binary tree and the optimization follows the 4-XOR example of Figure 5.3. Except the lists on the left branch of the tree, all lists are classically built and merged. They are of size $2^{\frac{n}{\kappa+2}}$. Then, the final quantum search requires $\sqrt{2^{\frac{2n}{\kappa+2}}} = 2^{\frac{n}{\kappa+2}}$ iterations.

When $k$ is not a power of 2, the optimization is more complex. We can see in Figure 5.7 that new leaves are introduced in the tree with non-empty zero-prefixes, in order to merge them with other lists that have already zeroes. The parameters can be derived recursively from the optimality proof of Section 5.4.3.

#### 5.4.2.2 Circuit Model

The situation is different in the circuit model, but the balanced $T_k$ defined above remains an optimal tree shape.

**Theorem 5.6.** *Let $k > 2, k \neq 3, 5, 7$ be an integer and $\kappa = \lfloor \log_2(k) \rfloor$. The best quantum merging tree finds a $k$-XOR on $n$ bits in quantum time and classical memory $\mathcal{O}(2^{\beta_k n})$ where:*

$$\beta_k = \begin{cases} \frac{1}{\kappa+1} \textit{ if } k < 2^\kappa + 2^{\kappa-1} \\ \frac{2}{2\kappa+3} \textit{ if } k \geqslant 2^\kappa + 2^{\kappa-1} \end{cases} .$$

*To find* $2^{cn}$ *k-XORs, the complexity exponent goes to* $\max\left(\beta_k(1+c), c\right)$*. Furthermore, for every* $k \neq 3, 5, 7$*, the optimum is realized by* $T_k$*.*

In this case, for $k \neq 2, 3, 5, 7$, the best strategy is always to use classical samplings (i.e., merging), except at some leaves of the tree, where some elements with zero-prefixes are produced using Grover search. If we use quantum search, we need to access the memory via a QRACM emulation circuit, and this becomes suboptimal.

This gives one intermediate level of complexity between two successive powers of 2. For collision search, we obtain Algorithm 5.1 with $\beta_2 = \frac{2}{5}$. Optimal merging strategies for $k = 3$ and $k = 5$ are given in Figure 5.8. The strategy for $k = 3$ was already obtained manually in [GNS18] and reaches $\beta_3 = \frac{5}{14}$. The strategy for $k = 5$ reaches a surprisingly non trivial[1] $\beta_5 = \frac{40}{129}$. The strategy for $k = 7$ also reaches $\beta_7 = \frac{15}{53}$.

#### 5.4.2.3 Comparisons

We plot the exponents depending on $k$ in Figure 5.9. We give numerical values of these exponents in Table 5.1. Our algorithms require as much memory as time, except the cases $k \leqslant 5$ in the circuit model.

**Table 5.1:** Best time complexity exponents of $k$-XOR optimized merging trees for $k$ up to 14. In the circuit model, we highlight speedups with respect to Wagner's algorithm.

| | Classical | | QRACM model | | Circuit model | |
|---|---|---|---|---|---|---|
| $k$ | As fraction | Rounded | As fraction | Rounded | As fraction | Rounded |
| 2 | 1/2 | 0.5 | 1/3 | 0.3333 | 2/5 | **0.4** |
| 3 | 1/2 | 0.5 | 2/7 | 0.2857 | 5/14 | **0.3571** |
| 4 | 1/3 | 0.3333 | 1/4 | 0.25 | 1/3 | 0.3333 |
| 5 | 1/3 | 0.3333 | 4/17 | 0.2353 | 40/129 | **0.3101** |
| 6 | 1/3 | 0.3333 | 2/9 | 0.2222 | 2/7 | **0.2857** |
| 7 | 1/3 | 0.3333 | 4/19 | 0.2105 | 15/53 | **0.2830** |
| 8 | 1/4 | 0.25 | 1/5 | 0.2 | 1/4 | 0.25 |
| 9 | 1/4 | 0.25 | 8/41 | 0.1951 | 1/4 | 0.25 |
| 10 | 1/4 | 0.25 | 4/21 | 0.1905 | 1/4 | 0.25 |
| 11 | 1/4 | 0.25 | 8/43 | 0.186 | 1/4 | 0.25 |
| 12 | 1/4 | 0.25 | 2/11 | 0.1818 | 2/9 | **0.2222** |
| 13 | 1/4 | 0.25 | 8/45 | 0.1778 | 2/9 | **0.2222** |
| 14 | 1/4 | 0.25 | 4/23 | 0.1739 | 2/9 | **0.2222** |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

[1] Note that the exponents for 5 and 7 are slightly below those of [NS20]. This comes from our redefinition of merging trees, with less constraints than before. This is the only difference with the results of [NS20] in this chapter.

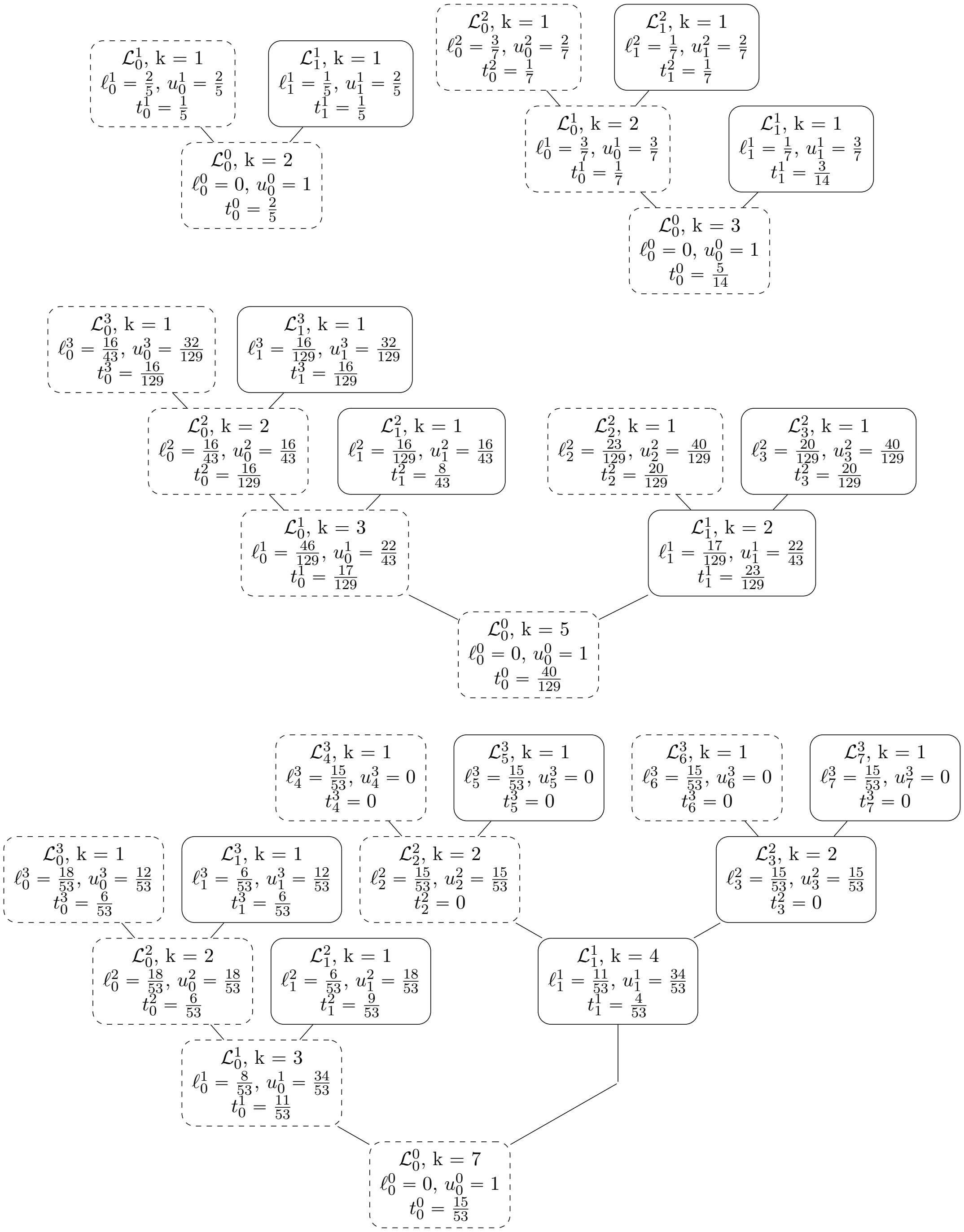

$\mathcal{L}_0^1$, k = 1
$\ell_0^1 = \frac{2}{5}$, $u_0^1 = \frac{2}{5}$
$t_0^1 = \frac{1}{5}$
$\mathcal{L}_1^1$, k = 1
$\ell_1^1 = \frac{1}{5}$, $u_1^1 = \frac{2}{5}$
$t_1^1 = \frac{1}{5}$
$\mathcal{L}_0^0$, k = 2
$\ell_0^0 = 0$, $u_0^0 = 1$
$t_0^0 = \frac{2}{5}$
$\mathcal{L}_0^2$, k = 1
$\ell_0^2 = \frac{3}{7}$, $u_0^2 = \frac{2}{7}$
$t_0^2 = \frac{1}{7}$
$\mathcal{L}_1^2$, k = 1
$\ell_1^2 = \frac{1}{7}$, $u_1^2 = \frac{2}{7}$
$t_1^2 = \frac{1}{7}$
$\mathcal{L}_0^1$, k = 2
$\ell_0^1 = \frac{3}{7}$, $u_0^1 = \frac{3}{7}$
$t_0^1 = \frac{1}{7}$
$\mathcal{L}_1^1$, k = 1
$\ell_1^1 = \frac{1}{7}$, $u_1^1 = \frac{3}{7}$
$t_1^1 = \frac{3}{14}$
$\mathcal{L}_0^0$, k = 3
$\ell_0^0 = 0$, $u_0^0 = 1$
$t_0^0 = \frac{5}{14}$
$\mathcal{L}_0^3$, k = 1
$\ell_0^3 = \frac{16}{43}$, $u_0^3 = \frac{32}{129}$
$t_0^3 = \frac{16}{129}$
$\mathcal{L}_1^3$, k = 1
$\ell_1^3 = \frac{16}{129}$, $u_1^3 = \frac{32}{129}$
$t_1^3 = \frac{16}{129}$
$\mathcal{L}_0^2$, k = 2
$\ell_0^2 = \frac{16}{43}$, $u_0^2 = \frac{16}{43}$
$t_0^2 = \frac{16}{129}$
$\mathcal{L}_1^2$, k = 1
$\ell_1^2 = \frac{16}{129}$, $u_1^2 = \frac{16}{43}$
$t_1^2 = \frac{8}{43}$
$\mathcal{L}_2^2$, k = 1
$\ell_2^2 = \frac{23}{129}$, $u_2^2 = \frac{40}{129}$
$t_2^2 = \frac{20}{129}$
$\mathcal{L}_3^2$, k = 1
$\ell_3^2 = \frac{20}{129}$, $u_3^2 = \frac{40}{129}$
$t_3^2 = \frac{20}{129}$
$\mathcal{L}_0^1$, k = 3
$\ell_0^1 = \frac{46}{129}$, $u_0^1 = \frac{22}{43}$
$t_0^1 = \frac{17}{129}$
$\mathcal{L}_1^1$, k = 2
$\ell_1^1 = \frac{17}{129}$, $u_1^1 = \frac{22}{43}$
$t_1^1 = \frac{23}{129}$
$\mathcal{L}_0^0$, k = 5
$\ell_0^0 = 0$, $u_0^0 = 1$
$t_0^0 = \frac{40}{129}$
$\mathcal{L}_4^3$, k = 1
$\ell_4^3 = \frac{15}{53}$, $u_4^3 = 0$
$t_4^3 = 0$
$\mathcal{L}_5^3$, k = 1
$\ell_5^3 = \frac{15}{53}$, $u_5^3 = 0$
$t_5^3 = 0$
$\mathcal{L}_6^3$, k = 1
$\ell_6^3 = \frac{15}{53}$, $u_6^3 = 0$
$t_6^3 = 0$
$\mathcal{L}_7^3$, k = 1
$\ell_7^3 = \frac{15}{53}$, $u_7^3 = 0$
$t_7^3 = 0$
$\mathcal{L}_0^3$, k = 1
$\ell_0^3 = \frac{18}{53}$, $u_0^3 = \frac{12}{53}$
$t_0^3 = \frac{6}{53}$
$\mathcal{L}_1^3$, k = 1
$\ell_1^3 = \frac{6}{53}$, $u_1^3 = \frac{12}{53}$
$t_1^3 = \frac{6}{53}$
$\mathcal{L}_2^2$, k = 2
$\ell_2^2 = \frac{15}{53}$, $u_2^2 = \frac{15}{53}$
$t_2^2 = 0$
$\mathcal{L}_3^2$, k = 2
$\ell_3^2 = \frac{15}{53}$, $u_3^2 = \frac{15}{53}$
$t_3^2 = 0$
$\mathcal{L}_0^2$, k = 2
$\ell_0^2 = \frac{18}{53}$, $u_0^2 = \frac{18}{53}$
$t_0^2 = \frac{6}{53}$
$\mathcal{L}_1^2$, k = 1
$\ell_1^2 = \frac{6}{53}$, $u_1^2 = \frac{18}{53}$
$t_1^2 = \frac{9}{53}$
$\mathcal{L}_1^1$, k = 4
$\ell_1^1 = \frac{11}{53}$, $u_1^1 = \frac{34}{53}$
$t_1^1 = \frac{4}{53}$
$\mathcal{L}_0^1$, k = 3
$\ell_0^1 = \frac{8}{53}$, $u_0^1 = \frac{34}{53}$
$t_0^1 = \frac{11}{53}$
$\mathcal{L}_0^0$, k = 7
$\ell_0^0 = 0$, $u_0^0 = 1$
$t_0^0 = \frac{15}{53}$

**Figure 5.8:** Optimal $k$-XOR merging trees in the circuit model for $k = 2, 3, 5$ and 7.

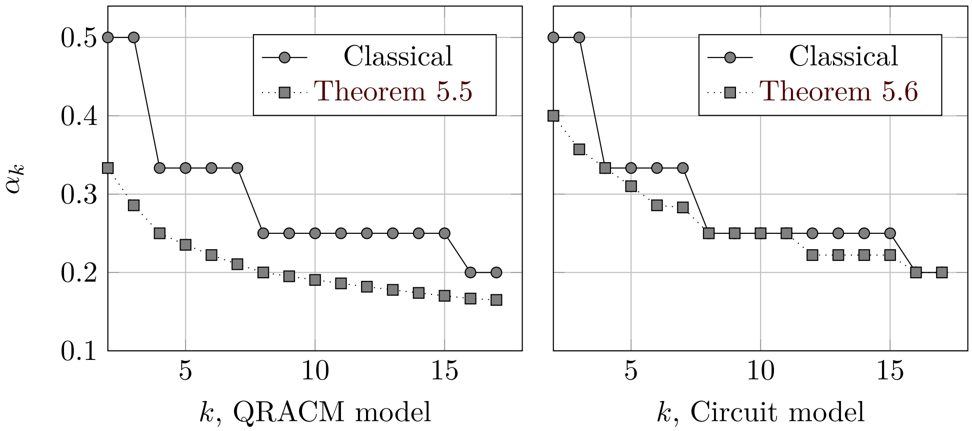


**Figure 5.9:** Comparison of time complexity exponents between the classical case and the best exponents given by Theorem 5.5 and Theorem 5.6. The complexities are $\mathcal{O}(2^{\alpha_k n})$ or $\mathcal{O}(2^{\beta_k n})$.

### 5.4.3 Proof of Optimality in the QRACM Setting

We prove Theorem 5.5:

> For any integer $k$ and $c > 0$, the best quantum merging procedure that *samples* $2^{cn}$ times a $k$-XOR on $n$ bits has a time complexity exponent $\max(\alpha_k(1+2c), c)$, where $\alpha_k = \frac{2^\kappa}{(1+\kappa)2^\kappa + k}$.

*Proof.* We use a recurrence on $k$. For $k = 2$, we have $\kappa = 1$ and $\alpha_2 = \frac{1}{3}$. Finding $2^{cn}$ collisions $(x_1, x_2, h(x_1) \oplus h(x_2) = 0)$, or *sampling a collision* $2^{cn}$ *times* can be done in time $2^{\left(\frac{2c}{3}+\frac{1}{3}\right)n}$, using a re-optimization of the steps in BHT collision search. This works as long as $c \leqslant 1$, i.e., $2c + 1 \leqslant 3$. Thus, the theorem is true for $k = 2$.

Let us consider a merging tree $\mathcal{T}_k$ for some $k > 2$, with a list size $c > 0$ at the root. The root node has two subtrees: the "list" one $\mathcal{T}_r$, on the right, and the "sampled" one $\mathcal{T}_l$, on the left. Let $u$ be the length of the zero-prefix in both nodes. Let $\ell_r$ and $\ell_l$ be their respective sizes, let $k_r + k_l = k$ be their width.

First, notice that we have $1 - u - \ell_r \geqslant 0$, otherwise we could reduce the value of the parameter $\ell_r$ without increasing the time complexity.

We use the recurrence hypothesis on $\mathcal{T}_l$ and $\mathcal{T}_r$, relatively to the number of zeros $u$ that they have (since they contain XORs on $un$ bits instead of $n$). The right list, of size $u\frac{\ell_r}{u}$, is produced in time $\max\left(\alpha_{k_r}(1 + 2\frac{\ell_r}{u}), \frac{\ell_r}{u}\right) u = \max\left(\alpha_{k_r}(u + 2\ell_r), \ell_r\right)$.

Since we want to sample $c$ times from the root node, we need to sample $c+\frac{1}{2}(1-u-\ell_r)$ times from the left list, which costs:

$$\max\left(\alpha_{k_l}\left(u + 2\left(c + \frac{1}{2}(1-u-\ell_r)\right)\right), \left(c + \frac{1}{2}(1-u-\ell_r)\right)\right) \quad .$$

We obtain that the time complexity exponent $t$ must be minimized under the constraints:

$$\begin{array}{ll} \text{(C1) } t \geqslant \alpha_{k_r}(u + 2\ell_r) & \text{(C2) } t \geqslant \ell_r \\ \text{(C3) } t \geqslant \alpha_{k_l}(2c + 1 - \ell_r) & \text{(C4) } t \geqslant c + \frac{1}{2} - \frac{u}{2} - \frac{\ell_r}{2} \end{array}$$

By combining these inequalities, we will obtain information about the shape of the optimal trees. We combine (C1), (C4) and (C3) to eliminate $u$ and $\ell_r$:

$$\text{(C1)} + 2\alpha_{k_r}\text{(C4)} + \frac{\alpha_{k_r}}{\alpha_{k_l}}\text{(C3)} \iff t\left(1 + 2\alpha_{k_r} + \frac{\alpha_{k_r}}{\alpha_{k_l}}\right) \geqslant 2\alpha_{k_r}(2c+1) \enspace .$$

Then this inequality becomes:

$$t \geqslant \frac{2\alpha_{k_r}}{1 + 2\alpha_{k_r} + \frac{\alpha_{k_r}}{\alpha_{k_l}}}(2c+1) = \frac{2\alpha_{k_r}\alpha_{k_l}}{\alpha_{k_l} + \alpha_{k_r} + 2\alpha_{k_r}\alpha_{k_l}}(2c+1) \enspace .$$

We are interested in the quantity $\frac{2\alpha_{k_r}\alpha_{k_l}}{\alpha_{k_l}+\alpha_{k_r}+2\alpha_{k_r}\alpha_{k_l}}$ when $k_l$ and $k_r$ vary. We would like to make it minimal, since this loosens the constraint on $t$. Thus, we want to maximize its inverse:

$$1 + \frac{1}{2\alpha_{k_l}} + \frac{1}{2\alpha_{k_r}} \enspace .$$

Since $\alpha_{k_l}$ is a decreasing function of $k_l$, this sum becomes maximal when $k_l$ is close to $k_r = k - k_l$. More precisely: if $k$ is even, then $k_r = k_l = \frac{k}{2}$ gives the smallest sum possible. If $k$ is odd, then $k_l = \left\lfloor \frac{k}{2} \right\rfloor$ and $k_r = k - \left\lfloor \frac{k}{2} \right\rfloor$ *or the converse.*

In both cases, if we write $\kappa = \lfloor \log_2 k \rfloor$, then $\lfloor \log_2 \lfloor k/2 \rfloor \rfloor = \lfloor \log_2(k - \lfloor k/2 \rfloor) \rfloor = \kappa - 1$. Using the recurrence hypothesis, we obtain that:

$$\begin{aligned} 1 + \frac{1}{2\alpha_{k_l}} + \frac{1}{2\alpha_{k_r}} &= 1 + \frac{(1+\kappa-1)2^{\kappa-1} + k - \left\lfloor \frac{k}{2} \right\rfloor}{2^\kappa} + \frac{(1+\kappa-1)2^{\kappa-1} + \left\lfloor \frac{k}{2} \right\rfloor}{2^\kappa} \\ &= \frac{2^\kappa(1+\kappa) + k}{2^\kappa} \enspace . \end{aligned}$$

Thus, we can write: $t \geqslant (2c+1)\frac{2^\kappa}{2^\kappa(1+\kappa)+k}$, which gives the expected formula for $\alpha_k$. The second inequality $t \geqslant c$ stems trivially from (C4). We finish the proof of optimality by showing, also by induction on $k$, that the optimization of the balanced trees $T_k$ indeed reaches this exponent.

**Lemma 5.5.** *Optimizing the balanced trees $T_k$ yields the optimal exponents of Theorem 5.5.*

First, we focus on the case $c \leqslant \frac{\alpha_k}{1-2\alpha_k}$, where the complexity exponent is expected to be $(2c+1)\alpha_k$, and we consider an even $k$. We choose $u = (1-3\alpha_k)(2c+1)$ and $\ell_r = \alpha_k(2c+1)$. This gives that $c + \frac{1}{2} - \frac{u}{2} - \frac{\ell_r}{2} = (2c+1)\alpha_k$, so (C4) is satisfied. Second, we have:

$$\alpha_{k/2}(u + 2\ell_r) = (2c+1)\alpha_{k/2}(1-\alpha_k) = (2c+1)\alpha_k$$

by definition of the $\alpha_k$ (their formula implies $\frac{\alpha_{k/2}}{1+\alpha_{k/2}} = \alpha_k$). Thus (C1) is satisfied. By a similar computation, (C3) is satisfied since $\alpha_{k/2}(2c+1-\ell_r) = \alpha_k(2c+1)$. Finally, (C2) is trivially satisfied by our choice of $\ell_r$.

If $k$ is odd, we choose

$$u = \left(1 - 3\alpha_k + \frac{1}{(1+\kappa)2^\kappa + k}\right)(2c+1) \text{ and } \ell_r = \left(\alpha_k - \frac{1}{(1+\kappa)2^\kappa + k}\right)(2c+1) \enspace .$$

Again, (C4) becomes an equality. (C1) is an equality as well, using the fact that $\alpha_{k_r} = \alpha_{\lfloor k/2 \rfloor} = \alpha_{(k-1)/2}$. Indeed, we have:

$$\begin{aligned}\alpha_{k_r}(u + 2\ell_r) &= \alpha_{(k-1)/2}\left(1 - \alpha_k - \frac{1}{(1+\kappa)2^\kappa + k}\right) \\ &= (2c+1)\frac{2^{\kappa-1}}{\kappa 2^{\kappa-1} + (k-1)/2}\left(\frac{2^\kappa(1+\kappa) + k - 2^\kappa - 1}{(1+\kappa)2^\kappa + k}\right) = (2c+1)\alpha_k \enspace .\end{aligned}$$

The constraints (C2) and (C3) become strict inequalities, but they are also satisfied.

When $c \geqslant \frac{\alpha_k}{1 - 2\alpha_k}$, all the merges become classical. The only quantum operations remaining are the Grover searches in some newly inserted leaves. □

### 5.4.4 Proof of Optimality in the Circuit Model

We prove Theorem 5.6:

> For any integer $k \geqslant 8$ and $c > 0$, the best quantum merging procedure *without QRACM* that *samples* $2^{cn}$ times a $k$-XOR on $n$ bits has a time complexity exponent $\max(\beta_k(1+c), c)$, where:
>
> $$\beta_k = \begin{cases} \frac{1}{\kappa+1} \text{ if } k < 2^\kappa + 2^{\kappa-1} \\ \frac{2}{2\kappa+3} \text{ if } k \geqslant 2^\kappa + 2^{\kappa-1} \end{cases} \enspace .$$
>
> And when $k \geqslant 8$, this procedure samples *classically.*

*Proof.* We prove this by induction on $k$. For small values of $k$, the experimental results give us the optimal trees. We consider a merging tree $\mathcal{T}_k$ for $k \geqslant 8$, with a list size $c > 0$ at the root. We use the same notations as in Section 5.4.3, and introduce $k_l, k_r, \beta_{k_l}, \beta_{k_r}$ and the variables $u, \ell_r$.

For $k \leqslant 7$, we notice that the exponent is always *at least* $\max(\beta_k(1+c), c)$ (it will lie somewhere between $\beta_k(1+c)$ and $\beta_k(1+2c)$). Having $+c$ instead of $+2c$ comes from the use of a classical merging at the root. Once we know that the merge is classical, we can deduce easily that both subtrees must be of similar shapes, hence $k_\ell = \left\lfloor \frac{k}{2} \right\rfloor$ and $k_r = \left\lceil \frac{k}{2} \right\rceil$ or the converse. Then we can use the recurrence hypothesis easily: the two subtrees have the same complexity, which depends on the case for $k$. If $k < 2^\kappa + 2^{\kappa-1}$, then $\left\lfloor \frac{k}{2} \right\rfloor < 2^{\kappa-1} + 2^{\kappa-2}$; and conversely, if $k \geqslant 2^\kappa + 2^{\kappa-1}$, then $\left\lfloor \frac{k}{2} \right\rfloor \geqslant 2^{\kappa-1} + 2^{\kappa-2}$.

In order to prove that the root merge is classical, let us assume that it is quantum instead. We use the recurrence hypothesis for both subtrees. Although the actual optimal merging trees do not allow to sample quantumly, we suppose that they do. Thus, sampling $c + \frac{1}{2}(1 - u - \ell_r)$ times from the left child is done in time $\max(c + \frac{1}{2}(1 - u -$

$\ell_r), \beta_{k_l}\left(u + c + \frac{1}{2}(1 - u - \ell_r)\right))$. We also do the same number of QRACM emulations, in time $\ell_r$ each. With the right child, we have at least $\max(\beta_{k_r}(u + \ell_r), \ell_r)$ (notice that this is not tight for small $k_r$). Let $t$ be the time exponent, then we have the constraints:

$$\text{(C1) } t \geqslant \beta_{k_r}(u + \ell_r) \qquad \text{(C2) } t \geqslant \frac{\beta_{k_l}}{2}(2c + 1 + u - \ell_r)$$
$$\text{(C3) } t \geqslant \frac{1}{2}\left(2c + 1 - u + \ell_r\right)$$

By combining (C2) and (C3), we obtain:

$$\left(\frac{2}{\beta_{k_l}} + 2\right) t \geqslant 2(2c+1) \implies t \geqslant \frac{\beta_{k_l}}{1 + \beta_{k_l}}(2c+1) = \beta_{2k_l}(2c+1) \ ,$$

where the last equality follows by definition of the $\beta_i$. But since $t \leqslant \beta_{k-1}(c+1)$, we obtain that $\beta_{k-1} \geqslant \beta_{2k_l} \implies 2k_l \geqslant k - 1 \implies k_l \geqslant \lfloor k/2 \rfloor$.

Furthermore, at the optimal point we expect:

$$\frac{\beta_{k_l}}{2}(2c + 1 + u - \ell_r) = \frac{1}{2}(2c + 1 - u + \ell_r) \implies u = \ell_r + \frac{\beta_{k_l} - 1}{\beta_{k_l} + 1}(2c+1) \ .$$

Next, we remark that an algorithm in the circuit model should cost at least as much as in the QRACM model, so we introduce:

$$\text{(C4)} \quad t \geqslant \alpha_{k_r}(u + 2\ell_r) \implies t \geqslant \alpha_{k_r}\left(3\ell_r + \frac{\beta_{k_l} - 1}{\beta_{k_l} + 1}(2c+1)\right) \ .$$

Since $u \geqslant 0$, we should have $\ell_r \geqslant \frac{1 - \beta_{k_l}}{\beta_{k_l} + 1}(2c+1)$. But then we find $t \geqslant 2\alpha_{k_r}\frac{\beta_{k_l} - 1}{\beta_{k_l} + 1}(2c+1)$.

Since we have $k_l \geqslant \lfloor k/2 \rfloor$, at the same time, we should have $k_r \leqslant \lceil k/2 \rceil$ so $k_r \leqslant k_l + 1$ and $\alpha_{k_r} \geqslant \alpha_{k_l+1}$. This inequality becomes $t \geqslant 2\alpha_{k_l+1}\frac{1-\beta_{k_l}}{\beta_{k_l}+1}(2c+1)$. A quick computation of the first values of $\alpha_i$ and $\beta_i$ shows that for $k_l \geqslant 15, 2\alpha_{k_l+1}\frac{1-\beta_{k_l}}{\beta_{k_l}+1} \geqslant \beta_{k_l+1}$. In other words, while small values of $k$ may benefit from using a quantum search at the root of the tree (and this is indeed the case), for a general $k$, the root node is a classical merge between two classically stored lists.

Again, we can verify that the balanced trees $T_k$ give the optimal results for $k \geqslant 8$. An example is given in Figure 5.10. □

## 5.5 Applications

The algorithms that we presented in this chapter can efficiently replace Wagner's algorithm when $k$ *is a constant* and when quantum query access is allowed. The second condition actually occurs very often, e.g., when attacking hash functions. However, the first is not always satisfied. Some applications of Wagner's algorithm merge $\mathcal{O}\left(2^{\sqrt{n}}\right)$ lists of size $\mathcal{O}\left(2^{\sqrt{n}}\right)$ in subexponential time, a situation in which quantum merging does not seem to bring any advantage.

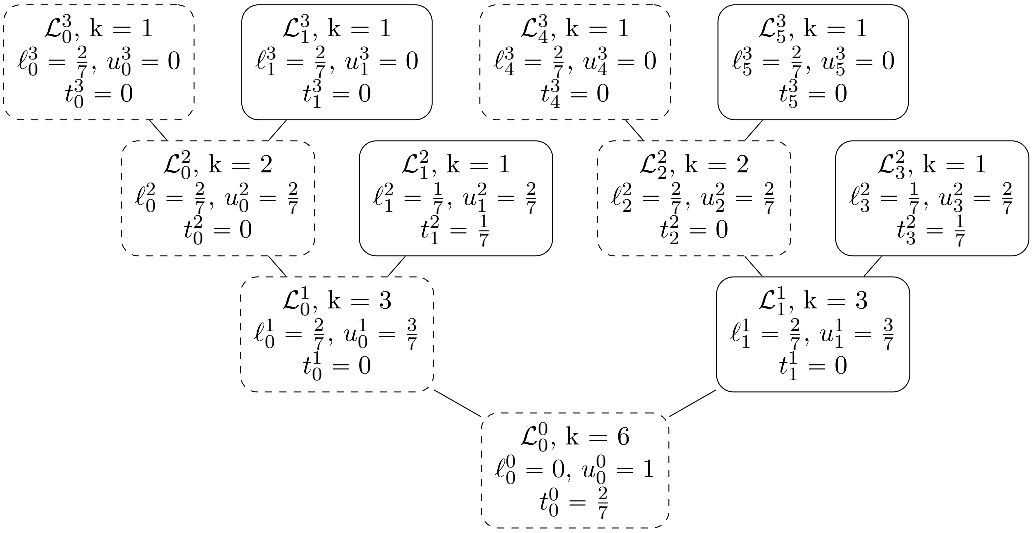

$\mathcal{L}_0^3$, k = 1
$\ell_0^3 = \frac{2}{7}$, $u_0^3 = 0$
$t_0^3 = 0$
$\mathcal{L}_1^3$, k = 1
$\ell_1^3 = \frac{2}{7}$, $u_1^3 = 0$
$t_1^3 = 0$
$\mathcal{L}_4^3$, k = 1
$\ell_4^3 = \frac{2}{7}$, $u_4^3 = 0$
$t_4^3 = 0$
$\mathcal{L}_5^3$, k = 1
$\ell_5^3 = \frac{2}{7}$, $u_5^3 = 0$
$t_5^3 = 0$
$\mathcal{L}_0^2$, k = 2
$\ell_0^2 = \frac{2}{7}$, $u_0^2 = \frac{2}{7}$
$t_0^2 = 0$
$\mathcal{L}_1^2$, k = 1
$\ell_1^2 = \frac{1}{7}$, $u_1^2 = \frac{2}{7}$
$t_1^2 = \frac{1}{7}$
$\mathcal{L}_2^2$, k = 2
$\ell_2^2 = \frac{2}{7}$, $u_2^2 = \frac{2}{7}$
$t_2^2 = 0$
$\mathcal{L}_3^2$, k = 1
$\ell_3^2 = \frac{1}{7}$, $u_3^2 = \frac{2}{7}$
$t_3^2 = \frac{1}{7}$
$\mathcal{L}_0^1$, k = 3
$\ell_0^1 = \frac{2}{7}$, $u_0^1 = \frac{3}{7}$
$t_0^1 = 0$
$\mathcal{L}_1^1$, k = 3
$\ell_1^1 = \frac{2}{7}$, $u_1^1 = \frac{3}{7}$
$t_1^1 = 0$
$\mathcal{L}_0^0$, k = 6
$\ell_0^0 = 0$, $u_0^0 = 1$
$t_0^0 = \frac{2}{7}$

**Figure 5.10:** Optimal merging tree for 6-XOR (circuit model).

### 5.5.1 Generalized Birthday Instances

We can list a few constructions that can be attacked with a $k$-XOR or $k$-SUM algorithm. Others are listed in [Wag02].

**Incremental hash functions.** XHASH [BM97] is an incremental hash function defined as $H(x) = \bigoplus_{i=1}^{k} h(i|x_i)$, where each $x_i$ is a $b$-bit block and $h(\cdot) : \{0,1\}^\ell \to \{0,1\}^n$ for some parameters $b$ and $\ell$. One can find collisions in XHASH, when restricted to $k$ blocks, with a $k$-XOR algorithm [BM97; CJ04].

Among other designs, this construction appears in the “fast syndrome-based” hash function (R)FSB [Aug+], a candidate of the SHA-3 competition, which has been analyzed in [CJ04; Ber+09; Kir11; NCB11].

SWIFFT [Mic+] is a candidate of the SHA-3 competition that admits a simple algebraic expression over some polynomial ring $R = \mathbb{Z}_p[\alpha]/(\alpha^n + 1)$:

$$\text{SWIFFT}(a, x) = \sum_{i=1}^{m} f(x_i) \bmod (\alpha^n + 1) = \sum_{i=1}^{m} (a_i \cdot x_i) \bmod (\alpha^n + 1) \ ,$$

where the $m$ fixed elements $a_1, ..., a_m \in R$ – called multipliers – specify the hash function, and each $x_i$ is an element of $R$. Examples of attacks on the SWIFFT hash function based on the $k$-SUM problem over the additive group $(\mathbb{Z}/2^{256}\mathbb{Z}, +)$ are given in [Kir11; NCB11; Bai+19].

**XLS and CAESAR candidates.** Some authenticated encryption schemes proposed at the CAESAR competition [CAESAR] can be attacked with $k$-XOR algorithms, namely those that use the XLS construction of Ristenpart and Rogaway [RR07]. Although

XLS was initially proven secure, Nandi [Nan14] pointed out flaws in the security proof and showed a very simple attack that required three queries to break the construction. Actually, the CAESAR candidates that rely on XLS do not allow this trivial attack as the required decryption queries are not permitted by the schemes. A possible way to overcome this limitation has been proposed by Nandi in [Nan14], whose forgery attack requires only encryption queries. It reduces to the 3-XOR problem. We refer to [Nan15; NS15] for concrete examples of attacks.

With the algorithms presented in this chapter, Q2 access will be necessary to obtain a speedup. However, it is also possible to find a 3-XOR on $n$ bits in quantum time $\mathcal{O}\left(2^{n/3}\right)$ with only $\mathcal{O}\left(2^{n/3}\right)$ classical queries, and this is effective against XLS.

### 5.5.2 Approximate k-list Problem

In [BM17], Both and May introduce and study the *approximate k-list problem.* It is a generalization of $k$-XOR in which the final $n$-bit value only needs to have a Hamming weight lower than $\alpha n$ for some fraction $0 \leqslant \alpha \leqslant \frac{n}{2}$ (so the $k$-XOR is the special case $\alpha = 0$). Its main application is solving the parity check problem.

**Problem 5.3** (Parity-check)**.** *Given an irreducible polynomial $P(X) \in \mathbb{F}_2[X]$ of degree $n$, find a multiple $Q(X)$ of $P(X)$ of a certain weight and degree.*

This is used in fast correlation attacks on stream ciphers. For this application, we can consider quantum oracle access (the lists actually contain polynomials of the form $X^a \mod P(X)$ for many choices of $a$).

The match-and-filter algorithm of [BM17, Section 3] consists in running a $k$-XOR algorithm with a restricted number of bits to put to zero, and to tailor the length of the final list so that it will contain one element of low Hamming weight with certainty. With a quantum $k$-merging tree, we can always improve on this classical method in the QRACM model. Let $\alpha_k$ be the $k$-XOR optimal QRACM time exponent as defined in Theorem 5.5. We cut the left branch of the tree: in time $\mathcal{O}(2^{\alpha_k un})$, we can obtain a tuple of lists $\mathcal{L}_1, \ldots, \mathcal{L}_t$ such that, given an $n$-bit element $x$, we can find $x_1 \in \mathcal{L}_1, \ldots, x_t \in \mathcal{L}_t$ such that $x \oplus x_1 \oplus \ldots \oplus x_t$ has $(1 - 2\alpha_k)un$ bits to zero. Indeed, the Grover search at the root of the tree has also cost $\mathcal{O}(2^{\alpha_k un})$ since everything is balanced, so it eliminates $2\alpha_k un$ bits.

Hence, if we want to be able to eliminate $un$ bits for some fraction $0 \leqslant u \leqslant 1$, we build all these lists in time $\mathcal{O}\left(2^{\frac{\alpha_k}{(1-2\alpha_k)}un}\right)$.

**Proposition 5.4.** *We can build in time $\mathcal{O}\left(2^{\frac{\alpha_k}{(1-2\alpha_k)}un}\right)$ a structure that allows, in constant time, to sample a $k$-tuple with $un$ bits to zero.*

There remains $(1 - u)n$ (random) bits. We want the Hamming weight of the result to be less than a target $c_w n$. The proportion of $(1 - u)n$-bit strings of Hamming weight less than $c_w n$ is approximately:

$$\binom{(1-u)n}{c_w n} / 2^{(1-u)n} \simeq 2^{(1-u)n(\mathrm{H}(c_w/(1-u))-1))}$$

**Table 5.2:** Quantum speedup of the approximate $k$-list problem of [BM17], in the QRACM model.

| $k = 2$ | | | $k = 3$ | | |
|---|---|---|---|---|---|
| $c_w$ | $\log T/n$ (classical) | $\log T/n$ (quantum) | $c_w$ | $\log T/n$ (classical) | $\log T/n$ (quantum) |
| 0 | 0.5000 | 0.3333 | 0 | 0.5000 | 0.2857 |
| 0.1 | 0.2920 | 0.1876 | 0.1 | 0.2769 | 0.1641 |
| 0.2 | 0.1692 | 0.1046 | 0.2 | 0.1590 | 0.0935 |
| 0.3 | 0.0814 | 0.0481 | 0.3 | 0.0778 | 0.0440 |
| 0.4 | 0.0232 | 0.0129 | 0.4 | 0.0221 | 0.0122 |

| $k = 8$ | | | $k = 1024$ | | |
|---|---|---|---|---|---|
| $c_w$ | $\log T/n$ (classical) | $\log T/n$ (quantum) | $c_w$ | $\log T/n$ (classical) | $\log T/n$ (quantum) |
| 0 | 0.2500 | 0.2000 | 0 | 0.1667 | 0.1429 |
| 0.1 | 0.1576 | 0.1200 | 0.1 | 0.1091 | 0.0889 |
| 0.2 | 0.0984 | 0.0714 | 0.2 | 0.0704 | 0.0548 |
| 0.3 | 0.0518 | 0.0355 | 0.3 | 0.0387 | 0.0284 |
| 0.4 | 0.0170 | 0.0106 | 0.4 | 0.0914 | 0.0091 |

if $c \leqslant (1-u)$ and 1 otherwise, where H is the binary entropy function. Hence, we run a quantum search with: $2^{\frac{1}{2}(1-u)n(1-\mathrm{H}_e(c_w/(1-u))))}$ iterations, where $\mathrm{H}_e$ is an "extended" entropy function that gives $\mathrm{H}_e(x) = 0$ if $x \geqslant 1$. It suffices to look for $0 \leqslant u \leqslant 1$ which optimizes the sum of the time complexities of the two steps:

$$2^{\frac{\alpha_k}{(1-2\alpha_k)}un} + 2^{\frac{1}{2}(1-u)n(1-\mathrm{H}_e(c_w/(1-u))))} .$$

We obtain the results of Table 5.2 by numerical optimization.

### 5.5.3 Open Questions

The presentation of merging trees that we gave in this chapter is slightly extended with respect to [NS20], but for a general $k$, we found the same results. Our proof of optimality holds only in this framework. With an extended framework, perhaps combining quantum walks instead of quantum searches, it might be possible to improve the complexities.

**Open Problem 5.4.** *Does there exist better algorithms for k-XOR than our merging trees, that would reach exponentially better time complexities?*

Finally, we have focused on the problem for constant $k$. But when $k$ is unbounded, Wagner's algorithm achieves a subexponential time $\mathcal{O}(2^{2\sqrt{n}})$ using $k = \mathcal{O}(2^{\sqrt{n}})$ and $\sqrt{n}$ merging levels. In that case, the quantum advantage of merging algorithms vanishes, and no significant improvement is known.

**Open Problem 5.5.** *Does there exist a quantum algorithm for the k-SUM problem with unbounded $k$, with a significant speedup over Wagner's algorithm?*

# Chapter 6
# Solving the k-XOR Problem with a Single Solution

In this chapter, we amend the merging trees defined in Chapter 5 in order to solve $k$-XOR problems *with any number of solutions*. We show that both classical and quantum methods for such problems can be described in terms of *extended merging trees*. Focusing on the $k$-XOR problem with a single solution, we give strategies for every $k$ and prove their optimality among all merging trees. We study various cryptanalytic applications, including subset-sums, the generalization for $r$-composite problems and $r$-encryption. Some of the results that we present here are original and unpublished, as we introduce a more general definition of merging trees than in [NS20], which allows us to improve these previous results.

**Contents**

## 6.1 Extending the Problem

In Chapter 5, we studied the $k$-XOR problem with *many solutions*:

**Problem 5.2** ($k$-XOR with an oracle). *Given oracle access to a random $n$-bit to $n$-bit function $h$, find distinct inputs $x_1, \ldots, x_k$ such that $h(x_1) \oplus \ldots \oplus h(x_k) = 0$.*

Indeed, when $k$ is a constant and $h$ a random function, there are expectedly $2^{(k-1)n}$ distinct solution tuples $x_1, \ldots, x_k$. But the applications of Problem 5.2 for a constant $k$ are relatively rare.

However, many problems of cryptographic interest can be modeled as a variant of Problem 5.2 with *limited domain*, where $h$ is a random function from $\{0,1\}^{dn}$ to $\{0,1\}^n$ for some parameter $d \leqslant 1$. Of special interest is the limit case where there is expectedly a single solution: we name it the *Unique k-XOR problem*.

**Problem 6.1** (Unique $k$-XOR with an oracle)**.** *Given query access to a random $\lceil n/k \rceil$-bit to $n$-bit function $h$, expecting that there exists a single $k$-tuple $x_1, \ldots, x_k$ such that $h(x_1) \oplus h(x_2) \oplus \ldots \oplus h(x_k) = 0$, find it.*

If $k$ is even, we must enforce the solution to be non-trivial: there does not exist a subset $X'$ of the $x_i$ of size $k/2$ which is equal to $\{x_1, \ldots, x_k\} \backslash X'$. The alternative with lists, accessed only classically, is a very close problem that we will also study.

**Problem 6.2** (Unique $k$-XOR with lists)**.** *Given classical data as $k$ lists $L_1, \ldots, L_k$ of uniformly random $n$-bit strings, of size $2^{\lceil n/k \rceil}$, find a $k$-tuple $x_1, \ldots, x_k \in L_1 \times \ldots \times L_k$ such that $x_1 \oplus \ldots \oplus x_k = 0$, if it exists.*

In the following, we will assume that $n$ is a multiple of $k$, writing $\frac{n}{k}$ instead of $\left\lceil \frac{n}{k} \right\rceil$ for the sake of simplicity, and without incidence on the complexities.

### 6.1.1 Classical Algorithms

We will first recall classical algorithms that solve Problem 6.1. As in the previous chapter, their complexities are exponential in $n$.

**Naive meet-in-the-middle.** A first idea is to cut the problem into halves (Algorithm 6.1). We compute all sums of $\left\lfloor \frac{k}{2} \right\rfloor$ inputs, store them and look among all tuples of $k - \left\lfloor \frac{k}{2} \right\rfloor = \left\lceil \frac{k}{2} \right\rceil$ inputs for a collision on the stored data. The algorithm uses $\mathcal{O}\left(2^{\left\lfloor \frac{k}{2} \right\rfloor \frac{n}{k}}\right)$ memory and $\mathcal{O}\left(2^{\left\lceil \frac{k}{2} \right\rceil \frac{n}{k}}\right)$ time.

To date, there does not exist any generic improvement on this time complexity for the Unique $k$-XOR problem and all variants explored in this chapter, except for logarithmic factors. However, many classical algorithms allow to improve significantly on the *memory complexity*.

**Parallel collision search.** The case $k = 2$ in Problem 6.1 arises in the literature under many names, such as *golden collision* search or *element distinctness*. The formulation of *unique* collision search is Problem 6.1: we are given a function $h : \{0,1\}^{n/2} \to \{0,1\}^n$, we want to find its only (expected) collision. With the *element distinctness* problem, we want to know whether $h$ has such a collision or not. With the *claw-finding* variant, we

**Algorithm 6.1** Naive meet-in-the-middle $k$-XOR algorithm.

```
Input: oracle access to $h : \{0,1\}^{n/k} \mapsto \{0,1\}^n$
Output: $x_1, \ldots, x_k$ such that $h(x_1) \oplus \ldots \oplus h(x_k) = 0$.
Create a sorted list $\mathcal{L}$ of size $2^{\lfloor \frac{k}{2} \rfloor \frac{n}{k}}$ that contains:
    $\forall (x_1, \ldots, x_{\lfloor k/2 \rfloor}) \in \left(\{0,1\}^{n/k}\right)^{\lfloor k/2 \rfloor}, (x_1, \ldots, x_{\lfloor k/2 \rfloor}, h(x_1) \oplus \ldots \oplus h(x_{\lfloor k/2 \rfloor}))$
for all $(x_{\lfloor k/2 \rfloor + 1}, \ldots, x_k) \in \left(\{0,1\}^{n/k}\right)^{k - \lfloor k/2 \rfloor}$ do
    if $\exists (x_1, \ldots, x_{\lfloor k/2 \rfloor}, y) \in \mathcal{L}, h(x_{\lfloor k/2 \rfloor + 1}) \oplus \ldots \oplus h(x_k) = y$ then
        return $(x_1, \ldots, x_{\lfloor k/2 \rfloor}, x_{\lfloor k/2 \rfloor + 1}, \ldots, x_k)$
```

consider *two* functions $g$ and $h$, looking for $x$ such that $g(x) = h(x)$, given the promise that there exists a single one or none.

The cryptanalytic applications are rich. For example, the best attack on the isogeny-based key-exchange SIKE [Jao+17], one of the candidates of the NIST PQC project, reduces to a claw-finding problem.

While the naive algorithm solves the unique 2-XOR problem in time $\mathcal{O}(2^{n/2})$ and memory $\mathcal{O}(2^{n/2})$, it achieves only a time-memory trade-off curve $T \cdot M = 2^n$. The *parallel collision search* algorithm of van Oorschot and Wiener [vOW99] improves this to $T \cdot M^{1/2} = 2^{3n/4}$.

**Schroeppel and Shamir's algorithm.** In [SS81], Schroeppel and Shamir give an algorithm that improves the memory usage over the naive 2-list merge, whenever the problem can be rephrased as a 4-list merging problem. Rephrased in the setting of Problem 6.1, and reformulated in the merging framework, their idea is to merge 4 lists pairwise, with an *arbitrary* constraint $s$ of $\frac{n}{4}$ bits. By Lemma 5.1, the whole merging tree can be computed in time $\mathcal{O}(2^{n/4})$ and memory $\mathcal{O}(2^{n/4})$. However, as there is only one solution, we must try all possible prefixes $s$ before finding a non-empty list at the root.

**Algorithm 6.2** Schroeppel and Shamir's algorithm.

```
Input: oracle access to $h : \{0,1\}^{n/4} \to \{0,1\}^n$
Output: $x_1, x_2, x_3, x_4$ such that $h(x_1) \oplus h(x_2) \oplus h(x_3) \oplus h(x_4) = 0$
Create 4 lists $\mathcal{L}_i, 1 \leqslant i \leqslant 4$, of size $2^{n/4}$, where pairs $x, h(x)$ have arbitrary indices
for all Prefix $s$ of $\frac{n}{4}$ bits do
    $\mathcal{L}_{12}^s \leftarrow \textsc{Merge}(\mathcal{L}_1, \mathcal{L}_2, s)$        ▷ $\mathcal{L}_{12}^s$ is of average size $\frac{2^{n/4} \times 2^{n/4}}{2^{n/4}} = 2^{n/4}$
    $\mathcal{L}_{34}^s \leftarrow \textsc{Merge}(\mathcal{L}_3, \mathcal{L}_4, s)$
    if there is a collision between $\mathcal{L}_{12}$ and $\mathcal{L}_{34}$ then
                                        ▷ Happens for a single $s$ (or with probability $2^{-n/4}$
        return The collision
```

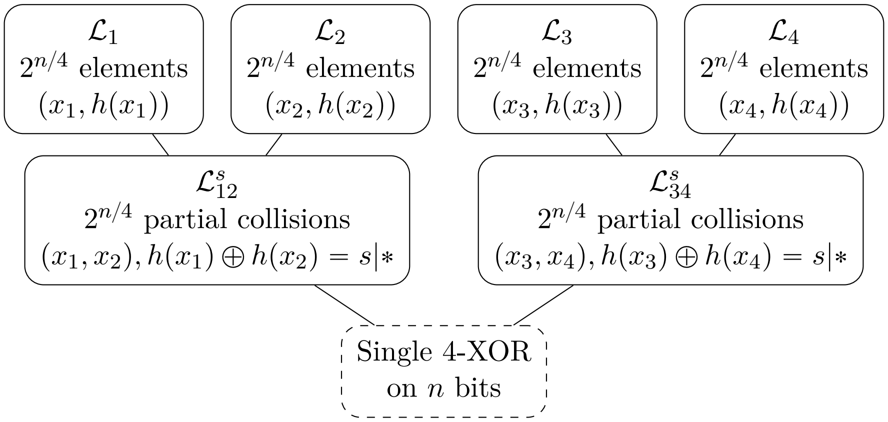


**Figure 6.1:** Structure of Schroeppel and Shamir's merging tree.

The merging tree of Schroeppel and Shamir's algorithm is given in Figure 6.1. It has the same structure as the naive merging tree, which would simply put in $\mathcal{L}_{12}$ the product of $\mathcal{L}_1$ and $\mathcal{L}_2$ (and respectively for $\mathcal{L}_{34}$) as it needs to remember all the possibilities. It is *contained* in the naive tree, in the sense that all lists are contained in the corresponding naive lists.

**Dissection algorithms.** The Dissection Algorithms of Dinur, Dunkelman, Keller, and Shamir [Din+12] generalize both Schroeppel and Shamir's technique and the parallel collision search technique to any $k$. In [Din+12], they are formulated in the framework of *bicomposite problems*, which is more general than unique $k$-XOR. Their method consists in guessing some intermediate values, then producing efficiently lists of partial guesses, before matching them. A bigger meet-in-the-middle instance is broken down into smaller ones. The algorithms of [Din+12, Section 3] correspond, in the classical setting, to the *extended merging trees* that we will shortly define. The algorithms of [Din+12, Section 4] improve the memory usage with parallel collision search. However, their power comes from iterating a random function, and we have seen in Chapter 5 that in the quantum setting, the advantage of iterating vanishes with respect to the quadratic speedup of quantum search, which is why we won't consider them for a quantum improvement.

**Other improvements.** The other works that we considered for the $k$-XOR problem and other merging problems [MS12; Bai+19; Din19; NS15] do not seem to perform better than Dinur, Dunkelman, Keller, and Shamir for the Unique $k$-XOR problem, and none of the techniques seemed to improve over the quantum merging algorithms that we will detail in the next section.

### 6.1.2 Quantum Algorithms

Despite its cryptanalytic applications, the Unique $k$-XOR problem has not been previously studied in full generality in the quantum setting. However, the *element distinctness problem* received significant attention, and an algorithm for $k = 4$ was also known. In both cases, the best time complexity is obtained using a *quantum walk*.

**Element distinctness.** In [Buh+05], Buhrman et al. give an algorithm running in time $\mathcal{O}\left(2^{3n/8}\right)$, with memory (QRAQM) $\mathcal{O}\left(2^{n/4}\right)$. This algorithm is simply based on quantum search and can be reframed as a 2-list merging algorithm with a repeating loop.

**Algorithm 6.3** Element distinctness (unique 2-XOR) algorithm of Buhrman et al.

```
Input: oracle access to $h : \{0,1\}^{n/2} \to \{0,1\}^n$
Output: $x, y$ such that $h(x) \oplus h(y) = 0$
Repeat $2^{n/4}$ times                                   ▻ $2^{n/8}$ quantum search iterates
    Select a subset of size $2^{n/4}$ of the inputs of $h$
                                                         ▻ e.g., using a prefix
    Fill a list $\mathcal{L} = (x, h(x))$ of size $2^{n/4}$
    With Grover's algorithm, look for $y \in \{0,1\}^n$ such that $\exists x, (x, h(y)) \in \mathcal{L}$
                                                         ▻ The search requires $\mathcal{O}\left(2^{n/4}\right)$ iterations
    if there is a solution $y$ then
        return $(x, y)$
EndRepeat
```

**Ambainis' algorithm.** Ambainis' celebrated algorithm [Amb07], based on a quantum walk, finds the solution in time $\widetilde{\mathcal{O}}\left(2^{2n/3}\right)$ using $2^{2n/3}$ QRAQM. For a random oracle, it cannot be exponentially improved, as this would yield a better algorithm for collision search than the proven lower bound $\Omega(2^{n/3})$. With a memory restricted to $M$, both Ambainis' algorithm and the previous one move on the time-memory trade-off curve $T^2 \cdot M = 2^{2n}$. This is also the case of the algorithms for golden collision search that we give in [JS20], which allow to decrease the time complexity down to $\widetilde{\mathcal{O}}\left(2^{6n/7}\right)$ with plain quantum circuits, and without QRAQM.

**4-List merge with a quantum walk.** In [Ber+13], Bernstein, Jeffery, Lange, and Meurer give a quantum walk algorithm for 4-SUM. It can solve Problem 6.1 in time $\widetilde{\mathcal{O}}\left(2^{0.3n}\right)$, using $\widetilde{\mathcal{O}}\left(2^{0.2n}\right)$ QRAQM. This represents an exponential quantum time and memory improvement with respect to $k = 2$, which shows once again that quantum merging algorithms differ from their classical cousins.

**3-List merge with a Grover search.** For $k = 3$, we can give Algorithm 6.4. It gives actually the best quantum speedup known, for all $k$, when QRACM only is available.

**Algorithm 6.4** Quantum algorithm for the Unique 3-XOR problem, with QRACM.

```
Input: oracle access to $h : \{0,1\}^{n/3} \to \{0,1\}^n$
Output: $x_1, x_2, x_3$ such that $h(x_1) \oplus h(x_1) \oplus h(x_3) = 0$
Store all $(x_1, h(x_1))$ in a list $\mathcal{L}$ of size $2^{n/3}$ (in QRACM)
Sample $x_2, x_3 \in (\{0,1\}^{n/3})^2$ such that
    if $\exists x_1, (x_1, h(x_2) \oplus h(x_3)) \in \mathcal{L}$ then
        return $x_2, x_3$
EndSample
```

This algorithm solves Unique 3-XOR in time and memory $\mathcal{O}\left(2^{n/3}\right)$. When the data is accessed via an oracle, and in the QRAQM model, we can reduce the memory down to $2^{n/6}$ with Algorithm 6.5.

**Algorithm 6.5** Quantum algorithm for the Unique 3-XOR problem, with QRAQM.

```
Input: oracle access to $h : \{0,1\}^{n/3} \to \{0,1\}^n$
Output: $x_1, x_2, x_3$ such that $h(x_1) \oplus h(x_1) \oplus h(x_3) = 0$
Sample $s \in \{0,1\}^{n/6}$ such that
    Build a list $\mathcal{L}_2$ of $2^{n/6}$ elements with prefix $s$, using Grover search in time $2^{n/12} \times 2^{n/6} = 2^{n/4}$
    Repeat $2^{n/6}$ times
        Build a list $\mathcal{L}_1$ of $2^{n/6}$ elements by querying $h$
        Sample $x_3 \in \{0,1\}^{n/3}$ such that
            Find $(x_1, h(x_1)) \in \mathcal{L}_1$ with $h(x_1) \oplus h(x_3) = s|*$
            if there exists $(x_2, h(x_2)) \in \mathcal{L}_2$ such that $h(x_1) \oplus h(x_3) \oplus h(x_2)$ then
                return $(x_1, x_2, x_3)$
        EndSample
    EndRepeat
EndSample
```

The time complexity of Algorithm 6.5 is:

$$\underbrace{2^{n/12}}_{\text{Search on } s} \left(2^{n/4} + 2^{n/12}\left(\underbrace{2^{n/6}}_{\text{Produce } \mathcal{L}_1} + \underbrace{2^{n/6}}_{\text{Search on } x_3}\right)\right) = \mathcal{O}\left(2^{n/3}\right) .$$

**Naive algorithms.** For any other $k$, a naive solution is to fall back on the previous strategies:

- Cut the problem in halves and use Ambainis' algorithm.
- Cut the problem in three and use Algorithm 6.4.
- Cut the problem in four and use the 4-list merging of [Ber+13].

*Remark* 6.1 (Memory models). To date, all quantum speedups for Problem 6.1 require either QRAQM or QRACM. This includes all the algorithms presented in this section. In the QRACM model, no improvement over the naive algorithm (cutting in three) is known, and for any $k$, the best time complexity achievable remains $\mathcal{O}(2^{n/3})$.

### 6.1.3 Extended Merging Trees

We will now adapt the merging trees defined in Chapter 5. Recall that we have formulated the Unique $k$-XOR problem in a way such that the domain of the oracle $h$ is restricted to $\{0,1\}^{n/k}$, while its codomain remains $\{0,1\}^n$. We extend the definition in order to encompass Schroeppel and Shamir's algorithm, and the dissection algorithms of [Din+12, Section 3]. In short, *extended* merging trees are merging trees with additional variables, which handle the *repetition* of some of their subtrees. When subtrees are *repeated*, we change the arbitrary merging prefixes *or* we take an arbitrary sublist, as in Algorithms 6.3 and 6.5. Then, the mapping from trees to algorithms, and the formulas that determine the time and memory complexities, are adapted to take into account these loops.

Structurally, we still consider binary trees as in Definition 5.3. We adopt the same numbering of nodes and keep the variables $k_i^j$, $u_i^j$, $\ell_i^j$ that determine the shape of the list $\mathcal{L}_i^j$. Thus, the tree still represents an appropriate merging process.

We introduce a new variable $r$ for *repetitions*. Since we are solving Problem 6.1, we are in the situation of Schroeppel and Shamir's algorithm, where we cannot expect the tree to *always* contain a $k$-XOR. However, we can expect to find a partial $k$-XOR. Then, the tree must be re-computed many times before we find a full $k$-XOR to zero.

Constraints 5.3 and 5.2 are the same, but we adapt the constraint for the root node:

**Constraint 6.1** (Root node). *At the root node:* $u_0^0 + r = 1$ *and* $\ell_0^0 = 0$.

Next, we add the new *repetition* variables $r^j$. On most nodes we set $r = 0$, but we single out the *right subtrees on the main branch*, as depicted in Figure 6.2. Thus, there is only one non-zero repetition variable at each level, which is why we simply number them level by level.

For each subtree $\mathcal{T}^j$, $r^j$ represents the number of times it must be recomputed. Each computation should produce a new, independent list of elements, possibly with a new prefix.

**Constraint 6.2** (Repetitions). *We have:* $r = \sum_j r^j$, *and for each subtree* $\mathcal{T}^j$ *of width* $k^j$*:*

$$r \leqslant \frac{k_j}{k} - \ell^j \tag{6.1}$$

*where* $\ell^j$ *is the size of the list at the root of* $\mathcal{T}^j$.

In Constraint 6.2, the factor $\frac{k_j}{k}$ should be replaced by $k_j d$ if the codomain of $h$ is $2^{dn}$.

We still denote by $t_i^j$ the sampling time of a node. Constraint 5.4 still applies, in its simplest form, since we use the QRAQM model only.

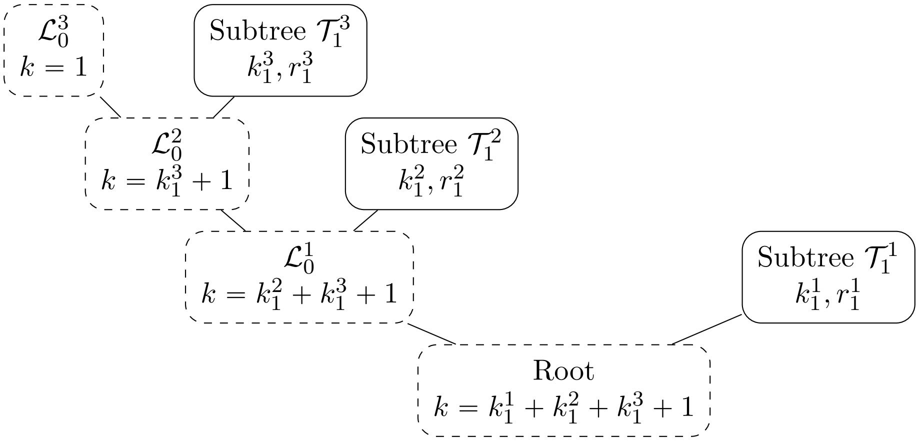


**Figure 6.2:** *Main branch* of a merging tree and all the subtrees that are attached to it.

**Constraint 6.3** (Sampling)**.** *Let $T_i^j$ be a node in the tree, either an S-node or an L-node.*

- *If $T_i^j$ is a leaf, $t_i^j = \frac{u_i^j}{2}$.*
- *Otherwise, $T_i^j$ has an S-child $S_{2i}^{j+1}$ and an L-child $S_{2i+1}^{j+1}$, and:*

$$t_i^j = \begin{cases} t_{2i}^{j+1} + \frac{1}{2}\max(u_i^j - u_{2i}^{j+1} - \ell_{2i+1}^{j+1}, 0) \text{ \textit{quantumly}} \\ t_{2i}^{j+1} + \max(u_i^j - u_{2i}^{j+1} - \ell_{2i+1}^{j+1}, 0) \text{ \textit{classically}} \end{cases} . \tag{6.2}$$

However, the total time complexity will be computed differently. Focusing on the subtrees $\mathcal{T}^j$ of the main branch, we let $t^j$ denote their respective *complete* time complexities, that is, the time to build the whole subtree with quantum merging.

**Constraint 6.4** (Subtrees)**.** *Let $\mathcal{T}^j$ be the right subtree at level $i$. Then:*

$$t^j = \max\left( \max_{\textit{List nodes of } \mathcal{T}^j} (t_i^j + \ell_i^j) \right) , \tag{6.3}$$

*where the sum is over all list nodes of $\mathcal{T}^j$, including its own root (since this is a list node itself).*

Then, we can now define the formulas for the time and memory complexities.

**Definition 6.1.** Let $\mathcal{T}$ be an extended $k$-merging tree. Let $\mathcal{T}^1, \dots, \mathcal{T}^p$ be the right subtrees of the main branch. We define $\mathsf{T_q}(\mathcal{T})$, $\mathsf{T_c}(\mathcal{T})$ and $\mathsf{M}(\mathcal{T})$ as:

$$\mathsf{M}(\mathcal{T}) = \max_{\text{List nodes}} (\ell_i^j)$$

$$\mathsf{T_c}(\mathcal{T}) = \max\left(r + t_0^0, r^1 + t^1, r^1 + r^2 + t^2, \dots, \left(\sum_{j'=1}^{j} r^{j'}\right) + t^j, \dots, r + t^p\right)$$

$$\mathsf{T_q}(\mathcal{T}) = \max\left(\frac{r}{2} + t_0^0, \frac{r^1}{2} + t^1, \frac{r^1 + r^2}{2} + t^2, \dots, \frac{1}{2}\left(\sum_{j'=1}^{j} r^{j'}\right) + t^j, \dots, \frac{r}{2} + t^p\right)$$

The idea of this definition is that the algorithm performs $p$ nested loops, one for each subtree of the main branch. We choose to nest from level 1 to $p$, as in Algorithm 6.6, with the idea that bigger and more costly subtrees may be attached to nodes at lower levels. We will see however that in the optimal algorithm for Unique $k$-XOR, all these levels collapse into a single one.

**Algorithm 6.6** Generic algorithm defined by an extended merging tree.

```
Input:  oracle access to $h : \{0,1\}^{n/k} \to \{0,1\}^n$
Output:  $k$-XOR solution tuple
for all Choices of $\mathcal{T}^1$ do
                    ▷ Either defined with a change of prefix, or a new choice of elements.
    Build $\mathcal{T}^1$
    for all Choices of $\mathcal{T}^2$ do
        Build $\mathcal{T}^2$
...
        for all Choices of $\mathcal{T}^p$ do
            Build $\mathcal{T}^p$
            Sample $x \in \mathcal{L}_0^p$ such that
                Find a match in $\mathcal{T}^p$
                Find a match in $\mathcal{T}^{p-1}$
...
                Find a match in $\mathcal{T}^1$
                if this gives a complete $k$-XOR to zero then
                    return the solution
            EndSample
```

We can see that, with our definitions of $t^j$, $t^0$, $r^j$ and $r$, the time complexity of Algorithm 6.6, up to a polynomial factor, is going to be:

$$2^{nr_1}\left(\underbrace{2^{nt_1}}_{\mathcal{T}_1} + 2^{nr_2}\left(\underbrace{2^{nt_2}}_{\mathcal{T}_2} + \dots + 2^{nr_p}\left(\underbrace{2^{nt_p}}_{\mathcal{T}_p} + \underbrace{2^{nt^0}}_{\text{Sample}}\right)\dots\right)\right),$$

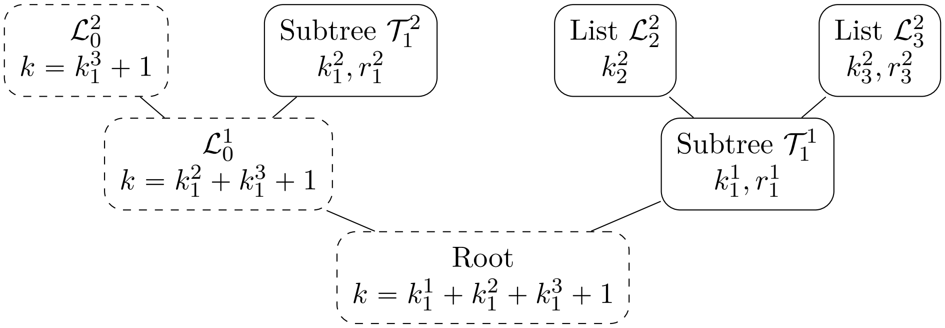


**Figure 6.3:** More complex repetition loops.

where we recover the equation of Definition 6.1. In the quantum setting, all these loops become nested quantum searches, and the time complexity can easily be obtained by the classical-quantum correspondence of Section 2.2.

*Remark* 6.2 (QRAQM requirement). In Chapter 5, we only needed QRACM, as all intermediate lists could be written down classically, and quantum access was necessary to sample new elements. However, here, the merging operations (writing down the lists) are performed *under a quantum search*, which is why QRAQM is necessary.

### 6.1.4 New Generalization

Until now, the presentation follows the merging trees of [NS20], but we remark that repetition loops can be laid out in an even more complex way. For example, not only subtrees of the main branch can be repeated, but also their subtrees.

With the example of Figure 6.3, we could write the complexity as:

$$\max\left(\frac{r}{2} + t_0^0, \frac{r_3^2}{2} + t_3^2, \frac{r_3^2 + r_1^1}{2} + t_1^1, \frac{r_3^2 + r_1^1}{2} + t_1^1, \frac{r_3^2 + r_1^1 + r_1^2}{2} + t_1^2\right) \quad ,$$

where $t_1^1$ has only to take into account the node $\mathcal{T}_1^1$, not its sublist $\mathcal{L}_3^2$ that is already computed at this point. However, it is possible to show that adding new loops inside a subtree $\mathcal{T}^i$ of the main branch cannot improve the time complexity, with an exception: taking a list *out of the loops* over $\mathcal{T}^i$. Thus, we make the following remark.

*Remark* 6.3 (Amending the constraints). A subtree $\mathcal{T}^j$ of width $k^j$, can cost 0 inside the repetitions if a global cost $\frac{k^j}{k}$ (in time and memory) has been already paid. Indeed, when $\mathcal{T}^j$ is of width 1, a full lookup table of $h$ can be prepared beforehand and reused instead of rebuilding it at each iterate. Likewise, we can prepare the sorted list of all $k^j$-tuples in order to retrieve quickly those having a wanted prefix.

The more general case is ruled out by a proof similar to that of Lemma 6.3.

### 6.1.5 Correspondence between Trees and Algorithms

Similarly as in Chapter 5, to any extended merging tree corresponds a classical (respectively quantum) extended merging algorithm that has the wanted complexity.

**Theorem 6.1** (Quantum extended merging strategies)**.** *Let $\mathcal{T}_k$ be an extended $k$-merging tree and $\mathsf{T_q}(\mathcal{T}_k)$ computed as in Definition 6.1. Then there exists a* quantum extended merging algorithm *that, given access to a quantum oracle $O_h$, finds a $k$-XOR.*

*This algorithm succeeds with probability more than $1 - e^{-an}$ for some $a > 0$. It runs in time $\widetilde{\mathcal{O}}\left(2^{\mathsf{T_q}(\mathcal{T}_k)n}\right)$, counted in $n$-qubit register operations, makes the same number of queries to $O_h$. It uses a memory $\mathcal{O}\left(2^{\mathsf{M}(\mathcal{T}_k)n}\right)$, counted in $n$-qubit registers (QRAQM). The constants in the $\mathcal{O}$ depend on $k$.*

*Proof.* We rely on Theorem 5.4 for the correctness of merging strategies. Each level of quantum search builds a new subtree, as in Algorithm 6.6. The search space itself is defined by an arbitrary prefix, either of the codomain (a merging constraint, as in Schroeppel and Shamir's algorithm) or of the domain (an input sublist, as in Algorithm 6.3). A given bit-string of the search space at level $j$ is good if, after building the corresponding subtree $\mathcal{T}^j$, and after running the search at level $j+1$, we find a solution $k$-XOR.

Among all repetitions of the subtrees $\mathcal{T}^1, \dots, \mathcal{T}^p$, only one choice shall lead to a solution (otherwise there would be too many repetitions). We miss it if the corresponding merging tree fails to find it; but Theorem 5.4 ensures a probability of success exponentially close to 1. However, this requires to increase the list sizes by a constant factor, which is impossible in the limit case of a codomain $2^{n/k}$. In this case, we only get a constant success probability. □

We can give an example for $k = 4$ (Algorithm 6.7) which is a quantum variant of Schroeppel and Shamir's 4-list merging.

The classical time complexity of Algorithm 6.7 would be:

$$\underbrace{2^{0.25n}}_{\text{Choice of } u} \left( 2^{0.125n} \left( \underbrace{2^{0.125n}}_{\substack{\text{Intermediate} \\ \text{list } \mathcal{L}_1^1}} + \underbrace{2^{0.25n}}_{\substack{\text{Exhaustive} \\ \text{search}}} \right) \right) = 2^{0.625n} \enspace ,$$

which is not optimal. However, as a quantum algorithm with nested Grover searches, it optimizes differently, since exhaustive search factors are replaced by their square roots:

$$2^{0.125n/2} \times 2^{0.125n/2} \left( 2^{0.125n} + 2^{0.25n/2} \right) = 2^{0.3125n} \enspace .$$

This is more than the complexity $\widetilde{\mathcal{O}}\left(2^{0.3n}\right)$ of [Ber+13], but we will see in Section 6.2 that this merging strategy has more applications than only the $k$-SUM problem.

**Algorithm 6.7** Optimal merging tree algorithm for Problems 6.2 and 6.1 with $k = 4$.

```
Input:   query access to $h : \{0,1\}^{n/4} \to \{0,1\}^n$
Output:   a 4-tuple $(x_i)_{0 \leqslant i \leqslant 3}$ such that $\bigoplus h(x_i) = 0$
Query the codebook and build a list $\mathcal{L}_1^2 = \{(x, h(x)), x \in \{0,1\}^{n/4}\}$
Set $\mathcal{L}_3^2 = \mathcal{L}_1^2$
Sample each $u \in \{0,1\}^{0.25n}$ such that
    Repeat $2^{0.125n}$ times
        Build a list $\mathcal{L}_1^1$ of $2^{0.125n}$ partial collisions: $(x_2, x_3)$ such that $h(x_2) \oplus h(x_3) = u|*$, in time $2^{0.125n}$.
        ▻ if we take any element, we expect a partial collision on $0.25n$ bits with some other in $\mathcal{L}_3^2$. Thus, each element of $\mathcal{L}_1^1$ is drawn in expected time 1
        Sample $x_0 \in \{0,1\}^{n/4}$ such that
            Find $x_1 \in \mathcal{L}_1^2$ such that $h(x_0) \oplus h(x_1) = u|*$
            if $\exists (x_2, x_3) \in \mathcal{L}_1^1, \bigoplus_i h(x_i) = 0$ then
                Exit and return the tuple $x_0, x_1, x_2, x_3$
        EndSample
    EndRepeat
EndSample
```

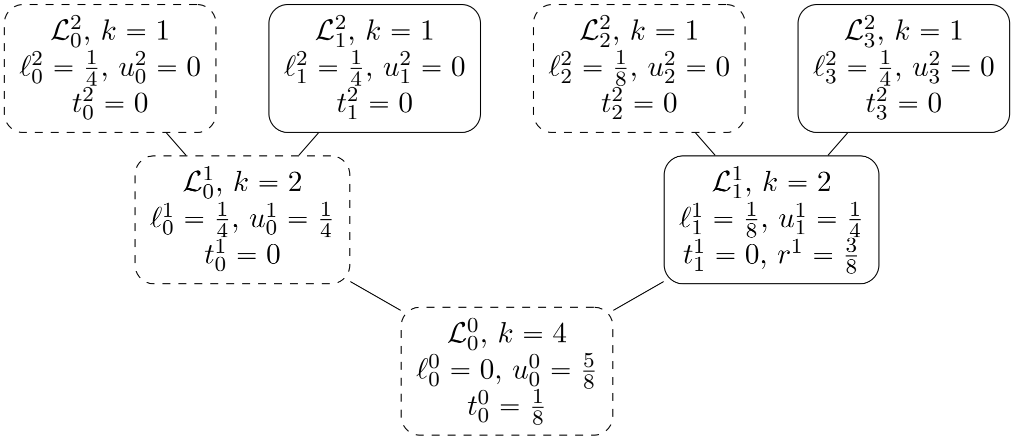

$\mathcal{L}_0^2$, $k = 1$
$\ell_0^2 = \frac{1}{4}$, $u_0^2 = 0$
$t_0^2 = 0$
$\mathcal{L}_1^2$, $k = 1$
$\ell_1^2 = \frac{1}{4}$, $u_1^2 = 0$
$t_1^2 = 0$
$\mathcal{L}_2^2$, $k = 1$
$\ell_2^2 = \frac{1}{8}$, $u_2^2 = 0$
$t_2^2 = 0$
$\mathcal{L}_3^2$, $k = 1$
$\ell_3^2 = \frac{1}{4}$, $u_3^2 = 0$
$t_3^2 = 0$
$\mathcal{L}_0^1$, $k = 2$
$\ell_0^1 = \frac{1}{4}$, $u_0^1 = \frac{1}{4}$
$t_0^1 = 0$
$\mathcal{L}_1^1$, $k = 2$
$\ell_1^1 = \frac{1}{8}$, $u_1^1 = \frac{1}{4}$
$t_1^1 = 0$, $r^1 = \frac{3}{8}$
$\mathcal{L}_0^0$, $k = 4$
$\ell_0^0 = 0$, $u_0^0 = \frac{5}{8}$
$t_0^0 = \frac{1}{8}$

**Figure 6.4:** Extended merging tree for Algorithm 6.7.

**Correspondence with Dissection algorithms.** The idea of guessing intermediate values is the core of the Dissection framework of [Din+12]. This is exactly what we do here, since under the exhaustive search loops, we merge using arbitrary prefixes: these intermediate values are the prefixes of the subtrees $\mathcal{T}^j$.

**Proposition 6.1.** *Any Dissection algorithm for k-XOR (by their definition in [Din+12, Section 3]) can be reframed as a classical extended merging tree.*

Our definition allows to extend the Dissection framework, which was intended to solve the problem with a single solution, to any intermediate domain size.

### 6.1.6 New Results for Unique k-XOR

For a general $k$, the algorithms for Problem 6.2 and Problem 6.1 achieve the same time complexities in the QRAQM model: accessing the data classically or quantumly makes little difference for the unique $k$-XOR problem. The reason is that the amount of data accessed (for example, $2^{n/4}$ for $k = 4$) is lower than the best time complexity (for example, $2^{0.3125n}$ for $k = 4$). Thus, we can query the whole codebook, store it in QRAQM and emulate quantum oracle queries. For $k = 3$, this makes only a difference in the memory complexity.

If we optimized the trees without Remark 6.3, we would obtain the following optimal complexity, which is from [NS20]:

**Theorem 6.2** (Optimal trees from [NS20])**.** *Let $k > 2$ be an integer. There exists a merging tree that given k lists of uniformly distributed n-bit strings, of size $2^{n/k}$ each, a k-XOR on n bits (if it exists) in quantum time $\mathcal{O}(2^{\gamma_k n})$ where $\gamma_k = \frac{1}{k}\frac{k+\lceil k/5\rceil}{4}$. In particular, it converges towards a minimum $0.3$, which is reached by multiples of $5$. For $k \geqslant 5$ the memory (QRAQM) used is $\leqslant 2^{0.2n}$.*

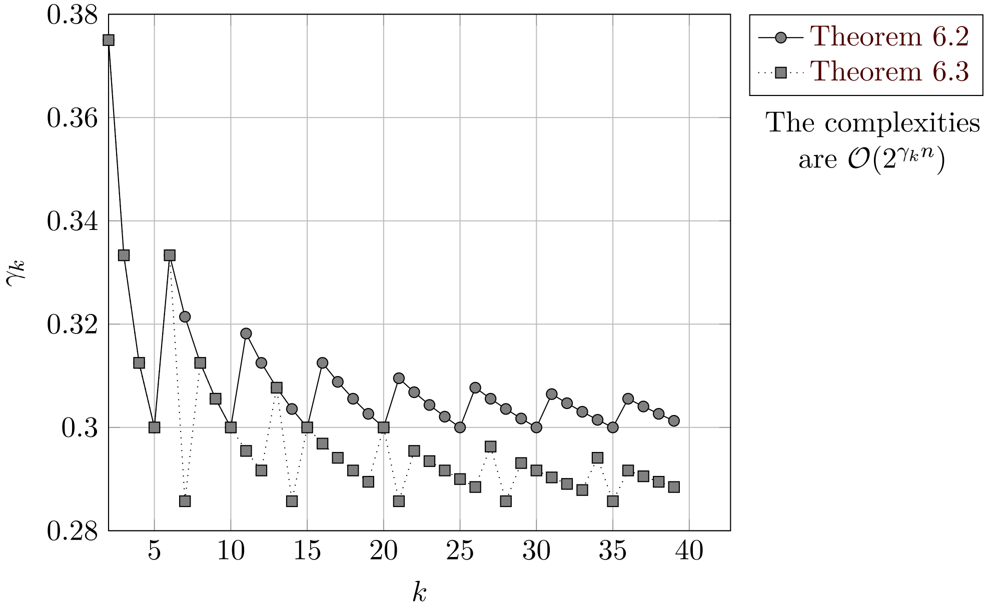
However, this small remark (taking some subtrees outside the repetitions, by performing trivial products of lists) allows to reach slightly improved exponents, and to break the previous lower bound 0.3 for $k$-list merging. This is a new result from this thesis.

**Theorem 6.3** (New trees for unique $k$-XOR)**.** *Let $k > 2$ be an integer. The best merging tree (with our definition) finds a k-XOR in time $\widetilde{\mathcal{O}}(2^{\gamma_k n})$ where:*

$$\gamma_k = \frac{k + \left\lfloor \frac{k+6}{7} \right\rfloor + \left\lfloor \frac{k+1}{7} \right\rfloor - \left\lfloor \frac{k}{7} \right\rfloor}{4k} \quad . \tag{6.4}$$

*In particular, it converges towards a minimum $\frac{2}{7}$, which is reached by multiples of $7$.*

*Remark* 6.4. Many different equivalent closed expressions for $\gamma_k$ are possible, but none of them without at least three $\lfloor\rfloor$ or $\lceil\rceil$.



The complexities are $\mathcal{O}(2^{\gamma_k n})$

**Figure 6.5:** Best time complexities obtained for Unique $k$-XOR with merging trees, with Theorem 6.2 and Theorem 6.3.

**Table 6.1:** Quantum time and memory (QRAQM or QRACM) complexities for Problem 6.1.

| | Best time | | Corresponding mem. | | Best t.-m. product | |
|---|---|---|---|---|---|---|
| $k$ | As fraction | Rounded | As fraction | Rounded | As fraction | Rounded |
| 2 | 3/8 | 0.375 | 1/4 | 0.25 | 1/2 | 0.5 |
| 3 | 1/3 | 0.3333 | 1/6 | 0.1667 | 1/2 | 0.5 |
| 4 | 5/16 | 0.3125 | 1/4 | 0.25 | 11/24 | 0.4583 |
| 5 | 3/10 | 0.3 | 1/5 | 0.2 | 9/20 | 0.45 |
| 6 | 1/3 | 0.3333 | 1/6 | 0.1667 | 4/9 | 0.4444 |
| 7 | 2/7 | 0.2857 | 2/7 | 0.2857 | 19/42 | 0.4524 |
| 8 | 5/16 | 0.3125 | 1/8 | 0.125 | 7/16 | 0.4375 |
| 9 | 11/36 | 0.3056 | 1/6 | 0.1667 | 4/9 | 0.4444 |
| 10 | 3/10 | 0.3 | 1/5 | 0.2 | 9/20 | 0.45 |
| 11 | 13/44 | 0.2955 | 3/11 | 0.2727 | 5/11 | 0.4545 |
| 12 | 7/24 | 0.2917 | 1/4 | 0.25 | 7/16 | 0.4167 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

A comparison of these results is given in Figure 6.5, and corresponding values in Table 6.1.

We defer the full proof of this optimality to Section 6.1.7 and focus for now on the description of the optimal trees, which is actually very simple, despite the number of *a priori* possible strategies.

#### 6.1.6.1 Description of the Optimal Trees

For $k \leqslant 5$, the results of Theorem 6.3 and Theorem 6.2 coincide. Examples for $k = 4$ and $k = 3$ were given in Algorithm 6.7 and Algorithm 6.5. Next, we can have a look at the Unique 5-XOR algorithm (Algorithm 6.8). It uses a single repetition loop, over a prefix $s$ of $\frac{n}{5}$ bits. Inside this loop, a merging of two lists of size $2^{\frac{n}{5}}$ is performed using the prefix $s$. One can then compute the time complexity as:

$$\underbrace{2^{\frac{n}{10}}}_{\text{Loop over } s} \left( \underbrace{2^{\frac{n}{5}}}_{\text{Merging}} + \underbrace{2^{\frac{n}{10}} \cdot 2^{\frac{n}{10}}}_{\substack{\text{Quantum search over}\\ \text{a pair of lists}}} \right) .$$

Thus, the intermediate merge and the quantum search are nicely balanced, and a time complexity of $\mathcal{O}\left(2^{0.3n}\right)$ is reached.

**Algorithm 6.8** Quantum algorithm for the unique 5-XOR problem, with or without oracle access.

```
Input: query access to $h : \{0,1\}^{n/5} \to \{0,1\}^n$
Output: a 5-tuple $(x_i)_{1 \leqslant i \leqslant 5}$ such that $\bigoplus_i h(x_i) = 0$
Query the codebook and build a list $\mathcal{L}_1^3 = \{(x, h(x)), x \in \{0,1\}^{n/5}\}$
Set $\mathcal{L}_1^2 = \mathcal{L}_1^3$
Set $\mathcal{L}_3^2 = \mathcal{L}_1^3$
Sample $s \in \{0,1\}^{n/5}$ such that
    Using $\mathcal{L}_3^2$, build a list $\mathcal{L}_1^1$ of $2^{n/5}$ 3-tuples $(x_1, x_2, h(x_1) \oplus h(x_2))$ such that $h(x_1) \oplus h(x_2) = s|*$.
    Sample $(x_4, h(x_4))$ in $\mathcal{L}_1^3$ such that
        Sample $x_5 \in \{0,1\}^{n/5}$ such that
            Find $(x_3, h(x_3)) \in \mathcal{L}_1^2$ such that $h(x_3) \oplus h(x_4) \oplus h(x_5) = s|*$
            Find $(x_1, x_2, h(x_1) \oplus h(x_2)) \in \mathcal{L}_1^1$ such that: $\bigoplus_{i=1}^5 h(x_i) = 0_{2n/5}|*$
            if This is equal to zero then
                Exit and return $(x_i)_{1 \leqslant i \leqslant 5}$
        EndSample
    EndSample
EndSample
```

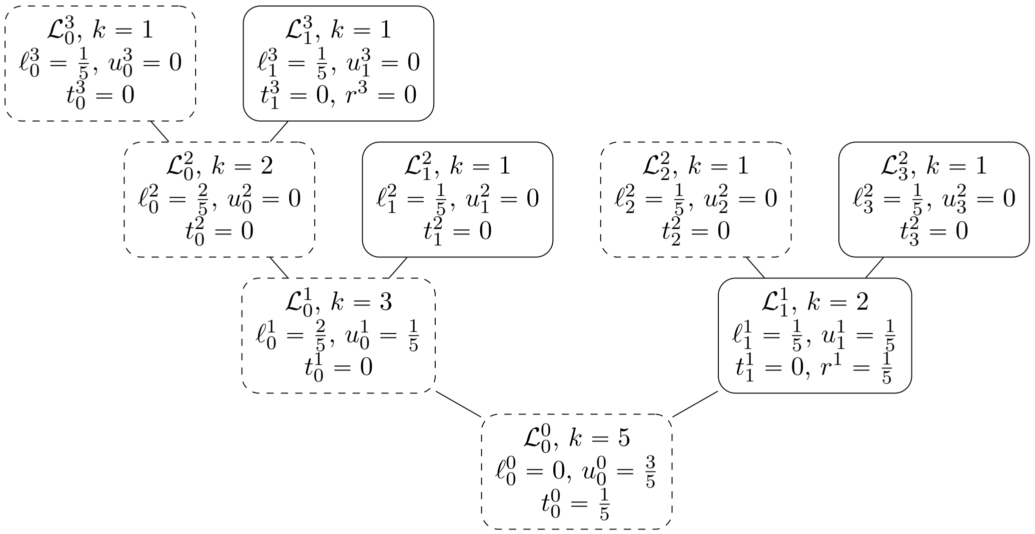

$\mathcal{L}_0^3$, $k = 1$
$\ell_0^3 = \frac{1}{5}$, $u_0^3 = 0$
$t_0^3 = 0$
$\mathcal{L}_1^3$, $k = 1$
$\ell_1^3 = \frac{1}{5}$, $u_1^3 = 0$
$t_1^3 = 0$, $r^3 = 0$
$\mathcal{L}_0^2$, $k = 2$
$\ell_0^2 = \frac{2}{5}$, $u_0^2 = 0$
$t_0^2 = 0$
$\mathcal{L}_1^2$, $k = 1$
$\ell_1^2 = \frac{1}{5}$, $u_1^2 = 0$
$t_1^2 = 0$
$\mathcal{L}_2^2$, $k = 1$
$\ell_2^2 = \frac{1}{5}$, $u_2^2 = 0$
$t_2^2 = 0$
$\mathcal{L}_3^2$, $k = 1$
$\ell_3^2 = \frac{1}{5}$, $u_3^2 = 0$
$t_3^2 = 0$
$\mathcal{L}_0^1$, $k = 3$
$\ell_0^1 = \frac{2}{5}$, $u_0^1 = \frac{1}{5}$
$t_0^1 = 0$
$\mathcal{L}_1^1$, $k = 2$
$\ell_1^1 = \frac{1}{5}$, $u_1^1 = \frac{1}{5}$
$t_1^1 = 0$, $r^1 = \frac{1}{5}$
$\mathcal{L}_0^0$, $k = 5$
$\ell_0^0 = 0$, $u_0^0 = \frac{3}{5}$
$t_0^0 = \frac{1}{5}$

**Figure 6.6:** Optimal Unique 5-XOR merging tree.

The novelty of Theorem 6.3 appears with Algorithm 6.9, whose total time complexity is:

$$\underbrace{2^{2n/7}}_{\substack{\text{Building } \mathcal{L}_{34}\\ \text{and } \mathcal{L}_{67}}} + \underbrace{2^{n/7}}_{\text{Search of } s}\left(\underbrace{2^{n/7}}_{\text{Computing } \mathcal{L}_{567}} + \underbrace{2^{n/7}}_{\text{Search in } \mathcal{L}_{12}}\right) = \mathcal{O}\left(2^{2n/7}\right) .$$

It benefits from computing some products of lists outside any repetitions. Interestingly, this also modifies the memory requirements: only $2^{n/7}$ QRAQM is required, in order to hold $\mathcal{L}_{567}$, and $2^{2n/7}$ QRACM is needed for $\mathcal{L}_{34}$ and $\mathcal{L}_{67}$.

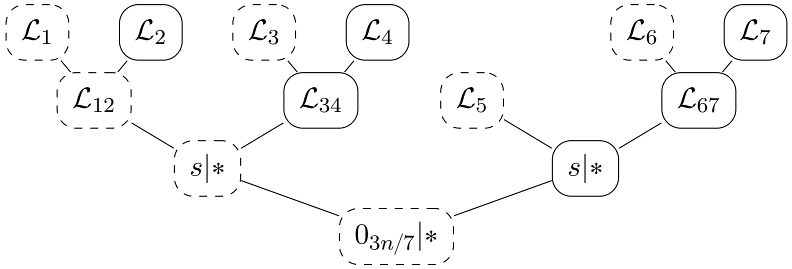


**Figure 6.7:** Unique 7-XOR merging tree of Algorithm 6.9.

For a bigger $k$, it is possible to reach the optimal complexity by mimicking the 7-XOR strategy. We introduce two integer variables $k_1$ and $k_2$ with the values:

$$\begin{cases} k_1 = \left\lfloor \frac{3k}{7} \right\rfloor \\ k_2 = \left\lfloor \frac{2k}{7} \right\rfloor - \left\lfloor \frac{k-1}{7} \right\rfloor + \left\lfloor \frac{k-2}{7} \right\rfloor \end{cases} \tag{6.5}$$

**Algorithm 6.9** Unique 7-XOR algorithm.

```
Input: 7 lists $\mathcal{L}_i$
Output: a 7-tuple $(x_i) \in \prod_i \mathcal{L}_i$ that XORs to 0
Build $\mathcal{L}_{67} = \mathcal{L}_6 \bowtie_0 \mathcal{L}_7$ (all sums between these two lists)
Build $\mathcal{L}_{34} = \mathcal{L}_3 \bowtie_0 \mathcal{L}_4$ (all sums between these two lists)
Sample $s \in \{0,1\}^{2n/7}$ such that          ▷ $2^{n/7}$ quantum search iterates
    Build $\mathcal{L}_{567} = \mathcal{L}_5 \bowtie_s \mathcal{L}_{67}$          ▷ Time $2^{n/7}$, which is the size of the list
    Sample $x \in \mathcal{L}_1 \times \mathcal{L}_2$ such that          ▷ $2^{n/7}$ quantum search iterates
        Find $y \in \mathcal{L}_{34}$ such that $x \oplus y = s|*$
        Find $z \in \mathcal{L}_{567}$ such that $x \oplus y \oplus z = 0_{3n/7}|*$
        if $x \oplus y \oplus z = 0$ then
            Exit and return the result
    EndSample
EndSample
```

and we perform Algorithm 6.10. Thus, the tree structure is overly simple (Figure 6.8): there are four subtrees, each of which is a trivial product of lists (a merge with empty prefixes). There is only a single repetition loop, and the whole algorithm contains only two levels of quantum search. The fact that this choice of structure matches the complexity given by Theorem 6.3 is not obvious, but it will follow naturally from the proof of the theorem in Section 6.1.7.

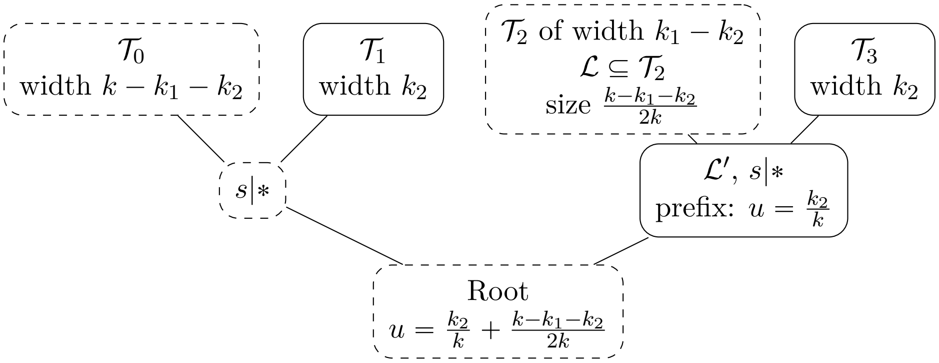

$\mathcal{T}_0$ width $k - k_1 - k_2$
$\mathcal{T}_1$ width $k_2$
$\mathcal{T}_2$ of width $k_1 - k_2$
$\mathcal{L} \subseteq \mathcal{T}_2$
size $\frac{k-k_1-k_2}{2k}$
$\mathcal{T}_3$ width $k_2$
$s|*$
$\mathcal{L}'$, $s|*$
prefix: $u = \frac{k_2}{k}$
Root
$u = \frac{k_2}{k} + \frac{k-k_1-k_2}{2k}$

**Figure 6.8:** Generic merging tree that reaches the optimal complexity for unique $k$-XOR (see Algorithm 6.10).

### 6.1.6.2 Memory Usage

Our algorithms for Unique $k$-XOR reach the best time complexity $\mathcal{O}(2^{2n/7})$ when $k$ is a multiple of 7, but at these points, they require a QRACM of size $2^{2n/7}$. This is suboptimal with respect to the time-memory product, a metric that makes sense if the qubits used for QRAQM storage need active error correction (see [GR04] or [JS19]),

**Algorithm 6.10** Generic Unique $k$-XOR algorithm.

```
Input: $k$ lists $\mathcal{L}_i$
Output: a $k$-tuple $(x_i) \in \prod_i \mathcal{L}_i$ that XORs to 0
Select $k_1, k_2$ by Equation 6.5
Build $\mathcal{T}_1$ and $\mathcal{T}_3$, each with the product of $k_2$ lists
Sample $s \in \{0,1\}^{\frac{k_2}{k}n}$ such that
    Sample Sublists $\mathcal{L}$ of $\mathcal{T}_2$ of size $\frac{k-k_1-k_2}{2k}$ such that
        Merge $\mathcal{L}$ with $\mathcal{T}_3$ to obtain an intermediate list $\mathcal{L}'$ with prefix of $\frac{k_2}{k}n$ bits and size $\frac{k-k_1-k_2}{2k}$
        Sample $x \in \mathcal{T}_0$ such that        ▷ $2^{\frac{k-k_1-k_2}{2k}n}$ quantum search iterates
            Find $y \in \mathcal{T}_1$ such that $x \oplus y = s|*$
            if There is a collision on $\mathcal{L}'$ then
                Exit and return the result
        EndSample
    EndSample
EndSample
```

meaning that a memory of size $M$ costs $M$ classically controlled operations at each time step.

In order to find the best time-memory product, we change the optimization goal. We find that for Problem 6.2, the memory used is exactly the storage of the lists.

**Lemma 6.1.** *The optimal $k$-merging trees for Problem 6.2 with respect to the time-memory product use a memory of $2^{n/k}$.*

For Problem 6.1, there are improvements for small values of $k$, but not for $k \geqslant 8$. Thus, we actually optimize the tree under a memory constraint of $2^{n/k}$. We obtain results analogous to the classical Dissection technique [Din+12, Section 3], given in Table 6.1. In particular:

**Proposition 6.2.** *The optimal time complexity exponent with $2^{n/k}$ memory is: $\frac{k-\mathcal{O}(\sqrt{k})}{2k}$. It converges towards $\frac{1}{2}$ when $k \to +\infty$. The time-memory product admits a global minimum over $k$.*

Note that these results are given by the trees of [NS20], as our improvements in time given by Theorem 6.3 rely on a higher memory consumption. We remark that these algorithms often reach a square root speedup upon [Din+12, Section 3]. As we have discussed before, the iteration technique of [Din+12, Section 4] is unlikely to improve the time nor the memory usage in the quantum setting. Thanks to a simplification of the constraints of [NS20], we are able to compute the time for higher values of $k$ and to obtain the results of Table 6.2 and Figure 6.9.

**Proposition 6.3.** *The time-memory product for unique $k$-XOR merging trees is lower bounded by: $\widetilde{\mathcal{O}}\left(2^{\frac{7}{17}n}\right)$ which is reached by $k = 17$.*

**Table 6.2:** Quantum time and memory (QRAQM or QRACM) complexities for Unique $k$-XOR when the memory is $2^{n/k}$.

| | Time | | QRAQM | | Product | |
|---|---|---|---|---|---|---|
| $k$ | As fraction | Rounded | As fraction | Rounded | As fraction | Rounded |
| 4 | 5/16 | 0.3125 | 1/4 | 0.25 | 9/16 | 0.5625 |
| 5 | 3/10 | 0.3 | 1/5 | 0.2 | 1/2 | 0.5 |
| 6 | 1/3 | 0.3333 | 1/6 | 0.1667 | 1/2 | 0.5 |
| 7 | 9/28 | 0.3214 | 1/7 | 0.1429 | 13/28 | 0.4643 |
| 8 | 5/16 | 0.3125 | 1/8 | 0.125 | 7/16 | 0.4375 |
| 9 | 1/3 | 0.3333 | 1/9 | 0.1111 | 4/9 | 0.4444 |
| 10 | 7/20 | 0.35 | 1/10 | 0.1 | 9/20 | 0.45 |
| 11 | 15/44 | 0.3409 | 1/11 | 0.0909 | 19/44 | 0.4318 |
| 12 | 1/3 | 0.3333 | 1/12 | 0.0833 | 5/12 | 0.4167 |
| 13 | 9/26 | 0.3462 | 1/13 | 0.0769 | 11/26 | 0.4231 |
| 14 | 5/14 | 0.3571 | 1/14 | 0.0714 | 3/7 | 0.4286 |
| 15 | 11/30 | 0.3667 | 1/15 | 0.0667 | 13/30 | 0.4333 |
| 16 | 23/64 | 0.3594 | 1/16 | 0.0625 | 27/64 | 0.4219 |
| **17** | **6/17** | **0.3529** | **1/17** | **0.0588** | **7/17** | **0.4118** |
| 18 | 13/36 | 0.3611 | 1/18 | 0.0556 | 5/12 | 0.4167 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

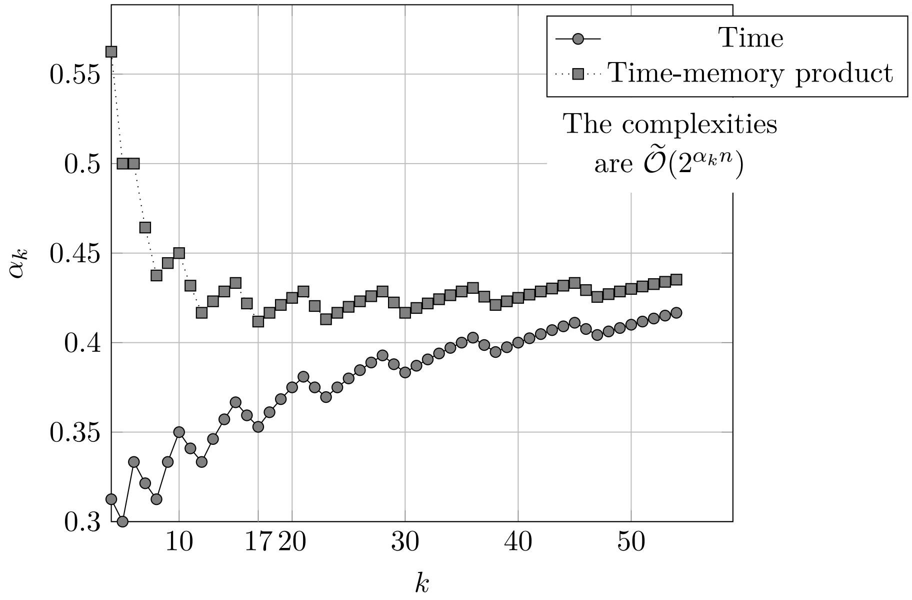


**Figure 6.9:** Optimization of merging trees for Unique $k$-XOR with a memory limited to $2^{n/k}$.

### 6.1.7 Proof of Optimality

We now prove Theorem 6.3. We will first prove the following result, and show later how it implies the formula for the optimal complexity and the shape of the optimal trees that we gave.

**Theorem 6.4.** *For any $k$, the optimal time $t$ for the unique $k$-XOR problem, with our merging tree framework, is given by:*

$$t = \min_{k_1,k_2 \in \mathbb{N}^2, k_1+k_2 \leqslant k} \left( \max \left( \frac{k_2}{k}, \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right)\right)\right) \quad , \tag{6.6}$$

*and can thus be obtained by solving a simple mixed integer linear program.*

We cut down the proof of this statement into two steps.

1. We first massively restrict the space of possible strategies, and show that we can still obtain the optimal strategy. In Lemma 6.2, we show that a merging tree using QRAQM (without repetitions) can be optimized so that only the list nodes (including the root) have non-trivial sampling procedures. This helps us to show Lemma 6.3: the subtrees of an extended merging tree are *classical* merging trees, that do not use quantum search, and finally Lemma 6.4: these classical trees can be seen as trees with only 2 levels, merging two lists of trivial sums with a single non-empty prefix.
2. The optimal time complexity becomes the solution of a much simpler linear program, that we write down. We simplify it considerably in order to obtain Equation 6.6; at the same time, we show that there indeed exists merging trees that allow to reach this complexity. These are the trees given in Theorem 6.3.

#### 6.1.7.1 Step 1: Reducing the Search Space

**Lemma 6.2** (Optimization of merging with QRAQM)**.** *A quantum merging tree using QRAQM can be optimized such that all Sample nodes are sampled in time 1.*

*Proof.* As an example, consider the situation of Figure 6.10. The time to build the root is the following, depending on the list sizes of the subtrees, and prefixes chosen at each level:

$$\ell^0 + \frac{1}{2}\left(\max(u - u^1 - \ell^1_1, 0) + \max(u^1 - u^2 - \ell^2_1, 0) + \max(u^2 - u^3 - \ell^3_1, 0) + u^3\right) \quad .$$

Note that we will only change the prefix sizes, so we may consider lists of fixed or limited size (as is the case in our problem). We prove that we can set these prefixes so that $u^1 = u^2 + \ell^2_1, u^2 = \ell^3_1, u^3 = 0$ and similarly for any tree height. First, we perform the same operation recursively on the subtrees. Next, we remark that $u^3 = 0$ can be set without increasing the time complexity. Finally, assume e.g. that $u^1 - u^2 - \ell^2_1 > 0$,

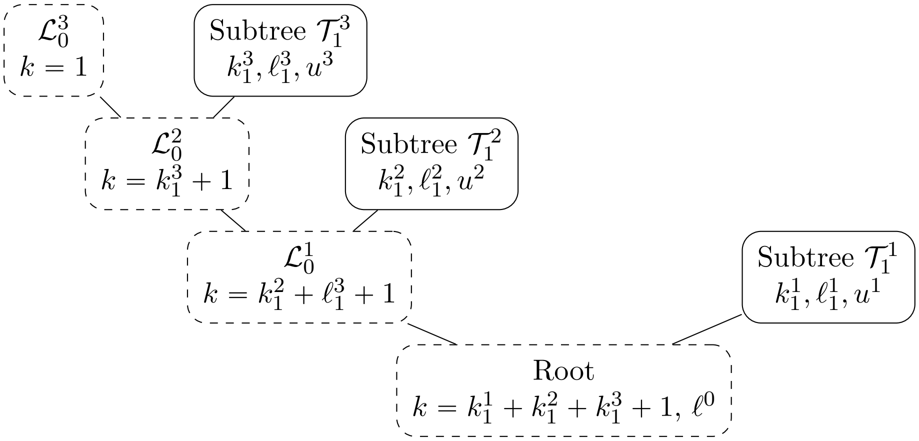


**Figure 6.10:** A merging tree using QRAQM.

then we will decrease $u^1$ to $u^2 - \ell_1^2$. The time complexity for sampling the root is left unchanged, because we increase the number of iterations at the next level in the same amount as we decrease it at this level. Since we are reducing $u^1$, the time complexity of $\mathcal{T}_1^1$ can only be decreased as well. □

This is also true of the merging trees in the QRACM model of Chapter 5: only at list nodes, and in particular the root, is the sampling time different from 1. Note, however, that this property does not hold for merging trees with classical memory, for $k \leqslant 7$, as our examples have shown.

**Lemma 6.3.** *For any $k$, the optimal time complexity is reached by an extended merging tree where all subtrees $\mathcal{T}^1, \ldots, \mathcal{T}^p$ of the main branch are built by classical merges, with their complexities equal to their size.*

*Proof.* The reason of this fact is that, when merging, we drop many tuples. Since all possibilities must be studied in the end, the more we merge, the more repetitions need to be made. For example, Schroeppel and Shamir's algorithm requires $2^{n/4}$ repetitions due to the intermediate prefix of $\frac{n}{4}$ bits. A subtree which uses quantum merging can be replaced by a subtree using classical merging only, which is bigger, but which will be repeated less.

We define the *total guessed prefixes* TGP of a tree to be the sum of all prefix sizes $u_j^i$, where $u_r = u_l$ is counted once only for two sibling nodes. For example, $\mathrm{TGP}(\mathcal{T}) = u$ if $\mathcal{T}$ is a leaf list with a prefix size $u$. Denoting by $u$ the prefix of $\mathcal{T}$, we remark:

$$\mathrm{TGP}(\mathcal{T}) = u + \mathrm{TGP}(\mathcal{T}^1) + \ldots + \mathrm{TGP}(\mathcal{T}^p) \ .$$

This corresponds to the total amount of prefixes that are arbitrarily fixed, and restrict the number of solutions of the merge. The reason for defining TGP is that, while solving the Unique $k$-XOR problem, a subtree $\mathcal{T}$ merging a fixed number of lists of fixed size will be repeated at least $\mathrm{TGP}(\mathcal{T})$ times: we must try all possible values for these prefixes. We can show:

> If $\mathcal{T}$ is a quantum merging tree producing a list of size $\ell$ with prefix $u$, then there exists a classical merging tree $\mathcal{T}'$, producing a list of size $\ell' \geqslant \ell$ with prefix $u' \leqslant u$, satisfying: $u' + \ell' = u + \ell$, and:
>
> $$\mathsf{T_c}\left(\mathcal{T}'\right) \leqslant \mathsf{T_q}\left(\mathcal{T}\right) + \frac{\mathrm{TGP}(\mathcal{T}) - \mathrm{TGP}(\mathcal{T}')}{2} \ . \tag{6.7}$$

*Remark* 6.5. Here we assume that neither $\mathcal{T}$ nor $\mathcal{T}'$ contain internal repetition loops, which greatly simplifies the definition of $\mathsf{T_c}\left(\mathcal{T}'\right)$ and $\mathsf{T_q}\left(\mathcal{T}\right)$ (time complexities of the complete subtrees). But a more involved proof would dismiss this case as well.

We prove (6.7) by induction on the tree structure. First, this is obviously true for leaf lists: if we produce a list of size $\ell$ having a prefix $u$, this costs quantumly $\ell + \frac{u}{2}$. Classically, we remove the prefix condition: the time increases to $\ell + u = \ell + \frac{u}{2} + \frac{u}{2}$.

Consider a generic tree, properly optimized, with subtrees $\mathcal{T}^1, \ldots, \mathcal{T}^p$ on the main branch. We will first replace them by the classical $\mathcal{T}'^1, \ldots, \mathcal{T}'^p$. Since $u^i + \ell^i = u'^i + \ell'^i$, the tree structure remains the same. We have: $\mathsf{T_c}\left(\mathcal{T}'^i\right) \leqslant \mathsf{T_q}\left(\mathcal{T}^i\right) + \frac{\mathrm{TGP}(\mathcal{T}^i) - \mathrm{TGP}(\mathcal{T}'^i)}{2}$.

At the root $\mathcal{T}'$, we will set $u' = u^1 + \ell^1$ and $\ell' = \ell + u - u'$. In other words, we set precisely the prefix that allows to sample the list in time 1. Let $t'$ be the time to build the whole list at the root, and $t$ the corresponding time for $\mathcal{T}$. We have in particular:

$$\mathsf{T_c}\left(\mathcal{T}'\right) = \max_i(t', \mathsf{T_c}\left(\mathcal{T}'^i\right)) \text{ and } \mathsf{T_q}\left(\mathcal{T}\right) = \max_i(t, \mathsf{T_q}\left(\mathcal{T}^i\right)) \ .$$

Then $t' = \ell + u - (u^1 + \ell^1) = \ell + \frac{(u-(u^1+\ell^1))}{2} + \frac{(u-u')}{2} = t + \frac{(u-u')}{2}$. Next, we use the recurrence hypothesis for the subtrees. We have:

$$\begin{aligned}
\max_i(t', \mathsf{T_c}\left(\mathcal{T}'^i\right)) &\leqslant \max_i\left(t + \frac{u-u'}{2}, \mathsf{T_q}\left(\mathcal{T}^i\right) + \frac{\mathrm{TGP}(\mathcal{T}^i) - \mathrm{TGP}(\mathcal{T}'^i)}{2}\right) \\
\mathsf{T_c}\left(\mathcal{T}'\right) &\leqslant \frac{u-u'}{2} + \sum_i \frac{\mathrm{TGP}(\mathcal{T}^i) - \mathrm{TGP}(\mathcal{T}'^i)}{2} + \max_i(t, \mathsf{T_q}\left(\mathcal{T}^i\right)) \\
&\leqslant \mathsf{T_q}\left(\mathcal{T}\right) + \frac{\mathrm{TGP}(\mathcal{T}) - \mathrm{TGP}(\mathcal{T}')}{2} \ .
\end{aligned}$$

Which proves the property. This formula displays the idea of this lemma in a clearer way: although there is an advantage in merging quantumly instead of classically, this advantage vanishes if we must repeat the merge for all possible prefix guesses. We obtain:

$$\mathsf{T_c}\left(\mathcal{T}'\right) + \frac{\mathrm{TGP}(\mathcal{T}')}{2} \leqslant \mathsf{T_q}\left(\mathcal{T}\right) + \frac{\mathrm{TGP}(\mathcal{T})}{2} \ .$$

Thus, replacing $\mathcal{T}$ by $\mathcal{T}'$ in the extended merging tree does not increase the time complexity. □

**Lemma 6.4.** *For any k, the optimal time complexity is reached by a tree where all main subtrees are classical trivial merges: $\mathcal{T}^i$ has no guessed prefix, except at its root.*

*Proof.* The idea is that, when repeating a subtree, we might as well take sublists instead of the intermediate prefixes. The number of repetitions will be the same and the merge, when optimized, costs the same as well.

As an example, consider a 2-level merging tree with lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$ of sizes $\ell_1, \ell_2, \ell_3, \ell_4$, a prefix $u_1$ at level 1 and a prefix $u_0$ at level 0. Merging pairwise gives two lists $\mathcal{L}_{12}$ and $\mathcal{L}_{34}$ of sizes $\ell_1 + \ell_2 - u_1$ and $\ell_3 + \ell_4 - u_1$, then a final list of size $\sum_i \ell_i - u - u_1$, and the merging needs to be repeated $\frac{u_0+u_1}{2}$ times. The fact that the tree is correctly balanced, and that the merging is purely classical, implies that $\ell_1 + \ell_2 - u_1 \geqslant \ell_1$ and that $\ell_3 + \ell_4 - u_1 \geqslant \ell_3$, i.e., $\ell_2 \geqslant u_1$ and $\ell_3 \geqslant u_1$. We also have $\ell_3 + \ell_4 \geqslant u_0$. Suppose without loss of generality that $\ell_1 + \ell_2 \geqslant \ell_3 + \ell_4$, that is, in the sampling-based version, we sample from $\mathcal{L}_1$ and try to match against $\mathcal{L}_2$, then $\mathcal{L}_{34}$.

Now, we remark that the constructions of $\mathcal{L}_{12}$ and $\mathcal{L}_{34}$ only need to be repeated $\frac{u_1}{2}$ times. The total cost is hence:

$$\frac{u_1}{2} + \max(\ell_4, \ell_3 + \ell_4 - u_1, \ell_2, \ell_1 + \ell_2 - u_1, \frac{u_0}{2} + \sum_i \ell_i - u_1 - u_0) \ .$$

Now, consider a modified merging tree in which we replace $\mathcal{L}_4$ by a sublist of size $\ell_4 - u_1 \geqslant 0$. There still is a repetition loop of $\frac{u_1}{2}$ iterations, since we need to span all such sublists. However, $\mathcal{L}_{12}$ does not need the prefix $u_1$ anymore, so it can go outside the repetition loop for $u_1$.

*Remark* 6.6. The change brought by Remark 6.3 appears here, as $\mathcal{L}_{12}$ has become a list independent from all repetition loops, that is built only once.

Thus, we total cost becomes:

$$\max(\ell_1 + \ell_2, \frac{u_1}{2} + \ell_4 - u_1, \frac{u_1}{2} + \ell_3 + \ell_4 - u_1, \ell_2, \frac{u_0 + u_1}{2} + \sum_i \ell_i - u_1 - u_0) \ .$$

But we remark that:

$$\frac{u_0 + u_1}{2} + \sum_i \ell_i - u_1 - u_0 \geqslant \frac{u_0 + u_1}{2} + \ell_1 + \ell_2 - u_1 \geqslant \ell_1 + \ell_2 \ .$$

In other words, removing the prefix $u_1$ has two consequences: • we can put the merging of $\ell_1$ and $\ell_2$ outside the loop over $u_0$, which makes the procedure overall more efficient; • we can replace the loop over a prefix $u_1$ by a loop over sublists of size $\ell_4 - u_1$ of the previous $\ell_4$.

□

Thus, when we consider the *main branch* of an extended merging tree, all its subtrees are built by merging two classical lists, *which can be precomputed* from the data, on a given prefix $u$ that needs to be repeated. This simplifies considerably the shape of the trees and our complexity analysis.

#### 6.1.7.2 Step 2: Simplifying the Constraints

From now on, we write $\mathcal{T}_i$ the subtrees at level $i$, instead of $\mathcal{T}^i$, and put also $i$ in index for all parameters. By Lemmas 6.3 and 6.4, we can write down the shape of an optimal extended merging tree for unique $k$-XOR as follows:

- It has a main branch with $p$ levels, and subtrees $\mathcal{T}_1, \dots, \mathcal{T}_p$ attached respectively at levels 1 to $p$.
- There is a subtree $\mathcal{T}_p'$ at level $p$ on the left, which is simply sampled. We have $\ell_p' = \frac{k_p'}{k}$, i.e., this subtree does not need to be repeated.
- Each $\mathcal{T}_i$ has a width $k_i$, a list length $\ell_i$, a prefix size $u_i$ and we have $\sum_{i=1}^{p} k_i + k_p' = k$
- The first subtree $\mathcal{T}_p$ builds a list $\mathcal{L}_p$ with no prefix: $u_p = 0$ .
- Each $\mathcal{T}_i$ induces a repetition loop with $r_i = \frac{1}{2}\left(\frac{k_i}{k} - \ell_i\right)$ .
- Each $\mathcal{T}_i$, except $\mathcal{T}_p$, has two children: $\mathcal{L}_i^l$ and $\mathcal{L}_i^r$, of width $k_i^l$ and $k_i^r$. We have $k_i = k_i^r + k_i^l$. We also have $\ell_i = \ell_i^l$ and $u_i = \ell_i^r$. We have $\ell_i^l \leqslant \frac{k_i^l}{k}$ and $\ell_i^r \leqslant \frac{k_i^r}{k}$, and each of these lists induces a global (but not repeated) cost of $\frac{k_i^l}{k}$ and $\frac{k_i^r}{k}$ respectively.
- We have $\forall i, u_{i-1} = u_i + \ell_i$, hence $u_i = \sum_{j=i+1}^{p} \ell_j$ .

Hence, all constraints on $t$ can be rewritten in the $k_i$ and the $\ell_i$:

$$\begin{cases} t \geqslant \max_{1 \leqslant i \leqslant p-1}\left(\frac{k_p}{k}, \frac{k_i^l}{k}, \frac{k_i^r}{k}\right) \\ t \geqslant \sum_i r_i + \frac{1}{2}\left(1 - \frac{\sum_i k_i}{k}\right) = \frac{1}{2}\left(1 - \sum_i \ell_i\right) \\ \frac{k_p}{k} \geqslant \ell_p \\ \forall 1 \leqslant i \leqslant p-1 \left(t \geqslant \left(\sum_{j=1}^{i} r_j\right) + \ell_i, \quad \frac{k_i^l}{k} \geqslant \ell_i, \quad \frac{k_i^r}{k} \geqslant \sum_{j=i+1}^{p} \ell_j\right) \end{cases} \tag{6.8}$$

The problem becomes much easier to solve, since the cut of $k$ can be handled by integer variables. However, we can simplify (6.8) further.

For more simplicity, let us consider a fixed height of 4, as in Figure 6.11. By rewriting (6.8) with the $\ell_i$ only, we obtain:

$$\begin{aligned} &(\text{C1}) \quad t \geqslant \frac{1}{2}\left(1 - \ell_1 - \ell_2 - \ell_3\right) \\ &(\text{C2}) \quad t \geqslant \frac{1}{2}\left(\frac{k_1 + k_2}{k} - \ell_1 + \ell_2\right) \\ &(\text{C3}) \quad t \geqslant \frac{1}{2}\left(\frac{k_1}{k} + \ell_1\right) \\ &(\text{C4}) \quad \frac{k_3}{k} \geqslant \ell_3, \frac{k_2^r}{k} \geqslant \ell_3, \frac{k_2^l}{k} \geqslant \ell_2, \frac{k_1^r}{k} \geqslant \ell_3 + \ell_2, \frac{k_1^l}{k} \geqslant \ell_1 \end{aligned}$$

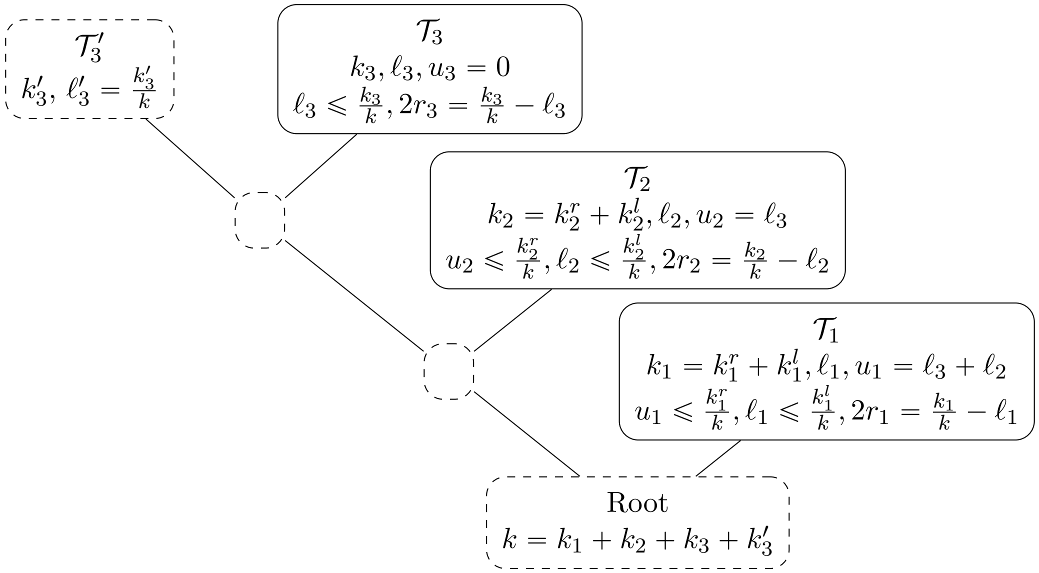

$\mathcal{T}_3'$
$k_3', \ell_3' = \frac{k_3'}{k}$
$\mathcal{T}_3$
$k_3, \ell_3, u_3 = 0$
$\ell_3 \leqslant \frac{k_3}{k}, 2r_3 = \frac{k_3}{k} - \ell_3$
$\mathcal{T}_2$
$k_2 = k_2^r + k_2^l, \ell_2, u_2 = \ell_3$
$u_2 \leqslant \frac{k_2^r}{k}, \ell_2 \leqslant \frac{k_2^l}{k}, 2r_2 = \frac{k_2}{k} - \ell_2$
$\mathcal{T}_1$
$k_1 = k_1^r + k_1^l, \ell_1, u_1 = \ell_3 + \ell_2$
$u_1 \leqslant \frac{k_1^r}{k}, \ell_1 \leqslant \frac{k_1^l}{k}, 2r_1 = \frac{k_1}{k} - \ell_1$
Root
$k = k_1 + k_2 + k_3 + k_3'$

**Figure 6.11:** Generic *main branch* for an optimal extended merging tree, with 4 levels.

$$\text{(C5)} \quad t \geqslant \frac{1}{k} \max\left(k_3, k_2^r, k_2^l, k_1^l, k_1^r\right)$$

We will first obtain the lower bound $\frac{2}{7}$ for $t$. By combining (C1) with (C4), we get: $t \geqslant \frac{1}{2}\left(1 - \frac{k_1}{k}\right)$. Next, we know that $t \geqslant \frac{k_3}{k}$, and that $\frac{k_2}{k} = \frac{k_2^r}{k} + \frac{k_2^l}{k} \geqslant \ell_2 + \ell_3$. By combining this with (C1) we obtain: $t \geqslant \frac{1}{2}(1 - \ell_1 - \frac{k_2}{k})$. We add this with (C3) to get: $4t \geqslant 1 + \frac{k_1 - k_2}{k}$. Summing (C1) with (C2), we eliminate $\ell_2$, replace $\ell_3$, use $t \geqslant \frac{1}{2}\left(1 - \frac{k_1}{k}\right)$ to eliminate $\ell_1$ and obtain: $8t \geqslant 1 + \frac{3k_1 + k_2 - k_3}{k}$.

Hence, we have four constraints that depend only on $k_1, k_2, k_3$:

$$8t \geqslant 1 + \frac{3k_1 + k_2 - k_3}{k} \qquad 2t \geqslant 1 - \frac{k_1}{k} \qquad 4t \geqslant 1 + \frac{k_1 - k_2}{k} \qquad t \geqslant \frac{k_3}{k} \quad . \tag{6.9}$$

Taking an appropriate linear combination yields $t \geqslant \frac{2}{7}$. For $k = 7$, it is possible to realize this optimum by setting $k_1 = 3, k_2 = 2, k_3 = 2$. In other words, the list $\ell_3'$ is actually empty and not used; the main branch collapses to a length 3 only. We obtain the unique 7-XOR example mentioned above.

If we reduce the depth by 1, we remove one subtree and obtain similar constraints:

$$t \geqslant \frac{k_2}{k} \qquad 2t \geqslant 1 - \frac{k_1}{k} \qquad 4t \geqslant 1 + \frac{k_1 - k_2}{k} \quad . \tag{6.10}$$

Summing all three of them gives $7t \geqslant 2$ again, which suggests that a main branch with three levels may actually be enough. This is what we prove next: minimizing $t$ under the constraints of Equation (6.9) cannot give a better result than under the constraints of Equation (6.10). A similar result will follow for any depth.

**Lemma 6.5.** *Let $k_1, k_2, k_3$ be integers such that $k_1 + k_2 + k_3 \leqslant k$. We have:*

$$\max\left(\frac{k_3}{k}, \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right), \frac{1}{8}\left(1 + \frac{3k_1 + k_2 - k_3}{k}\right)\right)$$
$$\geqslant \min_{k_1, k_2 \in \mathbb{N}^2, k_1 + k_2 \leqslant k}\left(\max\left(\frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right), \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{k_2}{k}\right)\right) \enspace .$$

*Proof.* On the right, we choose the same $k_1$ and $k_2$ as on the left. Since $k_1 + k_2 + k_3 \leqslant k$, the constraint $k_1 + k_2 \leqslant k$ is trivially satisfied. Removing one of the terms of the left hand side, we will show that:

$$\max\left(\frac{k_3}{k}, \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right)\right)$$
$$\geqslant \max\left(\frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right), \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{k_2}{k}\right) \enspace ,$$

which implies the statement of the Lemma. We separate two cases.

- If $\frac{k_3}{k} \geqslant \frac{1}{2}\left(1 - \frac{k_1}{k}\right)$: this implies $2k_3 \geqslant k - k_1$, that is, $k_3 \geqslant k - k_1 - k_3 \geqslant k_2$. Thus, we have $\frac{k_3}{k} \geqslant \frac{k_2}{k}$ and the result follows.
- Otherwise, the same result follows trivially. □

By Lemma 6.5, we cannot obtain better merging trees by increasing the depth of the main branch. Finally, we show that the constraints (6.10) are necessary and sufficient: given a choice of $k_1, k_2$, we exhibit merging trees that reach the prescribed complexity. These trees are those given in Theorem 6.3.

**Lemma 6.6.** *Let $k_1, k_2$ be such that $k_1 + k_2 \leqslant k$. Then there exists an extended merging tree algorithm solving Unique k-XOR in time:*

$$t = \max\left(\frac{k_2}{k}, \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right)\right) \enspace . \tag{6.11}$$

*Proof.* First of all, consider the case $k_1 \leqslant k_2$, i.e., the subtree at level 2 is bigger than the subtree at level 1. This implies in particular $t \geqslant \frac{k_2}{k} \geqslant \frac{k_1}{k}$ and $t \geqslant \frac{1}{2}(1 - \frac{k_1}{k})$, thus $t \geqslant \frac{1}{3}$: this is unlikely to be a good parameter choice. In that case, we must find $t \leqslant \max\left(\frac{k_2}{k}, \frac{1}{2}\left(1 - \frac{k_1}{k}\right)\right)$. This is easily obtained with a trivial tree, that has only two subtrees: one is obtained by the product of $k_1$ lists (time $\frac{k_1}{k} \leqslant \frac{k_2}{k}$), and the other is an exhaustive search over all $k - k_1$ remaining lists, in time $\frac{1}{2}\left(1 - \frac{k_1}{k}\right)$. There are no repetitions. Notice that this is actually the optimal strategy for $k = 3, 6$ with $k_1 = k_2 = \frac{k}{3}$.

So we can now suppose that $k_1 \geqslant k_2$. We notice that:

$$\frac{1}{2}\left(1 - \frac{k_1}{k}\right) \geqslant \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right) \iff 1 - \frac{2k_1}{k} \geqslant \frac{k_1 - k_2}{k} \iff k \geqslant 3k_1 - k_2 \enspace .$$

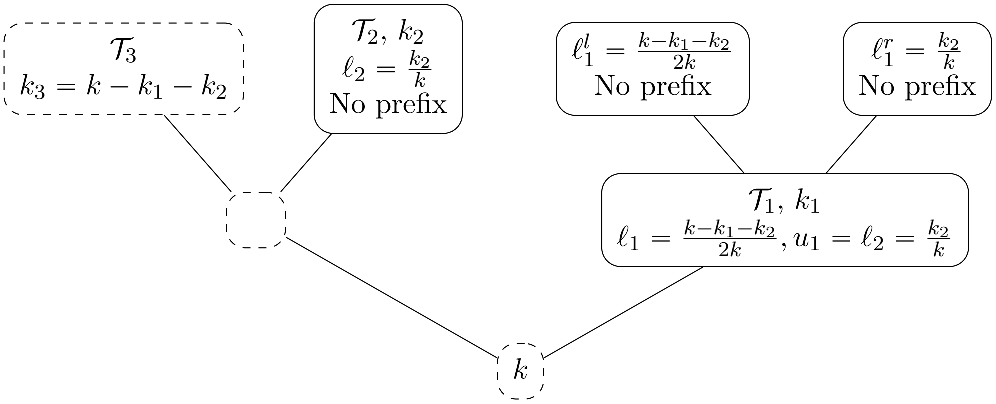


**Figure 6.12:** Tree of Lemma 6.6.

So we next focus on the case $k \leqslant 3k_1 - k_2$ and we try to obtain a complexity $t \leqslant \max\left(\frac{k_2}{k}, \frac{1}{4}\left(1 + \frac{k_1-k_2}{k}\right)\right)$. The merging tree that we use is drawn in Figure 6.12. We attach two subtrees of width $k_2$ and $k_1$ to the main branch, there remains a subtree of width $k - k_1 - k_2$ to explore exhaustively. We build the subtree $\mathcal{T}_2$ in time $\frac{k_2}{k}$, externally, by constructing the product of $k_2$ lists. Then we repeat $\mathcal{T}_1$, which builds a list of size $\ell_1 = \frac{k-k_1-k_2}{2k}$.

With our choice of parameters, we have to iterate:

$$\frac{1}{2}\left(\frac{k_1}{k} - \ell_1\right) = \frac{1}{4k}\left(2k_1 - k + k_1 + k_2\right) = \frac{1}{4k}\left(3k_1 + k_2 - k\right)$$

times, which is positive, since $k \leqslant 3k_1 - k_2 \leqslant 3k_1 + k_2$. In each iteration, we build the subtree $\mathcal{T}_1$ in time $\ell_1$ and exhaust the subtree $\mathcal{T}_3$ with quantum search, with the same time. Thus, the total time complexity is given by:

$$t = \max\left(\frac{k_2}{k}, \frac{1}{2}\left(\frac{k_1}{k} + \ell_1\right)\right) = \max\left(\frac{k_2}{k}, \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right)\right) \quad .$$

Finally, in the case $3k_1 + k_2 \leqslant k$, we notice that $3k_1 \leqslant k$ implies $\frac{1}{2}\left(1 - \frac{k_1}{k}\right) \geqslant \frac{1}{3}$, as it happened before for the case $k_1 \leqslant k_2$. Further, $\frac{1}{2}\left(1 - \frac{k_1}{k}\right) \geqslant \frac{k_1}{k}$. The same strategy works: we build an intermediate subtree with a product of $k_1$ lists, in time $\frac{k_1}{k}$, then look for a collision on it with a single quantum search in the product of the $k - k_1$ remaining lists.

Thus, regardless the choice of $k_1$ and $k_2$, we can meet the time complexity given by Equation (6.11). □

Finally, we observe that the minimization over $k_1, k_2$ of this quantity gives the formula of $\gamma_k$ of Theorem 6.3, finishing the proof of the theorem:

$$\min_{\substack{k_1,k_2 \in \mathbb{N}^2 \\ k_1+k_2 \leqslant k}} \max\left(\frac{k_2}{k}, \frac{1}{2}\left(1 - \frac{k_1}{k}\right), \frac{1}{4}\left(1 + \frac{k_1 - k_2}{k}\right)\right) = \frac{k + \left\lfloor \frac{k+6}{7} \right\rfloor + \left\lfloor \frac{k+1}{7} \right\rfloor - \left\lfloor \frac{k}{7} \right\rfloor}{4k} \quad .$$

The minimum can be reached by choosing:

$$\begin{cases} k_1 = \left\lfloor \frac{3k}{7} \right\rceil \\ k_2 = \left\lfloor \frac{2k}{7} \right\rceil - \left\lfloor \frac{k-1}{7} \right\rceil + \left\lfloor \frac{k-2}{7} \right\rceil \end{cases} ,$$

where $\left\lfloor \frac{3k}{7} \right\rceil$ is the integer that is closest to $\frac{3k}{7}$. We have obtained the tree structure given in Figure 6.8 and Algorithm 6.10 and proven its optimality among merging trees.

### 6.1.8 Optimizing the Time-Memory Product

In the case where the optimization goal is the time-memory product, we will also be able to restrict the space of possible trees and obtain simpler constraints. Let us follow the proof for the time optimization and amend it where necessary.

Lemma 6.2 can be used unchanged, since adding the memory does not change the optimization of the formulas. Next, we find that:

**Lemma 6.7.** *In the time-memory product optimization of Problem 6.2, the memory used is $2^{n/k}$.*

*Proof (informal).* We use at least $2^{n/k}$ memory since this is required to store the lists. If some list in the tree was bigger than that, we would be able to reduce it and transfer this complexity to a repetition loop. This can only improve the time complexity. □

Thus, only algorithms with little memory may be optimal for the time-memory product. We can write down the shape of an optimal tree as follows:

- It has a main branch with $p$ levels, subtrees $\mathcal{T}_i$, and $\mathcal{T}_p$ is a leaf node with $u_p = 0$. Each $\mathcal{T}_i$ is of width $k_i$, and the $k_i$ decrease (otherwise, exchanging two of the subtrees would reduce the time complexity).
- At level $p$ on the left, there is a subtree $\mathcal{T}'_p$ which is of width $k'_p$, with $\sum_{i=1}^{p} k_i + k'_p = k$ .
- Each $\mathcal{T}_i$ induces a repetition loop with $r_i = \frac{1}{2}\left(\frac{k_i}{k} - \ell_i\right)$. Since the $\ell_i$ are of size $\frac{1}{k}$ at most, and $k_i$ decreases with $i$, the number of repetitions decreases with $i$.
- $\forall i, \ell_i \leqslant \frac{1}{k}$ and $u_{i-1} = u_i + \ell_i$ (as before), so we still have $u_i = \sum_{j=i+1}^{p} \ell_j$. Furthermore $u_i + \ell_i \leqslant \frac{k_i}{k}$ .

Thus the time complexity $t$ satisfies the constraints:

$$\begin{cases} \forall i, \frac{1}{k} \geqslant \ell_i \geqslant 0, \frac{k_i}{k} \geqslant \sum_{j=i}^{p} \ell_j \\ t \geqslant \sum_i r_i + \frac{1}{2}\left(1 - \frac{\sum_i k_i}{k}\right) = \frac{1}{2}\left(1 - \sum_i \ell_i\right) \\ \forall 1 \leqslant i \leqslant p-1, t \geqslant \left(\sum_{j=1}^{i} r_j\right) + \frac{1}{2}\max\left(\left(\sum_{j=i+1}^{p} \ell_j\right) - \frac{\lfloor \log_2 k_i \rfloor}{k}, 0\right) + \ell_i \end{cases} \quad (6.12)$$

Contrary to the previous minimization, we will find that the depth of the tree increases with $k$, and the merging is not as trivial as before. By bounding the $\ell_i$, we obtain that $\frac{k_1}{k} \geqslant \sum_{j=1}^{p} \ell_j$ and $\frac{p}{k} \geqslant \sum_{j=1}^{p} \ell_j$. Thus, $t$ is lower bounded by:

$$t \geqslant \frac{1}{2}\left(1 - \max\left(\frac{k_1}{k}, \frac{p}{k}\right)\right) \quad .$$

This is to say, the tree should have the bigger depth possible, while at the same time, having the widest possible subtree at level 1. In practice, we observe that the constraints (6.12) are sufficient to obtain the best trees. The successive $k_i$ are strictly decreasing, except in some cases. We can still find optimal trees by enforcing: $|k_{i+1} - k_i| \leqslant 1$, and forbidding to have three successive equal $k_i$, which reduces greatly the search space. Both $k_1$ and the number of levels in the tree are of order $\mathcal{O}(\sqrt{k})$.

## 6.2 Applications

In this section, we elaborate on different problems that can be reduced to $k$-XOR or $k$-SUM. We show that, more generally, the optimal algorithms that we presented above apply to the class of *bicomposite* problems studied by Dinur, Dunkelman, Keller, and Shamir [Din+12].

### 6.2.1 Bicomposite Problems

The classical Dissection algorithms of [Din+12] are not formulated as Unique $k$-XOR algorithms, although they can be used in this context. They solve a slightly more general problem that we will now define. In this definition, the notation $r$ assumes the same role as $k$.

**Problem 6.3** ($r$-composite search)**.** *Let $(X_i, 1 \leqslant i \leqslant r+1)$ be a family of $r+1$ finite sets of same cardinality (say, $2^n$ for some parameter $n$). Let $\mathcal{F}_1, \ldots, \mathcal{F}_r$ be $r$ families of functions: $f_i \in \mathcal{F}_i : X_i \to X_{i+1}$, e.g., permutations or random functions, with the same cardinality $2^n$.*

*Let $(x_1^1, \ldots, x_r^1) \in X_1$ and $(x_1^{r+1}, \ldots, x_r^{r+1}) \in X_{r+1}$. Find $f_1 \in \mathcal{F}_1, \ldots, f_r \in \mathcal{F}_r$ such that:*

$$\forall i, (f_r \circ \ldots \circ f_1)(x_i^1) = x_i^{r+1} \quad .$$

In other words, we are given lists of transitions (the families $\mathcal{F}_i$) that act on $r$ inputs (the $x_i^1$), and we want to find a sequence of such transitions that brings these inputs to $r$ given targets. The *bicomposite* nature of this problem comes from the orthogonality between the choices of the transitions and the targets. For example, assume that $x_1^{i+1}, \ldots, x_i^{i+1}$ are given, then we can find immediately the sequence of $i$ transitions $f_i \circ \ldots \circ f_1$ that lead to these values: it is not necessary to guess the full tuple $x_1^{i+1}, \ldots, x_r^{i+1}$.

*Remark* 6.7. This property of *partial guesses* is the only one necessary for the definition of the problem, not that the $f_i$ are permutations or random functions.

A prominent example of a bicomposite problem is the case where all the $f_i$ are permutations: this is the multiple-encryption problem.

**Problem 6.4** ($r$-encryption)**.** *Let $E^1, \dots, E^r$ be $r$ block ciphers on $n$ bits, indexed by key spaces of the same size $2^n$. Assume that we are given $r$ plaintext-ciphertext pairs $(p_i, c_i)$, encrypted by the composition of the $E^i$ under a sequence of independent keys $k_1, \dots, k_r$:*

$$\forall i, c_i = \left(E^r_{k_r} \circ \ldots \circ E^1_{k_1}\right)(p_i) \enspace ,$$

*then retrieve $k_1, \dots, k_r$.*

The number of given plaintext-ciphertext pairs is enough to discriminate the good sequence of keys with constant probability, as each key is of $n$ bits and each plaintext is of $n$ bits.

**Relation with $k$-XOR.** The Unique $k$-XOR and $k$-SUM problems are naturally $k$-composite, and any algorithm that solves $k$-composite problems will also solve them.

**Proposition 6.4.** *Unique $k$-XOR (with random input lists) can be reduced to $k$-composite search.*

*Proof.* Given an instance of unique $k$-XOR, with $k$ lists $\mathcal{L}_1, \dots, \mathcal{L}_k$ of $n$-bit strings, we are looking for $y_1 \in \mathcal{L}_1, \dots, y_k \in \mathcal{L}_k$ such that $\bigoplus_i y_i = 0$. In the asymptotic world, we assume that $n$ is divisible by $k$.

For all $1 \leqslant i \leqslant k$, we set $x_i^1 = (0_{n/k}, i-1)$. We then define the families of functions $\mathcal{F}_1, \dots, \mathcal{F}_k$ as follows: for each $j$, $f_j \in \mathcal{F}_j$ is the addition of some bits of some element $y_j \in \mathcal{L}_j$. More precisely, the initial state:

$$x_1^1, \dots, x_k^1 = (0_{n/k}, 0), (0_{n/k}, 1), (0_{n/k}, 2), \dots, (0_{n/k}, k-1)$$

will be transformed by $f_1$ by adding the $n/k$ first bits of $y_1$ to $x_1^1$, the $n/k$ second bits to $x_2^1$, *etc.* Since the lists are assumed to contain random bit-strings, the knowledge of partial intermediate states is enough to know which elements $y_j$ led to these values. □

It is interesting to notice that the bicomposite nature of the problem stems from our capacity to divide our intermediate state $x^i$, which is an $n$-bit string, into a product of $k$ strings of $n/k$ bits. This is also the case in the $r$-encryption problem, since $r$ $\{\text{plaintext}, \text{ciphertext}\}$ pairs are given. A $k$-composite algorithm applied to the problem will make some guesses for partial intermediate states, which correspond to merging the lists $\mathcal{L}_i$ with arbitrary prefixes. In contrast, the commutativity of the $+$ or $\oplus$ operations is not necessary. For example, given a permutation $\sigma \in S_n$, finding a composition of $r$ permutations $\sigma_i$ of $S_n$ such that $\sigma_r \circ \ldots \circ \sigma_1 = \sigma$ is an $r$-composite problem.

This relation goes the other way. Instead of trying to prove a full-fledged correspondence between unique $r$-XOR algorithms of our framework and $r$-composite problems, we can focus on the optimal algorithms presented in Section 6.1.6. For simplicity, we consider a single family of permutations $\mathcal{F}$, of size $2^n$, although this works as well with $r$ families and for one-way functions. We write down Algorithm 6.11. Its time complexity is computed with the same formula as the $k$-XOR variant.

**Algorithm 6.11** Quantum $r$-composite search.

**Input:** two $r$-tuples $x^1 = (x_1^1, \ldots, x_r^1)$ and $x^{r+1} = (x_1^{r+1}, \ldots, x_r^{r+1})$, a family of permutations $\mathcal{F}$ indexed by a "key" $k \in K$, with quantum oracle access to:

$$x, k \mapsto f_k(x)$$

**Output:** a sequence of "keys" $k_1, \ldots, k_r$ such that $x^1$ is mapped to $x^{r+1}$ by $f_{k_r} \circ \ldots \circ f_{k_1}$

```
Select $r_1, r_2$ with Equation 6.5
Build a list $\mathcal{L}_1$:
    $\mathcal{L}_1 = \{(f_{k_{r_2}} \circ \ldots \circ f_{k_1})(x^1), k_{r_2}, \ldots, k_1 \in K\}$
    ▻ Thus, the list contains all possible choices for the $r_2$ first steps
Build a list $\mathcal{L}_3$:
    $\mathcal{L}_3 = \{(f^{-1}_{k_{r-r_2}} \circ \ldots \circ f_r^{-1})(x^{r+1}), k_{r-r_2+1}, \ldots, k_r \in K\}$
    ▻ Thus, the list contains all possible choices for the $r_2$ last steps
Sample $y_1, \ldots, y_{r_2}$ such that
    ▻ These guesses are from the intermediate state $x_{r-r_1}$
    Define: $\mathcal{L}_2$ the list of all key sequences for steps $r - r_1 + 1, \ldots, r - r_2 + 1$
    Sample Sublists $\mathcal{L}_2'$ of $\mathcal{L}_2$ of size $r - r_1 - r_2$ such that
        Compute the list $\mathcal{L}_{23}$ of all key sequences from $\mathcal{L}_2' \times \mathcal{L}_3$ for steps $r-r_1+1, \ldots, r$ such that $y_1, \ldots, y_{r_2}$ is mapped to $x_1^{r+1}, \ldots, x_{r_2}^{r+1}$.
        ▻ This is done by computing the images of $y_1, \ldots, y_{r_2}$ by all keys and trying to match an element of $\mathcal{L}_3$
        Sample Key sequences for step $r_2 + 1, \ldots, r - r_1$ such that
            ▻ There remains $r - r_1 - r_2$ subkeys to try
            Compute the inverse by $f_{k_{r_2+1}} \circ \ldots \circ f_{k_{r-r_1}}$ of $y_1, \ldots, y_{r_2}$
            Match against the values in $\mathcal{L}_1$
            Obtain a sequence of keys for steps $1, \ldots, r_2, r_2 + 1, \ldots, r - r_1$ that map $x_1^1, \ldots, x_{r_2}^1$ to $y_1, \ldots, y_{r_2}$
            Compute the rest of the intermediate state $x^{r-r_1}$ and try to match against $\mathcal{L}_{23}$
            if There is a match then
                Exit and return the full key sequence
        EndSample
    EndSample
EndSample
```

**Theorem 6.5.** *For any $r \geqslant 2$, let $\gamma_r$ be the optimal time complexity exponent for unique $r$-XOR with merging trees, given by Theorem 6.3. Then there exists a quantum algorithm for $r$-composite search, with domain size $2^n$, of time complexity $\mathcal{O}(2^{\gamma_r r n})$.*

## 6.2.2 Solving the Multiple-encryption Problem

We now focus on Problem 6.4. Classically, the best time complexity to date is essentially $\mathcal{O}\big(2^{\lceil r/2 \rceil n}\big)$ and the question is to obtain better time-memory trade-offs than a simple meet-in-the-middle approach, as it is the case in [Din+12]. Let us start with the example of Schroeppel and Shamir's algorithm, that can be used to solve 4-encryption, as done in [Din+12], in time $\mathcal{O}\big(2^{2n}\big)$ and memory $\mathcal{O}(2^n)$. For ease of notation, we assume that the successive encryption functions are the same block cipher $E$, with blocks and keys of $n$ bits. Notice that in Algorithm 6.12, decryptions are required. However, if we replaced $E$ by a one-way function, we could tabulate its inverse.

**Algorithm 6.12** Schroeppel and Shamir's algorithm for 4-encryption.

```
Input:  4 plaintext-ciphertext pairs $(p_i, c_i)$ which form each an $n$-bit condition
Output:  4 $n$-bit keys $k_1, k_2, k_3, k_4$ such that $E_{k_4} \circ E_{k_3} \circ E_{k_2} \circ E_{k_1}(p_i) = c_i$
for all Intermediate state $s$ of $n$ bits do
    Compute the list $\mathcal{L}^s_{12}$ of all keys $(k_1, k_2)$ such that $E_{k_2} \circ E_{k_1}(p_1) = s$
                                                  ▷ $\mathcal{L}^s_{12}$ is of average size $2^n$
    Compute also $E_{k_2} \circ E_{k_1}(p_i)$ for the other plaintexts
    Compute the list $\mathcal{L}^s_{34}$ of all keys $(k_3, k_4)$ such that $E_{k_4} \circ E_{k_3}(s) = c_1$
    Compute also $(E_{k_4} \circ E_{k_3})^{-1}(c_i)$ for the other ciphertexts
    Find $(k_1, k_2) \in \mathcal{L}^s_{12}$ and $(k_3, k_4) \in \mathcal{L}^s_{34}$ such that all values in the middle match
    if there exists such a solution then                  ▷ Happens for a single $s$
        return the sequence of keys
```

**Results in the quantum setting.** Classically, using more but smaller keys makes the problem easier from the point of view of memory complexity. The best known algorithms do not obtain a better time than the simple meet-in-the-middle technique. In the quantum setting, this will be different.

In [Kap14], Kaplan proves that 2-encryption is (quantumly) equivalent to element distinctness. For $r = 4$, it must be remarked that the 4-XOR algorithm of [Ber+13] cannot be used to attack 4-encryption. Indeed, in the quantum optimization of [Ber+13], the size of the "intermediate value" that is guessed is not a multiple of $n$ bits. This has no consequence on the Unique $k$-XOR problem, but if we try to translate the algorithm to attack multiple-encryption, we cannot solve efficiently the smaller meet-in-the middle problems. It would require to produce efficiently (in time $2^{0.8n}$), from $2^{0.8n}$ choices of $k_1$ and $k_2$, the list of $2^{0.8n}$ pairs $k_1, k_2$ such that $E_{k_1} \circ E_{k_2}(p)$ has some fixed $0.8n$-bit prefix.

Thus, the best quantum algorithm for 4-encryption to date ($\mathcal{O}\big(2^{5n/4}\big)$ for 4 keys of $n$ bits) is given by Algorithm 6.13, which adapts Algorithm 6.7.

**Algorithm 6.13** Quantum 4-encryption algorithm.

```
Input: 4 plaintext-ciphertext pairs $(p_i, c_i)$ which form each an $n$-bit condition
Output: 4 $n$-bit keys $k_1, k_2, k_3, k_4$ such that $E_{k_4} \circ E_{k_3} \circ E_{k_2} \circ E_{k_1}(p_i) = c_i$
Compute $E_{k_2}^{-1}(c_1)$ for all $k_2$
for all Intermediate state $s$ of $n$ bits do
    for all $x$ of $n/2$ bits do
        Compute the list $\mathcal{L}_{12}^s$ of all keys $(k_1, k_2)$ such that $E_{k_2} \circ E_{k_1}(p_1) = s$ and $k_1$ has prefix $x$
                                                        ▻ $\mathcal{L}_{12}^s$ is of average size $2^{n/2}$
        Encrypt the other plaintexts
        for all $k_3$ do
            Find $k_4$ such that $E_{k_3}(s) = E_{k_4}^{-1}(c_1)$
            ▻ We now have that $E_{k_4} \circ E_{k_3} \circ E_{k_2} \circ E_{k_1}(p_1) = c_1$ for each $k_1, k_2$ in $\mathcal{L}_{12}^s$
            Decrypt the other ciphertexts with $k_3, k_4$
            Find if there exists $k_1, k_2$ in $\mathcal{L}_{12}^s$ such that the intermediate states match.
```

For other values of $r$, the correspondence with $r$-XOR gives the best quantum time complexities known for $r$-encryption, and shows that contrary to the classical setting, the time complexity decreases significantly with respect to 2-encryption.

### 6.2.3 Improved Quantum Time-Memory Trade-off for Subset-sums

The *random* subset-sum problem is the following.

**Problem 6.5.** *Given $a_1, \ldots, a_n, t$ chosen uniformly at random from $\mathbb{Z}_\ell$, find a subset of indices $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = t \mod 2^\ell$.*

The hardness of this problem is related to the *density* $n/\ell$. When $\ell = poly(n)$, i.e., the density is close to 1, we expect a single solution with high probability. Finding this solution can be recast as a $k$-SUM problem *for any* $k$. Indeed, it suffices to separate the set $\{1, \ldots, n\}$ into $k$ disjoint parts $J_1 \cup \ldots \cup J_k$ and to start from the lists $L_1, \ldots, L_k$, with list $L_j$ containing all the sums $\sum_{i \in I} a_i$ for $I \subset J_j$ (we may assume that $n$ is a multiple of $k$ for this purpose).

The best classical worst-case algorithm for subset-sum known is Schroeppel and Shamir's algorithm. But in the case of random subset-sums, an algorithm with time $\widetilde{\mathcal{O}}(2^{0.337})$ was given by Howgrave-Graham and Joux [HJ10], later improved to $\widetilde{\mathcal{O}}(2^{0.291n})$ by Becker, Coron, and Joux [BCJ11], then $\widetilde{\mathcal{O}}(2^{0.283n})$ [Bon+20].

Quantum algorithms for the problem include those of [Ber+13; HM18] and [Bon+20]. The best quantum time complexity given in [Bon+20] is $\widetilde{\mathcal{O}}(2^{0.216n})$, using as much QRAQM.

Using the extended merging trees for $k$-SUM, we can reach a better quantum time – memory product (this was also the case in [Din+12] for classical algorithms). We have seen that $k = 17$ minimizes the time-memory product, with a time $\mathcal{O}\left(2^{6n/17}\right)$ and a memory $2^{n/17}$. The product is $\mathcal{O}\left(2^{7n/17}\right) = \mathcal{O}\left(2^{0.4118n}\right)$, which is less than the exponent $0.423n$ obtained with the dedicated methods.

### 6.2.4 Quantum Algorithms for LPN and LWE

Merging algorithms can be used in the context of Learning Parity with Noise (LPN).

**Problem 6.6** (Learning Parity with Noise (in dimension $n$))**.** *We can query* noisy samples *of the form* $(a, a \cdot s \oplus e)$ *where* $s \in \{0,1\}^n$ *is a secret,* $a \in \{0,1\}^n$ *is chosen uniformly at random and* $e \in \{0,1\}$ *is a Bernoulli noise:* $e \sim Ber_p$*, i.e.,* $\Pr(e = 1) = p$ *for some constant* error rate $p$. *We must find* $s$*.*

Learning With Errors (LWE) is a generalization of LPN from $\mathbb{F}_2$ to $\mathbb{F}_q$ for some prime $q$. The hardness of this problem for classical and quantum computers underlies several proposals in the NIST PQC project.

In [BKW03], Blum, Kalai, and Wasserman introduced an algorithm solving LPN in time $\mathcal{O}\left(2^{n/\log n}\right)$, using $2^{n/\log n}$ samples. Their idea is to combine samples: given $(a_1, a_1 \cdot s \oplus e_1)$ and $(a_2, a_2 \cdot s \oplus e_2)$ one can compute $(a_1 \oplus a_2, (a_1 \oplus a_2) \cdot s \oplus e_1 \oplus e_2)$. This combination brings the noise closer to uniform, as prescribed by the Piling-Up Lemma.

**Lemma 6.8** (Adding Bernoulli noises)**.** *Let* $e_1, \ldots, e_k \sim Ber(p)$ *be Bernoulli noises of* correlation $\epsilon = 1 - 2p$. *Then* $e_1 \oplus \ldots \oplus e_k \sim Ber(p_k)$ *is a Bernoulli noise of correlation* $\epsilon^k$*, that is,* $p_k = \frac{1}{2}\left(1 - (1-2p)^k\right)$*.*

The goal is to produce sufficiently many sums of $a_i$ with almost all bits to zero, so that one can learn a bit of $s$ from the samples gathered. If there was no noise, this would require $2^{\sqrt{n}}$ samples with Wagner's algorithm. However, the noise limits the number of samples that we can sum together. The same principle applies to LWE, although we focus on LPN for simplicity.

In its original version, the BKW algorithm uses $2^{n/\log n}$ samples and memory. It starts from the list of samples and repeatedly finds partial collisions, canceling $n/\log n$ bits in the $a_i$, until it produces a list of $2^{n/\log n}$ samples with a single nonzero bit. Thus, the algorithm can be seen as a binary merging tree with $\log n$ levels.

The *c-sum-BKW algorithm* is a variant proposed by Esser et al. [Ess+18], that combines $c > 2$ samples at a time, and replace the 2-list merges in BKW by a $c$-list algorithm. This allows to reduce the memory used, which is crucial for practical implementations of the BKW algorithm. This also reduces the number of samples: we start from a smaller list. The $c$-sum-BKW algorithm is built upon the $c$-SUM-Problem as defined in [Ess+18, Definition 3.1]:

> *Given a list $\mathcal{L}$ of $N$ uniformly random b-bit strings, given $t \in \{0,1\}^n$, find at least $N$ distinct c-tuples of elements of $\mathcal{L}$ that XOR to $t$.*

They use the Dissection algorithms, with a memory limited to $N$. They also prove that:

**Theorem 6.6** ([Ess+18])**.** *If there exists an algorithm solving c-SUM in time $T_{c,N}$ and memory $M_{c,N}$ with overwhelming probability, for $b = \log c \frac{n(1+\epsilon)}{\log n}$ and $N \geqslant 2^{\frac{b+c\log c+1}{c-1}}$, then there exists an algorithm solving LPN in dimension $n$ in time $T_{c,N}^{1+o(1)}$ and memory $M_{c,N}^{1+o(1)}$.*

Theorem 6.6 applies even if the building block is a quantum algorithm. The authors proposed to solve $c$-SUM with a naive Grover search in the QRACM model: they store $\mathcal{L}$ in QRACM and perform a Grover search on all $c-1$ tuples of $\mathcal{L}$, for those who XOR to an element of $\mathcal{L}$. The memory used is $N$. As the parameters are tailored for $N$ solutions in total, the quantum time complexity is $N^{c/2-1}$ for a single solution and $N^{c/2}$ for all of them.

**Better trade-offs.** We can answer the open question of [Ess+18, Section 1] and improve the quantum-BKW algorithm with our new optimizations. We are in a situation in which the input list is of size $N_c$ and there are $N_c$ solutions to recover. It is as if we were solving a $c$-XOR problem on $n$ bits with $c$ lists of size $N_c = 2^{n/(c-1)}$ each, and wanted all the $2^{n/(c-1)}$ expected solutions. Furthermore, we limit the memory (QRAQM or QRACM) used to $N_c$. We simply solve the problem $\widetilde{\mathcal{O}}\left(2^{n/(c-1)}\right)$ times, as in the naive Grover case.

**Table 6.3:** Improvements on the quantum-BKW algorithm of [Ess+18] (see Table 1 in [Ess+18]).

| | Previous (naive + Grover) | | New results | | |
|---|---|---|---|---|---|
| $c$ | Memory | Time | Memory | Time | Time exponent |
| 3 | $N_c$ | $N_c^{3/2}$ | $N_c$ | $N_c^{5/3}$ | $5/3 = 1.667$ |
| 4 | $N_c$ | $N_c^{2}$ | $N_c$ | $N_c^{13/7}$ | $13/7 = 1.857$ |
| 5 | $N_c$ | $N_c^{5/2}$ | $N_c$ | $N_c^{2}$ | 2 |
| 6 | $N_c$ | $N_c^{3}$ | $N_c$ | $N_c^{5/2}$ | $5/2 = 2.5$ |
| 7 | $N_c$ | $N_c^{7/2}$ | $N_c$ | $N_c^{11/4}$ | $11/4 = 2.75$ |
| 8 | $N_c$ | $N_c^{4}$ | $N_c$ | $N_c^{3}$ | 3 |

### 6.2.5 Discussion

In Chapter 5 and this chapter, we have seen that some merging problems ($k$-XOR, $k$-SUM or bicomposite) can benefit from a significant quantum speedup. However, this class of quantum merging algorithms contains many counterintuitive speedups, and we managed to explore them only thanks to an automatic search for merging strategies, which provided us with some intuition for the best strategies.

While it is easy to account for these algorithms after having found them, we can only claim their optimality *up to the constraints* which were used in the search. In particular, removing only a little constraint from [NS20] allowed us to obtain the exponent $\frac{2}{7}$, which sets at $\mathcal{O}\left(2^{2n/7}\right)$ the lowest time complexity for Unique $k$-XOR, for any $k$.

**Open Problem 6.1.** *Can we extend further the search space, and reach an exponent below $\frac{2}{7}$?*

Again remains the question whether other families of quantum algorithms, for example, quantum walks, could reach better complexities. Even the quantum walk for 4-XOR of [Ber+13] is beaten by quantum merging when $k$ is a multiple of 4 greater than 20, as our exponent converges towards $\frac{2}{7}$.

**Open Problem 6.2.** *Does better quantum algorithms for $k$-XOR exist, not based on the quantum merging framework, for a generic $k$?*

We conjecture that this should be true at least in the single-solution case, because quantum walks offer generally an incompressible speedup when the sizes of the lists to merge are very constrained: this is what happens for $k = 4$, and this is also what we observed when comparing quantum walks and quantum merging in the case of the dense subset-sum problem [Bon+20].

# Chapter 7
# The Offline Simon's Algorithm

This chapter is dedicated to the *Offline Simon's Algorithm*, that we introduced in [Bon+19], and its applications. This new quantum algorithm is dedicated to key-recovery attacks against symmetric cryptographic constructions, including some of those seen in Chapter 4. It provides the first application of Simon's algorithm in the Q1 setting, solving a problem that had remained open since the dawn of Q2 attacks, and showing that the structure behind quantum polynomial-time attacks in the Q2 setting can also be leveraged with classical queries only.

We give a presentation of this algorithm different from [Bon+19], that emphasizes its similarities with the *Grover-meets-Simon* attack of Leander and May. Next, we show how it improves the query complexity of some Q2 attacks, and how to apply it in the Q1 setting.

**Contents**

## 7.1 The Offline Simon's Algorithm

In the Grover-meets-Simon problem (Problem 4.1 in Section 4.1.5), we search a family of functions for the single periodic one. Our results in [Bon+19] focus on a variant of this problem, where the search space has additional structure.

**Problem 7.1** (Asymmetric Search of a Period)**.** *Let* $f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^\ell$ *and* $g : \{0,1\}^n \to \{0,1\}^\ell$ *be two functions. Assume that there exists exactly one* $i_0 \in \{0,1\}^m$ *such* *that $f(i_0,\cdot)\oplus g$ has a hidden period, i.e.: $\forall x \in \{0,1\}^n, f(i_0,x)\oplus g(x) = f(i_0,x\oplus s)\oplus g(x\oplus s)$ for some $s$. Find $i_0$.*

The formulation of Problem 7.1 captures most cryptanalytic scenarios such as the FX construction, and the algorithm presented in this chapter can improve Grover-meets-Simon in this context.

With this formulation, in the functions $f(i,\cdot)\oplus g$, the function $g$ intervenes always in the same way. In Leander and May's Grover-meets-Simon algorithm, at each iteration of the search, we perform $\mathcal{O}(n)$ new queries to $g$. But these queries are always the same: they produce a uniform superposition $\sum_x |x\rangle |g(x)\rangle$, before XORing $f(i,x)$ into it and running Simon's algorithm.

Instead of re-constructing the same states $\sum_x |x\rangle |g(x)\rangle$ at each iteration, and throwing them away, our idea is to make the queries to $g$ onces and for all, at the beginning of the search, and *store* these states.

### 7.1.1 Description of the Algorithm

We can represent our intuition with a single quantum circuit. Consider two successive iterations of the Grover search for $i_0$, with a register for $i$ and ancilla registers required for the test. This test (Circuit 4.1) contains a call to the quantum operator $\mathsf{QSimon}$ and oracle calls to $x \mapsto (f_i \oplus g)(x)$. These oracle calls are realized by first computing $g(x)$ in an output register, then XORing $f_i(x)$ into this register (we simply draw a box for this second step).

We draw these two iterations in Circuit 7.1. The successive tests use ancilla qubits initialized to $|0^n\rangle$ and put them back to this state. We remark that they do not need to. In fact, the operations in the dotted box ($O_g$, then $\mathsf{H}$, then $\mathsf{H}$, then $O_g$) are redundant, and they cancel out.

Thus, almost all queries to $g$ are removed from the algorithm, and they only need to be made at the beginning of the Grover search, in an *offline* manner. This changes dramatically our point of view: the ancilla qubits that were before put back to $|0\rangle$ at each iteration contain now the state:

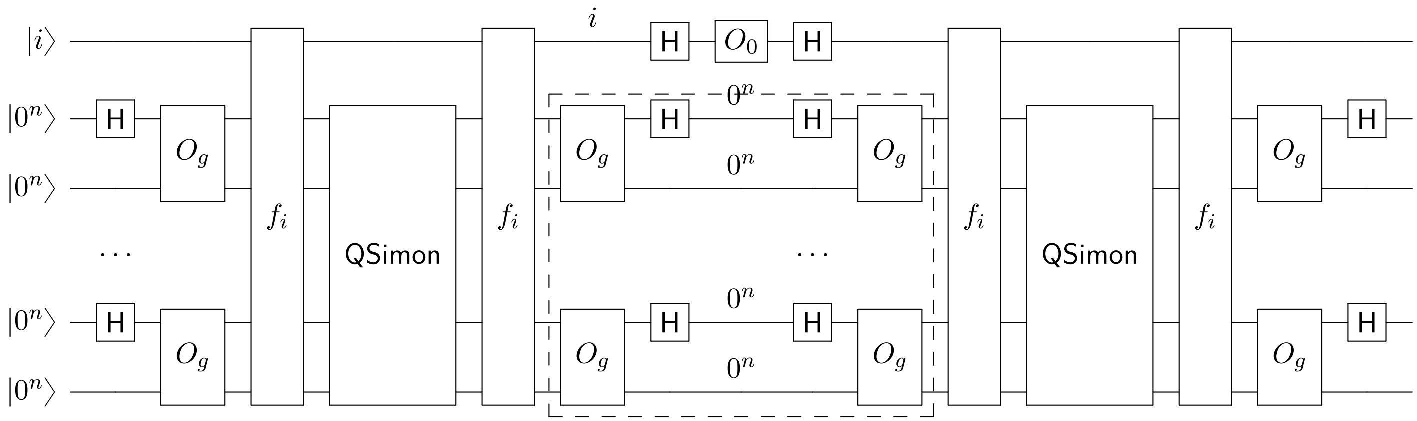
$$|\psi_g\rangle = \bigotimes^{n+r} \sum_x |x\rangle |g(x)\rangle \quad .$$

This state $|\psi_g\rangle$, that we name the *g-database*, contains all the information that we need about this function $g$. We summarize our procedure in Algorithm 7.1.

**Theorem 7.1** (Offline Simon)**.** *Consider the setting of Problem 7.1 and assume that $f(i,\cdot)$ and $g$ satisfy the following condition:*

$$\exists a > 0, \max_{\substack{i \in \{0,1\}^m \setminus \{i_0\} \\ t \in \{0,1\}^n \setminus \{0^n\}}} \Pr_{x \leftarrow \{0,1\}^n} \left[ (f(i,\cdot)\oplus g)(x\oplus t) = (f(i,\cdot)\oplus g)(x) \right] \leqslant \frac{an}{2^n} \quad . \tag{7.1}$$

*Then there exists a quantum algorithm running in expected time $\mathcal{O}\big((n+m)^3 2^{m/2}\big)$, making $\mathcal{O}\big((n+m)2^{m/2}\big)$ queries to $f$* and *$\mathcal{O}(n+m)$ queries to $O_g$, that outputs $i_0$.*



**Quantum Circuit 7.1:** Two iterations of Grover-meet-Simon, with the internal cancellation of $O_g$

*Proof.* The condition and proof are exactly the same as in the Grover-meet-Simon attack, since we simply remove the queries to $O_g$ inside the algorithm. By Lemma 1.4, the condition (7.1) is satisfied for families of random functions of which a single one is a random periodic function. We can apply Theorem 4.1. In particular, our algorithm simply runs a Grover search as if QSimon was perfect, then measures and outputs a good result with high probability. □

**Algorithm 7.1** The offline Simon's algorithm OfflineSimon. Steps 5 to 8 simply realize an iteration of the quantum search for $i_0$.

```
1: Start in the all-zero state.   ▷ $|0^m\rangle \bigotimes^{n+r} |0^n\rangle |0^n\rangle$
2: Apply Hadamard gates and $O_g$ to create the $g$-database state.
       ▷ $|0^m\rangle \bigotimes^{n+r} \left(\sum_x |x\rangle |g(x)\rangle\right)$
3: Initialize the first register with the uniform superposition of indices:
       ▷ $\left(\sum_{i \in \{0,1\}^m} |i\rangle\right) \otimes |\psi_g\rangle$
4: Repeat $\mathcal{O}(2^{m/2})$ times
5:     Compute $f$, controlled on the value of $i$, directly into the database.
           ▷ $\sum_i \alpha_i |i\rangle \bigotimes^{n+r} \left(\sum_x |x\rangle |g(x) \oplus f_i(x)\rangle\right)$
6:     Apply the circuit QSimon
7:     Uncompute $f$
8:     Apply $\mathsf{H}^{\otimes m} O_0 \mathsf{H}^{\otimes m}$ on the first register
9: EndRepeat
10: Measure the index register $i$, return $i$
```

## 7.1.2 Consequence in the Q2 Setting

The obvious advantage of OfflineSimon in the context of Q2 attacks is to reduce drastically the number of queries required (although the time complexity is unchanged). In what follows, we consider that all conditions of Theorem 7.1 are met, and that $m = \mathcal{O}(n)$.

Let us recall the FX construction:

$$\mathrm{FX}_{k,k_{in},k_{out}}(x) := E_k(x \oplus k_{in}) \oplus k_{out} \enspace .$$

Then, similarly as in the Grover-meet-Simon attack, we consider:

$$i, x \mapsto \mathrm{FX}(x) \oplus E_i(x) \enspace .$$

This function is periodic (of period $k_{in}$) if and only if $i = k$. It is the sum of $f(i, x) := E_i(x)$, that depends on $i$ but not on the secret, and of $g(x) := \mathrm{FX}(x)$, that does not depend on $i$, but contains the secret. Finding the key $k$ is an instance of Problem 7.1, and Algorithm 7.1 can be applied. Although FX and $E$ are random permutations, and not random functions, the condition (7.1) is still valid. Once $k$ has been found, Simon's algorithm can recover $k_{in}$.

**Proposition 7.1.** *The keys $k, k_{in}, k_{out}$ can be recovered in time $\mathcal{O}\big((n+m)^3 2^{m/2}\big)$ by making $\mathcal{O}(n+m)$ superposition queries to $\mathrm{FX}_{k,k_{in},k_{out}}$.*

## 7.2 New Q1 Attacks based on Simon's Algorithm

Besides reducing the number of queries in Q2 attacks, we can use this algorithm to propose new improvements in the Q1 setting.

### 7.2.1 Using Classical Queries Only

The main idea involved here is to substitute the superposition queries to $g$ by classical queries, that we use to build artificially the $g$-database $|\psi_g\rangle$. A superposition on the whole domain of $g$ requires too many queries and is generally too expensive. But the structure of the problem will often allow us to exploit superpositions on a subset of the domain.

**Lemma 7.1.** *Let $g : \{0,1\}^n \to \{0,1\}^n$ be any function computable in time $\mathsf{poly}(n)$. There exists a quantum procedure that creates $\sum_{x \in \{0,1\}^n} |x\rangle |g(x)\rangle$ using less than $(2n-3)2^n$ Toffoli gates, $n2^n$ NOT gates, $n2^n$ $\mathsf{CNOT}$ gates, $n-1$ ancillas qubits, $2^n$ classical evaluations of $g$ and only $\mathsf{poly}(n)$ classical memory.*

*Proof.* We reuse the same circuits as in QRACM emulation (Lemma 1.2). Indeed, building the state $\sum_{x \in \{0,1\}^n} |x\rangle |g(x)\rangle$ is the same task as performing a quantum random-access call to the codebook of $g$. Furthermore, as this codebook needs only to be read once, we shall not store it in classical memory. □

Previously, we remarked that the algorithm was always performing the same queries to $g$, and we stored these queries in the $g$-database. We are now building the same state $|\psi_g\rangle$ with mere classical access, confirming the asymmetry between $g$ and $f$ in the formulation of Problem 7.1. As only $g$ contains a secret, we are now falling in the Q1 setting.

### 7.2.2 Trade-offs for the Even-Mansour Construction

We have mentioned the Even-Mansour construction [EM97] in Section 4.1.2. In the Q2 setting, it is broken in polynomial time due to its strong algebraic structure. The best Q1 time-data trade-off to date is $T^2 \cdot D = 2^n$, and reaches a minimal $T = 2^{n/3}$. But prior to our results, this trade-off required a massive use of QRACM. Trying to overcome this requirement gives the trade-offs of [HS18a].

**New attack.** We will now show how to obtain the trade-off $T^2 \cdot D = 2^n$, the same as the attack of [KM12], up to a polynomial factor, using only $\mathsf{poly}(n)$ qubits and classical memory.

Contrary to previous work in the Q1 setting, we use the algebraic structure of the primitive. We artificially divide the $n$-bit key $k_1$ in $k_1^{(1)}$ of $u$ bits and $k_1^{(2)}$ of $n-u$ bits, in order to obtain an instance of Problem 7.1. If we define $f : \{0,1\}^{n-u} \times \{0,1\}^u \to \{0,1\}^n$ by $f(i,x) = \Pi(x\|i)$ and $g : \{0,1\}^u \to \{0,1\}^n$ by $g(x) = \mathrm{EM}_{k_1,k_2}(x\|0^{n-u})$, then $f(i,\cdot) \oplus g$ is periodic if and only if $i = k_1^{(2)}$, of period $k_1^{(1)}$. Indeed:

$$f(k_1^{(2)}, x) \oplus g(x) = P(x\|k_1^{(2)}) \oplus P((x \oplus k_1^{(1)})\|k_1^{(2)}) \oplus k_2 \ .$$

This idea is depicted in Figure 7.1 and detailed in Algorithm 7.2.

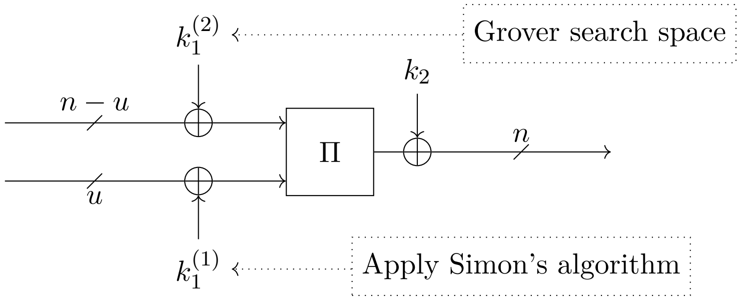


**Figure 7.1:** Idea of our Q1 attack on the Even-Mansour cipher, using the Offline Simon's algorithm.

**Algorithm 7.2** Q1 attack on Even-Mansour.

```
Input: parameter $u$, classical query access to $\mathrm{EM}_{k_1,k_2}$
Returns: $k_1$
Start in the all-zero state
Build the database state for the function: $x \mapsto \mathrm{EM}(x\|0^{n-u})$
Search for $i$ such that $x \mapsto \mathrm{EM}(x\|0^{n-u}) \oplus \Pi(x\|i)$ is periodic, using Algorithm 7.1
Now $i = k_1^{(2)}$. Find $k_1^{(1)}$ with Simon's algorithm.
```

**Proposition 7.2.** *Algorithm 7.2 returns $k_1$ in quantum time:*

$$\underbrace{\mathcal{O}\left(n^2 2^u\right)}_{\textit{Database}} + \mathcal{O}\left( \underbrace{2^{\frac{n-u}{2}}}_{\textit{Grover iterates}} \times \underbrace{n^3}_{\textit{Simon's circuit}} \right) ,$$

*classical memory* $\mathsf{poly}(n)$*, quantum memory* $\mathcal{O}\left(n^2\right)$ *(qubits of the database) and using* $\mathcal{O}(2^u)$ *classical chosen-plaintext queries to* EM.

*Proof.* Since the functions that we consider in the Grover search have $u$-bit inputs, and the search runs for $2^{\frac{n-u}{2}}$ iterates, the database needs $\mathcal{O}(u + (n-u))$ copies of the uniform query state, each of which is built with $\mathcal{O}(n2^u)$ quantum gates by Lemma 7.1. Each of Grover's iterates solves a system of dimension $\mathcal{O}(u + (n - u))$ in $\mathcal{O}(n^3)$ operations, and performs $n$ computations of $\Pi$ in $n^3$ operations. □

An exhaustive comparison with other attacks is given in Table 7.1. Note that while the classical time-data trade-off on Even-Mansour can be made *known-plaintext*, this is not the case here, as we need to query a vector subspace of $\{0,1\}^n$ for the database.

**Table 7.1:** Trade-offs for Q1 attacks on the Even-Mansour construction. In this table we omit $\mathcal{O}$ notations, and ignore polynomial factors in the first and last rows.

| Reference | Classical attack | Grover | [HS18a] | [KM12] | Algorithm 7.2 |
|---|---|---|---|---|---|
| Trade-off of $D$ and $T$ | $D \cdot T = 2^n$ | $T = 2^{n/2}$,<br>$D =$ cons | $D \cdot T^6 = 2^{3n}$<br>$(D \leqslant 2^{3n/7})$ | $D = 2^{n/3}$,<br>$T = 2^{n/3}$ | $D \cdot T^2 = 2^n$ |
| QRACM | - | - | - | $2^{n/3}$ | - |
| Qubits | - | poly$(n)$ | poly$(n)$ | poly$(n)$ | poly$(n)$ |
| Classical memory | $D$ | poly$(n)$ | $D^{1/3}$ | poly$(n)$ | poly$(n)$ |
| Balanced point of $D$ and $T$ | $2^{n/2}$ | - | $2^{3n/7}$ | - | $2^{n/3}$ |

### 7.2.3 Trade-offs for the FX Construction

A similar situation occurs for the FX construction $\mathrm{FX}_{k,k_{in},k_{out}}$, where we just need to balance differently the database setup and the Grover search. We divide the $n$-bit key $k_{in}$ in $k_{in}^{(1)}$ of $u$ bits and $k_{in}^{(2)}$ of $(n-u)$ bits. We apply Simon's algorithm to recover $k_{in}^{(1)}$ while we perform Grover search on $k$ in addition to $k_{in}^{(2)}$ (see Figure 7.2).

If we assume that the internal cipher $E$ can be evaluated in quantum time $\mathcal{O}(n^2)$, then similarly to Proposition 7.2, we obtain:

**Proposition 7.3.** *There exists a quantum algorithm that returns $k$ and $k_{in}$ in quantum time:*

$$\underbrace{\mathcal{O}((n+m)n2^u)}_{Database} + \mathcal{O}\Bigg( \underbrace{2^{\frac{n+m-u}{2}}}_{Grover\ iterates} \times \underbrace{(n+m)^3}_{Simon's\ circuit} \Bigg) ,$$

*classical memory* poly$(n)$*, quantum memory $\mathcal{O}((n+m)n)$ (qubits of the database) and using $\mathcal{O}(2^u)$ classical chosen-plaintext queries to* FX.

A comparison with previous attacks is given in Table 7.2.

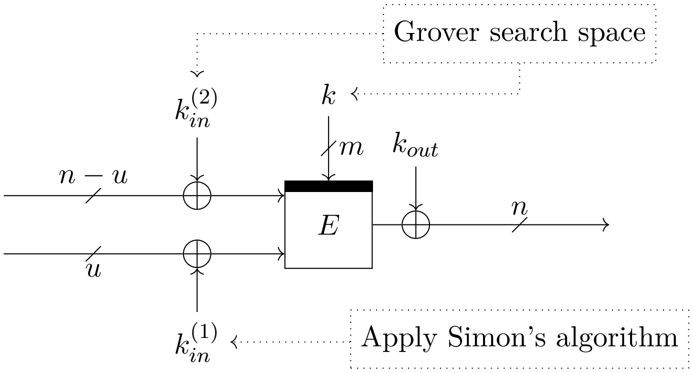


**Figure 7.2:** Idea of our Q1 attack on the FX construction.

**Table 7.2:** Trade-offs for Q1 attacks on the FX construction. We omit $\mathcal{O}$ notations, and ignore polynomial factors in the first and last rows.

| Reference | Classical | Grover | [HS18a] | Proposition 7.3 |
|---|---|---|---|---|
| Trade-off of $D$ and $T$ | $DT = 2^{n+m}$ $(D \leqslant 2^n)$ | $T = 2^{(n+m)/2}$ $D = \text{cons}$ | $D \cdot T^6 = 2^{3(n+m)}$ $(D \leqslant \min\{2^n, 2^{3n/7}\})$ | $D \cdot T^2 = 2^{n+m}$ $(D \leqslant 2^n)$ |
| QRACM | - | - | - | - |
| Qubits | - | $\mathsf{poly}(n)$ | $\mathsf{poly}(n)$ | $\mathsf{poly}(n)$ |
| Classical memory | $D$ | $\mathsf{poly}(n)$ | $D^{1/3}$ | $\mathsf{poly}(n)$ |
| Balanced point of $D$ and $T$ | $2^{(n+m)/2}$ $(m \leqslant n)$ | - | $2^{3(n+m)/7}$ $(m \leqslant 4n/3)$ | $2^{(n+m)/3}$ $(m \leqslant 2n)$ |

### 7.2.4 Related-key Attacks

We will now show that the offline Simon's algorithm provides an efficient generic upper bound for related-key attacks against block ciphers in the Q1 scenario.

Let $E_k$ be a block cipher with a key and block size of $n$ bits. In the related-key setting, as introduced in [WH87], we are allowed to query $E_{k\oplus\ell}(m)$ for any $n$-bit difference $\ell$ and message $m$. Classically, this is a very powerful model, but it becomes especially meaningful when the block cipher is used inside a mode of operation (e.g., a hash function) in which key differences can be controlled by the attacker. It is shown in [WH87] that a secret key recovery in this model can be performed in $2^{n/2}$ operations, as it amounts to find a collision between some query $E_{k\oplus\ell}(m)$ and some offline computation $E_{\ell'}(m)$.

In the Q2 setting, we have mentioned in Chapter 3 the attack of Rötteler and Steinwandt [RS15], which recovers $k$ with Simon's algorithm if quantum query access to $\ell \mapsto E_{k\oplus\ell}(m)$ is allowed.

The offline Simon's algorithm translates this related-key superposition attack into an attack where the related-key oracle is queried only classically, making it more practical. We write $k = k_1||k_2$ where $k_1$ has $n/3$ bits and $k_2$ has $2n/3$ bits. We can query $E_{(k_1||k_2)\oplus(\ell_1||0)}(m)$ for a fixed $m$ and all $n/3$-bit differences $\ell_1$. Then we can perform a Grover search on $k_2$. The classical security level in presence of a related-key oracle of this form, which is $2^{n/2}$, is reduced quantumly to $2^{n/3}$. This shows that the transition to a quantum setting has an impact on the related-key security even if the oracle remains classical.

### 7.2.5 Attacking Concrete Instances in Q1

We give below a non-exhaustive list of designs that can be targeted by Algorithm 7.1, providing better attacks than the previous ones (that would either use QRACM or have a worse time complexity).

**Even-Mansour instances.** Although the Even-Mansour construction is very common, the offline Simon's algorithm is not the best attack in all instances: if the masks are derived from a smaller key, a direct key-recovery using Grover's algorithm may be more efficient. This is for example the case in the CAESAR candidate Minalpher [Sas+15]. In general, we need to have a secret key of at least two thirds of the state size for our attack to beat the exhaustive search.

The Farfalle construction [Ber+17c] degenerates to an Even-Mansour construction if the input message is only 1 block long. Instances of this construction use variable states and key sizes. Xoofff [Dae+18] has a state size of 384 bits and a key size between 192 and 384 bits. Using $2^{128}$ data (which is exactly the data limit of Xoofff) and time, our attack can recover the key. Hence, it is relevant if the key size is greater than 256. Note that the authors made a security claim in the Q1 setting that this attack does not contradict (nor any attack using QRACM).

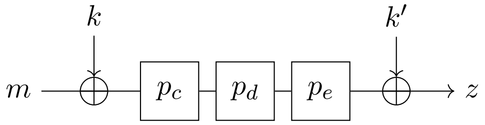


**Figure 7.3:** One-block Farfalle.

**FX instances.** DESX [KR96] has a 64-bit state, two 64-bit whitening keys and one 56-bit inner key. We can run an offline Simon's attack with roughly $2^{42}$ classical queries and $2^{40}$ quantum computations of the cipher circuit.

PRINCE [Bor+12], and PRIDE [Alb+14] use the FX construction with a 64-bit state, a 64-bit inner key and two 64-bit whitening keys. Hence, from Proposition 7.3, we can estimate roughly $2^{45}$ classical queries and $2^{43}$ quantum computations of the cipher circuit.

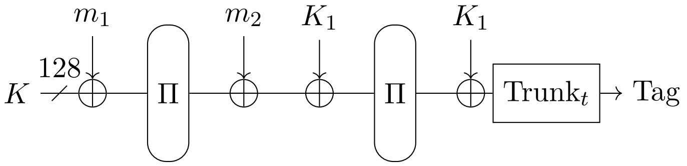


**Figure 7.4:** Two-block Chaskey example.

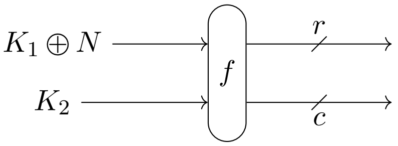


**Figure 7.5:** Beetle state initialization.

**Chaskey.** The lightweight MAC Chaskey [Mou+14] (Figure 7.4) is very close to an Even-Mansour construction operating on 128 bits. Since the last message block ($m_2$ in Figure 7.4) is XORed to the key $K_1$, we can immediately apply our Q1 attack and recover $K_1$ and the value of the state before the XORing of the last message block. As $\Pi$ is a permutation easy to invert, this allows to retrieve $K$. Chaskey contains 16 rounds with 4 modular additions on 32 bits, 4 XORs on 32 bits and some rotations. With a data limit of $2^{48}$, as advocated in the specification, our attack would require roughly $2^{(128-48)/2} \times 2^{19} = 2^{59}$ quantum gates, where the dominant cost is solving the 80-dimensional linear system inside each iteration of Grover's algorithm.

**Sponges.** The offline Simon's algorithm can be used to recover an internal state if this state has a fixed value, and is XORed to a controlled input. A simple duplexed Sponge does not seem to have such vulnerability, but there are some cases in which this algorithm applies effectively.

The Beetle mode of lightweight authenticated encryption [Cha+18], has an initialization phase described as $(K_1 \oplus N) \| K_2 \mapsto f((K_1 \oplus N) \| K_2)$, where $K_1, N \in \{0,1\}^r$, $K_2 \in \{0,1\}^c$, and $f$ is a $(r+c)$-bit permutation. Here, we will use the *nonce* as a varying value, and encrypt the same message under a sequence of nonces forming an affine subspace. Nonce-respecting adversaries are often assumed to freely choose the nonces that they use, but our assumption is weaker. In practice, nonces are often implemented using internal counters, so even if the adversary has no control over the nonce, the attack may still be possible.

In Beetle[Light+], the rate is $r = 64$ bits and the capacity $c = 80$ bits. The rate is sufficiently large to embed 48 varying bits for the nonce; in that case, by making $2^{48}$ classical queries and $2^{48}$ Grover iterations, we can recover the secret $K_1 || K_2$. In Beetle[Secure+], $r = c = 128$ bits. We can recover $K_1 || K_2$ with $2^{85}$ messages and Grover iterations. As the key has the same length as the state, the attack would still work if the nonce was added after the first permutation.

### 7.2.6 Discussion

The algorithm that we presented in this chapter is the first use of Simon's algorithm in a Q1 setting. It shows that a strong algebraic structure, besides paving the way for powerful Q2 attacks, can also make a difference in the Q1 setting.

The results that we obtained are three-fold: • we improved the query complexity of many Q2 attacks from exponential to polynomial; • we increased the applicability of many Q1 attacks by removing their QRACM usage, and in doing so, • we could improve the memory complexity in addition to the time complexity.

As an example, let us consider the attack on Even-Mansour. At any point of the trade-off curve $T^2 \cdot D = 2^n$, the quantum algorithm does not require more than $\mathsf{poly}(n)$ memory. Let us consider the scenario where $D = 2^{n/3}$ is the data limitation (which is coherent with the example of Chaskey). Then at this point, the algorithm provides a quadratic time speedup with respect to the classical attack, up to a small polynomial factor.

Exhaustive key search also has a quadratic speedup. However, it competes against a *very efficient* classical algorithm, which is easily distributed. Meanwhile, the classical attack on Even-Mansour runs in time $2^{2n/3}$ under a data limitation $2^{n/3}$, *and* requires a memory of size $2^{n/3}$. Such an algorithm cannot be parallelized without accounting for massive hardware, memory accesses, and so on [Wie04]. This makes our attack relatively more appealing.

The main open question is whether an *offline* variant, using classical queries only, exist for more Q2 attacks. As an example, quantum slide attacks need to look for a shift between two functions containing the secret key. Thus, the asymmetry of Problem 7.1 is lost. Only some of the attacks given in [BNS19a] can benefit from the offline Simon's algorithm. These are typically the attacks that, similarly to the Grover-meets-Simon attack, need to guess some subkey and run Simon's algorithm for each guess.

# Chapter 8
# Cryptanalysis Based on Symmetries

In this chapter, we will present cryptanalysis results on two of the 32 second-round candidates of the NIST lightweight cryptography project [NIS18]: Spook [Bel+19] and Gimli [Ber+19]. These results were respectively published in [Der+20] and [Fló+20]. Most of them are classical, although we will also obtain dedicated quantum cryptanalysis results on Gimli. But quantum cryptanalysis is a superset of classical cryptanalysis, and as we have remarked before, the latter retains all its importance.

These results have in common that they draw on the power of *internal symmetries*. In both cases, the baseline primitive that we will analyze is a permutation with a quite large internal state, from 384 up to 512 bits. Notably, in the case of Spook, we will exploit the cancellation of round constants within the round function of the permutation. In the case of Gimli, we exploit the slowness of the diffusion. In both cases, we are able to create symmetric internal states and propagate this property through many rounds.

Both candidates are based on permutations and define an authenticated cipher (and a hash function for Gimli) using Sponges and duplexing. We will show that the above properties enable distinguishers on the full permutations. These distinguishers have low complexities and admit practical variants. In the case of Spook, the same properties intervene in a reduced-round forgery attack, and in the case of Gimli, in reduced-round collision attacks.

## Contents

## 8.1 Cryptanalysis of Spook

Spook [Bel+19] is a second-round candidate of the NIST LWC process. It is based on the Sponge One-Pass mode of operation (S1P) [Guo+20], which allows to obtain security guarantees even when nonces are misused and all intermediate computations are leaked to the adversary (except the secret key). The S1P mode is a duplex-like Authenticated Encryption with Associated Data (AEAD) based on a permutation, that requires in addition to call a tweakable block cipher during its initialization and finalization phases. Thus, the authors define the tweakable block cipher Clyde-128 and the permutation Shadow, our main target. Its two variants, Shadow-384 and Shadow-512, use the same subcomponents as Clyde.

Since the submission aims at lightness and side-channel resistance, it follows the paradigm of *LS-designs*, which was introduced with the block ciphers Robin and Fantomas [Gro+14]. In such a block cipher, the state is represented as a matrix of $s \times \ell$ bits; a linear L-Box $L$ is defined, alongside a nonlinear S-Box $S$. Each round has a substitution layer and a linear layer: the substitution layer simply applies $S$ to each column of the matrix, while the linear layer applies $L$ to each row, then adds the round key (i.e., the master key K and a round constant). This is represented in Figure 8.1.

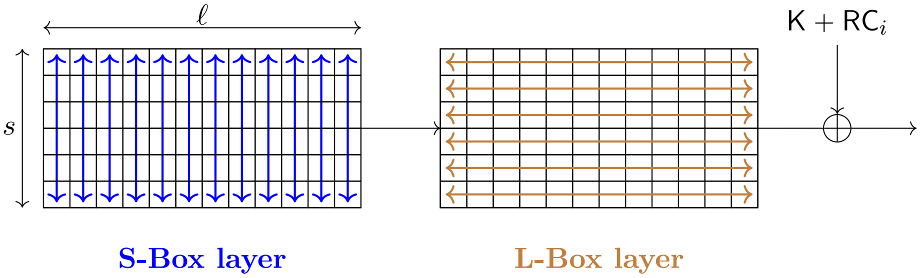


**Figure 8.1:** Round $i$ of a generic LS-design.

Our main target Shadow can be seen as a keyless LS-design modified to accommodate a large state size. Its specification will be given in Section 8.1.1. We do not target the block cipher Clyde-128 and we will consider it as a black-box.

**Spook and Spook V2.** In the rest of this chapter, “Spook” refers to the version of Spook submitted for the second round of the LWC process [Bel+19]. After private communication of the results presented here, the designers of Spook decided to tweak their design. This is discussed in [Bel+20], which defines “Spook V2” as opposed to the initial submission. The attacks of this section are not applicable to Spook V2.

**Summary of the results.** The S1P mode of operation is proven secure if used with a random permutation. However, the designers of Spook do not rely explicitly on the hermetic sponge strategy, and they allow the existence of distinguishers, if they have no incidence on the cipher.

In [Bel+19], Section 4.3, they mention a "minimum [of] 128-bit security against linear cryptanalysis", and "an algebraic degree 128", both of which are achieved by Shadow-512. However, in order to avoid forgeries, it is also necessary to "ensure collision resistance for the 255 bits that are used to generate the tag". In terms of differential cryptanalysis, this means that "truncated differentials with probability larger than $2^{-128}$ for those 255 bits" are avoided. The authors declare:

> *A conservative heuristic for this purpose is to require that no differential characteristic has probability better than $2^{-385}$, which happens after twelve rounds (six steps).*

In Section 8.1.5, we exhibit a practical truncated differential distinguisher on the full version of Shadow-512 (6 steps), and on a shifted version of Shadow-384 (6 steps, or 5 steps of the actual permutation), thus disproving this heuristic. The distinguisher against Shadow-512 would even work against an extended version with 7 steps.

In Section 5 of the specification document, the authors encourage the cryptanalysis of reduced-round version of Spook, and state:

> *The recommended parameters are 12 rounds for Clyde-128 and 12 rounds for Shadow-512. We additionally provide aggressive parameters, with 12 rounds for Clyde-128 and 8 rounds for Shadow-512, as an interesting target for cryptanalysis.*

In Section 8.1.7, we show that our observations on Shadow allow to mount a practical existential forgery attack on the S1P mode, when Shadow-512 is reduced to 8 rounds (4 steps) out of 12, thus breaking the version with "aggressive" parameters proposed by the authors. In this attack, we stand in the nonce-misuse scenario, which is allowed by the CIML2 security game considered by the authors of Spook.

### 8.1.1 Specification of Shadow and Notations

The Spook algorithm uses two variants of the permutation Shadow, of state sizes 512 and 384 respectively, but Shadow-512 is the one used in the primary candidate of the NIST LWC process. The permutation uses a *multiple LS-design* (mLS), which is a variant of an LS-design represented in Figure 8.2.

**Internal state.** The internal state associates $m$ *bundles* of dimensions $s \times \ell$, where $s = 4$, $\ell = 32$, $m = 3$ in Shadow-384 and 4 in Shadow-512. (Thus, the bundles are always of 128 bits.) As in Figure 8.2 a *round* of Shadow applies the 4-bit S-Box on the columns of each bundle (S-Box layer), and then one of two linear functions: one that mixes inside each bundle with the L-Box (round A), and one that mixes the bundles together with

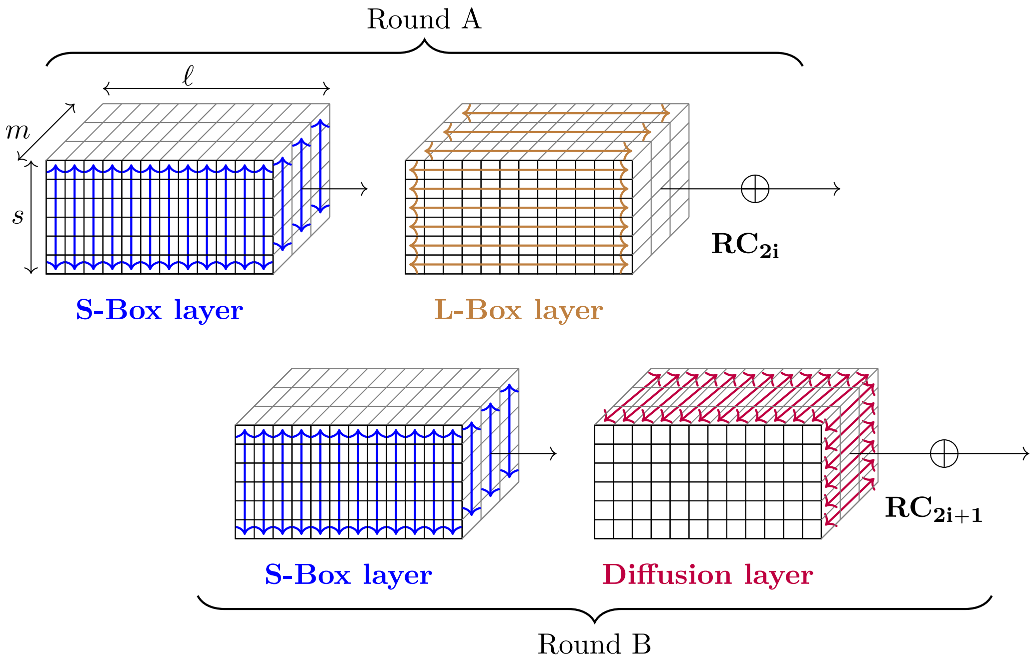


**Figure 8.2:** Step $i$ of a generic mLS-design, with two rounds. Note that in Shadow, the L-Box is applied to two rows instead of a single one.

a diffusion layer (round B). Two such rounds form a *step*. The full versions of both variants iterate 6 steps, thus 12 rounds. As they use similar bundles, only the D layer and the round constants are different. We will be interested in the values of individual columns rather than rows of a bundle. For a bundle $y$, we let $y^{31}, \dots, y^{0}$ denote its columns. We represent bundles and states with column 31 on the left, and column 0 on the right.

**S-Box layer.** The 4-bit S-Box used in Shadow will be denoted $S$, and its lookup table is given in Table 8.1. By abuse of notation, we also denote by $S$ the S-Box layer applied to a bundle, or to a complete Shadow state. The L-Box is denoted $\mathsf{L}'$, and it applies to *two rows* of a bundle instead of a single one. It is defined by:

$$\mathsf{L}'(x, y) = \begin{pmatrix} \mathrm{circ}(\texttt{0xec045008}) \cdot x^T \oplus \mathrm{circ}(\texttt{0x36000f60}) \cdot y^T \\ \mathrm{circ}(\texttt{0x1b0007b0}) \cdot x^T \oplus \mathrm{circ}(\texttt{0xec045008}) \cdot y^T \end{pmatrix} , \tag{8.1}$$

where circ$(A)$ stands for a circulant matrix whose first line is a row vector given by the binary decomposition of $A$. We let $L$ denote the parallel application of the L-Box.

**Diffusion layer.** The diffusion layer applies a transformation D on the 4 (respectively 3) bits having the same positions in each bundle. For Shadow-512, D is an involution given by a near-MDS matrix which appeared previously in the design of the block ciphers

**Table 8.1:** Lookup table of the S-Box used in Shadow.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S(x)$ | 0 | 8 | 1 | f | 2 | a | 7 | 9 | 4 | d | 5 | 6 | e | 3 | b | c |

Midori [Ban+15] and Mantis [Bei+16]:

$$\mathsf{D}(a,b,c,d) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \quad . \tag{8.2}$$

It can be observed that this matrix has branching number 4. That is, the minimal number of non-zero values in $\mathsf{D}(a,b,c,d)$ and $(a,b,c,d)$ is 4, while an MDS matrix would have a branching number 5.

For Shadow-384, a different $\mathsf{D}$ needs to be defined, as there are only 3 bundles to mix:

$$\mathsf{D}(a,b,c) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad . \tag{8.3}$$

**Round constants.** The rounds constants used, both in rounds of type A and of type B, are given in Table 8.2. These constants are of 4 bits only, and correspond to the internal state of a 4-bit LFSR. If bundles are numbered from 0 to 3 (or 2 for Shadow-384), then the constant addition is defined as follows: for $b = 0, 1, 2, 3$, the 4-bit constant given in Table 8.2 is XORed to column number $b$ in bundle $b$. When considering a single step, we will often name $c$ the 4-bit round constant of Round A and $c'$ the constant of Round B.

**Table 8.2:** Round constants $\mathsf{RC}_i$ used in Shadow. Note that the LSB is on the left.

| Round | Constant | Round | Constant | Round | Constant | Round | Constant |
|---|---|---|---|---|---|---|---|
| 0 | (1,0,0,0) | 1 | (0,1,0,0) | 2 | (0,0,1,0) | 3 | (0,0,0,1) |
| 4 | (1,1,0,0) | 5 | (0,1,1,0) | 6 | (0,0,1,1) | 7 | (1,1,0,1) |
| 8 | (1,0,1,0) | 9 | (0,1,0,1) | 10 | (1,1,1,0) | 11 | (0,1,1,1) |

**Super S-Box.** It is possible to rewrite Shadow as an SPN operating on $m$ 128-bit *Super S-Boxes*, which are the bundles. The transformation $\mathsf{D}$ becomes the linear layer of this SPN, and the Super S-Box is the composition of the S-Box layer of round A, the L-Box layer of round A, the constant addition of round A, and the S-Box layer of round B (see Algorithm 8.1).

We let $\sigma_j$ for $j \in \{0, \ldots, m-1\}$ denote the parallel Super S-Boxes applying to bundle $j$. Note that, since the round constant $\mathsf{RC}_{2i}$ is XORed in different positions, $\sigma_j$ are different functions. However, we will observe in the next section that despite this difference, a natural symmetry property can be transmitted through some Shadow steps.

**Algorithm 8.1** The Shadow permutation.

```
Input: $m$ bundles $x_0, \ldots, x_{m-1}$ of dimensions $s \times \ell$
for $i$ from 0 to 5 do                                              ▻ 6 steps
    Apply $S$ column-wise to the bundles                  ⎫
    Apply $\mathsf{L}'$ to each pair of lines in the bundles  ⎬ Super S-Box
    XOR $\mathsf{RC}_{2i}$ to column $b$ in bundle $b$          ⎪
    Apply $S$ column-wise to the bundles                  ⎭
    Apply $\mathsf{D}$ to the bundles: $x_0, \ldots, x_{m-1} \leftarrow \mathsf{D}(x_0, \ldots, x_{m-1})$
    XOR $\mathsf{RC}_{2i+1}$ to column $b$ in bundle $b$
```

## 8.1.2 Defining i-Identical States in Shadow-512

Although our distinguishers on Shadow variants are limited-birthday distinguishers, they do not rely on finding a differential trail with high probability, which would likely contradict the security analysis of the authors. Instead, we will define internal states with strong symmetry properties, namely equal bundles, and show that despite the addition of round constants, on certain conditions, this symmetry can be preserved.

**Definition 8.1** ($i$-identical state)**.** We call *$i$-identical* an internal state of Shadow in which $i$ bundles are equal.

The propagation of $i$-identical states is a strong property that stems from the interaction of round constants in Shadow. We illustrate this with the evolution of a 4-identical state through a Shadow-512 step, in Figure 8.3.

**4-Identical states.** Let $X = (x, x, x, x)$ be a 4-identical state in input to a Shadow step. We let $c$ and $c'$ denote the two 4-bit constants added in round A and B respectively (their values depend on the round index). After the first S-Box layer and the L-Box layer, this state is still 4-identical, thus we omit these operations, and denote by $Y = (y, y, y, y)$ the state obtained before XORing the first constant $c = \mathsf{RC}_{2i}$. After applying the constant addition and the S-Box layer, only the first four columns of the bundles differ:

$$\begin{array}{lcccccc}
\text{Bundle } 0: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2) & S(y^1) & S(y^0 \oplus c) \\
\text{Bundle } 1: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2) & S(y^1 \oplus c) & S(y^0) \\
\text{Bundle } 2: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2 \oplus c) & S(y^1) & S(y^0) \\
\text{Bundle } 3: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2) & S(y^1) & S(y^0)
\end{array}$$

Next, we apply the transformation D. Column $i$ of bundle $b$ is mapped to the sum of columns $i$ of the three other bundles. Thus, the columns 31 downto 4 remain unchanged, and the four last columns are modified as follows:

$$\begin{array}{lccccccc}
\text{Bundle } 0: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0) \\
\text{Bundle } 1: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2 \oplus c) & S(y^1) & S(y^0 \oplus c) \\
\text{Bundle } 2: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2) & S(y^1 \oplus c) & S(y^0 \oplus c) \\
\text{Bundle } 3: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0 \oplus c)
\end{array}$$

Finally, $c'$ is added, exactly in the columns that were not affected by $c$:

$$\begin{array}{lccccccc}
\text{Bundle } 0: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0) \oplus c' \\
\text{Bundle } 1: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2 \oplus c) & S(y^1) \oplus c' & S(y^0 \oplus c) \\
\text{Bundle } 2: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2) \oplus c' & S(y^1 \oplus c) & S(y^0 \oplus c) \\
\text{Bundle } 3: & S(y^{31}) & \cdots & S(y^4) & S(y^3) \oplus c' & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0 \oplus c)
\end{array}$$

From these formulas, Proposition 8.1 follows immediately (see also Figure 8.3).

**Proposition 8.1.** *Let $Y = (y, y, y, y)$ be the value of a 4-identical state before the first constant addition in a Shadow-512 step. Let $c, c'$ be the constants. If the following holds:*

$$\begin{cases} S(y^3 \oplus c) = & S(y^3) \oplus c', \quad S(y^1 \oplus c) = S(y^1) \oplus c' \\ S(y^2 \oplus c) = & S(y^2) \oplus c', \quad S(y^0 \oplus c) = S(y^0) \oplus c' \ . \end{cases} ,$$

*then the state after this step remains 4-identical.*

Recall that $c$ and $c'$ vary with the step index. Depending on their values, the system of equations of Proposition 8.1 will either be satisfied with a rather high probability, or not at all. More precisely, the existence of solutions $(y^0, y^1, y^2, y^3)$ for a given pair $c, c'$, and the probability of success, depends on the coefficients of the differential distribution table of the S-Box $S$. We can easily compute them and verify the results experimentally. They are given in Table 8.3.

**3- and 2-identical states.** Similar results apply if we want only three or two bundles to be identical, as long as the indices of the equal bundles are the same in input and output. In that case, respectively three and two S-Box equations have to be satisfied, and the relations are the same as in Proposition 8.1. Thus, the probability to propagate an $i$-identical state is either nonzero for all $i = 2, 3, 4$, either zero for all. These values are given in Table 8.3.

As an example, consider a 3-identical state $X = (x, x, x, z)$ and its value $Y = (y, y, y, w)$ after the first S-Box and L-Box layers. We can still focus on the first four columns. Due to the choice of D, a quadruple of columns $S(y^i)$, $S(y^i)$, $S(y^i)$, $S(w^i)$ is mapped to $S(w^i)$, $S(w^i)$, $S(w^i)$, $S(y^i)$ which remains 3-identical in the same positions. The first four columns become:

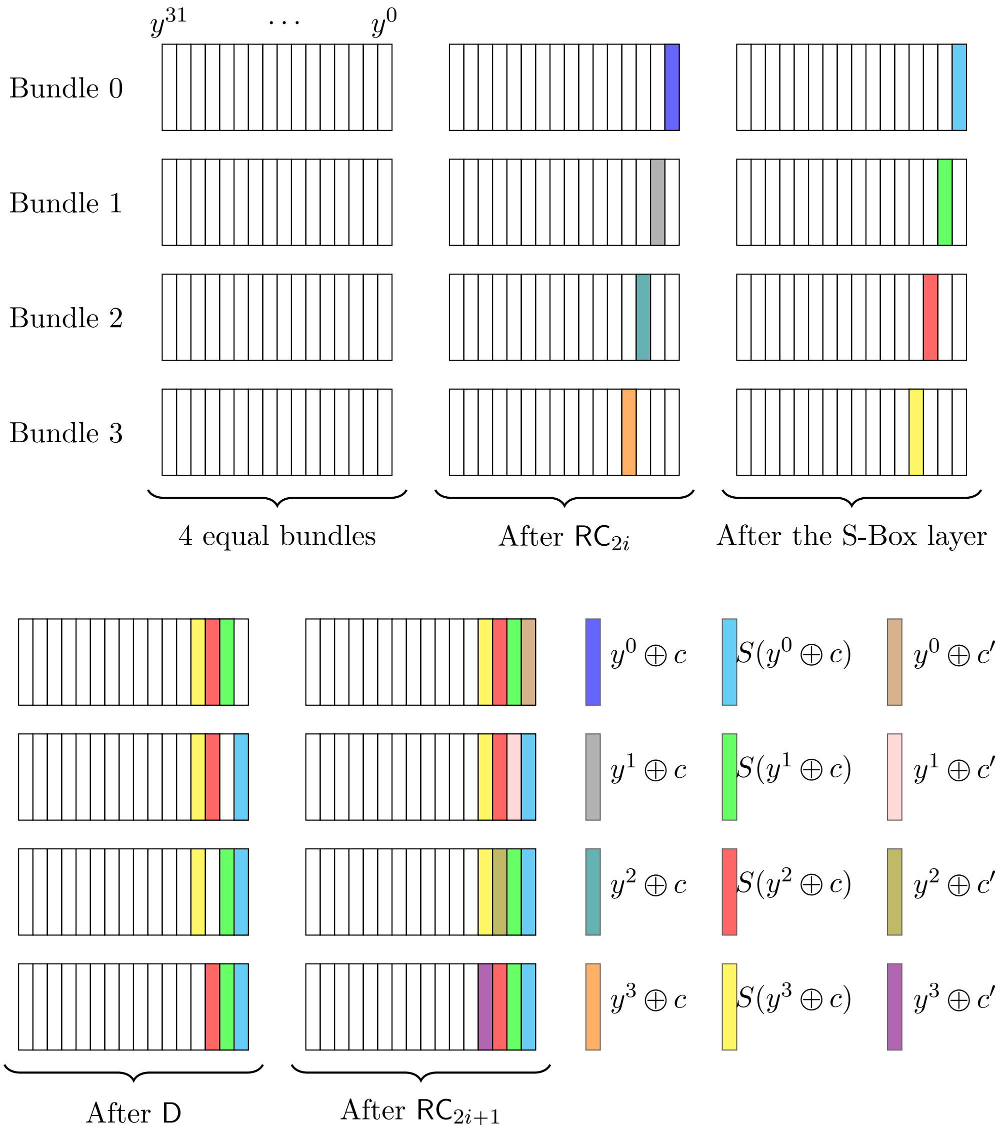


**Figure 8.3:** Propagation of a 4-identical state through a Shadow-512 step (illustration of Proposition 8.1). We represent only partially the bundles, in order to focus on equal columns. Two columns with the same color (except white) contain equal values.

$$\begin{array}{l} \text{Bundle 0:}\\ \text{Bundle 1:}\\ \text{Bundle 2:}\\ \text{Bundle 3:} \end{array} \underbrace{\begin{array}{c} S(w^3 \oplus c)\\ S(w^3 \oplus c)\\ S(w^3 \oplus c)\\ S(y^3) \oplus c \end{array}}_{\text{Column 3}}, \underbrace{\begin{array}{c} S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2)\\ S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2)\\ S(w^2) \oplus c',\\ S(y^2 \oplus c) \end{array}}_{\text{Column 2}},$$

$$\underbrace{\begin{array}{c} S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1)\\ S(w^1) \oplus c'\\ S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1)\\ S(y^1 \oplus c) \end{array}}_{\text{Column 1}}, \underbrace{\begin{array}{c} S(w^0) \oplus c'\\ S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0)\\ S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0)\\ S(y^0 \oplus c) \end{array}}_{\text{Column 0}}$$

Thus, the state remains 3-identical if and only if the following holds:

$$\begin{cases} S(y^2 \oplus c) = & S(y^2) \oplus c'\\ S(y^1 \oplus c) = & S(y^1) \oplus c'\\ S(y^0 \oplus c) = & S(y^0) \oplus c' \end{cases}, \tag{8.4}$$

which does not depend on the value of $w$, but only on the value $y$ taken by the three identical bundles.

**Table 8.3:** Probability that an i-identical state maps to an i-identical state through step $j$ of Shadow-512.

| Step $j$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 4-identical | 0 | 0 | 0 | $2^{-12}$ | $2^{-8}$ | 0 |
| 3-identical | 0 | 0 | 0 | $2^{-9}$ | $2^{-6}$ | 0 |
| 2-identical | 0 | 0 | 0 | $2^{-6}$ | $2^{-4}$ | 0 |

### 8.1.3 2-identical States in Shadow-384

The D-layer in Shadow-512 is particularly favorable to the propagation of identical states. This situation changes in Shadow-384. However, we are able to propagate 2-identical states where bundles 1 and 2 are equal. Similar equations hold, so the probabilities of remaining 2-identical with this form are the same as in Table 8.3.

Let $X = (x, z, z)$ be a 2-identical state and $Y = (y, w, w)$ be the state after the S-Box and L-Box layers. After the addition of the round constant $c$ and the second S-Box layer, we obtain the following:

$$\begin{array}{lcccccc} \text{Bundle 0}: & S(y^{31}) & \cdots & S(y^3) & S(y^2) & S(y^1) & S(y^0 \oplus c)\\ \text{Bundle 1}: & S(w^{31}) & \cdots & S(w^3) & S(w^2) & S(w^1 \oplus c) & S(w^0)\\ \text{Bundle 2}: & S(w^{31}) & \cdots & S(w^3) & S(w^2 \oplus c) & S(w^1) & S(w^0) \end{array}$$

Recall that in Shadow-384, D maps $(a, b, c)$ to $(a \oplus b \oplus c, a \oplus c, a \oplus b)$. After applying this mapping and the second constant addition, we obtain:

$$\begin{array}{lccc}
\text{Bundle 0:} & S(y^{31}) & \dots & S(y^3) \\
\text{Bundle 1:} & S(y^{31}) \oplus S(w^{31}) & \dots & S(y^3) \oplus S(w^3) \\
\text{Bundle 2:} & \underbrace{S(y^{31}) \oplus S(w^{31})}_{\text{Column 31}} & \dots & \underbrace{S(y^3) \oplus S(w^3)}_{\text{Column 3}}
\end{array},$$

$$\underbrace{\begin{array}{c} S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2) \\ S(y^2) \oplus S(w^2 \oplus c) \\ S(w^2) \oplus S(y^2) \oplus c' \end{array}}_{\text{Column 2}} , \underbrace{\begin{array}{c} S(y^1) \oplus S(w^1 \oplus c) \oplus S(w^1) \\ S(y^1) \oplus S(w^1) \oplus c' \\ S(y^1) \oplus S(w^1 \oplus c) \end{array}}_{\text{Column 1}} , \underbrace{\begin{array}{c} S(y^0 \oplus c) \oplus c' \\ S(y^0 \oplus c) \oplus S(w^0) \\ S(y^0 \oplus c) \oplus S(w^0) \end{array}}_{\text{Column 0}}$$

Thus, the state will remain 2-identical if the two following equations are satisfied:

$$\begin{cases} S(w^2 \oplus c) = & S(w^2) \oplus c' \\ S(w^1 \oplus c) = & S(w^1) \oplus c' \end{cases} . \tag{8.5}$$

### 8.1.4 A Distinguisher Against 5-round Shadow-512

All our attacks against Shadow combine the following properties:

1. For Shadow-512, D is not an MDS matrix;

2. Although Super S-Boxes are different functions, they still behave very closely one to another. Thus, it is easy to find differential pairs for multiple Super S-Boxes with the same differences, and potentially equal input values;

3. $i$-identical states can be mapped to $i$-identical states with high probability through steps 3 and 4.

In this section, we start by combining 1. and 2. to distinguish a reduced-round version of Shadow with 5 steps out of 6. This truncated differential distinguisher will work regardless of the round constants applied, so we may choose either the 5 first or the 5 last rounds. It has complexity 1, which means that it costs less than the equivalent of an evaluation of Shadow. It outputs a pair of states with difference 0 on the last bundle that, after 5 steps, arrive at a difference 0 on the last bundle. Since a bundle is of 128 bits, the complexity to obtain such a pair for a random permutation would be $2^{64}$ by the results of [IPS13]. Thus, this is a valid limited-birthday distinguisher.

The truncated differential path that we use is displayed in Figure 8.4, where an $*$ denotes an unknown difference. We use the Super S-Box notation and denote by $\sigma_i$ the Super S-Box applied to bundle $i$; we emphasize that the $\sigma_i$ are distinct functions, but we omit the fact that they are also distinct between different rounds, for more readability.

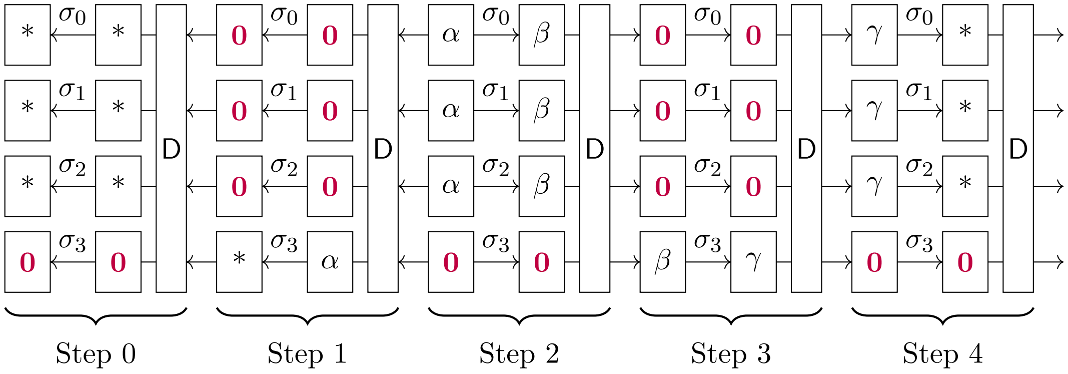


**Figure 8.4:** A 5-step distinguisher against Shadow-512.

**The slow diffusion of the D layer.** The fact that D is not an MDS matrix allows to span more steps. Indeed, if D was MDS, an input (respectively output) with one active Super S-Boxes would map to an output (resp. input) with all active Super S-Boxes, and the path would not span Round 0 and Round 4, as it does in Figure 8.4.

**Simultaneous differentials for the Super S-Boxes.** From the beginning of Step 2 forwards, and from the end of Step 2 backwards, all transitions represented in Figure 8.4 hold with probability one. Our distinguisher starts from the middle: we must find three values for the bundles $0, 1, 2$ that have the same difference in input to their respective Super S-Boxes, and in output.

If the Super S-Boxes were random functions, this would be difficult. However, only the position where the round constant $c$ is XORed differs between them. This makes it easy to build an input difference $\alpha$ that maps *to the same difference* for all three Super S-Boxes $\sigma_0, \sigma_1$ and $\sigma_2$ *simultaneously.*

**Lemma 8.1.** *If $x \in \mathbb{F}_2^{128}$ and $\alpha \in \mathbb{F}_2^{128}$ are such that $(L \circ S)(x) \oplus (L \circ S)(x \oplus \alpha) = \beta$ and if $\beta$ is set to 0 on the 4 columns that can receive the round constant $c$ (columns 0 to 3), then the value of $\sigma_b(x) \oplus \sigma_b(x \oplus \alpha)$ does not depend on the bundle index $b$.*

*Proof.* We let $y$ and $y \oplus \beta$ denote the respective values of $(L \circ S)(x)$ and $(L \circ S)(x \oplus \alpha)$. By expanding these into column notation we get:

$$\begin{array}{rccccccc} y = & y^{31} & \cdots & y^4 & y^3 & y^2 & y^1 & y^0 \\ y \oplus \beta = & y^{31} \oplus \beta^{31} & \cdots & y^4 \oplus \beta^4 & y^3 & y^2 & y^1 & y^0 \end{array}$$

We use the Kronecker symbol notation: $\delta_{ij} = 1$ iff $i = j$ and 0 otherwise. XORing the constant and applying $S$ to this pair of inputs, we obtain respectively for $\sigma_i(x)$ and $\sigma_i(x \oplus \alpha)$, for all $\leqslant i \leqslant 3$:

$$\begin{array}{ccccccc} S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus \delta_{i3}c) & S(y^2 \oplus \delta_{i2}c) & S(y^1 \oplus \delta_{i1}c) & S(y^0 \oplus \delta_{i0}c) \\ S(y^{31} \oplus \beta^{31}) & \cdots & S(y^4 \oplus \beta^4) & S(y^3 \oplus \delta_{i3}c) & S(y^2 \oplus \delta_{i2}c) & S(y^1 \oplus \delta_{i1}c) & S(y^0 \oplus \delta_{i0}c) \end{array}$$

so summing these equations yields

$$\sigma_i(x) \oplus \sigma_i(x \oplus \alpha) = \quad \gamma^{31} \quad \cdots \quad \gamma^{4} \quad 0 \quad 0 \quad 0 \quad 0$$

Thus, by making a right choice of $\alpha$, we can get the same difference in output of the 4 Super S-Boxes, regardless of the input value. □

Our distinguisher is described in Algorithm 8.2. Starting from the middle, it uses Lemma 8.1 and then follows the path depicted in Figure 8.4. Using the property of the Super S-Boxes and of the D layer alone, it does not seem easy to extend this to more steps. For this, we need to use the mapping of $i$-identical states described in Section 8.1.2.

**Algorithm 8.2** A distinguisher for 5-step Shadow-512.

```
Attacks: the permutation $\Pi$ defined as 5-step Shadow
Output: a pair of states $(X_0, X_0')$ such that:
    $\begin{cases} X_0 \oplus X_0' = & (*,*,*,0) \\ \Pi(X_0) \oplus \Pi(X_0') = & \mathsf{D}(*,*,*,0) \end{cases}$
Choose $\beta \in \mathbb{F}_2^{128}$ such that it is set to 0 on columns 0 to 3
Choose a random $y \in \mathbb{F}_2^{128}$ and a random $z \in \mathbb{F}_2^{128}$
Compute $x = \sigma_0^{-1}(y)$ and $x \oplus \alpha = \sigma_0^{-1}(y \oplus \beta)$,
Set the two states at step 2 to be
    $X_2 = (x,\ x,\ x,\ z)$ and $X_2' = (x \oplus \alpha,\ x \oplus \alpha,\ x \oplus \alpha,\ z)$ .
Invert step 1 and step 0 on $X_2$ and $X_2'$ to obtain a pair of states $(X_0, X_0')$
return $(X_0, X_0')$
```

## 8.1.5 A Distinguisher Against Full Shadow-512

In this section, $\Pi$ will now denote an *extended* Shadow-512 permutation, with 7 steps. We exhibit a distinguisher that output pairs $(X_0, X_0')$ of 512-bit states such that:

$$X_0 \oplus X_0' = (*,*,*,0) \text{ and } \Pi(X_0) \oplus \Pi(X_0') = \mathsf{D}(*,*,*,0) \ . \tag{8.6}$$

This is the trivial extension of a distinguisher for 6-step Shadow-512, that outputs $\Pi(X_0) \oplus \Pi(X_0') = (0,0,0,*)$; since $\mathsf{D}(0,0,0,*) = (*,*,*,0)$ the complexity is the same in both cases, regardless of the round constants that might be used in this additional step. In both cases, the limited-birthday problem would have a complexity of around $2^{64}$ queries for a random permutation, and we are able to solve it in practical time, using about $2^{15}$ queries.

The distinguisher combines the 5-step truncated differential path of Section 8.1.4 with the observation on 3-identical states of Section 8.1.2. It is summarized in Figure 8.5.

Step 0 and Step 6 are traversed exactly as in the 5-step truncated differential path of Section 8.1.4, by mapping the difference $(0, 0, 0, *)$ to $(*, *, *, 0)$ through D forwards and backwards.

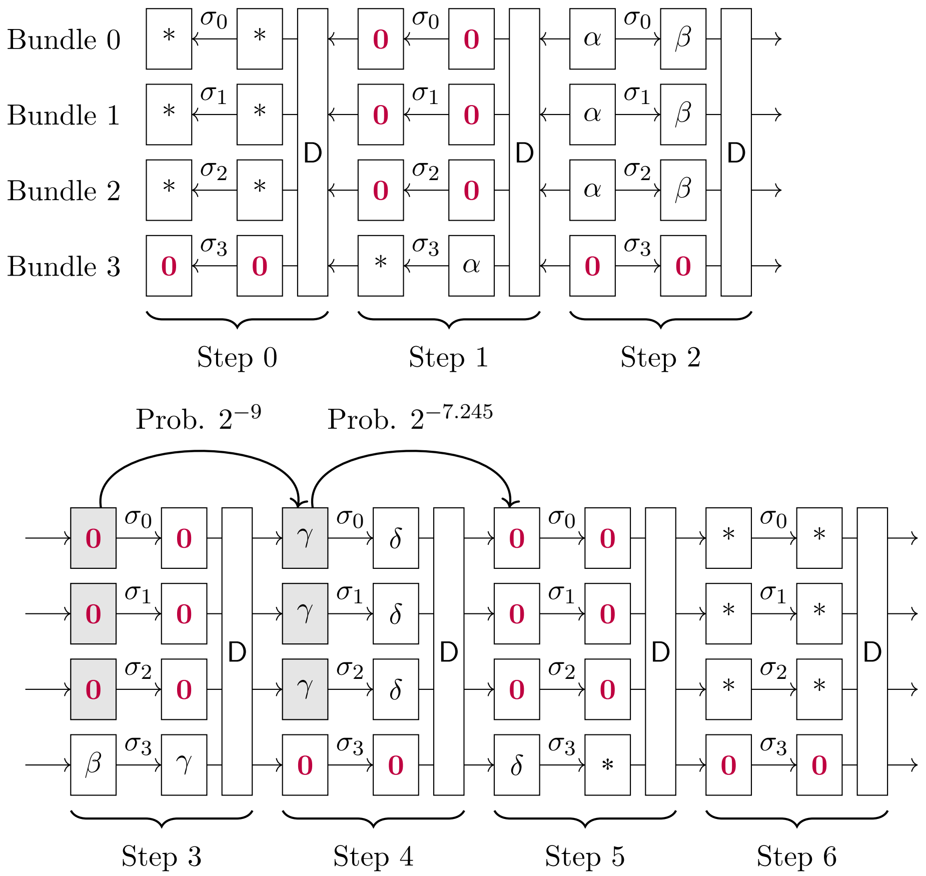


**Figure 8.5:** A 7-step distinguisher against Shadow-512. Gray boxes denote that, in addition to the constrained difference, both states in the pair are 3-identical.

#### 8.1.5.1 Structure of the Distinguisher

As before, we start from the middle, namely from an input at Step 2. This time, we constraint it more in order to have a pair of 3-identical states at the beginning of Step 3, and Step 4. The rest of the path follows the truncated differential path of Section 8.1.4. We let $X_i$ denote a Shadow-512 state (512 bits, 4 bundles) in input of step $i$.

- We start by building $X_2$ and $X_2'$ such that

$$\begin{cases} X_2 \oplus X_2' = & (\alpha, \alpha, \alpha, 0) \\ X_3 \oplus X_3' = & (0, 0, 0, \beta) \\ X_3 \text{ and } X_3' & \text{are 3-identical} \end{cases}$$

- We map both 3-identical states $X_3$ and $X'_3$ to 3-identical states $X_4$ and $X'_4$ at the beginning of Step 4. By the results of Table 8.3, this event occurs with probability $2^{-9}$. The input difference at the end of Step 3 is then equal to $(\gamma, \gamma, \gamma, 0)$.

- Next, we hope that the difference at end of Step 4, before the $\mathsf{D}$ layer, has the form $(\delta, \delta, \delta, 0)$. We will show that this occurs with probability $2^{-7.245}$.

- The rest of the path follows from the mapping of differences through $\mathsf{D}$, and is satisfied with probability 1.

We now detail these different steps.

#### 8.1.5.2 Satisfying the Conditions at Step 2

We reduce the conditions of Step 2 to two properties, (8.7) and (8.8) below, and we show how to satisfy them. More precisely, we need to find 128-bit values $x, \alpha, \beta, \epsilon$ and $\epsilon'$ such that:

$$\begin{cases} \sigma_0(x) + \sigma_0(x+\alpha) & = \beta \\ \sigma_1(x+\epsilon) + \sigma_1(x+\epsilon+\alpha) & = \beta \\ \sigma_2(x+\epsilon') + \sigma_2(x+\epsilon'+\alpha) & = \beta\ , \end{cases} \tag{8.7}$$

where $\sigma_0, \sigma_1, \sigma_2$ are the Super S-Boxes at Step 2. Setting $X_2 = (x, x+\epsilon, x+\epsilon', z)$ and $X'_2 = (x+\alpha, x+\epsilon+\alpha, x+\epsilon'+\alpha, z)$ for some $z$, this first condition ensures that $X_2 \oplus X'_2 = (\alpha, \alpha, \alpha, 0)$ as requested by our differential path. However, we also want $X_3$ and $X'_3$ to be 3-identical, that is:

$$\begin{cases} (\mathsf{RC} \circ \mathsf{D})\big(\sigma_0(x),\ \sigma_1(x+\epsilon),\ \sigma_2(x+\epsilon'),\ z\big) & = (y,\ y,\ y,\ z') \\ (\mathsf{RC} \circ \mathsf{D})\big(\sigma_0(x+\alpha),\ \sigma_1(x+\alpha+\epsilon),\ \sigma_2(x+\alpha+\epsilon'),\ z\big) & = (y,\ y,\ y,\ z'+\beta)\ . \end{cases} \tag{8.8}$$

where $\mathsf{RC}$ is the last constant addition of Step 3.

#### 8.1.5.3 Finding Starting Points at Step 2

In order to satisfy Properties (8.7) and (8.8), we use a specific truncated differential with probability 1.

**Lemma 8.2.** *Let $\alpha \in \mathbb{F}_2^{128}$ be a bundle difference that is non-zero only in columns 22 and 23. Then after any Super S-Box $\sigma$, at any step, and for any input $x \in \mathbb{F}_2^{128}$, we have:*

$$\sigma(x) \oplus \sigma(x \oplus \alpha) = (*, *, ..., *, 0, 0, 0, 0)\ .$$

*where the output difference depends only on $\alpha$ and on columns 31 to 4 of $\sigma(x)$.*

*Proof.* This special difference comes from the definition of the linear layer. Let $e_{22}$ and $e_{23}$ be the vectors of $\mathbb{F}_2^{64}$ (hence rows of a bundle) having respectively a single 1 in position 22 and 23. Using the reference implementation of $\mathsf{L}'$, defined in Equation (8.1), we see that:

$$\begin{aligned}
\mathsf{L}'(0, e_{22}) &= (\texttt{1b880510}, \texttt{6c06f000}) \\
\mathsf{L}'(e_{22}, 0) &= (\texttt{36037800}, \texttt{1b880510}) \\
\mathsf{L}'(0, e_{23}) &= (\texttt{37100a20}, \texttt{d80de000}) \\
\mathsf{L}'(e_{23}, 0) &= (\texttt{6c06f000}, \texttt{37100a20}) \ ,
\end{aligned}$$

that is, a difference in positions 22 or 23 does not influence the first four columns (those on which the constants are added).

Thus, after applying the first S-Box layer and the linear layer:

$$(L \circ S)(x) \oplus (L \circ S)(x \oplus \alpha) = (*, *, ..., *, 0, 0, 0, 0) \ .$$

The constant addition only modifies the four first columns, on which there is no difference. Thus, after the second S-Box layer, the difference between these columns remain the same. Meanwhile, the difference between columns 31 to 4 is independent of the constants, thus of the Super S-Box chosen.

Besides, the difference is a function of columns 31 to 4 of $(L \circ S)(x)$ and $(L \circ S)(x \oplus \alpha)$ (since only another $S$ is applied afterwards). Columns 31 to 4 of $(L \circ S)(x \oplus \alpha)$ are a function of columns 31 to 4 of $(L \circ S)(x)$ and of $\alpha$, since the introduction of $\alpha$ does not modify any of the first 4 columns. By inverting $S$, $(L \circ S)(x)$ is a function of $\sigma(x)$, which completes the result. □

Lemma 8.2 allows to define a vector space $\nabla \subset (\mathbb{F}_{2^4})^{32}$ that contains $2^{16} - 1$ such non-zero differences, by taking any non-zero difference in columns 22 and 23 of the bundle:

$$\nabla = \{a \times e_{22} + b \times e_{23}, a \in \mathbb{F}_2^4, b \in \mathbb{F}_2^4\} \ ,$$

where the multiplications are done in the finite field $\mathbb{F}_2^4$.

In order to satisfy conditions (8.7) and (8.8), we start from a 3-identical state $X_3 = (x_3, x_3, x_3, y_3)$ at the beginning of Step 3. We invert Step 2 on this state, obtaining a state $X_2 = (x_2, y_2, z_2, t_2)$ that is not 3-identical anymore. Then we add a difference $\alpha$ from the subspace $\nabla$ to three bundles of this state, obtaining $X_2' = (x_2 \oplus \alpha, y_2 \oplus \alpha, z_2 \oplus \alpha, t_2)$. We now use the observation of Lemma 8.2 to show that the difference after the Super S-Boxes $\sigma_0, \sigma_1, \sigma_2$ in Step 2 is the same $\beta$. Together with the 3-identicality of $X_3$, this implies that $X_3'$ is 3-identical.

**Lemma 8.3.** *Let $X_3 = (x_3, x_3, x_3, y_3)$ be a 3-identical state, $X_2 = (x_2, y_2, z_2, t_2)$ be the state obtained after inverting Step 2. Let $\alpha$ be a difference in $\nabla$. Then:*

$$\sigma_0(x_2) \oplus \sigma_0(x_2 \oplus \alpha) = \sigma_1(y_2) \oplus \sigma_1(y_2 \oplus \alpha) = \sigma_2(z_2) \oplus \sigma_2(z_2 \oplus \alpha) \ .$$

*Proof.* We invert the $\mathsf{D}$ layer and the constant addition on $X_3$. There exists a quadruple of values $c_0, c_1, c_2, c_3$, that are nonzero only on columns 0 to 3, such that the obtained state has the form: $(y_3 \oplus c_0, y_3 \oplus c_1, y_3 \oplus c_2, x_3 \oplus c_3)$ (these $c_i$ correspond to the inversion of $\mathsf{D}$ on the round constant addition).

With this notation, we have: $x_2 = \sigma_0^{-1}(y_3 \oplus c_0)$, $y_2 = \sigma_1^{-1}(y_3 \oplus c_1)$, $z_2 = \sigma_2^{-1}(y_3 \oplus c_2)$.

Thus, $\sigma_0(x_2)$, $\sigma_1(y_2)$ and $\sigma_2(z_2)$ differ only on columns 3 to 0, and they have the same values on columns 31 to 4. By Lemma 8.2, we find that the difference $\sigma_0(x_2)\oplus\sigma_0(x_2\oplus\alpha)$ is active only on columns 31 to 4, and does only depend on columns 31 to 4 of $\sigma_0(x_2)$. The same holds for $\sigma_1(y_2)$ and $\sigma_2(z_2)$. Thus, the statement follows. □

#### 8.1.5.4 Verifying Steps 3 and 4

The transitions of Step 3 and Step 4 are probabilistic, and we have now to evaluate these probabilities. At the beginning of Step 3, we have two 3-identical states $X_3$ and $X'_3$ and we want to map them to two 3-identical states $X_4$ and $X'_4$. We have seen in Section 8.1.2 that each of these states remains 3-identical with probability $2^{-9}$. However, they have the same values in the identical bundles in input, and if we write down the equations that must be satisfied, we obtain twice the same. Thus, the probability of the transition for both states is $2^{-9}$ instead of its square.

**Going through Step 4.** We aim at a difference $(0,0,0,\delta)$ for a non-zero $\delta$ at the beginning of Step 5 (see Figure 8.5). This means that the difference at the end of Step 4, before the $\mathsf{D}$ layer, must be $(\delta,\delta,\delta,0)$. To estimate the probability of this event, let us write the corresponding equations. We let $(y,y,y,w)$ and $(y',y',y',w)$ denote the two respective states after the application of $S$ and $L$ of Step 4. Since the input of Step 4 is 3-identical, $y^i = y'^i$ for all $i > 3$. We just have to focus on the last 4 columns. After $\mathsf{D}$ and $\mathsf{RC}$, their values are as follows for $(y,y,y,w)$:

$$\begin{array}{l} \text{Bundle 0:}\\ \text{Bundle 1:}\\ \text{Bundle 2:}\\ \text{Bundle 3:}\end{array} \quad \underbrace{\begin{array}{c} S(w^3\oplus c)\\ S(w^3\oplus c)\\ S(w^3\oplus c)\\ S(y^3)\oplus c'\end{array}}_{\text{Column 3}}, \quad \underbrace{\begin{array}{c} S(y^2)\oplus S(y^2\oplus c)\oplus S(w^2)\\ S(y^2)\oplus S(y^2\oplus c)\oplus S(w^2)\\ S(w^2)\oplus c'\\ S(y^2\oplus c)\end{array}}_{\text{Column 2}},$$

$$\underbrace{\begin{array}{c} S(y^1)\oplus S(y^1\oplus c)\oplus S(w^1)\\ S(w^1)\oplus c'\\ S(y^1)\oplus S(y^1\oplus c)\oplus S(w^1)\\ S(y^1\oplus c)\end{array}}_{\text{Column 1}}, \quad \underbrace{\begin{array}{c} S(w^0)\oplus c'\\ S(y^0)\oplus S(y^0\oplus c)\oplus S(w^0)\\ S(y^0)\oplus S(y^0\oplus c)\oplus S(w^0)\\ S(y^0\oplus c)\end{array}}_{\text{Column 0}}$$

and similarly for $(y',y',y',w)$, by replacing $y^i$ by $y'^i$.

Since we want these two states to have a zero difference except in Bundle 3, we need the following relations to be satisfied:

$$\begin{aligned} S(y'^2)\oplus S(y'^2\oplus c) &= S(y^2)\oplus S(y^2\oplus c)\\ S(y'^1)\oplus S(y'^1\oplus c) &= S(y^1)\oplus S(y^1\oplus c)\\ S(y'^0)\oplus S(y'^0\oplus c) &= S(y^0)\oplus S(y^0\oplus c)\ . \end{aligned}$$

Since we are looking at Step 4, the constant $c$ is equal to $0x5$ and then each equality has a probability equal to $2^{-2.415}$ to be verified (assuming that the value of $y$ and $y'$ are independent).

#### 8.1.5.5 Description of the Full Distinguisher

Our distinguisher is detailed in Algorithm 8.3. Since Step 3 is passed with probability $2^{-9}$ and Step 4 with probability $(2^{-2.415})^3$, the path from Step 2 to Step 4 succeeds with probability $2^{-16.245}$. The rest has probability 1.

**Algorithm 8.3** Our 7-step distinguisher against Shadow-512.

```
Attacks: the permutation $\Pi$ defined as 7-step Shadow
Output: a pair of states $(X_0, X'_0)$ such that, with probability at least $2^{-16.245}$:
```

$$\begin{cases} X_0 \oplus X'_0 = & (*, *, *, 0) \\ \Pi(X_0) \oplus \Pi(X'_0) = & \mathsf{D}(*, *, *, 0) \end{cases}$$

```
1: Select a difference $\alpha \in \nabla$.
2: Select a state $X_3 = (y_3, y_3, y_3, z_3)$ at the beginning of Step 3
3: Invert step 2 on $(y_3, y_3, y_3, z_3)$, obtaining $(x_2, y_2, z_2, t_2)$.
4: Invert step 1 on $(x_2, y_2, z_2, t_2)$ and $(x_2 \oplus \alpha, y_2 \oplus \alpha, z_2 \oplus \alpha, t_2)$, obtaining $(x_1, y_1, z_1, t_1)$
   and $(x_1, y_1, z_1, t'_1)$.
5: Invert step 0, obtaining a pair of 512-bit states with a zero-difference in bundle 3
6: return this pair of states. With probability $\geqslant 2^{-16.24}$, it satisfies the truncated trail
   of Figure 8.5.
```

**Experimental results.** Experiments showed that the probability of the distinguisher is slightly higher than what we expected. In fact, the trail of Figure 8.5 is not the only one that leads to the required output difference (see the Appendix of [Der+20]). By running Algorithm 8.3 for $2^{22}$ times, we obtained 124 successful pairs, a probability close to $2^{-15}$. Our unoptimized C++ implementation found all these pairs in less than 30 seconds on a desktop computer.

### 8.1.6 A Distinguisher Against 6-step Shadow-384

In this section, $\Pi$ will denote the "shifted" Shadow-384 permutation, from Step 1 to 6 included. We build a distinguisher that uses essentially the same ideas as the one for full-round Shadow-512. It immediately implies a distinguisher on 5 steps of the original Shadow-384 permutation, where Step 0 has been removed.

- Due to the change of D, we must use another truncated differential path. At the end of Step 1, we have a difference $(0, \alpha, \alpha)$. Since D is now the transformation:

$$\mathsf{D}(a, b, c) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix},$$

it maps a difference $(0, \alpha, \alpha)$ to $(0, \alpha, \alpha)$ backwards, but after the inverse Super S-Box of Step 1, we obtain two unknown differences, whereas we had only one previously. This is why Step 0 cannot be covered anymore.

- We start in the middle, at Step 2, as before. We adapt properties (8.7) and (8.8), and Lemma 8.2, and obtain essentially the same results for a 3-bundle state.
- We propagate 2-identical states instead of 3-identical ones, using the property studied in Section 8.1.3. Before we only needed to do that through a single step; now we do that for one step more. This allows us to reclaim one of the steps that we lost with the change of D.

The path is represented in Figure 8.6 and the procedure is given in Algorithm 8.4.

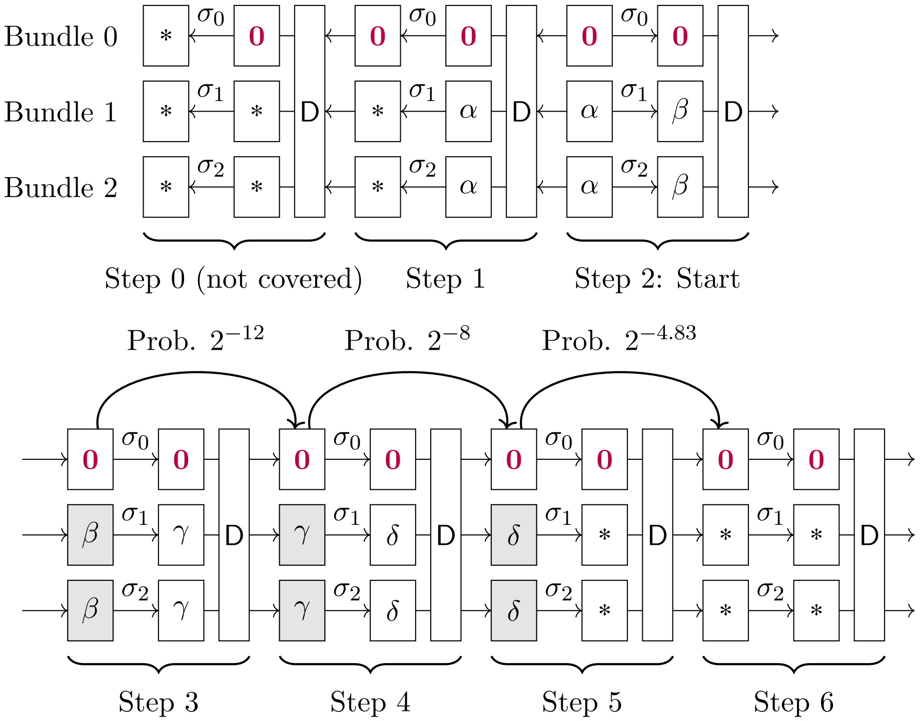


**Figure 8.6:** A (1-step shifted) 6-step distinguisher against Shadow-384. We emphasize that Step 0 is not covered anymore by the truncated differential pattern. Gray boxes denote that, in addition to the constrained difference, both states in the pair are 2-identical.

**Starting in the middle.** Our observation on $\mathsf{L}'$ is independent on the number of bundles involved. We choose $\alpha$ in the vector space $\nabla = \{a \times e_{22} + b \times e_{23}, a \in \mathbb{F}_2^4, b \in \mathbb{F}_2^4\}$. By taking a 2-identical state before Step 3, inverting Step 2, and adding the difference $\alpha$, we obtain a state $X_2'$ that maps to another 2-identical state, with $X_2' \oplus X_2 = (0, \beta, \beta)$.

**Algorithm 8.4** Our 6-step distinguisher against Shadow-384.

```
Attacks: the permutation $\Pi$ defined as 6-step Shadow-384 (from Step 1 to Step 6)
Input: no input
Output: a pair of states $(X_1, X'_1)$ such that, with probability at least $2^{-24.483}$:
    $\begin{cases} X_1 \oplus X'_1 = & (0,*,*) \\ \Pi(X_1) \oplus \Pi(X'_1) = & \mathsf{D}(0,*,*) \end{cases}$
Select a difference $\alpha \in \nabla$.
Select a 2-identical state $X_3 = (z_3, y_3, y_3)$ at the beginning of Step 3
Invert step 2 on $(z_3, y_3, y_3)$, obtaining $(x_2, y_2, z_2)$.
Invert step 1 on $(x_2, y_2, z_2)$ and $(x_2, y_2 \oplus \alpha, z_2 \oplus \alpha)$, obtaining $X_1$ and $X'_1$.
Invert step 0, obtaining a pair of 512-bit states with a zero-difference in bundle 3
return this pair of states. With probability $\geqslant 2^{-24.483}$, it satisfies the truncated trail of Figure 8.6.
```

**Mapping 2-identical states.** Since $X_3$ and $X'_3$ are 2-identical, they can be mapped to 2-identical states through Step 3 and Step 4, with the probabilities given in Section 8.1.3. Notice that the steps that we attack are constrained by this property, which is why it works best for a shifted version. As in the distinguisher on Shadow-512, the two states must remain 2-identical, but the equations that must be satisfied are the same. Thus, the probabilities for Step 3 and 4 are respectively $2^{-12}$ and $2^{-8}$.

**Going through Step 5.** We aim at a difference $(0,*,*)$ at the end of Step 5 (see Figure 8.6). As before, we write the equations that must be satisfied for this event to occur, starting from the fact that the input to Step 5 is 2-identical. We let $(w, z, z)$ and $(w, z', z')$ denote the two states after the application of $S$ and $L$, and $c$ and $c'$ denote the two round constants. We focus on the last 3 columns, that are influenced by the constants. After $\mathsf{D}$ and $\mathsf{AC}$, their values are as follows for $(w, z, z)$:

$$\begin{array}{l} \text{Bundle 0:} \\ \text{Bundle 1:} \\ \text{Bundle 2:} \end{array} \underbrace{\begin{array}{c} S(w^2) \oplus S(z^2) \oplus S(z^2 \oplus c) \\ S(w^2) \oplus S(z^2 \oplus c) \\ S(w^2) \oplus S(z^2) \oplus c' \end{array}}_{\text{Column 2}},$$

$$\underbrace{\begin{array}{c} S(w^1) \oplus S(z^1 \oplus c) \oplus S(z^1) \\ S(w^1) \oplus S(z^1) \oplus c' \\ S(w^1) \oplus S(z^1 \oplus c) \end{array}}_{\text{Column 1}}, \underbrace{\begin{array}{c} S(w^0 \oplus c) \oplus c' \\ S(w^0 \oplus c) \oplus S(z^0) \\ S(w^0 \oplus c) \oplus S(z^0) \end{array}}_{\text{Column 0}}$$

and the expressions are the same for $(w, z', z')$, by replacing $z^i$ by $z'^i$. For the first bundles to be equal, the following relations must hold:

$$\begin{aligned} S(z'^2) \oplus S(z'^2 \oplus c) &= S(z^2) \oplus S(z^2 \oplus c) \\ S(z'^1) \oplus S(z'^1 \oplus c) &= S(z^1) \oplus S(z^1 \oplus c) \ , \end{aligned}$$

and each has probability $2^{-2.415}$ to be satisfied.

**Experimental results.** Experiments showed that the success probability of the distinguisher is very close to what we expected. By testing $2^{30}$ pairs, we obtained 31 successes, a probability close to $2^{-25}$. Our unoptimized `C++` implementation took less than 70 minutes on a desktop computer to find these pairs, i.e., about 2 minutes per pair on average.

### 8.1.7 Forgeries with 4-Step Shadow in the Nonce Misuse Scenario

In this section, we show how to use the properties exploited in the distinguishers to create existential forgeries for the S1P mode of operation [Bel+19]. We use a variant of Shadow-512 reduced to 4 steps, starting at Step 2 instead of 0. These 4 steps correspond to the "aggressive parameters" specified in [Bel+19, Section 5]. We consider the single user setting.

The authors of Spook used the S1P mode for its strong integrity guarantees in presence of nonce misuse and leakage, which are formalized in the CIML2 security definition [Ber+17b] in the unbounded leakage model. Our attack requires only to reuse the same nonce three times. Thus, it falls into this scenario, but no more control is necessary.

Given access to an oracle that enciphers and tags messages with Spook (with the reduced-step Shadow-512 considered), our attack builds two messages that have the same authentication tag. This also implies a collision of the hash function where one would remove duplexing and only output the authentication tag as a hash value.

Throughout this section, $\Pi$ will denote the Shadow-512 permutation reduced to Step 2-5 included. We represent the S1P mode in a very abstract manner in Figure 8.7. In particular:

- We use an empty associated data;
- We only consider two-block messages, i.e., two pairs of bundles $(m_0, m_1)$, $(m_2, m_3)$;
- We do need to specify the procedures Initialize and Finalize, both of which combine Shadow and the block cipher Clyde. The only requirement is that `Initialize` produces a 512-bit state from the nonce $N$ and the secret key $K$, and `Finalize` produces a 128-bit authentication tag from the current 512-bit state.

Messages are XORed to the rate part, which is of 256 bits, and constituted by bundles 0 and 1. The value of the rate is outputted as the ciphertext blocks, and thus, can be observed. The capacity (bundles 2 and 3) cannot be observed.

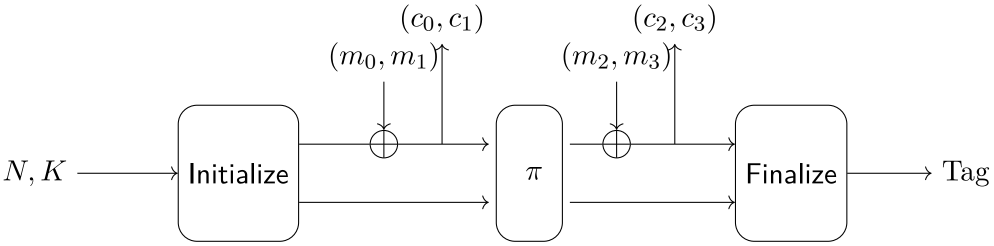


**Figure 8.7:** S1P mode in our attack. Π denotes the Shadow-512 permutation reduced to Steps 2 to 5 (included).

**Attack outline.** Our goal is to output two plaintexts and a nonce $N$ that yield the same authentication tag. In order to do that, we obtain a collision on the internal state before Finalize. This means that any pair of messages built by appending the same blocks to our colliding pair would also yield the same tag provided that the nonce is reused.

The attack calls an offline subroutine and performs three encryption queries with the same nonce. It succeeds with a probability given in Lemma 8.4.

**Offline step:** we create two pairs of rate values that, for a given capacity value, will yield to a collision on the capacity with high probability after the application of Π.

**First query:** we encrypt a zero message in order to retrieve the value of the rate after the Initialize function.

**Second and third query:** we inject the values computed in the offline step. We retrieve the corresponding ciphertexts.

**Result:** if a collision on the capacity has occurred, then by combining the results of the second and third queries, it is possible to output two plaintexts that yield a collision on the full state before Finalize, thus a collision of the tag.

The whole attack is described in Algorithm 8.6. Before going into its details, we present its offline subroutine (Algorithm 8.5) and study its success probability.

**Lemma 8.4.** *Let $(*, *, a_2, b_2)$ be a Shadow-512 state. Then Algorithm 8.5 returns 4 bundles $(x_2, y_2), (x'_2, y'_2)$ such that $\Pi(x_2, y_2, a_2, b_2) \oplus \Pi(x'_2, y'_2, a_2, b_2) = (*, *, 0, 0)$ with a probability $p \simeq 2^{-24.83}$.*

*Proof.* The property follows from the path depicted in Figure 8.8. On input $(x_2, y_2, a_2, b_2)$, let us apply the Super S-Boxes, D and add the second round constant, we obtain the

**Algorithm 8.5** Algorithm to generate candidate pairs for our 4-step property.

```
Output: two pairs of rate bundles $(x_2, y_2), (x'_2, y'_2)$ such that $\Pi(x_2, y_2, a_2, b_2) \oplus \Pi(x'_2, y'_2, a_2, b_2) = (*, *, 0, 0)$ with a probability $p$ given by Lemma 8.4.
Select a random 128-bit bundle $w_3$.
Invert step 2 on $(w_3, w_3, 0, 0)$, obtaining $(x_2, y_2, *, *)$
Select a difference $\epsilon \in \nabla$
return $(x_2, y_2), (x_2 \oplus \epsilon, y_2 \oplus \epsilon)$
```

state: $(\sigma_2(a_2) \oplus \sigma_3(b_2) \oplus w_3, \sigma_2(a_2) \oplus \sigma_3(b_2) \oplus w_3, *, *)$, which is 2-identical. We note it $(x_3, x_3, a_3, b_3)$. By definition of $\epsilon$, the difference goes through the Super S-Boxes and the state $(x_2 \oplus \epsilon, y_2 \oplus \epsilon, a_2, b_2)$ also maps to a 2-identical state.

The probability to remain 2-identical through Step 3 is $2^{-6}$ and through Step 4 it is $2^{-4}$; the states are independent so that each step counts twice.

Finally, we want a zero-difference in the capacity after Step 5. An analysis analogous to the one of Section 8.1.6 gives a probability approximately equal to $2^{-4.83}$. We obtain a total probability of:

$$\underbrace{1}_{\text{Step 2}} \times \underbrace{2^{-12}}_{\text{Step 3}} \times \underbrace{2^{-8}}_{\text{Step 4}} \times \underbrace{2^{-4.83}}_{\text{Step 5}} = 2^{-24.83} \quad . \qquad \square$$

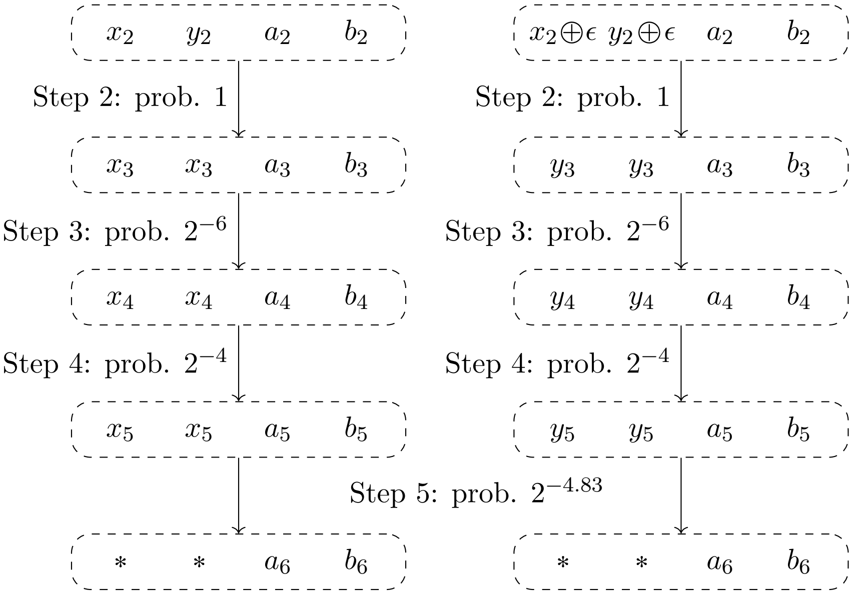


**Figure 8.8:** 4-Step path of Lemma 8.4.

In a nutshell, this property allows us to find a collision on the capacity part of the state after having applied $\Pi$. Since we can control the differences in the rate before and after $\Pi$, we then obtain a collision on the full 512-bit state. This is summarized in Algorithm 8.6.

**Algorithm 8.6** Collision attack on reduced-round Spook

```
Input: access to the S1P mode using 4-step Shadow-512, and reusing nonces
Output: two messages having the same authentication tag, with probability of success $2^{-24.83}$
1: First query: Encrypt an arbitrary two-block (4-bundle) message, for example, $(0,0),(0,0)$, and obtain ciphertexts $(d_0,d_1),(d_2,d_3)$. Let $x_2,y_2,a,b$ be the 4-bundle state after Initialize (immediately before the first application of $\Pi$). Then $d_0,d_1 = x_2,y_2$.
2: Offline step: Use Algorithm 8.5 to obtain two pairs of rate bundles $(x'_2,y'_2),(x''_2,y''_2)$ such that $\Pi(x'_2,y'_2,a,b) \oplus \Pi(x''_2,y''_2,a,b) = (*,*,0,0)$ with probability $p \simeq 2^{-24.83}$.
       ▻ Since $a,b$ is the unknown capacity value, it is impossible to know whether the algorithm has succeeded, and we must proceed to the next steps in all cases.
3: Second query: Encrypt (with the same nonce) $(x_2 \oplus x'_2, y_2 \oplus y'_2),(0,0)$ and obtain $(c'_0,c'_1),(c'_2,c'_3)$. Then $(c'_2,c'_3)$ is the value of the rate after the application of $\Pi$ on $(x'_2,y'_2,a,b)$.
4: Third query: Encrypt (with the same nonce) $(x_2 \oplus x''_2, y_2 \oplus y''_2),(0,0)$ and obtain $(c''_0,c''_1),(c''_2,c''_3)$. Then $(c''_2,c''_3)$ is the value of the rate after the application of $\Pi$ on $(x''_1,y''_1,a,b)$.
5: return the two 4-bundle plaintexts: $(x_2 \oplus x'_2, y_2 \oplus y'_2),(c'_2,c'_3)$ and $(x_2 \oplus x''_2, y_2 \oplus y''_2),(c''_2,c''_3)$ and the nonce $N$ that was used. Then these plaintexts, encrypted with this nonce, yield the same internal state before the Finalize procedure, and the same tag, with probability $p \simeq 2^{-24.83}$.
```

**Experimental results.** The probability of success given in Lemma 8.4 has been verified independently. Furthermore, we have fully implemented the attack against 4-step S1P itself. After $2^{30}$ trials, we obtained 41 successful collisions, with an experimental probability of success of $2^{-24.64}$ which backs the theoretical $2^{-24.83}$. In practice, our un-optimized C++ implementation needs about 15 minutes to find one collision.

### 8.1.8 Discussion

The attacks against Shadow and Spook that we presented do not, despite their efficiency, contradict the security analysis of the designers. For example, with Algorithm 8.5, we are able to obtain input pairs which will make the capacity collide with high probability, but this does not hold for random input pairs.

**Possible extensions.** Our forgery attack uses the same properties as the distinguishers, with a difference in $\nabla$ and the mapping of 2-identical states to 2-identical states. However, since we cannot control the capacity part, we cannot consider the steps before Step 2.

We remark that we can add a round at the end of the reduced-step Shadow, as this additional round does not traverse $L$. But this falls outside the scope of the primitive. We also remark that the differences obtained at the input of step 2 are very sparse. Since the generic complexity for a collision on the 256-bit capacity is $2^{128}$, there is much room for an extended attack with one more round at the beginning. But this seems far from trivial and less interesting now that the designers have moved to a new version.

**A design criterion for constants.** The goal of round constants is to break the symmetry in a primitive, whether between successive rounds or within the same round. This is especially important if the primitive handles a big state, as in permutation-based cryptography. Our attacks on Spook provide a new criterion for choosing round constants: they should not allow the kind of internal cancellation that we observed. However, we expect such a property to be rather rare, and it does not seem to apply to other designs in the LWC process.

**Changes made by the designers.** In the Spook V2 proposal [Bel+20], the authors made several changes to Shadow that aim at avoiding the attacks proposed in this section, without harming its performance: in fact, it becomes even better. No changes were made to Clyde nor the mode of operation.

First of all, they replace the D transform by an MDS matrix taken from [DL18]. Notably, the distinguisher presented in Section 8.1.5 cannot reach 6 steps anymore, and the forgery attack of Section 8.1.7 is also mitigated. Next, they change the constants and the way they are added to the bundles: • for round A, a constant word of 32 bits is added to the second row of each bundle. • for round B, a constant word of 32 bits is added to all rows of the first bundle. Since the internal symmetries that we exploited explicitly relied on the sparseness of the constants and their interaction in the first 4 columns of each bundle, it is extremely unlikely that such properties remain in Spook V2.

## 8.2 Cryptanalysis of Gimli

Gimli is another second-round candidate of the NIST LWC process. The submission is also permutation-based and proposes the following algorithms:

- Gimli, a cryptographic permutation operating on states of 384 bits;
- Gimli-Hash, a hash function using Gimli in a Sponge mode, producing digests of 256 bits;
- Gimli-Cipher, an AEAD using Gimli in a Sponge Duplex mode.

In particular, the definition of Gimli-Hash and Gimli-Cipher are much simpler than in Spook, since they are only based on the full permutation. Gimli was the subject of a previous publication in CHES 2017 [Ber+17a]. Apart from its lightness, it is also intended to be "cross-platform", thus being efficient and easy to implement both in software and hardware.

In this section, we analyze the permutation Gimli and describe reduced-round attacks on the hash function Gimli-Hash. On a very abstract level, our attacks work by combining two properties of the permutation: • it has a rather slow diffusion, and • it exhibits strong internal symmetries. The combination of these properties first allows us to build a distinguisher on full Gimli, with a cost $2^{64}$ against a generic $2^{96}$. A practical variant attacking 23 rounds out of 24, *or* 24 rounds of a shifted version of Gimli, costs $2^{32}$ instead of $2^{96}$, and has been implemented. Next, we study collisions on the hash function. Since Gimli is used in a Sponge mode, the attack setting is rather simple, as it amounts to find two rate inputs that, for a given capacity value, make the output capacity collide. We show how to attack 12 rounds out of 24 and 8 rounds in practical time. Finally, we also design *semi-free start* collision attacks, where the internal state value can be controlled, reaching up to 18 rounds.

In the quantum setting, the comparison with generic quantum collision search is more favorable, as it was exploited for the first time in [HS20]. Our collision and semi-free start collision attacks can be respectively extended to 14 and 20 rounds.

### 8.2.1 Description of Gimli and Gimli-Hash

We first describe Gimli and Gimli-Hash. We adopt the following notations: $\ll$, $\gg$, $\lll$, $\ggg$ represent respectively shift left, shift right, *rotate left* and rotate right operations on 32 bits. The variables $x, y, z$ will denote elements of $\mathbb{F}_2^{32}$. We let $x_i$ denote the ($i$ mod 32)$^{th}$ bit of $x$ ($x_{33} = x_1$) with $x_0$ least significant (right-most).

**Internal state.** We let $S$ denote a 384-bit Gimli state, which is the concatenation of 4 *columns* of 96 bits, that we name $A, B, C, D$, where $A$ is column number 0, and $D$ is column number 3. Each column is cut into three 32-bit *words* $x, y, z$ which are denoted $A_x, A_y, A_z$. Thus, the state is a $4 \times 3 \times 32$ parallelepiped. We will speak of the $x$ *lane* to denote the sequence or concatenation of words $A_x, B_x, C_x, D_x$.

**SP-Box.** The only non-linear operation in Gimli is the *SP-Box*, which is applied columnwise. On input $x, y, z$, it updates the three words as follows:

1. Rotate $x$ and $y$: $x \leftarrow x \lll 24, y \leftarrow y \lll 9$ .
2. Perform the following non-linear operations in parallel (note that shifts are used rather than rotations):
$$\begin{aligned} x &\leftarrow x \oplus (z \ll 1) \oplus ((y \wedge z) \ll 2) \ , \\ y &\leftarrow y \oplus x \oplus ((x \vee z) \ll 1) \ , \\ z &\leftarrow z \oplus y \oplus ((x \wedge y) \ll 3) \ . \end{aligned}$$
3. Swap $x$ and $z$: $(x, z) \leftarrow (z, x)$ .

**Rounds.** Gimli applies a sequence of 24 rounds numbered from 24 downto 1 inclusively. Each round applies an SP-Box layer, then performs a *swap* (every two rounds, either a "big swap" or a "small swap") and a constant addition (every four rounds). The constant at round $i$ is $\mathsf{RC}_i = \texttt{0x9e377900} \oplus i$. Note that all the attacks studied hereafter are independent of the choice of round constants. An algorithmic depiction of full Gimli is given in Algorithm 8.7, and the rounds are represented as in Figure 8.9. The state before round $i$ is denoted $S^i = A^i, B^i, C^i, D^i$. Thus, $A^i_x$ denotes the $x$ branch of the $A$ column before the SP-Box of round $i$ is applied, and so on. When considering reduced-round versions of Gimli, we let $\text{Gimli}(r, r')(S)$ denote the application of rounds $r$ to $r'$ *included*, where $r' \leqslant r$.

**Algorithm 8.7** The full Gimli permutation.

```
Input: State $S = A, B, C, D$
Output: Gimli($S$)
for $r = 24$ downto 1 inclusive do
    $A, B, C, D \leftarrow SP(A), SP(B), SP(C), SP(D)$        ▷ SP-Box layer
    if $r \mod 4 = 0$ then
        Swap $A_x$ and $B_x$, swap $C_x$ and $D_x$        ▷ small swap
        $A_x \leftarrow A_x \oplus \mathsf{RC}_r$        ▷ Constant addition
    else if $r \mod 2 = 0$ then
        Swap $A_x$ and $C_x$, swap $B_x$ and $D_x$        ▷ big swap
Return $S$
```

**Gimli-Hash.** This function is built using the Gimli permutation in a Sponge construction, as represented in Figure 8.10 and Algorithm 8.8. The Gimli state is initialized to the all-zero value. The message is padded and separated into blocks of size $r = 128$, introducing message blocks of 128 bits between two permutation applications by XORing them to the first 128 bits of the state. Once all the padded message blocks are processed, a 32-byte hash is generated by outputting the rate value, applying once

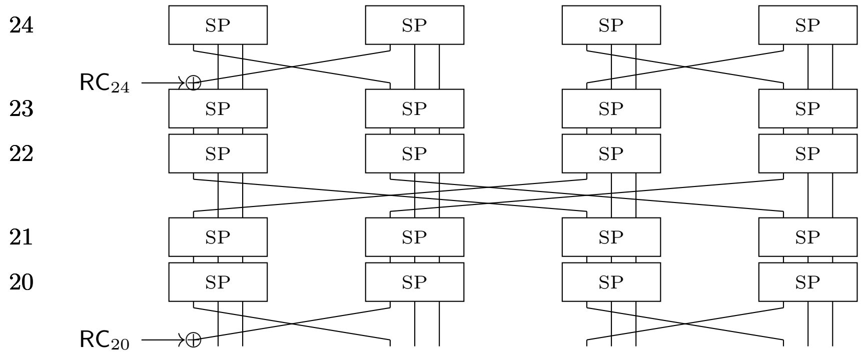


**Figure 8.9:** Our representation of GIMLI rounds (here GIMLI(24, 20)). Each wire represents a word.

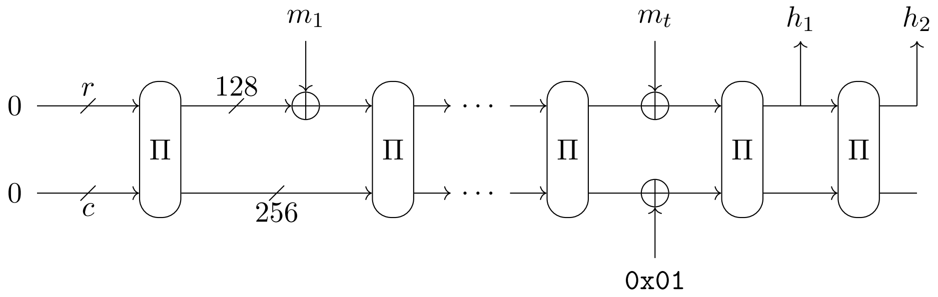


**Figure 8.10:** GIMLI-HASH (Π stands for the GIMLI permutation). The rate is the $x$ lane $S_x = A_x, B_x, C_x, D_x$. The capacity is $S_{y,z} = A_{y,z}, B_{y,z}, C_{y,z}, D_{y,z}$.

**Algorithm 8.8** GIMLI-HASH (from [Ber+19]).

```
Input: $M \in \{0,1\}^*$
Output: $h \in \{0,1\}^{256}$
$S \leftarrow 0$                                         ▷ Initialize state to 0
$m_1, \ldots, m_t \leftarrow \mathrm{pad}(M)$
for $i$ from 1 to $t$ do
    if $i = t$ then
        $D_z \leftarrow D_z \oplus$ 0x01000000
    $S \leftarrow \mathrm{absorb}(S, m_i)$        ▷ XOR $m_i$ in the rate: $A_x, B_x, C_x, D_x$, apply GIMLI
$h \leftarrow A_x \| B_x \| C_x \| D_x$
$S \leftarrow$ GIMLI$(S)$
$h \leftarrow h \| A_x \| B_x \| C_x \| D_x$
Return $h$
```

more the permutation, and outputting the rate again. In Gimli-Hash, the rate is the $x$ lane $S_x = A_x, B_x, C_x, D_x$ and the capacity is $S_{y,z} = A_{y,z}, B_{y,z}, C_{y,z}, D_{y,z}$.

### 8.2.2 Previous work

We provide here a brief overview of the main previous third-party results of cryptanalysis against either the permutation or the NIST candidate Gimli. A summary of these results, including the results of [Fló+20] that will not be presented in this chapter (linear and differential-linear cryptanalysis), is given in Table 8.4. Notice that all the cryptanalysis previously considered were classical attacks.

**Structural permutation distinguisher on 22.5 rounds.** In [Ham17], Hamburg proposed the first third-party cryptanalysis of the Gimli permutation, providing distinguishers on reduced-round versions of the permutation. This analysis does not depend on the details of the SP-Box, and is based only on the slow diffusion of Gimli. Thus, it follows a similar path as the distinguishers of Section 8.2.3. In his work, Hamburg defines a PRF with 192-bit input $x$ and 192-bit key $k$ that computes $F(k, x) = \mathrm{trunc}_{192}(\textsc{Gimli}(k\|x))$. He gives a distinguishing attack in time $2^{64}$ for 15.5 rounds (omitting the final swap), and a key-recovery attack on $F$ when using 22.5 rounds of Gimli, precisely rounds 25 to 2.5 (omitting again the final swap). This attack runs in time $2^{138.5}$ with a memory requirement of $2^{129}$, which is faster than the expected $2^{192}$, and thus shows that 22.5-round Gimli behaves differently than what could be expected from a random permutation.

Hamburg's attacks are based on a meet-in-the-middle approach, exploiting the slow diffusion by tabulating some of the values that are passed from an SP-Box to another. The 15.5-round distinguisher relies on a table of size $2^{64}$, and the 22.5-round attack on a table of size $2^{128}$. These attacks are far from practical.

**Zero-sum permutation distinguishers on 14 rounds.** In [Cai+19], Cai, Wei, Zhang, Sun and Hu present a zero-sum distinguisher on 14 rounds of Gimli. This distinguisher uses the inside-out technique and improves by one round the integral distinguishers given by the designers.

**ZID permutation distinguishers.** In an independent work [LIM20b], Liu, Isobe, and Meier present a "hybrid zero-internal differential" (ZID) distinguisher on full Gimli, which extends a ZID distinguisher of previous unpublished work. The basic ZID distinguisher happens to be what we call an *internal symmetry distinguisher*, where states with symmetries are produced in the input and in the output of a reduced-round variant of Gimli. A "hybrid" one adds a limited birthday-like property (which is absent from our distinguishers). The steps that they take are however different from ours, as this distinguisher only spans 14 rounds. Compared with our analysis in Section 8.2.3, they will actually start from a much more constrained middle state, which limits the number of rounds by which one can extend the distinguisher afterwards (or significantly increases

**Table 8.4:** Attacks against algorithms of the Gimli family, compared with the generic complexities expected. Time is counted in evaluations of Gimli, and memory in 128-bit blocks. Attacks that were actually implemented are written in **bold**. $\epsilon$ is a term that we only estimated experimentally ($\epsilon \approx 10$, see Section 8.2.4). In rounds attacked, $r_1 \to r_2$ means rounds $r_1$ to $r_2$ included.

| | Technique | Rounds | Time | Mem. | Gen. | Reference |
|---|---|---|---|---|---|---|
| Distinguishers on the permutation (real rounds: 24 → 1) | Key-recovery on Gimli-PRF | 25 → 2.5 | 138.5 | 128 | 192 | [Ham17] |
| | | 15.5 | 64 | 64 | 192 | [Ham17] |
| | Zero-sum | 14 | 351 | negl. | 384 | [Cai+19] |
| | ZID | 18 | 2 | negl. | 4 | [LIM20b] |
| | Linear | 12 | 198 | negl. | 384 | [Fló+20] |
| | Linear | 16 | 300 | negl. | 384 | [Fló+20] |
| | Differential-Linear | 15 | 87.4 | negl. | 192 | [Fló+20] |
| | Differential-Linear | 16 | 110.8 | negl. | 192 | [Fló+20] |
| | Differential-Linear | 17 | 154.8 | negl. | 192 | [Fló+20] |
| | **Symmetry** | **23 → 0** | **32** | **negl.** | **96** | Sec. 8.2.3 |
| | Symmetry | 27 → 0 | 64 | negl. | 96 | Sec. 8.2.3 |
| Preimages on Gimli-Hash | Divide-and-conquer | 2 | 42.4 | 32 | 128 | [LIM20b] |
| | | 5 | 96 | 65.6 | 128 | [LIM20b] |
| Preimages on Gimli-XOF-128 | | 9 | 104 | 70 | 128 | [LIM20b] |
| Collisions on Gimli-Hash | Divide-and-conquer | 5 | 65 | – | 128 | [LIM19] |
| | | **3** | **practical** | – | **128** | [LIM19] |
| | | 6 | 64 | 64 | 128 | [LIM20a] |
| | **Symmetry** | **21 → 14** | **32 + $\epsilon$** | **negl.** | **128** | **Sec. 8.2.4** |
| | Symmetry | 12 | 96 + $\epsilon$ | negl. | 128 | Sec. 8.2.4 |
| | Quantum | 14 | 64 + $\epsilon$ | negl. | 85.3 | Sec. 8.2.4 |
| Semi-free start collisions on Gimli-Hash | Symmetry | 8 | 64 | negl. | 128 | [LIM20a] |
| | Symmetry | 12 | 32 + $\epsilon$ | negl. | 128 | Sec. 8.2.4 |
| | Symmetry | 16 | 96 + $\epsilon$ | negl. | 128 | Sec. 8.2.4 |
| | Symmetry | 18 | 96 + $\epsilon$ | 64 | 128 | Sec. 8.2.4 |
| | Quantum | 20 | 64 + $\epsilon$ | 64 | 85.3 | Sec. 8.2.4 |

the complexity). In contrast, we complete the middle state in multiple successive steps, each step ensuring that more rounds will be later covered. After the publication of our distinguisher, the authors of [LIM20b] proposed to modify it in order to reduce the time complexity to $2^{52}$ instead of $2^{64}$.

**Collisions and preimages on Gimli-Hash.** In [ZDW19], Zong, Dong and Wang study Gimli among other candidates of the competition. They present a 6-round collision attack on Gimli-Hash of complexity $2^{113}$, using a 6-round differential characteristic where the input and output differences are active only in the rate. This differential characteristic was invalidated in [LIM20a].

In [LIM19], [LIM20b] and [LIM20a] Liu, Isobe and Meier give collision and preimage attacks on reduced-round Gimli-Hash. Their attacks rely on divide-and-conquer methods, exploiting the lack of diffusion between the columns, as did Hamburg, but they also rely on SP-Box equations in order to attack the hash function itself. These equations are different from those that we will solve in Section 8.2.4, and they mostly relate the input and outputs of a single SP-Box, whereas we study directly two SP-Boxes. Their analysis is also much more precise, since they prove running times of solving these equations.

After giving a meet-in-the-middle generic preimage attack of time and memory complexity $2^{128}$, which sets a bound against the sponge construction used in Gimli-Hash, they give practical preimage attacks on 2-round Gimli-Hash and practical collision attacks on 3-round Gimli-Hash. They give a collision attack on 5-round Gimli-Hash with a time complexity $2^{65}$ and a second preimage attack with time complexity $2^{96}$. They give in [LIM20b] a preimage attack on 5-round Gimli-Hash. In [LIM20a], they give a semi-free start collision attack on 8 rounds and a state-recovery attack on the AE scheme for 9 rounds.

### 8.2.3 Internal Symmetry Distinguishers against Gimli

In this section, we present distinguishers on the Gimli permutation that allow to reach its full 24-round version. A practical version on 23 runs in time $2^{32}$ and was implemented. Our results do not rely on the specification of the SP-Box, neither on the value of the round constants: we can replace the SP-Box by any permutation picked uniformly at random, and we can also replace the round constants by random 32-bit values.

The only property that we exploit is the slow diffusion of Gimli via the Small and Big Swap operations, allowing to propagate states with internal symmetries. We define a notion of *2-identical states* for Gimli, which is similar to the definition given in Section 8.1.2 for Shadow states.

**Definition 8.2** (2-identical states)**.** A state $S$ is *2-identical* if $B = D$, if $A = C$, or if one of these properties holds up to a swap and a constant addition.

Our *internal symmetry distinguishers* find a 2-identical input that is mapped to a 2-identical output. Since there are 96 bits of constraint, a generic algorithm returning such an input should run in time $2^{96}$ by evaluating the permutation on a set of inputs satisfying the property until the output matches it by chance. Our aim is to find more efficient algorithms in the case of Gimli. This is slightly different from the limited-birthday distinguishers that we used in the case of Shadow, since we do not consider pairs of states satisfying a truncated differential, but a single state that belongs to a vector space.

#### 8.2.3.1 23-round Practical Distinguisher

We design an internal symmetry distinguisher on 23 rounds of Gimli, that is represented in Figure 8.11, running in time equivalent to $2^{32}$ evaluations of Gimli on average. Algorithm 8.9 starts from a symmetric state in the middle and completes the state $S^{11}$ in three steps. Each step assigns a value to more words of the state, and ensures that the 2-identical symmetry property goes through more rounds.

**Algorithm 8.9** 23-round internal symmetry distinguisher.

**Output:** a 2-identical state $S$ such that $\textsc{Gimli}(23, 1)(S)$ is 2-identical
We start from the middle. We will be interested in the state $S^{11}$.

1. **Select** $A_x^{15}, A_y^{15}, A_z^{15}$ and $C_x^{15} = A_x^{15} \oplus \mathsf{RC}_{16}$, $C_y^{15} = A_y^{15}$, $C_z^{15} = A_z^{15}$ such that $B_x^{11} = D_x^{11}$.

   Notice that due to the small swap operation, the values $B_x^{11}$ and $D_x^{11}$ actually come from $A$ and $C$ and depend only on $A^{15}$ and $C^{15}$. At this point, we have ensured that for any values of $B^{15} = D^{15}$:

   - $S^{23}$ is 2-identical: indeed, $A$ and $C$ will remain identical from rounds 16 to 19 backwards. Then, the small swap backwards injects the same value in $A$ and $C$ since $B$ and $D$ are also identical. Thus, $A^{23} = C^{23}$.
   - $S^7$ is 2-identical: indeed, since $B_x^{11} = D_x^{11}$, $B$ and $D$ remain equal until the SP-Box layer of round 8, and the 2-identical property remains after the small swap of round 8.

   Once good values have been found, we can compute part of the state $S^{11}$: $A_{y,z}^{11}$, $C_{y,z}^{11}$, and $B_x^{11} = D_x^{11}$ are fixed. The rest remains free.

2. **Select** $A_x^{11} = C_x^{11} \oplus \mathsf{RC}_{12}$ such that $B_x^7 = C_x^7$. At this point, the two-identicality of the output state is preserved through 4 more rounds (until round 4 included): $S^3$ is 2-identical.

   In the state $S^{11}$, $B_{y,z}^{11} = D_{y,z}^{11}$ remain free.

3. **Select** $B_{y,z}^{11} = D_{y,z}^{11}$ such that $B_x^3 = C_x^3$. Thus, the output $S^0$ is 2-identical.

Each step of Algorithm 8.9 requires to evaluate a few SP-Boxes $2^{32}$ times (we do not even need to evaluate the inverse SP-Box). The total amount of computations is

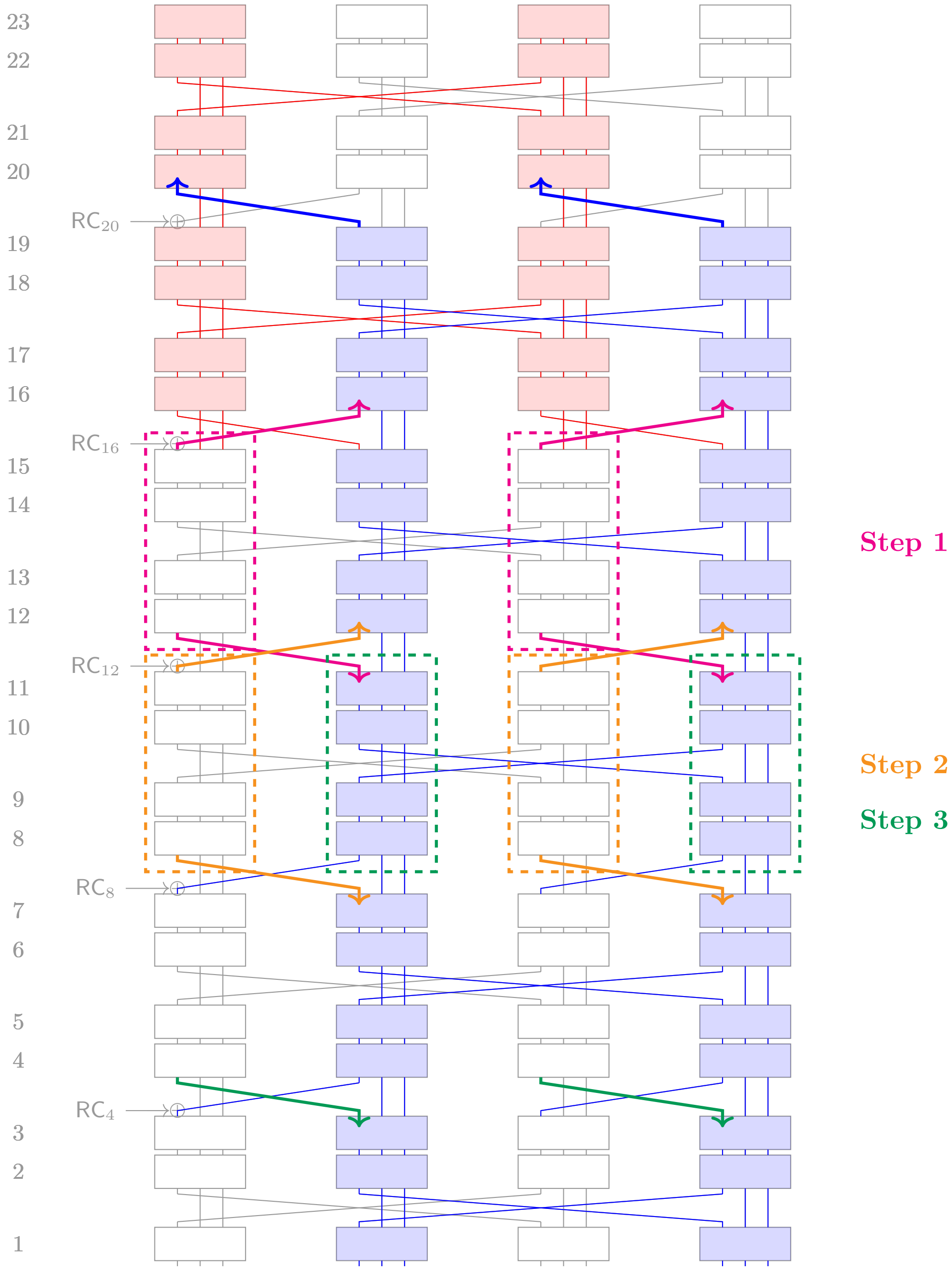


**Figure 8.11:** Distinguisher on GIMLI$(23, 1)$. The same color for symmetric branches or columns at a given round means that they are equal.

smaller than $2^{32}$ evaluations of 23-round Gimli. Notice also that the algorithm uses only a small amount of memory. Our implementation of Algorithm 8.9 ran in less than one hour on a regular laptop.

The time complexity of the algorithm can be computed as follows: $8 \times 2^{32}$ SP-Box evaluations for the first step, $8 \times 2^{32}$ for the second and $16 \times 2^{32}$ for the third, meaning a total of $8 \times 2^{32} + 8 \times 2^{32} + 16 \times 2^{32} = 40 \times 2^{32}$ which is less than $2^{32}$ evaluations of 23-round Gimli (each of them consisting essentially of 92 SP-Box evaluations). This complexity is to be compared to that of the generic algorithm for obtaining our internal symmetry property, which costs $2^{96}$.

An unoptimized C++ implementation finds a valid state in about 900 seconds on a standard laptop. Below is an example:

| | | | | |
|---|---|---|---|---|
| $S^{23}$: | 7f9fcf70 | 6aedf7e6 | 7f9fcf70 | cb2f0e6a |
| | 0ba2f1f9 | f339b619 | 0ba2f1f9 | f70cf15c |
| | b2ee8259 | df0b4801 | b2ee8259 | 3856106d |
| $S^0 = \text{Gimli}(23,1)(S^{23})$: | a8ef848d | 8c17b743 | 9615b3bc | 8c17b743 |
| | 541122c5 | 30530879 | 8d9d5d30 | 30530879 |
| | 74b6dbe6 | 18885a6e | 744b55c1 | 18885a6e |

#### 8.2.3.2 Distinguisher on full Gimli and Extensions

An extension of our distinguisher to the full Gimli is a trivial matter. Indeed, after running Algorithm 8.9, we obtain a 2-identical input state $S^{23} = A^{23}, B^{23}, C^{23}, D^{23}$ with $A^{23} = C^{23}$. Then, if $B^{23}_x = D^{23}_x$, which is a 32-bit condition, the state remains 2-identical after the inverse round 24. By repeating the previous procedure $2^{32}$ times, we should find an input value that verifies the output property. The generic complexity of finding a 2-identical input that generates a 2-identical output is still $2^{96}$. Thus, full Gimli can be distinguished in time less than $2^{32+32} = 2^{64}$ full Gimli evaluations, and constant memory.

An interesting question is: how many rounds of a Gimli-like permutation can we target? The distinguisher works mainly because the diffusion in Gimli is somewhat slow. Thus, a possible fix would be to increase the number of swaps, for example by having one in each round instead of every two rounds. An attack exploiting this behavior that worked previously for $r$ rounds would now *a priori* work for $r/2$ rounds only. Of course, the details of the SP-box could allow further improvement of these results given that a single iteration would now separate the swaps rather than a double.

**Extending to 28 rounds.** It is trivial to adapt this distinguisher to an extended version of Gimli with more rounds. The 2-identicality of $S^0$ is preserved after one round since the next round would apply only an SP-Box layer and a small swap. Similarly, the 2-identicality of $S^{24}$ is preserved after 3 more inverse rounds since the next swap operation is a big swap which exchanges data between $A$ and $C$ only. Thus, our practical distinguisher works against $\text{Gimli}(23, 0)$ (a 24-round version of Gimli shifted by one round), and our extended distinguisher works against Gimli$(27, 0)$ (a 28-round version of Gimli).

### 8.2.4 Classical Collisions on Reduced-Round Gimli-Hash

We now present collision attacks on the hash function Gimli-Hash instantiated with a round-reduced Gimli. We will consider two kinds of collisions:

**Full-state collisions (standard):** given an internal state $S = S_x, S_y, S_z$ of 384 bits that is not controlled, we find two rate values $S'_x$ and $S''_x$ (the $x$ lane) such that

$$\text{Gimli}(r, r')(S'_x, S_y, S_z) = T'_x, T_y, T_z \text{ and } \text{Gimli}(r, r')(S''_x, S_y, S_z) = T''_x, T_y, T_z$$

collide on the capacity (the $y$ and $z$ lanes). This gives two triples of message blocks $M_0, M_1, M_2$ and $M_0, M'_1, M'_2$ such that the full state after absorbing them is equal:

- $M_0$ determines the value of $S$;
- $M_1$ and $M'_1$ are $S_x \oplus S'_x$ and $S_x \oplus S''_x$ respectively;
- $M_2$ and $M'_2$ are 0 and $T'_x \oplus T''_x$ respectively.

Appending any sequence message blocks afterwards will yield a collision of the hash value.

**Semi-free start collisions:** we find pairs of (384-bit) states $S, S'$ such that $S$ and $S'$ differ only in the $x$ lane before and after Gimli$(r, r')$. This does not yield a collision on the hash function as we would need to choose the value of the same initial state; however, it represents a vulnerability that may be used in the context of the Gimli modes of operation, or if the adversary is given more control. In Gimli-Cipher, it enables someone to create a key, a nonce and a pair of messages which yield the same tags.

These results are summarized in Table 8.5.

**Table 8.5:** Collision attacks on round-reduced Gimli. $t_e$ denotes the time to solve one of the Equations (8.9) to (8.12).

| Type | Nbr of rounds | Time complexity | Memory |
|---|---|---|---|
| Standard | 8 | $8 \times 2^{32} \times t_e$ (practical) | negl. |
| Standard | 12 | $8 \times 2^{96} \times t_e$ | negl. |
| Quantum | 14 | $\simeq 8 \times 2^{64} \times t_e$ | negl. |
| Semi-free start | 12 | $10 \times 2^{32} \times t_e$ | negl. |
| Semi-free start | 16 | $10 \times 2^{96} \times t_e$ | negl. |
| Semi-free start | 18 | $7 \times 2^{96} \times t_e$ | $2^{64}$ |
| Semi-free start | 18 | $2^{96}$ | $2^{96}$ |
| Semi-free start, quantum | 20 | $\simeq 2^{64} \times 10 \times t_e$ | $2^{64}$ |

### 8.2.4.1 SP-Box Equations and How to Solve Them

Our attacks exploit the slow diffusion of Gimli and the simplicity of the SP-Box (contrary to the distinguishers on the permutation, which worked regardless of the SP-Box used). In this section, we describe a series of "double SP-Box equations"; solving them will be the main building block of our attacks. We define the following equations.

$$\text{Given } y, z, \text{ find } x \neq x' \text{ such that } SP^2(x,y,z)_x = SP^2(x',y,z)_x \ . \tag{8.9}$$

$$\text{Given } y, z, y', z', \text{ find } x \text{ such that } SP^2(x,y,z)_x = SP^2(x,y',z')_x \ . \tag{8.10}$$

$$\text{Given } y, z, y', z', \text{ find } x \text{ such that } SP^2(x,y,z)_z = SP^2(x,y',z')_z \ . \tag{8.11}$$

$$\text{Given } y, z, x', \text{ find } x \text{ such that } SP^2(x,y,z)_x = x' \ . \tag{8.12}$$

**Number of solutions.** Except Equation (8.9), all these equations have *on average*, when the inputs are drawn uniformly at random, a single solution. However, the variance on the number of solutions depends on the equation considered. For example, only approx. 6.2% of inputs to Equation (8.10) have a solution, and they have on average 82.4 solutions each. Equation (8.12) gives a little more than 1.5 solutions. This variance is not a problem for us, as long as we can produce efficiently all solutions of the equations, which remains the case. In order to simplify our presentation, we will do as if equations (8.10), (8.11) and (8.12) always gave exactly a single solution for each input.

**Solving the equations.** We use an off-the-shelf SAT solver [SNC09]. In some cases, more time seems spent building the SAT instance rather than solving it, and we believe that our current implementation is highly unoptimized.

The solver allows us to retrieve all solutions of a given equation (we treat Equation (8.9) differently because it has on average $2^{32}$ of them). Let us consider the average time to produce a solution when random inputs are given. On a standard laptop, this time varies between approximately 0.1 milliseconds (Equation (8.10)) and 1 millisecond (Equation (8.12)). This difference mainly stems from the fact that Equation (8.10) often has no solutions, and that the solver quickly finds a counterexample, while Equation (8.12) practically always has solutions that must be found.

On the same computer, an evaluation of the full Gimli permutation (not reduced-round) takes about 1 microsecond, so there is approximately a factor 1000 between computing Gimli and solving a double SP-Box equation.

We consider that all equations have approximately the same complexity and introduce a factor $t_e$ that expresses the time taken to solve them in number of evaluations of Gimli.

### 8.2.4.2 Full-State Collisions

We consider the reduced-round permutation Gimli$(21, 14)$ (8 rounds). We omit the last swap, because it does not mix between the rate and the capacity. The situation is represented in Figure 8.12 and described in Algorithm 8.10. As before, we name $S^i$ the state immediately before round $i$.

**Algorithm 8.10** 8-round collision attack on Gimli.

**Input:** an input state $A^{21}, B^{21}, C^{21}, D^{21}$.

**Output:** values $A^{21}_x, A'^{21}_x, B^{21}_x, C^{21}_x, D^{21}_x$ such that by putting $A^{21}_x, B^{21}_x, C^{21}_x, D^{21}_x$ and $A'^{21}_x, B^{21}_x, C^{21}_x, D^{21}_x$ respectively in the rate, after Gimli$(21, 14)$ (without the last swap), the state differs only on $A_x$.

Note that with this attack, the two 128-bit message blocks differ only on 32 bits.

**Complexity:** $7 \times 2^{32} \times t_e$ evaluations of Gimli and $2^{32}$ memory *or* $8 \times 2^{32} \times t_e$ and negligible memory.

The attack runs in two main steps, both of which must solve $2^{32}$ times a sequence of SP-Box equations.

**Step 1:** find good $A^{21}_x, A'^{21}_x$.

1. Find all pairs $A^{21}_x, A'^{21}_x$ such that the branch $B^{19}_x$ collides (there are $2^{32}$ such pairs, that can be found in time $2^{32}$).
2. For each pair, compute $A^{19}_y, A^{19}_z, A'^{19}_y, A'^{19}_z$ and solve the SP-Box equation (8.10): find $A^{19}_x$ such that the branch $C^{17}_x$ collides (there is on average one solution)
3. Given this value, compute $A^{17}_y, A^{17}_z, A'^{17}_y, A'^{17}_z$ and solve the SP-Box equation (8.10) again: find $A^{17}_x$ such that the branch $B^{15}_x$ collides (there is on average one solution)
4. Given these values, compute $A^{15}_y, A^{15}_z, A'^{15}_y, A'^{15}_z$ and solve Equation (8.11): find $A^{15}_x$ such that $A^{13}_z = A'^{13}_z$.

Since we do that $2^{32}$ times, we expect on average a single solution which also has $A^{13}_y = A'^{13}_y$.

Now that we have found $A^{21}_x, A'^{21}_x$, it remains to find $B^{21}_x, C^{21}_x, D^{21}_x$ that give the wanted $A^{19}_x, A^{17}_x, A^{15}_x$ (in **red** in Figure 8.12). We expect on average a single solution, and little variation on the number of solutions, as only Equation (8.12) is involved.

**Step 2:** find $B^{21}_x, C^{21}_x, D^{21}_x$.

**2.1** **Find $B^{21}_x$** by solving Equation (8.12), given the input $y$ and $z$, and the output $x$ wanted. Deduce the values of $B^{17}_y, B^{17}_z$

**2.2** Given $B^{17}_y, B^{17}_z$, and $A^{15}_x$, solve Equation (8.12) again to **find $B^{17}_x$.**

**2.3** **Find $C^{21}_x, D^{21}_x$** that lead to the wanted $A^{17}_x, B^{17}_x$. First guess the value of $C^{21}_x$, deduce $C^{19}_y, C^{19}_z$ and with $C^{19}_y, C^{19}_z, A^{17}_x$, solve Equation (8.12) to obtain $C^{19}_x$. Next, given $D^{21}_y, D^{21}_z$ and $C^{19}_x$, solve Equation (8.12) to obtain $D^{21}_x$. Deduce a value for $B^{17}_x$ and check if it matches what we want; we expect to find a match after trying all $2^{32}$ guesses for $C^{21}_x$.

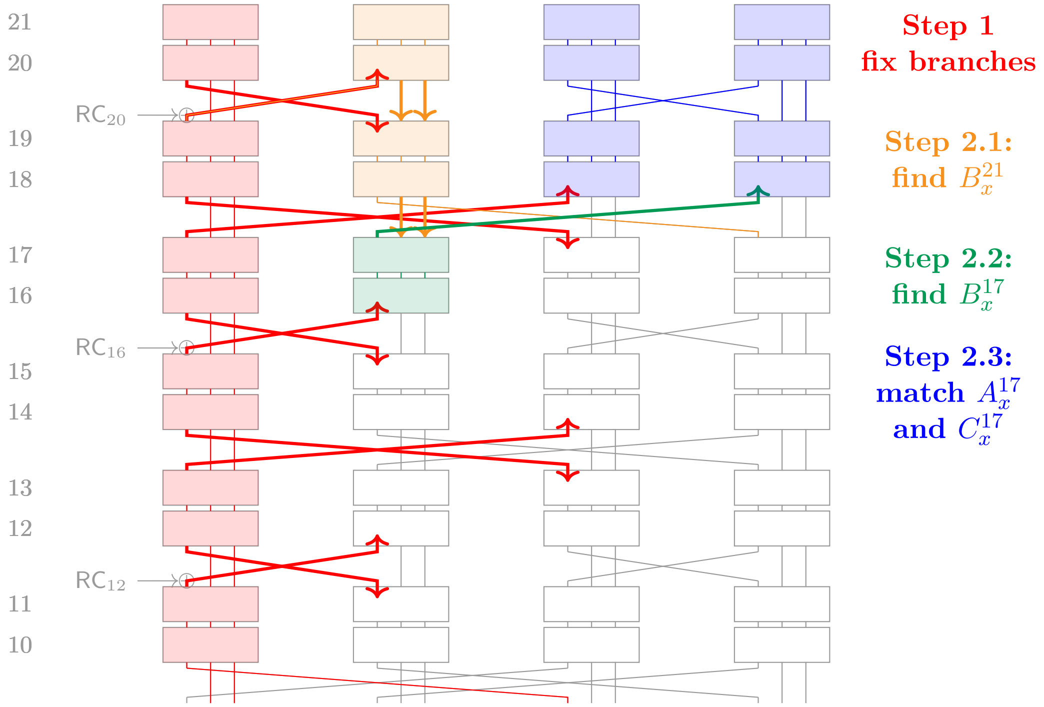


**Figure 8.12:** Collision attack on 8 rounds of Gimli, extended to 12 rounds. The first step fixes the branches in red, which have equal values for the two inputs $A_x^{21}, A_x'^{21}$. Then we find values of $B_x^{21}, C_x^{21}, D_x^{21}$ that will conform to these branches. Then, the whole states are deduced. The branches $A_x^{13}$ and $A_x^{11}$ remain to match.

Algorithm 8.10 finds on average a single solution, with any input state. There is some variance on the number of solutions, that is induced by the SP-Box equations, but it is small in practice. Furthermore, we can eliminate the memory requirement by solving Equation (8.9) for many input random states. Starting from a given state, it suffices to apply one more Gimli permutation with a random message block, in order to re-randomize the input.

Remark that if we omit the second step then we already have a semi-free-start collision attack, because we can reconstruct the inputs $C^{21}$ and $D^{21}$ immediately from the middle.

**Practical application: first step.** Since the experimental value of $t_e$ that we obtained was around $2^{10}$ evaluations of Gimli, the expected cost of this 8-round collision attack was around $8 \times 2^{32} \times 2^{10} = 2^{45}$ evaluations of Gimli, so we could try to perform a practical implementation. We considered rounds 21 to 14 included. We solved Step 1 first, starting from $0, 0, 0, 0$ and using a random message of the form $m_1, 0, 0, 0$ to randomize the first block. We also solved at the same time the two Equations (8.12) that enabled us to go back to $A_x^{17}, B_x^{17}$. We had to produce $15582838652 \simeq 2^{33.86}$ solutions for

Equation (8.9) until we found a solution for Step 1 *and* for both equations. We verified experimentally that each solution for Equation (8.9) yielded on average a solution for the final equation. We obtained in total 5 solutions, as shown below. There are two different solutions for $A_x^{15} \oplus \mathsf{RC}_{16}$, which yield two and three solutions respectively for $B_x^{17}$. The total computation ran in less than 5000 core-hours (parallelized).

| $m_1$ | $A_x^{21}$ | $A_x'^{21}$ | $A_x^{19} \oplus \mathsf{RC}_{20}$ | $A_x^{17}$ | $B_x^{21}$ |
|---|---|---|---|---|---|
| dc84bf38 | bbdb41f3 | 1b1da6e4 | 07f25303 | f793fb5f | aae48b72 |
| $A_x^{15} \oplus \mathsf{RC}_{16}$ | $B_x^{17}$ | $A_x^{15} \oplus \mathsf{RC}_{16}$ | $B_x^{17}$ | $A_x^{15} \oplus \mathsf{RC}_{16}$ | $B_x^{17}$ |
| ddfbc88b | 92f536b6 | ddfbc803 | f72044db | ddfbc803 | 55d2252a |
| ddfbc88b | 0d9605fe | ddfbc803 | b1c91a60 | | |

**Practical application: second step.** We solved Step 2, that is, looking for $C_x^{21}$, $D_x^{21}$ that lead to one of the pairs $A_x^{17}, B_x^{17}$. This step was much faster than the previous one, although it ought to have the same complexity: this is because we paid in step 1 the probability to find a solution (twice) in Equation (8.12), while in step 2 we benefited from having 5 different possible solutions. We found two solutions: $C_x^{21}, D_x^{21} =$ 819b1392, 9f4d3233 and $C_x^{21}, D_x^{21} =$ aa9f6f2d, 3a6e613a.

**Putting both steps together.** With these solutions, we built two full-state collisions on Gimli-Hash using 8-round Gimli(21, 14). We start from $m_1, 0, 0, 0$, then after one round, we inject the values $A_x^{21}$, $B_x^{21}$, $C_x^{21}$, $D_x^{21}$ and $A_x'^{21}$, $B_x^{21}$, $C_x^{21}$, $D_x^{21}$ respectively in the rate; then we obtain two states that differ only on the $x$-coordinate of the third column (not the first, due to a big swap), and we inject two different blocks to cancel out this difference, obtaining the same state. The full state then collides, and we can append any message block that we want. An example is given in Table 8.6.

**Table 8.6:** An 8-round collision on Gimli-Hash.

| First message block | | | |
|---|---|---|---|
| dc84bf38 | 00000000 | 00000000 | 00000000 |
| dc84bf38 | 00000000 | 00000000 | 00000000 |
| Second message block | | | |
| bbdb41f3 | 4333192c | bc17e444 | 8a9d06c7 |
| 1b1da6e4 | 4333192c | bc17e444 | 8a9d06c7 |
| Third message block | | | |
| 00000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | afad801e | 00000000 |

**Extending the attack.** Remark that the first step can be extended to span any number of $SP^2$-boxes. However, each time we add two more rounds, there is one more branch coming from the $B, C, D$ states which has to match an expected value, so we multiply the complexity by a factor $2^{32}$. Since $t_e \ll 2^{32}$, we can do that twice before

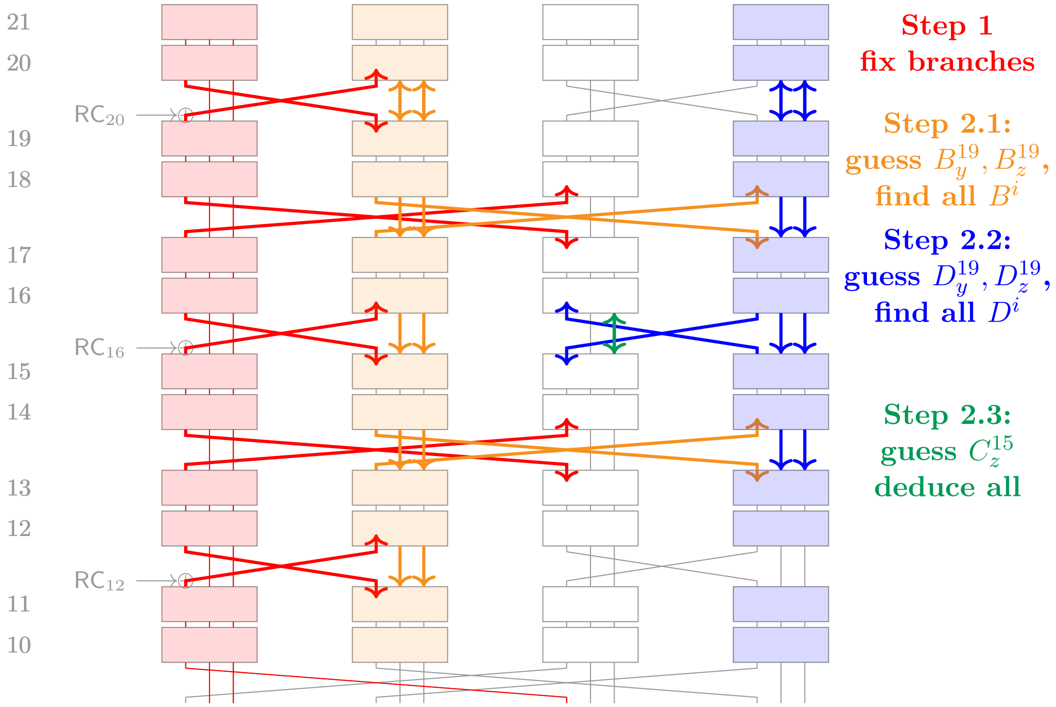


**Figure 8.13:** Semi-free start collision attack on 12 rounds of GIMLI.

meeting the bound $2^{128}$. Thus, a collision on 12-round GIMLI-HASH can be built in time $2^{96} \times 4 \times t_e$.

#### 8.2.4.3 Semi-free Start Collisions

Exploiting the same principles (the weak diffusion of GIMLI and the simplicity of solving the SP-Box equations), we can design semi-free start collision attacks reaching more rounds. This time, our goal is to obtain two input states $S, S'$ that differ only in the rate (in practice, only in $A_x$) and such that after applying a reduced-round GIMLI, the output states differ only in the rate (in practice, only in $A_x$ before the swap). The previous "first step" remains the same, with an extension to whichever number of rounds we are targeting. The "second step" is changed, because we can now choose completely the columns $B, C, D$, e.g., by starting from the middle instead of having to choose only the input rate.

Doing this allows us to reach 4 rounds more for the same cost as before, as outlined in Figure 8.13 and Algorithm 8.11. We can then append new rounds as before, reaching 16 rounds classically in time $2^{96} \times 10 \times t_e$.

**Another improvement using precomputations.** We are going to win a factor $2^{32}$ using $2^{64} \times t_e$ precomputations and a table of size $2^{64}$. This way, we can attack two more rounds. Indeed, once we have computed the first step, the two branches $C_x^{17}$ and

**Algorithm 8.11** 12-round semi-free start collision attack (see Figure 8.13).

**Input:** an initial $A$ (can be given)
**Output:** $A_x, A'_x, B, C, D$ such that after GIMLI(21, 10), only the rate differs.
**Complexity:** $2^{32} \times 10 \times t_e$ evaluations of GIMLI

As before, we don't write the last swapping step.

**Step 1:** Same step as in Algorithm 8.10, extended to 12 rounds. It gives a total of 10 32-bit branches (input values) that are required, that are represented in **red** on Figure 8.13.

**Step 2:** we will start from the middle.

2.1 We take an arbitrary value for $B^{19}_{y,z}$. This guess enables to deduce all values of the column $B$, from $B^{21}$ to $B^{10}$, either by simply computing the SP-Box, or by solving Equation (8.12) (given two input branches $y, z$, given the output $x$, deduce the input $x$). From this, we deduce the value in all **branches that go from $B$ to $D$** in Figure 8.13, hence 4 branches.

2.2 We take an arbitrary value for $D^{19}_{y,z}$. Again, this enables to deduce the whole sequence of states from $D^{20}$ to $D^{10}$, either by computing the SP-Box when possible, or by finding the input $x$ value corresponding to a given output. We also obtain the values of **branches that are transmitted from $D$ to $C$.**

2.3 **We now guess $C^{15}_z$.** Given this, and $C^{15}_x$, and the output $A^{13}_x$ that must be met, we obtain the whole state by solving another simple SP-Box equation (which is not Equation (8.12)), but has a similar form).

1. Having deduced $C^{15}$, we have only $2^{-32}$ chances of obtaining the right $C^{17}_x$, so we have to repeat all of this $2^{32}$ times.

In total, we have to solve 5 SP-Box equations, $2^{32}$ times, in both steps, so the time complexity is $2^{32} \times 10 \times t_e$.

$A^{13}_x$ contain arbitrary fixed values. Then, when we try to find the right $C$, we could have a table that for all $C^{15}_y, C^{15}_z$, gives all input-output values for $C^{17}$ and $C^{14}$, and we could directly use this table to match the values $C^{15}_x$ and $D^{15}_x$ that come from $D$ (instead of having to make a guess of $C^{15}_z$.

Let us fix $C^{17}_x = A^{13}_x = 0$. Thus, we repeat step 1 in Algorithm 8.11 a total of $2^{64}$ times in order to have $C^{17}_x = A^{13}_x = 0$. Step 1 now costs $2^{96} \times t_e$.

The table that we precompute shall contain: for each $x', x''$, all values (on average 1) of $y', z'$ such that $SP^2(0, *, *) = x', y', z'$ and $SP^2(x'', y', z') = 0, *, *$.

Now, in Algorithm 8.11, for each guess of $B^{19}_{y,z}$, and for each guess of $D^{19}_{y,z}$, we can find the value of $C$ that matches all the fixed branches in time 1, using this table. Thus, we can repeat this $2^{96}$ times, extending the attack by 6 rounds.

- Step 1 costs $2 \times 2^{96} \times t_e$ (we solve only 2 equations most of the time, before aborting if the wanted "0" do not appear).
- The table costs $2^{64} \times t_e$, which is negligible
- Step 2 costs $2^{96} \times 5 \times t_e$, since it is the same as before, and we only need forwards computation of SP-Boxes to check if the full path is correct.

Note that we can get rid of the term $t_e$ if we use a memory of size $2^{96}$ to store the solutions of the SP-Box equations. In that case, the overall time complexity is slightly below $2^{96}$ evaluations of Gimli, since fewer SP-Boxes are evaluated in each step than in the full primitive.

### 8.2.5 Quantum Collision Attacks on Gimli

As was shown for the first time in [HS20], collision attacks may reach more rounds in the quantum setting than in the classical one. This will be the case here. Note that these attacks do not require any form of black-box query, since the hash function is public by nature, hence we stay in the Q1 setting. Intuitively, each step in these attacks is an independent exhaustive search on some well-defined search space. The generic quantum collision search complexity on Gimli-Hash is approximately $2^{256/3} = 2^{85.3}$ computations of Gimli, using the BHT algorithm (Algorithm 2.10 in Section 2.3). In Section 8.2.4, we limited ourselves to attacks of complexity approximately $2^{96} \times t_e$, because the generic complexity lies at $2^{128}$ Gimli evaluations. But classical attacks with a time as high as $2^{128} \times t_e$, if they benefit from a square-root speedup, will quantumly fall at $2^{64} \times t_e$ which is below generic.

In this section, we will rely on two generic tools to turn a classical attack algorithm into a quantum one: Amplitude Amplification (Section 2.1) to replace the exhaustive search steps and generic quantum embeddings of classical algorithms (Section 1.2.3). Indeed, we know that the SP-Box equations that we defined can be solved in a time complexity of about $t_e = 2^{10}$ evaluations of Gimli. Consequently, there exists quantum algorithms to solve these equations with a quantum time complexity of about $t_{qe} \simeq t_e$ quantum evaluations of Gimli. This allows to compare our complexities with the generic ones without going into the details of a quantum circuit for Gimli.

**Example.** We consider the 8-round collision attack of Algorithm 8.10. Both its steps run in classical time $2^{32} \times 4 \times t_e$ by running $2^{32}$ iterates of a randomized algorithm of time complexity $4 \times t_e$:

- In the first step, we try all $A_x^{21}$ to find the one such that, after solving a sequence of SP-Box equations, a 32-bit condition is met: the first equation finds $A_x'^{21}$ such that there is a collision in $x$ after two SP-Boxes, the second equation finds $A_x^{19}$ such that there is a collision in $x$ after two SP-Boxes, *etc.*, and the final 32-bit condition is that $A_z'^{13}$ and $A_z^{13}$ must collide.
- In the second step, we try all $C_x^{21}$ to find the one that satisfies a 32-bit condition.

Using Amplitude Amplification, we obtain a corresponding quantum algorithm with time complexity *approximately* $2^{16} \times 4 \times t_e$. This approximation comes from different factors:

- a small constant factor $\frac{\pi}{2}$ which is inherent to quantum search.
- the trade-offs between time and space in the detailed implementations of the primitive and its components. In fact, Gimli is not so difficult to implement using Clifford+T quantum gates, compared with the AES (Chapter 9) that requires a custom quantum circuit for its S-Box.

We are mainly interested in the security margin, and these approximations will be sufficient for us to determine whether a given algorithm runs faster or slower than the corresponding quantum generic attack. Thus, we will write that the quantum 8-round attack on Gimli-Hash runs in time $\simeq 2^{16} \times 4 \times t_e$.

**Collisions.** By adding another 32-bit condition in the classical 12-round collision attack, we obtain a procedure which runs classically in time $4 \times 2^{128} \times t_e$, which is too high. However, using Amplitude Amplification, we obtain a procedure that runs in quantum time $\simeq 4 \times 2^{64} \times t_e$ and reaches 14 rounds.

**Semi-free start collisions.** We can extend the 18-round semi-free start collision attack in the same way. Building the table will still cost a time $2^{64}$. This table must be stored in a classical memory with quantum random access. The first step goes from $2 \times 2^{96} \times t_e$ classically to approximately $2 \times 2^{48} \times t_e$ quantumly. The second step does as well. Thus, adding a 32-bit condition enables us to attack 20 rounds in quantum time $2^{64} \times 4 \times t_e$.

### 8.2.6 Discussion

The results presented in this section exploit the relatively slow diffusion between the columns of the Gimli state. Indeed, swaps occur only once every two rounds. Thus, the Gimli SP-Box is always applied twice, except at the first and last rounds, and the permutation can be viewed as an SPN with only 12 rounds, with very simple linear layers.

The distinguisher on Gimli also draws on the fact that the constant addition occurs only every four rounds and only on a single column. In short, a 32-bit condition is enough for two columns to remain symmetric after four rounds of Gimli. Starting from the middle is a clear advantage. The slow diffusion helps in guessing the middle state piece by piece.

The collision attacks on Gimli combine this with the simplicity of the double SP-Box. Solving equations between inputs and outputs of a double SP-Box is a trivial matter. Thus, it becomes easier to set some state values in order to satisfy a condition many rounds later.

The strength of this distinguisher and its low complexity shows that Gimli cannot be used in a hermetic sponge strategy. Although the collision attacks do not target the full mode, it seems worth to tweak the permutation in order to increase its security margin. The designers of Gimli mention in the submission package [Ber+19] that they had studied other linear layers than the swaps, in particular using an MDS transformation or the linear layer of SPARX [Din+16]. They state:

> *We found some advantages in Gimli-MDS and Gimli-SPARX in proving security against various types of attacks, but it is not clear that these advantages outweigh the costs.*

The attacks of this section do not apply to the other variants, and they seem *a priori* much stronger to our internal symmetry and guess-and-determine approach. This makes their advantage clearer, although a renewed analysis would be needed.

**Further developments.** These results may have appeared surprising, as Gimli was a permutation proposed much earlier than the LWC process (in 2017), and it had already attracted a reasonable amount of third-party cryptanalysis.

In August 2020, our paper [Fló+20] was accepted at ASIACRYPT 2020. It later received a “best paper” award. The full version contains more results than the contents of this section, notably an analysis of linear and linear-differential distinguishers on the permutation, which had not been done before.

In September 2020, between the second and third round of the process, the NIST proposed to the submitters of second-round algorithms to publish status updates. In their document, the designers of Gimli [Ber+20] briefly dismissed the notion of a distinguisher for a permutation as follows:

> *The state-of-the-art permutation distinguisher is a generic distinguisher of the form “is $P(0) = x$?” that succeeds against every permutation in time 1, when $x$ is chosen appropriately.*

This statement obviously misinterprets the notion of a “distinguisher” in this context. It has also the undesirable effect of dismissing all security analyses of permutations and block ciphers in the known-key model in general, which represents a long-standing line of work.

# Chapter 9
# Quantum Security Analysis of AES

As Grover's algorithm speeds up exhaustive key search by a square root factor, a 128-bit security level in the quantum world seems to require keys of at least 256 bits. Fortunately, the current block cipher standard [AES] allows such key lengths, and its 256-bit version is recommended for post-quantum security. This is stated in the report on quantum computing from the National Academy Sciences, Engineering, and Medicine [Nat18]:

> *Even if a computer existed that could run Grover's algorithm to attack AES-GCM, the solution is quite simple: increase the key size of AES-GCM from 128-bit to 256-bit keys.*

At the same time, some desired levels of security in the NIST call for post-quantum public-key cryptosystems [NIS16] are defined relatively to the hardness of exhaustive key search on AES variants. Thus, the optimization of quantum circuits for AES key search is motivated not only by determining the post-quantum security of AES, but also by reaching a better precision in the NIST post-quantum cost metrics.

AES is one of the most used and studied block ciphers classically (if not the most). It has been the subject of a continued cryptanalysis effort, allowing to determine and refine its security margin and to have a more than justified confidence in it. However, prior to our results from [BNS19b], no such study had been performed against quantum adversaries, Q1 or Q2 alike. The results of [BNS19b], that we will detail in this chapter, give a first overview of the post-quantum security of AES. They suggest that AES seems as safe in the quantum world as in the classical one. As a side effect, our results also improved the memory complexity of some of the best known classical attacks on AES.

## Contents

## 9.1 Introduction

We study reduced-round attacks on AES variants, in the secret-key model, in a chosen-plaintext or chosen-ciphertext scenario. There has been many other results on *related-key* attacks against AES, that are discussed e.g. in [DR12], but the secret-key setting remains the most significant.

**Problem 9.1** (AES secret-key recovery)**.** *Given access to an oracle, quantum or classical, for a block cipher $E_k$ that implements a reduced-round variant of AES, and possibly to its inverse, find $k$.*

Classically, a cipher is broken if a procedure more efficient than exhaustive search exists. Quantumly, exhaustive search benefits from a quadratic speedup, but some algorithms perform worse (for example, collision search). Thus, the quantum security margin may be higher. On the other hand, higher than quadratic speedups have occurred in the Q2 setting (Chapter 4) and nothing prevents AES from a devastating, yet unknown, quantum attack. This is also stated in [Nat18]:

> *[...] it is possible that there is some currently unknown clever quantum attack on AES-GCM that is far more efficient than Grover's algorithm.*

The same set of principles that gave confidence in the classical security of AES apply to its post-quantum security. This algorithm should have a detailed security evaluation, evolving through time, advancing hand in hand with our understanding of quantum algorithms.

### 9.1.1 Summary

Previous security analyses of AES in the quantum world targeted only exhaustive key search with Grover's algorithm, and whether it put AES variants at risk or not. A general discussion was given in [Rao+17], an analysis of Grover combined with side channel attacks in [Mar+17].

We start in Section 9.3 by discussing the attacks that did not seem to obtain a good speedup. Next, we describe quantum Square attacks on 6-round AES-128, 7-round AES-192 and 7-round AES-256 (Section 9.4), and a quantum Demirci-Selçuk Meet-in-the-Middle attack on 8-round AES-256 (Section 9.5), which is the best known quantum key-recovery attack against AES-256 in number of rounds.

We write these attacks as **Sample** programs, with the notations of Section 2.2. We will consider Q1 or Q2 adversaries, but our best attacks can be placed in the Q1 model. We will also consider qRAM, but only the quantum Square attacks of Section 9.4.2 require QRAQM. Our 8-round attack can rely exclusively on classical memory thanks to the mapping from QRACM to CSAM, by the emulation unitary of Lemma 1.2. Our results are summarized in Table 9.6 at the end of this chapter.

Classically, DS-MITM attacks against AES-256 (together with impossible differential attacks) reach up to 9 rounds, and as we will see, designing such a quantum attack is a nontrivial process. In Section 9.6, we show that some of our ideas can be used in the classical setting as well, providing notably the best known attacks so far on 9-round AES-256.

### 9.1.2 Description of AES

Designed by Daemen and Rijmen, the block cipher [AES] is the current encryption standard, chosen by an open competition organized by the NIST in 2000. It is a Substitution-Permutation Network alternating between linear layers, non-linear layers and round key additions. It has three different key sizes: 128, 192 and 256, with different key schedules, and respectively 10, 12 and 14 rounds.

| | | | |
|---|---|---|---|
| 0 | 4 | 8 | 12 |
| 1 | 5 | 9 | 13 |
| 2 | 6 | 10 | 14 |
| 3 | 7 | 11 | 15 |

**Figure 9.1:** AES byte ordering.

**AES state and round function.** The cipher encrypts blocks of 128 bits, split into 16 bytes organized in a square (Figure 9.1). The round function has four operations: first, AddRoundKey (ARK), which XORs the round key (and round constant) with the current state. Second, SubBytes (SB), which applies the AES S-Box to each byte. The

S-Box is a nonlinear function over $\mathbb{F}_{2^8}$, which combines an affine transformation of the byte with the multiplicative inverse in $\mathbb{F}_{2^8}$. Third, ShiftRows (SR), which shifts the $i$-th row by $i$ bytes left and finally, MixColumns (MC), which multiplies each column by the AES MDS matrix. It is a linear transformation over $(\mathbb{F}_{2^8})^4$:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} .$$

It has branching number 5, meaning that among $b_0, b_1, b_2, b_3$, $a_0, a_1, a_2, a_3$ defined as above, no more than 3 of the bytes can be 0 at the same time. The last round has no MixColumns and applies a final key addition (there are $r + 1$ subkeys for $r$ rounds). In this chapter, we consider reduced-round versions with a MixColumns in their last round.

**Notations.** We write $x_i, y_i, z_i, w_i$ the successive AES states (see Figure 9.7) after applying the 4 round operations ARK, SB, SR and MC at round $i$. The S-Box is denoted $S$. We note $k_i$ the key at round $i$ and $u_i = MC^{-1}(k_i)$ the "equivalent" round keys, such that adding $k_i$ after the MC operation is equivalent to adding $u_i$ *before* this step. We note $x[0, 1, \ldots]$ when selecting bytes from these states. We use the usual AES byte numbering of Figure 9.1. When we consider a pair, the states are denoted $x_i, x'_i$. We also note $\Delta x_i = x_i \oplus x'_i$. Furthermore, equalities such as $x_4[1, 2, 3] = x'_4[1, 2, 3]$ are to be understood byte per byte.

**Key-schedule relations.** While the round function has a strong design (it ensures total diffusion after two rounds), the key schedule has been held as a weaker point (as in [DKS15]). An AES with $r$ rounds needs $r + 1$ round keys. Key-schedule relations have often been used to speed up cryptanalysis of reduced-round AES-192 and 256. Later on in this chapter, we will use some relations derived from the AES-256 key schedule, which is depicted in Figure 9.2, where the symbol « denotes shifting upwards the bytes of a column. It maps a pair of round keys $k_{2i}, k_{2i+1}$ to $k_{2i+2}, k_{2i+3}$.

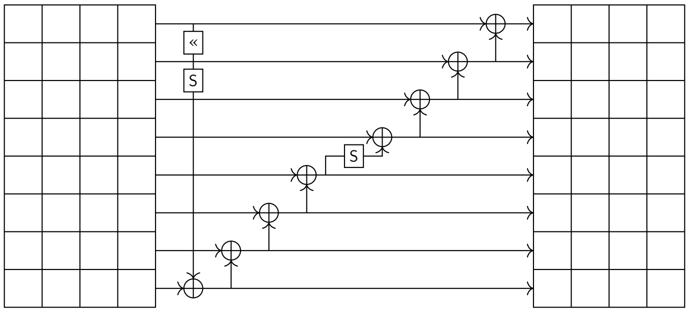


**Figure 9.2:** AES-256 key schedule [Jea16].

**Lemma 9.1** (Key-schedule Relations in AES-256)**.** *Let $k_0, \dots k_8$ be the 8-round expansion of the key-schedule of AES-256. The following relations hold:*

$$\begin{cases} k_0[10] & = k_4[2] \oplus k_4[10] \\ k_0[15] & = k_4[7] \oplus k_4[15] \\ k_5[3] & = k_1[3] \oplus S(k_4[15]) \oplus S(k_4[11] \oplus k_4[15]) \\ k_2[4\text{–}7] & = k_4[0\text{–}3] \oplus k_4[4\text{–}7] \end{cases} .$$

*Proof.* The AES-256 key schedule is represented in Figure 9.2. We have:

$$k_2[0] = k_0[0] \oplus S(k_1[13]); k_2[4] = k_0[0] \oplus k_0[4] \oplus S(k_1[13]); \dots$$

and in particular, $k_0[10] = k_2[10] \oplus k_2[6]$ and $k_0[15] = k_2[11] + k_2[15]$.

Besides, the same relations hold between $k_4$ and $k_2$, so: $k_2[10] = k_4[10] \oplus k_4[6]$, $k_2[6] = k_4[6] \oplus k_4[2]$, $k_2[11] = k_4[7] \oplus k_4[11]$ and $k_2[15] = k_4[11] \oplus k_4[15]$, so:

$$k_0[10] = k_4[2] \oplus k_4[10] \text{ and } k_0[15] = k_4[7] \oplus k_4[15] \ .$$

When $i$ is odd, the first column of $k_i$ is equal to the first column of $k_{i-2}$, to which we add the last column of $k_{i-1}$, going through S-Boxes. Hence:

$$k_5[3] = k_3[3] \oplus S(k_4[15]) \text{ and } k_3[3] = k_1[3] \oplus S(k_2[15]),$$

$$\text{so: } k_5[3] = k_1[3] \oplus S(k_4[15]) \oplus S(k_4[11] \oplus k_4[15]) \ . \qquad \square$$

## 9.2 Exhaustive Search

In order to compare it with our attacks, we must first focus on the exhaustive search of the key. The first quantum resource estimates were done in [Gra+16] and they have been improved afterwards in [Alm+18; LPS19], with the best estimates to date obtained in [Jaq+20]. The estimates used in [BNS19b] were those of [Gra+16], but we will use [Jaq+20] here, for a meaningful comparison.

### 9.2.1 Full-Round AES Key Search

Quantum exhaustive search requires one or two plaintext-ciphertext pairs $(p_i, c_i)$ depending on the key length $\kappa$. Then it applies a quantum search QSearch to find a key $k$ such that:

$$\forall 1 \leqslant i \leqslant r, \mathsf{AES}_k(p_i) = c_i \ .$$

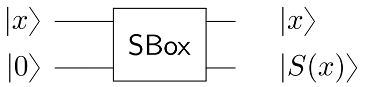
There is always a solution, but there may be false positives. By using two pairs with AES-128, two with AES-192 and three with AES-256, we obtain a negligible probability that a false positive exists (which is important, as in this case, the procedure may fail).

**Quantum circuit for the S-Box.** Next, we need a quantum circuit that computes $|k\rangle |0\rangle \mapsto |k\rangle |\mathsf{AES}_k(p)\rangle$ for a given $p$.

First of all, notice that the linear layer is easy to implement. The ShiftRows amounts only to a renumbering of the qubits, so it can be done without any quantum computation. The MixColumns step requires only some CNOTs that can be computed in place, without any involvement of ancilla qubits. The evaluation of the S-Box, which is the only nonlinear component of AES, happens to be the crucial operator in the quantum circuit. Indeed, classically, the S-Box can be realized with a lookup table, but the quantum equivalent would require QRACM storage. Thus, quantum circuits for the AES depend on efficient quantum circuits for computing the S-Box.

*Remark* 9.1. Since our attacks use the components of AES in the same proportions, the S-Box remains also the most costly component in all our procedures.

In Table 9.1, we give the counts from [Jaq+20]. The S-Box circuits in [LPS19] and [Jaq+20] are respectively based on the classical circuits of [BP10] and [BP12]. They are *out of place*, meaning that the output is computed in an additional register, as shown in Circuit 9.1.



**Quantum Circuit 9.1:** Out-of-place circuit for the S-Box.

**Table 9.1:** Resource estimates for the AES S-Box quantum circuit, taken from [Jaq+20], Table 1.

| Reference | CNOT | 1-qubit Clifford | T-gates | Measure-ments | T-depth | Full depth | Qubits |
|---|---|---|---|---|---|---|---|
| [Gra+16] | 8683 | 1028 | 3584 | 0 | 217 | 1692 | 44 |
| [LPS19] | 818 | 264 | 164 | 41 | 35 | 497 | 41 |
| [Jaq+20] | 654 | 184 | 136 | 34 | 6 | 101 | 137 |

**Quantum circuit for AES.** A rapid comparison between the cost of the S-Box and the rest (MixColumns, but also the operator $O_0$ in the Grover iterate) suggests to take the S-Box as the main cost metric, instead of the Clifford+T gate set (Section 1.2). This makes the analysis simpler. Counts for complete AES circuits are given in Table 9.2. The number of S-Boxes used stems from the rounds and from the key-expansion together. For example, in AES-128, there are 10 SubBytes operations and 10 round keys to compute, each requiring 4 S-Boxes. As each S-Box will be uncomputed, the total number of S-Box circuits is: $2 \times (10 \times 16 + 10 \times 4) = 400$. Finally, as the S-Boxes are out of place, the computation of each round leaves behind the state at the previous round, and adds roughly 128 ancilla qubits.

**Table 9.2:** Number of S-Box circuits used in the quantum circuits for AES in [Jaq+20] (see Table 3 in [Jaq+20]). Both forwards and backwards computations are counted.

| Variant | Number of S-Boxes | Number of qubits |
|---|---|---|
| AES-128 | 400 | 1785 |
| AES-192 | 448 | 2105 |
| AES-256 | 552 | 2425 |

**Table 9.3:** Estimates for a full-round key-recovery on AES variants, for an overwhelming success probability (see Table 10 in [Jaq+20]).

| Variant | Number of plaintexts | Gate counts |  |  | S-Boxes |
|---|---|---|---|---|---|
|  |  | Clifford | T-gates | Measures |  |
| AES-128 | 2 | $2^{83.2}$ | $2^{80.4}$ | $2^{78.4}$ | $\lfloor \frac{\pi}{4} 2^{64} \rfloor \times 800 = 2^{73.30}$ |
| AES-192 | 2 | $2^{115.3}$ | $2^{112.6}$ | $2^{110.6}$ | $\lfloor \frac{\pi}{4} 2^{96} \rfloor \times 896 = 2^{105.46}$ |
| AES-256 | 3 | $2^{148.2}$ | $2^{145.4}$ | $2^{143.4}$ | $\lfloor \frac{\pi}{4} 2^{128} \rfloor \times 1656 = 2^{138.34}$ |

Note that in the Grover oracle, we need to compute AES, to check equality with the target ciphertext, and to uncompute AES. This leaves the counts in Table 9.2 unchanged, as the uncomputation of the rounds can be deferred. Thus, Grover oracles for AES-128, AES-192 and -256 require respectively 400, 896 and 1656 S-Boxes. Multiplying by the number of Grover iterates $\lfloor \frac{\pi}{4} 2^{\kappa/2} \rfloor$ where $\kappa$ is the key length, one obtains the counts of Table 9.3, with optimized gate counts of [Jaq+20] and approximate counts in number of S-Boxes.

*Remark* 9.2 (On improved exhaustive search). In the classical setting, there exists procedures that accelerate the exhaustive search, e.g., using bicliques [BKR11]. This is currently not the case in the quantum setting. It seems for now that by trying to translate these procedures into quantum searches, one would lose more than there is to gain, due to uncomputation factors (early-abort techniques require a high level of nesting).

## 9.2.2 Resource Estimates for Reduced-round AES

As our attacks target reduced-round variants of AES, we must adapt the key-recovery accordingly. This is summarized in Table 9.4.

The most precise estimation that we shall need is for 8-round AES-256.

**Lemma 9.2** (Grover Search on 8-round AES-256)**.** *Using Grover search and three classical queries to a secret-key 8-round AES-256 oracle, the key can be recovered in approximately* $\lfloor \frac{\pi}{4} 2^{128} \rfloor \times 160 \times 6 = 2^{137.56}$ *reversible S-Boxes, using at most 1500 qubits.*

*Proof.* We use three plaintexts to have an overwhelming probability of success (two plaintexts would give $\frac{1}{e}$). Forward computations of 8-round AES-256 require 160 S-Boxes,

**Table 9.4:** Cost benchmarks for quantum reversible AES components. The S-Boxes have 8-bit inputs and 8-bit outputs. The number of qubits required can be approximated by 8 times the number of (forward) S-Boxes, plus 137 ancillas.

| Component | Number of S-Boxes |
|---|---|
| S-Box | 1 |
| 128-bit key-schedule (10 rounds/10 keys) | 40 |
| 192-bit key schedule (12 rounds/8 keys) | 32 |
| 256-bit key schedule (14 rounds/7 keys) | 56 |
| 6-round AES (no key schedule) | $6 \times 16 = 96$ |
| 7-round AES (no key schedule) | $7 \times 16 = 112$ |
| 8-round AES (no key schedule) | $8 \times 16 = 128$ |
| 6-round AES-128 (with key schedule) | 120 |
| 7-round AES-192 (with key schedule) | 132 |
| 7-round AES-256 (with key schedule) | 140 |
| 8-round AES-256 (with key schedule) | $8 \times 16 + 8 \times 4 = 160$ |

and they are uncomputed afterwards. The number of qubits comes from the number of rounds computed and the key-schedule, which are below a full AES-256. □

## 9.3 Discussion on Quantum Attacks

In order to provide a starting point in the secret-key setting, the most reasonable approach seems to start from the best classical attacks against reduced-round AES and try to *quantize* them, in a way similar to [Kap+16b] for (generic) linear and differential attacks against block ciphers. Table 9.5 provides a summary of the best known classical attacks on AES in the secret key model.

The results in this chapter follow the observation of [Kap+16b], that the best quantum attack is not always as simple as a quantum version of the best classical one.

In this section, we give a short discussion on attacks based on Impossible Differentials and Simon's algorithm. More detailed explanations for Square and DS-meet-in-the-middle attack will follow in Section 9.4 and Section 9.5 respectively, as these are the attack patterns that we managed to quantize.

### 9.3.1 Impossible Differential Attacks

Impossible differential attacks rely on differential patterns that cannot occur in the cipher. Such a pattern provides a distinguisher for several middle rounds, which is extended a few rounds backwards and forwards, involving some bits of the secret key. The attacker then guesses these bits. For a good guess, the middle pattern cannot occur by definition, and for bad guesses, it may occur. Thus, the attacker sieves the key space

**Table 9.5:** Summary of classical cryptanalysis on AES in the single secret key setting. Time is given in equivalent trial encryptions and memory in 128-bit blocks. We omit generic attacks, including the ones that perform an intelligent exhaustive search on the key like [BKR11]. We include the new significant trade-offs that we introduce in Section 9.6.

| Version | Rounds | Data | Time | Memory | Technique | Reference |
|---|---|---|---|---|---|---|
| Any | 6 | $2^{32}$ | $2^{44}$ | $2^{32}$ | Square | [Fer+00] |
| | 7 | $2^{113}$ | $2^{113} + 2^{80}$ | $2^{80}$ | DS MITM | [DFJ13] |
| | 7 | $2^{105}$ | $2^{105} + 2^{99}$ | $2^{90}$ | DS MITM | [DFJ13] |
| | 7 | $2^{97}$ | $2^{99}$ | $2^{98}$ | DS MITM | [DFJ13] |
| | 7 | $2^{113}$ | $2^{113} + 2^{84}$ | $2^{74}$ | DS MITM | Section 9.6 |
| | 7 | $2^{105}$ | $2^{105} + 2^{95}$ | $2^{81}$ | DS MITM | Section 9.6 |
| | 7 | $2^{113.1}$ | $2^{113.1} + 2^{105.1}$ | $2^{74.1}$ | ID | [Bou+18] |
| | 7 | $2^{105}$ | $2^{106.88}$ | $2^{74}$ | ID | [Bou+18] |
| 192 | 7 | $2^{34}$ | $2^{155}$ | $2^{32}$ | Square | [Fer+00] |
| | 7 | $2^{99}$ | $2^{99}$ | $2^{96}$ | DS MITM | [DFJ13] |
| | 8 | $2^{113}$ | $2^{172}$ | $2^{82}$ | DS MITM | [DFJ13] |
| | 8 | $2^{107}$ | $2^{172}$ | $2^{96}$ | DS MITM | [DFJ13] |
| 256 | 7 | $2^{99}$ | $2^{98}$ | $2^{96}$ | DS MITM | [DFJ13] |
| | 7 | $2^{34}$ | $2^{172}$ | $2^{32}$ | Square | [Fer+00] |
| | 8 | $2^{113}$ | $2^{196}$ | $2^{82}$ | DS MITM | [DFJ13] |
| | 8 | $2^{107}$ | $2^{196}$ | $2^{96}$ | DS MITM | [DFJ13] |
| | 9 | $2^{113+x}$ | $2^{210-x} + 2^{196+x}$ | $2^{210-x}$ | DS MITM | [DFJ13] |
| | 9 | $2^{113+x/2}$ | $2^{210-x} + 2^{194+x}$ | $2^{194+x}$ | DS MITM | Section 9.6 |

by removing wrong guesses, allowing to significantly reduce the amount of possible key candidates. This strategy is generalized and improved against SPNs in [Bou+18].

The best impossible differential attack on AES-128 [Bou+18] targets 7 rounds (Table 9.5), and provides a comparable trade-off, with some advantages, to the best DS-MITM attacks.

Impossible differential attacks run in two steps. First, the attacker computes pairs $(P, P')$, $(C, C')$ of plaintexts and ciphertexts satisfying the input-output conditions of the differential path: $P, P'$ and $C, C'$ must partially collide. These pairs might lead to the impossible differential in the middle rounds. But the quantum speedup of this step tends to be less than a square root, even in the QRAQM model and with Q2 queries. Second, the attacker discards the wrong keys associated to each pair in the most efficient way possible, thanks to the early abort technique. The good key will be among the ones that have not been discarded. This sieving step can be accelerated, although all the classical techniques (like state-test techniques or multiple differentials as in [BNS14]) need or would need specific adaptations. In [Dav19], David studied quantum impossible differential attacks and concluded that they did not seem to give an advantage in the case of AES.

### 9.3.2 Attacks based on Simon's Algorithm

Attacks in the Q2 model that use Simon's algorithm and its offline variant (detailed in Section 4.1 and Chapter 7), for example quantum slide attacks, crucially rely on a strong algebraic structure. Thanks to its key-schedule, AES seems immune to them, even on reduced-round variants. Thus, to date, there does not exist an attack with an exponential acceleration in the Q2 setting.

## 9.4 Quantum Square Attacks on the AES

The Square or integral attack was proposed in [DKR97] against the block cipher Square. It was studied in the original specification document AES [DR99] targeting 6 rounds, and extended later to 7 rounds when considering AES-192 and 256 [Fer+00]. It relies on a distinguisher for 3 rounds of AES, that is extended into a key-recovery.

In this section, we design quantum variants of the Square attack. We remain in the Q1 setting and in the QRAQM model. For quantum exhaustive search on 6 and 7-round AES-128, we give estimates using Table 9.4.

### 9.4.1 The Classical Square Attack

The Square attack relies on a distinguisher on 3 rounds of AES, depicted in Figure 9.3.

**Definition 9.1** ($\delta$-set)**.** A $\delta$-set in byte 0 is a set of $2^8$ AES states that take all values on byte 0 but are constant on the other bytes.

**Proposition 9.1** (Square distinguisher [DR99])**.** *Let $P_0, \dots, P_{255}$ be a $\delta$-set. Let $C_0$, …, $C_{255}$ be the set of encryptions of $P_j$ through 3 rounds of AES, regardless of the round keys. Let $0 \leqslant i \leqslant 15$ be any byte. Then $\bigoplus_{0 \leqslant j \leqslant 255} C_j[i] = 0$. The byte is said* balanced.

*Proof.* The S-Box is a bijective function, hence after the first SubBytes layer, the $\delta$-set remains a $\delta$-set. Besides, each byte in the active column at the end of the first round takes 256 values. This remains true after the second round and the third round, again by bijectivity of the S-Box. The sum in any byte position before the last MixColumns is zero. This remains zero after MixColumns by linearity. Notice that Proposition 9.1 remains valid for any input byte, by symmetry. □

Proposition 9.1 yields a distinguisher since, for a random function, the probability that a selected byte is balanced is $\frac{1}{2^8}$.

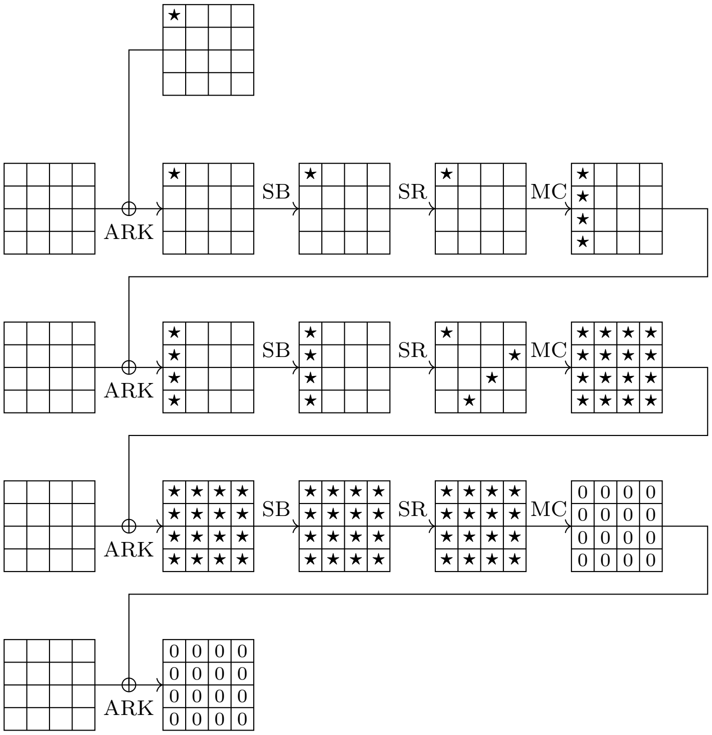


**Figure 9.3:** Integral distinguisher on 3 rounds of AES. A byte marked by 0 is balanced.

**Square Attack on 6-round AES.** The resulting attack on 6 rounds was designed in [DKR97] and improved in [Fer+00]. It is described in Figure 9.4. We append one round before and two rounds after the distinguisher. We encrypt a set of $2^{32}$ plaintexts that take all values in the main diagonal but are constant on the other bytes. Then, regardless of the first round key, they give $2^{24}$ $\delta$-sets at the end of the first round. When given to the three inner rounds, this gives balanced bytes as well. Using this, we don't have to guess the first round key.

### 9.4.2 Q1 Square Attack on 6-round AES

First, we focus on the initial 6-round Square attack from [DKR97]. In Algorithm 9.1, we rewrite it as a classical **Sample**, with a simple translation as a quantum algorithm.

Using the square property as a 3-round distinguisher, each structure gives a one-byte condition. Although 5 would be enough, we use 8 structures in order to ensure that only the right key guess passes the test, with overwhelming probability. Hence the number of Grover iterations is exactly known. As each partial decryption requires 5 S-Boxes, the classical time complexity in S-Boxes is:

$$2^{40} \times 2^{32} \times 8 \times 5 \leqslant 2^{78}$$

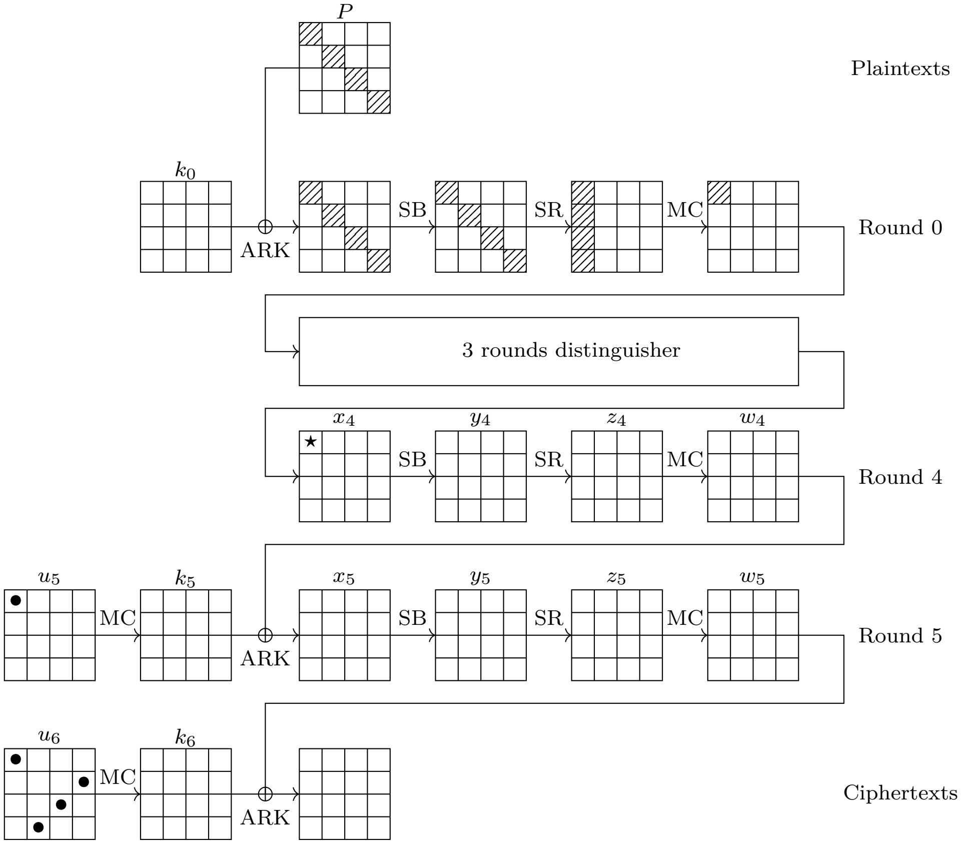


**Figure 9.4:** Square attack on 6-round AES.

**Algorithm 9.1** Quantum square attack on 6-round AES.

```
Input: 8 structures of $2^{32}$ classical chosen-plaintext queries such that the main diagonal $x_0[0, 5, 10, 15]$ takes all values
Output: the key bytes $u_5[0], u_6[0, 7, 10, 13]$
Sample $u_5[0], u_6[0, 7, 10, 13]$ such that
                                                        ▻ One solution among $2^{40}$
    for all Structures do
        Partially decrypt the $2^{32}$ ciphertexts $C_j$ through the two last rounds
        Compute the XOR of all values in $x_4[0]$
        if it is non-zero, abort
EndSample
return $u_5[0], u_6[0, 7, 10, 13]$
```

and the quantum time complexity is: $\lceil \frac{\pi}{4} 2^{20} \rceil \times (2^{32} \times 2 \times 8 \times 5) \leqslant 2^{58}$. The input data is only classical and no superposition queries are required.

**Partial Sums Technique.** The partial sums technique from [Fer+00] already makes the classical time complexity decrease to $2^{52}$ S-Boxes ($2^{44}$ encryptions). In order to adapt this technique, we will rely on QRAQM.

For readability, we omit the last MixColumns and Shiftrows: the five guessed key bytes are now $u_5[0]$ and $k_6[0,1,2,3]$. We want to find values of those bytes such that the sum of $x_4[0]$ on all $2^{32}$ ciphertexts $C_i$ is zero:

$$\bigoplus_i x_4[0]^i = \bigoplus_i S^{-1}(y_4[0]^i) \enspace .$$

We write $y_4[0]$ as a combination of $x_5[0,\dots,3]$:

$$\bigoplus_i x_4[0]^i = \bigoplus_i S^{-1} \left( a_0 x_5[0]^i \oplus a_1 x_5[1]^i \oplus a_2 x_5[2]^i \oplus a_3 x_5[3]^i \oplus u_5[0] \right)$$

for some known coefficients $a_0, a_1, a_2, a_3$. In turn this rewrites:

$$\bigoplus_i S^{-1} \Bigg( a_0 S^{-1}(C_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(C_i[1] \oplus k_6[1]) \\ \oplus a_2 S^{-1}(C_i[2] \oplus k_6[2]) \oplus a_3 S^{-1}(C_i[3] \oplus k_6[3]) \oplus u_5[0] \Bigg) \enspace . \tag{9.1}$$

The presentation from [Fer+00] constructs successive tables containing *partial sums*. The final table contains, for each 5-byte key guess ($2^{48}$), the value of the whole sum. In Algorithm 9.2, we rewrite this procedure as a composition of **Sample**s. Having 8 structures ensures that only the right key guess passes at each step with overwhelming probability. For each structure, we create successive tables $T_1$, $T_2$, $T_3$ such that:

1. $T_1$ counts, for each triple of bytes, how many times

   $$a_0 S^{-1}(C_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(C_i[1] \oplus k_6[1]), C_i[2], C_i[3]$$

   takes this value when $C_i$ runs over all ciphertexts in the structure.

2. $T_2$ counts, for each pair of bytes, how many times

   $$a_0 S^{-1}(C_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(C_i[1] \oplus k_6[1]) \oplus a_2 S^{-1}(C_i[2] \oplus k_6[2]), C_i[3]$$

   takes this value when $C_i$ runs over all ciphertexts in the structure.

3. $T_3$ counts, for each byte value, how many times the following quantity takes this value when $C_i$ runs over all ciphertexts in the structure:

   $$a_0 S^{-1}(C_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(C_i[1] \oplus k_6[1]) \\ \oplus a_2 S^{-1}(C_i[2] \oplus k_6[2]) \oplus a_3 S^{-1}(C_i[3] \oplus k_6[3]) \oplus u_5[0] \enspace .$$

**Algorithm 9.2** Square attack on 6-round AES with the partial sums technique.

```
Input: 8 structures of $2^{32}$ classical chosen-plaintext queries such that the main diagonal $x_0[0,5,10,15]$ takes all values
Output: the key bytes $u_5[0]$, $k_6[0,1,2,3]$
Sample $k_6[0], k_6[1]$ such that                                  ▻ One solution among $2^{16}$
    for all Input structures $C$ do
        Allocate a table $T_1$ of $2^{24}$ one-byte entries, labeled by three bytes, initially 0
        for all Ciphertext $C_i \in C$ do
            Compute:
            $b_1, b_2, b_3 \leftarrow a_0 S^{-1}(C_i[0] \oplus k_6[0]) \oplus a_1 S^{-1}(C_i[1] \oplus k_6[1]), C_i[2], C_i[3]$
            $T_1[b_1, b_2, b_3] \leftarrow T_1[b_1, b_2, b_3] + 1$
    Sample $k_6[2]$ such that                                      ▻ One solution among $2^8$
        for all Input structures $C$ do
            Allocate a table $T_2$ of $2^{16}$ one-byte entries, labeled by two bytes, initially 0
            for all Address $b_1, b_2, b_3$ in $T_1$ do
                Compute $b_4, b_5 \leftarrow b_1 \oplus a_2 S^{-1}(b_2 \oplus k_6[2]), b_3$
                $T_2[b_4, b_5] \leftarrow T_2[b_4, b_5] + 1$
        Sample $k_6[3]$ such that                                  ▻ One solution among $2^8$
            for all Input structures $C$ do
                Allocate $T_3$ of $2^8$ one-byte entries, initially 0
                for all Address $b_4, b_5$ in $T_2$ do
                    Compute $b_6 \leftarrow b_4 \oplus a_3 S^{-1}(b_5 \oplus k_6[3])$
                    $T_3[b_6] \leftarrow T_3[b_6] + 1$
            Sample $u_5[0]$ such that                              ▻ One solution among $2^8$
                for all Input structures $C$ do
                    Using table $T_3$, compute the sum (9.1)
                    If the sum is not zero, abort
            EndSample
            If there is a result for $u_5[0]$, return this guess of $k_6[3]$
        EndSample
        If there is a result for $k_6[3]$, return this guess of $k_6[2]$
    EndSample
    If there is a result for $k_6[2]$, return this guess of $k_6[0,1]$
EndSample
```

The classical time complexity of this procedure (in S-Boxes and inverse S-Boxes) is:

$$\underbrace{2^{2\times 8}}_{\substack{\text{Over}\\ k_6[0,1]}} \left(16\cdot 2^{32} + \underbrace{2^8}_{\substack{\text{Over}\\ k_6[2]}} \left(2^{24}\cdot 8 + \underbrace{2^8}_{\substack{\text{Over}\\ k_6[3]}} \left(2^{16}\cdot 8 + \underbrace{2^8}_{\substack{\text{Over}\\ k_6[3]}} \cdot 2^8\cdot 8\right)\right)\right) .$$

We can bound it as less than $2^{54}$ S-Boxes. Furthermore, this procedure uses $2^{35}$ classical queries and $8\times 2^{24}$ 32-bit registers of classical RAM (each step needs efficient random-access to the table of the previous step).

**Quantum equivalent.** The quantum equivalent of Algorithm 9.2 performs nested quantum searches, as in Section 2.2. It requires $8\times 2^{24}$ 32-qubit registers in the QRAQM model, as some data is written in the tables $T_1, T_2, T_3$ *inside quantum searches*, and $2^{35}$ 256-bit registers of *classical* memory to store the chosen-plaintext queries. The nested searches need to distinguish whether there is a solution or none, and if there is a solution, we are guaranteed that there is exactly one. Thus, we can run Exact Amplitude Amplification (Theorem 2.3). The time complexity, adapted from the classical one, is:

$$\left\lceil \frac{\pi}{4}2^8 \right\rceil \left(2\times 2^{36} + 2\left\lceil \frac{\pi}{4}2^4 \right\rceil \left(2\times 2^{27} + 2\times \left\lceil \frac{\pi}{4}2^4 \right\rceil \left(2\times 2^{19} + 2\left\lceil \frac{\pi}{4}2^4 \right\rceil \times 2\times 2^{11}\right)\right)\right)$$

quantum S-Boxes. Notice that each level of nesting adds a factor $\frac{\pi}{2}$, since quantum search contains a factor $\frac{\pi}{4}$ and the nested searches need to be uncomputed. The number of iterations of each subprocedure are $\left\lceil \frac{\pi}{4}2^4 \right\rceil = 13$ and $\left\lceil \frac{\pi}{4}2^8 \right\rceil = 201$. We obtain a quantum time equivalent to $2^{44.73}$ reversible S-Boxes.

### 9.4.3 Q1 Square Attack on 7-round AES

To attack 7 rounds of AES, we append a round to the previous attack and guess completely the last round key $k_7$. With this method, the 256 and 192 variants are within reach.

**Without partial sums.** Using the attack framework of [DKR97], we retrieve the key with $2^{37}$ chosen-plaintext queries, a quantum time equivalent to $2^{121}$ reversible S-Boxes, a small number of qubits, $2^{37}$ classical memory and no QRAQM.

First of all, we increase the key search space to 20 unknown bytes, so we need more chosen plaintext queries as before. $2^5$ structures of $2^{32}$ plaintexts are sufficient and ensure to have only one result with high probability. We perform quantum search over a search space of size $2^{20\times 8}$ (the partial key bytes) and expect one solution; testing is done sequentially in time $2^{32}\times 2^5\times 5$ S-Boxes, by computing the $2^5$ XORs in $x_4[0]$. So the quantum time complexity is $\left\lceil \frac{\pi}{4}2^{20\times 4} \right\rceil \left(2^{32}\times 2^5\times 10\right) = 2^{119.97}$ S-Boxes.

This complexity falls between Grover search for 7-rounds AES-192 and for AES-256. This procedure does also not better than the classical 7-round impossible differential and Meet-in-the-middle attacks recalled in Table 9.5.

**With partial sums.** We obtain a better time complexity by wrapping Algorithm 9.2 inside a Grover search over the additional key bytes. This "outer" search is actually merged with the outer sampling loop in Algorithm 9.2, to avoid losing a factor $\frac{\pi}{2}$ to uncomputations. For AES-256, there are 15 more key bytes to search for, hence the number of iterations is multiplied by $2^{60}$ and the complexity becomes $2^{44.73+2+60} = 2^{106.73}$ as we use 4 times more structures. For AES-192, there is one less key byte to guess, due to key-schedule properties.

## 9.5 Quantum DS-MITM Attack on 8-round AES-256

The Demirci-Selçuk Meet-in-the-Middle (DS-MITM) attack against reduced-round AES was introduced in [DS08]. Many improvements have been proposed since; the most efficient ones are described in [DFJ13]. This attack also relies on a distinguisher on 4 rounds of AES, that is extended into a key-recovery.

In this section, we give a quantum DS-MITM attack on 8 rounds of AES-256. This attack will be completely described in Section 9.5.6 on page 227. It reaches the highest number of rounds in the quantum setting. Before going into the details of our procedure, we will recall the classical DS-MITM attack on 7-round AES and detail the main ideas that enable to add one more round. The new quantum attack procedure will differ significantly from the classical ones, although it uses the same ideas. We will show in Section 9.6 how to exploit this new attack design to improve the memory cost of classical attacks.

### 9.5.1 The AES S-Box Differential Equation

As a building block of our attack, we need to solve the AES S-Box differential equation: given $\Delta_x, \Delta_y$, find $x$ such that:

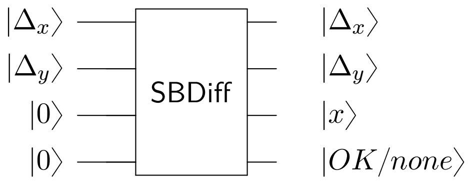
$$S(x) \oplus S(x \oplus \Delta_x) = \Delta_y \quad . \tag{9.2}$$

This cost is classically neglected, as we would use a $2^8 \times 2^8$ lookup table. However, the quantum equivalent would require QRACM. Thus, this analysis is crucial if we want our attack to beat Grover search *without QRACM*. As we count the complexity in S-Boxes, note that making the comparison with the circuit of [Jaq+20] instead of [Gra+16] induces a little change with respect to the analysis of [BNS19b], to our disadvantage.

**Lemma 9.3** (S-Box differential equation). *Given $\Delta_x$ and $\Delta_y$ such that $\Delta_x\Delta_y \neq 0$, there exists either zero, two or four pairs $x, y, x', y'$ such that $S(x) = y, S(x') = y'$, $x \oplus x' = \Delta_x$, $y \oplus y' = \Delta_y$.*

*There exists a quantum unitary* SBDiff *that, given such $\Delta_x$ and $\Delta_y$, finds a solution $x$ if it exists and output (x, OK) in this case, and outputs (0, none) otherwise. The time complexity of* SBDiff *is around 18 S-Box computations and it uses 22 ancilla qubits.*

*If we only want to know if a solution exists and not an explicit solution, the cost drops to 9 S-Box computations and 15 ancilla qubits.*



**Quantum Circuit 9.2:** Circuit that computes a solution of a differential equation on the AES S-Box.

The technical material skipped here can be found in [BNS19b]. The idea is to reduce the differential equation to a quadratic equation in $\mathbb{F}_{2^8}$, whose solutions are tabulated, and to use an optimized QRACM emulation circuit. The counts obtained in [BNS19b] are the following:

- 32 qubits (16 inputs, 1 output, 15 ancilla) and 8886 Clifford+T gates to check the existence of a solution;
- 47 qubits (16 inputs, 9 output, 22 ancilla) and 17507 gates to get an explicit solution.

These numbers correspond respectively to 9 and 18 S-Box circuits of [Jaq+20].

### 9.5.2 Distinguisher on 4-round AES

Recall that $x_i, y_i, z_i, w_i$ denote the successive AES states after each transformation at round $i$, $k_i$ denotes the subkey at round $i$ and $u_i = MC^{-1}(k_i)$. We denote by $[j]$ the byte $j$ corresponding to the byte ordering described in Figure 9.1, so $x_i[j]$ refers to the $j$th byte of the state before $SB$ from round $i$. The classical DS-MITM attack relies on a distinguisher for 4-round AES, which is depicted in Figure 9.5 and proven in Lemma 9.4.

**Lemma 9.4** (4-round property ([DFJ13], Proposition 2))**.** *Suppose that we are given a plaintext-ciphertext pair $(P, P'), (C, C')$ active in one byte $i$ before and one byte $j$ after 4 AES rounds. Consider the plaintexts $P_0 = P, \ldots, P_{255}$ obtained from $P$ by making the difference in byte $i$ assume all values, that is, $P_1$ corresponds to adding* 1 *to $P$ in byte $i$,* etc. *Collect the corresponding ciphertexts $C_0 = C, \ldots, C_{255}$ and the unordered multiset of differences in byte $j$. Then this multiset can only assume* $2^{80}$ *values. There is a procedure* MultisetTable *that tabulates these values in* $2^{88}$ *equivalent encryptions.*

*Proof.* This comes from the fact that the AES S-Box equation has on average one solution. Thus, knowing differences before and after a SubBytes layer constrains the values of the states.

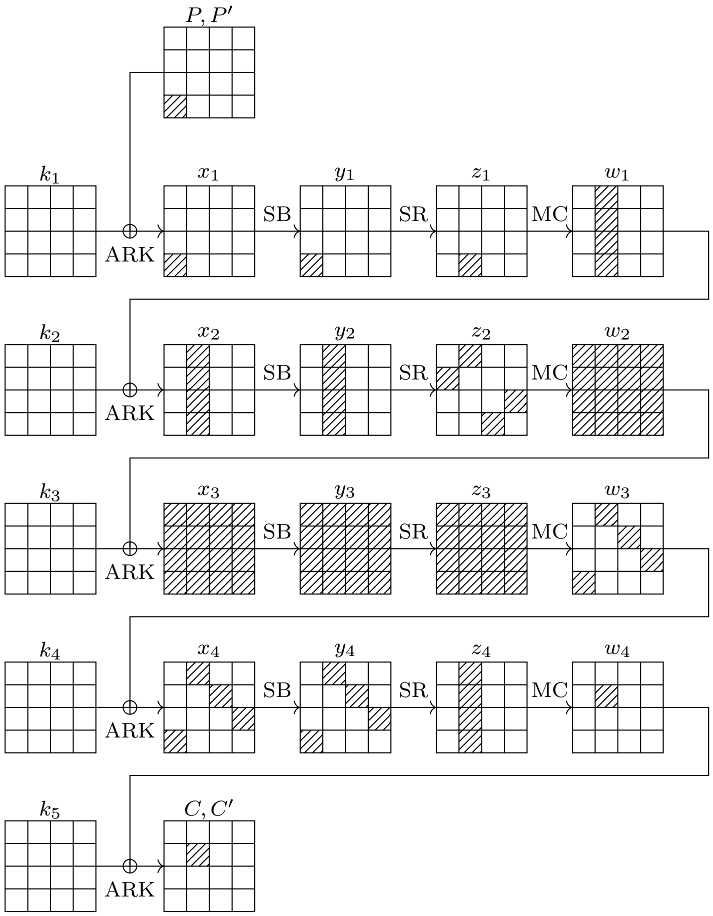


**Figure 9.5:** Distinguisher on 4-round AES. It is valid for any position of the active bytes in $P$ and $C$.

**Algorithm 9.3** Distinguisher on 4-round AES.

```
Precomputation: create the table of multisets: $T \leftarrow$ MultisetTable()
                                                        ▷ See Lemma 9.4
Input: access to a permutation $\Pi$ that is either random, or a 4-round AES
Output: true if this is AES, false otherwise

Find a pair of plaintexts $P, P'$ such that the differences $P \oplus P'$ and $\Pi(P) \oplus \Pi(P')$
   are active respectively in bytes $i$ and $j$ (as in Figure 9.5)
Compute the sequence $C_0 = \Pi(P_0), \ldots, C_{255} = \Pi(P_{255})$ by making byte $i$ of $P$
   assume all values
if the multiset $\{C_0[j], \ldots, C_{255}[j]\}$ is in $T$ then
    return true
else
    return false
```

Without loss of generality, consider the path of Figure 9.5, where we have taken $i = 3$ and $j = 5$. In the forwards direction, we guess the values of $x_1[3]$, $x'_1[3]$, $x_2[4-7]$. Then we can deduce $y_1[3]$ and $y'_1[3]$ and propagate the difference $\Delta y_1[3]$ through the linear layer, until $\Delta x_2[4-7]$. Since we know $x_2[4-7]$ and $\Delta x_2[4-7]$, we also deduce $x'_2[4-7]$ and go through the SubBytes layer at round 2. We deduce $\Delta x_3[-]$.

In the backwards direction, we guess $\Delta x_4[5] = \Delta w_4[5]$, $y_4[0,5,0,14]$. Thus we know $y'_4[0,5,0,14]$ as well and we can go through the SubBytes layer backwards. Next, we obtain $\Delta y_3[-]$.

For each byte, we solve the S-Box differential equation. There is on average one solution. Thus, for each guess of the 10 bytes $x_1[3]$, $x'_1[3]$, $x_2[4-7]$, $\Delta x_4[5]$, $y_4[0,5,0,14]$, we obtain on average one complete path, and we know all the states depicted in Figure 9.5.

Since we know all these states, we can now make the difference in $x_1[3]$ vary and collect the corresponding differences in $x_5[5]$. Thus, we obtain a table of $2^{80}$ possible multisets. □

As a random multiset of 256 bytes can assume $2^{506}$ values ([DFJ13]), we can distinguish 4 rounds of AES from a random permutation with a negligible probability of error (smaller than $2^{80-506}$).

### 9.5.3 Classical DS-MITM Attack on 7-round AES

The classical DS-MITM attack on 7-round AES appends 1 round before and 2 rounds after the distinguisher of Figure 9.5, as displayed in Figure 9.6.

In order to reach a single-byte difference before and after the 4 rounds of the distinguisher, the attack starts by producing many plaintext-ciphertext pairs with input differences in $\Delta_{in}$ and output differences in $\Delta_{out}$, where $\Delta_{in}$ is a diagonal and $\Delta_{out}$ is the MixColumns of an antidiagonal. Without loss of generality, we use again the path of Figure 9.7. We need $2^{48}$ such pairs.

**Lemma 9.5** (Finding pairs [DFJ13, Section 4.1])**.** *There exists a classical procedure* FindPairs *that, with* $2^{113}$ *encryption queries, returns* $2^{48}$ *plaintext-ciphertext pairs* $(P, P')$, $(C, C')$ *such that:*

- *The difference* $\Delta P = P \oplus P'$ *is active only in bytes* $0, 5, 10, 15$*,*
- *The difference* $MC^{-1}(\Delta C) = MC^{-1}(C \oplus C')$ *is active only in bytes* $0, 7, 10, 13$*.*

*Proof.* We encrypt structures of $2^{32}$ plaintexts that make the main diagonal take all values, hence the input differential is always satisfied. By sorting the ciphertexts, we find easily the pairs satisfying the output differential. Each structure yields $2^{32}(2^{32}-1)/2$ pairs and each pair has a probability $2^{-96}$ of satisfying the output constraint. Hence, we need to encrypt $2^{81}$ structures. □

After having precomputed the table of Lemma 9.4 and these pairs, we perform an exhaustive search for the 9 key bytes of $k_0$, $u_6$ and $u_7$ shown on Algorithm 9.4. The probability of failure of the distinguisher is so low that computing a single multiset is

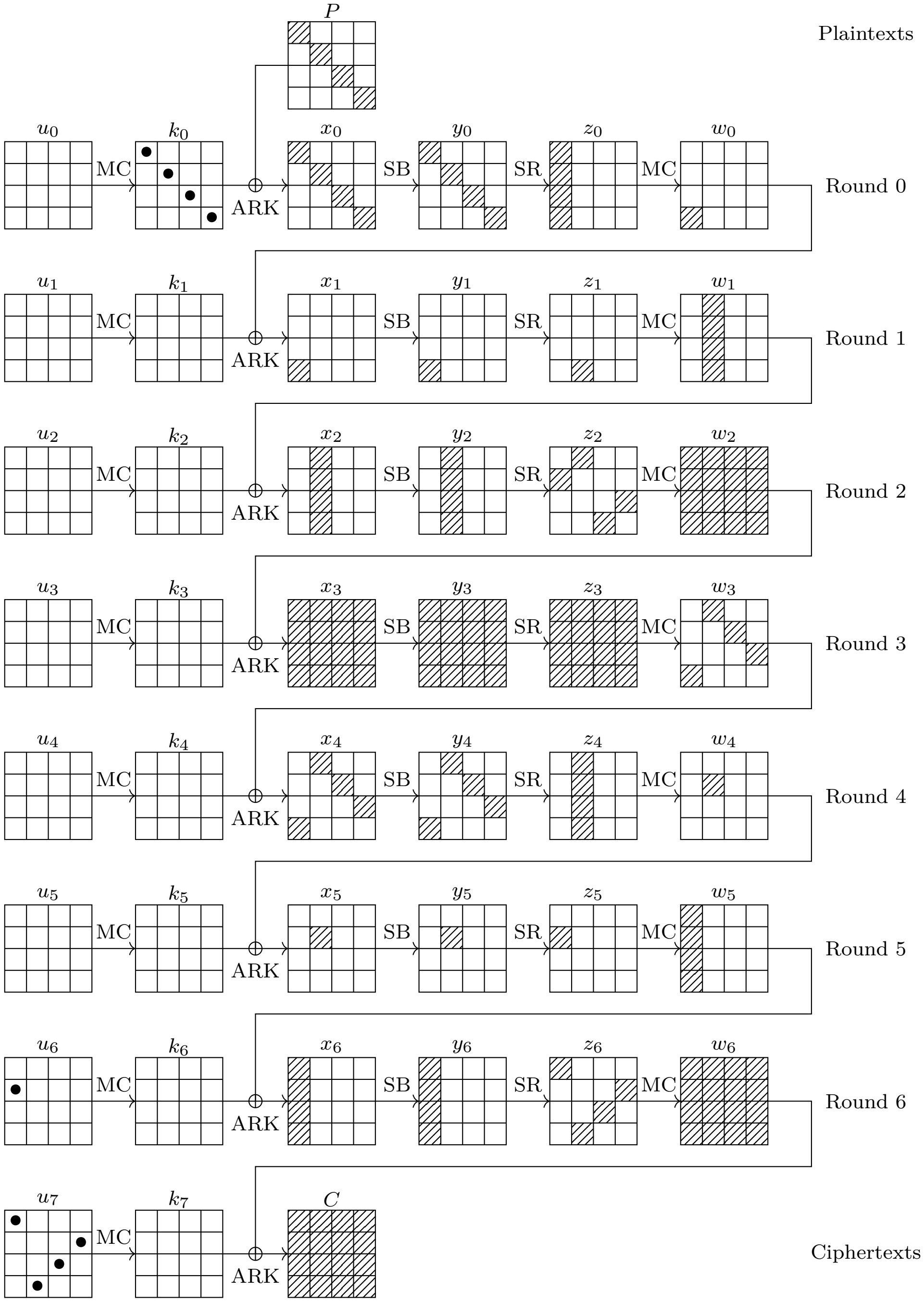


**Figure 9.6:** Full differential path used in the classical 7-round attack. Guessed key bytes are denoted by •.

enough to discriminate the good key guess with overwhelming probability. Once these key bytes have been found, the recovery can be finished by exhaustive search or by running Algorithm 9.4 with another choice of input diagonal.

**Algorithm 9.4** Classical DS-MITM attack on 7-round AES [DFJ13, Section 4.1], based on the path of Figure 9.6.

```
Input: encryption query access to a 7-round AES oracle
Output: secret key bytes $k_0[0,5,10,15], u_6[1], u_7[0,7,10,13]$
Create the table of multisets: $T \leftarrow \mathsf{MultisetTable}()$    ▷ See Lemma 9.4
Create a table of pairs: $T' \leftarrow \mathsf{FindPairs}()$    ▷ See Lemma 9.5
for all values of $k_0[0,5,10,15], u_6[1], u_7[0,7,10,13]$ do
    Find in $T'$ a pair $P, P', C, C'$ such that $x_1$ is active only in byte 3 and $y_6$ is active only in byte 5
        ▷ This is a 6-byte condition, which is why we need $2^{48}$ pairs in $T'$
    Make $x_1[3]$ assume all values $0, \dots, 255$
    Using the known key bytes of $k_0$, find the corresponding plaintexts $P_0, \dots, P_{255}$
    Encrypt $P_0, \dots, P_{255}$, obtain $C_0, \dots, C_{255}$
    Decrypt partially $C_0, \dots, C_{255}$ using the known key bytes of $u_6$ and $u_7$, and obtain the sequence of 256 differences in $x_5[5]$
    if the multiset of differences is in $T$ then
        return $k_0[0,5,10,15], u_6[1], u_7[0,7,10,13]$
```

### 9.5.4 Modifications to the Distinguisher

A classical DS-MITM attack on 8-round AES-256 and AES-192 already exists, but it adds the 8th round at the end of the cipher. Unfortunately, with this attack setting, the complexity of finding the input-output pairs (Lemma 9.5) increases dramatically and we need many more of them. Similarly as impossible differential attacks (Section 9.3.1), this makes a significant speedup unlikely. Thus, we use the alternative idea of adding a round in the *middle* of the differential path, which is classically used in a 9-round attack on AES-256 [DFJ13]. Lemma 9.4 is adapted as follows. Its proof is similar to Lemma 9.4, except that since we traverse one more round, we have a full 16-byte state to guess.

**Lemma 9.6** (5-round distinguisher on AES [DFJ13, Section 4.2])**.** *Suppose that we are given a plaintext-ciphertext pair active in one byte before and after 5 AES rounds. If we make the difference in input take all $2^8$ values and collect the multiset of output differences in output, there are only $2^{208} = 2^{26 \times 8}$ (26 byte-conditions) possibilities.*

Next, our attack strategy will allow to replace multisets by $\delta$*-sequences*. While multisets are of fixed size $2^8$, $\delta$-sequences can be smaller, since we only need to ensure a negligible probability of failure in the distinguisher. As they are ordered, less information is also lost. We will take sequences of $2^5$ single-byte differences.

**Definition 9.2** ($\delta$-sequence)**.** Let $P, P'$ be a pair of plaintexts that satisfies the full differential path of Figure 9.7. Let $P_1, \dots, P_{32}$ the 255 plaintexts obtained by making the difference with the state of $P$ in $x_1[3]$ ($\Delta x_1[3]$) assume the values $1, \dots, 32$. We name *$\delta$-sequence* the corresponding sequence of differences in $x_6[5]$.

By definition, a $\delta$-sequence contains $32 \times 8 = 256$ bits of information. We adapt the distinguisher of Lemma 9.6 for $\delta$-sequences.

**Lemma 9.7** (5-round distinguisher for sequences)**.** *Suppose that we are given a plaintext-ciphertext pair $(P, P'), (C, C')$ active in one byte $i$ before and one byte $j$ after 5 AES rounds. Suppose also that the input and output differences in these bytes are given. Consider the plaintexts $P_0 = P_1, \dots P_{32}$ obtained from $P$ by making the difference in byte $i$ assume all values from 1 to 32, that is, $P_1$ corresponds to adding 1 in byte $i$, etc. Collect the corresponding ciphertexts $C_1, \dots C_{32}$ and the* ordered sequence *of differences with $C$ in byte $j$ ($\delta$-sequence). Then there are only $2^{192}$ possibilities among $2^{256}$.*

*Proof.* The proof follows Lemmas 9.4 and 9.6, except that, since the input and output differences of the pair are known, the whole space of $\delta$-sequences is reduced by $2^{16}$ (two bytes) in size. □

**False positives.** The probability of failure of the distinguisher based on $\delta$-sequences seems to be dangerously low. Indeed, by Lemma 9.7, the distinguisher incorrectly recognizes a random permutation as a 5-round AES with probability $2^{192-256} = 2^{-64}$. Thus, if we use the distinguisher approximately $2^{64}$ times, we will start to find false positives. The same goes if we use the distinguisher inside a Grover search.

However, in what follows, we will decrease the number of possible $\delta$-sequences to $2^{20\times 8} = 2^{160}$, and we will use the distinguisher $2^{80}$ times. Thus, the expected number of false positives encountered during the search is $2^{-16}$. The error in Grover's algorithm will also be low enough to be neglected.

### 9.5.5 Idea of Our Attack

The full differential path of our attack is depicted in Figure 9.7. A major change with respect to the classical setting is that we do not build the multiset table, which might be somewhat counterintuitive at first sight. This is summarized in Algorithm 9.5.

We will first compute enough plaintext-ciphertext pairs so that, given a guess for the outer key bytes (denoted by • in Figure 9.7), we find a pair that satisfies the middle round ($y_1$ to $x_6$) differential. After that, we compute the corresponding $\delta$-sequence, by making the value in $x_1[3]$ assume the values $1, \dots, 32$. This $\delta$-sequence is of length 256 bits. With Lemma 9.7, we are ensured that the sequence takes one of $2^{192}$ possibilities; it is determined by 24 state and key bytes.

If the guess of the key bytes • is good, then the computed $\delta$-sequence can be obtained by some choice of these 24 inner byte-conditions. To verify this, we will do another search. If the guess of the bytes • is not the good one, then with high probability, the $\delta$-sequence that we computed will not appear. Having exactly 256 bits of information

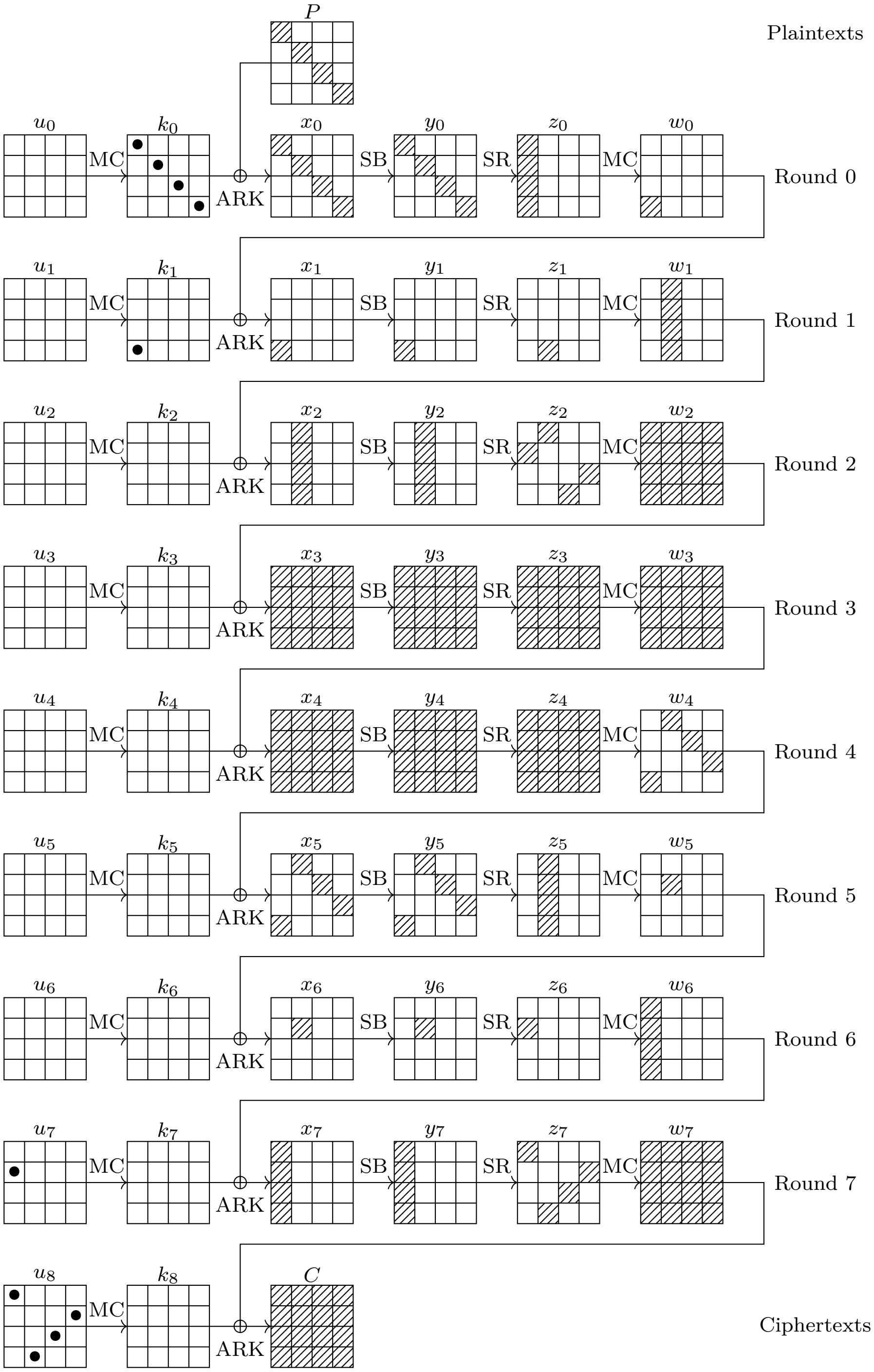


**Figure 9.7:** Full differential path used in the quantum attack. Key bytes guessed in the outer Grover procedure are denoted by •.

**Algorithm 9.5** Our new attack on 8-round AES-256 (sketch).

```
    Input:  query access to an 8-round AES-256 oracle
    Output: some key bytes
 1: Create a table of pairs: $T \leftarrow \mathsf{FindPairs}()$          ▷ See Lemma 9.5
 2: for all Values of the key bytes $\bullet$ do                          ▷ See Figure 9.7
 3:     Find in $T'$ a pair $P, P', C, C'$ such that $x_1$ is active only in byte 3 and $y_6$ is active
    only in byte 5
 4:     Make the value in $x_1[3]$ assume all values $1, \ldots, 32$
 5:     Using the known key bytes $\bullet$, find the corresponding plaintexts $P_1, \ldots, P_{32}$
 6:     Encrypt $P_1, \ldots, P_{32}$, obtain $C_1, \ldots, C_{32}$
 7:     Decrypt partially $C_1, \ldots, C_{32}$ and obtain the $\delta$-sequence in $x_6[5]$
 8:     Check if this $\delta$-sequence is a possible one:
 9:     for all possible values of the $\delta$-sequence do
10:         if there is a match then
11:             return the current guess of the key bytes $\bullet$
```

ensures a good overall success probability, and reducing the amount of computations from $2^8$ to $2^5$ partial encryptions is crucial for the complexity of our attack.

**First complexity estimation.** As there are 10 outer key bytes in the search and possibly as high as $2^{10+16}$ $\delta$-sequences to sieve at Step 9 in Algorithm 9.5, the complexity could reach higher than exhaustive search of the key. But we use different ideas to reduce the amount of work when trying all possible $\delta$-sequences at Step 9. First of all, using $\delta$-sequences instead of multisets reduces the amount of partial encryptions from $2^8$ to $2^5$ for each one. Next, since the pair $P, P', C, C'$ is known at the time of test, there are two less byte-degrees of freedom than in the table-based approach, as the input and output differences $P \oplus P'$ and $C \oplus C'$ are known. Finally, we use the key-schedule relations of Lemma 9.1, specific to AES-256, that remove 4 degrees of freedom from this search.

All in all, we now expect the complexity to be below exhaustive search. By writing the algorithm as a **Sample** program, we will obtain the same result in the quantum setting, and beat the bound set by the exhaustive search of the key with Grover's algorithm.

### 9.5.6 Classical Description of Our Attack

In Algorithm 9.6, we describe our attack classically, as a **Sample** procedure, in the framework introduced in Chapter 2. The remaining of this section is devoted to its details and complexity. This attack uses memory, but except in FindPairs, we resort only to sequential lookup circuits such as Lemma 1.2. We count the running time in AES S-Boxes.

Classical exhaustive search of the key takes approximately $160 \times 2^{256} = 2^{263.32}$ S-Boxes. We prove that our procedure goes below that. We use the fact (Section 9.5.1) that solving the S-Box differential equation costs approximately 9 S-Boxes existentially (when we only test if there is a solution) and 18 S-Boxes to output the solution.

*Remark* 9.3. We suppose that all the differential equations have either zero or two solutions. Only 40 S-Box equations will be involved in the distinguisher, and there is only one good guess of key bytes, and one corresponding good $\delta$-sequence. The probability that this good path encounters an S-Box equation with 4 solutions is $1 - \left(\frac{127}{128}\right)^{40} \simeq 27\%$. Our assumption that this event does not occur multiplies the average complexity by a small factor.

**Finding the pairs.** We use the same set of plaintext-ciphertext pairs as in the 7-round classical one, given in Lemma 9.5. They are found *classically* by the procedure FindPairs, and since the running time is below $2^{128}$, this procedure can remain the same in the quantum attack.

**Lemma 9.8** (The Good Pair)**.** *Given the set of pairs of Lemma 9.5, given a guess for* $k_0[0,5,10,15]$ *and* $u_8[0,7,10,13]$*, there exists approximately one pair* $(P,C),(P',C')$ *that satisfies the full inner differential characteristic. Besides, there exists a quantum unitary that, on input* $k_0[0,5,10,15]$ *and* $u_8[0,7,10,13]$*, returns this pair. It runs in approximately* $2^{53}$ *S-Boxes computations.*

*Proof.* We test sequentially each of the $2^{48}$ possible pairs. There are 16 S-Box computations to do for each pair (4 in round 0 and round 8 for both members of the pair), to check if it is the good one, and to uncompute. □

**Computing a $\delta$-sequence.** Once we have the pair $P, P', C, C'$ from Lemma 9.8, we can compute the associated $\delta$-sequence, using the key guesses. The plaintexts are computed thanks to $k_1[3]$ and $k_0[0,5,10,15]$, then encrypted, and the ciphertexts are partially decrypted using $u_8$ and $u_7$ in order to obtain the sequence of differences in $x_6[5]$. This list contains $2^5$ byte values, hence 256 bits are sufficient to store it. This overhead is small with respect to the amounts (thousands) needed to perform reversible AES encryption.

**Lemma 9.9** (Computing the $\delta$-sequence)**.** *Given the subkey guesses in* $k_0$*,* $k_1$*,* $u_7$*,* $u_8$*, and a pair satisfying the inner differential, we can compute the expected* $\delta$*-sequence using* $2^{13}$ *S-Boxes.*

*Proof.* Each of the $2^5$ elements of the $\delta$-sequence requires a call to the secret-key oracle, which costs $\leqslant 2^8$ S-Boxes. The other computations are negligible. □

**State equations.** Let $\ell_2$ and $\ell_3$ be the linear functions that, on input a column, give the third (respectively fourth) byte of this mixed column, and $\ell$ the linear function which, on input a column $C$, gives the first byte of $MC^{-1}(C)$. Equations (9.3), (9.4), (9.5) and (9.6) below are respectively derived from the four key-schedule relations given in Lemma 9.1, where we replace some key bytes by sums of state bytes.

$$\begin{aligned}
k_0[10] &= k_4[2] \oplus k_4[10] \\
&\implies k_0[10] = x_4[2] \oplus x_4[10] \oplus \ell_2(y_3[0,5,10,15]) \oplus \ell_2(y_3[8,13,2,7]) \quad (9.3)
\end{aligned}$$

$$\begin{aligned}
k_0[15] &= k_4[7] \oplus k_4[15] \\
&\implies k_0[15] = x_4[7] \oplus x_4[15] \oplus \ell_3(y_3[3,4,9,14]) \oplus \ell_3(y_3[1,6,11,12]) \quad (9.4)
\end{aligned}$$

$$\begin{aligned}
k_5[3] &= k_1[3] \oplus S(k_4[15]) \oplus S(k_4[11] \oplus k_4[15]) \\
&\implies x_5[3] \oplus \ell_3(y_4[0,5,10,15]) = k_1[3] \oplus S\Big(x_4[15] \oplus \ell_3(y_3[1,6,11,12])\Big) \\
&\qquad \oplus S\Big(x_4[15] \oplus \ell_3(y_3[1,6,11,12]) \oplus x_4[11] \oplus \ell_3(y_3[8,13,2,7])\Big) \quad (9.5)
\end{aligned}$$

$$\begin{aligned}
k_2[4\text{–}7] &= k_4[0\text{–}3] \oplus k_4[4\text{–}7] \\
&\implies x_2[4\text{–}7] \oplus MC(y_1[3,4,9,14]) = \\
&\quad x_4[0\text{–}3] \oplus MC(y_3[0,5,10,15]) \oplus x_4[4\text{–}7] \oplus MC(y_3[3,4,9,14]) \\
&\implies \ell(x_2[4\text{–}7]) \oplus y_1[3] = \ell(x_4[0\text{–}3]) \oplus y_3[0] \oplus \ell(x_4[4\text{–}7]) \oplus y_3[3] \quad (9.6)
\end{aligned}$$

**Sieving with the state equations.** After Step 14 in our attack, we have two choices for each byte of $x_2[0\text{–}3]$ (one column of $x_2$), each byte of $x_3$, each of $x_4$ and each byte of $x_5[3,4,9,14]$,which are solutions of the corresponding differential equations. We use the 4 key relations (9.3) to (9.6), which are translated into relations between these bytes, to sieve them. As there are 40 bit-degrees of freedom and 4 byte constraints, we expect $2^8$ possibilities to pass. These relations turn out to constrain completely 32 of the byte values and leave the 8 others free: these 8 bytes appear in none of the relations. This is represented in Figure 9.8, with the bytes of $x_2$, $x_3$, $x_4$ and $x_5$ concerned. We consider that the states $x_2, x_3, x_4, x_5$ are independent, due to the MixColumns or SubBytes operations traversed.

### 9.5.7 Classical Complexity

There are three nested **Sample**s in Algorithm 9.6. We compute their complexity from the outermost to the innermost one.

**Sampling the key guesses (Step 2).** We guess 10 key bytes; there are thus $2^{10\times 8}$ possibilities. We expect exactly one of them to be good. Given such a guess, we can compute the good pair (Line 3) in $2^{53}$ S-Boxes, and start to compute the middle states. The cost is in total:

$$2^{80}\left(2^{53} + \mathbf{s}\right) \ ,$$

where $\mathbf{s}$ is the next **Sample**.

**Algorithm 9.6** Classical version of our attack on 8-round AES-256.

```
Input:  query access to an 8-round AES-256 oracle
Output: $k_0[0, 5, 10, 15]$, $k_1[3]$, $u_7[1]$, $u_8[0, 7, 10, 13]$        ▷ See Figure 9.7
Compute the table $T \leftarrow \mathsf{FindPairs}()$ of $2^{48}$ plaintext-ciphertext pairs
with input difference active in bytes $0, 5, 10, 15$ and
output difference active in (mixed) bytes $0, 7, 10, 13$        ▷ See Lemma 9.5
Sample $k_0[0, 5, 10, 15]$, $k_1[3]$, $u_7[1]$, $u_8[0, 7, 10, 13]$ such that
    Find a pair $P, C, P', C'$ which satisfies the differential path        ▷ $2^{53}$ S-Boxes
    Compute the associated $\delta$-seq. of differences in $\delta w_5[5]$        ▷ oracle queries
    Compute $x_1[3], x'_1[3]$ and obtain $\Delta x_2[4–7]$
    Compute $x_6[5], x'_6[5]$ and obtain $\Delta y_5[3, 4, 9, 14]$
    Sample $\Delta y_2[4–7]$, $\Delta x_5[3, 4, 9, 14]$, $\Delta x_4$ such that
        if $\Delta y_2[4–7]$ and $\Delta x_2[4–7]$ do not match abort        ▷ prob. $2^{-4}$
        if $\Delta x_5[3, 4, 9, 14]$ and $\Delta y_5[3, 4, 9, 14]$ don't match abort        ▷ prob. $2^{-4}$
        Store the two solutions for each byte
        From $\Delta y_2[4–7]$, compute $\Delta x_3$
        From $\Delta x_5[3, 4, 9, 14]$, compute $\Delta y_4$
        Match $\Delta x_4$ against $\Delta y_4$ and $\Delta x_3$, column by column.
        if they do not match abort        ▷ prob. $2^{-8}$ for each column
                ▷ Here, one guess over $2^{40}$ has passed the S-Box differential equations
        Write (9.3) for all $2^{10}$ choices of $x_4[2], x_4[10], y_3[0, 5, 10, 15, 8, 13, 2, 7]$
          4 of them are expected to pass. Store them.
        Write (9.4) for all $2^{10}$ choices of $x_4[7], x_4[15], y_3[3, 4, 9, 14, 1, 6, 11, 12]$
          4 of them are expected to pass. Store them.
        For each of the $4 \times 4$ stored choices and $2^4$ choices of $x_5[3], x_4[0, 5, 11]$,
          write Equation (9.5): one of these $2^8$ possibilities is expected to pass
                ▷ The full state $y_3$ and $x_4[0, 2, 5, 7, 10, 11, 15], x_5[3]$ are now determined.
        For these bytes and each $2^8$ choices of $x_2[4–7], x_4[1, 3, 4, 6]$, write Equation
    (9.6): one choice is expected to pass
                ▷ In the end, the full state $x_3$ is determined, alongside $x_5[3]$, $x_2[4–7]$ and
    $x_4[0–7, 10, 11, 15]$. Bytes $x_4[8, 9, 12, 13, 14]$ and $x_5[4, 9, 14]$ remain free.
        Sample choices for $x_4[8, 9, 12, 13, 14]$ and $x_5[4, 9, 14]$ such that
                ▷ There are exactly $2^8$ possibilities to explore
            Since the whole sequence of states $x_2[4, 5, 6, 7]$, $x_3$, $x_4$, $x_5[3, 4, 9, 14]$ is
    known, compute the expected $\delta$-sequence in $\delta w_5[5]$        ▷ $2^5 \times 40$ S-Boxes
            If it does not equal the expected sequence, abort
        EndSample
        If there is no solution, abort
    EndSample
    If there is no solution, abort
EndSample
```

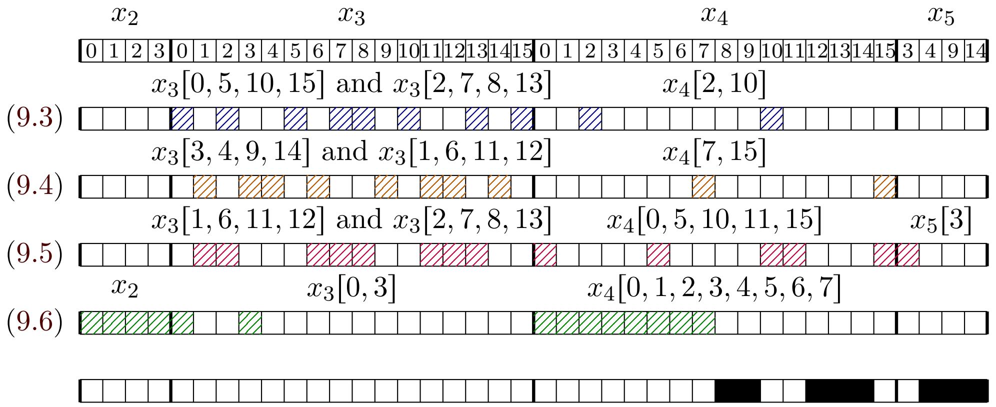


**Figure 9.8:** All key relations (9.3), (9.4), (9.5), (9.6) and the 8 remaining "free" bytes

**Sampling the $16+8$ differences (Step 7).** The search space of differences is of size $2^{24\times 8} = 2^{192}$. We solve 40 S-Box differential equations in total. In solving them, the most costly step is matching between $\Delta x_3$, $\Delta x_4$ and $\Delta y_4$: as we traverse one round, we cannot match byte per byte, but column by column. For each column, we need to solve the AES S-Box differential equation 4 times between $\Delta x_3$ and $\Delta x_4$, and for each of the $2^4$ solutions if they exist, 4 times again between $\Delta x_4$ and $\Delta y_4$. This takes $(2^8 \times \times 18)$ equivalent S-Boxes per group of 4 bytes.

We have now determined all state bytes. There are 40 bytes (represented in Figure 9.8) that can take two values each. Thus the full sequence of states can admit one of $2^{40}$ values, and is not completely determined yet. But the state equations stemming from the key-schedule allow to reduce this space.

1. In Equation (9.3), there are 10 bytes involved, 2 choices for each, thus $2^{10}$ computations to make. But no S-Boxes are involved, so this cost is negligible. The same goes for Equation (9.4). 4 solutions are expected for each.

2. We test Equation (9.5) afterwards. There are $2^4$ choices of $x_5[3], x_4[0,5,11]$, and the other bytes were involved in (9.3) and (9.4), thus $2^8$ choices on average. Due to the previous constraints, $x_4[15] \oplus \ell_3(y_3[1,6,11,12])$ can take on average only 4 values and $x_4[11] \oplus \ell_3(y_3[8,13,2,7])$ only 8. Thus, we don't have to recompute the S-Boxes in this equation for all $2^8$ choices,: we evaluate only $4+8$ S-Boxes, which is negligible.

3. At this point, a significant proportion of the state should be determined. Although there are 14 state bytes in Equation (9.6), only 8 of them are truly free. Thus we do $2^8$ computations without S-Boxes.

So the full cost of the **Sample** is:

$$2^{(16+8)\times 8-40}\left(16\times 9+4\times(2^8\times 4\times 18)+\mathbf{s}'\right) \quad ,$$

where $16 \times 9$ stems from the outer S-Box differential equations, **s**' is the next **Sample**, and $2^{-40}$ is the probability that a guess satisfies the differential equations.

**Sampling the state sequences.** At Step 19, we have 8 remaining free state bytes, represented in black in Figure 9.8, that can take two values. Thus there are $2^8$ possible states that we must check. For each possibility, we compute the $\delta$-sequence using $2^5 \times 40 = 1280$ S-Boxes and match against the expected one. If there is no match, this wasn't the good state sequence. If there is one, we have found the solution. The cost is $2^8 \times 1280$ .

In total we obtain

$$\mathbf{2^{80}} \left(2^{53} + \mathbf{2^{152}} \left((4 \times (\mathbf{2^8} \times 72)) + \mathbf{2^8} \times 1280\right)\right) \quad , \tag{9.7}$$

where we highlight the number of iterations of each **Sample**, on which we are going to take the square root.

A direct computation gives a classical complexity of $2^{250.61}$ S-Boxes.[1] We can check that the computation of $\delta$-sequences is the dominant term. Taking into account the key-schedule relations, we do an exhaustive search over 30 bytes, so the best complexity that we can expect is $2^{240}$ times the computation of a $\delta$-sequence, which stands at $2^{250.3}$.

### 9.5.8 Quantum Complexity

By the correspondence of Section 2.2, Algorithm 9.4 has a quantum equivalent in terms of nested Amplitude Amplification procedures. A first estimation, by replacing the iteration counts in Equation (9.7), gives:

$$2\left\lceil \frac{\pi}{4} 2^{40} \right\rceil \left(2\left\lceil \frac{\pi}{4} 2^{76} \right\rceil \left(\left\lceil \frac{\pi}{4} 2^{4} \right\rceil \times 4 \times 72 \times 2 + \left\lceil \frac{\pi}{4} 2^{4} \right\rceil \times 2 \times 1280\right)\right) \quad , \tag{9.8}$$

where each new factor "2" comes from an uncomputation level. As we have $\left\lceil \frac{\pi}{4} 2^4 \right\rceil = 13$, this gives approximately $2^{132.62}$ S-Boxes. As this is very close to the Grover search complexity of $2^{137.56}$ S-Boxes, we have to go into further detail.

**S-Box property.** The differential property can yield 4 solutions, and our quantum circuit SBDiff (Lemma 9.3) yields only 2 of them. As we go through 40 S-Boxes, the probability to hit only differential equations with 2 solutions (for the good path) is $\left(\frac{127}{128}\right)^{40} \simeq 2^{-0.45}$. Furthermore, when restricting to non-zero differentials, the differential equation does not admit exactly $\frac{1}{2}$ solutions on average, but $\frac{127}{255}$. Thus, the total search space for the state bytes (Line 7) is of size $2^{191.86}$ instead of $2^{192}$, and the probability to pass the differential equations is $2^{-40.23}$ instead of $2^{-40}$. The space of state differences admitting solutions is thus of size $2^{151.64}$.

[1] In [BNS19b], a slightly smaller count of $2^{250.3}$ is obtained, because the S-Box differential equations are assumed to cost less with respect to an S-Box.

**Precision.** The variance in the number of solutions will increase the complexity with respect to (9.8).

1. **Sample** of $k_0[0, 5, 10, 15]$, $k_1[3]$, $u_7[1]$, $u_8[0, 7, 10, 13]$ (Line 2): there is exactly a single solution which is accepted by the test. Even the error of the distinguisher, which we know exactly, can be taken into account by Exact Amplitude Amplification (Theorem 2.3). Thus, there is no error term here.

2. **Sample** of $\Delta y_2[4\text{–}7]$, $\Delta x_5[3, 4, 9, 14]$, $\Delta x_4$ (Line 7). When matching $\Delta y_2[4\text{–}7]$ and $\Delta y_5[4, 9, 14]$ against $\Delta x_2$ and $\Delta y_5$, the number of solutions is always the same regardless of the previous key guesses. However, the equations involving $\Delta x_4$ have a varying number of solutions.

   We simulated the variations column by column. We found that the number of solutions when fixing $\Delta x_3$ (deduced from $\Delta y_2$) and $\Delta y_4$ (deduced from $\Delta x_5$) was in 98% of the cases in the interval $2^{27.95}(1 \pm 2^{-9})$. As we have 4 columns, we can estimate that in more than 90% cases we will have a number of solutions that varies by a factor less than $2^{-7}$ around the mean.

   This varying number of solutions changes the probability of success of the **Sample** from $2^{-151.6}$ to a value somewhere between $2^{-151.6}(1 - 2^{-7})$ and $2^{-151.6}(1 + 2^{-7})$. Here we are exactly in the situation of Theorem 2.5, with a success probability that is known with a small relative error. By doing $\frac{\pi}{4}2^{75.8}$ iterations, we obtain a value that passes the test with probability greater than $1 - 2^{-14}$. This modifies in turn the success probability of the **Sample** 1, and increases negligibly the number of iterations that we should perform.

3. Sequential test of the 4 key conditions: the right guess of key and state differences may admit more solutions than the average. Overall, we consider that the 4 equations for the good path might have 4 solutions.

4. Innermost **Sample** (Line 19): the search spans exactly $2^8$ tuples (note that the superposition of them can be computed efficiently), and since there is exactly one solution or none, we can use Exact Amplitude Amplification. However, although this **Sample** loop runs on average once, we need to take into account the maximal number of solutions for the key equations. Thus, the search space increases to a size $2^{10}$, and we can still use Exact Amplitude Amplification (if the equations do not have enough solutions to fill the space, we add dummy elements which always fail).

The final complexity, taking into account the success probability and the precision problems is:

$$\underbrace{\frac{10}{9}2^{0.45}}_{\substack{\text{Assumptions on}\\ \text{the good path}}} \times \underbrace{\left\lceil \frac{\pi}{4}2^{40} \right\rceil}_{\text{Exact AA}} \times 2\left( \left\lceil \frac{\pi}{4}2^{75.8} \right\rceil 2\left( \left\lceil \frac{\pi}{4}2^{4} \right\rceil \times 8 \times 72 + \underbrace{\left\lceil \frac{\pi}{4}2^{4+1} \right\rceil}_{\text{Exact AA}} \times 2 \times 1280 \right)\right)$$

which is less than $2^{134.8}$ S-Boxes and falls indeed below exhaustive search. The dominating term remains the computation of $\delta$-sequences, as expected.

**Making the attack Q1.** The quantum version of Algorithm 9.4 requires superposition encryption queries. Indeed, some of them (for computing the $\delta$-sequences) appear inside a **Sample**, and would occur inside an Amplitude Amplification subroutine. However, it is possible to make the attack Q1 only.

Suppose we have guessed the 9 key bytes $k_0[0, 5, 10, 15]$, $k_1[3]$, $u_8[0, 7, 10, 13]$ and we are given a corresponding good pair, as computed at Line 3 in Algorithm 9.6. The expected $\delta$-sequence is obtained by encrypting a set of plaintexts which depends on this pair and on $k_0[0, 5, 10, 15]$, $k_1[3]$. There are $2^{48}$ pairs. There are $2^{40}$ choices of $k_0[0, 5, 10, 15]$, $k_1[3]$. This means that in total, the whole procedure actually queries $2^{48} \times 2^{40} = 2^{88}$ distinct plaintexts.

It is possible to perform all these queries beforehand. In the outermost **Sample**, instead of computing the $\delta$-sequence on the fly, we go through the set of stored queries and find the ones that interest us (those which correspond to the current pair). This can be done using a sequential circuit similar to QRACM emulation (Lemma 1.2). Each outer iteration now requires $2^{88}$ computations (comparisons only, not S-Boxes). This term is not dominant, and does not change the attack complexity.

## 9.6 Improving Classical DS-MITM Attacks

During this chapter, we went from the classical to the quantum world, as we started with classical cryptanalytic techniques and applied them in the quantum setting. In this last section, we will go backwards and show that our quantum DS-MITM design can have applications in classical cryptanalysis. This can seem counter-intuitive. The fact is that the quantum setting forced us to adopt a new viewpoint (re-ordering the steps) that had been previously overlooked. Notably, this helps in reducing the memory complexity in the classical DS-MITM attacks, which had been for a long time their bottleneck. For 7-round AES, we can reach new intersting time-memory trade-offs, comparable to the best previous ones (if not better) and for 9-round AES-256, the best known classical attack to date.

**Re-ordering the steps.** In the standard DS-MITM attack, the distinguisher in the middle rounds uses a precomputed table, while the online phase (making a key guess, finding pairs, querying and computing multisets) does not use memory. In our new quantum attack, we computed the distinguisher online. Going back to the classical setting, we can decide to keep it this way and, instead, store what was previously the online phase: all key guesses, corresponding pairs and multisets (or $\delta$-sequences).

Now, we will perform an exhaustive search over the middle multiset values (what was previously stored in a table) and look for a match among the precomputed key guesses. The data and time complexities will be similar (as we are basically doing the same computations in a different order), but the memory can be reduced.

### 9.6.1 Improved Attack on 9-rounds AES-256

We focus on the 9-round AES-256 attack of [DFJ13, Fig.6], which is similar to Figure 9.7, with an additional round at the end. The attacker first obtains $2^{144}$ plaintext-ciphertext pairs satisfying the input differential with $2^{113}$ queries, then sieves the outer key bytes. Each pair gives $2^{48}$ possible values for $k_{-1}[0, 5, 10, 15]$, $k_8$, $u_7$ that satisfy the whole differential pattern of Figure 9.7. These values can be enumerated in time $2^{48}$. For each of them, a $\delta$-set is encrypted and the associated multiset is compared to the table of precomputed possibilities: as there are 26 byte parameters that determine the middle rounds, the precomputed table has size $2^{210}$. This can be reduced by a factor $2^7$, by replaying the attack $2^7$ times (with an increase in the data and time complexities).

By reordering the steps, we obtain a new attack based on this previous one that still needs $2^{113}$ data, $2^{210}$ time and now only $2^{144} \times 2^{48} \times 4 = 2^{194}$ memory (the 4 factor stands for the storage of a multiset). We can propose a trade-off different from [DFJ13], by considering a factor of $2^x$ less states to try in the middle if we store $2^x$ times more possible pairs. For this we need a data complexity increased by a factor $2^{x/2}$ (that generates $2^x$ times more pairs), and a time complexity that will be the maximum between $2^{210-x}$ and $2^{194-x}$.

In order to reach a memory of $2^{194}$ with the attack from [DFJ13], one would need a time complexity of $2^{212}$ and a data complexity of $2^{124.5}$. Hence, we obtained a better trade-off, as summarized in Table 9.5.

### 9.6.2 New Trade-offs on 7-rounds AES-128

The best attacks on AES-128 are impossible differential, and our new trade-offs can not always improve on them. However, we can improve previous DS-MITM attacks, at least with respect to the memory. Let us consider the DS-MITM 7-round AES-128 attack of Section 9.5.3. By reordering the steps, we obtain for instance $2^{113}$ data, $2^{113} + 2^{84}$ time and $2^{74}$ memory. This can be improved by considering multiple differentials, and enables us to reach $2^{105}$ data, $2^{105} + 2^{95}$ time and $2^{81}$ memory.

## 9.7 Discussion

In this chapter, we presented *quantum square attacks* against 6-round AES, 7-round AES-192 and AES-256, and a *quantum DS-MITM attack* against 8-round AES-256. All these attacks could be applied in the Q1 setting. The best known classical non-generic key-recoveries target up to 9 rounds of AES-256. Hence, the security margin of AES-256 determined by these attacks is bigger in the post-quantum world.

Although our DS-MITM attack uses the same weaknesses that enabled the classical attacks in the first place, we had to modify considerably its design pattern, in order to compete against Grover's algorithm. Some of the ideas that we used on the way to our quantum design have classical implications: quantum cryptanalysis is a good motivation to think differently about classical attack designs.

**Table 9.6:** Summary of quantum attacks on reduced-round AES of this chapter, compared with Grover search. Quantum time is given in reversible S-Boxes. Memory is counted in 128-bit registers.

| Version | Rounds | Data | Time | Q. mem | C. mem | Technique |
|---|---|---|---|---|---|---|
| 128 | 6 | 2 | $2^{72.56}$ | negl. | negl. | Exhaustive search |
| | 6 | $2^{35}$ | $2^{44.73}$ | $2^{25}$ | $2^{36}$ | Square (partial sums) |
| 192 | 7 | 2 | $2^{104.70}$ | negl. | negl. | Exhaustive search |
| | 7 | $2^{37}$ | $2^{102.73}$ | $2^{27}$ | $2^{38}$ | Square (partial sums) |
| 256 | 7 | 3 | $2^{137.37}$ | negl. | negl. | Exhaustive search |
| | 7 | $2^{37}$ | $2^{106.73}$ | $2^{27}$ | $2^{38}$ | Square (partial sums) |
| | 8 | 3 | $2^{137.56}$ | negl. | negl. | Exhaustive search |
| | 8 | $2^{113}$ | $2^{134.8}$ | $2^{88}$ | negl. | DS-MITM |

The results of this chapter concern only quantum key-recovery attacks. The work of Hosoyamada and Sasaki [HS20] showed that AES-based hash functions might benefit from a *smaller* security margin in the quantum world, due to the gap between the quadratic speedup of Grover search and the non-quadratic speedup of collision search.

Going further, many modes of operation offer only a security up to the birthday bound, and can be attacked by finding collisions. AES has a block size of 128 bits which, contrary to the key length, cannot be raised. The corresponding *quantum* collision bound is of $2^{42.7}$ only. As we discussed in Chapter 2, no generic quantum collision speedup with polynomial memory exists at the moment. However, both the collision search without qRAM of Section 5.1 and the offline Simon's algorithm of Chapter 7 show that it is possible to work around these memory requirements. In the long term, it seems preferable to use a cipher with a higher block size.

# Chapter 10 Saturnin

*Approchez tous les amis, les grands et les petits,*
*Regardez bien*

*Le cœur fier et l'œil malin, voici venir au loin*
*Votre ami Saturnin*

*On dit que rien ne lui fait peur*
*Quand il s'agit d'aventure*
*Il est têtu et bagarreur*
*Mais gentil, je vous le jure !*

*Et s'il est un peu bavard, et même un peu vantard*
*Ça ne fait rien*

*Car plus vous le connaîtrez et plus vous l'aimerez*
*Votre ami Saturnin !*

*Les Aventures de Saturnin (1965), Theme song* [Tou65]

Saturnin, pronounced satyʁnɛ̃ , is the eponymous hero of the french TV series *Les Aventures de Saturnin (The Adventures of Saturnin)* [Tou65], which was broadcasted between 1965 and 1970. The series narrates the life of the duck Saturnin in a village inhabited by animals, using real life footage of animals and voice actors.

Saturnin [Can+20] is a family of lightweight symmetric algorithms aiming at post-quantum security, which was submitted to the NIST LWC project and is currently[1] in the second round[2]. The core primitive of the submission is the block cipher Saturnin, which has 256-bit keys, 256-bit blocks and a 9-bit *domain separator*. Its design allows simultaneously an abstract representation which makes it similar to the AES, and an efficient bitsliced implementation, using only lightweight components (e.g., 4-bit S-Boxes).

[1] October 2020.

[2] Contrary to the TV series of the same name, no animals were harmed during the production of Saturnin. During the shooting of the series, more than 50 ducklings had to play the role of Saturnin, as they often died or grew up too fast [Rid18].

Saturnin is then used in block cipher operation modes for AEAD and hashing that benefit from quantum security guarantees, and that are immune to the Q2 attacks based on Simon's algorithm explored in Chapter 4.

In this chapter, we present the specification of Saturnin, its rationale and security claims; we give a security analysis of the cipher and its modes of operation.

**Contents**

## 10.1 Introduction

The objective of Saturnin is to provide a *lightweight* and *quantum secure* family of symmetric algorithms. Quantum security is mentioned in the report on lightweight cryptography NISTIR8114 [NIS17].

> *When long-term security is needed, these algorithms should either aim for post-quantum security, or the application should allow them to be easily replaceable by algorithms with post-quantum security.*

Before getting to the specification, let us give some general thoughts about the constraints of a quantum-secure cipher and how to build one.

**Key size.** We chose a block cipher design with a key of 256 bits. This naturally follows from the applicability of Grover search against smaller key sizes. In Chapter 9, we recalled that the gate counts for Grover exhaustive search of the keys against AES-128 were of the order of $2^{80}$ Clifford+T gates. This represents actually a massive amount of operations, orders of magnitude higher than Shor's algorithm applied to standard RSA parameters. And this cost will increase if the adversary tries to parallelize such key-recovery. This is why the NIST PQC project actually considers Grover's exhaustive key search of AES-128 as one of its standard security levels.

It is unlikely that a first large-scale quantum attacker, capable of breaking RSA-2048, will also be able of running this Grover search. However, new optimizations may be reachable. A *multi-user* key-recovery will be more efficient than a single-user one, as it increases the number of good solutions; it can also improve the parallelization trade-off. In a related-key setting, the offline Simon's attack of Chapter 7 can decrease the security level down to $2^{128/3} \simeq 2^{43}$, using only negligible quantum memory.

**State size.** The security of most modes of operation (encryption or MACs) is given by a *collision* bound, which is $\mathcal{O}\left(2^{n/2}\right)$ for an internal state of size $n$. Using a cipher with 128-bit blocks, this represents a bound $2^{64}$ which is usually enhanced at the specification level, with a data limit. For example, Chaskey [Mou+14] has a 128-bit state, and limits the data encrypted to $2^{48}$, in order to meet a security level of $2^{80}$ instead of $2^{64}$. Most candidates in the LWC process enforce such limitations.

However, there are many ways in which a quantum adversary can use a small block or internal state size to attack a construction. First of all, the quantum collision bound on 128-bit blocks falls from $2^{128/2}$ to $2^{128/3} \simeq 2^{43}$. Although a quantum speedup for collision search with polynomial memory is not known at the moment, the algorithms that we have presented in Section 5.1 always apply. Better trade-offs between classical and quantum resources may still be at reach. Finally, the offline Simon's attack of Chapter 7 reaches the collision bound $\mathcal{O}\left(2^{n/3}\right)$ for block sizes $n$ *with* $\mathsf{poly}(n)$ *memory*, although it concerns only specific constructions.

Thus, we choose an internal state of 256 bits, twice the size of the AES. Many of the designs submitted to the NIST LWC project also use keys of 256 bits and internal states of at least the same size, but these are often Sponge constructions. Among the second-round candidates based on block ciphers, 11 proposals use variants of AES, GIFT, Skinny, Speck and CHAM (by order of frequency) [GJN19; Cha+20; And+19; Ban+19b; Cha+19b; Cha+19a; CN19; Gou+19; Nai+19; Bei+20; Ban+19a] and three proposals, including SATURNIN, define custom block ciphers [Gou+20; Bel+19]. *All* these designs, except SATURNIN, use blocks of 128 bits or less.

**Quantum-secure modes of operation.** We have seen in Chapter 4 that many classically secure authentication modes suffer from polynomial-time attacks in the Q2 model. Since post-quantum security is one of our main goals, we choose the most efficient possible modes of operation with Q2 security.

**AES with bigger states.** Our first aim was to create a 256-bit block cipher with an efficient bitsliced implementation, that could be used on devices with little computing power, e.g., microcontrollers. At the same time, we took advantage of the knowledge obtained from the analysis of the [AES]. AES can safely be considered the most analyzed block cipher to date, in the classical setting. The results that we presented in Chapter 9 also provide a first security analysis in the quantum setting.

Saturnin can be seen as a 3-dimensional AES, in which the strongly-aligned version of the wide-trail strategy [Dae95; DR02a] can be applied. In short, the wide-trail strategy aims at minimizing low-weight differential and linear trails by choosing small S-Boxes with good differential and linear properties, and using a linear layer with a high branch number. Alignment between the linear and substitution layers allows to easily derive bounds on the number of active S-Boxes. In the AES, the composition of a state-wise permutation of the bytes and of a column-wise MDS transformation is a good compromise. These transformations will be adapted to our design.

This is not the first time that variants of AES with larger states have been proposed. In fact, the Rijndael family of block ciphers [DR99] included versions with larger states than 128 bits. However, these states were represented as a rectangle of bytes instead of a square, which made the diffusion relatively slower. Many attacks exploited this fact [GM08; MPP09; NP07; Zha+08; Wan+12; Sas10]. The variants of AES or Rijndael with longer keys than blocks are also relatively less secure with respect to related-key attacks [BKN09; BK09].

The applicability of the wide-trail strategy in more dimensions than 2 was studied in [DR02b]. In [Nak08], Nakahara Jr defined the block cipher 3D, an AES-like cipher in three dimensions with similar subroutines. However, our cipher was designed with lightness in mind, and all our operations benefit from an efficient bitsliced implementation. This means that our three-dimensional state can be projected onto 2 dimensions, and represented as a set of 32-bit registers.

Some hash function candidates of the SHA-3 competition, like Lane [Ind+08], Grøstl [Gau+08], and ECHO [Gil+08], used AES transformations (ShiftRows, MixColumns) with bigger internal states. However, these designs handle their state in a different way from Saturnin.

## 10.2 Specification of Saturnin

In this section, we specify the different components of the Saturnin suite[3]:

- Saturnin-CTR-Cascade is an authenticated cipher with associated data (AEAD) that follows the Encrypt-then-MAC composition [BN08]. Its encryption uses the Counter mode (CTR) and its MAC uses the Cascade construction [BCK96].
- Saturnin-Short is an authenticated cipher for messages of length smaller than 128 bits. It calls Saturnin only once.

[3] The detail of the specification, and the illustrations, are taken from [Can+20].

- Saturnin-Hash is a hash function with 256-bit output.

We note that among these modes, only Saturnin-Short requires the implementation of the inverse of Saturnin. We defer the security claims of Saturnin to the next section, in which we will define in detail what kind of classical and quantum access is allowed to the block cipher and the modes.

### 10.2.1 The Block Cipher Saturnin

Saturnin is a Substitution-Permutation Network with 256-bit blocks, a 256-bit key, and a 4-bit *domain separator*, which simplifies the layout of its modes of operation (they use the same cipher with different values of $D$). It has a 256-bit internal state. We define in this section a family of keyed permutations $\textsc{Saturnin}_R^D$ parameterized by:

- A number $R$ of *super-rounds*. $R$ belongs to $\{10, \ldots, 31\}$ and is equal to 10 by default; smaller values of $R$ can be used to define weakened versions of the cipher for analysis purposes)

- A *domain separator* $D \in \mathbb{F}_2^4$

$$\begin{cases} \textsc{Saturnin}_R^D : \mathbb{F}_2^{256} \times \mathbb{F}_2^{256} \to \mathbb{F}_2^{256} \\ \qquad\qquad\qquad \mathsf{X} \quad , \quad \mathsf{K} \mapsto \textsc{Saturnin}_R^D(\mathsf{X}, \mathsf{K}) \end{cases} \tag{10.1}$$

The key $\mathsf{K}$ in input is denoted the *master key*. Since $R = 10$ by default, we write $\textsc{Saturnin}^D$ for $\textsc{Saturnin}_{10}^D$.

#### 10.2.1.1 Internal State

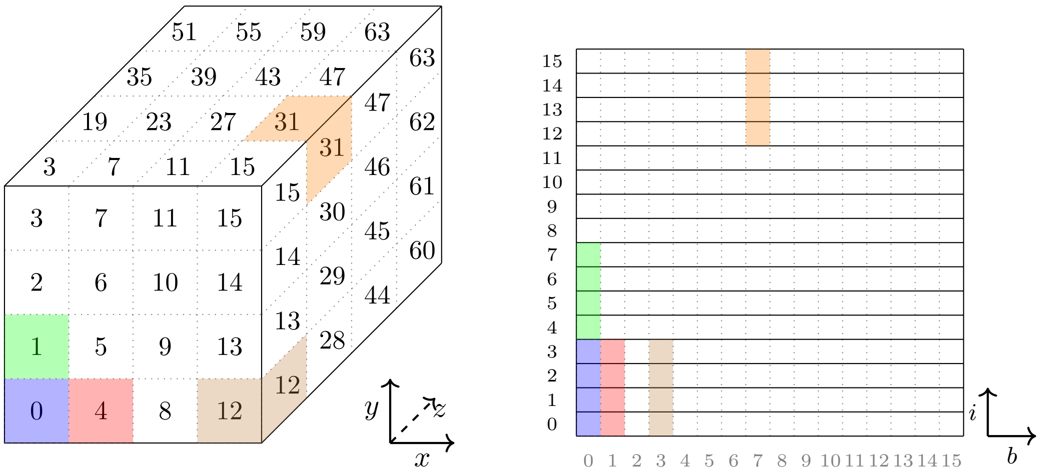
Round keys and internal states of Saturnin can be equivalently represented as:

- a $4 \times 4 \times 4$ cube of 4-bit nibbles: the 3-dimensional *abstract* representation (Figure 10.1, left);

- sixteen 16-bit registers, indexed from 0 to 15: the 2-dimensional *bitsliced* representation (Figure 10.1, right).

In this section, we use the abstract representation, as it allows to represent the operations of the cipher in a more intuitive way. In Section 10.2.2, we will show how to translate the operations on the cube into the bitsliced representation.

The numbering of the cube nibbles goes from 0 to 63, as displayed in Figure 10.1. Each nibble can also be defined by coordinates $(x, y, z)$ such that the coordinates $(x, y, z)$ correspond to the nibble with index $(y + 4x + 16z)$. We number the bits within each nibble from 0 to 3, where 0 is the least significant bit.



As a 4 × 4 × 4 cube of 4-bit nibbles. The boundaries between the nibbles are in gray.

As sixteen 16-bit registers. The indices and boundaries of the registers are in black, those of the bits are in gray.

**Figure 10.1:** The 3-dimensional and 2-dimensional representations of the 256-bit state of SATURNIN. Nibbles and their corresponding bits are represented with the same color in each representation.

**Subsets of the state.** The various transformations in SATURNIN are applied on different subsets of its state. Below, we define three such subsets of the cube, based on the same terminology as for parts of the Keccak-f state in SHA-3 (see Figure 1.1 in [Ber+11c]).

**Slice.** In the cube, a slice is a subset of the nibbles such that $z$ is constant.

**Sheet.** In the cube, a sheet is a subset of the nibbles such that $x$ is constant.

**Columns.** Columns are the intersection of a sheet and a slice. They correspond to the sets of nibbles with $x$ and $z$ constant.

### 10.2.1.2 Specification of the Block Cipher

The SPN in $\text{SATURNIN}_R^D$ has $2R$ rounds numbered from 0 to $2R - 1$. We define a *super-round* as the composition of two consecutive rounds with indices $2r$ and $(2r + 1)$. We now detail the operations of the cipher, acting on a a 256-bit internal state $\mathsf{X}$ and a 256-bit key $\mathsf{K}$, both represented as a $(4 \times 4 \times 4)$-cube of nibbles. Two additional 16-bit words $\mathsf{RC}_0$ and $\mathsf{RC}_1$ are used for generating the successive round constants. Pseudo-code describing a full block encryption is given in Algorithm 10.1.

**Initialization.** $\mathsf{X}$ and $\mathsf{K}$ are respectively initialized with the input and with the master key. Both 16-bit registers $\mathsf{RC}_0$ and $\mathsf{RC}_1$ are initialized as the bit-string

$$\underbrace{1\dots1}_{\text{7 ones}}\underbrace{R_4\dots R_0}_{R}\underbrace{D_3\dots D_0}_{D}\ ,$$

where the rightmost bit of the register is the least significant bit. The first four bits are given by the domain separator $\sum_{i=0}^{3} D_i 2^i = D$, while the 5-bit integer $\sum_{i=0}^{4} R_i 2^i$ is equal to the number of super-rounds $R$. Round 0 starts by XORing $\mathsf{K}$ to the internal state.

**Round function.** Each round, starting from Round 0, then successively applies the following transformations to the internal state:

- An **S-Box layer** $\mathsf{S}$, which applies the same 4-bit S-Box $\sigma_0$ to all nibbles with an even index, and the same 4-bit S-Box $\sigma_1$ to all nibbles with an odd index. These two S-Boxes are defined by their lookup tables which are given in Table 10.1, where $x$ such that $\sum_{i=0}^{3} x_i 2^i = x$ corresponds to a nibble containing $(x_3, x_2, x_1, x_0)$. An efficient implementation of the S-Boxes is shown in Figure 10.2.

- A **nibble permutation** $\mathsf{SR}_r$ which depends on the round number $r$. For all even rounds, $\mathsf{SR}_r$ is the identity function. For odd rounds of index $r$ with $r \bmod 4 = 1$, $\mathsf{SR}_r = \mathsf{SR}_{\text{slice}}$ consists of the parallel application of $R_{\text{slice}}$ on each slice independently. This operation maps the nibble with coordinates $(x, y, z)$ to $(x + y \mod 4, y, z)$. For odd rounds of index $r$ with $r \bmod 4 = 3$, $\mathsf{SR}_r = \mathsf{SR}_{\text{sheet}}$ consists of the parallel application of $R_{\text{sheet}}$ on each sheet independently. This operation maps the nibble with coordinates $(x, y, z)$ to $(x, y, z+y \mod 4)$. The $\mathsf{SR}_r$ transformation is depicted in Figure 10.4 and Figure 10.5.

- A **linear layer** $\mathsf{MC}$ composed of 16 copies of a linear operation $M$ over $(\mathbb{F}_2^4)^4$ which is applied in parallel to each column of the internal state. $M$ is defined as:

$$M : \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \mapsto \begin{pmatrix} \alpha^2(a) \oplus \alpha^2(b) \oplus \alpha(b) \oplus c \oplus d \\ a \oplus \alpha(b) \oplus b \oplus \alpha^2(c) \oplus c \oplus \alpha^2(d) \oplus \alpha(d) \oplus d \\ a \oplus b \oplus \alpha^2(c) \oplus \alpha^2(d) \oplus \alpha(d) \\ \alpha^2(a) \oplus a \oplus \alpha^2(b) \oplus \alpha(b) \oplus b \oplus c \oplus \alpha(d) \oplus d \end{pmatrix}$$

  where $a$ is the nibble with the lowest index, and $\alpha$ transforms the four bits $x_0, \dots, x_3$ of each nibble by the following multiplication

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} .$$

  This transformation corresponds to the next-state function of an LFSR of length 4, in Fibonacci mode, with feedback polynomial $X^4 + X^3 + 1$.

The transformation $M$ can be implemented efficiently as depicted in Figure 10.3 (which corresponds to Figure 13 in [DL18] up to a rotation of the nibbles). It is chosen among the mappings exhibited in [DL18] for its implementation cost.

- The **inverse** of the previous nibble permutation, namely $\mathsf{SR}_r^{-1}$.
- A **sub-key addition** at odd rounds only (i.e., at the end of each super-round). The sub-key is composed of the XOR of a round constant and either the master key or a rotated version of the master key:

**Round constant.** The round constants $\mathsf{RC}_0$ and $\mathsf{RC}_1$ are updated by clocking 16 times two independent LFSRs of length 16 in Galois mode with respective feedback polynomials $X^{16} + X^5 + X^3 + X^2 + 1$ and $X^{16} + X^6 + X^4 + X + 1$. In other words, we repeat 16 times the following operation: if the most significant bit of $\mathsf{RC}_i$ is 0, $\mathsf{RC}_i$ is replaced by $\mathsf{RC}_i \ll 1$, otherwise, it is replaced by $(\mathsf{RC}_i \ll 1)\texttt{\^{}poly}_i$ with $\texttt{poly}_0 = \texttt{0x1002d}$ and $\texttt{poly}_1 = \texttt{0x10053}$.

The two 16-bit words $\mathsf{RC}_0, \mathsf{RC}_1$ are then XORed to the internal state. Bit number $i$ in $\mathsf{RC}_0$ is added to Bit 0 of the nibble with index $4i$, for $0 \leqslant i \leqslant 15$. Similarly, Bit number $i$ in $\mathsf{RC}_1$ is added to Bit 0 of the nibble with index $(4i+2)$, for $0 \leqslant i \leqslant 15$.

**Round key.** If the round index $r$ is such that $r \bmod 4 = 3$, the master key $\mathsf{K}$ is XORed to the internal state; otherwise (i.e., when $r \bmod 4 = 1$), a rotated version of the key is added instead: the nibble with index $i$ receives the key nibble with index $(i + 20) \bmod 64$, for $0 \leqslant i \leqslant 63$.

**Table 10.1:** Lookup tables of the S-Boxes in Saturnin.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\sigma_0(x)$ | 0 | 6 | 14 | 1 | 15 | 4 | 7 | 13 | 9 | 8 | 12 | 5 | 2 | 10 | 3 | 11 |
| $\sigma_1(x)$ | 0 | 9 | 13 | 2 | 15 | 1 | 11 | 7 | 6 | 4 | 5 | 3 | 8 | 12 | 10 | 14 |

### 10.2.2 Operations on the Bitsliced Representation

The bitsliced or *register representation* is at the heart of the design of Saturnin. Although the specification that we gave above relies on the cube, as do many security arguments, all implementations of Saturnin will represent its internal state as sixteen 16-bit registers as shown in Figure 10.1. In this bitsliced representation, the bit of lowest weight of each register has index 0 and the bit of highest weight has index 15. A bit is then defined by the index $i$ of its register and its index $b$ within its register. The bits with coordinates $(4i, b), (4i + 1, b), (4i + 2, b)$ and $(4i + 3, b)$, which are therefore taken from registers $4i$ to $4i + 3$, correspond to nibble $(x, y, z) = (b \mod 4, i, \lfloor b/4 \rfloor)$ in the cube.

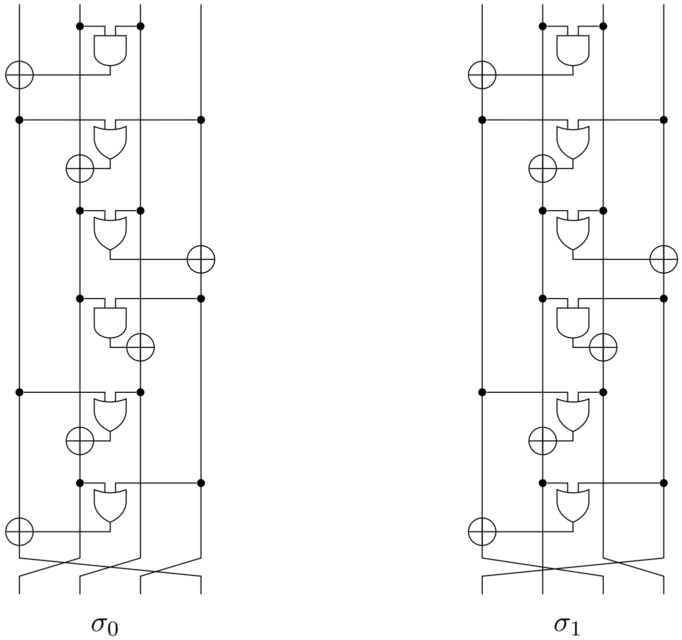


**Figure 10.2:** Bitslice implementation of the S-Boxes.

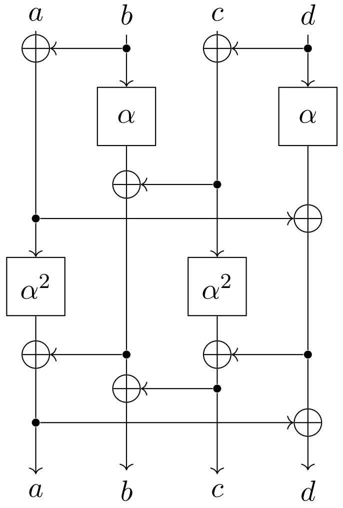


**Figure 10.3:** MDS mapping over 4 nibbles used in Saturnin. The input/output $(a, b, c, d)$ corresponds to nibbles with index $(4i, 4i + 1, 4i + 2, 4i + 3)$, for $0 \leqslant i \leqslant 15$.

Initial state:

| | | | |
|---|---|---|---|
| 3 | 7 | 11 | 15 |
| 2 | 6 | 10 | 14 |
| 1 | 5 | 9 | 13 |
| 0 | 4 | 8 | 12 |

| | | | |
|---|---|---|---|
| 19 | 23 | 27 | 31 |
| 18 | 22 | 26 | 30 |
| 17 | 21 | 25 | 29 |
| 16 | 20 | 24 | 28 |

| | | | |
|---|---|---|---|
| 35 | 39 | 43 | 47 |
| 34 | 38 | 42 | 46 |
| 33 | 37 | 41 | 45 |
| 32 | 36 | 40 | 44 |

| | | | |
|---|---|---|---|
| 51 | 55 | 59 | 63 |
| 50 | 54 | 58 | 62 |
| 49 | 53 | 57 | 61 |
| 48 | 52 | 56 | 60 |

Internal state after $\mathsf{SR}_r$ at Rounds $r$ with $r \equiv 1 \bmod 4$, that is, after $\mathsf{SR}_{\text{slice}}$:

| | | | |
|---|---|---|---|
| 7 | 11 | 15 | 3 |
| 10 | 14 | 2 | 6 |
| 13 | 1 | 5 | 9 |
| 0 | 4 | 8 | 12 |

| | | | |
|---|---|---|---|
| 23 | 27 | 31 | 19 |
| 26 | 30 | 18 | 22 |
| 29 | 17 | 21 | 25 |
| 16 | 20 | 24 | 28 |

| | | | |
|---|---|---|---|
| 39 | 43 | 47 | 35 |
| 42 | 46 | 34 | 38 |
| 45 | 33 | 37 | 41 |
| 32 | 36 | 40 | 44 |

| | | | |
|---|---|---|---|
| 39 | 43 | 47 | 35 |
| 42 | 46 | 34 | 38 |
| 45 | 33 | 37 | 41 |
| 32 | 36 | 40 | 44 |

Internal state after $\mathsf{SR}_r$ at Rounds $r$ with $r \equiv 3 \bmod 4$, that is, after $\mathsf{SR}_{\text{sheet}}$:

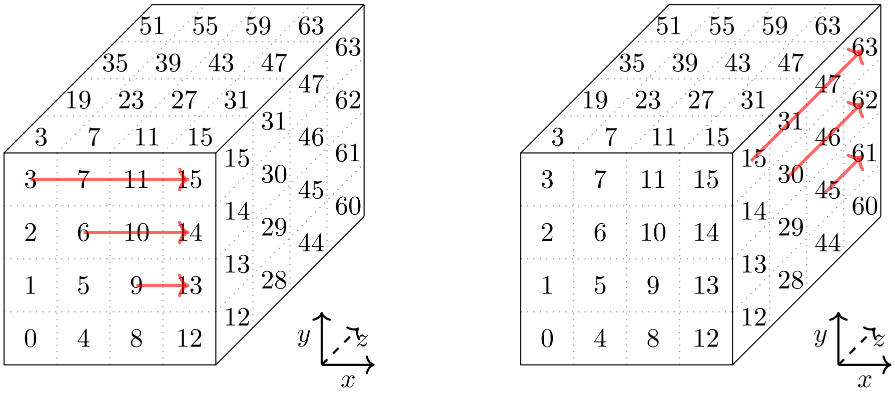
| | | | |
|---|---|---|---|
| 19 | 23 | 27 | 31 |
| 34 | 38 | 42 | 46 |
| 49 | 53 | 57 | 61 |
| 0 | 4 | 8 | 12 |

| | | | |
|---|---|---|---|
| 35 | 39 | 43 | 47 |
| 50 | 54 | 58 | 62 |
| 1 | 5 | 9 | 13 |
| 16 | 20 | 24 | 28 |

| | | | |
|---|---|---|---|
| 51 | 55 | 59 | 63 |
| 2 | 6 | 10 | 14 |
| 17 | 21 | 25 | 29 |
| 32 | 36 | 40 | 44 |

| | | | |
|---|---|---|---|
| 3 | 7 | 11 | 15 |
| 18 | 22 | 26 | 30 |
| 33 | 37 | 41 | 45 |
| 48 | 52 | 56 | 60 |

**Figure 10.4:** Ordering of the 64 nibbles of the internal state after applying $\mathsf{SR}_r$ depending on $r$ mod 4, when the cube is represented as the collection of its 4 slices.



$\mathsf{SR}_{\text{slice}}$ (when $r \equiv 1 \bmod 4$)

$\mathsf{SR}_{\text{sheet}}$ (when $r \equiv 3 \bmod 4$)

**Figure 10.5:** Representation of the $\mathsf{SR}_r$ operations on the cube.

**Algorithm 10.1** Saturnin block encryption

*In/Out:* $\mathsf{X} \in (\mathbb{F}_2^{16})^{16}$, $\mathsf{K} \in (\mathbb{F}_2^{16})^{16}$, $R \in \mathbb{N}$, $D \in \mathbb{F}_2^4$

```
$\mathsf{X} \leftarrow \mathsf{X} \oplus \mathsf{K}$
$\mathsf{RC}_0 \leftarrow$ 0xfe00$|(R \ll 4)|D$; $\mathsf{RC}_1 \leftarrow \mathsf{RC}_0$
for all $r$ from 0 to $R-1$ do
// First round of the super-round
    $\mathsf{X} \leftarrow \mathsf{S}(\mathsf{X})$
    $\mathsf{X} \leftarrow \mathsf{MC}(\mathsf{X})$
// Second round of the super-round
    $\mathsf{X} \leftarrow \mathsf{S}(\mathsf{X})$
    if $r \mod 2 \equiv 0$ then
        $\mathsf{X} \leftarrow \mathsf{SR}_{\text{slice}}(\mathsf{X})$
        $\mathsf{X} \leftarrow \mathsf{MC}(\mathsf{X})$
        $\mathsf{X} \leftarrow \mathsf{SR}_{\text{slice}}^{-1}(\mathsf{X})$
        $\mathsf{X} \leftarrow \mathsf{X} \oplus \mathsf{rot}(\mathsf{K})$
    else
        $\mathsf{X} \leftarrow \mathsf{SR}_{\text{sheet}}(\mathsf{X})$
        $\mathsf{X} \leftarrow \mathsf{MC}(\mathsf{X})$
        $\mathsf{X} \leftarrow \mathsf{SR}_{\text{sheet}}^{-1}(\mathsf{X})$
        $\mathsf{X} \leftarrow \mathsf{X} \oplus \mathsf{K}$
// Updating round constants
    for all $j$ from 0 to 15 do
        $\mathsf{RC}_0 \leftarrow \mathsf{clockLFSR}_0(\mathsf{RC}_0)$
        $\mathsf{RC}_1 \leftarrow \mathsf{clockLFSR}_1(\mathsf{RC}_1)$
// Constant addition
    $\mathsf{X}_0 \leftarrow \mathsf{X}_0 \oplus \mathsf{RC}_0$
    $\mathsf{X}_8 \leftarrow \mathsf{X}_8 \oplus \mathsf{RC}_1$
return $\mathsf{X}$
```

A *slice* in the cube corresponds to bits with indices $(i, b)$ such that $\lfloor b/4 \rfloor$ is constant, while a *sheet* corresponds to bits with indices $(i, b)$ such that $(b \mod 4)$ is constant. Therefore, the *columns* in the cube are composed of bits $(i, b)$ with a constant $b$ in the registers. The transformations involved in the round function of Saturnin are then implemented as follows.

**S-box layer.** The 4-bit S-boxes $\sigma_0$ and $\sigma_1$ are applied in parallel, each over a half of the whole state. As can be seen in Figure 10.2, these permutations are such that $\sigma_0 = \pi_0 \circ \sigma$ and $\sigma_1 = \pi_1 \circ \sigma$ where $\sigma$ is another S-Box, $\pi_0$ and $\pi_1$ are bit permutations. Their action on the index of the bits in each nibble is given in Table 10.2.

In the register representation, the S-Boxes are applied in a bitsliced fashion over all registers with indices $4i + 0, ..., 4i + 3$, where $i \in \{0, 2\}$ for $\sigma_0$ and $i \in \{1, 3\}$ for $\sigma_1$. The full parallel application of the S-Boxes is denoted $\mathsf{S}$ and, in the register representation,

**Table 10.2:** Correspondence between the input and output bit indices in $\pi_0$ and $\pi_1$.

| $i$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $\pi_0(i)$ | 3 | 0 | 1 | 2 |
| $\pi_1(i)$ | 2 | 1 | 3 | 0 |

it is summarized in Figure 10.6.

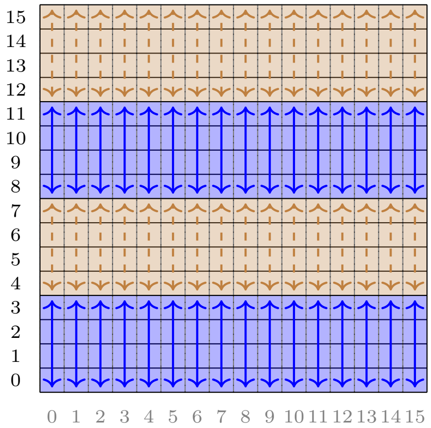


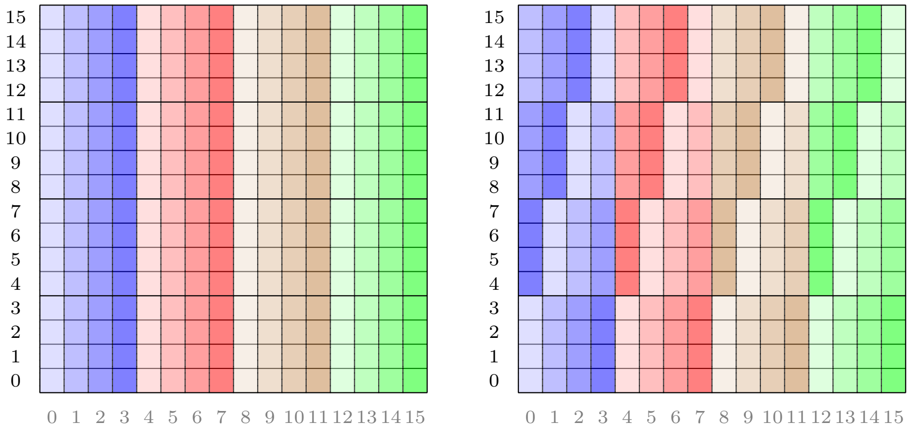
**Figure 10.6:** The application of the 4-bit S-Boxes (S) in the bitsliced representation. $\sigma_0$ is represented by continuous blue arrows, $\sigma_1$ by dashed brown ones.

**Nibble permutation in a slice.** In order to mix the columns in each slice, we use $\mathsf{SR}_{\text{slice}}$ which consists in the parallel application of $R_{\text{slice}}$ on each slice independently. This operation maps the nibble with coordinates $(x, y, z)$ to $(x + y \mod 4, y, z)$. Its implementation on 16-bit registers is summarized in Figure 10.7. It can be implemented by rotating each 4-bit word in register $i$ by $\lfloor i/4 \rfloor$.

**Nibble permutation in a sheet.** We proceed with sheets as we did with slices. In order to mix the columns in each sheet, we use $\mathsf{SR}_{\text{sheet}}$ which consists in the parallel application of $R_{\text{sheet}}$ on each sheet independently. This operation maps the nibble with coordinates $(x, y, z)$ to $(x, y, z + y \mod 4)$. It is summarized in Figure 10.8. It can be implemented with a rotation of the 16-bit register $i$ by $4\lfloor i/4 \rfloor$.

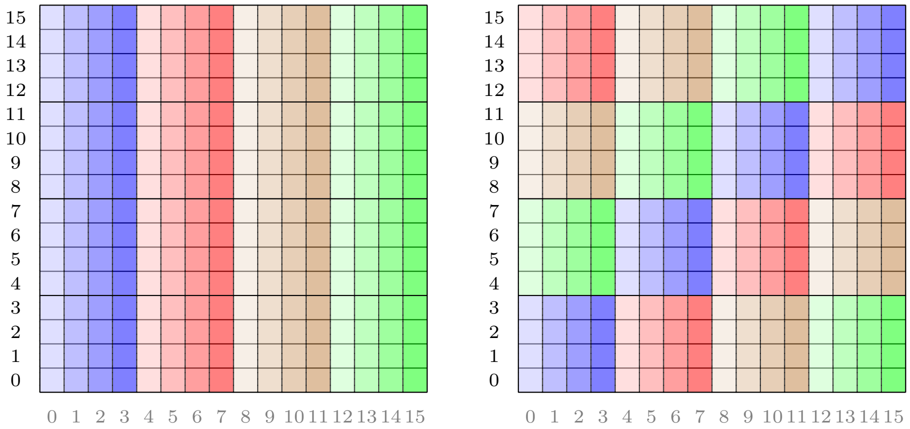
**Linear layer.** The matrix $M$ is defined over $(\mathbb{F}_{2^4})^4$ and is applied in parallel on each column of the state. In the register representation, it is applied in a bitsliced fashion over the whole state at once as summarized in Figure 10.9.

**Key addition.** The key addition is applied in a very straightforward way by XORing 16-bit registers in the key state with their counterparts in the internal state of the cipher.



Before. After.

**Figure 10.7:** The $\mathsf{SR}_{\text{slice}}$ operation that mixes the columns in each slice separately.



Before. After.

**Figure 10.8:** The $\mathsf{SR}_{\text{sheet}}$ operation that mixes the columns in each sheet separately.

In odd super-rounds, each register of the master key is rotated just before being added, meaning we can use operators combining the XOR and the rotation when available.

**Constant addition.** The bits in which the constants $\mathsf{RC}_0$ and $\mathsf{RC}_1$ are XORed correspond to registers 0 and 8 respectively. This operation is implemented using two word-wise XORs. The 16-bit state of each LFSR naturally corresponds to one 16-bit register.

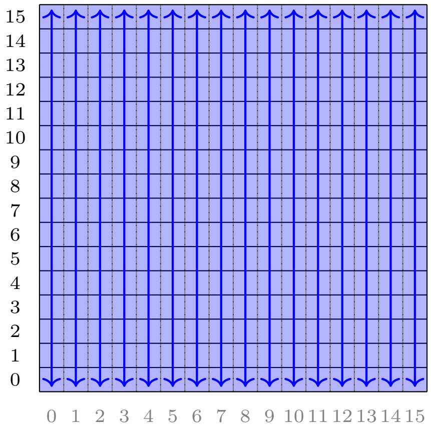


**Figure 10.9:** The parallel application of the 16 × 16 matrix $M$, i.e., the operation $\mathsf{MC}$.

### 10.2.3 The Authenticated Cipher Saturnin-CTR-Cascade

Saturnin-CTR-Cascade is an AEAD mode designed for use with Saturnin, based on the Encrypt-then-MAC composition [BN08]. It takes in input a key $\mathsf{K}$ of 256 bits, a nonce of (up to) 160 bits, a message $m$ and an AD $a$ of any length; it outputs a ciphertext $c$ of the same length and an authentication tag $t$ of 256 bits.

$$\begin{cases} \textsc{Saturnin}\text{-CTR-Cascade}: \\ \{0,1\}^{256} \times \{0,1\}^{160} \times \{0,1\}^{*} \times \{0,1\}^{*} \to \{0,1\}^{*} \times \{0,1\}^{256} \\ \quad \mathsf{K} \quad , \quad \text{nonce} \quad , \quad a \quad , \quad m \quad \mapsto \quad c \quad , \quad t \end{cases} \tag{10.2}$$

The mode is specified in Algorithm 10.2 and depicted in Figures 10.10, 10.11 and 10.12. We first encrypt using the Counter mode, then process the associated data and the ciphertexts by a MAC based on the Cascade construction [BCK96]. This MAC is very similar to an NMAC [BCK96] using Saturnin in Matyas-Meyer-Oseas (MMO) mode [MMO85].

When decrypting, the authentication tag is first recomputed, and compared with the received value; on mismatch, the encrypted message is rejected. If the authentication tags match, then decryption is identical to encryption. The comparison between authentication tags should endeavor not to reveal the bit position at which mismatched tags diverge.

In general, whenever our proposed modes require padding a value of less than 256 bits into a 256-bit block, we use the following padding rule, and denote as $\text{pad}(x)$ the padding of block $x$. We use the same rule to pad the nonce into the 161-bit string $N$.

**Definition 10.1** (Padding rule)**.** The padding rule consists in appending a single bit of value 1, followed by as many zeroes as necessary to reach the next block boundary.

In the Counter mode (CTR) depicted in Figure 10.10, we define a keystream by encrypting blocks composed of $N$ concatenated with a 95-bit counter which is increased by one for each new call: $z_i = \textsc{Saturnin}^1(k, N \| i+1)$ (the counter starts at 1 for the block $z_0$). The counter is encoded with the big-endian convention: if the counter numerical value $u$ is:

$$u = \sum_{i=0}^{94} u_i 2^i \ ,$$

then each bit $u_i$ becomes bit $255 - i$ in the input block. Thus, in a byte-oriented implementation and with a 128-bit nonce, the input to $\textsc{Saturnin}^1$ for the computation of $z_0$ will consist in the 16 bytes of the nonce, followed by a byte of value `0x80` (first padding byte for the nonce), followed by 14 bytes of value `0x00`, followed by one byte of value `0x01`.

**Algorithm 10.2** Saturnin-CTR-Cascade. All Saturnin calls use Algorithm 10.1 with $R = 10$ super-rounds. 5 domain separators (1, 2, 3, 4, 5) are used in total.

```
Input: message $m$, associated data $a$, nonce of 160 bits or less, key $k$
Requirements: nonces should not be reused, $m$ should have a length less than $2^{94} \times 256$ bits
Output: ciphertext $c$, tag $t$

Pad the nonce to obtain the 161-bit value $N$
Split $m$ into full blocks $m_0, m_1, \dots m_\ell$ and a final partial block $m_*$
Split $a$ into $a_0, a_1, \dots a_j, a_*$
for all $i = 0$ to $\ell$ do
    $c_i \leftarrow m_i \oplus \textsc{Saturnin}^1(k, N \| i+1)$        ▷ Encryption of block $i$
$c_* \leftarrow m_* \oplus \mathrm{trunc}_n(\textsc{Saturnin}^1(k, N \| \ell+2))$        ▷ Encryption of the final block
Where $n$ is the bit-length of $m_*$ and $\mathrm{trunc}_n$ truncates to the first $n$ bits
$t \leftarrow (N \| 0) \oplus \textsc{Saturnin}^2(k, N \| 0)$
for all $i = 0$ to $j$ do
    $t \leftarrow a_i \oplus \textsc{Saturnin}^2(t, a_i)$        ▷ Absorb AD block $i$
$t \leftarrow \mathrm{pad}(a_*) \oplus \textsc{Saturnin}^3(t, \mathrm{pad}(a_*))$        ▷ Absorb the final AD block
for all $i = 0$ to $\ell$ do
    $t \leftarrow c_i \oplus \textsc{Saturnin}^4(t, c_i)$        ▷ Absorb ciphertext block $i$
$t \leftarrow \mathrm{pad}(c_*) \oplus \textsc{Saturnin}^5(t, \mathrm{pad}(c_*))$        ▷ Absorb the final ciphertext block
return $c = (c_0 \| c_1 \| \dots \| c_\ell \| c_*), t$
```

### 10.2.4 The Authenticated Cipher Saturnin-Short

For some practical scenarios, we propose a second authenticated cipher[4] for handling messages of length strictly less than 128 bits (without additional data) and nonces of size up to 128 bits: Saturnin-Short (Figure 10.13). We adopt the same padding convention as in Section 10.2.3 for messages shorter than 128 bits.

[4] Codename: *poussin*.

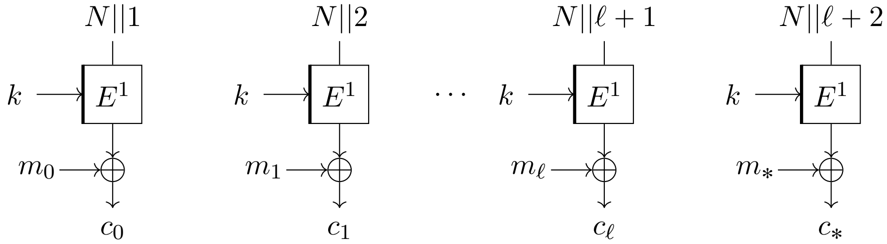


**Figure 10.10:** SATURNIN-CTR encryption, where $E^i$ denotes SATURNIN with domain separator $i$. A thick line represents the input of the key.

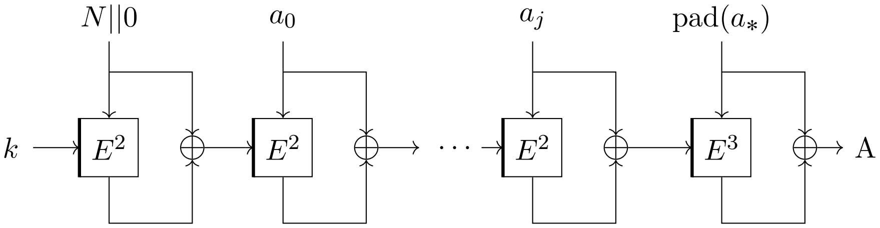


**Figure 10.11:** SATURNIN-Cascade, processing of the associated data, where $E^i$ denotes SATURNIN with domain separator $i$. A thick line represents the input of the key.

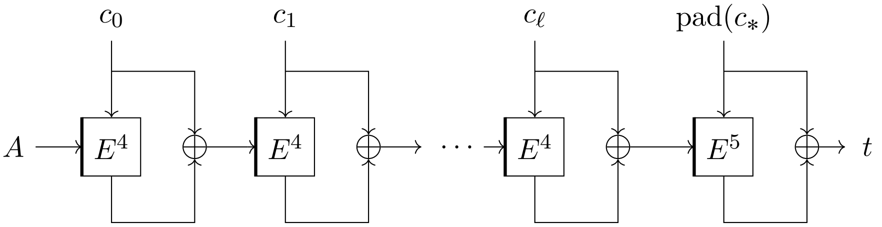


**Figure 10.12:** SATURNIN-Cascade, processing of the ciphertext and computation of the tag, where $E^i$ denotes SATURNIN with domain separator $i$. A thick line represents the input of the key.

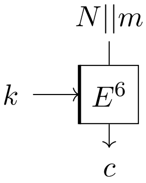


**Figure 10.13:** SATURNIN-Short with key $k$, message $m$ and nonce $N$. A thick line represents the input of the key.

In SATURNIN-Short, we use the fact that SATURNIN has a 256-bit block size, allowing to mix together the nonce and the message. We use the variant SATURNIN$^6$. It is worth noticing that the ciphertext and the tag are not two separate values. Given a nonce $N$ and a ciphertext $c$, the tag can be verified by deciphering $c$ and comparing the left half with $N$.

Since SATURNIN-Short does not allow additional data, we recommend that protocols based on SATURNIN-Short use a counter as the nonce to prevent reordering of the ciphertexts and to detect replay attacks.

### 10.2.5 The Hash Function Saturnin-Hash

We also propose a 256-bit hash function[5] based on SATURNIN with the Merkle-Damgård construction [Mer89; Dam89], as shown in Figure 10.14. We use SATURNIN$_{16}$ (i.e., SATURNIN with 16 super-rounds[6]) because we want the compression function to be resistant to related-key attacks. The compression function uses the MMO [MMO85] mode ($f(c_i, m) = E_{c_i}(m) \oplus m$) to compress a 256-bit chaining value $c_i$ and a 256-bit message block $m$. Therefore, SATURNIN-Hash (Algorithm 10.3) is similar to the above Cascade, except that there is no key as starting point but a fixed value, 0.

$$\begin{cases} \textsc{Saturnin}\text{-Hash} : \{0,1\}^* & \rightarrow \{0,1\}^{256} \\ \qquad\qquad\qquad m & \mapsto \quad t \end{cases} . \tag{10.3}$$

### 10.2.6 Values of the Domain Separator

We summarize the values of $D$ used in the different modes in Table 10.3. SATURNIN$^D$ can either be seen as a small family of independent block ciphers or as a tweakable block cipher with tweak $D$.

### 10.2.7 Security Claims

Having entirely described the members of the SATURNIN suite, we formulate security claims. In general, we claim that for the block cipher and the constructions, there is no attack *significantly*[7] better than the generic ones, *either classical or quantum.* In this section, we will use the term *attack* in an informal way; more details will be given in Section 10.3. The success probability of an attack is denoted $p$. Its time complexity $\mathcal{T}$, data complexity $\mathcal{D}$ and memory complexity $\mathcal{M}$ are counted in the following way.

**Classical Complexities.** Classically, $\mathcal{T}$ is expressed in units equivalent to the cost of one evaluation of the involved SATURNIN block cipher (either SATURNIN$_{10}$ or SATURNIN$_{16}$ for the primary designs). $\mathcal{D}$ is the number of encryption and decryption queries, in

[5] Codename: *parmentier de canard.*

[6] Codename: *faturnin.*

[7] Thus putting aside biclique cryptanalysis and the accelerations of exhaustive search.

**Algorithm 10.3** SATURNIN-Hash. All SATURNIN calls use Algorithm 10.1 with $R = 16$ super-rounds. 2 domain separators (6, 7) are used.

```
Input: message $m$
Output: hash output $t$
Split $m$ into full blocks $m_0, m_1, \dots m_\ell$ and a final partial block $m_*$
                                        ▷ $m_*$ may be empty, but never full
$t \leftarrow 0$                          ▷ the all-zero 256-bit block
for all $i = 0$ to $\ell$ do
    $t \leftarrow m_i \oplus \text{SATURNIN}^7_{16}(t, m_i)$
$t \leftarrow \text{pad}(m_*) \oplus \text{SATURNIN}^8_{16}(t, \text{pad}(m_*))$
return $t$
```

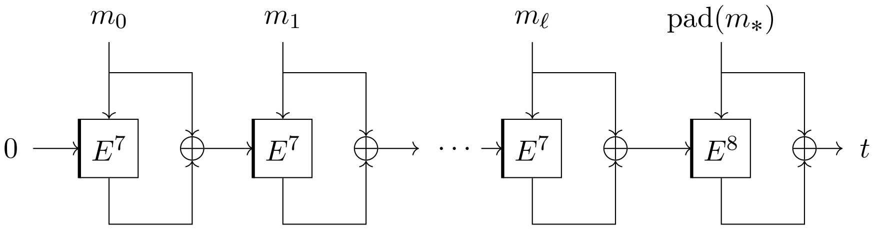


**Figure 10.14:** The hash function SATURNIN-Hash, where $E^i$ is SATURNIN$_{16}$ with domain separator $i$. A thick line represents the input of the key.

**Table 10.3:** Correspondence between the value of the domain separator $D$ and the usage of SATURNIN$^D$.

| Value of $D$ | Use |
|---|---|
| 0 | SATURNIN block cipher |
| 1 | SATURNIN-CTR |
| 2 | SATURNIN-Cascade AD |
| 3 | SATURNIN-Cascade AD final |
| 4 | SATURNIN-Cascade message |
| 5 | SATURNIN-Cascade message final |
| 6 | SATURNIN-Short |
| 7 | SATURNIN-Hash |
| 8 | SATURNIN-Hash final |

number of 256-bit blocks of message and AD (that is, a query of $\ell$ blocks counts $\ell$). It is worth noticing that $\mathcal{T} \geqslant \mathcal{D}$ since the time for generating the data must be taken into account.

**Quantum Complexities.** $\mathcal{T}$ is expressed in units equivalent to the cost of one quantum circuit for Saturnin, written with the Clifford+T gate set, without qRAM access (see Section 1.2). This makes our security levels independent of the actual optimization of a quantum circuit for Saturnin, which would be required to run a Grover search for the keys. Our quantum adversaries are Q2. When attacking the block cipher with a secret key, $\mathcal{D}$ is simply the number of encryption or decryption queries, with superposed messages. When attacking Saturnin-Hash, there is no notion of data. When attacking Saturnin-CTR-Cascade or Saturnin-Short, we give the attacker superposition access to the message *and* AD. She can also call a decryption oracle in superposition (which will return $\perp$ unless the tag is valid). In both cases, *nonces and message lengths remain classical*, and nonces are not repeated. This means that the adversary actually accesses a family of quantum oracles $O_{\mathrm{Enc}_k}^{N,\mathrm{ml},\mathrm{al}}$ for (padded) nonces $N$, message bit-lengths ml and AD bit-lengths al, and similarly, a family of quantum oracles $O_{\mathrm{Dec}_k}^{N,\mathrm{ml},\mathrm{al}}$, with the restriction that a given $N$ can be used only once. Before each query, a message length, AD length and a nonce are chosen. $\mathcal{D}$ is the sum of the number of 256-bit blocks of message and AD (full or partial) over all oracle queries. These quantum oracles also come with a time cost, which is the number of single-block evaluations of Saturnin inside the construction.

**Hybrid Claims.** We consider that classical oracle calls cost the same as quantum ones. On the one hand, classical queries are a particular type of quantum queries, so they should be at least easier to make. On the other hand, assuming the converse means that quantum queries are cheap, and this not only simplifies, but also strengthens our claims. Some generic attacks that can target our modes (e.g., BHT collision search, see Section 2.3) are hybrid, and may run differently if different costs are assigned to classical and quantum queries.

**Block cipher.** By default, Saturnin denotes the block cipher with at least 10 super-rounds, i.e., 20 single-rounds. Saturnin$_{16}$ corresponds to the block cipher with 16 super-rounds. For related-key attacks, we consider a small number of related keys derived under the conditions of [BK03], which ensure that no generic attack exists.

**Claim 1** (Security of the block cipher Saturnin)**.**
- *There exists no* ***classical*** *attack in the single-key setting with* $\frac{\mathcal{T}}{p} < 2^{224}$. *Saturnin$_{16}$ provides a similar security level against related-key attacks involving a small number of keys.*
- *There exists no* ***quantum*** *attack in the single-key setting with* $\frac{\mathcal{T}^2}{p} < 2^{224}$.

**Authenticated encryption.** In the following security claims, $\tau$ is the length of the tags in bits (256 by default, but it can be less if the tags are truncated). We do not claim security in nonce-misuse or nonce-repetition scenarios. As mentioned above, we do not consider nonce-superposition scenarios, and our Q2 adversaries choose also a *classical* nonce value.

**Claim 2** (Security of Saturnin-CTR-Cascade)**.**
- *There exists no* ***classical*** *attack satisfying* $\frac{\mathcal{D}^2+\mathcal{T}+\mathcal{D}2^{256-\tau}}{p} < 2^{224}$.
- *There exists no* ***quantum*** *attack satisfying* $\frac{\mathcal{D}^3+\mathcal{T}^2+\mathcal{D}^2 2^{256-\tau}}{p} < 2^{224}$.

**Claim 3** (Security of Saturnin-Short)**.**
- *There exists no* ***classical*** *attack with* $\frac{\mathcal{D}^2+\mathcal{T}+\mathcal{D}2^{128}}{p} < 2^{224}$.
- *There exists no* ***quantum*** *attack with* $\frac{\mathcal{D}^3+\mathcal{T}^2+\mathcal{D}^2 2^{128}}{p} < 2^{224}$.

We also claim security against classical related-key attacks (with the same restrictions on the related keys as for Claim 1) when Saturnin is replaced by Saturnin$_{16}$.

**Hash function.** Saturnin-Hash is based on Saturnin$_{16}$. In what follows, $\mathcal{M}_q$ is the size of the quantum memory measured in registers of 256 qubits. In these claims we consider $p \geqslant 1/2$. We assume of course that $\mathcal{M}_q \geqslant 1$. The claim for collisions implies that there is no quantum attack with $\mathcal{T} < 2^{75}$, because we necessarily have $\mathcal{M}_q < \mathcal{T}$.

**Claim 4** (Security of Saturnin-Hash)**.**
- *There exists no* ***classical*** *collision attack with* $\mathcal{T} < 2^{112}$. *There exists no* ***classical*** *second-preimage attack with* $\mathcal{T} < 2^{224-\ell}$ *for messages of length* $2^{\ell}$. *There exists no* ***classical*** *preimage attack with* $\mathcal{T} < 2^{224}$.
- *There exists no* ***quantum*** *collision attack satisfying* $\mathcal{T}^5 \times \mathcal{M}_q < 2^{448}$. *There exists no* ***quantum*** *second-preimage attack with* $\mathcal{T} < 2^{112-\ell/2}$ *for messages of length* $2^{\ell}$. *There exists no* ***quantum*** *preimage attack with* $\mathcal{T} < 2^{112}$.

## 10.3 Security of the Modes of Operation

The available literature on post-quantum modes of operation conditioned our choice for Saturnin. In this section, we introduce some standard classical and quantum security definitions and show how the security of the modes of operation used in Saturnin-CTR-Cascade, Saturnin-Hash and Saturnin-Short can be reduced formally to that of the block cipher. For convenience, we recall some notations: nonces are denoted $N$, messages are denoted $m$, ADs are denoted $a$ and their respective bit-length ml and al.

Throughout this section, we define an *adversary* as a *quantum* algorithm, that accesses one or more oracles, classical or quantum. In the latter case, we consider standard oracles without loss of generality. Thus, although we give some classical definitions, we consider a quantum attacker making classical queries (Q1) in this context. The quantum definitions that we introduce afterwards model Q2 adversaries. Note that a quantum adversary is free to call unkeyed primitives, such as SATURNIN-Hash or SATURNIN itself, in superposition.

## 10.3.1 Interface and Security Goals for an AEAD Scheme

Our primary target is SATURNIN-CTR-Cascade, which is modeled as a nonce-based AEAD scheme $(\mathsf{Encrypt}_k, \mathsf{Decrypt}_k)$ of signature:

$$\begin{cases} \mathsf{Encrypt}_k : \{0,1\}^\nu \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^\tau \\ \qquad\qquad N \quad , \quad a \quad , \quad m \mapsto c \quad , \quad t \\ \mathsf{Decrypt}_k : \{0,1\}^\nu \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^\tau \to \{\top, \bot\} \times \{0,1\}^* \cup \bot \\ \qquad\qquad N \quad , \quad a \quad , \quad c \quad , \quad t \mapsto \text{Accept} \quad , \quad m \end{cases}$$

where $k$ is taken from some key space $\mathcal{K}$. Accept is $\top$ if and only if the tag recomputed from $N, a, c$ matches $t$, and in that case, the ciphertext is correctly decrypted, otherwise $\mathsf{Decrypt}_k$ returns $\bot, \bot$. Alternatively, $N$ can be considered to be part of the ciphertext (since it must be transmitted alongside).

We use game-based classical and quantum security definitions, whose goal is to show that, under some assumptions, an adversary cannot win some security game with more than non-negligible advantage. More modern classical proofs consider instead the ability of the adversary to distinguish between a *real world* and an *ideal world*, in which she interacts with different oracles. We keep the former definitions for their proximity to the quantum definitions that we will introduce later. Furthermore, we adapt all definitions to the nonce-based case (the literature often considers randomness instead of nonces, with no practical difference for us).

### 10.3.1.1 Example: IND-CPA

We define the *IND-CPA security game*, in which an adversary $\mathcal{A}$ interacts with a *challenger*.

**IND-CPA game**

**Key generation:** $k \xleftarrow{\$} \mathcal{K}, b \xleftarrow{\$} \{0, 1\}$.

**Challenge queries:** $\mathcal{A}$ chooses two message / AD pairs $a^0, m^0$, $a^1, m^1$, a nonce $N$ that has not been used before; the challenger responds with $\mathsf{Encrypt}_k\left(N, a^b, m^b\right)$.

**Encryption queries:** $\mathcal{A}$ chooses a message / AD $a, m$, a nonce $N$ that has not been used before; the challenger responds with $\mathsf{Encrypt}_k\left(N, a, m\right)$.

**Guess:** $\mathcal{A}$ produces a bit $b'$ and wins if $b = b'$.

On this simple example, we will define the *advantage* of $\mathcal{A}$ and the IND-CPA security of $\mathsf{Encrypt}_k$.

**Definition 10.2.** The IND-CPA advantage of an adversary $\mathcal{A}$ making $q$ queries to $\mathsf{Encrypt}_k$ is defined as:

$$\mathbf{Adv}_{\text{IND}-\text{CPA}}\left(\mathcal{A}, q\right) = \left| 2 \Pr_{k \xleftarrow{\$} \mathcal{K}}\left(\mathcal{A} \text{ wins}\right) - 1 \right|$$

and the IND-CPA advantage is the maximum over all adversaries making $q$ queries:

$$\mathbf{Adv}_{\text{IND}-\text{CPA}}\left(q\right) = \max_{\mathcal{A} \text{ with } q \text{ queries}} \left| 2 \Pr_{k \xleftarrow{\$} \mathcal{K}}\left(\mathcal{A} \text{ wins}\right) - 1 \right| .$$

A scheme defined as above is said *indistinguishable under chosen-plaintext attacks* (IND-CPA) if the IND-CPA advantage in function of $q$ is small (i.e., an exponential number of queries is required to make it constant). More generally, we let $\mathbf{Adv}^{O}_{\text{Game}}\left(q, \mathcal{T}\right)$ denote the maximal advantage of adversaries running in Game, making $q$ queries to an oracle $O$ and running in time $\mathcal{T}$, and we give bounds on this quantity.

#### 10.3.1.2 Unforgeability and IND-CCA2.

Indistinguishability under Adaptive Chosen-ciphertext Attacks (IND-CCA2) is an enhanced version in which the adversary is allowed encryption and decryption queries, but cannot decrypt one of the challenge queries.

**IND-CCA2 game**

**Key generation:** $k \overset{\$}{\leftarrow} \mathcal{K}, b \overset{\$}{\leftarrow} \{0,1\}$.

**Challenge queries:** $\mathcal{A}$ chooses two message / AD pairs $a^0, m^0$, $a^1, m^1$, a nonce $N$ that has not been used before; the challenger responds with $\mathsf{Encrypt}_k\left(N, a^b, m^b\right)$.

**Encryption queries:** $\mathcal{A}$ chooses a message / AD $a, m$, a nonce $N$ that has not been used before and; the challenger responds with $\mathsf{Encrypt}_k\left(N, a, m\right)$.

**Decryption queries:** $\mathcal{A}$ chooses a ciphertext / tag $c, t$; the challenger responds with $\mathsf{Decrypt}_k\left(c, t\right)$ *if* $c, t$ was not the result of a challenge query, and $\perp$ otherwise.

**Guess:** $\mathcal{A}$ produces a bit $b'$ and wins if $b = b'$.

Notice that we remove the nonce from the decryption call, and consider that it is part of the ciphertext. Thus we follow the definition where the encryption is randomized and the decryption is deterministic. Next, we define Strong Unforgeability under Chosen-Message Attacks (SUF-CMA) for a nonce-based MAC scheme.

**SUF-CMA game**

**Key generation:** $k \overset{\$}{\leftarrow} \mathcal{K}$.

**MAC queries:** $\mathcal{A}$ chooses a message $m$, a nonce $N$ that has not been used before and queries the MAC.

**Verification queries:** $\mathcal{A}$ chooses a message $m$, a tag $t$. The challenger returns $\top$ if the tag is valid and $\perp$ otherwise. The adversary wins if she queries here a pair which is not the result of a previous MAC query.

Bellare and Namprempre [BN08] studied several compositions of a MAC and an encryption scheme in the classical setting: MAC-then-encrypt (used in TLS 1.0), Encrypt-and-MAC (used in SSH), and encrypt-then-MAC (used in IPSec). The latter is defined from an encryption scheme $E_k, D_k$ (randomized or stateful) and a MAC scheme $M_k, V_k$ as: $\mathsf{Encrypt}_k\left(N, m\right) = E_k(N, m), M_k(N, E_k(N, m))$.

**Theorem 10.1** (From [BN08])**.** *Encrypt-then-MAC is IND-CCA secure if the underlying encryption scheme is IND-CPA secure and the MAC is SUF-CMA secure:*

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{Encrypt}_k,\mathsf{Decrypt}_k}_{\mathrm{IND-CCA}}\left(q_e, q_d, \mathcal{T}\right)\\ &\qquad\qquad\leqslant 2\cdot\mathbf{Adv}^{M_k,V_k}_{\mathrm{SUF-CMA}}\left(q_e, q_d, \mathcal{T}\right) + \mathbf{Adv}^{E_k}_{\mathrm{IND-CPA}}\left(q_e, \mathcal{T}\right)\quad .\end{aligned}$$

In addition to this strong security reduction, the encrypt-then-MAC composition allows to reject forgeries without even decrypting the ciphertext.

#### 10.3.1.3 IND-qCPA

Indistinguishability against quantum chosen-plaintext attacks (IND-qCPA) is one of the first quantum security notions introduced in the literature. It takes into account some of the Q2 attacks recalled in Chapter 4. We adapt the definition given by Boneh and Zhandry [BZ13b], as it considers initially an encryption scheme based on randomness. Hereafter, IND-qCPA will denote the adaptation for a nonce.

In this game, superposition access to the messages is allowed. As remarked in Section 10.2.7, *nonces, message and AD lengths remain classical values.* The encryption oracle $\mathsf{Encrypt}_k$ is replaced by a family of standard oracles parameterized by a choice of nonce and input length. For simplicity, we omit these additional parameters and denote by $O_{\mathsf{Encrypt}_k}$ any of these oracles. As in the classical game, nonces are not repeated.

**IND-qCPA game**

**Key generation:** $k \xleftarrow{\$} \mathcal{K}, b \xleftarrow{\$} \{0,1\}$.

---

**Challenge queries:** $\mathcal{A}$ chooses two message / AD pairs of same lengths $a^0, m^0,\ a^1, m^1$, a nonce $N$ that has not been used before; the challenger responds with $\mathsf{Encrypt}_k\left(N, a^b, m^b\right)$.

**Encryption queries:** $\mathcal{A}$ chooses a message and AD length, a nonce $N$ that has not been used before and calls $O_{\mathsf{Encrypt}_k}$ on her input registers.

**Guess:** $\mathcal{A}$ produces a bit $b'$ and wins if $b = b'$.

In this game, the *encryption queries* are quantum, while the *challenge queries* are classical. Intuitively, IND-qCPA security prevents key-recovery attacks based on Simon's algorithm; if the adversary finds the key, then it is able to distinguish between the classical challenges. Conversely, modes which are known not to be IND-qCPA secure usually admit such powerful breaks.

Both the power and the limits of the IND-qCPA definition can be illustrated on encryption modes that XOR a pseudo-random keystream to their input. On the positive side, it is very easy to prove the IND-qCPA security of such a mode.

**Proposition 10.1** (From [Ana+16])**.** *Let* $\mathsf{Encrypt}_k$ *be a mode defined by:*

$$\mathsf{Encrypt}_k\left(N, m\right) = m \oplus f(k, N)$$

*where* $f$ *is an extendable-output function. Then:*

$$\mathbf{Adv}^{\mathsf{Encrypt}_k}_{\mathrm{IND-qCPA}}\left(q, \mathcal{T}\right) \leqslant \mathbf{Adv}^{\mathsf{Encrypt}_k}_{\mathrm{IND-CPA}}\left(q, \mathcal{T}\right) \quad . \tag{10.4}$$

*In other words, if* $\mathsf{Encrypt}_k$ *is IND-CPA secure, then it is IND-qCPA secure.*

*Proof.* Given an adversary $\mathcal{A}$ for the IND-qCPA game, we construct an adversary $\mathcal{B}$ for the IND-CPA game with the same advantage, runtime and number of queries. $\mathcal{B}$ simulates $\mathcal{A}$. Whenever $\mathcal{A}$ makes a superposition query to $O_{\mathsf{Encrypt}_k(N,\cdot)}$, $\mathcal{B}$ queries classically $\mathsf{Encrypt}_k\left(N, 0\right) = f(k, N)$ and maps $|m\rangle |y\rangle$ to $|m\rangle |y \oplus m \oplus f(k, N)\rangle$, which perfectly simulates the query. Thus, the final state of $\mathcal{B}$ is the same as the final state of $\mathcal{A}$, and they have the same advantage. □

On the negative side, we have seen in Chapter 4 that the Deutsch-Jozsa algorithm allows to distinguish such a mode with Q2 queries (Proposition 4.1), even though this distinguisher does not contradict the IND-qCPA security. This means that the adversary cannot obtain from the Q2 queries an information that would help her win the IND-CPA security game, for example the secret key.

#### 10.3.1.4 Unforgeability and IND-qCCA

Defining a quantum notion of unforgeability is a bigger challenge than IND-qCPA, because the output forgery must be that of a message previously unqueried, while the queries are made in superposition. Again, the literature considers deterministic MACs without nonces, and we adapt the definitions to our setting.

Boneh and Zhandry [BZ13a] proposed the following definition: the adversary makes $q$ quantum queries and shouldn't be able to output $q + 1$ valid $\{\text{message}, \text{tag}\}$ pairs with more than negligible probability. However, Alagic, Majenz, Russell, and Song [Ala+20b] noticed a caveat: it is possible to build a MAC such that a part of its message space contains a secret value that enables the adversary to find a valid $\{\text{message}, \text{tag}\}$ outside this space. Such a pair should be intuitively regarded as a forgery. A pathological example of such a MAC scheme can be found in [Ala+20b]. To date, no practical BU-secure and BZ-insecure MAC is known.

In order to overcome this issue, they define the Blind-Unforgeability (BU) security game. The challenger chooses a *random blinding* $B_\varepsilon$, which specifies a fraction of the message space in which the adversary *cannot* query the MAC. The candidate forgery must then occur in this space. The BU game is a quantum version of the classical EUF-CMA game, to which it reduces if only classical queries are considered. Note that in the definition of [Ala+20b] the adversary is not allowed superposition access to the verification oracle.

In our case, the nonce-based MAC $M_k$ will be replaced by a "blinded" version $B_\varepsilon M_k$:

$$\begin{cases} B_\varepsilon M_k : \{0,1\}^* \times \{0,1\}^\nu \to \{0,1\}^\tau \\ \qquad (N,m) \;,\; m \notin B_\varepsilon \mapsto M_k(N,m) \\ \qquad (N,m) \;,\; m \in B_\varepsilon \mapsto \perp \end{cases} .$$

**BU game**

**Setup:** the challenger picks a random key $k$ and a random blinding $B_\varepsilon$

**MAC queries:** $\mathcal{A}$ chooses a nonce $N$ that has not been used before, and performs a call to the unitary $O_{B_\varepsilon M_k}$ on her registers.

**Forgery attempt:** $\mathcal{A}$ chooses a nonce $N$ that has not been used before, and produces a candidate forgery $(m,N),t$. The adversary wins if $V_k(m,N,t) = \top$ and $m \in B_\varepsilon$.

While BU-unforgeability avoids the problematic case explained in [Ala+20b], it also implies the security notion of Boneh and Zhandry (BZ) [BZ13a].

**Proposition 10.2** (Theorem 13 in [Ala+20b])**.** *A BU-secure MAC is BZ-secure.*

**Composition results.** The reason for proving IND-qCPA and BU-security together is that these two notions should imply indistinguishability of under quantum chosen-ciphertext attacks. A definition of IND-qCCA2 is given by Boneh and Zhandry in [BZ13b]. In [SJS16], Soukharev, Jao, and Seshadri proved that IND-qCPA and BZ imply IND-qCCA2 for Encrypt-then-MAC, as in the classical setting. However, in [CEV20], Chevalier, Ebrahimi, and Vu state that the proof contains a loophole, and incorrectly assumes the ability to record some quantum queries. A similar composition result is proven in [CEV20], but for a stronger security notion than IND-qCPA, which allows the one-time pad attack.

### 10.3.2 qPRFs and qPRPs

The basic security that we expect from Saturnin (with 10 or 16 super-rounds) is to be a strong *quantum pseudorandom permutation* (sqPRP) in the following sense.

**Definition 10.3** (Strong qPRP advantage)**.** Let $E_k$ be a family of permutations of $\{0,1\}^n$ indexed by $k \in \mathcal{K}$. The strong qPRP distinguishing advantage against $E_k$ is:

$$\mathbf{Adv}^{E_k}_{\mathrm{sqPRP}}(q, \mathcal{T}) = \max_{\mathcal{A}} \left| \Pr_{k \xleftarrow{\$} \mathcal{K}} \left( \mathcal{A}^{O_{E_k^\pm}}(q) = 1 \right) - \Pr_{\Pi \xleftarrow{\$} \mathrm{Perm}(\{0,1\}^n)} \left( \mathcal{A}^{O_{\Pi^\pm}}(q) = 1 \right) \right|$$

where $\mathcal{A}^O(q)$ is a *distinguisher*, i.e., a quantum adversary making $q$ queries to its oracle $O$ and outputting a bit $b$; $E_k^\pm$ and $\Pi^\pm$ are functions of $n+1$ bits, that either call the permutation or its inverse.

Note that it is equivalent, up to a constant factor in the number of queries, to have access to two separate oracles $O_{E_k}$ and $O_{E_k^{-1}}$, or to a single oracle $O_{E_k^{\pm}}$.

**Definition 10.4** (Strong qPRP)**.** $E_k$ is a *strong qPRP* if $\mathbf{Adv}_{\text{sqPRP}}^{E_k}(q, \mathcal{T})$ is negligible for polynomial-time adversaries. It is a qPRP if the same holds without the inverse queries.

With this definition, the expected security of Saturnin (Claim 1) can be formulated more precisely:

$$\forall q, \mathcal{T}, \mathbf{Adv}_{\text{sqPRP}}^{\textsc{Saturnin}_k}(q, \mathcal{T}) \leqslant \frac{\mathcal{T}^2}{2^{224}} \enspace . \tag{10.5}$$

This bound is matched by a secret key-recovery using Grover's algorithm. A stronger requirement is to behave as a (quantum) ideal cipher [HY18], which we expect from $\textsc{Saturnin}_{16}$. In this model, the cipher behaves as a family of independent permutations drawn at random.

**qPRF distinguishers.** Similarly to a qPRP, it is possible to define a *quantum pseudorandom function* (qPRF).

**Definition 10.5** (qPRF advantage)**.** Let $F_k$ be a family of functions from $\{0,1\}^n$ to $\{0,1\}^m$, indexed by $k \in \mathcal{K}$. The qPRF distinguishing advantage against $F_k$ is:

$$\mathbf{Adv}_{\text{qPRP}}^{F_k}(q, \mathcal{T}) = \max_{\mathcal{A}} \left| \Pr_{k \xleftarrow{\$} \mathcal{K}} \left( \mathcal{A}^{O_{F_k}}(q) = 1 \right) - \Pr_{G \xleftarrow{\$} \{\{0,1\}^n \to \{0,1\}^m\}} \left( \mathcal{A}^{O_G}(q) = 1 \right) \right| \enspace .$$

**Definition 10.6** (qPRF)**.** $F_k$ is a qPRF if $\mathbf{Adv}_{\text{qPRF}}^{F_k}(q, \mathcal{T})$ for polynomial-time adversaries is negligible.

Notice that in both these definitions, while the adversary queries her oracle in superposition, $k$ is always a classical value. The following proposition makes qPRFs particularly interesting for us.

**Proposition 10.3** ([Ala+20b])**.** *Let $F_k$ be a qPRF. Then $F_k$ is a BU-secure MAC.*

So far, all MACs with quantum unforgeability that can be found in the literature are actually quantum PRFs. This is the case of NMAC, HMAC and the Cascade construction, which were studied in [SY17]. Thus, we will obtain BU-unforgeability thanks to Proposition 10.3.

### 10.3.3 Security Reductions

We are now sufficiently equipped to prove the security of our modes of operation.

#### 10.3.3.1 Indistinguishability of Saturnin-CTR

We prove the following property of our $\textsc{Saturnin}_k$-CTR encryption. Notice that in this bound, $\mathbf{Adv}_{\mathrm{PRP}}^{\textsc{Saturnin}_k}(\mathcal{T}, q)$ is the distinguishing advantage of *quantum* adversaries running in time $\mathcal{T}$ and making $q$ *classical* queries to $\textsc{Saturnin}_k$ and its inverse, which is weaker than qPRP security.

**Proposition 10.4.** *if $\textsc{Saturnin}_k$ is a PRP, then $\textsc{Saturnin}_k$-CTR is IND-qCPA:*

$$\mathbf{Adv}_{\mathrm{IND-qCPA}}^{\textsc{Saturnin}_k - CTR}(q, \mathcal{T}) \leqslant 2 \cdot \mathbf{Adv}_{\mathrm{PRP}}^{\textsc{Saturnin}_k}(q, \mathcal{T}) + \frac{q^2}{2^{257}} \ .$$

*Proof.* The classical security of the CTR mode has been studied in [Bel+97]. The proof assumes that for any fixed key, the inputs of block cipher calls are never reused [Nat01]. This is the case in $\textsc{Saturnin}$-CTR-Cascade, since we concatenate the nonce with a block counter. By Theorem 13 in [Bel+97], if $\textsc{Saturnin}$ is replaced by a pseudorandom function $F_k$:

$$\mathbf{Adv}_{\mathrm{IND-CPA}}^{F-CTR}(q, \mathcal{T}) \leqslant 2 \cdot \mathbf{Adv}_{\mathrm{PRF}}^{F}(q, \mathcal{T}) \tag{10.6}$$

where $q$ denotes the total number of block cipher calls in the queries. Next, the PRP-PRF switching lemma can be used (Proposition 8 in [Bel+97]) which states that it is impossible to distinguish $\textsc{Saturnin}_k$ from a PRF, up to a birthday bound.

$$\mathbf{Adv}_{\mathrm{PRF}}^{\textsc{Saturnin}}(q, \mathcal{T}) \leqslant \mathbf{Adv}_{\mathrm{PRP}}^{\textsc{Saturnin}}(q, \mathcal{T}) + \frac{q^2}{2^{256+1}} \ . \tag{10.7}$$

Furthermore, this bound is independent of the time $\mathcal{T}$. Indeed, one cannot distinguish a function from a permutation unless outputting a collision of this function. Thus, in the classical setting, we have:

$$\mathbf{Adv}_{\mathrm{IND-CPA}}^{\textsc{Saturnin}-CTR}(q, \mathcal{T}) \leqslant 2 \cdot \mathbf{Adv}_{\mathrm{PRP}}^{\textsc{Saturnin}}(q, \mathcal{T}) + \frac{q^2}{2^{257}} \ . \tag{10.8}$$

The result in the quantum setting follows by Proposition 10.1:

$$\mathbf{Adv}_{\mathrm{IND-qCPA}}^{\textsc{Saturnin}-CTR}(q, \mathcal{T}) \leqslant \mathbf{Adv}_{\mathrm{IND-CPA}}^{\textsc{Saturnin}-CTR}(q, \mathcal{T}) \ . \qquad \square$$

#### 10.3.3.2 Unforgeability of Saturnin-Cascade

We define the compression function $h(k, m) = \textsc{Saturnin}_k(m) \oplus m$, where $\textsc{Saturnin}$ is used with a domain separator equal to 2, 3, 4 or 5 (but for simplicity, we do not consider the different domain separators, nor associated data). We define the *length-$\ell$ Cascade* $H^\ell$ by:

$$H_k^\ell(m_0, \dots x_\ell) = h(\dots h(h(k, m_0), m_1) \dots m_\ell)) \ .$$

Then, we have the following.

**Proposition 10.5** (Theorem 5.1 in [SY17])**.** *If $h(k, \cdot)$ is a qPRF, then $H_k^\ell$ is also a qPRF:*

$$\mathbf{Adv}_{\mathrm{qPRF}}^{H_k^\ell}(q) \leqslant 34 \ell q^{3/2} \sqrt{\mathbf{Adv}_{\mathrm{qPRF}}^{h(k, \cdot)}(4q)} \ .$$

*Remark* 10.1. The proofs given in [SY17] do not seem tight, and they do not match the best upper bound known.

It remains to show that if Saturnin is a qPRP, then $h(k\cdot)$ is a qPRF of $k$.

**Lemma 10.1.** *Let $E_k$ be a block cipher with block size $n$. Then for any number of queries $q$ and time $\mathcal{T}$:*

$$\mathbf{Adv}_{\text{qPRF}}^{m \mapsto E_k(m) \oplus m}(q, \mathcal{T}) \leqslant \mathbf{Adv}_{\text{qPRP}}^{E_k}(q, \mathcal{T}) + \mathcal{O}\left(\frac{q^3}{2^n}\right) .$$

*Proof.* First of all, we show that:

$$\mathbf{Adv}_{\text{qPRF}}^{m \mapsto E_k(m) \oplus m}(q, \mathcal{T}) \leqslant \mathbf{Adv}_{\text{qPRF}}^{E_k}(q, \mathcal{T}) .$$

Indeed, if $\mathcal{A}$ is a qPRF distinguisher for $m \mapsto E_k(m) \oplus m$, we can create a qPRF distinguisher $\mathcal{B}$ for $E_k$ as follows: $\mathcal{B}$ runs $\mathcal{A}$. Whenever $\mathcal{A}$ queries $m \mapsto E_k(m) \oplus m$, $\mathcal{B}$ queries $E_k(m)$ and XORs $m$. If the oracle is a random function, feedforwarding keeps the output random and the result is unchanged.

Next, we show that $\mathbf{Adv}_{\text{qPRF}}^{E_k}(q, \mathcal{T}) \leqslant \mathbf{Adv}_{\text{qPRP}}^{E_k}(q, \mathcal{T}) + \mathcal{O}\left(\frac{q^3}{2^n}\right)$. This is the quantum PRP-PRF switching lemma, shown in [Zha15]: random functions are indistinguishable from random permutations up to the quantum birthday bound. □

Combining this with Proposition 10.3, we obtain that if Saturnin is a qPRP, then Saturnin-Cascade is a BU-secure MAC.

#### 10.3.3.3 Saturnin-Short

In [BR00a], the authors prove the security (assuming a strong PRP) of an encode-then-encrypt construction. Saturnin-Short can be seen as such, where the encoding corresponds to appending the nonce $N$, which is transmitted with the message. With small modifications, confidentiality and authenticity come from Theorems 4.1 and 4.2 in [BR00a]. Quantumly, Saturnin$_k$-Short is a qPRF of $k$, which implies security against forgeries by Proposition 10.3.

#### 10.3.3.4 Saturnin-Hash

Contrary to Saturnin-CTR-Cascade and Saturnin-Short, the security of Saturnin-Hash does not stem from the security of Saturnin as a PRP (resp. qPRP), but as an ideal cipher, which is why the mode Saturnin-Hash uses a version of Saturnin with more rounds.

Classically, the security of Merkle-Damgård is related to that of the compression function. Ours uses the Matyas–Meyer–Oseas (MMO) mode, similar to the Cascade: $h_{i+1} = \text{Saturnin}_{h_i}(m_i) \oplus m_i$. It is dual to the Davies–Meyer construction, which injects the message block $m_i$ into the key, and would give rather $h_{i+1} = \text{Saturnin}_{m_i}(h_i) \oplus h_i$ [PGV93].

In [Zha19], Zhandry proves the quantum indifferentiability of the Merkle-Damgård construction if it uses a prefix-free encoding (hence a good padding), and if the underlying compression function is a random function. The compression function we use here is: $h(x, y) = \textsc{Saturnin}_x(y) \oplus y$. If Saturnin is a quantum ideal cipher by the definition of [HY18], then $h$ cannot be distinguished from a random function. Consequently, Zhandry's proof applies.

### 10.3.4 Attacks against the Modes of Operation

We summarize here the best generic attacks known against Saturnin-CTR-Cascade, Saturnin-Short and Saturnin-Hash, explaining the rationale behind the claims of Section 10.2.7. These attacks use either search or collision search with a limited number of queries, or quantum collision search with a limited memory. We do not consider parallelization or multiple targets.

**Classical attacks against Saturnin-CTR-Cascade.** Let $\tau$ be the tag length and $p$ the probability of success of a classical adversary against Saturnin-CTR-Cascade in breaking its confidentiality, integrity or authenticity. We claim:

$$p < \frac{\mathcal{T}}{2^{256}} + \frac{\mathcal{D}^2}{2^{256}} + \mathcal{D}2^{-\tau} \ ,$$

where the adversary is entitled to encryption or verification queries of a total of $\mathcal{D}$ blocks, and $\mathcal{T}$ computation time.

Among these terms, $\frac{\mathcal{T}}{2^{256}}$ accounts for the exhaustive search of the key, which breaks the PRP security of Saturnin. The term $\frac{\mathcal{D}^2}{2^{256}}$ accounts for finding a collision on an internal state. The term $\mathcal{D}2^{-\tau}$ accounts for asking the verification queries of random ciphertexts and tags. Each of them has a probability $2^{-\tau}$ of being accepted: the verifier will decipher and recompute the corresponding tag, ending up with a random string that must match the adversary's tag. Hence the probability of success after $\mathcal{D}$ verification queries is $\mathcal{D}2^{-\tau}$.

**Quantum attacks against Saturnin-CTR-Cascade.** With the same notations as above, we claim:

$$p < \frac{\mathcal{T}^2}{2^{256}} + \frac{\mathcal{D}^3}{2^{256}} + \mathcal{D}^2 2^{-\tau} \ ,$$

where the adversary is entitled to encryption or verification queries of a total of $\mathcal{D}$ blocks (in superposition), and $\mathcal{T}$ computation time.

The term $\frac{\mathcal{T}^2}{2^{256}}$ accounts for quantum search of the block cipher key, with limited time (see Lemma 2.3). The term $\frac{\mathcal{D}^3}{2^{256}}$ accounts for finding a collision on 256 bits after $\mathcal{D}$ queries. Finally, the term $\mathcal{D}^2 2^{-\tau}$ accounts for making the verifier accept a random tag, using Grover's algorithm with the verifier as oracle.

**Attacks against Saturnin-Short.** In SATURNIN-Short, the generic attacks are slightly different from SATURNIN-CTR-Cascade. Although the tag is not truncated to 128 bits, the attack using verification queries applies as if it were the case. Indeed, a forgery using verification queries amounts to finding $c, N$ such that $E_k^{-1}(c)$ contains $N$ on its 128 right bits. $N$ is not queried in superposition, and the adversary has only access to the verification result, not the value. So this amounts to look for $2^{256}$ good elements (sound pairs $c, N$) among $2^{256+128}$ (all choices) with the verifier as oracle. Classically, the probability of success after $\mathcal{D}$ trials is $\mathcal{D}2^{-128}$.

Quantumly, Grover's algorithm speeds up the search for a random ciphertext and nonce giving a good verification result. With $\mathcal{D}$ superposition verification queries, the success probability is $\mathcal{D}^2 2^{-128}$, so everything happens as if the tag was actually truncated to 128 bits.

**Attacks against Saturnin-Hash.** For finding preimages, the best attacks are classical or quantum exhaustive search, which give the bounds in the claims. Our security claim against collision search takes into account the quantum memory available to the attacker. We detail the rationale of this claim.

Given a standard oracle $O_h$ for a random function $h : \{0,1\}^n \rightarrow \{0,1\}^n$, the BHT algorithm (Algorithm 2.10 in Section 2.3) reaches a time-memory trade-off curve $\mathcal{T}^2 \cdot \mathcal{M} = 2^n$ until $\mathcal{M} = 2^{n/3}$ using QRACM. Algorithm 5.1, presented in Section 5.1 reaches the same curve until $\mathcal{M} = 2^{n/5}$, without QRACM. By subsuming these two results, as shown in Figure 10.15, we can make the following conjecture.

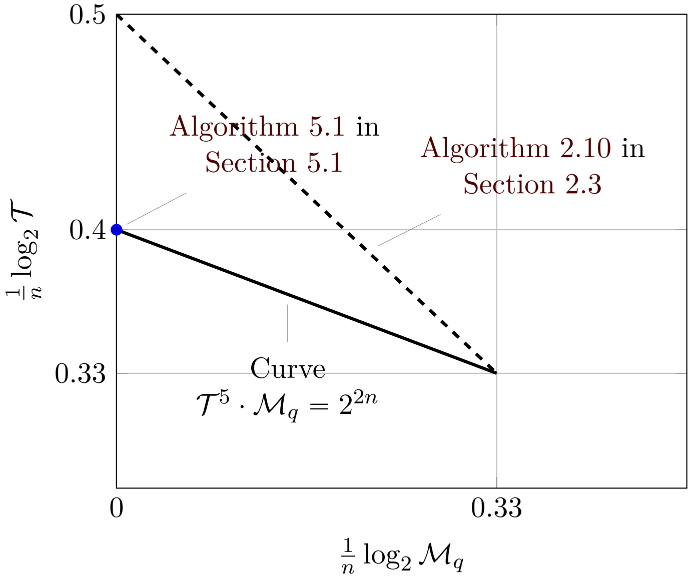


**Figure 10.15:** Quantum collision search with limited quantum memory: putting together the algorithms of Section 2.3 and Section 5.1 in a single (maybe non-tight) curve.

**Conjecture 10.1.** *There exists no quantum algorithm that, on input a random function $h$, outputs a collision of $h$, runs in time $\mathcal{T}$ and uses a quantum memory $\mathcal{M}_q$, with: $\mathcal{T}^5 \cdot \mathcal{M}_q < 2^{2n}$.*

Conjecture 10.1 formalizes the idea that if one tries to go below the point $\mathcal{T} = 2^{2n/5}$, a massive quantum memory has to be used (we shall not specify which type, but QRACM seems sufficient). This conjecture is implicit in the collision security claim of Saturnin-Hash, which takes into account the amount of quantum memory given to the adversary. Thus, should a better quantum collision algorithm disprove Conjecture 10.1, it could also break the security claim.

Otherwise, the best time achievable is (roughly) $2^{85}$ with a QRACM of size $2^{85}$, given by Algorithm 2.10. Without qRAM, the best quantum time known is $2^{102}$, given by Algorithm 5.1.

## 10.4 Rationale of the Block Cipher

We have previously specified the block cipher Saturnin and its bitsliced representation, which motivates some of the design choices. In this section, we explain the rest of the design by trying to obtain the best resistance against linear, differential, algebraic, invariant and related-key attacks. We introduce notably the *Super S-Box* representation of Saturnin, which will be used in the security analysis of Section 10.5.

### 10.4.1 Super S-Box Representation

The general structure of Saturnin mimics the AES, which is arguably the best understood symmetric primitive. Let us denote by $\mathsf{S}_{16}$ the 16-bit Super S-Box in Saturnin, i.e., the permutation of $\mathbb{F}_2^{16}$ composed of the succession of an S-Box layer, the linear MDS function $M$, and a second S-Box layer. The composition of two rounds of indices $(2r, 2r+1)$ can then be seen as the application of this Super S-Box to each column of the internal state, followed by $\mathsf{SR}_{2r+1}^{-1} \circ \mathsf{MC} \circ \mathsf{SR}_{2r+1}$.

- If $r$ is even, then the slices are invariant under $\mathsf{SR}_{2r+1}$. This implies that the linear layer $\mathsf{SR}_{2r+1}^{-1} \circ \mathsf{MC} \circ \mathsf{SR}_{2r+1}$ consists of the concatenation of four copies of the same function $\mathsf{L}_{64}$ of $(\mathbb{F}_2^{16})^4$, which applies to the slices independently.

- If $r$ is odd, then the sheets are invariant under $\mathsf{SR}_{2r+1}$. In this case, the linear layer consists of the concatenation of four copies of the same function $\mathsf{L}_{64}$, which applies to the sheets independently. Moreover, it is easy to prove, e.g., by Theorem 1 in [Alb+14], that $\mathsf{L}_{64}$ has branch number 5 with respect to $\mathbb{F}_2^{16}$.

Let us then represent in a $4 \times 4$ matrix $\mathsf{C}$ the 16-bit words $\mathsf{C}_{0,0}, \dots, \mathsf{C}_{3,3}$ corresponding to the 16 columns of the cube, where $\mathsf{C}_{i,j}$ corresponds to the column defined by $x = i, z = j$. This means that each slice in the cube is a column of the matrix $\mathsf{C}$, while each sheet in the cube is a row of $\mathsf{C}$. When $r$ is even, the linear function $\mathsf{L}_{64}$ applies to the columns of $\mathsf{C}$ independently, while for odd $r$, it applies to its rows. In other words, a super-round has the following structure: the S-box $\mathsf{S}_{16}$ is applied to each 16-bit word, then $\mathsf{L}_{64}$ is applied to the four columns of $\mathsf{C}$, and the matrix $\mathsf{C}$ is transposed.

Saturnin is then very similar to an AES operating on 16-bit words (*supernibbles*) except that the ShiftRows transformation is replaced by a transposition, exactly as it was in Square [DKR97], a predecessor of the AES. The key-schedule of this cipher is also very simple. The subkey at even rounds is the master key $\mathsf{K}$. At odd rounds, the key is shifted by 5 bytes and transposed.

This Super S-Box representation will help us analyze the resistance of Saturnin to the main classes of attacks.

### 10.4.2 Design of the Super-S-Box

We recall the following standard definitions.

**Definition 10.7** (Linearity)**.** The *Walsh transform* of an S-Box $S$ is defined as:

$$\forall \alpha, \beta \in \mathbb{F}_2^{16}, \beta \neq 0, \widehat{S}(\alpha, \beta) = \sum_{x \in \mathbb{F}_2^{16}} (-1)^{\beta \cdot S(x) + \alpha \cdot x} \ .$$

The *linearity* of $S$ is the highest magnitude taken by its Walsh transform.

**Definition 10.8** (Differential Uniformity)**.** The *differential uniformity* of an S-Box $S$ is defined as the largest number of solutions $x$ to the equation:

$$S(x \oplus \alpha) \oplus S(x) = \beta, \alpha, \beta \in \mathbb{F}_2^{16}, \beta \neq 0 \ .$$

A standard requirement of cryptographic designs is to lower at best the linearity and differential uniformity of their subcomponents, in order to resist linear and differential cryptanalysis. This constrains the design of the Super S-Box in Saturnin.

The two S-Boxes $\sigma_0$ and $\sigma_1$, which apply to the nibbles with an even index and with an odd index respectively, are the composition of the same S-Box $\sigma$, followed by two different permutations of the output bits given in Table 10.2.

**Choice of $\sigma$.** The 4-bit S-Box $\sigma$ has been chosen among the 4-bit S-Boxes with optimal cryptographic parameters, i.e., in one of the equivalence classes named *optimal* in the classification established by Leander and Poschmann [LP07]. These S-Boxes are those with differential uniformity 4 and linearity 8. Moreover, we chose for $\sigma$ an S-Box minimizing the number of operations in the bitslice implementation, both for $\sigma$ and $\sigma^{-1}$ in order to compute efficiently the inverse cipher for decryption with Saturnin-Short.

**Design of $\mathsf{S}_{16}$.** The MDS matrix $M$ has the property of mapping the subspace of $\mathbb{F}_{2^4}^4$ defined by $\{(x, x, 0, 0), x \in \mathbb{F}_{2^4}\}$ onto the subspace $\{(y, y, 0, y), y \in \mathbb{F}_{2^4}\}$. This implies that, if the nonlinear layer in $\mathsf{S}_{16}$ uses four copies of the same S-Box $\sigma$, then $\mathsf{S}_{16}$ transforms the affine subspace of dimension 4 $\{(x, x, \sigma^{-1}(0), \sigma^{-1}(0)), x \in \mathbb{F}_{2^4}\}$ into the affine subspace

$\{(y, y, \sigma(0), y), y \in \mathbb{F}_{2^4}\}$. The propagation of this simple 4-dimensional subspace is a strong property that could be cryptanalytically exploited. Furthermore, if we base the Super S-Box $\mathsf{S}_{16}$ only on the single $\sigma$, we obtain a linearity equal to $2^{12} = 4096$, which is higher than what we could achieve.

Thus, we use the two different S-Boxes $\sigma_0$, applied to the nibbles with an even index, and $\sigma_1$, applied to the nibbles with an odd index. We define both from $\sigma$ using a pair of bit permutations $(\pi_0, \pi_1)$ of the four output bits. $(\pi_0, \pi_1)$ were chosen among all possible pairs as to minimize the differential uniformity (80) and linearity (3072) of the corresponding $\mathsf{S}_{16}$.

### 10.4.3 Key Schedule and Round Constants

In the Super S-Box representation of Saturnin, the key schedule corresponds to shifting the supernibbles by 5 positions (with the standard AES byte numbering). This increases the resistance of Saturnin against related-key attacks. A detailed analysis is given in [Can+19].

As for the round constants, there are $2^9$ possible seeds. They generate $2^9$ sequences of values of $\mathsf{RC}_0, \mathsf{RC}_1$. Between any pair of such sequences, seen as two pairs of 512-bit words $(x, y), (x', y')$, there does not exist $t_0 < 496$ such that $\forall t \geqslant 0, x_t = x'_{t+t_0}$ and $y_t = y'_{t+t_0}$. In other words, the states of the LFSRs between position 0 in the first sequence and position $t_0$ in the second cannot collide.

## 10.5 Security of the Block Cipher

As explained in Section 10.4.1, the structure of the Saturnin block cipher is very similar to the structure of an AES operating on 16-bit words, replacing the AES S-Box by the Saturnin Super S-Box. Saturnin then benefits from the 20-year cryptanalytic effort against the AES and from the quantum security analysis done in Chapter 9. For most attacks, a super-round in Saturnin offers a resistance similar to a single round in AES-128 (relatively to the different block sizes).

As AES-128 has 10 rounds, and still benefits from a 3-round security margin, 10 super-rounds (i.e., 20 rounds) seem to be a natural choice. In [Can+19], we give a key-recovery attack attack on 7.5 Saturnin super-rounds, slightly more than the AES, using the simplified key-schedule of Saturnin. This leaves currently a 2.5-super-round security margin. 8 super-rounds might be reachable by exploiting further the key-schedule, but we shall leave this as an open question.

We did not perform related-key attacks, but the available literature on the related-key cryptanalysis of the AES (including full AES-256) motivates us to choose more super rounds when related-key security is necessary.

### 10.5.1 Security of the Block Cipher against Classical Attacks

**Differential cryptanalysis.** The Super S-Box $S_{16}$ has differential uniformity 80, so the highest probability for a non-trivial differential is $80 \times 2^{-16} = 2^{-9.68}$. The AES structure guarantees that any four consecutive rounds have at least 25 active Super S-Boxes. This implies that the best differential characteristics over 4 super-rounds and 8 super-rounds have probability at most $2^{-241.9}$ and $2^{-483.9}$ respectively. Moreover, it is worth noticing that the proportion of differentials for the Super S-Box with probability higher than $2^{-10}$ is very small since there are only 110 such differentials (among $2^{32}$); moreover, all these 110 differentials have exactly five active nibbles.

**Linear cryptanalysis.** The *linearity* of the Super S-Box is equal to 3072, or equivalently the maximal squared correlation for a linear relation equals $3072^2 \times 2^{-32} = 2^{-8.83}$. By the same arguments as previously, we deduce that the highest squared correlation for a linear trail (*linear potential*) over 4 rounds and over 8 rounds is at most $2^{-220.7}$ and $2^{-441.5}$ respectively.

**Algebraic degree.** It is estimated in [Can+19], using standard arguments, that the full algebraic degree of Saturnin is reached after five super-rounds.

**Bicliques.** The exhaustive search attack with bicliques [BKR11] always allows to gain a small factor against the baseline exhaustive key search, by testing all the keys faster than an evaluation of the cipher. It is applicable to Saturnin, as to any cipher, with a likely transposition of the results previously obtained for AES. This does not contradict our security claims.

**Impossible differential attacks.** Impossible differential attacks were introduced by Knudsen [Knu94] and by Biham, Biryukov and Shamir [BBS99]. As we saw in Chapter 9, they provide some of the best known attacks on reduced-round AES [Bou+18], together with the Demirci-Selçuk MITM attacks [DS08]. To date, the best impossible differential attack on AES-128 [Bou+18] targets 7 rounds and it requires, with $2^{105}$ chosen plaintexts, a time of $2^{106.88}$ and memory of $2^{74}$. We estimate that a similar impossible differential attack can be applied to 7 super-rounds of Saturnin, with twice these complexity exponents. As the key-schedule of Saturnin is simpler than that of the AES, it may be possible to extend this attack to 7.5 super-rounds, or even 8 super-rounds, which would be a very impressive result.

**DS-MITM attacks.** A classical DS-MITM attack similar to the ones of Section 9.6 is given in [Can+19]. This is currently the best key-recovery on reduced-round Saturnin, reaching 7.5 super-rounds.

**Subspace trails.** Recent distinguishers on 5-round AES [Sun+16; GRR17; RBH17; Gra18] and improved attacks on reduced-round versions [Bar+18] have used the existence of *subspace trails*: two linear subspaces $U$ and $V$ of states such that the image by the round function of any coset of $U$ is included in a coset of $V$. Such subspace trails occur for two rounds of AES. A limitation on the existence of such trails was given in [LTW18]: if the S-Box does not have a linear structure (which is the case of Saturnin's Super S-Box), then subspace trails are the direct product of subspace trails of the single S-Box, i.e., the trivial $\{0\}$ or $\mathbb{F}_2^{16}$. Thus Saturnin behaves exactly as the AES with respect to subspace trails, and a distinguisher based on this property cannot reach more than five super-rounds.

### 10.5.2 Security of the Block Cipher Against Quantum Attacks

Quantum exhaustive search of the key requires approximately $2^{256/2} = 2^{128}$ iterations, each of which calls a quantum embedding of Saturnin. A quantum circuit for Saturnin may cost likely less than the AES, since the AES S-Box, its most costly quantum component, has been replaced by 4-bit S-Boxes only (see Section 9.2).

The analysis of the AES done in Chapter 9 can be transferred to Saturnin. In particular, a quantum version of the AES-128 Square attack can target up to 6 super-rounds of Saturnin in the Q1 setting. Counting the time spent in Super S-Boxes instead of AES S-Boxes, we take the square of the numbers in Table 9.6.

**Proposition 10.6.** *There exists a quantum algorithm that retrieves the key of a secret-key oracle for Saturnin with 6 super-rounds, using approximately $2^{35}$ classical chosen-plaintext queries, $2^{89}$ Super S-Boxes, $2^{50}$ QRAQM registers of 256 bits, $2^{72}$ classical registers of 256 bits.*

Without QRAQM, the quantum version of the Square attack cannot use the partial sums technique, and it reaches a higher time than the classical version. Thus, with these preliminary results: • with or without QRAQM, the highest number of super-rounds broken in the quantum setting (with a complexity lower than $2^{128}$ quantum circuits for Saturnin) is 6; • without QRAQM, the best quantum attack is the *classical* 6-round Square attack with partial sums of [Fer+00]; • these results are unchanged if Q2 queries are allowed.

The DS-MITM attack on 8-round AES-256 cannot apply to Saturnin, as it relies heavily on the difference between key and state size. Reaching more than 6 super-rounds is yet an open problem, similar to attacking more than 6 AES-128 rounds in the quantum setting.

## 10.6 Discussion

Improving the classical and quantum key-recovery attacks is an interesting open question, with ties to the security analysis of AES. Saturnin with 8 super-rounds seems an interesting target for impossible differential attacks, while 6.5-super-round Saturnin may be the next target of quantum key-recoveries.

**Open Problem 10.1.** *Design a classical key-recovery on more than 7.5 super-rounds, or a quantum key-recovery on more than 6 rounds.*

Before the start of the NIST LWC process, we had tried to include in Saturnin an AEAD mode of rate one, that is, with on average a single block cipher call per message block. Indeed, in Saturnin-CTR-Cascade, the CTR mode requires an encryption per message block, and the Cascade another encryption for each ciphertext block. More modern modes based on block ciphers, such as OCB3 [KR11], have rate one. OCB3 is proven secure in the classical world if it uses a strong PRP, but it admits a Q2 key-recovery attack based on Simon's algorithm. The suitable security definition to use is not even completely clear. Indeed, IND-qCPA and BU are respectively defined with encryption or MACs in mind, not the combination of both.

At the time of writing[8], our work on this topic has significantly moved on. The submitted paper [Bha+20] defines QCB, a rate-one authenticated encryption with associated data. It follows the TAE [LRW02; LRW11] and ΘCB [KR11; Rog04] paradigms. It is based on a *tweakable block cipher* (TBC), a construction that defines a family of independent block ciphers indexed by a *tweak*. In order to use QCB with Saturnin, we define a TBC from $\text{Saturnin}_{16}$ by XORing the tweak to the key. This is the only known construction giving us, with a rate one, the quantum security expected in QCB. Consequently, the security of Saturnin-QCB relies on the related-key security of $\text{Saturnin}_{16}$.

In order to encourage third-party analysis of $\text{Saturnin}_{16}$ in the related-key setting, and to gain a better understanding of its security margin, a challenge[9] has been launched in December 2020.

[8] October 2020

[9] It can be found on this webpage.

# Conclusion and Perspectives

Since the dawn of Q2 quantum attacks, the field of symmetric quantum cryptanalysis has seen many surprises, and several conjectural thoughts or beliefs about quantum attacks[10] have been undermined by further results. At the same time, the quantum Eve is not an all-powerful attacker, and our understanding of her limits is constantly improving. I[11] will conclude this document by recalling some of the results obtained and pointing out some open problems and promising research directions. At the time of writing, I am involved in several projects that investigate some of these questions.

**New "offline" attacks.** In Chapter 7, we saw that a quantum attacker in the Q1 model (classical queries) could exploit the algebraic structure of some symmetric primitives, in order to reduce an exponential memory requirement to polynomial. While the classical attack consists in solving a more generic problem (a multi-target preimage search), we introduced the dedicated *offline Simon's algorithm* that replaces the classical memory with a *compressed* quantum state.

The idea that underlies the offline Simon's algorithm could be applied in more settings. There are two interesting directions to follow: the first one would be to study quantum algorithms with a rather small query complexity, so that the few superposition queries that they make can be emulated by classical queries only. Another direction would be to look at classical algorithms using *structured memories*, or memories in which a hidden structure is embedded. It may then be possible to compress these memories in such a way that a quantum algorithm may detect this structure, while using only a few qubits of memory.

We have briefly mentioned in Section 3.4.2 that there exists breaks of some public-key schemes in the superposition query model (for example LWE [GKZ19]). Although these breaks are not considered meaningful, they may reveal a structure that could be exploited, as we did with Simon's algorithm, in order to devise an improved quantum time-memory trade-off.

We have mentioned in Chapter 7 that Kuperberg's algorithm could be performed in a reversible way and combined with Grover's exhaustive search. However, no more precise analysis has been done yet. We are currently working on the *offline Kuperberg's algorithm* and its applications.

[10] Including many conjectures made by the author of these lines.

[11] The use of a personal pronoun emphasizes that, while all of these works are collaborative, the opinions drawn here are personal.

**Merging and memory.** In Chapter 5 and Chapter 6, we showed that *quantum merging algorithms*, apart from reaching generic speedups and the best quantum time complexities known for generalized birthday algorithms, benefited from some degrees of freedom that have no consequence on the classical time complexity.

In some generalized birthday algorithms, we have seen how to switch between quantum memory models, replacing more powerful quantum memories by less powerful ones. This replacement leaves us on the same time-memory trade-off curve, but it is only applicable for some portion of the curve, degrading the time complexity when memory is unbounded.

There is currently no generic principle or impossibility result dictating the replacement, say, of QRAQM by QRACM, or of QRAQM by plain quantum circuits. Quantum cryptanalysis would certainly benefit of further investigation of these trade-offs, since they may bring us towards more realizable quantum algorithms.

**Improving our quantum toolbox.** During this thesis, we have used and combined several quantum tools: quantum search, Abelian and Boolean hidden shift algorithms, quantum walks based on the MNRS framework. Adapting quantum search to our use cases (Chapter 2) has been easy. Simon's algorithm is well understood, and the family of Abelian hidden shift algorithms that originated with Kuperberg's work has benefited from a non-asymptotic study in several recent works (including [BS20]). However, some shadowy areas remain. In particular, some cryptanalytic applications of the MNRS quantum walk framework have been accompanied by an *ad hoc* conjecture which was first discussed in [HM18], in the context of subset-sum algorithms.

In our work on the subset-sum problem [Bon+20], we tried to overcome this conjecture using modified quantum data structures, but this led to a split between a conjectured time complexity and a non-conjectured one. I believe that a more general study is required to remove this conjecture, which would make quantum walks more flexible and easier to use in cryptanalytic algorithms, and we are working on the case of subset-sums.

**Q2 attacks.** In the Q2 setting, a quantum adversary is allowed to query keyed primitives in superposition over their inputs, as black boxes in her quantum computations. This is a very powerful model. But so far, Q2 attacks have remained very limited. In our analysis of AES of Chapter 9, all the interesting attacks that we obtained used classical queries only, and using superposition queries could not allow better attacks. As shown in Chapter 4, known Q2 breaks are only based on a direct application of Simon's and Kuperberg's algorithms, sometimes in combination with an exhaustive search as in Leander and May's attack. Some quantum linear and differential attacks of [Kap+16b] are of type Q2, but they do not lead to more than quadratic speedups. Other quantum attacks include the distinguishers based on Deutsch-Jozsa's algorithm (the distinguisher on the one-time pad of Section 4.1.7 and its application to modes of operation), but this is a weaker attack model, as it does not recover a key or allow a classical forgery.

There are two possible ways to interpret this situation: either Q2 attacks are actually rare, and rapidly vanish if we use ciphers without strong structures; or we lack dedicated

Q2 cryptanalysis. I believe this question to be one of the most interesting, and perhaps the most challenging, in quantum cryptanalysis.

**Dedicated cryptanalysis.** In Chapter 9, we performed one of the first studies of dedicated quantum attacks on a block cipher, namely the standard AES. In this setting, the quantum attacks corresponding to classical design patterns seem to perform relatively worse compared to exhaustive search. However, the quantum collision attacks of [HS20] have shown that the converse can occur. They are the first example of a reduced security margin in the Q1 setting, since the less than quadratic speedup for generic quantum collision search favors quantum dedicated attacks based on quantum search components. We have encountered this situation in Chapter 8 in the cryptanalysis of Gimli. Many hash functions will certainly be attacked in the future, and this area of research will certainly help us expand our quantum cryptanalysis toolbox. I will start to work on this topic in a few months.

**Better quantum-safe symmetric cryptography.** In Chapter 10, we defined the new block cipher Saturnin and its associated modes of operation. As symmetric cryptosystems become lighter and lighter, the goal of Saturnin is to show that post-quantum security is not orthogonal to lightness, but complementary, and that tomorrow's authenticated ciphers may have the best of both worlds.

In the design of Saturnin, we wanted to use a block cipher-based AEAD with quantum indistinguishability and unforgeability guarantees. However, the constructions available from the literature allowed us only to define an authenticated cipher of rate two (Saturnin-CTR-Cascade), with two block cipher calls per message block processed. Since then, we have been able to define a rate-one parallelizable authenticated cipher, QCB [Bha+20]. It relies on a *quantum-secure tweakable block cipher* (TBC). In the classical setting, there exists a rate-one construction of a TBC from a block cipher secure as a pseudorandom permutation (PRP). In the quantum setting, we currently need a stronger security assumption on the cipher, namely its related-key security. Whether a quantum-secure rate-one AE can be built from a qPRP only remains an open question.

# Bibliography

[Aar02] Scott Aaronson. “Quantum lower bound for the collision problem”. In: *STOC*. ACM, 2002, pp. 635–642 (cit. on p. 45).

[AES] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard*. FIPS Publication 197. 2001 (cit. on pp. xxi, xxv, 50, 54, 55, 61, 201, 203, 238).

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. “An $\mathcal{O}(n \log n)$ Sorting Network”. In: *STOC*. ACM, 1983, pp. 1–9 (cit. on p. 80).

[Al-92] Ibrahim A. Al-Kadit. “Origins of cryptology: The Arab contributions”. In: *Cryptologia* 16.2 (1992), pp. 97–126 (cit. on p. 49).

[Ala+20a] Gorjan Alagic, Stacey Jeffery, Maris Ozols, and Alexander Poremba. “On Quantum Chosen-Ciphertext Attacks and Learning with Errors”. In: *Cryptography* 4.1 (2020), p. 10 (cit. on p. 59).

[Ala+20b] Gorjan Alagic, Christian Majenz, Alexander Russell, and Fang Song. “Quantum-Access-Secure Message Authentication via Blind-Unforgeability”. In: *EUROCRYPT (3)*. Vol. 12107. Lecture Notes in Computer Science. Springer, 2020, pp. 788–817 (cit. on pp. 259–261).

[Alb+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. “Block Ciphers - Focus on the Linear Layer (feat. PRIDE)”. In: *CRYPTO (1)*. Vol. 8616. LNCS. Springer, 2014, pp. 57–76 (cit. on pp. 68, 153, 266).

[Alm+18] Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. “Quantum reversible circuit of AES-128”. In: *Quantum Inf. Process.* 17.5 (2018), p. 112 (cit. on p. 205).

[Amb05] Andris Ambainis. “Polynomial Degree and Lower Bounds in Quantum Complexity: Collision and Element Distinctness with Small Range”. In: *Theory Comput.* 1.1 (2005), pp. 37–46 (cit. on p. 45).

[Amb07] Andris Ambainis. “Quantum Walk Algorithm for Element Distinctness”. In: *SIAM J. Comput.* 37.1 (2007), pp. 210–239 (cit. on pp. 12, 36, 42, 113).

[Amy+16] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. "Estimating the Cost of Generic Quantum Pre-image Attacks on SHA-2 and SHA-3". In: *SAC*. Vol. 10532. LNCS. Springer, 2016, pp. 317–337 (cit. on p. 11).

[Ana+16] Mayuresh Vivekanand Anand, Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. "Post-Quantum Security of the CBC, CFB, OFB, CTR, and XTS Modes of Operation". In: *PQCrypto*. Vol. 9606. LNCS. Springer, 2016, pp. 44–63 (cit. on pp. 58, 259).

[And+08] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. "Second Preimage Attacks on Dithered Hash Functions". In: *EUROCRYPT*. Vol. 4965. LNCS. Springer, 2008, pp. 270–288 (cit. on p. 43).

[And+19] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. *ForkAE v.1*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. "Quantum Lattice Enumeration and Tweaking Discrete Pruning". In: *ASIACRYPT (1)*. Vol. 11272. LNCS. Springer, 2018, pp. 405–434 (cit. on p. 52).

[AR17] Gorjan Alagic and Alexander Russell. "Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts". In: *EUROCRYPT (3)*. Vol. 10212. LNCS. 2017, pp. 65–93 (cit. on p. 69).

[Aru+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. "On the Robustness of Bucket Brigade Quantum RAM". In: *TQC*. Vol. 44. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 226–244 (cit. on p. 19).

[Aru+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510 (cit. on pp. xx, 8).

[Aru+20] Srinivasan Arunachalam, Aleksandrs Belovs, Andrew M. Childs, Robin Kothari, Ansis Rosmanis, and Ronald de Wolf. "Quantum Coupon Collector". In: *TQC*. Vol. 158. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 10:1–10:17 (cit. on p. 35).

[AS04] Scott Aaronson and Yaoyun Shi. "Quantum lower bounds for the collision and the element distinctness problems". In: *J. ACM* 51.4 (2004), pp. 595–605 (cit. on p. 45).

[Aug+] Daniel Augot, Matthieu Finiasz, Philippe Gaborit, Stéphane Manuel, and Nicolas Sendrier. *SHA-3 proposal: FSB*. https://www.rocq.inria.fr/secret/CBCrypto/fsbdoc.pdf (cit. on p. 105).

[Bai+19] Shi Bai, Steven D. Galbraith, Liangze Li, and Daniel Sheffield. “Improved Combinatorial Algorithms for the Inhomogeneous Short Integer Solution Problem”. In: *J. Cryptology* 32.1 (2019), pp. 35–83 (cit. on pp. 96, 105, 112).

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. “Midori: A Block Cipher for Low Energy”. In: *ASIACRYPT (2)*. Vol. 9453. LNCS. Springer, 2015, pp. 411–436 (cit. on p. 161).

[Ban+19a] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, Elmar Tischhauser, and Yosuke Todo. *SUNDAE-GIFT v1.0*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[Ban+19b] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. *GIFT-COFB v1.0*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[Bar+18] Achiya Bar-On, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. “Improved Key Recovery Attacks on Reduced-Round AES with Practical Data and Memory Complexities”. In: *CRYPTO (2)*. Vol. 10992. LNCS. Springer, 2018, pp. 185–212 (cit. on p. 269).

[BB17] Gustavo Banegas and Daniel J. Bernstein. “Low-Communication Parallel Quantum Multi-Target Preimage Search”. In: *SAC*. Vol. 10719. LNCS. Springer, 2017, pp. 325–335 (cit. on pp. 43, 45).

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. “Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials”. In: *EUROCRYPT*. Vol. 1592. LNCS. Springer, 1999, pp. 12–23 (cit. on p. 269).

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. “Improved Generic Algorithms for Hard Knapsacks”. In: *EUROCRYPT*. Vol. 6632. LNCS. Springer, 2011, pp. 364–385 (cit. on pp. xxvii, 141).

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. “Keying Hash Functions for Message Authentication”. In: *CRYPTO*. Vol. 1109. LNCS. Springer, 1996, pp. 1–15 (cit. on pp. 238, 248).

[BDF18] Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. “Revisiting and Improving Algorithms for the 3XOR Problem”. In: *IACR Trans. Symmetric Cryptol.* 2018.1 (2018), pp. 254–276 (cit. on p. 89).

[Bea+13] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. “Efficient distributed quantum computing”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469.2153 (2013) (cit. on p. 80).

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *CRYPTO (2)*. Vol. 9815. LNCS. Springer, 2016, pp. 123–153 (cit. on p. 161).

[Bei+20] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "SKINNY-AEAD and SKINNY-Hash". In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 88–131 (cit. on p. 237).

[Bel+19] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, and Friedrich Wiemer. *Spook: Sponge-Based Leakage-Resilient Authenticated Encryption with a Masked Tweakable Block Cipher*. Submission to NIST-LWC (2nd Round). 2019 (cit. on pp. 57, 157–159, 176, 237).

[Bel+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaétan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. "Spook: Sponge-Based Leakage-Resilient Authenticated Encryption with a Masked Tweakable Block Cipher". In: *IACR Trans. Symmetric Cryptol.* (2020) (cit. on pp. xxv, 53, 158, 180).

[Bel+97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *FOCS*. IEEE Computer Society, 1997, pp. 394–403 (cit. on p. 262).

[Ben+97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani. "Strengths and Weaknesses of Quantum Computing". In: *SIAM J. Comput.* 26.5 (1997), pp. 1510–1523 (cit. on pp. 3, 31, 36).

[Ben89] Charles H. Bennett. "Time/Space Trade-Offs for Reversible Computation". In: *SIAM J. Comput.* 18.4 (1989), pp. 766–776 (cit. on p. 10).

[Ber+07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Sponge functions*. Ecrypt Hash Workshop. 2007 (cit. on p. 56).

[Ber+09] Daniel J. Bernstein, Tanja Lange, Ruben Niederhagen, Christiane Peters, and Peter Schwabe. "FSBday". In: *INDOCRYPT*. Vol. 5922. LNCS. Springer, 2009, pp. 18–38 (cit. on pp. 87, 96, 105).

[Ber+11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *Selected Areas in Cryptography*. Vol. 7118. LNCS. Springer, 2011, pp. 320–337 (cit. on p. 57).

[Ber+11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Cryptographic sponge functions*. Available at https://keccak.team/files/CSF-0.1.pdf. 2011 (cit. on pp. 56, 61).

[Ber+11c] Guido Bertoni, Joan Daemen, Mickaël Peeters, and Gilles Van Assche. *The Keccak reference*. Submission to the SHA-3 competition (round 3). https://keccak.team/files/Keccak-reference-3.0.pdf. 2011 (cit. on p. 240).

[Ber+13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. "Quantum Algorithms for the Subset-Sum Problem". In: *PQCrypto*. Vol. 7932. Lecture Notes in Computer Science. Springer, 2013, pp. 16–33 (cit. on pp. 113, 114, 119, 140, 141, 144).

[Ber+17a] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. "Gimli : A Cross-Platform Permutation". In: *CHES*. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 299–320 (cit. on pp. xxv, 181).

[Ber+17b] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "On Leakage-Resilient Authenticated Encryption with Decryption Leakages". In: *IACR Trans. Symmetric Cryptol.* 2017.3 (2017), pp. 271–293 (cit. on p. 176).

[Ber+17c] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. "Farfalle: parallel permutation-based cryptography". In: *IACR Trans. Symmetric Cryptol.* 2017.4 (2017), pp. 1–38 (cit. on p. 153).

[Ber+19] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. *Gimli*. Submission to NIST-LWC (2nd Round). 2019 (cit. on pp. 53, 57, 157, 183, 199).

[Ber+20] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. *Gimli: NIST LWC Second-round Candidate Status Update*. NIST-LWC 2nd Round update. 2020 (cit. on p. 199).

[Ber09] Daniel J. Bernstein. "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?" In: *SHARCS* 9 (2009), p. 105 (cit. on p. 45).

[Ber10] Daniel J. Bernstein. "Grover vs. McEliece". In: *PQCrypto*. Vol. 6061. LNCS. Springer, 2010, pp. 73–80 (cit. on pp. 37, 52).

[BH97] Gilles Brassard and Peter Høyer. "An Exact Quantum Polynomial-Time Algorithm for Simon's Problem". In: *ISTCS*. IEEE Computer Society, 1997, pp. 12–23 (cit. on p. 20).

[Bha+20] Ritam Bhaumik, Xavier Bonnetain, André Chailloux, Gaëtan Leurent, María Naya-Plasencia, André Schrottenloher, and Yannick Seurin. "QCB: Efficient Quantum-secure Authenticated Encryption". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1304 (cit. on pp. 271, 275).

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum Cryptanalysis of Hash and Claw-Free Functions". In: *LATIN*. Vol. 1380. LNCS. Springer, 1998, pp. 163–169 (cit. on pp. 43, 44).

[Bia+19] Jean-François Biasse, Xavier Bonnetain, Benjamin Pring, André Schrottenloher, and William Youmans. "A Trade-off Between Classical and Quantum Circuit Size for an Attack Against CSIDH". In: *Journal of Mathematical Cryptology* (2019), pp. 1–16 (cit. on pp. xxvii, 11).

[BK03] Mihir Bellare and Tadayoshi Kohno. "A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications". In: *EUROCRYPT*. Vol. 2656. LNCS. Springer, 2003, pp. 491–506 (cit. on p. 253).

[BK09] Alex Biryukov and Dmitry Khovratovich. "Related-Key Cryptanalysis of the Full AES-192 and AES-256". In: *ASIACRYPT*. Vol. 5912. LNCS. Springer, 2009, pp. 1–18 (cit. on p. 238).

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. "Distinguisher and Related-Key Attack on the Full AES-256". In: *CRYPTO*. Vol. 5677. LNCS. Springer, 2009, pp. 231–249 (cit. on p. 238).

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. "Biclique Cryptanalysis of the Full AES". In: *ASIACRYPT*. Vol. 7073. LNCS. Springer, 2011, pp. 344–371 (cit. on pp. 207, 209, 269).

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *J. ACM* 50.4 (2003), pp. 506–519 (cit. on p. 142).

[Bla06] John Black. "The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function". In: *FSE*. Vol. 4047. Lecture Notes in Computer Science. Springer, 2006, pp. 328–340 (cit. on p. 54).

[Ble10] Miles Blencowe. "Quantum RAM". In: *Nature* 468.7320 (2010), pp. 44–45 (cit. on p. 19).

[BM17] Leif Both and Alexander May. "The Approximate k-List Problem". In: *IACR Trans. Symmetric Cryptol.* 2017.1 (2017), pp. 380–397 (cit. on pp. 106, 107).

[BM97] Mihir Bellare and Daniele Micciancio. "A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost". In: *EUROCRYPT*. Vol. 1233. LNCS. Springer, 1997, pp. 163–192 (cit. on p. 105).

[BN08] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm". In: *J. Cryptology* 21.4 (2008), pp. 469–491 (cit. on pp. 238, 248, 257, 258).

[BN18] Xavier Bonnetain and María Naya-Plasencia. "Hidden Shift Quantum Cryptanalysis and Implications". In: *ASIACRYPT (1)*. Vol. 11272. LNCS. Springer, 2018, pp. 560–592 (cit. on pp. 65, 69).

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. "Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon". In: *ASIACRYPT (1)*. Vol. 8873. LNCS. Springer, 2014, pp. 179–199 (cit. on p. 209).

[BNS19a] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "On Quantum Slide Attacks". In: *SAC*. Vol. 11959. LNCS. Springer, 2019, pp. 492–519 (cit. on pp. xxvi, 67, 70, 155).

[BNS19b] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Trans. Symmetric Cryptol.* 2019.2 (2019), pp. 55–93 (cit. on pp. xxiii, xxv, 27, 28, 34, 201, 205, 216, 217, 229).

[Bon+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *ASIACRYPT*. Vol. 7073. LNCS. Springer, 2011, pp. 41–69 (cit. on p. 59).

[Bon+19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. "Quantum Attacks Without Superposition Queries: The Offline Simon's Algorithm". In: *ASIACRYPT (1)*. Vol. 11921. LNCS. Springer, 2019, pp. 552–583 (cit. on pp. xxiv, 145).

[Bon+20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. "Improved Classical and Quantum Algorithms for Subset-Sum". In: *ASIACRYPT (2)*. Vol. 12492. LNCS. Springer, 2020, pp. 633–666 (cit. on pp. xxvii, 141, 144, 274).

[Bon17] Xavier Bonnetain. "Quantum Key-Recovery on Full AEZ". In: *SAC*. Vol. 10719. LNCS. Springer, 2017, pp. 394–406 (cit. on p. 65).

[Bon19] Xavier Bonnetain. "Hidden Structures and Quantum Cryptanalysis". PhD thesis. Paris-Sorbonne University, France, 2019 (cit. on p. 22).

[Bon20] Xavier Bonnetain. "Tight Bounds for Simon's Algorithm". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 919 (cit. on p. 23).

[Bor+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In: *ASIACRYPT*. Vol. 7658. LNCS. Springer, 2012, pp. 208–225 (cit. on pp. 68, 153).

[Bou+18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. "Making the Impossible Possible". In: *J. Cryptology* 31.1 (2018), pp. 101–133 (cit. on pp. 209, 269).

[Boy+98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. "Tight bounds on quantum searching". In: *Fortschritte der Physik: Progress of Physics* 46.4-5 (1998), pp. 493–505 (cit. on p. 31).

[BP10] Joan Boyar and René Peralta. "A New Combinational Logic Minimization Technique with Applications to Cryptology". In: *SEA*. Vol. 6049. LNCS. Springer, 2010, pp. 178–189 (cit. on p. 206).

[BP12] Joan Boyar and René Peralta. "A Small Depth-16 Circuit for the AES S-Box". In: *SEC*. Vol. 376. IFIP Advances in Information and Communication Technology. Springer, 2012, pp. 287–298 (cit. on p. 206).

[BP17] Alex Biryukov and Léo Perrin. "State of the Art in Lightweight Symmetric Cryptography". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 511 (cit. on pp. xx, 53).

[BR00a] Mihir Bellare and Phillip Rogaway. "Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography". In: *ASIACRYPT*. Vol. 1976. LNCS. Springer, 2000, pp. 317–330 (cit. on p. 263).

[BR00b] John Black and Phillip Rogaway. "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions". In: *CRYPTO*. Vol. 1880. LNCS. Springer, 2000, pp. 197–215 (cit. on pp. 65, 66).

[BR02] John Black and Phillip Rogaway. "A Block-Cipher Mode of Operation for Parallelizable Message Authentication". In: *EUROCRYPT*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, 2002, pp. 384–397 (cit. on p. 65).

[Bra+02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. "Quantum amplitude amplification and estimation". In: *Contemporary Mathematics* 305 (2002), pp. 53–74 (cit. on pp. 27, 32, 33).

[BS20] Xavier Bonnetain and André Schrottenloher. "Quantum Security Analysis of CSIDH". In: *EUROCRYPT (2)*. Vol. 12106. LNCS. Springer, 2020, pp. 493–522 (cit. on pp. xxvi, xxvii, 11, 69, 274).

[BS91] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *J. Cryptology* 4.1 (1991), pp. 3–72 (cit. on p. 60).

[Buh+05] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. "Quantum Algorithms for Element Distinctness". In: *SIAM J. Comput.* 34.6 (2005), pp. 1324–1330 (cit. on p. 113).

[BV93] Ethan Bernstein and Umesh V. Vazirani. "Quantum complexity theory". In: *STOC.* ACM, 1993, pp. 11–20 (cit. on pp. 3, 59).

[BW99] Alex Biryukov and David A. Wagner. "Slide Attacks". In: *FSE.* Vol. 1636. LNCS. Springer, 1999, pp. 245–259 (cit. on p. 66).

[BZ13a] Dan Boneh and Mark Zhandry. "Quantum-Secure Message Authentication Codes". In: *EUROCRYPT.* Vol. 7881. LNCS. Springer, 2013, pp. 592–608 (cit. on pp. 52, 259, 260).

[BZ13b] Dan Boneh and Mark Zhandry. "Secure Signatures and Chosen Ciphertext Security in a Quantum Computing World". In: *CRYPTO (2).* Vol. 8043. LNCS. Springer, 2013, pp. 361–379 (cit. on pp. 52, 59, 258, 260).

[Cae69] Julius Caesar. *Caesar's Gallic War.* Vol. 5. Translator. W. A. McDevitte. Translator. W. S. Bohn. New York: Harper & Brothers, 1st Edition, 1869 (cit. on p. 48).

[CAESAR] *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness.* http://competitions.cr.yp.to/caesar.html (cit. on p. 105).

[Cai+19] Jiahao Cai, Zihao Wei, Yingjie Zhang, Siwei Sun, and Lei Hu. "Zero-sum Distinguishers for Round-reduced GIMLI Permutation". In: *ICISSP.* SciTePress, 2019, pp. 38–43 (cit. on pp. 184, 185).

[Can+19] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. *Saturnin.* Submission to NIST-LWC (2nd Round). 2019 (cit. on pp. xxvi, 55, 268, 269).

[Can+20] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. "Saturnin: a Suite of Lightweight Symmetric Algorithms for Post-quantum Security". In: *IACR Trans. Symmetric Cryptol.* (2020) (cit. on pp. xxvi, 54, 235, 238).

[Cas+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *ASIACRYPT (3).* Vol. 11274. LNCS. Springer, 2018, pp. 395–427 (cit. on p. xxvii).

[CEV20] Céline Chevalier, Ehsan Ebrahimi, and Quoc Huy Vu. "On the Security Notions for Encryption in a Quantum World". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 237 (cit. on p. 260).

[CG18] Yu-Ao Chen and Xiao-Shan Gao. "Quantum Algorithms for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 8 (cit. on p. 73).

[Cha+18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. "Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.2 (2018), pp. 218–241 (cit. on p. 154).

[Cha+19a] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancilias Lopez, Mridul Nandi, and Yu Sasaki. *LOTUS-AEAD and LOCUS-AEAD*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[Cha+19b] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, and Mridul Nandi. *HyENA*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[Cha+20] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. "ESTATE: A Lightweight and Low Energy Authenticated Encryption Mode". In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 350–389 (cit. on p. 237).

[CHS19] Jan Czajkowski, Andreas Hülsing, and Christian Schaffner. "Quantum Indistinguishability of Random Sponges". In: *CRYPTO (2)*. Vol. 11693. LNCS. Springer, 2019, pp. 296–325 (cit. on p. 56).

[CJ04] Jean-Sébastien Coron and Antoine Joux. "Cryptanalysis of a Provably Secure Cryptographic Hash Function". In: *IACR Cryptol. ePrint Arch.* 2004 (2004), p. 13 (cit. on p. 105).

[CJS14] Andrew M. Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *J. Mathematical Cryptology* 8.1 (2014), pp. 1–29 (cit. on p. 69).

[CN19] Bishwajit Chakraborty and Mridul Nandi. *MixFeed*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[CNS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. "An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography". In: *ASIACRYPT (2)*. Vol. 10625. LNCS. Springer, 2017, pp. 211–240 (cit. on pp. xxiii, 75, 77, 79, 80).

[CP91] Paul Camion and Jacques Patarin. "The Knapsack Hash Function proposed at Crypto'89 can be broken". In: *EUROCRYPT*. Vol. 547. LNCS. Springer, 1991, pp. 39–53 (cit. on p. 81).

[Cza+18] Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, Christian Schaffner, and Dominique Unruh. "Post-quantum Security of the Sponge Construction". In: *PQCrypto*. Vol. 10786. LNCS. Springer, 2018, pp. 185–204 (cit. on p. 56).

[Dae+18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. "The design of Xoodoo and Xoofff". In: *IACR Trans. Symmetric Cryptol.* 2018.4 (2018), pp. 1–38 (cit. on p. 153).

[Dae91] Joan Daemen. “Limitations of the Even-Mansour Construction”. In: *ASIACRYPT*. Vol. 739. LNCS. Springer, 1991, pp. 495–498 (cit. on pp. 65, 72).

[Dae95] Joan Daemen. “Cipher and hash function design, strategies based on linear and differential cryptanalysis”. PhD thesis. K.U. Leuven, 1995 (cit. on p. 238).

[Dam+13] Ivan Damgård, Jakob Funder, Jesper Buus Nielsen, and Louis Salvail. “Superposition Attacks on Cryptographic Protocols”. In: *ICITS*. Vol. 8317. LNCS. Springer, 2013, pp. 142–161 (cit. on p. 59).

[Dam89] Ivan Damgård. “A Design Principle for Hash Functions”. In: *CRYPTO*. Vol. 435. LNCS. Springer, 1989, pp. 416–427 (cit. on pp. 56, 251).

[Dav19] Nicolas David. “Quantum impossible differential attack. Applications to CLEFIA, AES and SKINNY”. MA thesis. MPRI, Sept. 2019 (cit. on p. 209).

[Der+20] Patrick Derbez, Paul Huynh, Virginie Lallemand, María Naya-Plasencia, Léo Perrin, and André Schrottenloher. “Cryptanalysis Results on Spook - Bringing Full-Round Shadow-512 to the Light”. In: *CRYPTO (3)*. Vol. 12172. LNCS. Springer, 2020, pp. 359–388 (cit. on pp. xxiv, 157, 173).

[DES] National Institute of Standards and Technology (NIST). *Data Encryption Standard*. FIPS Publication 46-3 (archived). 1999 (cit. on pp. 49, 55, 60).

[Deu85] David Deutsch. “Quantum theory, the Church–Turing principle and the universal quantum computer”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117 (cit. on pp. xx, 2).

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. “Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting”. In: *EUROCRYPT*. Vol. 7881. LNCS. Springer, 2013, pp. 371–387 (cit. on pp. 209, 216, 217, 219, 221, 232).

[DH76] Whitfield Diffie and Martin E. Hellman. “New directions in cryptography”. In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654 (cit. on pp. 48, 50).

[Din+12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. “Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems”. In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 719–740 (cit. on pp. 96, 112, 115, 121, 126, 137, 140, 141).

[Din+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. “Design Strategies for ARX with Provable Bounds: Sparx and LAX”. In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 484–513 (cit. on p. 199).

[Din15] Itai Dinur. “Cryptanalytic Time-Memory-Data Tradeoffs for FX Constructions with Applications to PRINCE and PRIDE”. In: *EUROCRYPT (1)*. Vol. 9056. LNCS. Springer, 2015, pp. 231–253 (cit. on p. 67).

[Din19] Itai Dinur. “An algorithmic framework for the generalized birthday problem”. In: *Des. Codes Cryptogr.* 87.8 (2019), pp. 1897–1926 (cit. on pp. 87, 96, 112).

[DJ92] David Deutsch and Richard Jozsa. “Rapid solution of problems by quantum computation”. In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558 (cit. on pp. 3, 70).

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. “The Block Cipher Square”. In: *FSE*. Vol. 1267. LNCS. Springer, 1997, pp. 149–165 (cit. on pp. 210, 211, 215, 267).

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. “Minimalism in Cryptography: The Even-Mansour Scheme Revisited”. In: *EUROCRYPT*. Vol. 7237. LNCS. Springer, 2012, pp. 336–354 (cit. on p. 65).

[DKS15] Orr Dunkelman, Nathan Keller, and Adi Shamir. “Improved Single-Key Attacks on 8-Round AES-192 and AES-256”. In: *J. Cryptology* 28.3 (2015), pp. 397–422 (cit. on p. 204).

[DL18] Sébastien Duval and Gaëtan Leurent. “MDS Matrices with Lightweight Circuits”. In: *IACR Trans. Symmetric Cryptol.* 2018.2 (2018), pp. 48–78 (cit. on pp. 180, 242).

[DN06] Christopher M. Dawson and Michael A. Nielsen. “The Solovay-Kitaev algorithm”. In: *Quantum Inf. Comput.* 6.1 (2006), pp. 81–95 (cit. on p. 7).

[DR02a] Joan Daemen and Vincent Rijmen. “AES and the Wide Trail Design Strategy”. In: *EUROCRYPT*. Vol. 2332. LNCS. Springer, 2002, pp. 108–109 (cit. on p. 238).

[DR02b] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002 (cit. on p. 238).

[DR07] Joan Daemen and Vincent Rijmen. “Probability distributions of correlation and differentials in block ciphers”. In: *J. Mathematical Cryptology* 1.3 (2007), pp. 221–242 (cit. on p. 23).

[DR12] Joan Daemen and Vincent Rijmen. “On the related-key attacks against AES”. In: *Proceedings of the Romanian Academy, Series A* 13.4 (2012), pp. 395–400 (cit. on p. 202).

[DR99] Joan Daemen and Vincent Rijmen. *AES proposal: Rijndael*. Submission to NIST AES competition. 1999 (cit. on pp. 210, 238).

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. “A Meet-in-the-Middle Attack on 8-Round AES”. In: *FSE*. Vol. 5086. LNCS. Springer, 2008, pp. 116–126 (cit. on pp. 216, 269).

[dW19] Ronald de Wolf. *Quantum Computing: Lecture Notes*. 2019. arXiv: 1907.09415 [quant-ph] (cit. on p. 1).

[Dwo05] Morris Dworkin. *Recommendation for block cipher modes of operation: The CMAC mode for authentication (NIST Special Publication 800-38B)*. Tech. rep. National Institute of Standards and Technology, 2005 (cit. on p. 65).

[EM97] Shimon Even and Yishay Mansour. “A Construction of a Cipher from a Single Pseudorandom Permutation”. In: *J. Cryptology* 10.3 (1997), pp. 151–162 (cit. on pp. 65, 149).

[Ess+18] Andre Esser, Felix Heuer, Robert Kübler, Alexander May, and Christian Sohler. “Dissection-BKW”. In: *CRYPTO (2)*. Vol. 10992. LNCS. Springer, 2018, pp. 638–666 (cit. on pp. 142, 143).

[Fer+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. “Improved Cryptanalysis of Rijndael”. In: *FSE*. Vol. 1978. LNCS. Springer, 2000, pp. 213–230 (cit. on pp. 209–211, 213, 270).

[Fey82] Richard P. Feynman. “Simulating physics with computers”. In: *International Journal of Theoretical Physics* 21.6/7 (1982) (cit. on pp. 2, 4).

[Fey85] Richard P. Feynman. “Quantum mechanical computers.” In: *Foundations of Physics* 16.6 (1985), pp. 507–532 (cit. on p. 4).

[Fló+20] Antonio Flórez-Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sibleyras. “New Results on Gimli: Full-Permutation Distinguishers and Improved Collisions”. In: *ASIACRYPT (1)*. Vol. 12491. LNCS. Springer, 2020, pp. 33–63 (cit. on pp. xxiv, 157, 184, 185, 199).

[FO89] Philippe Flajolet and Andrew M. Odlyzko. “Random Mapping Statistics”. In: *EUROCRYPT*. Vol. 434. LNCS. Springer, 1989, pp. 329–354 (cit. on pp. 44, 46, 83).

[Gag17] Tommaso Gagliardoni. “Quantum Security of Cryptographic Primitives”. PhD thesis. Darmstadt University of Technology, Germany, 2017 (cit. on p. 58).

[Gau+08] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. *Grøstl - a SHA-3 candidate*. Submission to the NIST SHA-3 competition. http://www.groestl.info/Groestl.pdf. 2008 (cit. on p. 238).

[GHS16] Tommaso Gagliardoni, Andreas Hülsing, and Christian Schaffner. "Semantic Security and Indistinguishability in the Quantum World". In: *CRYPTO (3)*. Vol. 9816. LNCS. Springer, 2016, pp. 60–89 (cit. on pp. 52, 59).

[Gil+08] Henri Gilbert, Ryad Benadjila, Olivier Billet, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. *SHA-3 proposal: ECHO*. Submission to the NIST SHA-3 competition. https://ehash.iaik.tugraz.at/uploads/9/91/Echo.pdf. 2008 (cit. on p. 238).

[GJN19] Shay Gueron, Ashwin Jha, and Mridul Nandi. *COMET: COunter Mode Encryption with authentication Tag*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[GKZ19] Alex B. Grilo, Iordanis Kerenidis, and Timo Zijlstra. "Learning-with-errors problem is easy with quantum samples". In: *Physical Review A* 99.3 (2019), p. 032314 (cit. on pp. 59, 273).

[Gle+18a] Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. *The SCIP Optimization Suite 6.0*. Technical Report. Optimization Online, 2018 (cit. on p. 97).

[Gle+18b] Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. *The SCIP Optimization Suite 6.0*. ZIB-Report 18-26. Zuse Institute Berlin, 2018 (cit. on p. 97).

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Architectures for a quantum random access memory". In: *Physical Review A* 78.5 (2008), p. 052310 (cit. on p. 18).

[GM08] Samuel Galice and Marine Minier. "Improving Integral Attacks Against Rijndael-256 Up to 9 Rounds". In: *AFRICACRYPT*. Vol. 5023. LNCS. Springer, 2008, pp. 1–15 (cit. on p. 238).

[GNS18] Lorenzo Grassi, María Naya-Plasencia, and André Schrottenloher. "Quantum Algorithms for the k-xor Problem". In: *ASIACRYPT 2018*. Vol. 11272. LNCS. Springer, 2018, pp. 527–559 (cit. on pp. xxiii, 75, 89, 97, 99).

[Got10] Daniel Gottesman. “An introduction to quantum error correction and fault-tolerant quantum computation”. In: *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*. Vol. 68. 2010, pp. 13–58 (cit. on p. 8).

[Got98a] Daniel Gottesman. *The Heisenberg Representation of Quantum Computers*. 1998. arXiv: quant-ph/9807006 [quant-ph] (cit. on p. 7).

[Got98b] Daniel Gottesman. “Theory of fault-tolerant quantum computation”. In: *Physical Review A* 57.1 (1998), p. 127 (cit. on p. 7).

[Gou+19] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. *Romulus v1.2*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[Gou+20] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. “Pyjamask: Block Cipher and Authenticated Encryption with Highly Efficient Masked Implementation”. In: *IACR Trans. Symmetric Cryptol.* 2020.S1 (2020), pp. 31–59 (cit. on p. 237).

[GP10] Henri Gilbert and Thomas Peyrin. “Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations”. In: *FSE*. Vol. 6147. LNCS. Springer, 2010, pp. 365–383 (cit. on pp. 61, 62).

[GR04] Lov K. Grover and Terry Rudolph. “How significant are the known collision and element distinctness quantum algorithms?” In: *Quantum Inf. Comput.* 4.3 (2004), pp. 201–206 (cit. on pp. 18, 44, 125).

[Gra+16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. “Applying Grover’s Algorithm to AES: Quantum Resource Estimates”. In: *PQCrypto*. Vol. 9606. LNCS. Springer, 2016, pp. 29–43 (cit. on pp. 11, 205, 206, 216).

[Gra18] Lorenzo Grassi. “Mixture Differential Cryptanalysis: a New Approach to Distinguishers and Attacks on round-reduced AES”. In: *IACR Trans. Symmetric Cryptol.* 2018.2 (2018), pp. 133–160 (cit. on p. 269).

[Gro+14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. “LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations”. In: *FSE*. Vol. 8540. LNCS. Springer, 2014, pp. 18–37 (cit. on p. 158).

[Gro96] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *STOC*. ACM, 1996, pp. 212–219 (cit. on pp. xxi, 27, 29).

[GRR17] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. “A New Structural-Differential Property of 5-Round AES”. In: *EUROCRYPT (2)*. Vol. 10211. Lecture Notes in Computer Science. 2017, pp. 289–317 (cit. on p. 269).

[Guo+20] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. "Towards Low-Energy Leakage-Resistant Authenticated Encryption from the Duplex Sponge Construction". In: *IACR Trans. Symmetric Cryptol.* 2020.1 (2020), pp. 6–42 (cit. on p. 158).

[Ham17] Mike Hamburg. "Cryptanalysis of 22 1/2 rounds of Gimli". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 743 (cit. on pp. 184, 185).

[Hän+20] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. "Improved Quantum Circuits for Elliptic Curve Discrete Logarithms". In: *PQCrypto*. Vol. 12100. LNCS. Springer, 2020, pp. 425–444 (cit. on p. 11).

[Hel80] Martin E. Hellman. "A cryptanalytic time-memory trade-off". In: *IEEE Trans. Inf. Theory* 26.4 (1980), pp. 401–406 (cit. on p. 43).

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum algorithm for linear systems of equations". In: *Physical review letters* 103.15 (2009), p. 150502 (cit. on p. 73).

[HI19a] Akinori Hosoyamada and Tetsu Iwata. "4-Round Luby-Rackoff Construction is a qPRP". In: *ASIACRYPT (1)*. Vol. 11921. LNCS. Springer, 2019, pp. 145–174 (cit. on p. 65).

[HI19b] Akinori Hosoyamada and Tetsu Iwata. "Tight Quantum Security Bound of the 4-Round Luby-Rackoff Construction". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 243 (cit. on p. 65).

[HJ10] Nick Howgrave-Graham and Antoine Joux. "New Generic Algorithms for Hard Knapsacks". In: *EUROCRYPT*. Vol. 6110. LNCS. Springer, 2010, pp. 235–256 (cit. on pp. xxvii, 141).

[HJM] Martin Hell, Thomas Johansson, and Willi Meier. *Grain - A Stream Cipher for Constrained Environments*. http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf (cit. on pp. 55, 57).

[HM18] Alexander Helm and Alexander May. "Subset Sum Quantumly in $1.17^n$". In: *TQC*. Vol. 111. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 5:1–5:15 (cit. on pp. 141, 274).

[HM20] Alexander Helm and Alexander May. "The Power of Few Qubits and Collisions - Subset Sum Below Grover's Bound". In: *PQCrypto*. Vol. 12100. LNCS. Springer, 2020, pp. 445–460 (cit. on p. 80).

[HNS20] Akinori Hosoyamada, María Naya-Plasencia, and Yu Sasaki. "Improved Attacks on sLiSCP Permutation and Tight Bound of Limited Birthday Distinguishers". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1089 (cit. on p. 62).

[Hoe94] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426 (cit. on p. 76).

[Hon+12] Fang-Yu Hong, Yang Xiang, Zhi-Yan Zhu, Li-zhen Jiang, and Liang-neng Wu. “Robust quantum random access memory”. In: *Physical Review A* 86.1 (2012), p. 010306 (cit. on p. 19).

[Hos+19] Akinori Hosoyamada, Yu Sasaki, Seiichiro Tani, and Keita Xagawa. “Improved Quantum Multicollision-Finding Algorithm”. In: *PQCrypto*. LNCS Vol. 11505. Springer, 2019, pp. 350–367 (cit. on pp. 45, 46).

[HS18a] Akinori Hosoyamada and Yu Sasaki. “Cryptanalysis Against Symmetric-Key Schemes with Online Classical Queries and Offline Quantum Computations”. In: *CT-RSA*. Vol. 10808. LNCS. Springer, 2018, pp. 198–218 (cit. on pp. 58, 72, 80, 149, 151, 152).

[HS18b] Akinori Hosoyamada and Yu Sasaki. “Quantum Demiric-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions”. In: *SCN*. Vol. 11035. LNCS. Springer, 2018, pp. 386–403 (cit. on p. 58).

[HS20] Akinori Hosoyamada and Yu Sasaki. “Finding Hash Collisions with Quantum Computers by Using Differential Trails with Smaller Probability than Birthday Bound”. In: *EUROCRYPT (2)*. Vol. 12106. LNCS. Springer, 2020, pp. 249–279 (cit. on pp. 58, 72, 73, 181, 197, 233, 275).

[HSX17] Akinori Hosoyamada, Yu Sasaki, and Keita Xagawa. “Quantum Multicollision Finding Algorithm”. In: *ASIACRYPT (2)*. Vol. 10625. LNCS. Springer, 2017, pp. 179–210 (cit. on p. 45).

[HY18] Akinori Hosoyamada and Kan Yasuda. “Building Quantum-One-Way Functions from Block Ciphers: Davies-Meyer and Merkle-Damgård Constructions”. In: *ASIACRYPT (1)*. Vol. 11272. LNCS. Springer, 2018, pp. 275–304 (cit. on pp. 261, 264).

[IK03] Tetsu Iwata and Kaoru Kurosawa. “OMAC: One-Key CBC MAC”. In: *FSE*. Vol. 2887. LNCS. Springer, 2003, pp. 129–153 (cit. on p. 65).

[Ind+08] Sebastiaan Indesteege, Elena Andreeva, Christophe De Cannière, Orr Dunkelman, Emilia Käsper, Svetla Nikova, Bart Preneel, and Elmar Tischhauser. *The LANE hash function*. Submission to the NIST SHA-3 competition. https://www.esat.kuleuven.be/cosic/publications/article-1181.pdf. 2008 (cit. on p. 238).

[IPS13] Mitsugu Iwamoto, Thomas Peyrin, and Yu Sasaki. “Limited-Birthday Distinguishers for Hash Functions - Collisions beyond the Birthday Bound Can Be Meaningful”. In: *ASIACRYPT (2)*. Vol. 8270. LNCS. Springer, 2013, pp. 504–523 (cit. on pp. 62, 166).

[Ito+19] Gembu Ito, Akinori Hosoyamada, Ryutaroh Matsumoto, Yu Sasaki, and Tetsu Iwata. “Quantum Chosen-Ciphertext Attacks Against Feistel Ciphers”. In: *CT-RSA*. Vol. 11405. LNCS. Springer, 2019, pp. 391–411 (cit. on p. 65).

[Jao+17] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. *Supersingular Isogeny Key Encapsulation.* Submission to NIST post-quantum project (April 16, 2020 version). 2017 (cit. on pp. xxvi, 111).

[Jaq+20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. "Implementing Grover Oracles for Quantum Key Search on AES and LowMC". In: *EUROCRYPT (2).* Vol. 12106. LNCS. Springer, 2020, pp. 280–310 (cit. on pp. 11, 205–207, 216, 217).

[Jea16] Jérémy Jean. *TikZ for Cryptographers.* `http://www.iacr.org/authors/tikz/`. 2016 (cit. on p. 204).

[Jef14] Stacey Jeffery. "Frameworks for Quantum Algorithms". PhD thesis. University of Waterloo, Ontario, Canada, 2014 (cit. on pp. 12, 13).

[JS19] Samuel Jaques and John M. Schanck. "Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE". In: *CRYPTO (1).* Vol. 11692. LNCS. Springer, 2019, pp. 32–61 (cit. on p. 125).

[JS20] Samuel Jaques and André Schrottenloher. "Low-gate Quantum Golden Collision Finding". In: *SAC.* LNCS. Springer, 2020 (cit. on pp. xxvi, 42, 43, 80, 113).

[Kap+16a] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. "Breaking Symmetric Cryptosystems Using Quantum Period Finding". In: *CRYPTO (2).* Vol. 9815. Lecture Notes in Computer Science. Springer, 2016, pp. 207–237 (cit. on pp. xxvi, 22, 52, 58, 65–67).

[Kap+16b] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. "Quantum Differential and Linear Cryptanalysis". In: *IACR Trans. Symmetric Cryptol.* 2016.1 (2016), pp. 71–94 (cit. on pp. 58, 61, 208, 274).

[Kap14] Marc Kaplan. "Quantum attacks against iterated block ciphers". In: *CoRR* abs/1410.1434 (2014) (cit. on p. 140).

[Ker83a] Auguste Kerckhoffs. "La cryptographie militaire [Military cryptography, part 1]". In: *Journal des sciences militaires [Military Science Journal]* (January 1883) (cit. on p. 48).

[Ker83b] Auguste Kerckhoffs. "La cryptographie militaire [Military cryptography, part 2]". In: *Journal des sciences militaires [Military Science Journal]* (February 1883) (cit. on p. 48).

[Kir11] Paul Kirchner. "Improved Generalized Birthday Attack". In: *IACR Cryptol. ePrint Arch.* 2011 (2011), p. 377 (cit. on p. 105).

[KM10] Hidenori Kuwakado and Masakatu Morii. “Quantum distinguisher between the 3-round Feistel cipher and the random permutation”. In: *ISIT*. IEEE, 2010, pp. 2682–2685 (cit. on pp. xxii, 52, 64).

[KM12] Hidenori Kuwakado and Masakatu Morii. “Security on the quantum-type Even-Mansour cipher”. In: *ISITA*. IEEE, 2012, pp. 312–316 (cit. on pp. xxii, 52, 65, 72, 150, 151).

[Kni95] Emanuel Knill. “An analysis of Bennett’s pebble game”. In: *CoRR* abs/math/9508218 (1995) (cit. on p. 10).

[Knu14] Donald E. Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014 (cit. on p. 42).

[Knu94] Lars R. Knudsen. “Truncated and Higher Order Differentials”. In: *FSE*. Vol. 1008. LNCS. Springer, 1994, pp. 196–211 (cit. on pp. 61, 269).

[KR11] Ted Krovetz and Phillip Rogaway. “The Software Performance of Authenticated Encryption Modes”. In: *FSE*. Vol. 6733. LNCS. Springer, 2011, pp. 306–327 (cit. on pp. 65, 271).

[KR96] Joe Kilian and Phillip Rogaway. “How to Protect DES Against Exhaustive Key Search”. In: *CRYPTO*. Vol. 1109. LNCS. Springer, 1996, pp. 252–267 (cit. on pp. 68, 153).

[KSW04] Hartmut Klauck, Robert Spalek, and Ronald de Wolf. *Quantum and Classical Strong Direct Product Theorems and Optimal Time-Space Tradeoffs*. 2004. arXiv: quant-ph/0402123 [quant-ph] (cit. on p. 35).

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. “Quantum Information Set Decoding Algorithms”. In: *PQCrypto*. Vol. 10346. LNCS. Springer, 2017, pp. 69–89 (cit. on p. 52).

[Kup05] Greg Kuperberg. “A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem”. In: *SIAM J. Comput.* 35.1 (2005), pp. 170–188 (cit. on pp. xxvii, 69).

[Kup13] Greg Kuperberg. “Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem”. In: *TQC*. Vol. 22. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 20–34 (cit. on pp. 12, 18, 69).

[Kut05] Samuel Kutin. “Quantum Lower Bound for the Collision Problem with Small Range”. In: *Theory Comput.* 1.1 (2005), pp. 29–36 (cit. on p. 45).

[LIM19] Fukang Liu, Takanori Isobe, and Willi Meier. “Preimages and Collisions for Up to 5-Round Gimli-Hash Using Divide-and-Conquer Methods”. In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1080 (cit. on pp. 185, 186).

[LIM20a] Fukang Liu, Takanori Isobe, and Willi Meier. “Automatic Verification of Differential Characteristics: Application to Reduced Gimli”. In: *CRYPTO (3)*. Vol. 12172. LNCS. Springer, 2020, pp. 219–248 (cit. on pp. 185, 186).

[LIM20b] Fukang Liu, Takanori Isobe, and Willi Meier. “Exploiting Weak Diffusion of Gimli: A Full-Round Distinguisher and Reduced-Round Preimage Attacks”. In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 561 (cit. on pp. 184–186).

[LL+93] Arjen K. Lenstra, Hendrik W. Jr. Lenstra, et al. *The development of the number field sieve.* Vol. 1554. Springer Science & Business Media, 1993 (cit. on p. xix).

[LM17] Gregor Leander and Alexander May. “Grover Meets Simon - Quantumly Attacking the FX-construction”. In: *ASIACRYPT (2).* Vol. 10625. LNCS. Springer, 2017, pp. 161–178 (cit. on pp. xxiii, 67, 68).

[LMP13] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. “Solving the Shortest Vector Problem in Lattices Faster Using Quantum Search”. In: *PQCrypto.* Vol. 7932. LNCS. Springer, 2013, pp. 83–101 (cit. on p. 52).

[LNS18] Gaëtan Leurent, Mridul Nandi, and Ferdinand Sibleyras. “Generic Attacks Against Beyond-Birthday-Bound MACs”. In: *CRYPTO (1).* Vol. 10991. LNCS. Springer, 2018, pp. 306–336 (cit. on p. 57).

[LP07] Gregor Leander and Axel Poschmann. “On the Classification of 4 Bit S-Boxes”. In: *WAIFI.* Vol. 4547. LNCS. Springer, 2007, pp. 159–176 (cit. on p. 267).

[LPS19] Brandon Langenberg, Hai Pham, and Rainer Steinwandt. “Reducing the Cost of Implementing AES as a Quantum Circuit”. In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 854 (cit. on pp. 205, 206).

[LR88] Michael Luby and Charles Rackoff. “How to Construct Pseudorandom Permutations from Pseudorandom Functions”. In: *SIAM J. Comput.* 17.2 (1988), pp. 373–386 (cit. on p. 63).

[LRW02] Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. “Tweakable Block Ciphers”. In: *CRYPTO.* Vol. 2442. LNCS. Springer, 2002, pp. 31–46 (cit. on p. 271).

[LRW11] Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. “Tweakable Block Ciphers”. In: *J. Cryptol.* 24.3 (2011), pp. 588–613 (cit. on p. 271).

[LS90] Robert Y. Levin and Alan T. Sherman. “A Note on Bennett’s Time-Space Tradeoff for Reversible Computation”. In: *SIAM J. Comput.* 19.4 (1990), pp. 673–677 (cit. on p. 10).

[LTW18] Gregor Leander, Cihangir Tezcan, and Friedrich Wiemer. “Searching for Subspace Trails and Truncated Differentials”. In: *IACR Trans. Symmetric Cryptol.* 2018.1 (2018), pp. 74–100 (cit. on p. 270).

[Lyu05] Vadim Lyubashevsky. “The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem”. In: *APPROX-RANDOM.* Vol. 3624. LNCS. Springer, 2005, pp. 378–389 (cit. on p. 83).

[LZ19] Qipeng Liu and Mark Zhandry. "On Finding Quantum Multi-collisions". In: *EUROCRYPT (3)*. Vol. 11478. LNCS. Springer, 2019, pp. 189–218 (cit. on p. 45).

[Mar+17] Daniel P. Martin, Ashley Montanaro, Elisabeth Oswald, and Daniel James Shepherd. "Quantum Key Search with Side Channel Advice". In: *SAC*. Vol. 10719. LNCS. Springer, 2017, pp. 407–422 (cit. on p. 203).

[ME98] Michele Mosca and Artur Ekert. "The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer". In: *QCQC*. Vol. 1509. LNCS. Springer, 1998, pp. 174–188 (cit. on p. 11).

[Mer89] Ralph C. Merkle. "A Certified Digital Signature". In: *CRYPTO*. Vol. 435. LNCS. Springer, 1989, pp. 218–238 (cit. on pp. 56, 251).

[Mic+] Daniele Micciancio, Yuriy Arbitman, Gil Dogon, Vadim Lyubashevsky, Chris Peikert, and Alon Rosen. *SWIFFT*. `https://www.eecs.harvard.edu/~alon/PAPERS/lattices/swifftx.pdf` (cit. on p. 105).

[MMO85] Stephen M. Matyas, Carl H. Meyer, and Jeffrey M. Oseas. *Generating strong one-way functions with cryptographic algorithm*. IBM Technical Disclosure Bulletin 27. 1985 (cit. on pp. 248, 251).

[Mos15] Michele Mosca. "Cybersecurity in a quantum world: will we be ready?" In: *Workshop on Cybersecurity in a Post-Quantum World, Invited Presentation*. 2015 (cit. on p. xx).

[Mou+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. "Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers". In: *Selected Areas in Cryptography*. Vol. 8781. LNCS. Springer, 2014, pp. 306–323 (cit. on pp. 154, 237).

[MPP09] Marine Minier, Raphael C.-W. Phan, and Benjamin Pousse. "Distinguishers for Ciphers and Known Key Attack against Rijndael with Large Blocks". In: *AFRICACRYPT*. Vol. 5580. LNCS. Springer, 2009, pp. 60–76 (cit. on p. 238).

[MR10] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010 (cit. on p. 35).

[MS12] Lorenz Minder and Alistair Sinclair. "The Extended k-tree Algorithm". In: *J. Cryptology* 25.2 (2012), pp. 349–382 (cit. on pp. 82, 85, 96, 112).

[MV04] David A. McGrew and John Viega. "The Security and Performance of the Galois/Counter Mode (GCM) of Operation". In: *INDOCRYPT*. Vol. 3348. LNCS. Springer, 2004, pp. 343–355 (cit. on p. 65).

[Nai+19] Yusuke Naito, Mitsuru Matsui, Yasuyuki Sakai, Daisuke Suzuki, Kazuo Sakiyama, and Takeshi Sugawara. *SAEAES*. Submission to NIST-LWC (2nd Round). 2019 (cit. on p. 237).

[Nak08] Jorge Nakahara Jr. "3D: A Three-Dimensional Block Cipher". In: *CANS*. Vol. 5339. LNCS. Springer, 2008, pp. 252–267 (cit. on p. 238).

[Nan14] Mridul Nandi. “XLS is Not a Strong Pseudorandom Permutation”. In: *ASIACRYPT (1)*. Vol. 8873. LNCS. Springer, 2014, pp. 478–490 (cit. on p. 106).

[Nan15] Mridul Nandi. “Revisiting Security Claims of XLS and COPA”. In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 444 (cit. on p. 106).

[Nat01] National Institute of Standards and Technology. *SP 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. Dec. 2001 (cit. on pp. 55, 262).

[Nat18] National Academies of Sciences, Engineering, and Medicine. *Quantum Computing: Progress and Prospects*. Washington, DC: The National Academies Press, 2018. ISBN: 978-0-309-47969-1 (cit. on pp. xxi, 201, 202).

[Nay11] María Naya-Plasencia. “How to Improve Rebound Attacks”. In: *CRYPTO*. Vol. 6841. LNCS. Springer, 2011, pp. 188–205 (cit. on p. 89).

[NC00] Michael A. Nielsen and Isaac L. Chuang. “Quantum information and quantum computation”. In: *Cambridge: Cambridge University Press* 2.8 (2000), p. 23 (cit. on pp. xxiii, 1, 7, 8).

[NCB11] Robert Niebuhr, Pierre-Louis Cayrel, and Johannes Buchmann. “Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems”. In: *WCC 2011 - Workshop on coding and cryptography*. 2011, pp. 163–172 (cit. on p. 105).

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. 2016 (cit. on pp. xxi, 52, 59, 201).

[NIS17] NIST. *Report on Lightweight Cryptography (NISTIR 8114)*. 2017 (cit. on pp. 53, 236).

[NIS18] NIST. *Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process*. Published: August 27, 2018 (cit. on pp. xx, 53, 157).

[NIS19] NIST. *Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process (NISTIR 8268 )*. Published: October 2019 (cit. on pp. xx, xxiv, 53).

[NP07] Jorge Nakahara Jr. and Ivan Carlos Pavão. “Impossible-Differential Attacks on Large-Block Rijndael”. In: *ISC*. Vol. 4779. LNCS. Springer, 2007, pp. 104–117 (cit. on p. 238).

[NS15] Ivica Nikolic and Yu Sasaki. “Refinements of the k-tree Algorithm for the Generalized Birthday Problem”. In: *ASIACRYPT (2)*. Vol. 9453. LNCS. Springer, 2015, pp. 683–703 (cit. on pp. 42, 87, 96, 106, 112).

[NS20] María Naya-Plasencia and André Schrottenloher. “Optimal Merging in Quantum k-XOR and k-SUM Algorithms”. In: *EUROCRYPT (2)*. LNCS Vol. 12106. Springer, 2020, pp. 311–340 (cit. on pp. xxiii, xxiv, 75, 89, 99, 107, 109, 118, 121, 126, 144).

[Oec03] Philippe Oechslin. “Making a Faster Cryptanalytic Time-Memory Trade-Off”. In: *CRYPTO*. Vol. 2729. LNCS. Springer, 2003, pp. 617–630 (cit. on p. 43).

[Pat91] Jacques Patarin. “New Results on Pseudorandom Permutation Generators Based on the DES Scheme”. In: *CRYPTO*. Vol. 576. LNCS. Springer, 1991, pp. 301–312 (cit. on p. 64).

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. “Hash Functions Based on Block Ciphers: A Synthetic Approach”. In: *CRYPTO*. Vol. 773. LNCS. Springer, 1993, pp. 368–378 (cit. on p. 263).

[PO95] Bart Preneel and Paul C. van Oorschot. “MDx-MAC and Building Fast MACs from Hash Functions”. In: *CRYPTO*. Vol. 963. LNCS. Springer, 1995, pp. 1–14 (cit. on p. 57).

[Pol] John M. Pollard. “A Monte Carlo method for factorization, BIT 15 (1975), 331–334”. In: *MR* 52 (), p. 13611 (cit. on p. 42).

[Pre18] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (2018), p. 79 (cit. on p. 8).

[PS97] Alessandro Panconesi and Aravind Srinivasan. “Randomized Distributed Edge Coloring via an Extension of the Chernoff-Hoeffding Bounds”. In: *SIAM J. Comput.* 26.2 (1997), pp. 350–368 (cit. on p. 83).

[Rao+17] Sandeep Rao, Dindayal Mahto, Dilip K. Yadav, and Danish Khan. “The AES-256 Cryptosystem Resists Quantum Attacks”. In: 8 (Apr. 2017), pp. 404–408 (cit. on p. 203).

[RBH17] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. “Yoyo Tricks with AES”. In: *ASIACRYPT (1)*. Vol. 10624. LNCS. Springer, 2017, pp. 217–243 (cit. on p. 269).

[Reg04] Oded Regev. *A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space*. 2004. arXiv: `quant-ph/0406151 [quant-ph]` (cit. on p. 69).

[Rid18] Philippe Ridet. “Saturnin, Rintintin, Flipper... le douloureux destin des animaux stars”. In: *M, le magazine du Monde* (July 2018), pp. 25–30 (cit. on p. 235).

[Ril89] Rainer M. Rilke. *Lettres à un jeune poète*. Trans. by Claude Mouchard et Hans Hartje. Le Livre de Poche, 1989 (cit. on p. iii).

[Rog04] Phillip Rogaway. “Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC”. In: *ASIACRYPT*. Vol. 3329. LNCS. Springer, 2004, pp. 16–31 (cit. on p. 271).

[RR07] Thomas Ristenpart and Phillip Rogaway. “How to Enrich the Message Space of a Cipher”. In: *FSE*. Vol. 4593. LNCS. Springer, 2007, pp. 101–118 (cit. on p. 105).

[RS15] Martin Rötteler and Rainer Steinwandt. “A note on quantum related-key attacks”. In: *Inf. Process. Lett.* 115.1 (2015), pp. 40–44 (cit. on pp. 59, 152).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (1978), pp. 120–126 (cit. on pp. xix, 50).

[Sas+15] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. *Minalpher v1.1*. Submission to the CAESAR competition. 2015 (cit. on p. 153).

[Sas10] Yu Sasaki. “Known-Key Attacks on Rijndael with Large Blocks and Strengthening *ShiftRow* Parameter”. In: *IWSEC*. Vol. 6434. Lecture Notes in Computer Science. Springer, 2010, pp. 301–315 (cit. on p. 238).

[SHA3] National Institute of Standards and Technology (NIST). *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS Publication 202. 2015 (cit. on p. 56).

[Sha49] Claude E. Shannon. “Communication theory of secrecy systems”. In: *Bell Syst. Tech. J.* 28.4 (1949), pp. 656–715 (cit. on pp. 48, 54, 70).

[Sho94] Peter W. Shor. “Algorithms for Quantum Computation: Discrete Logarithms and Factoring”. In: *FOCS*. IEEE Computer Society, 1994, pp. 124–134 (cit. on pp. xx, 3, 19, 51).

[Sim94] Daniel R. Simon. “On the Power of Quantum Computation”. In: *FOCS*. IEEE Computer Society, 1994, pp. 116–123 (cit. on pp. xx, 1, 19, 20).

[Sin99] Simon Singh. *The code book: the secret history of codes and codebreaking*. Fourth Estate London, 1999 (cit. on p. 49).

[SJS16] Vladimir Soukharev, David Jao, and Srinath Seshadri. “Post-Quantum Security Models for Authenticated Encryption”. In: *PQCrypto*. Vol. 9606. LNCS. Springer, 2016, pp. 64–78 (cit. on p. 260).

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. “Extending SAT Solvers to Cryptographic Problems”. In: *SAT*. Vol. 5584. LNCS. Springer, 2009, pp. 244–257 (cit. on p. 191).

[SS17] Thomas Santoli and Christian Schaffner. “Using Simon’s algorithm to attack symmetric-key cryptographic primitives”. In: *Quantum Inf. Comput.* 17.1&2 (2017), pp. 65–78 (cit. on pp. 20, 65, 66).

[SS81] Richard Schroeppel and Adi Shamir. “A $T = \mathcal{O}(2^{n/2}), S = \mathcal{O}(2^{n/4})$ Algorithm for Certain NP-Complete Problems”. In: *SIAM J. Comput.* 10.3 (1981), pp. 456–464 (cit. on p. 111).

[Sun+16] Bing Sun, Meicheng Liu, Jian Guo, Longjiang Qu, and Vincent Rijmen. "New Insights on AES-Like SPN Ciphers". In: *CRYPTO (1)*. Vol. 9814. LNCS. Springer, 2016, pp. 605–624 (cit. on p. 269).

[SY17] Fang Song and Aaram Yun. "Quantum Security of NMAC and Related Constructions - PRF Domain Extension Against Quantum attacks". In: *CRYPTO (2)*. Vol. 10402. LNCS. Springer, 2017, pp. 283–309 (cit. on pp. 52, 261–263).

[Tof80] Tommaso Toffoli. "Reversible Computing". In: *ICALP*. Vol. 85. LNCS. Springer, 1980, pp. 632–644 (cit. on p. 6).

[Tou65] Jean Tourane. *Les Aventures de Saturnin [Television series], Première chaîne de l'ORTF*. Produced by Maintenon Films. 1965 (cit. on p. 235).

[Tur36] Alan M. Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *J. of Math* 58.345-363 (1936), p. 5 (cit. on p. 2).

[Unr17] Dominique Unruh. "Post-quantum Security of Fiat-Shamir". In: *ASIACRYPT (1)*. Vol. 10624. LNCS. Springer, 2017, pp. 65–95 (cit. on p. 52).

[vOW99] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *J. Cryptology* 12.1 (1999), pp. 1–28 (cit. on pp. 42, 76, 111).

[Wag02] David A. Wagner. "A Generalized Birthday Problem". In: *CRYPTO*. Vol. 2442. LNCS. Springer, 2002, pp. 288–303 (cit. on pp. 80–82, 85, 87, 96, 105).

[Wan+12] Qingju Wang, Dawu Gu, Vincent Rijmen, Ya Liu, Jiazhe Chen, and Andrey Bogdanov. "Improved Impossible Differential Attacks on Large-Block Rijndael". In: *ICISC*. Vol. 7839. LNCS. Springer, 2012, pp. 126–140 (cit. on p. 238).

[WH87] Robert S. Winternitz and Martin E. Hellman. "Chosen-Key Attacks on a Block Cipher". In: *Cryptologia* 11.1 (1987), pp. 16–20 (cit. on p. 152).

[Wie04] Michael J. Wiener. "The Full Cost of Cryptanalytic Attacks". In: *J. Cryptology* 17.2 (2004), pp. 105–124 (cit. on pp. 12, 155).

[Zal99] Christof Zalka. "Grover's quantum searching algorithm is optimal". In: *Physical Review A* 60 (4 1999), pp. 2746–2751 (cit. on pp. 31, 32).

[ZDW19] Rui Zong, Xiaoyang Dong, and Xiaoyun Wang. "Collision Attacks on Round-Reduced Gimli-Hash/Ascon-Xof/Ascon-Hash". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1115 (cit. on p. 186).

[Zha+08] Lei Zhang, Wenling Wu, Je Hong Park, Bonwook Koo, and Yongjin Yeom. "Improved Impossible Differential Attacks on Large-Block Rijndael". In: *ISC*. Vol. 5222. LNCS. Springer, 2008, pp. 298–315 (cit. on p. 238).

[Zha12] Mark Zhandry. "How to Construct Quantum Random Functions". In: *FOCS*. IEEE Computer Society, 2012, pp. 679–687 (cit. on pp. 52, 59).

[Zha15] Mark Zhandry. “A note on the quantum collision and set equality problems”. In: *Quantum Inf. Comput.* 15.7&8 (2015), pp. 557–567 (cit. on pp. 45, 263).

[Zha19] Mark Zhandry. “How to Record Quantum Queries, and Applications to Quantum Indifferentiability”. In: *CRYPTO (2)*. Vol. 11693. LNCS. Springer, 2019, pp. 239–268 (cit. on pp. 52, 81, 82, 264).