Machine-checked verification of digital signature schemes in EasyCrypt

Sara Zain

Department of Computer Science
University of Bristol

A PhD dissertation submitted to the University of Bristol
in accordance with the requirements of the degree of
Doctor in Philosophy in the Faculty of Engineering.

June 2022

Word count: 42404

Abstract

The signature schemes based on the discrete logarithm problem often suffer from their length. Their security reduction usually involves either rewinding [1] (leading to looseness) or complex reasoning (requiring non-standard assumptions) [2, 3, 4, 5]. At EUROCRYPT '03, Goh and Jarecki presented a signature scheme (EDL), a discrete logarithm-based scheme with tight security in the random oracle model. The EDL signature size is lengthier than the traditional ones (more exponentiations). At CRYPTO '05, Chevallier-Mames proposed a new signature scheme (CM) where he shortened the size of the EDL scheme. The scheme benefitted from a tight discrete logarithm-based reduction in the random oracle model. Later in 2007, Goh, Jarecki, Katz and Wang [6] (GJKW) proposed an efficient signature scheme, and the scheme is also a discrete logarithm-based scheme with tight security. Moreover, the last two decades have witnessed that the cryptographic proofs in the provable security are complicated and only focus on algorithmic definitions rather than the implementation, which automatically increases vulnerabilities. The gap between cryptographic engineering and provable security can be addressed with machine-checked frameworks.

This thesis aims to formalise and presents the first machine-checked security proofs of three digital signatures (EDL, CM and GJKW) and shows that the schemes are the first discrete logarithm-based schemes with tight security, produced using EasyCrypt proof assistant. Although the pen and paper proofs of all three schemes have differences, the core principles that make them secure in EasyCrypt are the same as the EasyCrypt formalisation allows one to identify the proof principles. This research shows that proofs extracted from all three signature schemes have a generic proof structure and also have the technique of reducing the local proof effort. One can adopt the technique and the proof pattern to other related existing or new constructions of discrete logarithm-based schemes to obtain proof of tight security.

Key words: Machine-checked proofs, Formal verification, EasyCrypt Proof assistant, Digital signature scheme, CDH assumption, Tight security.

Author's declaration:

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .....Sara..Zain..... DATE: .....30/06/2022.....

Students must print their name on the examination copy and on the final Library copy.

Dedication and Acknowledgements

Firstly, I would like to express my heartfelt gratitude to my supervisor Dr François Dupressoir for his endless support of my PhD research and his patience, motivation and immense knowledge. His guidance helped me throughout my studies, and I could not have asked for a better supervisor for my PhD research.

I would like to thank my former secondary supervisor Prof. Steve Schneider who kept checking on me during the first two years while I was doing my PhD at the University of Surrey. His guidance and encouragement helped me throughout the transferring period from Surrey to Bristol.

Besides my supervisors, I would like to thank Prof. Liqun Chen, who always answered my questions happily, and I am grateful for her insightful comments and encouragement. My sincere appreciation goes to Microsoft Research and ESPRC for funding my PhD; without their funds, it would not be possible to conduct this research.

Lastly, I would like to thank my mother for her unconditional support and blessings; and myself for never quitting and enduring the challenges I faced.

Publication

The work described in this thesis has been presented in the following publication:

F. Dupressoir and S. Zain, "Machine-checking unforgeability proofs for signature schemes with tight reductions to the computational diffie-hellman problem," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE, 2021, pp. 1–15.

Contents

List of Figures

Chapter 1

Introduction

Modern cryptography has developed into a systematic technique where the cryptographic proofs are analysed and developed through schemes. The schemes have rigorous proofs in which the ultimate goal is to prove that the given construction is secure. Over the past several decades, the proofs for cryptographic schemes have evolved into complex proofs that rely on the unproven algorithmic hardness of some mathematical assumptions. So, the increasing concern is not just that the cryptographic proof relies on unproven assumptions but also that the proofs in the schemes are not checked. The reason for not checking the proofs in detail is because the security proofs are lengthy and complicated and might be due to laziness. The proof errors are mostly oversight which automatically increases vulnerabilities. Finding vulnerabilities in cryptographic schemes has become one of the main challenges many cryptographers face. Therefore, in 2005, Halevi [7] advocated designing a machine-checking framework to help cryptographers build and verify the proofs.

Since then, many have designed and contributed to the development of machine-checked tools, which support checking the cryptographic proofs step by step and provide confidence and high assurance in the security proofs. Many tools and proof assistants are designed to deliver computer-supported proofs; examples are [8, 9, 10, 11]. The approach of verifying and building the security proofs whilst utilising state-of-the-art verification tools, SMT solvers, and interactive proof assistants aims to ensure the correctness of the program and its reasoning steps. This thesis gets the aid of a state-of-the-art EasyCrypt proof assistant [12, 13], which is an interactive proof assistant that aims to verify computational security proofs of the cryptographic constructions. The EasyCrypt operates with the Probabilistic Relational Hoare Logic (pRHL), Why3 Software veri-

fication platform, Satisfiability Modulo Theories (SMT) solvers and theorem provers for the verifications, where the semantics of languages are formalised in the Coq proof assistant [14]. The language of EasyCrypt includes the assertions of assignments, random samplings, while loops, conditionals, and calls to the functions. One can say that the expression language of EasyCrypt is based on higher-order strongly typed functional language. The proofs in the EasyCrypt are developed using the game-playing technique.

This thesis uses the methodology of the game-playing technique [15], which gives explicit mathematical statements, referred to as games, and security statements are expressed as a sequence of games. The proofs are developed by defining the game, where a successful attack on the security properties leads to winning the game and later defining the advantage. For instance, letting an imaginary adversary play game in security proof. If one can show that a game can not be won, the underlying scheme is secure, and this can be done by transforming the initial game into a new game ($Game_n$ ... $Game_{n+1}$). If the difference between the initial and subsequent game is small enough, it is sufficient to prove that $Game_{n+1}$ can not be won, and this technique can be applied to a long chain of games. So, if the advantage of a game is small, then this implies that the chance to attack the security property successfully has to be small. In the end, such transitions of games enable one to analyse a game through the properties of another game, and finally, one can prove the security of a scheme.

A central motivation for this research is to explore and learn the skills of the emerging approach of machine-checking security proofs. This approach is challenging and requires one to be consistent and stubborn, but the joy of proving goals is undefinable. Another motivation is to close the gap between the cryptographic proofs and the implementations of the proofs in the proof assistant by extracting the techniques so one can re-use the technique for related schemes.

1.1 Aims and Contributions

This thesis aims to formalise and machine-checked three digital signature schemes Goh and Jarecki (EDL) [16], Chevallier-Mames (CM) [17] and Goh et al. (GJKW) [6], using the EasyCrypt proof assistant. The security of the schemes is tightly related to the Computational Diffie-Hellman problem (CDH) in the random oracle model (ROM).

The research begins with the formal verification of the pen and paper proof of Goh and Jarecki's scheme [16] (EDL). In the random oracle model, they show a signature scheme that is existentially unforgeable under a chosen-message attack (EUF-CMA). The scheme is constructed over a group where the reduction is based on the hardness of the

CDH problem. The EDL scheme was firstly presented by Chaum and Pedersen [18] and Jakobsson and Schnorr [19]; however, their schemes did not include the security analysis. The scheme remained neglected until Goh and Jarecki proposed the security analysis, and that is because it was viewed as a modification of the Schnorr signatures [20] before. To our knowledge, this thesis presents the first machine-checked verification of the EDL signature scheme using EasyCrypt. During the formalisation of the signature scheme, the thesis encountered minor mistakes and found overlooked steps in Goh and Jarecki's pen and paper proof [16], leading the thesis to imprecise security bounds. However, the thesis reports a very close bound instead, which is suggested by Chevallier-Mames for the EDL scheme [17].

After machine-checked formalisation of the first signature scheme (EDL), it was easier to formally verify the second related signature scheme by Chevallier-Mames (CM) [17] in EasyCrypt. He also presented the security reduction of the EDL as tightly related to the CDH problem. He further claims that his signature is efficient, and the results are smaller than the original EDL signature scheme. The difference in the CM scheme is that the message is not included in the random oracle query, which served as the second base for the proof of discrete logarithm equality. Instead, the message is included as input to the challenge-generating random oracle query, similar to the standard Schnorr signatures. Consequently, it supports the security of the scheme but without additional randomness. For the machine-checked proofs, the security reduction of the CM is inlined with the given original pen and paper proof by Chevallier-Mames.

The pen and paper proofs of both EDL and CM schemes have direct reductions; however, machine-checked formal verification of the schemes is based on the code-based game-playing technique advocated by Shoup [15]. During the machine-checked verification, both schemes have been reformulated, and the statements of the security reductions are concrete and constructive. In the formal development of both schemes, the similarities are sought out and then factored in. Both schemes have three formal intermediate games, which bridge the gap between the EUF-CMA game and the CDH game. The *Shim* is extracted after deep insights into those intermediate games and written as an instance of a common game named *Shim*. The formalisation effort shows that any two successive games differ only in the oracles provided to the *Shim*. The property of the *Shim* is to reduce the boilerplate proof amount and focus reasoning on the parts of the proofs that change between the successive games. The insight gained from the formalisation of both schemes directs the thesis to further generalisation, and the research outlines a novel four-step proof structure (refactorisation, embedding, simulation and reduction). In the generic structure, the first step is refactorisation which refactors the EUF-CMA security of EDL as an instance of Shim. The second step is embedding, which embeds the CDH challenge into the responses of the oracle queries. The third step is

a simulation, which relies on the zero-knowledge proof. The fourth step is a reduction, which relies on the soundness of the zero-knowledge proof and underlying CDH problem. These proof-steps aim to introduce a general approach that can be applied to similar signature schemes.

The reason for the third machine-checked formalisation began when the thesis gained the full knowledge of the machine-checked proofs for the EDL and CM schemes. The thesis argues that one can use the technique extracted from the EDL and CM schemes that can be applied to similar signature schemes based on a tight discrete logarithm. So, this thesis shows that the third related signature scheme by Goh, Jarecki, Katz and Wang [6] (GJKW) can be formalised with the above techniques mentioned, using the EasyCrypt. However, there are changes in the overall machine-checked proofs for the GJKW's scheme. This is because the pen and paper proofs of the scheme are slightly different from the EDL and CM schemes, but the formalised proofs of the GJKW delivers the same security, that is, the tight discrete logarithm-based reductions. The main difference in the pen and paper proof of the GJKW's scheme is that they have shortened the size of the scheme to improve the EDL signature scheme and removed the random salt. The machine-checked formal verification of the GJKW's scheme shows that the technique extracted from the EDL and CM scheme has the potential, and one can apply the techniques to similar digital signature schemes.

The machine-checked formalisation of all three signature schemes EDL, CM and GJKW are based on discrete logarithm signature schemes. The proofs present a tight security reduction from the Existential unforgeability under the adoptive Chosen Message Attack (EUF-CMA) to the hardness of the CDH problem in the Random Oracle Model (ROM). The security proofs of the scheme are done through exact security and reasoned that the CDH problem is almost as hard to solve as the scheme to break. The formal verification of all three signature schemes shows that they are related to each other with slight differences in their pen and paper proofs, but the core principles that make them secure are the same in the EasyCrypt.

1.2 Thesis structure

The thesis is composed of six themed chapters. The first chapter of this thesis identifies the knowledge gap in the field of study and states the focus, aims, and contributions to the field. Chapter two overviews the digital signature schemes and introduces the basic knowledge of cryptography and its security model with its mathematical primitives. Chapter three begins with background knowledge of the formal semantics and

its notations. This chapter also includes the background definitions of the code-based game-playing techniques, followed by a game declaration in the EasyCrypt proof assistant. The fourth chapter presents the machine-checked formalisation of the two signature schemes, EDL and CM, presented at the conference. Chapter five formalises and machine-checks the scheme of GJKW and presents the dimension of comparing the security proofs with the EasyCrypt's proofs to help understand the proof structure. This chapter also includes a discussion of related and future work. Finally, the last chapter concludes with a discussion, conclusion and future work of the PhD study.

Chapter 2

Digital Signature Schemes

This chapter begins with an overview of the signature scheme, which is the main focus of this thesis. Then it defines some formal definitions and finishes with a discussion.

2.1 Introduction

According to the *Cambridge Dictionary*, cryptography has two definitions: "*the practice of creating and understanding codes that keep information secret*"; and the second "*the use of special codes to keep information safe in computer networks*". The first definition may refer to *classic cryptography*, which is the era of the Roman Empire. However, the second definition is more relevant and involves the term "computers," which means the twenty-first century's modern era.

Over the years, modern cryptography has emerged as a science where schemes are analysed and developed with rigorous proof to secure cryptographic constructions. The cryptographic schemes are constructed based on the required specifications and properties. The difference between classic and modern cryptography is that modern cryptography focuses on three principles formal definitions, having precise assumptions and the proofs of security. These principles are the key contributions to modern cryptography. To have a secure cryptographic scheme, one must clearly describe the scope of the threats and how it guarantees security. Then the underlying hard assumptions must be explicitly stated. Finally, the cryptographic scheme requires security proof, giving an iron-clad guarantee.

In modern cryptography, digital signature schemes play an essential role and are used widely to ensure digital data's integrity, authenticity, and non-repudiation. In modern cryptography, digital data refers to any form and shape of information represented and processed digitally. It includes electronic information such as messages, files, documents, images and videos and all those data and scope that can be stored and transmitted digitally. So, cryptographic algorithms and protocols are designed and constructed to secure digital data against unauthorised access and forgery. Most of the time, the mechanism of protecting digital data is to employ encryption, hashing and digital signatures.

2.2 Digital Signatures

One key focus of modern cryptography is developing digital signature schemes, where a digital signature is constructed to prove secure, relying on computational assumptions. The idea of the digital signature was first presented by Diffie and Hellman [21]. The idea served as a conceptual contribution toward modern cryptography, as Diffie and Hellman were unsuccessful in constructing a digital scheme; however, the idea was later formalised in the computational assumption by Goldwasser et al. [22].

Digital signatures are cryptographic mechanisms that sign messages and then verify the message signatures to prove the authenticity of digital messages. It delivers *authentication*: a proof that the sender has the private key and signed the message; *integrity*: a proof that the message was not modified after the sign; and *non-repudiation*: once the signature is made, the signer can not deny it.

A digital signature scheme is widely applied in many industries, especially in the financial industry, for money transfers, signing transactions in the public blockchain systems to transfer tokens, coins and assets, and exchanging signed online documents. So it applies to integrity (or authentication) and non-repudiation. As an authentication example, consider Alice, who wants to convince everyone that the message m published is by her. To do this, Alice generates a public key pk along with a secret key sk and publishes the public key in some reliable way to all verifiers while keeping her secret key. Alice generates a digital signature σ_m on m, signs m using her secret key, and sends (σ_m, m) to every receiver. Upon receiving (σ_m, m), each receiver can verify the signature σ_m and confirm the origin of the m with respect to the public key.

The digital signature schemes generally employ a public-key cryptosystem such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) and use the public and private key pairs. The signature scheme has three algorithms: key generation, signing and verification. Generally, the input message is hashed, and the signing

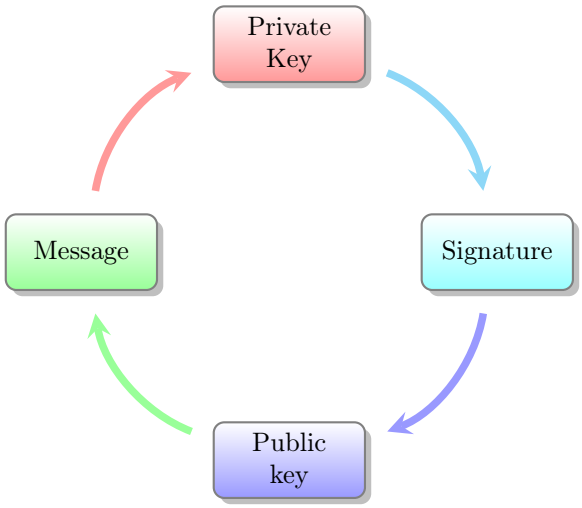

Figure 2.1: The private key signs a message, and the corresponding public key verifies the signature

algorithm calculates the signature. Most of the time, the signing algorithms accomplish some computations with the hashed message and the signing key so that the result can not be computed without the signing key. The following diagram 2.1 shows that a message is signed by the sender using his/her private key.

The computation of the message signature also ensures that the authentic signer signs a particular message by using his/her private key, which then checks the public key, which guarantees that the message is authenticated. In addition, if the public key of the message signer is public, then anyone can verify the signature. After receiving the signature, the signer can not deny the act of signing, which gives non-repudiation property.

For verifying the signature, the message for verification is hashed, and mathematical computations are executed between the digital signature, hashed message and the public key. Finally, it checks if the signature can be accepted or not.

A digital signature differs from Message Authentication Code (MAC) because MACs are generated and verified using the same key (private key) via *symmetric algorithms*. In contrast, digital signatures are generated by a signing key and are verified by a separate verification key via *asymmetric algorithms*. However, both deliver the message with authenticity and integrity.

The following section delves into the fundamental cryptography concepts, beginning with exploring Mathematical Primitives. These mathematical building blocks form the backbone of cryptographic algorithms and protocols.

2.2.1 Mathematical Primitives

In modern Cryptography, the schemes can be written with mathematical primitives for security reduction; then, one can define the hard problem over the same mathematical primitives, which means that one breaks the scheme by solving the mathematically hard problem. For instance, if schemes are constructed in a cyclic group $\mathbb{G}$, the underlying hard problem must be determined over the same group. So, the aim becomes to prevent the attacker's attack, which can either be computational or decisional in the security reduction. However, it is very challenging to lessen the attacks on the scheme to solve the hard problem because the instance of the hard problem and the schemes are produced separately. For instance, a proposed scheme is substituted with another well-prepared scheme in the reduction process, which correlates with a problem instance.

Furthermore, it is essential to be familiar with the following concepts in security reduction. In the following section, specific mathematical terminology extracted from the source [23] will be cited.

Computation over Prime Field

The prime field ($\mathbb{F}_q$) comprises exclusively numerical entities drawn from the set $\mathbb{F}_q = \{0, 1, 2, ..., q-1\}$. The following illustrates the various modular arithmetic operations conducted over the prime field.

Additive Inverse: Given an arbitrary element $y \in \mathbb{Z}_q$, one can proceed to derive

$$-y \; mod \; q.$$

Multiplicative Inverse: Given an arbitrary element $z \in \mathbb{Z}_q*$, one can proceed to derive

$$1/z = z^{-1} \; mod \; q.$$

Addition: Given an arbitrary element $y, z \in \mathbb{Z}_q$, one can proceed to derive

$$y + z \; mod \; q.$$

Multiplication: Given an arbitrary element $y, z \in \mathbb{Z}_q$, one can proceed to derive

$$y \cdot z \; mod \; q.$$

Exponentiation: Given an arbitrary element $y, z \in \mathbb{Z}_q$, one can proceed to derive

$$y^z \ mod \ q.$$

Cyclic Group

The cyclic group (denoted as $\mathbb{G}$) of prime order q uses an integer in $\{1, 2, ..., q-1\}$ in modular multiplicative inverse to construct the schemes. For instance, g^x is the group element of $\mathbb{G}$ and $g^{1/x}$ is a group element, and any group element except $1_{\mathbb{G}}$ in $\mathbb{G}$ is the generator (denoted by g).

Group Exponentiation: Consider a cyclic group denoted as $(\mathbb{G}, g, q)$. To establish the notion of group exponentiation within this framework, one can define the expression g^x as follows:

$$g^x = \underbrace{g \cdot g \cdots g \cdot g}_{x}$$

where x the positive integer.

So, the group exponentiation becomes

$$g^x = g^{x \ mod \ q}$$

where x is picked from the set $\mathbb{Z}_q$.

2.2.2 RSA signatures

RSA is a widely used cryptographic algorithm that can be utilised for digital signature schemes. The security of RSA-based digital signatures relies on the computational infeasibility of certain mathematical operations, such as factoring large composite numbers or computing discrete logarithms. The RSA algorithm's strength in these areas makes it challenging for adversaries to forge a valid signature without access to the private key. Furthermore, the RSA signatures can be deterministic, where the exact message and the private key produce the exact signature and non-deterministic, where they can be created by padding the input message with randomness before the sign. RSA signatures are employed for signing digital certificates to safeguard websites. For instance, Microsoft uses SHA256RSA for its digital certification. Regardless, the recent trend is

Gen : on input 1^n run $\mathsf{Gen}(1^n)$ to obtain (N, e, x). The public key is $\langle N, e\rangle$
and the secret key (sk) is: $\langle N, d\rangle$, where $ed = 1 \; mod \; \Phi(N)$.
Sign : on input a secret key and message $m \in \mathbb{Z}_q^*$, and output is
$\sigma_{\mathsf{sk}}(m) := \left[\mathsf{m}^d \; mod \; N\right]$.
Ver : on input the public key, a message, and a signature, $\mathsf{pk} = \langle N, e\rangle, \mathsf{m} \in \mathbb{Z}_q^*$, and $\sigma \in \mathbb{Z}_q^*$
respectively , output 1 if $\mathsf{m} = \sigma^e \; mod \; N$, output 0 otherwise.

Figure 2.2: Plain RSA signature scheme

to move from RSA to ECC signatures due to the shorter key lengths and signatures while providing high security and performance.

Public-key encryption is one of the essential aspects of public-key cryptography and has many uses, such as key exchange and data confidentiality. However, some might assume that digital signatures are the inverse of public-key encryption, with the interchangeable functions of the sender and receiver. The mistake often arises because the plain Rivest–Shamir–Adleman(RSA) signatures reverse the plain RSA encryption. That means signing a message m by decrypting it, using the private key to get the signature σ and verifying a signature σ by encrypting it using the public key and checking if the result is message m. This type of scheme construction is unfounded and simply inapplicable in cases where it can be used insecurely. So, both plain RSA and plain RSA encryption meet the slimmest notions of security, discussed next.

2.2.3 Plain RSA

Although the RSA-based signature scheme meets the minimal notions of security, it can be a valuable starting point.

The following figure 2.2 defines the construction of the plain RSA signature scheme [24], where it considers Gen be a PPT algorithm that, on input 1^n, outputs a modulus N that is the product of two n-bit primes, with integers e, x satisfying $ex = 1 \bmod \Phi(N)$. Plain RSA applies Gen and outputting $\langle N, e\rangle$ as the public key and $\langle N, x\rangle$ as the private key to generate keys. To sign a message $\mathsf{m} \in \mathbb{Z}_N^*$, the signer computes $\sigma := [\mathsf{m}^x \; mod \; N]$. Verifying a signature σ on a message m for the public key $\langle N, e\rangle$ is brought out by checking whether $\mathsf{m} \stackrel{?}{=} \sigma^e \; mod \; N$.

So the verification of a successfully generated signature is

$$\sigma_{\mathsf{sk}}(m)^e = (\mathsf{m}^d)^e = m^{[ed \bmod \Phi(N)]} = \mathsf{m}^1 = m \bmod N.$$

One can view it as secure, as for any adversary learning only the public key $\langle N, e\rangle$; computing a successful signature on a message m appears to mandate solving the RSA assumption. However, the above reasoning is inaccurate because, for one thing, the RSA problem only insinuates the hardness of computing a signature of a uniform message m. It does not express the hardness of computing a signature on non-uniform m or, for some message m of the adversary's preference. Also, the RSA problem does not state anything regarding the adversary's behaviour once she learns about the signatures on other messages.

The following instance shows that the above comment conducts an attack on the plain RSA signature scheme.

A no-message attack This attack yields a forgery of operating the public key independently and without acquiring any signatures from the valid signer. The attack ensues with a public key $\mathsf{pk} = \langle N, e\rangle$, picks a uniform σ, calculates m, and outputs the forger (m, σ). It is a forgery that σ is a valid signature on message m, as the public key proprietor did not assign the signature.

It is not a real attack as the adversary has no power over the message m. For that, it forges a valid signature. Additionally, the adversary can control message m by picking uniform values of σ, and it could be with a high probability of getting message m in some settings. By choosing σ in some desired way, there is a possibility to exploit the consequent message m for which a forgery is a result.

2.2.4 Signatures from Hash Functions

In contrast to public-key encryption, signature schemes can also be formed based on hash functions. In such settings, the scheme does not rely on random oracles as opposed to the initial construction. Examples of signature schemes based on cryptographic hash functions are Lamport's signature [25], Chain-based signatures [26], and Tree-based signatures [27]. The details of such construction are irrelevant, so the section is short.

2.3 Exploring Cryptographic Concepts

2.3.1 Hash Function

Hash functions are utilised so that one securely embeds strings of any length into group elements and exponents, enhancing the computational efficiency without employing the

large group. The security of group-based cryptography even relies on hash functions, and if the hash functions are violated, then the scheme is no longer secure.

The details about the hash function are explained later, and here it briefly explains the types. When the input is an arbitrary string, hash functions are categorised into the following types.

$\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$. This type embeds the hashing values in group exponents when the input value is not in the space of $\mathbb{Z}_q$. The output space is $\{0,1\}$, where q is the group order.

$\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}$. This type hashes the input string into a group element when the output space is a cyclic group.

$\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n$. This type generates a symmetric key from the key space $\{0,1\}$ for encryption when the output space is the set containing n-bit strings.

2.3.2 Random Oracle model (ROM)

During the security reduction of the proposed scheme, the random oracle (RO) $\mathcal{O}$ is adopted to illustrate an ideal hash function. The ideal hash function is denoted as $\mathcal{H}$, and the output of the distribution result is uniformly and randomly. The proposed scheme uses the $\mathcal{H}$, particularly as the primitives, and $\mathcal{H}$ is treated as a RO. The oracle acts as a *box*. All parties, in addition to the attacker, can interact with the box, takes a binary string x as an input, and returns a binary string y as an output; which refers to that x made a query to the oracle. The querying is kept secret; no party can learn who made the queries if someone queries the oracle. The box also has the property of "*consistency*," which means that if the box returns y for some x, so then it always returns the same result y once provided the same x again.

In security reductions, it is essential to know the difference between random oracles and hash functions.

Hash function	**Random oracle**
Given the arbitrary string x, the attacker has the knowledge of the hash functions $\mathcal{H}$ so that she can compute $\mathcal{H}(x)$.	Given the arbitrary string x, the attacker does not have the knowledge of the $\mathcal{H}(x)$ if hash function was set as a RO unless it was queried x to the RO.
The same input space plus the number of inputs to $\mathcal{H}$ are exponential.	The same input space and the number of inputs to RO are polynomial.
The output space is the same and relies on the description of $\mathcal{H}$. Given the input, the results from $\mathcal{H}$ are defined by the input and the algorithm of $\mathcal{H}$. The output from $\mathcal{H}$ is not required to be uniformly distributed.	The output space is the same, depending on the description of $\mathcal{H}$. Given the input, a simulator can define the output from RO, which can control the RO. The output from RO must be random and uniformly distributed.

In the ROs, the security proofs can become more effortless compared to those ROs which do not. The reason is that ROs are accommodating the simulator to program the reduction, and also, the simulator handles picking some random output which makes the attacker apply attacks.

Computational and Decisional

Earlier, it was mentioned that the attacks could be computational or decisional to solve the hard problem. In a computational attack, the attacker wins the game by forging a valid signature not asked before. Whereas in a decisional attack, the attacker guesses the message and guesses with either 0 or 1.

Loose and Tight Reduction

Tight and loose reductions are used to estimate the security reduction loss. According to Micali and Reyzin, "*A reduction in which the difficulty of forging and the difficulty of solving the underlying hard problem are close is called tight; otherwise, it is called loose*" [28]. They put it *close* if the reduction is less efficient and *loose* if it is significantly less efficient. The tight reduction is when the reduction loss is small; for instance, the number of queries is sub-linear. The simulator makes the signature queries on a message, and the relation between the hard problem and the difficulty of the forged signature can be quantified. So, given the definition of the hardness of the underlying problem, the security of a signature is called tight: if some problem is (t', ϵ', δ')-secure, then the signature scheme is $(t, \epsilon, \delta, q_{hash}, q_{sig})$-secure. If t is smaller than t' and epsilon ϵ, δ is larger than ϵ' and δ' and also independent of the attacker's queries, be it hash (q_{hash}) or signature (q_{sig}) queries.

The loose reduction is when the reduction loss is linear during the queries; for instance, the attacker makes the number of hash and signature queries.

Negligible function

The negligible function represents how large the function can be with its probability. The formal definition can be defined as follows:

Definition 1. A function μ is negligible, if for all constants $c \in N$, $\exists n \in N$ such that for $n > N$, $\mu(n) < \frac{1}{n^c}$.

2.3.3 Distinguishing the Given Scheme

The attacker can distinguish the real scheme from the simulated ones by their random numbers and correctness.

Randomness

In the simulated scheme, the random number and group element should be independent and random. The attacker can effortlessly find if the scheme is simulated; or if the real scheme did not produce a random number or random group element independently and randomly, causing the simulated scheme not to generate random numbers. Random numbers can be used in digital signatures and many other constructions.

The following paragraphs introduce the concept of randomness briefly.

Random numbers Numerous digital signature schemes depend on random numbers that are distinctive and non-predictable. The random numbers are also understood as a digital signature's ephemeral key, session key and nonce. The private signature keys can be compromised if the random number generators fail due to underlying interactive zero-knowledge proof (will be explained later). A notable instance of reusing the random numbers is when Sony Playstation3 reused it, which compromised the signature keys [29]. In the last few decades, the trend has been to use deterministic signature schemes as they are considered more securer to implement.

Pseudorandomness After defining and explaining the securities, it is now easier to introduce pseudorandomness, which grabs the concept that appears entirely random even though it is not. Pseudorandom generators (PRNG) play an essential part in private-key encryption. It refers to an efficient and deterministic algorithm that uses mathematical

formulas from a uniform string called a *seed* to produce a uniform-looking sequence (pseudorandom). The result of a PRNG should be computationally indistinguishable from a uniform string.

Pseudorandom Functions(PRF) The origin of the name, Pseudorandom Functions, can be seen as a long string of random numbers. The string is the function's truth table, where a distinguisher interacts with the functions. The result would be a random function, a function whose truth table is picked uniformly at random, which means that the result of the pseudorandom generator appears to be a uniformly yielded string to the efficient viewer.

Correctness

The attacker can also distinguish the simulated scheme from the real by correctness.

In the simulated scheme, all answers to queries by the attacker should be identical to the real scheme. Consequently, the attacker gets the corresponding answers when she queries the simulated scheme. So, if the answer is not valid, then the attacker can easily guess whether the scheme is simulated, and that is because the real one only correctly answers all the queries. For instance, the attacker guesses the signature scheme to be simulated if the message's signature cannot verify.

Several other security settings of signature schemes are defined by Micali, Rivest and Goldwasser [22, 30]. This motivates an emphasis on discrete-logarithm-based problems and concrete signature schemes based on the RSA problem, and the RSA was explained earlier already. The following section explains the discrete logarithm problem.

2.3.4 Discrete-Logarithm

The discrete logarithm problem (DL) is considered one of the most difficult problems to solve in cryptography and has been used in many security protocols. The famous security protocol which uses the discrete logarithm problem is the Diffie-Hellman key exchange. In this protocol, two parties, Alice and Bob, decide on some primitive root g and prime order p. Alice chooses a secret integer a, and Bob chooses b. When both parties choose their secret integers, Alice sends $g^a \mod p$ to Bob, and similarly, Bob sends $g^b \mod p$ to Alice. In order to have a secure session with each other publicly, both parties use their secret key. For instance, the first party (Alice) takes the second party's (Bob) group element g^b and raises it with its secret exponent a, which gives $(g^b)^a = g^{ab}$. Similarly, the second party (Bob) takes the first party's (Alice) group element g^a and raises it with its secret exponent b and gives $(g^a)^b = g^{ab}$. The computation of both

parties gives the same result, $g^{ab} \mod p$ as $g^{ab} \equiv (g^a)^b \equiv (g^b)^a(\mod p)$. So, in this way, given g^a, g^b, g and p, no adversary can find g^{ab} without knowing either a or b, which asks to solve the discrete logarithm problem.

Cryptography's discrete logarithm problem is one of the most studied number theoretic problems. Now, let us say that G is the cyclic group and g is $g \in G$, and x is the integer, and computation is $y = g^x \mod p$. So, the inverse of this computation is said to be a discrete logarithm problem which is to solve $g^x \equiv y(\mod p)$ for x.

Discrete logarithms are logarithms described for multiplicative cyclic group $\mathbb{G}$. Next, it defines the discrete logarithm problem formally.

Definition 2. Given a group $\mathbb{G}$, a generator g of a group and an element h of $\mathbb{G}$; an integer x satisfies $g^x = h$, then DL to the base g of h in the group $\mathbb{G}$. So, to compute x is known as the DL problem.

The DL problem is one of the fundamental problems in cryptography as there is no polynomial-time algorithm to solve the discrete logarithm problem in a cyclic group.

The discrete logarithm problem based schemes shown to be secure in the Random Oracle (RO) model [31] (ROs will be discussed later) include the Schnorr Signature [20], the EDL scheme [18, 16], an ElGamal scheme [32] variant suggested by Pointcheval and Stern [33], and Girault-Poupard-Stern scheme [34, 35]. The underlying concept is to run the adversary twice with different hash oracles to get two valid forgeries on the same message. The following subsection describes the Schnorr Signature Scheme first, then the Schnorr Identification Scheme, followed by the Fiat-Shamir transformation.

2.3.5 The Schnorr Signature Scheme

The scheme was described by Claus Schnorr [20], derived from the Schnorr's identification protocol [36] using the Fiat–Shamir heuristic [37]. The security stands in the RO model using the forking lemma [1] on the interactivity of discrete logarithm problem. This signature scheme is considered simple, efficient, generate short signatures, and is provably secure in the random oracle model (see Figure 2.3).

The following subsection describes the high-level intuition of the Schnorr identification scheme using the diagram.

Gen : picks a random secret key $x \in \mathbb{Z}_q$, sets $y = g^x$.

Sign : on input secret key x and message, the Signer picks random $k \in \mathbb{Z}_q$and sets $I = g^k$.
It calculates $r = \mathcal{H}(I, m)$($\mathcal{H}$ is function specified as part of key generation),
computes $s = rx + kmodq$.Then sends the signature (r, s).

Ver : on input public key, a message, a signature (r, s), calculates $I = g^s \cdot y^{-r}$.
Finally checks if $r \stackrel{?}{=} \mathcal{H}(I, m)$.

Figure 2.3: Schnorr Signature Scheme

2.3.6 Schnorr Identification Scheme

In this scheme, two parties, a *Prover* and a *Verifier*, interact. The goal of this scheme is that *Prover* wants to convince *Verifier* that he/she knows the secret knowledge; however, *Verifier* should not learn the secret information. To show the security notion of the scheme, the *Prover* chooses a random $\mathsf{x} \in \mathbb{Z}_q$ (secret key) and sets $\mathsf{y} = g^x$. The protocol has three rounds to protect against the eavesdropping attack (see Figure 2.4).

1. The *Prover* chooses a random $\mathsf{k} \in \mathbb{Z}_q$, sets $I = g^k$, and sends I to the *Verifier*.
2. The *Verifier* chooses a random challenge $\mathsf{r} \in \mathbb{Z}_q$ and sends it to the *Prover* as a response.
3. The *Prover* computes $\mathsf{s} = rx + k$ and sends s to the *Verifier*.

The *Verifier* accepts if and only if $I = g^s \cdot y^{-r}$, which intuitively means that if the legitimate *Prover* completed the protocol accurately, then the *Verifier* must always accept.

The above protocol 2.4 can be transformed into non-interactive using the Fiat-Shamir transformation [37], assuming a secure hash function $\mathcal{H}$ exists (see Figure 2.5). The intuition is that both *Prover* and *Verifier* have access to the hash function modeled as a random oracle, and *Prover* acts as a *Signer* and runs the identification protocol by itself. The *Prover* computes I, sets $\mathsf{r} = \mathcal{H}(g, I, y)$, computes s, and sends (r, s) to the *Verifier*. On the other hand, *Verifier* computes $I' = g^s \cdot y^{-r}$ and accepts if and only if $\mathsf{r} \stackrel{?}{=} \mathcal{H}(g, I', y)$.

The Schnorr non-interactive zero-knowledge proof can be obtained from the interactive identification scheme through the Fiat Shamir transformation. So, the following section introduces the high-level description and some formal definitions of the zero-knowledge proofs.

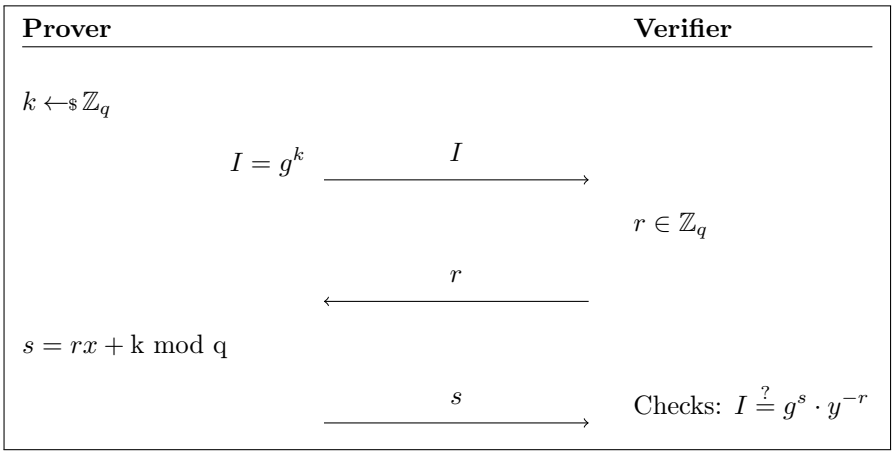

Figure 2.4: Schnorr Identification Scheme

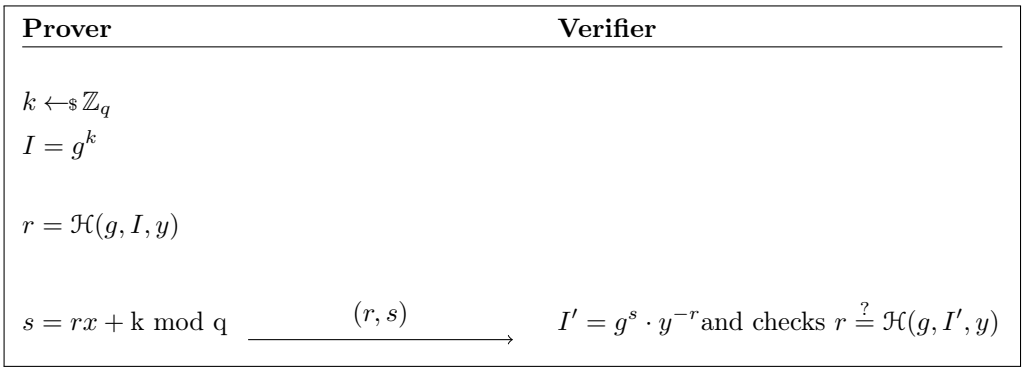

Figure 2.5: Non-interactive Schnorr

2.3.7 Zero-Knowledge Proofs (ZKPs)

The Zero-Knowledge Proofs (ZKPs) were first introduced by Goldwasser, Micali, and Rackoff [38]. The basic idea behind ZKPs is that it is a protocol between *Prover* and *Verifier*. In the protocol, the *Prover* convinces the *Verifier* that it has some secret information, and *Verifier* validates the claim without requiring the *Prover* to reveal any underlying information about its secret information. In this way, the *Verifier* does not learn anything about the information at all. The proposed model in [38] allows the *Prover* to validate the argument with *Verifier* through several rounds of interaction, called *interactive Zero-Knowledge proofs* or just simply *Zero-Knowledge proofs*. A well-known practical application of ZKPs is in the Trusted Platform Module (TPM) [39] and in the blockchain world [40, 41].

One can define an interactive proof system $\langle \mathrm{P}, \mathrm{V} \rangle$ as a Zero-Knowledge if, for any V', a simulator S produces a transcript of the conversation between P and V' on any provided input x. So, from the interaction with P can not lead V' to compute something that was not computed before. Also, Goldwasser, Micali, and Rackoff [38] define the definition

of the interactive proof system $\langle \mathrm{P}, \mathrm{V} \rangle$ for a language L as Zero-Knowledge; either the $\langle \mathrm{P}, \mathrm{V} \rangle$ is a proof system for language L, or it is not. Their formal definition is: *for polynomial time V', the distribution that V' can see on all its tapes while interacting with P on input x in language L, is indistinguishable from a distribution that can be computed from x in polynomial time* [38].

Now that we have the intuition that a proof system is said to possess Zero-Knowledge property if, following the interaction, the *Verifier*'s knowledge remains unchanged because no new information is acquired that surpasses the knowledge that could have been obtained independently. Then the formal definition is:

Definition 3. A proof system $\langle \mathrm{P}, \mathrm{V} \rangle$ for a language L is considered Zero-Knowledge if, for every efficient *Verifier* strategy V', there exists an efficient probabilistic algorithm S' (simulator) such that, for every x belonging to L, the random variables denoting the output of V' after interacting with P on input x and the output of S' on input x are computationally indistinguishable.

In other words, the Zero-Knowledge property asserts that the *Verifier*'s knowledge remains unaffected by the interaction with the *Prover*. Regardless of the algorithm V' employed by the *Verifier*, any information gained from the interaction can be equally obtained by running the stand-alone algorithm S' on the same input. Therefore, the *Verifier*'s knowledge acquisition is independent of the interaction and solely relies on the inherent properties of the input.

Earlier, we saw the Schnorr Identification scheme (see Figure 2.4), which was the recurrence of three round protocol. The *Prover* sent a message I as *commitment*, and the *Verifier* sent a random r-bit string as a *challenge*. The *Prover* sent a *response* as s, and the *Verifier* checks. So, the soundness of such protocol can be proved with ***special-soundness***. It can be defined as there exists an extractor that, on given input x, for any set of n valid transcripts and, in this case, pair of transcripts (I, r, s), (I, r', s') with sharing the same *commitments* but having different *challenges* such as $r \neq r'$. In other words, if a *Prover* can produce two or more valid proofs for the same instance, the special soundness property guarantees that there are different valid witnesses associated with each proof.

A Zero-Knowledge proof for L is a three-round interactive protocol between a *Prover* and a *Verifier*, satisfying the following properties:

1. Completeness
   This property is considered the high probability that if *Prover* tells the truth, then *Verifier* eventually be convinced that the *Prover* is telling the truth. Also, if the input is correct, then ZKP always return true.

   Definition 4. Formally one can define *completeness* as a game between two parties, *Prover* and *Verifier*. Let $\epsilon - completeness$ be a small positive value, and then the game proceeds as follows; the *Prover* receives an input x. On the other hand, *Verifier* sends a challenge c to the *Prover*, whereas the *Prover* now responds with a proof Π based on challenge c. The *Verifier* then checks the validity of proof Π based on the input x and challenge c. Hence, the completeness property is satisfied if, for every valid input x, there exists an honest *Prover* that can convince the *Verifier* in the game with a probability equal to **1**, and *Verifier* accepts the proof Π as valid. .

   So one can say that if x is in language L, then on input x, Prover and Verifier with the given x interact, and once the interaction is completed, the Verifier outputs *Accept* with a probability of at least 1.

2. Soundness
   This property is considered the low probability that *Prover* can only convince the *Verifier* if it tells the truth, and if the input is incorrect, then it is not possible to fake the output to the *Verifier* that it is true, or it is not possible to trick the ZKP to return true.

   Definition 5. Formally one can also define *soundness* as a game between two parties, *Prover* and *Verifier*. Let $\epsilon - soundness$ be a small positive value, and then the game proceeds as follows; the *Prover* receives an input x, which is either true or false. The *Verifier* then selects a random challenge c, and the *Prover* responds with a proof Π based on the challenge. Finally, the Verifier checks proof

Π's validity based on input x and challenge c. Hence, the *soundness* property is satisfied if, for every dishonest *Prover*, let us say *Prover'*, the probability that the *Verifier* accepts an incorrect input $x\prime$ is less than or equal to $\epsilon - soundness$. Ultimately, no cheating *Prover* can convince the *Verifier* of an incorrect input with a probability greater than $\epsilon - soundness$.

So one can say that if x **is not** in language L, then on input x, Prover and Verifier with the given x interact, and once the interaction is completed, the Verifier outputs *Reject* with a probability of at least $\frac{1}{2}$.

3. Zero-knowledge
   In this property, the *Verifier* does not learn anything about the *Prover*'s secret information. This property can be view as if the input is valid; then the *Verifier* does not learn any information except that the input is valid, and the information about the statement will not be revealed.

   Definition 6. Zero-knowledge can be defined through a game where three parties are involved: *Prover*, *Verifier* and a *Simulator*. The game proceeds such that the *Verifier* selects a random challenge c and the *Simulator* interacts with *Verifier* without access to the *Prover*. It constructs a transcript of the interaction that is statistically indistinguishable from the transcript of the real interaction between the *Verifier* and *Prover*. Finally, the *Verifier* cannot determine whether the transcript was generated by the *Simulator* or by a real interaction with the *Prover*. Hence, the zero-knowledge property is satisfied if, for every *Verifier*, there exists a *Simulator* that can construct a valid transcript indistinguishable from a real interaction, regardless of the *Verifier*'s knowledge or computational power. This property ensures that the *Verifier* learns no additional information about the statement's truthfulness beyond the fact that it is true.

   One can also define the *zero-knowledge* of a proof system ⟨Prover, Verifier⟩ for

a language L holds for every *Verifier*; there exists a probabilistic algorithm as *Simulator* S such that for all $x \in L$, the following random bits are computationally indistinguishable:

- After the interaction, the output of the *Verifier* with *Prover* on x.
- The output of S on x.

So in this way, the *Verifier* will only learn information from their interaction with the *Prover*, as its power is very limited in this case.

Intuitively, an interactive proof system is *zero-knowledge* if the *Verifier* does not learn anything from the interaction other than the statement is true; even the *Verifier* deviates from the protocol.

Furthermore, the Zero-Knowledge property has three variants: perfect Zero-Knowledge, statistical Zero-Knowledge, and computational Zero-Knowledge. If the probability distributions are identical, it is called *perfect Zero-knowledge*. If the two probability distributions are *statically* close, then it is referred to as *almost-perfect Zero-Knowledge*. On the other hand, if the computation can not distinguish the two probability distributions, then it is called *computational Zero-Knowledge*. The following are the formal definitions of the invariants.

Definition 7 (Perfect Zero-Knowledge)**.** A proof system $\langle \mathrm{P}, \mathrm{V} \rangle$ for a language L is said to be perfect Zero-Knowledge if there exists an efficient probabilistic polynomial-time simulator S that, for every $x \in L$, can generate a transcript that is perfectly indistinguishable from a real interaction between an honest *Prover* and *Verifier*, without revealing any additional information about x.

So, for all efficient verifiers V' and for every $x \in L$, the following holds:

$$Pr[V'(x, P(x)) = 1] \approx Pr[V'(x, S(x)) = 1],$$

where Pr is the probability and $\approx$ denotes computational indistinguishability. The $V'(x, P(x))$ denotes the output of *Verifier* V' when interacting with the honest *Prover* P on input x, and $V'(x, S(x))$ denotes the output of *Verifier* V' when interacting with the simulator S on input x.

Now, this equation implies that *Verifier* V' accepting the proof generated by the honest *Prover* and the probability of *Verifier* V' accepting the proof generated by the simulator are computationally indistinguishable. This means that no matter how sophisticated the *Verifier* V' is, there exists a simulator S that can produce an interactive transcript that is indistinguishable from a real interaction, providing perfect Zero-Knowledge.

The concept of perfect Zero-Knowledge ensures that the *Verifier* learns absolutely nothing about the witness or any additional information that could be used to deduce it. The proof system is designed in a way that the *Verifier*'s knowledge remains completely unchanged, even in theory, thereby preserving the highest level of privacy and confidentiality.

The following definition is the definition of statistical Zero-Knowledge.

Definition 8 (Statistical Zero-Knowledge)**.** A proof system $\langle \mathrm{P}, \mathrm{V} \rangle$ for a language L is said to be statistical Zero-Knowledge if there exists an efficient probabilistic polynomial-time simulator S, a negligible function ϵ, and a security parameter n, such that, for all efficient verifiers V' and for every $x \in L$.

So, the following holds:

$$|Pr[V'(x, P(x)) = 1] - Pr[V'(x, S(x)) = 1]| \leq \epsilon(n),$$

where Pr denotes probability, $V'(x, P(x))$ denotes the output of *Verifier* V' when interacting with the *Prover* P on input x, $V'(x, S(x))$ denotes the output of *Verifier* V' when interacting with the simulator S on input x.

Now, this equation implies that the statistical distance between the two distributions is negligibly small as the security parameter n increases. The definition ensures that the Verifier's knowledge of the witness, obtained through the proof system, is indistinguishable from the knowledge that could have been gained by interacting with the simulator alone.

The following definition is the definition of computational Zero-Knowledge.

Definition 9 (Computational Zero-Knowledge)**.** A proof system ($\langle \mathrm{P}, \mathrm{V} \rangle$ for a language L is said to be a computational Zero-Knowledge if, for every probabilistic polynomial-time algorithm V', there exists an efficient probabilistic polynomial-time simulator S,

a polynomial function p and security parameter n, such that, for every $x \in L$, the following holds:

$$|Pr[V'(x, P(x)) = 1] - Pr[V'(x, S(x)) = 1]| \leq 1/p(n),$$

where Pr denotes probability, $V'(x, P(x))$ denotes the output of *Verifier* V' when interacting with the *Prover* P on input x, $V'(x, S(x))$ denotes the output of *Verifier* V' when interacting with the simulator S on input x.

Now, the equation implies that the absolute difference between the probability of *Verifier* V' accepting the proof generated by the honest *Prover* and the probability of *Verifier* V^* accepting the proof generated by the simulator is bounded by the reciprocal of a polynomial function $p(n)$.

The definition ensures that regardless of the probabilistic polynomial-time algorithm used by the *Verifier*, there exists a simulator which is capable of producing a proof that is indistinguishable from the one produced by the *Prover*. Although the *Verifier* may have some knowledge or information after the interaction, this knowledge is computationally negligible and difficult to extract useful information from.

Alternatively, Blum, Feldman, and Micali [42] proposed another model called *non-interactive zero-knowledge proofs* (NIZK). In this model, a trusted third party, *the dealer*, generates a *reference string* randomly and shares it between the *Prover* and *Verifier*. After generating the reference string, the *Prover* sends a single message to the *Verifier*, who will decide whether to accept or reject. So, NIZK and ZKP are related cryptographic concepts that share some similarities but also have distinct differences in their properties.

Interactivity: One key distinction is the level of interactivity required in the protocols. In ZKP, multiple rounds of interaction occur between the *Prover* and *Verifier*, with the *Prover* providing responses to challenges posed by the *Verifier*. On the other hand, NIZK protocols are designed to be non-interactive, meaning they require only a single round of interaction. The *Prover* generates a proof independently and sends it to the *Verifier* without the need for further communication.

Efficiency: NIZK protocols are generally more efficient than ZKP protocols in terms of computational overhead and communication complexity. The non-interactive nature of NIZK allows for streamlined execution without the need for real-time back-and-forth exchanges, resulting in faster and more practical implementations.

Soundness: Both NIZK and ZKP protocols aim to achieve soundness, which ensures that a dishonest *Prover* cannot convince the *Verifier* of a false statement. However, the level of soundness may vary depending on the specific protocol and the underlying assumptions. In general, NIZK protocols strive for soundness under computational assumptions, whereas ZKP protocols often aim for perfect soundness, meaning that even computationally unbounded adversaries cannot be deceived.

Completenes and Zero-Knowledge: Both NIZK and ZKP protocols possess completeness, meaning that an honest *Prover* can convince an honest *Verifier* of a true statement. Additionally, both aim to achieve zero-knowledge, where the protocol reveals no additional information other than the validity of the statement. However, the specific techniques and assumptions used to achieve zero-knowledge may differ between NIZK and ZKP protocols.

Overall, the main difference lies in the level of interactivity and the resulting efficiency of the protocols. NIZK protocols prioritise non-interactivity for efficiency gains, while ZKP protocols involve multiple rounds of interaction at the expense of increased computational and communication overhead. Moreover, the interest of this thesis is more in *special soundness* and *honest verifier zero-knowledge*. The informal definitions of these two properties are defined below.

Definition 10 (Special Soundness in NIZK)**.** A NIZK protocol satisfies the Special Soundness property if, for any efficient adversary that can produce two accepting transcripts for the same statement, there exists an efficient algorithm that can extract the *Prover*'s secret witness information without knowing the secret witness itself.

The Special Soundness property ensures that if an adversary can produce multiple valid proofs for the same statement, it must have access to the secret witness information used in the proofs. The existence of an efficient extraction algorithm that can recover the secret witness from two accepting transcripts guarantees that the protocol is secure against malicious provers who attempt to generate multiple convincing proofs without possessing the necessary knowledge of the secret witness.

Definition 11 (Honest Verifier Zero-Knowledge in NIZK)**.** A NIZK protocol satisfies Honest Verifier Zero-Knowledge if, for every efficient *Verifier* algorithm V, there exists an efficient simulator algorithm S that, given an input statement, can produce a

transcript that is indistinguishable from the transcript obtained by interacting with an honest *Prover*. This indistinguishability holds even if V is replaced with any efficient adversary algorithm that interacts with S.

In other words, the honest *Verifier*, interacting with the real *Prover*, cannot distinguish between the actual execution of the protocol and interaction with the simulator. The simulator S is capable of generating transcripts that are indistinguishable from those produced during a real interaction without having access to the *Prover*'s secret information or the witness used in the proof. The property ensures that the *Verifier* does not learn any additional information beyond the statement's validity, even against a computationally unbounded adversary.

Earlier, we have seen how the Fiat-Shamir transformation technique converted interactive identification schemes, such as the Schnorr Identification Scheme, into non-interactive signature schemes. This transformation replaces the interactive challenge generation step with a deterministic computation using a hash function. One can apply the Fiat-Shamir transformation to convert the Schnorr Identification Scheme into a signature scheme. Additionally, the non-interactive zero-knowledge (NIZK) property is closely related to this transformation. The Fiat-Shamir transformation is a key component in constructing NIZK proofs. By applying the transform to interactive protocols, one can convert them into non-interactive proofs, achieving the NIZK property. In the context of Schnorr's protocol, applying the Fiat-Shamir transformation turns it into a non-interactive signature scheme that possesses the NIZK property. The NIZK property ensures that proof can be generated without any further interaction between the prover and verifier. It allows the prover to convince the verifier of a statement's truthfulness without revealing any additional information. The Fiat-Shamir transformation enables the conversion of interactive identification schemes into NIZK proofs, facilitating non-interactive protocols while preserving the security properties of the original scheme.

2.4 Background Definitions

Having gained an understanding of the foundational concepts and role of signature schemes in modern cryptography, we now turn the reader's attention to the formal definition of the signature scheme and the Computational Diffie-Hellman (CDH) problem. The CDH plays a significant role in signature schemes, and it is a mathematical problem which serves as the foundation for the security of the signature schemes.

The following subsections define the formal definitions of the Computational Diffie-Hellman problem, the signature scheme and the existential unforgeability under chosen message attack (EUF-CMA). Furthermore, the thesis also formally defines the EUF-CMA in the random oracle model. These two definitions are formally defined with code-based game-playing proofs to emphasise the need for a precise and rigorous specification of the cryptographic primitive. However, the thesis first briefly explains the code-based game-playing techniques concerning its security notion so that one can understand the formal definitions defined in this chapter. Nonetheless, these brief introductions will be explained in more detail in Chapter 3.

Code-Based Game-Playing Proofs and Exact Security

Cryptographic proofs often rely on complexity theoretic *reductions*, constructing an adversary against some hardness (or statistical) *assumption* from an adversary against the construction under study.

If early cryptographic proofs were indeed presented much like complexity theoretic reductions, the complexity of modern assumptions and constructions, the difficulty of reasoning rigorously about probabilistic algorithms, and the desire to use security proofs to inform parameter selection (and in particular key sizes) have led to a shift towards code-based game-playing security proofs [15]. These proofs still construct a reduction, but do so step-by-step—with each step called a *game*, allowing careful reasoning about the relation between the probabilities of events in successive games, and isolating complex events whose probability must be reasoned about.

Further, assumptions, constructions, security notions, and intermediate games are expressed as simple programs. This provides clarity in the definitions, but also helps reason about the *complexity* of the reduction and its *tightness*, stepping away from asymptotic results and providing concrete (or exact) security claims for given security parameters.

This research does not use the full power of the methodology. Instead, it will only rely on proving statements of the following form, with $\mathcal{D}_1$ and $\mathcal{D}_2$ distributions, and E_1 and E_2 events defined over the relevant distribution's support.

$$\Pr[\mathcal{D}_1 : E_1] \leq \Pr[\mathcal{D}_2 : E_2]$$

In places, it will also prove statements that involve some additional *failure event* F_2

defined over the support of $\mathcal{D}_2$.

$$\Pr[\mathcal{D}_1 : E_1] \leq \Pr[\mathcal{D}_2 : E_2] + \Pr[\mathcal{D}_2 : F_2]$$

It will do so simply by proving the following and applying the union bound.

$$\Pr[\mathcal{D}_1 : E_1] \leq \Pr[\mathcal{D}_2 : E_2 \vee F_2]$$

The distributions we consider are quite often defined by probabilistic programs with specified inputs. Code-based game-playing proofs, and their formalisation in EasyCrypt, leverage this by allowing a proof relating two distributions to be lifted from a proof relating the two programs that produce them.

EasyCrypt

This research utilises EasyCrypt[1] [43], which is an interactive proof assistant designed to construct machine-checked game-based cryptographic proofs. Cryptographic games and definitions are modelled as probabilistic procedures (with random sampling, conditional statements, loops and procedure calls) over a language of expressions that can be user-extended with various mathematical and data structures.

In addition to the usual features of an interactive proof assistant, EasyCrypt features specialized program logics to reason about probabilistic equivalences between programs (in the *probabilistic Relational Hoare Logic*; pRHL), and to directly reason about probabilities of events in given programs (in the *probabilistic Hoare Logic*; or pHL). We now explain how statements in these logics map to the practice of game-based proofs.

pRHL Judgments

A pRHL judgment of the form

$$\{\Phi\}\ c_1 \sim c_2\ \{\Psi\}$$

where c_1 and c_2 are programs, and the *precondition* and *postcondition* Φ and Ψ (respectively) are *relations* between the memories of c_1 and c_2, can be used to prove—for any

[1] `https://www.easycrypt.info`

pair of memories m_1, m_2 such that $m_1 \; \Phi \; m_2$—that

$$\Pr[c_1 \mathbin{@} m_1 : E_1] \leq \Pr[c_2 \mathbin{@} m_2 : E_2]$$

whenever for any pair of memories m_1' and m_2', such that $m_1' \; \Psi \; m_2'$, we have $E_1(m_1') \Rightarrow E_2(m_2')$. The notation $c \mathbin{@} m$ defines a distribution over final states by defining some code c and an initial memory m in which that code is executed. In the following and in our formal definitions, we always consider top-level games whose behaviour is independent of the initial memory (by ensuring that all variables are defined before use). We therefore omit initial memories where possible in the remainder of this paper.

pRHL for equivalence up to failure For readers who wish to match paper statements here to the formal development, we note that a single pRHL judgement can be used to discharge several statements on probabilities. Anticipating slightly on discussions of the proofs, this is useful in proving statements of the following form when the failure event F is defined over the support of both distributions.

$$\Pr[c_1 \mathbin{@} m_1 : E_1] \leq \Pr[c_2 \mathbin{@} m_2 : E_2] + \Pr[c_1 \mathbin{@} m_1 : F]$$

Indeed, to obtain such a result, a single pRHL judgement—such that the post-condition Ψ implies both that the failure event F occurs with the same probability in c_1 and c_2, and that whenever E_1 holds in c_1, then either E_2 or F occurred in c_2—is sufficient.[1]

pHL Judgments

A pHL judgment of the form

$$\{\Phi\} \; c \; \{\Psi\} \diamond b$$

where c is a program, the precondition and postcondition Φ and Ψ are predicates over the memory of c, $b \in [0,1]$ is some probability bound, and $\diamond \in \{=, \leq, \geq\}$ is some relation, can be used to prove—for any memory m such that $\Phi \; m$—that

$$\Pr[c \mathbin{@} m : \Psi] \diamond b$$

[1] The full formal interpretation of pRHL judgments—as originally given by Barthe et al. [10]—allows other deductions, which are not used in this paper.

Proofs of such statements are relatively straightforward when the event Ψ occurs in the program's main component, but involve invocations of the *failure event lemma* when the event is triggered by oracle queries. The failure event lemma, in the proof we describe below, allows us to lift a bound on the probability of an event occurring during any one oracle query into a bound on the probability of that event occurring during the run of the experiment.

As discussed in relation to game-based proofs, these two ingredients suffice to follow the proofs given below. We note that, although we do not give details, all statements below are given proofs that are fully machine-checked in EasyCrypt. We instead focus our writing on giving an intuition of the reasoning formalised in those machine-checked proofs, and of the aspects of it that might be abstracted into higher-level proof principles.

Now, before defining the formal definition of the signature scheme, the following definition defines the Computational Diffie-Hellman (CDH) problem.

2.4.1 The Computational Diffie-Hellman Assumption

The security of the schemes considered in this work relies on the hardness of the Computational Diffie-Hellman (CDH) problem, of finding g^{ab} given g^a and g^b. We define this assumption more formally, aligning it with principles of concrete security, and first defining the advantage of a constrained adversary in solving CDH.

The following formal definitions are extracted from our conference paper [44].

Definition 12 (CDH advantage)**.** Let G be a cyclic group of finite order n, and g_G be a generator of G. Let $\mathcal{A}$ be a computational Diffie-Hellman adversary that, on input a description of G, n, g_G, and g_G^a and g_G^b for uniformly chosen exponents a and b, returns an element $\mathsf{r} \in G$. The CDH advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{\mathsf{cdh}}_{G,g_G,n}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}}(G, g_G, n) : \mathsf{r} = g_G^{\mathsf{ab}}\right]$$

where the experiment $\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}}(G, g_G, n)$ is defined in Figure 2.6.

The definition of hardness is natural.

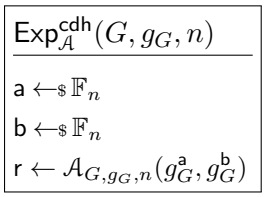

Figure 2.6: The CDH experiment.

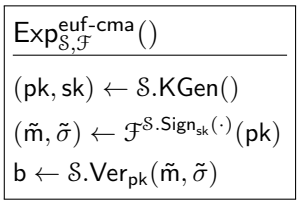

Figure 2.7: The EUF-CMA experiment.

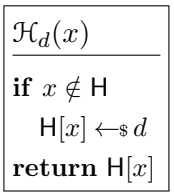

Figure 2.8: A Random Oracle that lazily samples its responses from some distribution d. H is a map or association list, initially empty.

Definition 13 (Hardness of CDH)**.** The CDH problem is said to be (t, ϵ)-hard in G with generator g_G if, for any adversary $\mathcal{A}$ that runs in time at most t, we have

$$\mathsf{Adv}^{\mathsf{cdh}}_{G,g_G,n}(\mathcal{A}) \leq \epsilon$$

The research will be more particularly interested in the hardness of the CDH problem in our cyclic group $\mathbb{G}$ of prime order q with generator g, namely $\mathsf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g,q}(\mathcal{A})$.

In the security model of the signature scheme, a challenger plays a game against the adversary. The challenger generates the signature scheme during the interaction, and the adversary constantly tries to break it. The challenger first generates $(\mathsf{pk}, \mathsf{sk})$, keeps her secret key, and sends the public key to the adversary. The adversary has the liberty to choose adaptively on any messages to make signature queries.

Next, the formal definition of the signature can be defined as the following.

2.4.2 Signature Scheme

Definition 14. A signature scheme consists of three algorithms key generation (KGen), signing (Sign), and verification (Ver), such that $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ for messages in some set $\mathcal{M}$:

- A key generation algorithm KGen that outputs a key pair composed of a public

(pk) and private key (sk);

- A signing algorithm Sign that, upon input a private key sk and a message $\mathsf{m} \in \mathcal{M}$, returns a signature σ;

- A verification algorithm Ver that, upon input a public key pk, a message $\mathsf{m} \in \mathcal{M}$ and signature σ, returns a boolean signifying acceptance (with value true, denoted with 1) or rejection (with value false, denoted with 0).

As is standard in cryptography, the desired security property is captured as a game between an adversary attempting to break the security of the scheme (by forging a valid signature) and a challenger that mediates interactions between the adversary and the signature scheme under study to specify adversarial powers and prevent trivial wins.

More specifically, we consider the notion of *existential forgery under adaptive chosen message attacks* ($\mathsf{EUF\text{-}CMA}$), in which the adversary—given the public key and oracle access to a signing oracle that signs messages of the adversary's choice under the challenger's key—attempts to produce a valid signature on a *fresh* message—that is, one that has not been queried to the signing oracle. As for CDH, we first define the advantage of an adversary in breaking the security of a signature scheme.

Definition 15 ($\mathsf{EUF\text{-}CMA}$ advantage)**.** Given a signature scheme $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ and an adversary (or *forger*) $\mathcal{F}$ that, upon input a public key for $\mathcal{S}$ and given oracle access to a signing oracle for $\mathcal{S}$ (which produces a signature upon input a message), the advantage of $\mathcal{A}$ in breaking the $\mathsf{EUF\text{-}CMA}$ security of $\mathcal{S}$ is

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathcal{S}}(\mathcal{F}) := \Pr\left[\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{S},\mathcal{F}}() : \mathsf{b} \wedge \tilde{\mathsf{m}} \notin \mathcal{Q}_{\mathcal{S}}\right]$$

where the experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{S},\mathcal{F}}()$ is defined in Figure 2.7, and $\mathcal{Q}_{\mathcal{S}}$ is the set of queries issued by the forger $\mathcal{F}$ to its signing oracle.

We can then naturally define $\mathsf{EUF\text{-}CMA}$ security as the relevant hardness notion.

Definition 16 (EUF-CMA security)**.** A signature scheme $\mathcal{S}$ is said to be $(t, q_{\mathcal{S}}, \epsilon)$-*EUF-CMA-secure* if, for any adversary $\mathcal{A}$ that runs in time at most t, making at most $q_{\mathcal{S}}$ queries to its $\mathcal{S}$ oracle, we have

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathcal{S}}(\mathcal{A}) \leq \epsilon$$

2.4.3 EUF-CMA security in the ROM

As mentioned about the the Random Oracle Model (ROM) before, it was first introduced by Bellare and Rogaway [31] as a tool to reason about the security of efficient cryptographic constructions that make use of hash functions. In the ROM, hash functions are modelled as *public* oracles that compute a function sampled uniformly from the appropriate function space at the beginning of the experiment. Formally, we equivalently capture random oracles as stateful algorithms whose outputs to *unique* requests are sampled following some distribution, as shown in Figure 2.8. In the following, we consider uniform random oracles (where d is the uniform distribution over the relevant (finite) output space).

When analysing the security of a scheme that uses hash functions in the ROM, it is crucial to ensure that the adversary is given oracle access also to the hash functions. All the schemes we consider in this paper make use of two distinct hash functions, and we therefore consider forgers who have access to two random oracles $\mathcal{H}$ and $\mathcal{G}$ in addition to the signing oracle. The resulting security experiment is shown in Figure 2.9. In such a context, the number of queries the forger makes to the random oracles should also be bounded as a resource; in the following we consider a notion of $(t, q_{\mathcal{H}}, q_{\mathcal{G}}, q_{\mathcal{S}}, \epsilon)$-EUF-CMA security naturally extending that from Definitions 15 and 16.

Definition 17 (EUF-CMA security in the ROM)**.** Given two random oracles $\mathcal{H}$ and $\mathcal{G}$ and a signature scheme $\mathcal{S}^{\mathcal{H},\mathcal{G}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ and an adversary (or *forger*) $\mathcal{F}$ that, upon input a public key for $\mathcal{S}$ and given oracle access to $\mathcal{H}$, $\mathcal{G}$, and to a signing oracle for $\mathcal{S}^{\mathcal{H},\mathcal{G}}$ (which produces a signature upon input a message), the advantage of $\mathcal{A}$ in

$\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{H},\mathcal{G},\mathcal{S},\mathcal{F}}()$

$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{S}.\mathsf{KGen}()$

$(\tilde{\mathsf{m}},\tilde{\sigma}) \leftarrow \mathcal{F}^{\mathcal{H},\mathcal{G},\mathcal{S}.\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$

$\mathsf{b} \leftarrow \mathcal{S}.\mathsf{Ver}_{\mathsf{pk}}(\tilde{\mathsf{m}},\tilde{\sigma})$

Figure 2.9: EUF-CMA with two Random Oracles $\mathcal{H}$ and $\mathcal{G}$.

breaking the EUF-CMA security of $\mathcal{S}$ in the ROM is

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathcal{H},\mathcal{G},\mathcal{S}}(\mathcal{F}) := \Pr\left[\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{H},\mathcal{G},\mathcal{S},\mathcal{F}}() : \mathsf{b} \wedge \tilde{\mathsf{m}} \notin \mathcal{Q}_{\mathcal{S}}\right]$$

where the experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{H},\mathcal{G},\mathcal{S},\mathcal{F}}()$ is that defined in Figure 2.9, and $\mathcal{Q}_{\mathcal{S}}$ is the set of queries issued by the forger $\mathcal{F}$ to its signing oracle. A signature scheme $\mathcal{S}$ is said to be $(t, q_{\mathcal{H}}, q_{\mathcal{G}}, q_{\mathcal{S}}, \epsilon)$-*EUF-CMA-secure* in the ROM if, for any adversary $\mathcal{A}$ that runs in time at most t, making at most $q_{\mathcal{H}}$ (resp. $q_{\mathcal{G}}$, $q_{\mathcal{S}}$) queries to its $\mathcal{H}$ (resp. $\mathcal{G}$, $\mathcal{S}$) oracle, we have

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathcal{H},\mathcal{G},\mathcal{S}}(\mathcal{A}) \leq \epsilon$$

Although this is not important in this paper, we note in particular the importance—in general—of exposing these two oracles also to any adversary against a construction that *uses* any of the signature schemes we prove secure here. The reductions we formally prove here require *control* over both ROs, which must thus—in general—be independent from any ROs taken over by any further reductions—for example, if the schemes are used in larger protocols.

2.5 Discussion

Several signature schemes are proved in the standard model [45, 46, 47, 48] and the Random Oracle model [49, 50, 51, 52, 53]. The signature schemes based on the RSA [54] mostly seem secure with strong underlying assumptions in the standard models [55, 56]. The standard model is where adversary (attacker) $\mathcal{A}$ is restricted by some time she takes to break the system. In contrast, the ROM restricts the adversary, and

one can assume that one of the scheme's hash functions is set as a RO controlled by the simulator. The first efficient signature scheme whose security is tightly related to the RSA assumption [54], in the Random Oracle model, is the scheme of Bellare and Rogaway [57], where they describe an RSA-based signing scheme. The literature shows that the shortest signature scheme in the Random Oracle model is the BLS scheme [49].

It is assumed that the Diffie-Hellman problems [21] (Computational Diffie-Hellman (CDH) and Decisional Diffie-Hellman (DDH)) are stronger than the discrete logarithm problem. There is no known reason to solve the Diffie Hellman problems faster than the discrete logarithm problem [58, 59]. Furthermore, there are particular shreds of evidence that the CDH problem may be equivalent to the discrete logarithm problem in specific groups [58, 59, 60].

Earlier, the Schnorr scheme was based on the hardness of the discrete logarithm, and the security is assessed with a forking lemma (relies on the proof of knowledge property of an interactive proof system). The drawback of using the forking lemma lands into loose security reduction. In contrast, this thesis shows the formal proofs and the formalisation of the scheme based on the CDH problem, and the security notion relies on the property of soundness (Chapter 4) and honest-verifier zero knowledge (Chapter 6).

In cryptography, it is assumed that the hardness of the CDH assumption is closely related to the hardness of the discrete logarithm assumption. The digital signatures can be constructed in the discrete logarithm settings [60]. The signature schemes based on the discrete logarithm are Schnorr [20], DSS [61] and ElGamal [62]. These signature schemes either involve loose security or require non-standard assumptions. Note that the scheme's security is tight when the success probability is close to the adversary's success probability; otherwise, the scheme is loose (explained earlier). In , Goh and Jarecki [16] proposed a signature scheme (EDL), which claimed that they offer better security guarantees than existing discrete logarithm-based signature schemes. The security of the EDL scheme showed that it is tightly related to the CDH problem in the Random Oracle model. However, the drawback of the EDL scheme is that the signature length is bigger than the length of traditional signature schemes, meaning that signature schemes require more exponentiations. In public-key cryptography, discrete exponentiation is a costly computational component because it dominates the computational cost. Later, in 2005, Chevallier-Mames [17] shortened the size of the EDL scheme. He proposed a new EDL scheme (this thesis calls the new EDL scheme by Chevallier-Mames as CM scheme), whose security is also tightly related to the CDH problem in the Random Oracle model. The CM scheme is efficient, and the signature size is smaller than the EDL scheme, resulting in 25% efficiency on the discrete logarithm-based schemes. Katz and Wang [63] modified the EDL scheme and improved the signature's efficiency and length. However, their security is tightly related to the hardness of the decisional

Diffie-Hellman (DDH) problem [64], and the scheme does not have random salt. In 2007, Katz and Wang collaborated with the authors (Goh and Jarecki) of the EDL scheme. Goh et al. [6] proposed two schemes whose security is tightly related to both Diffie-Hellman problems (CDH and DDH) in the Random Oracle model. The security of the first scheme relies on the hardness of the CDH problem, and the second scheme relies on the DDH problem. They claimed that their schemes present better efficiency than the existing schemes based on the discrete logarithm problem.

Chapter 3

Transforming security proofs into semantic framework games

This chapter introduces the necessary background of game-based proofs in the cryptography literature. Then, it introduces higher-order logic and state-of-the-art EasyCrypt. This chapter delves into the syntax and semantics of EasyCrypt, focusing on reasoning principles.

3.1 Introduction

Last few decades, provable security has been extensively used to support emerging standards. However, the difficulty arose with validating the proofs when Shoup [65] discovered the gap in the security proof of OAEP against adaptive chosen-ciphertext attacks by Bellare and Rogaway [66]. In 1994, Bellare-Rogaway presented a well-written but short proof that became famous and became part of the SET electronic payment standard of Visa and MasterCard [67]. This interesting proof was unexamined for seven years and became apparent when Shoup found the gap. The exciting history of OAEP security proof shows that it was accepted initially, but the security was not read carefully with a critical eye. Now, one must wonder how many details and validation are required to present and check the reductionist arguments and, more importantly, how many famous/infamous proofs are overlooked. To address the above difficulties of

checking the complex arguments in the proofs, many suggested and proposed various techniques to structure the proofs and get correct proofs. For instance, the proposed techniques are game-based proofs [15, 68], a universal composability framework [69], and Maurer's constructive cryptography [70]. This thesis reports on the techniques of the Game-based proofs as the proof assistant EasyCrypt is based on the sequence of games formalism. The most significant difference in this technique is that the game's goal is not to specify every possible security definition with explicit *Initial* and the *Final* step but to represent all the security definitions as indistinguishability of two games, even if the definitions are regarding the property of unforgeability. Nevertheless, this technique's property of unforgeability can yet be reasoned. For instance, to declare that no adversary can forge a MAC, it is sufficient to state that no adversary can differentiate a subroutine of the MAC(in the verification) from a subroutine that yields false.

3.2 Formalisms for Security Definitions

Modern 21st-century cryptography has very much developed lately. Now, the proofs are proved so that code-makers can win against the code-breakers. The proofs are reasoned with formal definitions of what it signifies to be secure and defining insecurity employing attacks against the formal security definitions. The definition of security must guarantee the behaviour of the cryptosystem in the existence of an adversary. It is important to note that the security definitions assume the adversary's view of the cryptosystem, and the adversary can observe what she is exposed to. For instance, consider the scenarios where the adversary observes ciphertexts and sees how ciphertexts are generated.

The key is generated with the *key generation algorithm* KGen, and the private key is kept secret from the adversary. In the encryption scheme, the key is used to encrypt a plaintext. The adversary considers picking the plaintexts, permitting her with some power. Nonetheless, if the encryption scheme is secure whilst the adversary picks the plaintexts, it is indeed secure in real systems where the adversary has some uncertainty about the plaintexts. Consider the adversary (a calling program) to the following *CTXT* subroutine to assume the above details 3.1.

CTXT ($\mathsf{m} \in \Sigma.\mathcal{M}$):

$k \leftarrow_{\$} \Sigma.\mathsf{KGen}$;
$c := \Sigma.\mathsf{Enc}(k, \mathsf{m})$
return c

Figure 3.1: $\mathcal{A}$ calling program to subroutine

A calling program picks the plaintext to the subroutine and can only see the ciphertext and cannot see the private k. The fresh k is chosen each time the calling program constructs a couple of calls to the subroutine.

Next, the following diagrams show the interaction between the calling program (adversary) and the CTXT subroutine, and the output of the CTXT subroutine is uniformly distributed. This scenario shows that the CTXT subroutine has the identical impact on every calling program as a CTXT subroutine that samples its result uniformly.

CTXT ($\mathsf{m} \in \Sigma.\mathcal{M}$):	versus	CTXT ($\mathsf{m} \in \Sigma.\mathcal{M}$):
$k \leftarrow_{\$} \Sigma.\mathsf{KGen}$;		$c \leftarrow \Sigma.C$;
$c := \Sigma.\mathsf{Enc}(k, \mathsf{m})$		**return** c
return c		

Figure 3.2: Interaction b/w calling program and subroutine

Supposedly, there should be no way for the calling program (adversary) to determine which of these two executions response to subroutine calls.

The following section begins by revisiting the semantics of programming language.

3.3 The formal semantics of Programming language

The theoretical department of Computer Science has developed with great exposure lately. Their recognised works have established many areas such as formal language, computational complexity and automata theory. The progression of programming language semantics in cryptography expanded, and a programming language's formal semantics is involved in constructing a mathematical model. The purpose is to operate as a foundation to apprehend and maintain the program's behaviour with reasoning. The mathematical models are helpful for analysis and verification and, at a more initial level, because developing the program's definition exposes all types of subtleties of which it is crucial to be mindful.

Before explaining the semantics, it is essential to present the notation and concepts of informal, logical and set theories used to reason the security statements.

3.3.1 Summary of Notation

For statements A and B, the notations are:

$A \wedge B$	and
$A \vee B$	or
$A \Rightarrow B$	implies
$\neg A$	not
$A \Longleftrightarrow B$	A if and only if B
$\forall x.A(x)$	for all
$\exists x.A(x)$	exists
$a \in A$	element of
$\{a_1, ..., a_n\}$	the set with elements $a_1, ..., a_n$
$dom(s)$	set of elements in the domain of s
$\mathbb{Z} = \{.., -1, 0, 1, ...\}$	the set of integers
$x \in \mathbb{Z}$	integer

There are three tiers of the semantics of programming language; operational semantics, denotational semantics, and axiomatic semantics, which are defined next.

3.3.2 Operational Semantics

It defines a programming language by specifying how it can be conducted on an abstract machine. This semantic is helpful in implementation. The structure of a programming language, a small language of while programs, is called imperative language (IMP). It is called IMP because the program executes a sequence of involved commands to change the states.

The operational semantics of the programs begins with some formation rules.

$$a ::= b \mid c \mid skip.$$

The symbol ::= can be read as "can be" and the symbol | as "or".

The function of commands and programs is to enforce the change in their states. When the IMP program is executed then is assumed that the initial state is set to zero. One can define a relation if a pair $\langle c, s\rangle$ expresses the format from which it stays to run command n from state s with basic operations.

$$(op +) \;\; \langle c_1 + c_2, s\rangle \rightarrow \langle c, s\rangle \;\; \text{if } c = c_1 + c_2$$

$$(op \ \geq) \ \langle c_1 \geq c_2, s\rangle \rightarrow \langle b, s\rangle \ \text{ if } b = c_1 \geq c_2$$

Rules for commands

Atomic commands

$$\langle \text{skip}, s\rangle \rightarrow s$$

Sequencing

$$\frac{\langle c_0, s\rangle \rightarrow s'' \qquad \langle c_1, s''\rangle \rightarrow s'}{\langle c_0; c_1, s\rangle \rightarrow s'}$$

Conditional

$$\frac{\langle b, s\rangle \rightarrow \text{true} \qquad \langle c_0, s\rangle \rightarrow s'}{\langle \text{if } b \text{ then } c_0 \text{ else } c_1, s\rangle \rightarrow s'}$$

3.3.3 Denotational semantics

It is a technique to define the connotation of programming languages and employs more abstract mathematical concepts of continuous functions. This semantic gives the most profound and widely used functional techniques, braced by advanced mathematical theories.

The denotational semantics uses an emphatic bracket $[\![]\!]$ to diverge the syntactic from the semantics. For instance, the expression ("6+2")$s = 8$ can be written as $[\![6+2]\!]s = 8$.

$$C[\![skip]\!] = \{(s, s)|s \in \Sigma\}$$

$$C[\![c_0, c_1]\!] = C[\![c_1]\!] \circ C[\![c_0]\!]$$

$$C[\![\text{if } b \text{ then } c_0 \text{ else } c_1]\!] = \{(s, s')|\beta[\![b]\!]s = \text{true } \&(s, s') \in C[\![c_0]\!]\} \cup \{(s, s')|\beta[\![b]\!]s = \text{false } \&(s, s') \in C[\![c_1]\!]\}$$

3.3.4 Axiomatic semantics

The axiomatic semantics is more abstract than the denotational ones, based on logical deduction from the predicates. It provides proof rules within a program logic to revise

the definition of a programming construction. The axiomatic semantics highlights the proof's correctness and deliver elegant proofs, which can help construct and verify the programs. This semantic is also called logical, which can directly define a program's meaning by delivering the axioms of the logic of program statements. It is not wrong to say that this semantic has an appealing technique because it gives bug-free algorithms by proving the algorithm correct.

The axiomatic semantics aims to deliver axioms and proof rules that can catch each command's intended meaning in the programming language. The semantics of the program can be described by *assertion* (logical formula assembled employing the variables), which always becomes accurate when the program comes to the assertion point. For instance, the correctness of program C can be defined with an assertion; *precondition* and *postcondition*. The precondition is placed before the program and postcondition after.

$$\{Pre\} \quad C \quad \{Post\}.$$

The assertion (pre and post condition) is called the program's *specification*. So, given pre and post conditions, program C is correct w.r.t. the specification, provided that the program is executed, satisfying the precondition, terminating the program and the results satisfying postcondition.

One can not view the above three tiers of semantics as opposed to each other. It is just because all three tiers are dependent on each other. For instance, demonstrating that proof rules of axiomatic semantics are written correctly, it trusts the underlying operational or denotational semantics.

The following section begins by reviewing game-based proofs.

3.4 Game-based proofs

The security proofs of the cryptographic protocols are analysed and specified with several methodologies. The two popular methodologies are; game-based proof and simulation-based proof. This thesis focuses only on the former approach. The approach of game-based proof is classified into two categories, game-hopping and security reduction. The security reduction was discussed in the previous chapter, and now, the discussion begins with game-based proofs.

The technique of game hopping originated to tame the complexities of the cryptographic security proofs [71, 72]. Kilian and Rogaway [73] presented the idea of a game as a program written in a semi-formal language.

Later in 2004, Bellare and Rogaway [68] laid the foundation of the game-playing technique and suggested games as programs, where the game is considered with a collection of procedures. The adversary is also viewed as a program consisting of a single procedure. They believe their technique can lead to significant results and is more easily verifiable with minor error-prone proofs in cryptography. Under their framework, the game includes three procedures: initialise, oracle named, and finalise. The adversary has access to the oracles and can make queries; the initialising and finalising procedures can be absent. The game has global variables unavailable to the adversary, but the adversary's variables are local.

From the idea of Bellare and Rogaway [68], in 2005, Halevi [7] envisioned the idea of a proof checker. He advocated developing an automated framework to help cryptographers with their security statements in their proofs. In contrast to Halevi's idea, Shoup [15] opposed being limited to a tool. Instead, he suggested that games be the extent of their PL for their probability spaces. He suggested that the core idea of the game-based proof is that cryptographic primitives should be defined as an attacking game between the adversary and the challenger. Both parties are probabilistic and can communicate with each other. Then one can model the game with some probability space. So the proof's definitions can become easier to check and verifiable.

So to reflect on Shoup's idea, one can assume that there exists an adversary who can break the proposed scheme in some polynomial time with some advantage of winning under the related security model. The security reduction reduces loss and cost whilst the adversary solves the underlying hard problem. In the reduction, a simulator utilises the instance of an assigned problem to make a simulated scheme and eventually uses the adversary's attack on the scheme to answer the underlying hard problem. If the reduction loss is self-dependent on the number of queries the adversary makes, then the security reduction is *tight*; otherwise, it is *loose*.

So, for every *efficient* adversary, the probability that some particular event S happens is *very close to* some *target probability*: either 0, 1/2, or the probability of some event T in another game where the same adversary interacted with a different challenger. Here, "efficient" means the polynomial time-boundness, and "very close to" means it is less efficient.

So, a typical game in the cryptographic proof involves a challenger and an adversary interacting with each other. The adversary can be defined on what and where to query and how to win the game. So one should be careful with the adversary's interaction whilst proposing the schemes. Some proposed schemes can be proved secure in a weak security model by restricting her interaction. In contrast, the adversary in a strong security model can adaptively query more additional details than in a weak model.

So, all queries from the adversary that eventually helped her win the game must be carefully assigned; if not, she can always win the game regardless of the construction of the proposed scheme.

Basic Probability Before jumping into transitions of Game-based proofs, this subsection reminds the concept of basic probability with its notation and facts.

If the event is E, then the complement of that event is $\bar{E}$, which means $\bar{E}$ is the event that E does not happen, and the definition would be $Pr[E] = 1 - Pr[\bar{E}]$. If two events E_1 and E_2, occur, then the conjunction $E_1 \wedge E_2$ of these two events is $Pr[E_1 \wedge E_2] \leq Pr[E_1]$. If $Pr[E_1 \wedge E_2] = Pr[E_1] \cdot Pr[E_2]$ then E_1 and E_2 are independent.

If the disjunction of two Events, E_1 and E_2, occurs, i.e. $E_1 \vee E_2$, which means either E_1 happens or E_2 and, by definition, $Pr[E_1 \vee E_2] \geq Pr[E_1]$. The *union bound* is frequently a helpful upper bound of the quantity; defined next.

Definition 18 (Union Bound)**.**

$$Pr[E_1 \wedge E_2] \leq Pr[E_1] + Pr[E_2].$$

The *union bound* for the series of events $E_1, \cdots, E_k$ follows

$$Pr[\vee_{i=1}^{k} E_i] \leq \Sigma_{i=1}^{k} Pr[E_i].$$

The *conditional probability* of events i.e., $Pr[E_1|E_2]$ follows as

$$Pr[E_1|E_2] \stackrel{def}{=} \frac{Pr[E_1 \wedge E_2]}{Pr[E_2]}$$

The above statement only become true if $Pr[E_2] \neq 0$ and the probability denotes that the E_1 happens, given that E_2 has happened. It follows from the definition that

$$Pr[E_1 \wedge E_2] = Pr[E_1|E_2] \cdot Pr[E_2].$$

This equality still hold even if $Pr[E_2] = 0$ as long as the multiplication is not aggravated by zero on the right hand side.

Sequence of Games After reviewing some basic terminology and the probability bound, the game-based proofs begin by sequencing the games. To construct the sequence of games, $Game_0$, $Game_1$, . . . , $Game_n$, the $Game_0$ becomes the original game for a given adversary and the cryptographic primitives. The definition of security is bounded with some event S; for $i = 1, .., n$, the security constructions can define the event S_i in some *Game* i related to S's definition. So, to prove the security, one can show that $Pr[S_i]$ is negligibly close to $Pr[S_{i+1}]$ for $i = 0, .., n - 1$, and finally, the $Pr[S_n]$ becomes negligibly close or, in some cases, is equal to the *target probability* (n is a constant).

Shoup [15] suggested that the transition from one game to the next should be kept smaller to have simple changes at a time. He mentioned that the transition from one game to the next could be classified into three types.

3.4.1 Transition based on indistinguishability

In this transition, a small change made by the adversary implies an efficient method of distinguishing between two computationally indistinguishable distributions, say P_1 and P_2. Now, to prove $|Pr[S_i] - Pr[S_{i+1}]|$ is negligibly close, an algorithm D (distinguishing) introduces between $Game_i$ and $Game_{i+1}$ so that if an auxiliary input is drawn from P_1, algorithm D outputs 1 with $Pr[S_i]$, achieving $Game_i$, and if an auxiliary input is drawn from P_2, algorithm D outputs 1 with $Pr[S_{i+1}]$, achieving $Game_{i+1}$. Then the postulate of indistinguishability becomes negligible $|Pr[S_i] - Pr[S_{i+1}]|$.

3.4.2 Transitions based on failure events

In this transition, $Game_i$ and $Game_{i+1}$ proceed identically unless a "*failure event F*" happens. The two games can be defined on the same underlying probability space, but the rules to compute random variables are different. A boolean variable *bad* is introduced in a game to set the failure event. To be noted, the flag *bad* begins with false, and once the flag is set to true, it can never be unset(false). The following lemma is the formal definition of identical two games until the failure event *bad* is set to true.

Lemma 1 (Difference Lemma [68, 15]). *Let G_1, G_2 be two games and A and B be events defined in G_1 and G_2 respectively, and F an event defined in both games. So, if $Pr[G_1 : A \wedge \neg F] \Longleftrightarrow Pr[G_2 : B \wedge \neg F]$, then $|Pr[G_1 : A] - Pr[G_2 : B]| \leq Pr[F]$.*

Proof. More generally, the probability of any two games G_1 and G_2 proceed identically until the failure event F happens, which is equivalent to $Pr[G_1 \wedge \neg F] = Pr[G_2 \wedge \neg F]$ and $Pr[G_1 \wedge F]$ and $Pr[G_2 \wedge F]$ are numbers between 0 and $Pr[F]$. We have

$$|Pr[G_1 : A] - Pr[G_2 : B]| = |Pr[G_1 \wedge F] + Pr[G_1 \wedge \neg F] - Pr[G_2 \wedge F] - Pr[G_2 \wedge \neg F]|$$

$$= |Pr[G_1 \wedge F] - Pr[G_2 \wedge F]|$$

$$\leq Pr[F]$$

□

One can use the Difference lemma (also called the Fundamental lemma) to analyse the games G_1 and G_2 with an adversary and bound the difference in the probability of a given event in both games. One can identify the failure event F and argues that games G_1 and G_2 proceed identically unless the event F happens. Next, one can bound the difference in the probability of a different event by the probability of the failure event in either game.

3.4.3 Bridging step

This transition works as a bridging step between two games. For instance, the probability of the successive $Game_{i+1}$ can be computed equivalently in specific quantities from the previous $Game_i$, i.e. $Pr[S_i] = Pr[S_{i+1}]$.

3.5 Proof-assistants for Cryptography

Halevi [7] stated that there are problems in the cryptographic constructions, and cryptographers write more security proofs than carefully verify the existing ones. Nonetheless, the proofs are long, complicated and deal with non-trivial algorithms, making it hard to verify the proofs. His vision of developing computer-aided tools or proof assistants to verify the security proofs and protocols has increased and shown substantial prior research. The idea of utilising the tools is to achieve high assurance and confidence in the security proofs during the formal verification. Some well-known cryptographic constructions such as [74, 75, 76, 77] are proven secure from the machine's support.

One can use the support of computer-aided tools to ensure that their proofs are proved correct. So the proofs in security reductions are proved abstractly and have many counter-intuitive components which become mostly overlooked with a critical eye. However, every component of the proofs can be explored inside a computer memory to understand the proofs confidently.

Proof assistant is a software that combines two main functionalities:

1. to offer support and help the user build long and complicated proofs, and
2. verify the proofs that they are complete and correct using the rules of logic.

On a high level, the proof assistant can be defined as a computer system allowing the user to do some maths on a computer with the aspect of proving and defining. To do so, the user manually sets up the theories, defines the functionalities by importing or utilising the libraries, and uses logical reasoning to prove them. In contrast to manual proving, the system of automated theorem provers allows the user to choose from the set of well-chosen decision procedures to prove automatically. A powerful and robust tool but has restricted expressiveness in setting generic mathematical theories.

Moreover, the symbolic and computational models can verify the cryptographic constructions. The verified protocol in the symbolic model is [78]. In the symbolic approach, for instance, [79] initiated the work of automated methods where automated procedures can be implemented in their tool to verify the generic hardness of assumptions. They analysed the hardness of the assumptions in the generic group models. They presented an automated tool which takes the assumptions and provides either an algebraic attack against the assumption or shows the generic hardness of the proof.

This thesis focuses on the frameworks developed in the computational model, and the prominent examples are CryptoVerif [80], CryptHOL [81], FCF [11], CertiCrypt [82] and EasyCrypt [8, 9].

The CryptoVerif is an automatic framework which mechanised in proving the properties of the security protocols such as secrecy and authentications [80] in the computational model. The framework is based on the technique of the game-based proof, where the game sequence is formalised in probabilistic polynomial-time (PPT) methodology. The framework can be used automatically (repertoire game transformation) or manually. There are some examples, i.e. SSH's Transport Layer Protocol [83], avionic protocols [84], Signal Protocol [85], and the Kerberos network authentication system [77] that used this framework to verify.

The CryptHOL framework is embedded in the Isabelle/HOL theorem prover [81], which adopts Isabelle's proof mechanisation to game-based proofs. [86] extends CryptHOL to support the formalisation of security proofs in the Constructive Cryptography framework [70], a generic theory supporting clean, composable security assertions.

The *Foundational Cryptography Framework* (FCF) is embedded in the Coq [14] theorem prover. This framework is generic and can be used to reason security definitions and assumptions. The proofs produce concrete bounds and asymptotic judgments [11].

CertiCrypt [82] framework is built upon the Coq proof assistant to formalise the stateful languages based on the code-based proofs. CertiCrypt verifies the proof of semantic security of OAEP and the proof of existential unforgeability of FDH signatures [82]. This framework can work in the areas of probability, the semantics of PL, complexity and algebra.

The EasyCrypt is an interactive proof assistant designed to focus on formalising cryptographic proofs in the computational model and utilises the approach of game-based proofs.

The EasyCrypt framework is the successor of CertiCrypt, which introduced a clear separation between theoretic reasoning and program verification. The first version of EasyCrypt was developed in 2011 to formalise the Cramer-Shoup [8] encryption and the Merkle-Damgard [76]. Although this version helped implement the security statements back then, it had scalability issues because it could not deal with very complicated cryptographic constructions. This issue led to the development of an extended version 1.0 of EasyCrypt in 2012. The extended version of EasyCrypt dealt with scalability issues by building a robust framework with the ability to provide a functional platform with underlying support of automated proofs, which can offer the user to machine-checked formalise the complex proofs with fundamental reasoning principles.

So, suppose one compares EasyCrypt with CertiCrypt. In that case, it is not wrong to say that verified proofs in EasyCrypt are better attainable to the working cryptographer as it uses SMT solvers and automated theorem provers to get automation. Whereas the

CertiCrypt verifies the proofs to the semantics of programs and realises with guarantees of Coq. Furthermore, EasyCrypt was used to translate the proofs from the AutoG&P tool [87], where the proofs were further explained with relational program logic. They implemented a formal logic in pairing-based cryptography, demonstrating that formal proofs of the complicated pairing-based constructions can be done.

The EasyCrypt has emblematic examples of proven cryptographic constructions and protocols such as Merkle-Damgard [76], TLS handshake protocol [88], RSA-PSS [89], mechanise proofs of UC security [90], privacy and verifiability for Belenios in e-voting [91], and the most recent ones are Saber's Public-key encryption scheme [92] and unforgeability proofs for signature schemes with tight reduction [44].

This thesis focuses on the EasyCrypt framework, which is explained next.

3.6 EasyCrypt proof assistant

The EasyCrypt framework has *module* systems for formalising the game-based proofs, and these modular reductions were also presented by Halevi [7]. The *module* consists of global variables and *procedures* where procedures can be written in a simple imperative language. One can produce proofs in EasyCrypt using tactics, where the hypotheses are shown in the goal section with a conclusion, and a goal can be reduced using the tactics. Proofs can be developed as a sequence of lemmas and can be checked interactively. Since the EasyCrypt embraces the game-based technique, the security statements and the hard mathematical problems are expressed as probabilistic programs called games or experiments. The reductionist proofs are decomposed into a sequence of games, where they become easier to check. The EasyCrypt has the following logic, which are explained next:

1. a Probabilistic Relational Hoare Logic (pRHL) to prove relations between pairs of games or experiments,

2. a Probabilistic Hoare logic (pHL) to prove probabilistic arguments on a single experiment,

3. an ordinary Hoare logic (HL).

Probabilistic Relational Hoare Logic (pRHL)

The pRHL [93] judgment has quadrupled the form

$$\vdash c_1 \sim c_2 : \Psi \Rightarrow \Phi.$$

The relation evaluates a probabilistic program c_1 to a probabilistic program c_2 w.r.t. a precondition Ψ and a postcondition Φ, and both conditions are defined as relations on deterministic states.

The judgment is valid if two memories, m_1 and m_2, satisfy the precondition Ψ, and the distributions $[\![c_1]\!]m_1$ and $[\![c_2]\!]m_2$ satisfy postcondition Φ. One can derive a probability claim from the valid judgments and write equalities or inequalities between two probabilities, such as

$$c_1 \sim c_2 : \Psi \Longrightarrow E\langle 1\rangle \rightarrow F\langle 2\rangle$$

where E and F are events. The probability for every initial memories m_1 and m_2 related to Ψ are

$$Pr[c_1, m_1 : E] \leq Pr[c_2, m_2 : F].$$

In addition, if the probability of the form $Pr[c_1, m_1 : A] = Pr[c_2, m_2 : A]$ for every initial memories related to Ψ with event A which depends on some $\{a_1, ..., a_n\}$ then the observational equivalence for the failure event F becomes:

$$Pr[c_1, m_1 : A] \leq Pr[c_2, m_2 : A] + Pr[c_2, m_2 : F].$$

In EasyCrypt, the pRHL has the following form:

$$\text{equiv } [M.p \sim N.q : \phi ==> \psi]$$

where:

- p is a procedure of module M, and q is a procedure of module N;
- ϕ is global variables of modules, the parameter of $M.p$ is in memory m_1, and parameters of $N.q$ is in memory m_2;
- ψ is global variables of modules and return type of $M.p$ is res_1, and return type of $N.q$ is res_2.

Probabilistic Hoare Logic (pHL)

Another logic of EasyCrypt is a probabilistic Hoare logic (pHL), which helps one to establish proofs on the probability after executing the procedure in its postconditions. The pHL reasons on the probability of events in games or experiments and the judgment of the form is:

$$[c : \zeta \Longrightarrow \varphi] \diamond p$$

where c is a program, ζ and φ are assertions, $\diamond$ is operators and finally, p is the probability.

In EasyCrypt, the pHL has the following form:

$$\text{phoare } [M.p : \phi ==> \psi] \diamond \text{ expression}$$

where:

- p is a procedure of module M;
- ϕ is global variables of modules and parameters of $M.p$;
- ψ is global variables of modules and return type of $M.p$ is res;
- $\diamond \in \{=, <, >\}$ relation;
- the expression is type of *real*.

The above judgment can be read for all initial memories m, satisfying ϕ. The domain comprises the module's global variables, parameters, and local variables of $M.p$. So, the probability of executing the $M.p$ in memory m result from stopping with memory whose condition to res and global variables of modules satisfy ψ.

Hoare logic (HL)

In EasyCrypt, the Hoare Logic (**HL**) has the following form:

$$\text{hoare } [M.p : \phi ==> \psi]$$

where:

- p is a procedure of module M;
- ϕ is global variables of modules and parameters of $M.p$;
- ψ is global variables of modules and return type of $M.p$ is res.

The following section explains that how to write a game in EasyCrypt.

3.6.1 A primer on EasyCrypt

One can write a game in EasyCrypt using a set of global variables with a collection of modules. A module consists of a procedure or procedures where some can be written as abstract or concrete, and the procedure can be written as the simple imperative language with loops and random assignments.

There are four steps to develop cryptographic proofs in EasyCrypt.

1. The first step is to define a formal context that includes *types*, *operators*, *constants* and declare them as axioms and derived lemmas.
2. The second step is to define the number of games that consist of several procedures, and the adversary can be declared as the abstract procedure parametrised by a set of oracles. One should instantiate the oracles as other procedures in the game.
3. The third step is to prove logical judgments that set equivalences between games. One can establish this using tactics and strategies.
4. The final step is to derive inequalities probabilities claims in games.

3.6.2 Input Language

In a proof assistant, the input language can be procedural (users tell the proof assistant what to do) or declarative (users tell the proof assistant where to go). EasyCrypt follows game-based proofs, which forces a special severance between program verification and information-theoretic reasoning. Games are written in an imperative language, called *pWHILE*, where games are formalised with procedural calls. The following set of commands are defined inductively by the clauses:

`C`	`::=`	`skip/nill`	nop
	|	`V` $\leftarrow \varepsilon$	deterministic assignment

|	`V` $\leftarrow_{\$} D\varepsilon$	probabilistic assignment
|	`if` ε `then C else C`	conditional
|	`while` ε `do C`	while loop
|	`V` $\leftarrow P(\varepsilon, ..., \varepsilon)$	procedure call
|	`C; C`	sequence

where V is a set of variable identifiers, ε is a set of expressions, $D\varepsilon$ is a set of distribution expressions, and a set P of procedure identifiers.

The language *pWHILE* is considered as an imperative typed probabilistic programming, and EasyCrypt has an automated mechanism to prove the claims about probabilities, combining rules to bound the probabilities of events in games from judgments in pRHL [93]. More information about the probabilistic Relational Hoare logic (pRHL) is described in section 3.6.

In EasyCrypt, the transition between games are reasoned into two steps: first, it implements a probabilistic Hoare logic (pHL) to bound the probability of postcondition and embeds both pHL and probabilistic Relational Hoare logic (pRHL) into an ambient logic which can be used to perform hybrid arguments that involve equivalences on parameterised programs. Second, it implements a theory mechanism and a module system supporting compositional proofs through quantification over programs, types and values. This step employs information-theoretic reasoning to claim the probability of events from the pRHL. Each step is highly effective and builds upon purpose-specific tools. The pRHL judgment uses satisfactory verification conditions for its validity, expressed in the language of first-order logic. The outstanding feature of EasyCrypt is that the verification conditions are shown without introducing the probability, which can be automatically discharged by using the SMT solvers and theorem provers.

3.6.3 Axioms and Lemmas

In EasyCrypt, proofs are developed as a sequence of lemmas, and theories can be used to build up the types, operators, modules and lemmas. After writing the lemmas, proofs are solved using tactics and general reasoning principles that transform the current goal into more goals or zero with acceptable conditions for the lemmas to be held. The goals with simple ambient logic can easily be proved using the Satisfiability Modulo Theories (SMT) solvers (**smt()**).

One can write an axiom and lemma as in

---

axiom Example : forall (x y : int), x =y => y=x.

lemma Example : forall (x y : int), x =y => y=x.

The difference between the axiom and lemma is that EasyCrypt trusts the axiom while one must prove the lemma. The steps for proving the lemma have the form

proof.

$tactic_1 \cdots\cdots\cdots tactic_n$.

qed.

The first step is optional and can be omitted. Inside the proof, tactics are used. The quod erat demonstrandum (**qed**) preserves the lemma, which can be reusable and is only allowed when the proof is fully proved. One cannot use the same name for two lemmas; otherwise, there will be conflict upon running the **qed**.

Furthermore, axioms can be used to express abstract operators and types and express hypotheses over declared constants. The lemmas are written to verify goals in the security proofs.

3.6.4 Game declarations

The security reductions are proved in EasyCrypt using the code-based game-playing technique. The games are modelled as modules, and the module comprises of typed global variables, procedures, and abstract adversary assertions. The EasyCrypt expression language is established on the polymorphic typed λ calculus, and they are guaranteed to terminate. The declarations, statements of assignments, and procedures end with a semicolon. However, one cannot use a semicolon at the end of the conditional statements. One can declare multiple local variables together, as in:

---

var $x, y, z : int$;

var $x, y, z : int \leftarrow 5$;

var x, y;

The following listing 3.1 represents the definition of a simple module, CDH, which illustrates the module language.

Listing 3.1: A module in EasyCrypt

```
module CDH (A : CDH_Adv) = {
```

```
  proc run() = {
    var a, b, r;

    a <$ dZp;
    b <$ dZp;
    r <@ A.solve(g^a, g^b);
    return r = g^(a * b);
  }
}.
```

The module CDH does not have any global variables, which are generally declared after opening the curly braces. The procedure *main* does not take any parameters and returns some computations at the end ($r = g^{a \cdot b}$). The local variables a, b, and r are declared inside the procedure. The *main* uses a random assignment, where a and b are randomly sampled. After sampling, a *procedure call assignment* to call the procedure *solve* with two arguments (g^a, g^b) is used and assign *solve*'s return value to r.

In EasyCrypt, the module types prescribe the types of sets of procedures; for instance, the module type *Adv-CDH*:

```
module type Adv-CDH = {

    proc solve (ga gb: Group) : Group

}.
```

The *Adv-CDH* has one procedure (*solve*) and defines solving the CDH. The procedure takes two parameters of type *Group*. So in the above figure, the game is parametrised by an adversary with access to the solve procedure.

Chapter 4

Machine-Checked verification of EDL and CM Signature Schemes

This chapter includes the original research, co-authored with Dr François Dupressoir, and it was submitted at the conference [44]. This chapter reports on the first machine-checked proofs for EDL and CM signature schemes with tight reductions to the CDH problem using EasyCrypt proof assistant.

4.1 Introduction

The EDL signature scheme—based on the Discrete Logarithm (DL) problem—was independently proposed without proof by Chaum and Pedersen [18] and Jakobsson and Schnorr [19]. Goh and Jarecki [16] propose a proof in the random oracle model that EDL is existentially unforgeable under chosen-message attacks when constructed over a group in which the Computational Diffie-Hellman (CDH) problem is hard. Crucially, Goh and Jarecki's proposed security proof does not make use of Pointcheval and Stern's *Forking Lemma* [1]: although EDL signatures contain a Schnorr proof, it is used as a zero-knowledge proof of discrete logarithm equality, rather than as a proof of knowledge. This allows a much tighter reduction than most other DL-based signature schemes, theoretically supporting shorter signatures.

Chevallier-Mames [17] presents an improvement on EDL which reduces the size of signatures and allows cost-free use of *coupons*. The scheme (denoted CM in the following) relies on similar techniques to avoid relying on the forking lemma, but interestingly still relies on the special soundness property of Schnorr proofs.

4.1.1 Our Contributions

Our main contribution is the first machine-checked proof for a signature scheme with a tight reduction to CDH. We formalize proofs for both the EDL (Section 4.3) and CM (Section 4.4) signatures schemes, and make them publicly available.[1] We reformulate Goh and Jarecki [16] and Chevallier-Mames's [17] pen and paper proofs, and mechanise them in EasyCrypt. Like the original results by Goh and Jarecki [16] and Chevallier-Mames [17], our theorems are concrete—rather than asymptotic—security statements. However, the original proofs were direct reductions; ours follows the methodology based on sequences of games advocated by Shoup [15].

This, and the formalisation effort, allow us to identify proof principles that we believe are more general, and could be applied more broadly. In developing the proof of security for EDL, we develop a *shim*, used in all intermediate steps of the proof, and whose main interest is in reducing the amount of boilerplate proof. We reuse *the same shim* with very minor tweaks for the CM scheme, whose proof differs only in probability bounding and reduction steps. Section 4.5 closes with discussions of further potential generalizations as well as or related and future work.

Due to very minor mistakes and omissions in the original proof by Goh and Jarecki [16], we cannot prove that the bound given originally holds; we prove a very close bound instead, similar to that given for EDL by Chevallier-Mames [17], with minor amendments to address formal details (discussed where relevant). Our formal theorem for the CM scheme is in line with that given originally.

4.1.2 Related Work

We now discuss closely related work, first considering digital signature schemes, and then the state of machine-checked proofs for DL-based cryptographic constructions.

Proofs for digital signature schemes Tight proofs for digital signatures exist for PSS [94]. EDL, CM and their variants by Katz and Wang [63] and Goh et al. [6]

[1] `https://gitlab.org/ec-zksigs/edl.git`

are—to our knowledge—the only DL-based signature schemes equipped with tight proofs. Coron [94], and Bader, Jager, Li and Schäge [95] show the impossibility of obtaining tight reductions in some settings (for FDH, and for signing in multi-user settings). Proofs following our game sequence are not necessarily tight: the final reduction step is where looseness could be introduced, rather than the sequence itself. As such, we believe the techniques presented here are not limited to those schemes for which tight bounds can be established.

Katz and Wang [63] discuss a proof technique—based on claw-free permutations—that applies to PSS as well as to DL-based signing schemes to obtain tight security proofs. Machine-checking their proofs could yield interesting insights in future. Barthe et al [96] formalise a security proof for PSS. Although the proof involves the consideration of faults, it also establishes a tight machine-checked security bound for the security of PSS in the absence of faults, following proofs by Coron and Mandal [94, 97]. The proof's structure is in fact similar to that of the proofs discussed here; considering it as an instance of our shim may help generalize the shim further for broader applicability.

El Kaafarani, Katsumata and Pintore [98] propose a tightly-secure post-quantum signing scheme based on CSIDH [99] and CSI-FiSh [100]. Their proof relies on a generic result on the security of Fiat-Shamir in the Quantum-Random Oracle Model (QROM) due to Kiltz, Lyubashevsky and Schaffner [101]. The proof shape we formalise here does not consider the QROM, and is unlikely to apply in the presence of quantum-capable adversaries. Extensions in this direction would be natural as future work. We note, however, that the formalisation of such proof is still on the edge of feasibility [102].

Machine-checked proofs for DL-based cryptography Although this paper presents the first machine-checked proof for a DL-based signature scheme, other machine-checked proofs exist for DL-based cryptography more generally.

EasyCrypt was used to formalize the security of Cramer-Shoup [8], one-round key exchange protocols [103] and Amazon Web Services' Key Management Service [104].

Only the Cramer-Shoup proofs involve reasoning about arithmetic in cyclic groups similar to that done here, with the others focusing on DL-based key exchange and key encapsulation. (Complexity in these other cases often stems from considering multiple interactive instances with corruption.)

Proofs in other tools, such as F* [105] or CryptoVerif [106] (for Wireguard [107], TLS 1.3 [108, 109], Signal [110], OEKE [111] and HPKE [112]), also focus on protocol reasoning, with little to no reasoning about the group structure involved, and challenges arising from the complexity of the properties being proved rather than inherent complexity in the mathematical arguments involved.

4.2 Mathematical Preliminaries

To better support—and in fact add value to—the formalization, we carry out our formal proof on abstract mathematical objects, without specifying their implementation. As shown by Almeida, Barbosa, Barthe and Dupressoir [113, 114], this does not preclude further refinements, even all the way down to considering implementation adversaries. More importantly, the proof formalized then applies to all valid instantiations of the abstract objects, provided the refinement ensures that the expected interface is respected.[1]

We consider constructions that rely on a cyclic group $\mathbb{G}$ of prime order q, and on a specific generator g of $\mathbb{G}$ agreed upon ahead of time. $\mathbb{G}$, q and g are assumed to be public, and known—in particular—to the adversary. We use multiplicative notation for the group $\mathbb{G}$, denoting with $1_{\mathbb{G}}$ (or simply 1 when unambiguous) its identity element, and with $\times$ its operation. We use exponents to denote iterations of $\times$ but take exponents directly in the field $\mathbb{F}_q$ such that field addition ($+$) and multiplication ($\cdot$) in the exponent are *implicitly* carried out modulo q. We use $\cdot^{-1}$ for field inversion. Multiplication symbols are often omitted, as is standard.

4.3 EDL Signatures and their Security

The EDL signature scheme is defined over a set $\mathcal{M}$ of messages as shown in Figure 4.1, where, in addition to the cyclic group $\mathbb{G}$, we also consider a set $\mathcal{N}$ of *nonces*, equipped with some distribution $d_{\mathcal{N}}$.

Key generation is as standard for DH-style schemes. Signing is done by producing a non-interactive zero-knowledge (NIZK) proof of discrete logarithm equality with bases a message-dependent hash and the generator, and discrete logarithm the secret key. Verification recomputes the message-dependent hash and verifies the corresponding NIZK proof.

We formally prove a *tight* reduction to breaking CDH in $\mathbb{G}$ from breaking EUF-CMA security of EDL, where by tight we mean, like Goh and Jarecki [16] that there are no multiplicative factors involved in relating the time complexity and advantage of an adversary against the scheme and that of the corresponding reduction. More precisely, we are looking for reductions that—when applied to an adversary that breaks the signature scheme with probability ϵ in time t—break the underlying hard problem with

[1] In particular, this would normally require the refinement to ensure or check that inputs are indeed valid encodings of elements in the expected algebraic structure, or to prove that it does not matter for security.

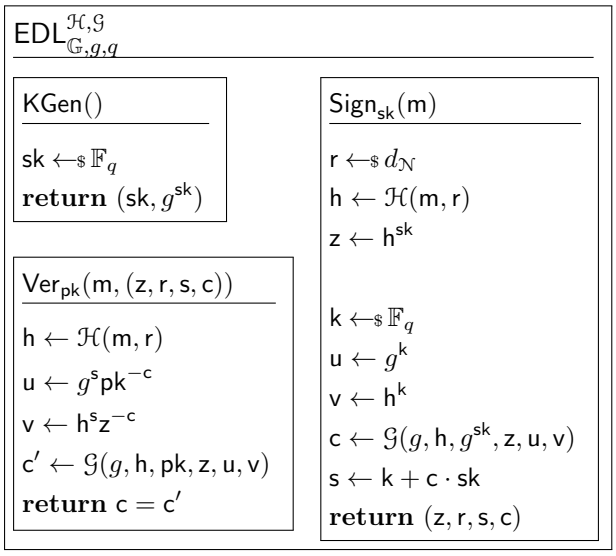

Figure 4.1: The EDL signature scheme, parameterized by two random oracles $\mathcal{H} : \mathcal{M} \times \mathcal{N} \to \mathbb{G}$ and $\mathcal{G} : \mathbb{G}^6 \to \mathbb{F}_q$.

probability $\epsilon' \approx \epsilon$ in time $t' \approx t$. Our proof is concrete and constructive: the concrete reduction used in proving Theorem 2 is displayed in Figure 4.2.

Theorem 2 (Security of EDL)**.** *If CDH is (t, ϵ)-hard in $\mathbb{G}$ with generator g, then EDL is $(t', q_{\mathcal{H}}, q_{\mathcal{G}}, q_{\mathcal{S}}, \epsilon')$-EUF-CMA-secure for all non-negative $q_{\mathcal{H}}$, $q_{\mathcal{G}}$, $q_{\mathcal{S}}$, and with*

$$\begin{aligned} t &\lesssim t' + (q_{\mathcal{H}} + 6 \cdot q_{\mathcal{S}} + 1) \cdot t_{exp} \\ \epsilon' &\leq \epsilon + q_{\mathcal{S}} \cdot \left(\frac{q_{\mathcal{H}} + q_{\mathcal{S}}}{|\mathcal{N}|} + \frac{q_{\mathcal{G}} + q_{\mathcal{S}}}{q^2} \right) \\ &\quad + \frac{q_{\mathcal{G}} + 1}{q} \end{aligned}$$

where t_{exp} is the cost of an exponentiation in $\mathbb{G}$.

Proof. The bound on the reduction's time complexity t is not formally verified, and we prove it here: the reduction (as shown in Figure 4.2) first runs the EUF-CMA forger and simulates its oracles. When simulating each of the forger's $q_{\mathcal{H}}$ queries to $\mathcal{H}$, the reduction computes one exponentiation in $\mathbb{G}$. When simulating each of the forger's

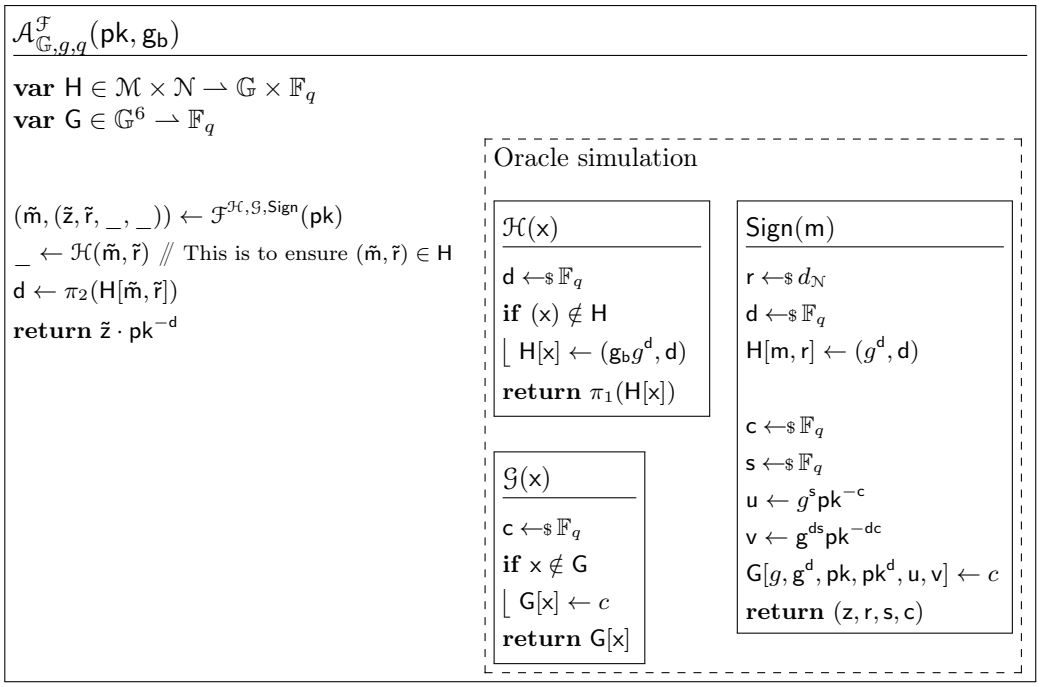

Figure 4.2: The reduction $\mathcal{A}$ from CDH. $\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}$ uses an EUF-CMA forger $\mathcal{F}$ as a black-box, and internally simulates $\mathcal{H}$ and $\mathcal{G}$. H keeps track of both the response h and i. its discrete logarithm in base g (for queries made by the signing oracle), or ii. the unique value $\mathsf{d} \in \mathbb{F}_q$ such that $\mathsf{h} = \mathsf{g_b} g^{\mathsf{d}}$ (for queries made by the forger). π_1 and π_2 are the first and second projections on pairs.

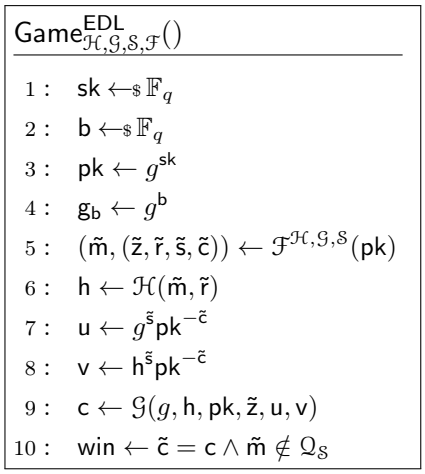

Figure 4.3: The EDL proof shim.

q_S queries to the signing oracle, the reduction computes two exponentiations and two double-exponentiations. Finally, the reduction computes one inverse in $\mathbb{F}_q$ (whose cost is omitted, as it is dominated by that of exponentiations) and one exponentiation to retrieve its answer. We overapproximate the cost of double-exponentiation as the cost of two exponentiations to get the bound shown above.

The rest of this Section is dedicated to discussing the proof for the bound on ϵ'. Figures 4.3-4.4 show the intermediate games that serve in this proof. □

Goh and Jarecki's original proof [16]—which uses $\mathsf{g_b}^{\mathsf{d}}$ instead of $\mathsf{g_b}g^{\mathsf{d}}$ when simulating $\mathcal{H}$—omits certain low probability cases in which their given simulation fails:

1. when b is 0;
2. when the value of d used in simulating $\mathcal{H}$ for forgery verification is 0; and
3. when the same random nonce is used in two separate signatures of the same message.

Accounting for these cases yields a valid (and indeed machine-checked) security proof, with a slightly worse bound than that given here. We instead formalize and check a different proof, which yields a more precise bound.

4.3.1 Intuition

We first note that a valid forgery $(\tilde{\mathsf{m}}, (\tilde{\mathsf{z}}, \tilde{\mathsf{r}}, \tilde{\mathsf{s}}, \tilde{\mathsf{c}}))$ cannot be such that the signature scheme queried $\mathcal{H}$ on $(\tilde{\mathsf{m}}, \tilde{\mathsf{r}})$ (if that were the case, the forgery would not be fresh), so either the forger has made that query herself, or we can make the query on her behalf after she returns. Note also that, in an honestly computed signature $(\mathsf{z}, \mathsf{r}, \mathsf{s}, \mathsf{c})$ for a message m, we have $\mathsf{h} = \mathsf{z}^{\mathsf{sk}}$, where $\mathsf{h} \leftarrow \mathcal{H}(\mathsf{m}, \mathsf{r})$. The key insight in the reduction is to embed the CDH challenge into the EUF-CMA game, using g^{a} as public key (so that $\mathsf{sk} = \mathsf{a}$), and embedding g^{b} into the answers given by $\mathcal{H}$ to the forger (so that $\mathsf{h} = g^{\mathsf{b}}g^{\mathsf{d}}$ for some d known to the simulator). In this setting, a valid forgery that is such that $\tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}}$ can be used to compute g^{ab} as $\tilde{\mathsf{z}} \cdot \mathsf{pk}^{-\mathsf{d}}$.

$$\tilde{\mathsf{z}} \cdot \mathsf{pk}^{-\mathsf{d}} = \mathsf{h}^{\mathsf{a}} \cdot (\mathsf{g}^{\mathsf{a}})^{-\mathsf{d}} = (g^{\mathsf{b}}g^{\mathsf{d}})^{\mathsf{a}} \cdot \mathsf{g}^{-\mathsf{ad}} = g^{\mathsf{ab}}$$

With this key insight, a proof can be obtained by further proving that 1. the reduction—without access to the private key—can simulate the signing oracle an EUF-CMA forger against EDL expects to have access to, and 2. the probability of verification succeeding for a forgery $(\tilde{m}, (\tilde{z}, \tilde{r}, \tilde{s}, \tilde{c}))$ such that $\tilde{z} \neq h^a$ (with $h \leftarrow \mathcal{H}(\tilde{m}, \tilde{r})$) is low.

We now detail the sequence of games and local claims which formalise and leverage this intuition. The sequence of games itself (Lemmas 3 to 5) shows that the simulated oracles from Figure 4.2 are indistinguishable from the actual EUF-CMA oracles for EDL. We then show (Lemma 7) that any run of the forger that finds a forgery in the final game lets the reduction solve the CDH challenge, except if the forgery is such that $\tilde{z} \neq h^a$, and bound the probability that this occurs with a successful forgery (Lemma 6).

Proofs for the local claims give an intuition of the machine-checked reasoning on programs involved, but do not delve into detailed discussions of the tactics used. We keep the description here as close as possible to the EasyCrypt proof, but omit some formal details in probability-bounding steps. Indeed, in the formal proof, we must first transform the game to explicitly count oracle queries. Our final theorem, like Theorem 2 does quantify over adversaries that make a bounded number of queries.

4.3.2 Formalisation

We formally define 3 intermediate games that bridge the gap between the EUF-CMA experiment and the CDH experiment. We write these intermediate games as instances of a common *shim* $\mathsf{Game}^{\mathsf{EDL}}_{\cdot,\cdot,\cdot,\cdot}()$ (shown in Figure 4.3), which is parameterized by oracles presenting the same interface as the random and signing oracles, and by a forger. Any two successive games in our formalization differ only in the oracles provided to the shim. The oracles we use in our proof, along with the definition of our intermediate games as instances of the shim, are shown in Figure 4.4.

Locally, using a shim allows us to focus reasoning on the parts of the experiment that do change between the successive games, and reduces the amount of boilerplate proof code we need to write. In EasyCrypt, we reuse a single proof base over the shim in all proof steps, which reduces the proof obligation to obligations on the oracles themselves, the shim essentially serving as a distinguisher between the various sets of oracles we consider in our games.

The formal proof proceeds in 4 steps:

1. we *refactor* the EUF-CMA security of EDL as an instance of the shim;
2. we *embed* the CDH challenge into the responses given to $\mathcal{H}$ queries;

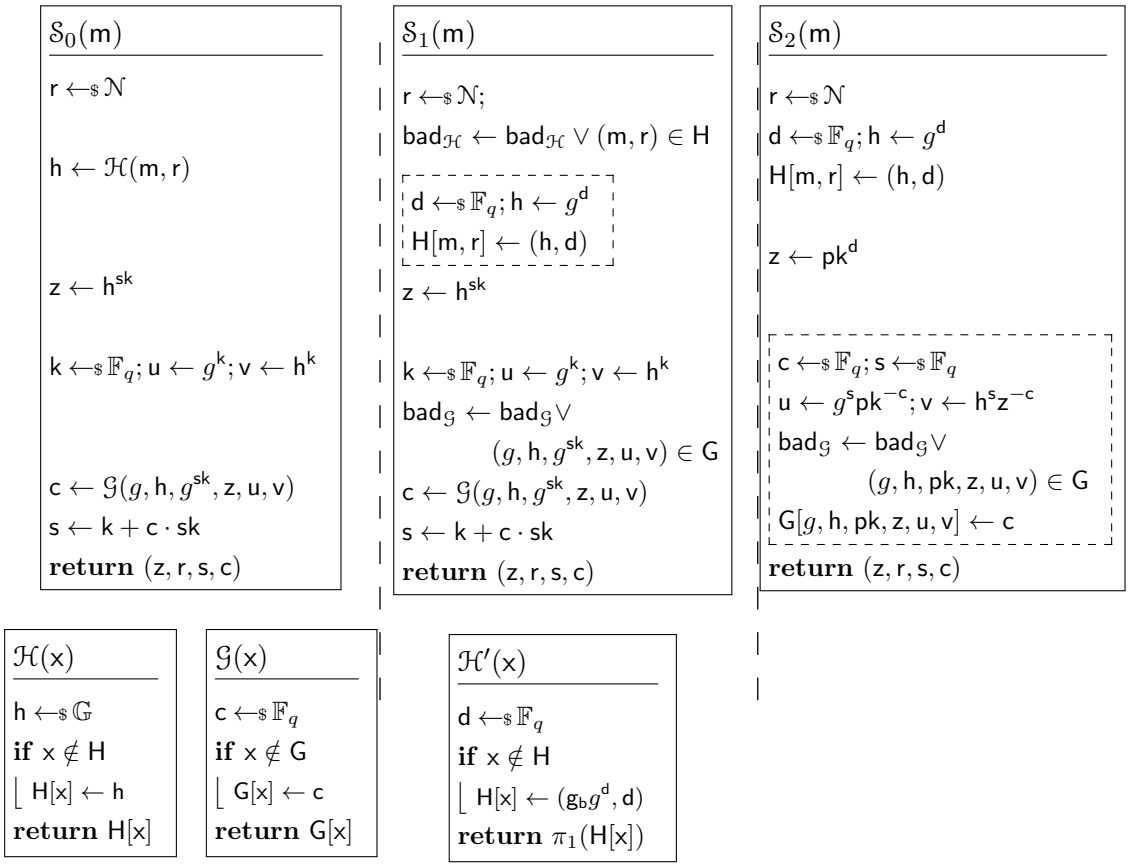

Figure 4.4: Oracle simulations for use in intermediate steps in the security proof for EDL signatures. Lemmas 3 and 4 use $\mathsf{Game}_0^{\mathsf{EDL}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{EDL}}_{\mathcal{H},\mathcal{G},\mathcal{S}_0,\mathcal{F}}()$. Lemmas 4 and 5 use $\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{EDL}}_{\mathcal{H}',\mathcal{G},\mathcal{S}_1,\mathcal{F}}()$. Lemmas 5 and 7 use $\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{EDL}}_{\mathcal{H}',\mathcal{G},\mathcal{S}_2,\mathcal{F}}()$.

3. we *simulate* the proof of discrete logarithm equality without using the witness (here the secret key); and

4. we show that a forgery either exploits the (negligible) unsoundness of the proof of discrete logarithm equality, or answers the given CDH challenge.

Refactoring

The first step refactors the EUF-CMA game to make use of the shim, as $\mathsf{Game}_0^{\mathsf{EDL}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{EDL}}_{\mathcal{H},\mathcal{G},\mathcal{S}_0,\mathcal{F}}()$ where $\mathcal{H}$, $\mathcal{G}$, and $\mathcal{S}_0$ are shown in Figure 4.4.

Lemma 3 (EDL Refactoring)**.** *For any forger $\mathcal{F}$, we have*

$$\Pr\left[\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{H},\mathcal{G},\mathit{EDL},\mathcal{F}}() : \mathsf{b} \wedge \tilde{\mathsf{m}} \notin \mathcal{Q}_{\mathcal{S}}\right] = \Pr\left[\mathsf{Game}_0^{\mathit{EDL}}(\mathcal{F}) : \mathsf{win}\right]$$

Proof. This is a simple program equivalence: $\mathsf{Game}_0^{\mathsf{EDL}}(\mathcal{F})$ is the EUF-CMA experiment for EDL with KGen and Ver inlined, and with the addition of ineffective operations (sampling of b and computation of g_{b}, which are not used in the rest of the game. □

Embedding

In our second step, we modify the way in which queries to $\mathcal{H}$ are handled to embed the CDH challenge in the log of forger queries. Consider $\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{EDL}}_{\mathcal{H}',\mathcal{G},\mathcal{S}_1,\mathcal{F}}()$, with $\mathcal{H}'$, $\mathcal{G}$, $\mathcal{S}_1$ as defined in Figure 4.4.

To the forger, we expose $\mathcal{H}'$—which computes its output as $\mathsf{g}_{\mathsf{b}} g^{\mathsf{d}}$ for some uniformly sampled d that is kept alongside the oracle's response.

The signing oracle $\mathcal{S}_1$ is modified so that it always samples a fresh value of h (although keeping in its log its discrete logarithm d), regardless of whether its (m, r) had already appeared as a query to $\mathcal{H}'$ or in a signing query. This change is visible to the adversary—who can distinguish between $\mathsf{Game}_0^{\mathsf{EDL}}(\mathcal{F})$ and $\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F})$ when the pair (m, r) being used in a signing query already appears in the map H. We capture this event as $\mathsf{bad}_{\mathcal{H}}$.

In addition, we keep track of an event $\mathsf{bad}_{\mathcal{G}}$, which is triggered when the query to $\mathcal{G}$ made by the signing oracle is not fresh. This event is not observable by the adversary, but is easier to bound in $\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F})$ than in the next game, where it does become observable.

Lemma 4 (EDL Embedding)**.** *For any forger $\mathcal{F}$, we have*

$$\Pr\left[\mathsf{Game}_0^{\mathit{EDL}}(\mathcal{F}) : \mathsf{win}\right] \leq \Pr\left[\mathsf{Game}_1^{\mathit{EDL}}(\mathcal{F}) : \mathsf{win}\right] + \Pr\left[\mathsf{Game}_1^{\mathit{EDL}}(\mathcal{F}) : \mathsf{bad}_{\mathcal{H}}\right]$$

In addition, for any forger $\mathcal{F}$ that makes at most $q_{\mathcal{H}}$ queries to her $\mathcal{H}$ oracle, and at

most $q_{\mathcal{S}}$ queries to her signing oracle, we have

$$\Pr\left[\mathsf{Game}_1^{EDL}(\mathcal{F}) : \mathsf{bad}_{\mathcal{H}}\right] \leq q_{\mathcal{S}} \cdot \frac{q_{\mathcal{H}} + q_{\mathcal{S}}}{q}$$

Proof. Thanks to the use of a common shim, we only need to consider differences in behaviour arising from the oracles.

First we observe that logging d in H has no effect on the adversary's view of the system: computing $\mathsf{g}_{\mathsf{b}} g^{\mathsf{d}}$ and g^{d} both yield the uniform distribution over $\mathbb{G}$ when d is uniform in $\mathbb{F}_q$, and the value of d is not used (other than in computing the response of the random oracle) while the forger is running. Focusing on the signing oracle, it is clear that $\mathcal{S}_1$ is equivalent to $\mathcal{S}_0$ as long as $\mathsf{bad}_{\mathcal{H}}$ is false: indeed, in that case, $\mathcal{S}_1$ is exactly $\mathcal{S}_0$ with $\mathcal{H}$ inlined, the conditional reduced to its true branch, and h sampled in a way that reveals its discrete logarithm (this is marked as the dashed box in Figure 4.4). This proves the first part of the lemma.

Second, we bound the probability of $\mathsf{bad}_{\mathcal{H}}$ occurring. During any one execution of $\mathcal{S}_1$, $\mathsf{bad}_{\mathcal{H}}$ becomes true if a fresh value r sampled in $d_{\mathcal{N}}$ already appears in H. H increases in size by at most 1 for each query to $\mathcal{H}'$ and to $\mathcal{S}_1$. So each query to $\mathcal{S}_1$ sets $\mathsf{bad}_{\mathcal{H}}$ with probability at most $\frac{q_{\mathcal{H}}+q_{\mathcal{S}}}{q}$. A forger that makes $q_{\mathcal{S}}$ query to $\mathcal{S}_1$ therefore triggers $\mathsf{bad}_{\mathcal{H}}$ with probability at most $q_{\mathcal{S}} \cdot \frac{q_{\mathcal{H}}+q_{\mathcal{S}}}{q}$. □

Simulation

In our third step, we simulate the proof of discrete logarithm equality. We rely on its zero-knowledge property to simulate the signing oracle without using the secret key.

$\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{EDL}}_{\mathcal{H}',\mathcal{G},\mathcal{S}_2,\mathcal{F}}()$ carries out this simulation by programming the random oracle $\mathcal{G}$ on inputs queried by the signing oracle—regardless of their freshness.

$\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F})$ can therefore be distinguished from $\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F})$ by the forger exactly when the signing oracle programs $\mathcal{G}$ in a point the forger had previously queried. This event—which we captured as $\mathsf{bad}_{\mathcal{G}}$ already in $\mathcal{S}_1$—occurs with low probability since some of its parameters are freshly sampled in high entropy distributions. This probability, however, is easier to bound in $\mathcal{S}_1$, and we rely on the argument outlined in Section 2.4 to prove the bound.

Lemma 5 (EDL Simulation)**.** *For any forger $\mathcal{F}$, we have*

$$\Pr\left[\mathsf{Game}_1^{EDL}(\mathcal{F}) : \mathsf{win}\right] \leq \Pr\left[\mathsf{Game}_2^{EDL}(\mathcal{F}) : \mathsf{win}\right] + \Pr\left[\mathsf{Game}_1^{EDL}(\mathcal{F}) : \mathsf{bad}_{\mathcal{G}}\right]$$

In addition, for any forger $\mathcal{F}$ that makes at most $q_{\mathcal{G}}$ queries to her $\mathcal{G}$ oracle, and at most $q_{\mathcal{S}}$ queries to her signing oracle, we have

$$\Pr\left[\mathsf{Game}_1^{EDL}(\mathcal{F}) : \mathsf{bad}_{\mathcal{G}}\right] \leq q_{\mathcal{S}} \cdot \frac{q_{\mathcal{G}} + q_{\mathcal{S}}}{q^2}$$

Proof. Note the fact that we bound the probability of $\mathsf{bad}_{\mathcal{G}}$ occurring in the embedding game, rather that the simulation game. This requires a bit of additional effort. We prove the following two relations, combining them to prove the first claim.

$$\Pr\left[\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{bad}_{\mathcal{G}}\right] = \Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{bad}_{\mathcal{G}}\right]$$

$$\Pr\left[\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{win}\right] \leq \Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{win}\right] + \Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{bad}_{\mathcal{G}}\right]$$

We do so—both in EasyCrypt and in the argument below—with a single equivalence proof, in which we establish that the $\mathsf{bad}_{\mathcal{G}}$ events occur with the same probability in both games, *and* that a forger $\mathcal{F}$ that wins against $\mathcal{S}_1$ also wins against $\mathcal{S}_2$ unless its run against $\mathcal{S}_2$ triggers $\mathsf{bad}_{\mathcal{G}}$.

The oracles $\mathcal{H}'$ and $\mathcal{G}$ are as in the previous game, with the only difference lying in the signing oracle $\mathcal{S}_2$. We must prove two facts relating executions of $\mathcal{S}_1$ and $\mathcal{S}_2$ starting from the same memory where $\mathsf{bad}_{\mathcal{G}}$ has not yet been set. First, we must prove—unconditionally—that they produce the same distribution over $\mathsf{bad}_{\mathcal{G}}$. Second, we must prove that executions that leave $\mathsf{bad}_{\mathcal{G}}$ unset always yield the same distribution over the output (z, r, s and c) and state (maps H and G) of the oracles.

Variables d and z follow the same distribution in both games, since d is sampled in the same distribution, and $\mathsf{z} = g^{\mathsf{d}\cdot\mathsf{sk}}$.

In order to reason about further equivalences, it is necessary to consider the code of $\mathcal{G}$. Consider a version of $\mathcal{S}_1$ with $\mathcal{G}$ inlined. Now it is clear that variable c can be sampled at the same time as k without change in the semantics. Now, the distribution in $\mathcal{S}_1$ of $(\mathsf{k}, \mathsf{c}, \mathsf{k} + \mathsf{c}\cdot\mathsf{sk})$ is the same as that of $(\mathsf{s} - \mathsf{c}\cdot\mathsf{sk}, \mathsf{c}, \mathsf{s})$ in $\mathcal{S}_2$. The distributions of u, v and $\mathsf{bad}_{\mathcal{G}}$ are therefore also equal. From here on, we only need consider the case where $\mathsf{bad}_{\mathcal{G}}$ does not occur. In this case, the conditional in $\mathcal{G}$ follows the **then** branch, yield the same distribution over output and state. This concludes the proof of the first claim.

Now, as discussed, we bound the probability of $\mathsf{bad}_{\mathcal{G}}$ occurring in $\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F})$ to conclude the proof. Variable $\mathsf{bad}_{\mathcal{G}}$ is set by a signing query if the tuple it uses as input to $\mathcal{G}$ coincides with one of the inputs on which $\mathcal{G}$ has already been queried (one that is set in G). Signing oracle $\mathcal{S}_1$ makes only queries of the form $\left(g, g^{\mathsf{d}}, g^{\mathsf{sk}}, \mathsf{z}, g^{\mathsf{k}}, \mathsf{v}\right)$ where k and d are sampled freshly and uniformly at random in $\mathbb{F}_q$. Therefore, the probability that such a query is one of the at most $q_{\mathcal{G}} + q_{\mathcal{S}}$ already set in G is upper-bounded by $\frac{q_{\mathcal{G}}+q_{\mathcal{S}}}{q^2}$. The forger makes at most $q_{\mathcal{S}}$ queries to its signing oracle, so $\mathsf{bad}_{\mathcal{G}}$ occurs with probability at most $q_{\mathcal{S}} \cdot \frac{q_{\mathcal{G}}+q_{\mathcal{S}}}{q^2}$. □

Reduction

In the final step of the proof, we show that if a forger $\mathcal{F}$ wins $\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F})$, the CDH adversary $\mathcal{A}^{\mathcal{F}}$ from Figure 4.2 succeeds in solving its given CDH instance, except with low probability. More specifically, a successful forgery either can be used to solve the given CDH instance, or relies on an unsound proof of discrete logarithm equality.

We start by stating and proving a lemma bounding the probability of the forger relying on unsoundness.

Lemma 6 ($\tilde{\mathsf{z}} \neq \mathsf{h}^{\mathsf{sk}}$)**.** *For any forger $\mathcal{F}$ that makes at most $q_{\mathcal{G}}$ queries to her $\mathcal{G}$ oracle, we have*

$$\Pr\left[\mathsf{Game}_2^{EDL}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} \neq \mathsf{h}^{\mathsf{sk}}\right] \leq \frac{q_{\mathcal{G}} + 1}{q}$$

Proof. If the forger produces a valid forgery such that $\mathsf{z} \neq \mathsf{h}^{\mathsf{sk}}$ then there exists in map G—at the end of the game's run—an input-output pair $((g, \mathsf{h}, \mathsf{y}, \mathsf{z}, \mathsf{u}, \mathsf{v}), \mathsf{c})$ such that $\mathsf{z} \neq \mathsf{h}^{\log \mathsf{y}}$ and $(\mathsf{u} \times \mathsf{y}^{\mathsf{c}})^{\log \mathsf{h}} = \mathsf{v} \times \mathsf{z}^{\mathsf{c}}$. Indeed, consider the input-output pair $\left(\left(g, \mathsf{h}, g^{\mathsf{sk}}, \tilde{\mathsf{z}}, g^{\tilde{\mathsf{s}}} g^{-\mathsf{sk} \cdot \tilde{\mathsf{c}}}, \mathsf{h}^{\tilde{\mathsf{s}}} g^{-\mathsf{sk} \cdot \tilde{\mathsf{c}}}\right), \mathsf{c}\right)$ involved in the $\mathcal{G}$ query made by the shim during verification of the forgery. Since the forgery is valid, it must be that $\tilde{\mathsf{c}} = \mathsf{c}$ and the conditions hold.

We now bound the probability that any specific query to $\mathcal{G}$ adds such an input-output pair to G. Since all $\mathcal{G}$ queries made by the signing oracle have $\mathsf{z} = \mathsf{h}^{\mathsf{sk}}$, it must be that the $\mathcal{G}$ query we consider was either made by the forger or shim—there are at most $q_{\mathcal{G}} + 1$ such queries. Each of them samples c at random in $\mathbb{F}_q$, and—all inputs being set—there is only one value in $\mathbb{F}_q$ that meets the constraints—namely $\mathsf{c} = {}^{\log(\mathsf{v}/\mathsf{u}^{\log \mathsf{h}})}/_{\log(\mathsf{h}^{\log \mathsf{y}}/\mathsf{z})}$. Oracle $\mathcal{G}$ samples this one value with probability $1/q$. □

Lemma 7 (EDL Reduction)**.** *For any forger $\mathcal{F}$ that makes at most $q_{\mathcal{G}}$ queries to her $\mathcal{G}$*

oracle, and at most $q_{\mathcal{S}}$ queries to her signing oracle, we have

$$\Pr\left[\mathsf{Game}_2^{EDL}(\mathcal{F}) : \mathsf{win}\right] \leq \mathsf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g,q}(\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}) + \frac{q_{\mathcal{G}}+1}{q}$$

Proof. We split the probability depending on whether the forgery breaks the discrete logarithm equality proof's soundness, and bound the summands pairwise to conclude.

$$\Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{win}\right] = \Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}}\right] + \Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} \neq \mathsf{h}^{\mathsf{sk}}\right]$$

Lemma 6 bounds the second summand. We now bound the first, proving that our reduction can make use of a successful forgery run that does not break the soundness of the proof to solve its CDH challenge.

$$\Pr\left[\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}}\right] \leq \Pr\left[\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}}(\mathbb{G},g,q) : \mathsf{r} = g^{\mathsf{ab}}\right]$$

We first show that $\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F})$ and $\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}}(\mathbb{G},g,q)$—where we recall that $\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}$ is defined in Figure 4.2—are equivalent as programs until the forger returns (Line 6 in Figure 4.3). Indeed, note that the shim—up to that point—is an inlined prefix of $\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}}(\mathbb{G},g,q)$. Its code samples two field elements, hides them in the group, then calls the forger with oracles that are themselves equivalent to those used in $\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}$:

- the simulated $\mathcal{H}$ oracle from Figure 4.2 is syntactically equal to $\mathcal{H}'$;
- the simulated $\mathcal{G}$ oracle from Figure 4.2 is syntactically equal to the oracle $\mathcal{G}$ defined in Figure 4.4; and the signing oracles differ only cosmetically

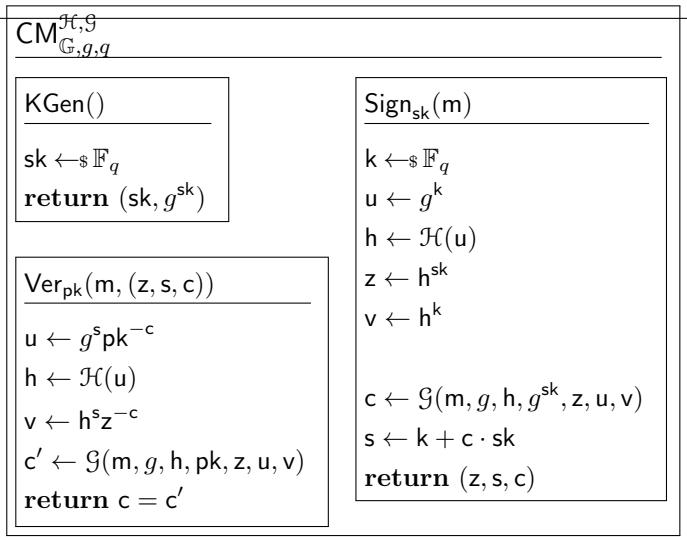

Figure 4.5: The CM signature scheme, parameterized by two random oracles $\mathcal{H} : \mathbb{G} \to \mathbb{G}$ and $\mathcal{G} : \mathcal{M} \times \mathbb{G}^6 \to \mathbb{F}_q$.

- we define $\mathsf{h} = g^{\mathsf{d}}$ in $\mathsf{Game}_2^{\mathsf{EDL}}(\mathcal{F})$, and simply propagate the definition in $\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}$; and
- we no longer keep track of $\mathsf{bad}_{\mathcal{G}}$ in the simulated signing oracle.

After the call to $\mathcal{H}$ made by the challenger at Line 6 in Figure 4.3), we have $\mathsf{H}[\mathsf{x}] = (\mathsf{g_b} g^{\mathsf{d}}, \langle \mathsf{d} \rangle_{\mathcal{A}})$ for some $\mathsf{d} \in \mathbb{F}_q$. In this case, we know that $\mathsf{h} = \mathsf{g_b} g^{\mathsf{d}}$ and the CDH adversary recovers the value d corresponding to h from the random oracle's record. We, therefore always have $\tilde{\mathsf{z}} \times \mathsf{pk}^{-\mathsf{d}} = \mathsf{g_b}^{\mathsf{sk}} g^{\mathsf{d}\cdot\mathsf{sk}} g^{-\mathsf{d}\cdot\mathsf{sk}} = \mathsf{g_b}^{\mathsf{sk}}$. Our reduction runs with $\mathsf{sk} = \mathsf{a}$ and $\mathsf{g_b} = g^{\mathsf{b}}$, where a and b are the CDH secrets. Therefore, if the forger wins with a valid forgery such that $\tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}}$, then the reduction returns the value of g^{ab}. □

Combining Lemmas 3, 4, 5, and 7 allows us to conclude the proof of Theorem 2.

4.4 CM Signatures and their Security

The CM signature scheme is defined over messages in $\mathcal{M}$ as shown in Figure 4.5. The most notable difference from EDL is that the message is not included in the random

oracle query to $\mathcal{H}$ whose output serves as the second base for the proof of discrete logarithm equality; instead, the message is included as an auxiliary input to the challenge-generating random oracle query, in a way similar to standard Schnorr signatures. This change supports security without the use of additional randomness (which shortens the signature), but also allows the use of coupons, where the most part of the signature's computation can be done offline and ahead of the message being produced. The online part of the signature then simply consists in one hash query, and two simple field arithmetic operations.

Following Chevallier-Mames [17], we formally prove a *tight* reduction to breaking CDH in $\mathbb{G}$ from breaking the EUF-CMA security of CM. The proof is, here again, concrete and constructive. The reduction is displayed in Figure 4.6. Unlike the EDL simulator, the simulator shown in Figure 4.6 must keep track of different information when simulating random oracle queries placed by the forger or internal to the simulation of signatures. We capture this in code using a tagged disjoint union, denoting with $\langle X \rangle_{t_\ell} \uplus \langle Y \rangle_{t_r}$ the set that contains values from set X tagged with t_l and values from set Y tagged with t_r (for distinct tags). We use a tagged tuple notation (with $\langle v \rangle_t$ denoting a value v tagged with tag t) to denote values in sum types, using **match** to match on the tag t and bind the tuple's elements to names provided in the pattern. We later use π as a notation to project out a tagged union's contents so that $\pi(\langle v \rangle_t) = v$.

Theorem 8 (Security of CM)**.** *If CDH is (t, ϵ)-hard in $\mathbb{G}$ with generator g, then CM is $(t', q_{\mathcal{H}}, q_{\mathcal{G}}, q_{\mathcal{S}}, \epsilon')$-EUF-CMA-secure for all non-negative $q_{\mathcal{H}}$, $q_{\mathcal{G}}$, $q_{\mathcal{S}}$ and with*

$$\begin{aligned} t &\lesssim t' + (6 \cdot q_{\mathcal{S}} + (q_{\mathcal{H}} + 1) + 5) \cdot t_{exp} \\ \epsilon' &\leq \epsilon + q_{\mathcal{S}} \cdot \left(\frac{q_{\mathcal{H}} + q_{\mathcal{S}}}{q} + \frac{q_{\mathcal{G}} + q_{\mathcal{S}}}{q^2} \right) \\ &\quad + \frac{q_{\mathcal{G}} + 1}{q} + 2 \cdot \frac{q_{\mathcal{S}} \cdot (q_{\mathcal{G}} + 1)}{q} \end{aligned}$$

where t_{exp} is the cost of an exponentiation in $\mathbb{G}$.

Proof. The complexity bound is not formally verified, so we argue the time bound here: the reduction (see Figure 4.6) first runs the EUF-CMA forger, simulating its oracles. When simulating $\mathcal{H}$, the reduction computes one exponentiation and one product in $\mathbb{G}$

$\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}(\mathsf{pk}, \mathsf{g_b})$

var $\mathsf{H} \in \mathbb{G} \rightharpoonup \mathbb{G} \times (\langle \mathbb{F}_q \rangle_{\mathcal{A}} \uplus \langle \mathbb{F}_q \times \mathbb{F}_q \rangle_{\mathcal{S}})$
var $\mathsf{G} \in \mathcal{M} \times \mathbb{G}^6 \rightharpoonup \mathbb{F}_q$

$(\tilde{\mathsf{m}}, (\tilde{\mathsf{z}}, \tilde{\mathsf{s}}, \tilde{\mathsf{c}})) \leftarrow \mathcal{F}^{\mathcal{H},\mathcal{G},\mathsf{Sign}}(\mathsf{pk})$
$\mathsf{u} \leftarrow g^{\tilde{\mathsf{s}}} \mathsf{pk}^{-\tilde{\mathsf{c}}}$
$\_ \leftarrow \mathcal{H}(\mathsf{u})$ // This is to ensure $\mathsf{u} \in H$
match $(\pi_2(\mathsf{H}[\mathsf{u}]))$ **with**
| $\langle \mathsf{d} \rangle_{\mathcal{A}} \mapsto$ **return** $\tilde{\mathsf{z}} \cdot \mathsf{pk}^{-\mathsf{d}}$
| $\langle \mathsf{s}, \mathsf{c} \rangle_{\mathcal{S}} \mapsto$ **return** $\mathsf{g_b}^{\frac{\mathsf{s}-\tilde{\mathsf{s}}}{\mathsf{c}-\tilde{\mathsf{c}}}}$

Oracle simulation

$\mathcal{H}(\mathsf{u})$
$\mathsf{d} \leftarrow\!\!\$\, \mathbb{F}_q$
if $\mathsf{u} \notin \mathsf{H}$
$\lfloor\ \mathsf{H}[\mathsf{u}] \leftarrow (\mathsf{g_b} \cdot g^{\mathsf{d}}, \langle \mathsf{d} \rangle_{\mathcal{A}})$
return $\pi_1(\mathsf{H}[\mathsf{u}])$

$\mathcal{G}(\mathsf{x})$
$\mathsf{c} \leftarrow\!\!\$\, \mathbb{F}_q$
if $\mathsf{x} \notin \mathsf{G}$
$\lfloor\ \mathsf{G}[\mathsf{x}] \leftarrow \mathsf{c}$
return $\mathsf{G}[\mathsf{x}]$

$\mathsf{Sign}(\mathsf{m})$
$\mathsf{d} \leftarrow\!\!\$\, \mathbb{F}_q$
$\mathsf{s} \leftarrow\!\!\$\, \mathbb{F}_q$
$\mathsf{c} \leftarrow\!\!\$\, \mathbb{F}_q$

$\mathsf{u} \leftarrow g^{\mathsf{s}} \mathsf{pk}^{-\mathsf{c}}; \mathsf{h} \leftarrow g^{\mathsf{d}}$
$\mathsf{z} \leftarrow \mathsf{pk}^{\mathsf{d}}; \mathsf{v} \leftarrow \mathsf{h}^{\mathsf{s}} \mathsf{z}^{-\mathsf{c}}$

$\mathsf{H}[\mathsf{u}] \leftarrow (\mathsf{h}, \langle \mathsf{s}, \mathsf{c} \rangle_{\mathcal{S}})$
$\mathsf{G}[\mathsf{m}, g, \mathsf{h}, \mathsf{pk}, \mathsf{z}, \mathsf{u}, \mathsf{v}] \leftarrow \mathsf{c}$
return $(\mathsf{z}, \mathsf{s}, \mathsf{c})$

Figure 4.6: The reduction $\mathcal{A}$ to CDH. $\mathcal{A}$ uses an EUF-CMA forger $\mathcal{F}$ as a black-box, and internally simulates $\mathcal{H}$ and $\mathcal{G}$ through initially empty finite maps H and G. H keeps track of both the response h and the random exponents s and c used in the related signature (for queries made by the signing oracle), or the value $\log_g(\mathsf{h} \cdot \mathsf{g_b}^{-1})$ (for direct queries). π_1 and π_2 are the first and second projections on pairs.

$\mathsf{Game}^{\mathsf{CM}}_{\mathcal{H},\mathcal{G},\mathcal{S},\mathcal{F}}()$

$\mathsf{sk} \leftarrow\!\!\$\, \mathbb{F}_q$
$\mathsf{b} \leftarrow\!\!\$\, \mathbb{F}_q$
$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
$\mathsf{g_b} \leftarrow g^{\mathsf{b}}$
$(\tilde{\mathsf{m}}, (\tilde{\mathsf{z}}, \tilde{\mathsf{s}}, \tilde{\mathsf{c}})) \leftarrow \mathcal{F}^{\mathcal{H},\mathcal{G},\mathcal{S}}(\mathsf{pk})$
$\mathsf{u} \leftarrow g^{\mathsf{s}} \mathsf{pk}^{-\tilde{\mathsf{c}}}$
$\mathsf{h} \leftarrow \mathcal{H}(\mathsf{u})$
$\mathsf{v} \leftarrow \mathsf{h}^{\tilde{\mathsf{s}}} \mathsf{pk}^{-\tilde{\mathsf{c}}}$
$\mathsf{c} \leftarrow \mathcal{G}(\tilde{\mathsf{m}}, g, \mathsf{h}, \mathsf{pk}, \tilde{\mathsf{z}}, \mathsf{u}, \mathsf{v})$
$\mathsf{win} \leftarrow \tilde{\mathsf{c}} = \mathsf{c} \wedge \tilde{\mathsf{m}} \notin \mathcal{Q}_{\mathcal{S}}$

Figure 4.7: The $\mathsf{Game}^{\mathsf{CM}}_{\cdot,\cdot,\cdot,\cdot}()$ shim used in the security proof for CM. We use a dashed box to isolate code that serves as the core of the reduction (Figure 4.6).

(whose cost we omit). Simulating $\mathcal{G}$ is straightforward and incurs no cost. Simulating signature queries requires two exponentiations and two double exponentiations (which we cost as four exponentiations). Finally, extracting the CDH solution from the simulator's state and the forger's output costs one $\mathcal{H}$ query, two double exponentiations and one exponentiation (with some operations on exponents whose cost is omitted, as dominated by the cost of exponentiations in $\mathbb{G}$). □

The rest of this Section is dedicated to discussing the proof. As with EDL, we include a high-level and intuitive overview and details of its formalization, but do so at a lesser level of detail, focusing instead of aspects that differ, and patterns that are common to both proofs.

4.4.1 Proof Overview

The security argument for CM signatures is slightly less intuitive. Indeed, the statement being proved as part of the signing process is entirely independent from the message which is instead included into the challenge. This means that valid forgeries can in fact involve a query to $\mathcal{H}$ that was made by the signing oracle.

If the forger produces a forgery such that $\mathsf{z} \neq \mathsf{h}^{\mathsf{sk}}$, or in cases where the verification of the forgery involves a fresh query to $\mathcal{H}$, or one that was made by the forger, the proof is exactly as that of EDL. However, there is one additional case to consider. Indeed, the final reduction for the EDL scheme exploits the fact that the message to be signed is included in the input to $\mathcal{H}$ to deduce that a fresh forgery *must* involve a query to $\mathcal{H}$ whose response contains the CDH challenge g^{b}.

In CM, we cannot exclude a valid signature $(\tilde{\mathsf{z}}, \tilde{\mathsf{s}}, \tilde{\mathsf{c}})$ on some fresh message $\tilde{\mathsf{m}}$ where $\mathsf{u} = g^{\tilde{\mathsf{s}}}\mathsf{pk}^{-\tilde{\mathsf{c}}}$ was previously used as input to $\mathcal{H}$ in a signing query. In such a situation, however, we have two *valid* pairs (c, s) and $(\tilde{\mathsf{c}}, \tilde{\mathsf{s}})$ such that $g^{\mathsf{s}}\mathsf{pk}^{-\mathsf{c}} = \mathsf{u} = g^{\tilde{\mathsf{s}}}\mathsf{pk}^{-\tilde{\mathsf{c}}}$ (the former from the simulator's log of the original query of $\mathcal{H}$ on u, and the latter from the forgery itself), and we can recover the secret key as $\mathsf{sk} = \frac{\mathsf{s}-\tilde{\mathsf{s}}}{\mathsf{c}-\tilde{\mathsf{c}}}$ when $\mathsf{c} \neq \tilde{\mathsf{c}}$. There is a low probability of having $\mathsf{c} = \tilde{\mathsf{c}}$ here, which is accounted for using a final failure event col, which captures collisions in the output of $\mathcal{G}$, and whose probability accounts for the final term in the probability bound. (We in fact need to capture only *some* such collisions, since we also know that the forgery is fresh. Details are discussed in relation to Lemma 13, below.) As such, and interestingly, the proof still leverages the special

soundness of Schnorr proofs, but does so *without* making use of the forking lemma. The scheme itself is designed so that the forger gives the simulator (via random oracle queries) enough information to fork its execution without rewinding when producing a forgery from which a CDH solution can only be extracted via special soundness.

4.4.2 Formalisation

As before, throughout the formal proof, we make use of a shim game (Figure 4.7), which remains unchanged except in the first and last steps, allowing us to focus the formal reasoning (and, here, the discussions) on meaningful changes in the oracles and to limit boilerplate proof artefacts. As before, variables shared between the shim and the intermediate oracle definitions in Figure 4.8 are simply made global in the shim so they can be accessed directly.

Figure 4.8 shows the oracles we use in the sequence of games, with claims on the game transitions displayed as Lemmas 9 and 10, and on the final reduction as Lemma 13. In our CM sequence of games, we omit details of the refactoring step, but display the corresponding oracles (in Figure 4.8) for completeness and clarity in proofs.

Step 0 — Embedding

In this proof step (Lemma 9), we change the way in which queries to $\mathcal{H}$ are handled. We expose $\mathcal{H}'$ to the forger so that $\mathsf{g_b}$ can easily be recovered by the simulator from the answers given, without affecting their distribution. In the signing oracle, we program answers to internal $\mathcal{H}'$ queries regardless of previous queries. The corresponding simulations are shown in Figure 4.8.

As with the EDL proof, oracle $\mathcal{S}_1$ keeps track of a failure event $\mathsf{bad}_\mathcal{G}$ that will only become relevant in Lemma 10.

Lemma 9 (CM Embedding)**.** *For all forgers $\mathcal{F}$ that make at most $q_\mathcal{H}$ queries to their $\mathcal{H}$ oracle, and at most $q_\mathcal{S}$ queries to their signing oracles, we have*

$$\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{CM}(\mathcal{F}) \leq \Pr\left[\mathsf{Game}^{CM}_1(\mathcal{F}) : \mathsf{win}\right] + q_\mathcal{S} \cdot \frac{q_\mathcal{H} + q_\mathcal{S}}{q}$$

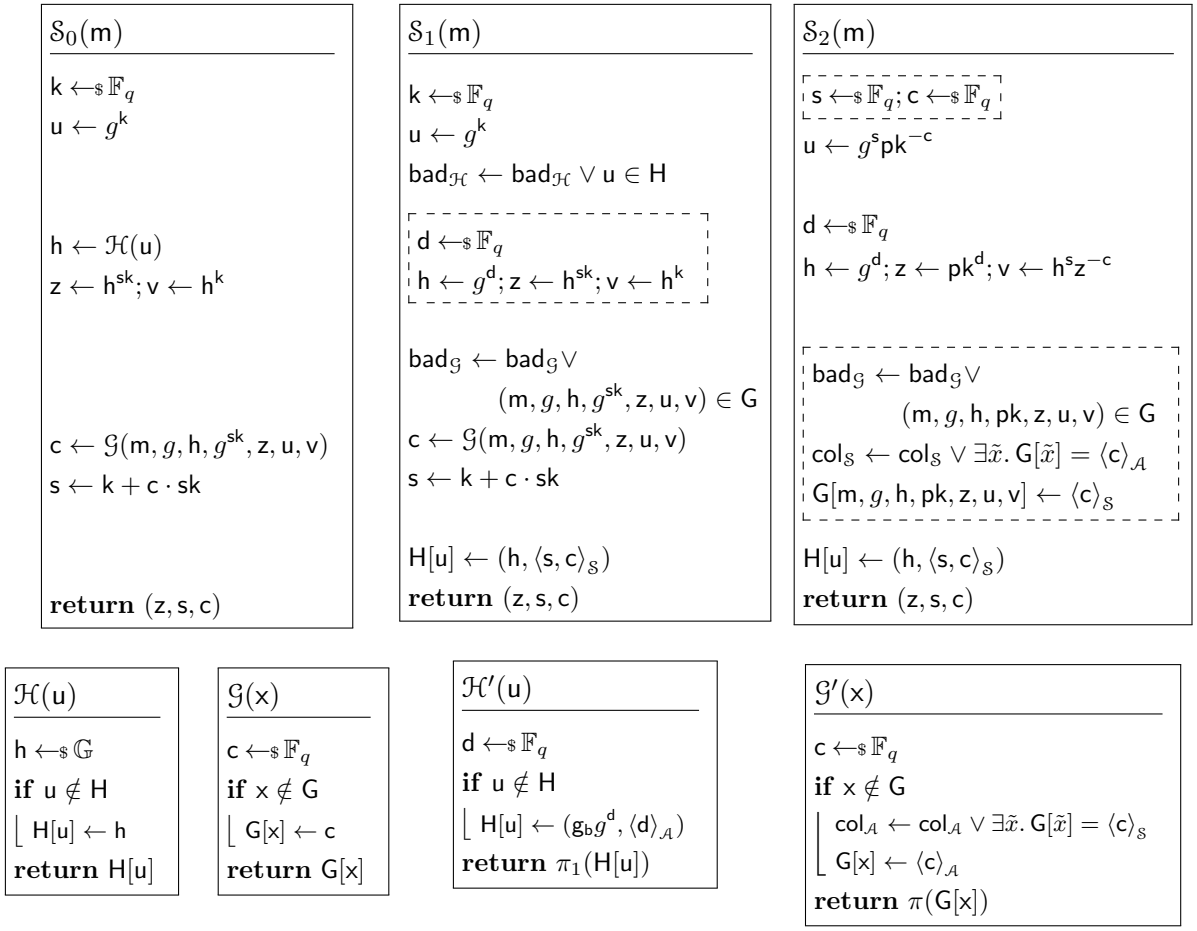

Figure 4.8: Oracle simulations for use in intermediate steps in the security proof for Chevallier-Mames signatures. Lemmas 9 and 10 use $\mathsf{Game}_1^{\mathsf{CM}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{CM}}_{\mathcal{H}',\mathcal{G},\mathcal{S}_1,\mathcal{F}}()$. Lemmas 10 and 13 use $\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) := \mathsf{Game}^{\mathsf{CM}}_{\mathcal{H}',\mathcal{G}',\mathcal{S}_2,\mathcal{F}}()$.

Proof. The proof is an easy replay of those of Lemma 3 and Lemma 4, noting that the randomiser in the input to $\mathcal{H}$ is now sampled in $\mathbb{G}$ instead of $\mathcal{N}$. □

Step 0 — Simulation

The second step (Lemma 10) simulates the zero-knowledge proof by programming the relevant random oracle in the signing oracle. In this proof, unlike in that for EDL, we need to modify also the random oracle exposed to the forger as $\mathcal{G}'$ to detect collisions in $\mathcal{G}'$ as they happen, and to keep track of which of the forger or signing oracle made particular queries (by internally tagging responses). We wield both of these tools in the proof of Lemma 13, below.

Lemma 10 (CM Simulation)**.** *For all forgers $\mathcal{F}$ that make at most $q_{\mathcal{G}}$ queries to their $\mathcal{G}$ oracle and at most $q_{\mathcal{S}}$ queries to their signing oracle, we have*

$$\Pr\left[\mathsf{Game}_1^{CM}(\mathcal{F}) : \mathsf{win}\right] \leq \Pr\left[\mathsf{Game}_2^{CM}(\mathcal{F}) : \mathsf{win}\right] + q_{\mathcal{S}} \cdot \frac{q_{\mathcal{G}} + q_{\mathcal{S}}}{q^2}$$

Proof. The proof is the same as that of Lemma 5, noting in addition that the $\mathsf{col}_{\mathcal{S}}$ and $\mathsf{col}_{\mathcal{A}}$ events do not affect the oracles' behaviour, and that neither does internally tagging responses from $\mathcal{G}'$ with the party that first observed them. □

Step 0 — Reduction

Finally, the reduction step shows that if a forger $\mathcal{F}$ wins $\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F})$, the CDH adversary $\mathcal{A}^{\mathcal{F}}$ from Figure 4.6 succeeds in solving its given CDH instance, except with low probability. We prove (Lemma 13) that a successful forgery: i. solves the given CDH instance; ii. relies on an unsound proof of discrete logarithm equality (Lemma 11); or iii. finds a (restricted) collision in the random oracle (Lemma 12).

We start by bounding the two events that prevent the simulator from solving CDH.

Lemma 11. *For all forgers $\mathcal{F}$ that make at most $q_{\mathcal{G}}$ queries to their $\mathcal{G}$ oracle, we have*

$$\Pr\left[\mathsf{Game}_2^{CM}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} \neq \mathsf{h}^{\mathsf{sk}}\right] \leq \frac{q_{\mathcal{G}} + 1}{q}$$

Proof. The proof is identical to that of EDL (Lemma 6). □

Lemma 12. *For all forgers $\mathcal{F}$ that make at most $q_{\mathcal{G}}$ queries to their $\mathcal{G}$ oracle and at most $q_{\mathcal{S}}$ queries to their signing oracle, we have*

$$\Pr\left[\mathsf{Game}_2^{CM}(\mathcal{F}) : \mathsf{col}_{\mathcal{A}} \vee \mathsf{col}_{\mathcal{S}}\right] \leq 2 \cdot \frac{q_{\mathcal{S}} \cdot (q_{\mathcal{G}} + 1)}{q}$$

Proof. First note that

$$\Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{col}_{\mathcal{A}} \vee \mathsf{col}_{\mathcal{S}}\right] \leq \Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{col}_{\mathcal{A}}\right] + \Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{col}_{\mathcal{S}}\right]$$

We now show that $\Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{col}_{\mathcal{A}}\right] \leq \frac{q_{\mathcal{S}} \cdot (q_{\mathcal{G}}+1)}{q}$.

The $\mathsf{col}_{\mathcal{A}}$ event occurs when the freshly sampled value c coincides with one of the c values previously sampled by the signing oracle. There are at most $q_{\mathcal{S}}$ such values, and the event occurs with probability at most $\frac{q_{\mathcal{S}}}{q}$ during each one of the at most $q_{\mathcal{G}} + 1$ queries made to $q_{\mathcal{G}}$ during the execution of $\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F})$ (at most $q_{\mathcal{S}}$ made by the forger, and one made to validate the forgery).

To conclude the proof, we establish the same bound on $\Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{col}_{\mathcal{S}}\right]$ using a symmetric argument (with the roles of $q_{\mathcal{G}} + 1$ and $q_{\mathcal{S}}$ reversed). □

Lemma 13 (CM Reduction)**.** *For all forgers $\mathcal{F}$ that make at most $q_{\mathcal{G}}$ queries to their $\mathcal{G}$*

oracle, and at most $q_{\mathcal{S}}$ queries to their signing oracle, we have

$$\Pr\left[\mathsf{Game}_2^{CM}(\mathcal{F}) : \mathsf{win}\right] \leq \mathsf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g,q}(\mathcal{A}^{\mathcal{F}}) + \frac{q_{\mathcal{G}}+1}{q} + 2 \cdot \frac{q_{\mathcal{S}} \cdot (q_{\mathcal{G}}+1)}{q}$$

Proof. First, note that we can split the forger's success probability as follows, for any forger $\mathcal{F}$.

$$\Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{win}\right] = \Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}}\right] + \Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} \neq \mathsf{h}^{\mathsf{sk}}\right]$$

We can further split the first summand based on whether one of the collision events captured as $\mathsf{col}_{\mathcal{A}}$ and $\mathsf{col}_{\mathcal{S}}$ occurred during the run.

$$\begin{aligned}\Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}}\right] \\ \leq \Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{col}_{\mathcal{A}} \vee \mathsf{col}_{\mathcal{S}}\right] \\ + \Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}} \wedge \neg(\mathsf{col}_{\mathcal{A}} \vee \mathsf{col}_{\mathcal{S}})\right]\end{aligned}$$

With Lemmas 11 and 12, it is now sufficient to prove that a forger that wins with a forgery such that $\mathsf{z} = \mathsf{h}^{\mathsf{sk}}$ and without causing a collision in $\mathcal{G}$ allows our reduction to solve its given CDH instance.

$$\Pr\left[\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F}) : \mathsf{win} \wedge \tilde{\mathsf{z}} = \mathsf{h}^{\mathsf{sk}} \wedge \neg(\mathsf{col}_{\mathcal{A}} \vee \mathsf{col}_{\mathcal{S}})\right] \leq \Pr\left[\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}}(\mathbb{G},g,q) : r = g^{ab}\right]$$

First observe that $\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F})$ and $\mathsf{Exp}^{\mathsf{cdh}}_{\mathcal{A}}(\mathbb{G},g,q)$, as programs, share i. their oracles (up to projection on $\mathcal{G}$, and the removal of code that tracks failure events $\mathsf{bad}_{\mathcal{G}}$, $\mathsf{col}_{\mathcal{S}}$

and $\mathsf{col}_{\mathcal{A}}$); and ii. a common prefix, which ends at the end of the dashed box shown in Figure 4.7 (for $\mathsf{Game}_2^{\mathsf{CM}}(\mathcal{F})$) and after the third line of $\mathcal{A}^{\mathcal{F}}_{\mathbb{G},g,q}$ (for the CDH game). Up until those points, the two programs are perfectly equivalent, and produce the same state (up to projections on G).

Consider two possible cases after the call to $\mathcal{H}$ made by the challenger as part of the verification query (or, rather, at the end of the dashed box in Figure 4.7): either i. $\mathsf{H}[\mathsf{u}] = (\mathsf{g_b} g^{\mathsf{d}}, \langle \mathsf{d} \rangle_{\mathcal{A}})$ for some d (when u was not used in a signature); here, the same proof as in EDL applies, or ii. $\mathsf{H}[\mathsf{u}] = (g^{\mathsf{d}}, \langle \mathsf{s}, \mathsf{c} \rangle_{\mathcal{S}})$ for some d, s and c (when u was used in a signature).

In the second case, we have $\mathsf{H}[\mathsf{u}] = (g^{\mathsf{d}}, \langle \mathsf{s}, \mathsf{c} \rangle_{\mathcal{S}})$ for some d, s and c such that $g^{\mathsf{s}-\mathsf{sk}\cdot\mathsf{c}} = \mathsf{u} = g^{\tilde{\mathsf{s}}-\mathsf{sk}\cdot\tilde{\mathsf{c}}}$ (where the first equality comes from the fact that the signing oracle only inserts pairs of values with that property into H, and the second equality comes from the construction of u by the reduction). If $\mathsf{c} \neq \tilde{\mathsf{c}}$, then we can compute the discrete logarithm $\mathsf{sk} = \frac{\mathsf{s}-\tilde{\mathsf{s}}}{\mathsf{c}-\tilde{\mathsf{c}}}$ of pk, allowing us to solve the CDH instance $(\mathsf{pk}, \mathsf{g_b})$ given as input to $\mathcal{A}$—simply by computing $\mathsf{g_b}^{\mathsf{sk}}$ ($\mathsf{sk} = \mathsf{a}$ and $\mathsf{g_b} = g^{\mathsf{b}}$, where a and b are the CDH secrets).

It remains to show that we are in a case where $\mathsf{c} \neq \tilde{\mathsf{c}}$. We know that u was queried to $\mathcal{H}$ by the signing oracle. Therefore, it must be that there exists a message m such that $\mathsf{G}[\mathsf{m}, g, \mathsf{h}, \mathsf{pk}, \mathsf{h}^{\mathsf{sk}}, \mathsf{u}, \mathsf{u}^{\mathsf{sk}}] = \langle \mathsf{c} \rangle_{\mathcal{S}}$ (since the signing oracle always inserts such an element in G after inserting an element in H). Since the forgery is fresh, it must be that $\mathsf{m} \neq \tilde{\mathsf{m}}$. Since the forgery is also valid, it must also be that $\mathsf{G}[\tilde{\mathsf{m}}, g, \mathsf{h}, \mathsf{pk}, \mathsf{h}^{\mathsf{sk}}, \mathsf{u}, \mathsf{u}^{\mathsf{sk}}] = \langle \tilde{\mathsf{c}} \rangle_{\mathcal{A}}$ (the query may be fresh; this does not impact the reasoning). Therefore, it must be that $\mathsf{c} \neq \tilde{\mathsf{c}}$ or one of $\mathsf{col}_{\mathcal{A}}$ or $\mathsf{col}_{\mathcal{S}}$ would have occurred. □

As before, combining Lemmas 9, 10 and 13 allows us to conclude the proof of Theorem 8.

4.5 Conclusion

We now briefly discuss and reflect on the formal development itself before discussing related work (both on tight DL-based signatures and machine-checked cryptographic proofs) and highlighting interesting directions for further generalizations beyond the shim presented here and beyond simple digital signatures.

We now discuss some aspects which are important to be discussed to have an understanding of how the formalisation of both schemes was developed.

Differences between paper and formal proofs The bounds we prove formally are slightly tighter than those presented in this paper.

In practice, our formal proof bounds the probability of $\mathsf{bad}_{\mathcal{H}}$ as $\Pr\left[\mathsf{Game}_1^{\mathsf{EDL}}(\mathcal{F}) : \mathsf{bad}_{\mathcal{H}}\right] \leq q_{\mathcal{S}} \cdot \frac{q_{\mathcal{H}}+\frac{q_{\mathcal{S}}-1}{2}}{q}$ instead of $q_{\mathcal{S}} \cdot \frac{q_{\mathcal{H}}+q_{\mathcal{S}}}{q}$ for EDL, and has similarly tighter bounds for $\mathsf{bad}_{\mathcal{G}}$ and for both events in the CM proof. This is simply due to more precise accounting of queries: during the ith query to $\mathcal{S}$, we can bound the number of entries in H by $q_{\mathcal{H}} + i - 1$ instead of $q_{\mathcal{H}} + q_{\mathcal{S}}$, and the bound is $\sum_{0 \leq i < q_{\mathcal{S}}} q_{\mathcal{H}} + i$ instead of the bound $\sum_{0 \leq i < q_{\mathcal{S}}} q_{\mathcal{H}} + q_{\mathcal{S}}$ discussed in the proofs above. We choose to keep the bound (and related discussions) simple in this presentation, as these details add little to the paper's contributions while making the pen-and-paper argument more difficult to trust.

In order to formally obtain the tight bounds discussed here, our formal shim is slightly more complex than the ones shown in Figures 4.3 and 4.7, and takes two different $\mathcal{G}$ oracles, with the second used only in the shim's verification query. This allows us to instantiate the shim's verification algorithm with a version of $\mathcal{G}$ that does not count towards the total number of queries when bounding the probability of $\mathsf{bad}_{\mathcal{G}}$ in the simulation step.

Related to this, an unfortunate amount of the complexity in the formal proof comes from the need to explicitly count oracle queries when bounding probabilities. In practice, EasyCrypt in fact requires that oracles that may trigger the event whose probability is being bounded be called a statically bounded number of times. Our formal development uses the manipulations discussed by Barthe et al. [43] to instead bound the adversary. We expect ongoing work on formalizing complexity analysis to better support such analyses in future developments.

Proof Effort and Challenges An initial version of the formal proof for EDL was developed over the course of 6 person-months by a novice to both cryptography and formal proof. The starting point was Goh and Jarecki's 2003 proof [16], which our initial proof followed and extended with additional failure events. The initial proof for CM was carried out in one person-week by an experienced EasyCrypt developer, relying heavily on the insights gained during the EDL formalization (in particular, adapting the sequence of games instead of starting from the direct reduction). The development of the common shim, along with adapting the existing proofs to make use of it, took one additional person-week. The effort involved in its ideation is more difficult to quantify, as it occurred continuously through the roughly 6.5 person months of development—and additional reading of related work.

The proof relies on algebraic arguments which form the core of its practical difficulty (past initially formalizing the sequence of games). Further support for symbolic techniques (perhaps inspired by the tools discussed by Barthe et al. [115]) would greatly simplify our proofs, and future proofs in DL-based settings.

Lessons Learned Our formal efforts started from existing pen-and-paper proofs, based on direct reductions. We first isolated the three steps on paper and almost immediately started formalising the arguments. Although we converged relatively quickly towards the pattern *embedding, simulation, reduction*, initial formalisation efforts which placed a different order on the proof steps were in large part wasted—derivations of group and field arithmetic results were mainly preserved. This—once again—highlights the value of first gaining as full an understanding as possible on paper of the arguments to formalise before starting the machine-checking effort itself.

On the other hand, later iterations to adapt the full proofs to make use of the shims were almost lossless, in the sense that the main changes we needed to make to the proofs were *removals* of repeated arguments. These modifications were only made possible by the identification—through the initial formalisations—of the proofs' common structure.

Striking the right balance between gaining understanding *before* embarking on a formalisation effort and gaining understanding *through* the formalisation effort remains a delicate exercise. We cannot offer insights, but hope that this simple observation encourages both more thorough pen-and-paper arguments and more deliberate experimentation with formalisation.

4.6 Further Generalisations

The security proofs of EDL and CM, as initially presented by Chevallier-Mames [17], are very similar—and indeed carry over very similar objects. In the formalisation effort of both EDL and CM schemes, we purposefully sought out similarities and attempted to factor them out. We believe the proof pattern extracted in this way could be adapted to other constructions—existing or new—to obtain new proofs of tight security. The EDL and CM schemes are formally defined with three intermediate games, which bridge the gap between the EUF-CMA game and CDH game.

In particular, we outline a three-step proof structure, that relies on:

- *embedding* the underlying challenge into random oracle answers;
- *simulating* the zero-knowledge proof; and
- *reducing* from a combination of *soundness* of the zero-knowledge proof and the underlying computational assumption (perhaps via *special soundness*).

Those intermediate games have deep insight, which gives an instance of a common game called Shim. It is not incorrect to say that this generalisation gives the first comprehensive use of the game Shim which only allows one to focus reasoning on the parts of the proofs that change between the two successive games, which helps hugely to reduce the boilerplate proof amount. The formalisation effort shows that any two successive games differ only in the oracles provided to the Shim. Throughout the formal proof of both schemes, the technique only used the Shim game, which remained unchanged except in the first and the last steps. This technique allows us to focus the formal reasoning on only meaningful changes which only happened in the oracles. We believe that proof extracted in such a pattern could be replayed and reused to similar signature schemes whose security is based on a tight discrete logarithm.

We now discuss potential directions for further generalizations.

Embedding in EDL and CM involves *computing* the random oracle answer h from $\mathsf{g_b}$ *and some trapdoor information* d such that h is distributed uniformly at random in $\mathbb{G}$ but such that $f(\mathsf{d}, \mathsf{h}^{\mathsf{sk}}) = \mathsf{g_b}$. In this context, the embedding operation is in fact a one-time pad, and our proof relies on its malleability. It is worth considering whether the proof technique could be generalised to a setting where the secrecy of the embedding is not perfect, which may be required in settings where the embedding function itself (or the extraction function f) is not as simple as in this case.

A related observation is that the relation being proved by the (sound, special sound and zero-knowledge) proof or argument needs to be tightly integrated with the embedding

function: it is only because we can malleably push exponents into the embedding that we can leverage the soundness of the proof system to extract the solution to the given hard problem instance.

The occurrence of special soundness in the CM proof may appear—at first glance—more ad hoc, but also highlights a precious ingredient in the proof pattern: it is important (for CM-style schemes that commit to the message only late in the signing process) that the witness for the relation be sufficient to solve the target hard problem.

Chapter 5

Machine-checked verification of GJKW

This chapter uses the techniques of the previous Chapter 4 to formalise the scheme of Goh et al. [6] (GJKW), using the EasyCrypt proof assistant. The formalised scheme gives insights to prove the unpredictability and show off the flexibility of the machine-checked security reduction with a visual representation.

5.1 Introduction

Chapter 4 formalised and machine-checked the two signature schemes, EDL and CM, using the technique of a general proof-step structure and common proof base technique. The techniques were used to reduce the amount of the boilerplate proof code and focused on reasoning the parts of the experiments that change between successive games. In the previous chapter also mentioned that the reduction of the Goh et al.'s [6] (GJKW) scheme is also based on a tight discrete logarithm. One can extend the proof schema and use the common proof base (Shim) from the machine-checked formalisation of the EDL and CM schemes to leverage unpredictability. So, keeping these ideas in mind, this chapter aims to formalise the third related signature scheme by Goh et al. [6], using the EasyCrypt proof assistant to show the techniques of EDL and CM schemes are general enough to apply to similar signature schemes.

Goh et al. [6] analyse the signature scheme whose security is also tightly related to the Diffie-Hellman assumption. The scheme relies on the hardness of the Computational Diffie-Hellman (CDH) problem (of finding g^{ab} given g^a and g^b). However, the highlight of their scheme is that the signer generates a single bit as a deterministic pseudorandom function of the message that results in the same signature being generated each time when a specific message is signed. Goh et al. [6] claimed to improve the efficiency of the signature while detouring the signer to record all the previous messages and signature (m, σ) pairs. For that, the signer produces b_m and r as pseudorandom functions of the message m, resulting in the same result signature being produced each time an individual message is signed. In addition, the difference between this and the proofs of EDL and CM schemes is that Goh et al. [6]'s scheme has an additional step called the *compute relevant* queries step, where the adversary performs three steps when any time the first relevant query for some message is made (see Figure 5.1). Thus, using this notable highlight and the difference in their scheme, this thesis reports on another machine-checked formalisation using EasyCrypt, which relies on automated theorem provers, SMT solvers and interactive proof assistants. Like denoting other schemes with short names during the formalisations, this chapter denotes Goh et al. [6]'s scheme with GJKW.

CompRel Queries [6]

1 : Chooses a random bit b_m and stores (b_m, m).
2 : Chooses a random $\gamma_m \in \mathbb{Z}_q$, define $h_m = \mathcal{H}(b_m, m) = g^{\gamma_m}$, and stores $(b_m, \mathsf{m}, \gamma_m)$.
3 : Computes $y = pk^{\gamma_m}$ and simulates a NIZK. Picks random (c, s), and computes $A = g^s pk^{-c}$, $B = h_m^{\mathsf{sk}} \cdot y^{-c}$.
Setting $\gamma_m = (y, c, s, b_m)$, defining $\mathcal{G}(h_m, y, A, B, \mathsf{m}) = c$ and stores $(sig, \mathsf{m}, \sigma_m)$.

Figure 5.1: Compute Relevant Queries

The security proof of the GJKW scheme is also constructed through exact security and develops the reasoning that the CDH problem is almost as hard to solve as to break the scheme. The scheme is formalised and verified in the computer-aided tool EasyCrypt, and the methodology of formally verifying the scheme is the code-based game-playing technique [15]. Remember, this technique makes small changes to the original security game until a condition is reached where the adversary cannot win the game. The proofs give a sequence of games which provide reasoning on the relations between the probabilities of events in the successive games and isolate the complicated events whose probability must be reasoned.

5.1.1 Contribution

The formalisation of the GJKW [6] scheme is verified using the EasyCrypt framework. To the best of our knowledge, this is the first machine-checked formalisation of the GJKW scheme with a tight reduction to a Diffie-Hellman assumption in the game-based proof settings. The mechanism of this proof is the evidence claimed in our research paper [44] that one can use the underlying techniques such as the common proof base and the general proof-step structure for the tight security reduction to the CDH problem. The machine-checked security proof of the GJKW uses the underlying technique to step to another level of refining the similarities and eliminating the events which are not needed, and the common proof base reduces the boilerplate codes. The resulting formalisation is now more straightforward and elegant. The chapter also presents the dimension of comparing the security proofs with EasyCrypt's proofs to help understand the novelty of the proof structure for a better understanding of the formalisation.

5.1.2 Related Work

Besides the machine-checked proofs of the EDL and CM schemes, published and presented at the conference (see Chapter 4), there are no related verified discrete logarithm-based schemes with tight reductions. However, this section only gives an overview of the related schemes with the view of cryptography rather than formal verifications.

In 1996, Bellare and Rogaway proposed a probabilistic signature scheme (PSS) that achieves a tight security reduction (i.e., the probability of forging a signature is nearly as low as inverting RSA) [57]. They also presented the Full Domain Hash (FDH) signature scheme [57], [31], which is secure in the random oracle model (ROM). They showed that the security of their scheme is tightly related to the security of the RSA function. The scheme is based on the reversing of the RSA; that is, it is hard to extract a root modulo a composite integer. Later, in 2000, Coron [116] delivered a better security reduction for the FDH, which offers tighter security proofs for bounds. In 2008, Hofheinz and Kiltz [117] proposed asymptotical tightness revisions for the CDH based signature scheme, secure without random oracles. Since then, many schemes with the techniques of security reductions have been presented to achieve tight security proofs.

One noteworthy scheme, EDL, was proposed by Katz and Wang [63], which is to tighten the security of the FDH signature scheme. In 2003, they showed that their scheme was based on their original work [16]. They claimed that their newer scheme is efficient and has the claw-free trapdoor Full Domain Hash (FDH), which fulfils tight security utilising the technique of *bit selector*. Chaum and Padersen proposed the first EDL

signature scheme in 1992 [18], and seven years later, Jakobsson and Schnorr [19] proposed its variants. However, both schemes were proved without security analysis. In 2003, Goh and Jarecki [16] proved that the EDL signature scheme is tightly related to the Computational Diffie Hellman (CDH) problem when it is constructed in the Random Oracle Model. Due to a minor omission of steps by Goh and Jarecki, Chevallier-Mames [17] presented an improvement of the EDL scheme, besides their new proposed signature scheme, which is denoted by CM here. The most recent related work is by the Chevallier-Mames [118], who proposed a new pairing-free signature scheme which extends the EDL family and its variants [6, 16, 18, 63, 119, 120]. The scheme is based on the strong Diffie-Hellman problem, and the signature is proved without using the random oracle model. This scheme certainly looks more complicated than all the prior schemes.

5.1.3 Outline

After defining the GJKW signature scheme, the next section dives directly into the machine-checked formalisation of the scheme with its security proof. Then finally, it closes with a discussion and future work.

5.2 GJKW signature scheme

The GJKW signature scheme is defined as shown in Figure 5.2, where, in addition to the cyclic group $\mathbb{G}$, it considers; a *bit* and a set $\mathcal{M}$ of messages. The scheme is defined over cyclic group $\mathbb{G}$ of prime order q with generator g.

The scheme consists of three probabilistic algorithms $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$. In the figure, the key generation algorithm (KGen) picks the secret key $\mathsf{sk} \in \mathbb{Z}_q$ and computes $\mathsf{pk} = g^{\mathsf{sk}}$, where pk is the public key. The signing algorithm (Sign) produces a non-interactive zero-knowledge (NIZK) proof of discrete logarithm equality with bases a message-dependent hash and the generator, and discrete logarithm the secret key. Upon input a secret key sk and a message $\mathsf{m} \in \mathcal{M}$, returns a signature $(\sigma = y, \pi, \mathsf{bit})$. Finally, the verification algorithm (Ver) recomputes the message-dependent hash and verifies the corresponding NIZK proof. That is, upon input a public key pk, a message $\mathsf{m} \in \mathcal{M}$ when given signature σ, returns whether σ is a valid signature of m with respect to pk.

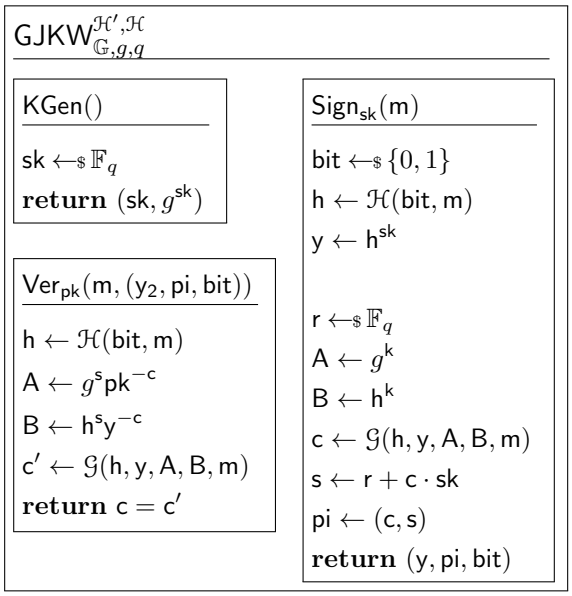

Figure 5.2: The GJKW signature scheme based on CDH, parameterized by two random oracles $\mathcal{H} : \mathcal{M} \times \mathsf{bits} \to \mathbb{G}$ and $\mathcal{G} : \mathbb{G}^5 \to \mathbb{F}_q$.

5.2.1 GJKW in EasyCrypt format

In EasyCrypt, the cryptographic algorithms and experiments are modelled as *modules*, consisting of global variables and procedures written in a simple imperative language with namespaces. The Listing 5.1 represents the definition of the GJKW's proof in EasyCrypt as a *module*. The module is parametrised by Sig-Scheme (Sig-Scheme is a sub-module and modelled as *module type* (as shown in Listing 5.2)) and two oracles, H and G, with name spaces GJKW, H and G, respectively. The name of the modules starts with an uppercase letter. In this module, there are three procedures (*proc*), named with algorithms keyGen, sign, and verify, respectively. The *map* is the global variable which any of the procedures can use, and it should be declared before the procedures which use them. Moreover, the modules can be parametrised as an abstract module, where the adversary is modelled and whose code is unknown and could be quantified over. In a procedure, after writing the local variables, there maybe a sequence of statements, such as allocating values picked from distributions to variables, if-and-else statements, and *returns* statements which should occur at the end of the calls.

Listing 5.1: Definition of GJKW scheme in EasyCrypt

```
module (GJKW : Sig_Scheme) (H : Ht.RO) (G : Gt.RO) = {
 var map : (msg, group * (zmod * zmod) * bool) fmap
 proc keygen() = {
   var sk;
   map <- empty;
   sk <$ dZp;
```

```
    return (g^sk, sk);
  }
  proc sign(sk : zmod, m) = {
    var b, h, r, c;
    if (m \notin map) {
    b <$ {0,1};
    h <@ H.get(m, b);
    r <$ dZp;
    c <@ G.get(h, h^sk, g^r, h^r, m);
    map.[m] <- (h^sk, (c, c * sk + r), b);
    }
    return oget map.[m];
  }
  proc verify(pk : group, m, sig) = {
    var y, pi, b, h, c';
    var c, s : zmod;
    (y, pi, b) <- sig;
    (c, s) <- pi;
    h <@ H.get(m, b);
    c' <@ G.get(h, y, g^s * pk^-c, h^s * y^-c, m);
    return c = c';
  }
 }.
```

Listing 5.2: Sig-Scheme is an abstract module

```
  module type Sig_Scheme(H : Ht.RO, G : Gt.RO)  = {
    proc keygen() : group * zmod
      {G.init, G.get, G.set, G.rem, G.sample,
       H.init, H.get, H.set, H.rem, H.sample}
    proc sign(_ : zmod * msg) : group * (zmod * zmod) * bool
      {G.init, G.get, G.set, G.rem, G.sample,
       H.init, H.get, H.set, H.rem, H.sample}
    proc verify(_ : group * msg * (group * (zmod * zmod) * bool)) : bool
      {G.init, G.get, G.set, G.rem, G.sample,
       H.init, H.get, H.set, H.rem, H.sample}
}.
```

module RO = { (** H oracle **)

var m : (msg * bool, group) fmap

proc get(x : msg * bool) : group = {

var r : group;

$r \leftarrow_{\$} dgp$;

if (x $\notin$ Ht.RO.m)

Ht.RO.m.[x] $\leftarrow$ r;

return oget Ht.RO.m.[x]; }

}

module RO = { (** G oracle **)

var m : (group * group * group * group * msg, zmod) fmap

proc get(x : group * group * group * group * msg) : zmod = {

var r : zmod;

$r \leftarrow_{\$} dZp$;

if (x $\notin$ Gt.RO.m)

Gt.RO.m.[x] $\leftarrow$ r;

return oget Gt.RO.m.[x]; }

}

The procedure keyGen initialises the *map* to empty, uses a random assignment to assign the secret key (sk) to uniformly chosen at random in $\mathbb{Z}_q$ and returns the public secret key pairs ($\mathsf{pk} = gg^{\mathsf{sk}}, \mathsf{sk}$).

The procedure sign takes two parameters (or arguments) sk of type $\mathbb{Z}_q$ and m of type messages. This procedure has conditional (*if*) statement, which states that if the message m is not in the *map*, then perform the following statements;

- random assignment to assign to b to a *bit*, i.e. 0 and 1,
- *procedure call assignment* to call the procedure *H.get* with arguments of m and b, (see *module* **RO** for oracle **H**),
- a random assignment to assign the variable (r) to uniformly chosen at random in $\mathbb{Z}_q$,
- *procedure call assignment* to call the procedure *G.get* with arguments (...) (see *module* **RO** for oracle **G**),
- the message m is mapped to output the values.

Finally, it returns the message m.

The procedure verify takes three parameters pk of type in cyclic group, m of type messages, and *sig* of type signature. The local variables (c, s) of type in $\mathbb{Z}_q$ and local variables can not be accessed by other procedures. After setting some values, it uses a *procedure call assignment* to call the procedure *H.get* with arguments of m and b

and another *procedure call assignment* to call $G.get$ with arguments(...). Assign $G.get$ returns $c = c'$, which checks if they are equal.

It can be seen that after writing each statement of a procedure and variable declarations, they are terminated with a semicolon. Also, a module can access the previously declared module, as mentioned in the *procedure call assignment*. Furthermore, the *module types* define the types of a set of procedures The module type *Sig-Scheme* takes two parameters, $\mathcal{H}$ and $\mathcal{G}$, and three procedures, keyGen, sign, and verify, with their types. The order of the procedures does not make any difference.

5.3 Security

The GJKW signature scheme follows the notion of *existential forgery under adaptive chosen message attacks* (EUF-CMA) in the ROM. However, this chapter skips the pen-and-paper definitions of EUF-CMA, CDH and their advantages but only illustrates the EasyCrypt's definitions (as shown in Listings 5.3 and5.4). One can find those pen-and-paper definitions in Chapter 4.

Listing 5.3: The definition of EUF-CMA experiment in EasyCrypt

```
module EUF_CMA(H : Ht.RO, G : Gt.RO, S : Sig_Scheme, A0 : CMA_Adv) = {
  var sk : zmod
  var qs : msg fset
  module O = {
    proc h(x : msg * bool) : group = H.get
    proc g(x : group * group * group * group * msg) : zmod = G.get
    proc sign(m : msg) : group * (zmod * zmod) * bool = {
      var r : group * (zmod * zmod) * bool;
      EUF_CMA.qs <- EUF_CMA.qs `|` fset1 m;
      r <@ S(H, G).sign(EUF_CMA.sk, m);
      return r;
    }
  }
  proc run() : bool = {
    var pk : group;
    var m : msg;
    var sig : group * (zmod * zmod) * bool;
    var b : bool;
    H.init();
    G.init();
    EUF_CMA.qs <- fset0;
    (pk, EUF_CMA.sk) <@ S(H, G).keygen();
    (m, sig) <@ A0(EUF_CMA(H, G, S, A0).O).forge(pk);
    b <@ S(H, G).verify(pk, m, sig);
    return b /\ ! (m \in EUF_CMA.qs);
  }
}.
```

Listing 5.4: The definition of CDH experiment in EasyCrypt

```
module CDH(A0 : CDH_Adv) = {
  proc run() : bool = {
    var a : zmod;
    var b : zmod;
    var r : group;
    a <$ dZp;
    b <$ dZp;
    r <@ A0.solve(g^a, g^b);
    return r = g^(a * b);
  }
}.
```

The formalisation of GJKW aims to be an efficient signature scheme with a *tight* security reduction; that is, to break the scheme must be as hard as to solve the underlying hard problem that the scheme employs. So, Theorem 14 shows that the adversary's success probability must be roughly equal to the probability that solves the underlying hard problem.

Also, the proof of the scheme is concrete, and shows the reduction used in proving Theorem 14 is displayed in Figure 5.3.

Theorem 14 (Security of GJKW)**.** *If the group* $\mathbb{G}$ *is* (t', ϵ')*-CDH with generator such that simulating a query to* $\mathcal{G}$ *takes time* t*, then GJKW is* $(t, q_{\mathcal{G}}, q_{\mathcal{S}}, \epsilon)$*-EUF-CMA secure, and with*

$$\begin{aligned} t &\approx t' - \mathcal{O}((q_{\mathcal{G}} + q_{\mathcal{S}}) \cdot (t)) \\ \epsilon &\leq 2\epsilon' + \frac{(q_{\mathcal{G}} + 1)}{2^p} \end{aligned}$$

where $q_{\mathcal{G}}$ *is queries to the hash function,* $q_{\mathcal{S}}$ *is queries to the signing oracle,* ϵ' *is the success probability of the adversary, and* ϵ *is the success probability that the Diffie-Hellman problem can be solved within* t*.*

Proof. The adversary has EUF-CMA forger as a black-box which internally simulates oracles $\mathcal{H}$, and $\mathcal{G}$. Figure 5.3 shows the reduction from the CDH that first runs the EUF-CMA forger and simulates its oracles. The key insight in the reduction is to embed

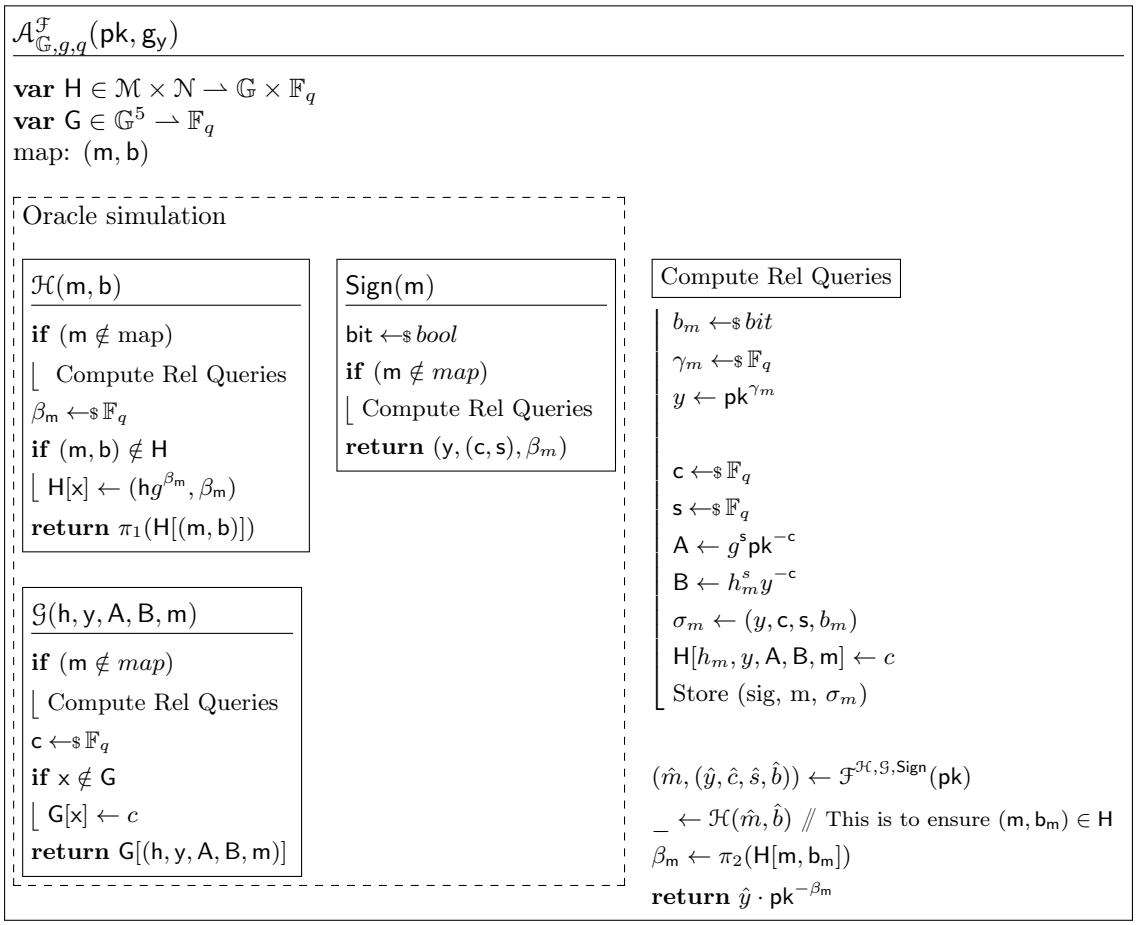

Figure 5.3: The reduction $\mathcal{A}$ from CDH.

the CDH challenge across the signing scheme and its random oracles, then simulate the signing oracle and finally use the forger's queries to simulated oracles to solve the CDH challenge. In other words, the CDH is embedded in oracle $\mathcal{H}$, NIZK proof is simulated in the signing oracle and then extracts the solution from the random oracle. Thus, it proves that the probability that this reduction solves its CDH challenge is, in fact, closely related to the probability that the forger $\mathbb{F}_q$ finds a valid forgery. □

The techniques of reducing this scheme are almost the same as EDL and CM schemes, as it was believed to be applied broadly to related signature schemes.

5.4 Formalisation

The security reduction of the GJKW scheme is constructed to prove the unforgeability under the EUF-CMA security notion in the ROM, using the code-based game-based technique in the EasyCrypt framework. The initial game is the EUF-CMA experiment, and the succeeding games are constructed by a small transition of the proceeding games. The successive games are built so that the difference in their view is efficiently quantifiable.

5.4.1 Proof Overview

The machine-checked formal verification of the GJKW's proof begins with a sequence of games, which bridges the gap between the EUF-CMA experiment and the CDH experiment. Diagram 5.4 shows the high-level overview of the proof as a sequence of games which binds the sequence of games into a proof of Theorem 14 and gives an intuitive explanation of each game transition. The machine-checked proof of the GJKW's scheme is divided into two sections, i.e. **1st simplification** and **2nd simplification**. The diagram begins with the equivalence of two games, first the experiment of EUF-CMA of GJKW and second the $Game_0$.

The $Game_0$ is split into two branches, 1 and 2, with events $(res \wedge \hat{y} \neq \hat{h}^{\mathsf{sk}})$ and $(res \wedge \hat{y} = \hat{h}^{\mathsf{sk}})$ respectively. In the second branch, $Game_0$ is further split into two branches, A and B. So, the whole second branch (i.e. on 2) bounds the probability, which is the sum of the probabilities that the simulator wins the CDH at points a and b. Here, res means the event.

The details of the games are not necessary at the moment, but the visual representation emphasises the importance of the games with their events.

Then, one can say that the bound becomes twice the probability that the simulator wins the CDH challenge. Overall the figure shows how the final statement arises, which then can lead to bound the probability of EUF-CMA of GJKW is bounded by the sum of the probabilities, that is:

$$Pr[\mathsf{EUF\text{-}CMA}(\mathsf{GJKW})] \leq \frac{(q_{\mathcal{G}} + 1)}{p} + 2 \cdot Pr[\mathsf{CDH}(Simulator(A))].$$

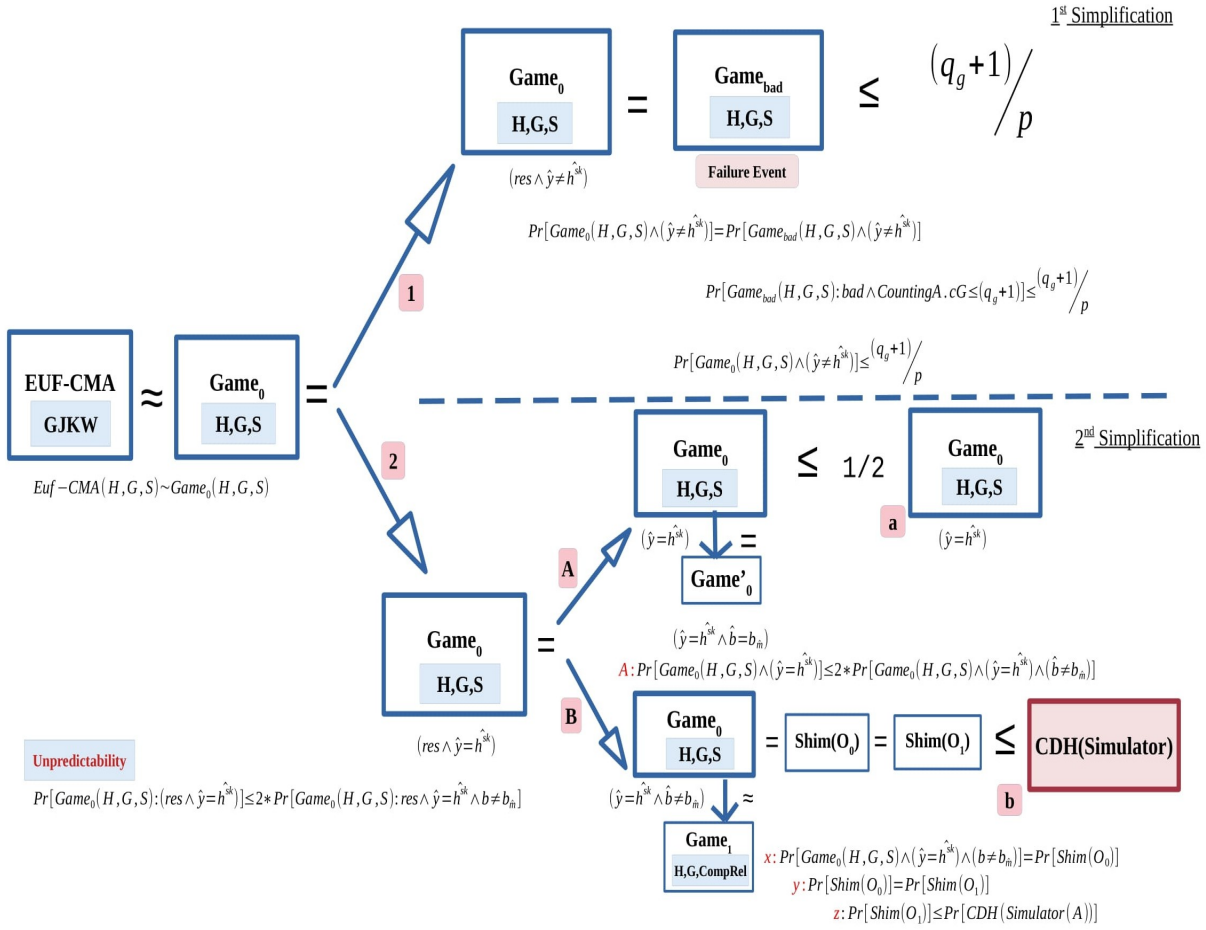

Figure 5.4: Overview of the proofs

5.4.2 The Sequence of Games

The machine-checked formalisation of the GJKW's scheme begins with the sequence of the games, where the initial game is EUF-CMA game and the final game is $Game_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$, applied with some events. Then the construction of the sequence of games is simplified into two sections (see Diagram 5.4 where the dashed line separates the simplifications). The first simplification encounters the failure event by eliminating the other events that are not essential, which later gives simpler proofs in the second simplification. The second simplification removes the cases where the adversary guesses the value used for his forgeries.

[EUF-CMA$(\mathcal{H}, \mathcal{G}, \mathsf{GJKW}, \mathcal{F})$]

The security proof begins with the initial game EUF-CMA of GJKW, where the public and secret key pairs are produced with the key generation algorithm. The adversary $\mathcal{A}$, given the public key, has access to the random oracles $\mathcal{H}$, $\mathcal{G}$ and Sign, which sign messages and produce a valid signature upon a message.

[$Game_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$]

The next game is $Game_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$, where the experiment of EUF-CMA is transformed. The game is expressed in pseudocode (as shown in Figure 5.5) to make it more transparent.

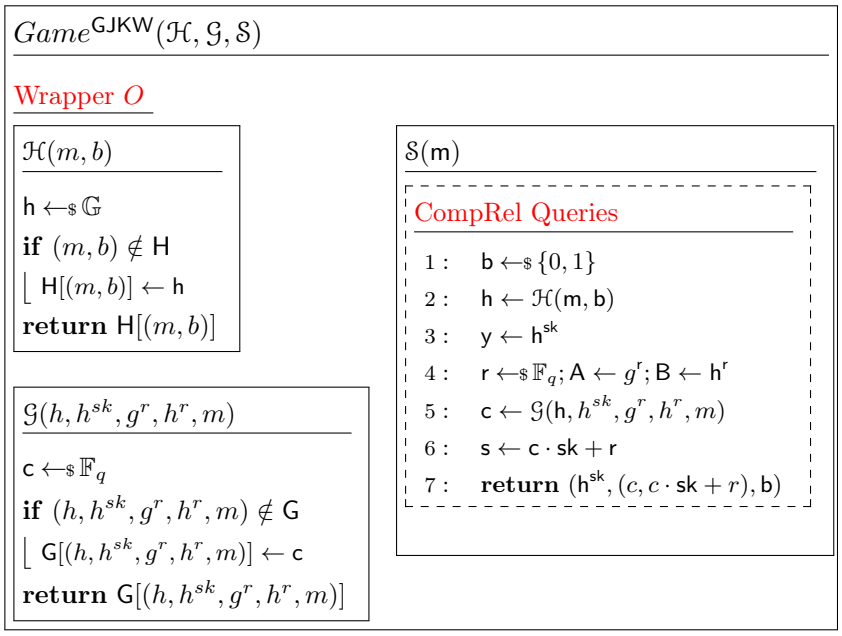

Figure 5.5: $\mathsf{Game}_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$

The following lemma shows the probabilities between the two successive experiments.

Lemma 15. *For any forger $\mathcal{F}$,*

$$\Pr\left[Exp^{\mathsf{euf\text{-}cma}}(\mathcal{H}, \mathcal{G}, \mathit{GJKW}, \mathcal{F}) : \mathsf{y} \wedge \tilde{\mathsf{m}} \notin \mathcal{Q}_{\mathcal{S}}\right] = \Pr\left[Game^{\mathit{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S}) : \mathsf{win}\right]$$

Proof. The lemma shows a simple program equivalence. The right-hand side of the

equality is the $Game_0^{\mathsf{GJKW}}$ which is the EUF-CMA experiment, applied to GJKW scheme with algorithms of key generation (KGen), signature Sign and verification (Ver) inlined in the game. In Listing 5.5, the equivalence lemma is expressed in EasyCrypt. It proves that the transition from the EUF-CMA experiment to $Game_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$ is equal, with the requirement of invariant to keep track of statements throughout the states. □

Listing 5.5: $\mathsf{Game}_0^{\mathsf{GJKW}}(\mathcal{F})$ in EasyCrypt

```
local equiv Step0:
  EUF_CMA(Ht.RO, Gt.RO, GJKW, A).run ~
  Game0(Bt.LRO, Ht.RO, Gt.RO).run:
    ={glob A} ==> ={res}.
proof.
```

The pRHL judgment is used to relate EUF-CMA.run and $Game0.run$, and ={res} means that the result of EUF-CMA.run is equal to the result of $Game0.run$.

Listing 5.6: $\mathsf{Game}_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$ in EasyCrypt

```
  local module Game0 (B : Bt.RO) (H : Ht.RO) (G : Gt.RO) = {
    var sk : zmod
    var qs : msg fset
    var  m : msg
    var  y : group
    var  c : zmod
    var  s : zmod
    var  b : bool
    var  h : group
    module O = {
      proc h = H.get
      proc g = G.get
      proc sign(m) = {
        var b, h, r, c;
          qs <- qs `|` fset1 m;
          b <@ B.get(m);
          r <@ Rt.RO.get(m);
          h <@ H.get(m, b);
          c <@ G.get(h, h^sk, gg^r, h^r, m);
        return (h ^ sk, (c, c * sk + r), b);
      }
    }
    proc run() = {
      var pk, sig, pi, c';
        H.init();
        G.init();
        B.init();
        Rt.RO.init();
        qs <- fset0;
        sk <$ dZp;
        pk <- gg^sk;
        CountingA.cG  <- 0;
```

```
        (m, sig) <@ A(O).forge(pk);
        (y, pi, b) <- sig;
            (c, s) <- pi;
            h  <@ H.get(m, b);
            c' <@ G.get(h, y, gg^s * pk^-c, h^s * y^-c, m);
                B.sample(m);
      return c = c' /\ !m \in qs;
    }
  proc distinguish = run (* Fit the RO distinguisher interface *)
}.
```

The model of developing the pseudocode of this game in the EasyCrypt framework is shown in Listing 5.6. In EasyCrypt, games are modelled as modules, containing the procedures written in a simple user-extensible imperative language. B, H, and G are used as parameters in the game. Module O wraps the procedures h, g and sign.

The signing oracle is inlined, and it makes the sampling operations for values b and r instead of sampling in the distributions. These are provided from the random oracles, parametrised with m. So, instead of sampling a fresh value every time it gets the same m, it retrieves the previously sampled value. The main idea is to ensure that values of b and r are produced pseudorandomly from m, allowing getting rid of the if-and-else conditions present in the original hand proof by GJKW, which makes the reasoning easier in the framework.

The $Game_0^{\mathsf{GJKW}}$ has global variables $(\mathsf{sk}, qs, m, y, c, s, b, h)$ with their respective types, which can be used by any inside procedures of game and outside modules. The procedure *run* takes no parameters, initialises the oracles H, G, B and R and sets the list (qs) to empty. The highlight of this procedure is that the public key (pk) is computed as g^{sk} for randomly chosen $\mathsf{sk} \in \mathbb{Z}_q$ in the procedure, and *procedure call assignment* is used to call the adversary.

The procedure *distinguish* calls directly the procedure *run*.

1st Simplification

As mentioned in the overview (see Diagram 5.4) that $Game_0^{\mathsf{GJKW}}$ further splits into two statements, 1 and 2. The first statement has the event where $(y_2 \neq h^{sk})$, which eventually proves the probability of the forger relying on an unsound proof of discrete logarithm equality. The second statement has the event where $(y_2 = h^{sk})$, which eventually proves that the reduction uses the successful forgery run by breaking the CDH challenge. The second statement will be explained in the second simplification, whereas the first statement is in the first simplification, which begins next.

[$Game_{bad}^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$: bad] (failure event)

The next game is $Game_{bad}^{\mathsf{GJKW}}$, similar to the previous game, except small changes are introduced with red texts, and the oracle $\mathcal{G}$ is programmed to account for the failure event (see the r.h.s in Figure5.6). The first change is to introduce a (global) boolean flag (*bad*) which becomes true if the signing oracle has already queried $\mathcal{G}$ on the input string $(h, h^{\mathsf{sk}}, g^r, h^r, m)$. This game is a simple game where it only cares to check if the signing oracle makes the query to oracle $\mathcal{G}$ is fresh or not.

The game programs the random oracle $\mathcal{G}$ on inputs queried by the signing oracle to return c (see $Game_{bad}^{\mathsf{GJKW}}$, point 5 in Figure5.6). Doing so introduces a difference in the behaviour if the signer has already queried $\mathcal{G}$ on the point being programmed. The event is captured as *bad* and becomes observable by the adversary. This is because the game $Game_{bad}^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$ can be distinguished from the game $Game_{0}^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$ by the forger when the signing oracle programmed oracle $\mathcal{G}$ at a point the forger had previously queried. So, the event *bad* occurs with a low probability and can be seen later in the Lemma 16.

Another minor difference between $Game_{0}^{\mathsf{GJKW}}$ and $Game_{bad}^{\mathsf{GJKW}}$ is setting the *CountingA* (see red texts in $Game_{bad}^{\mathsf{GJKW}}$ Figure 5.6), a counter and allowing one to bound the probability by counting the number of queries and placing a bound on the adversary.

Before bounding the failure event, one can show the transition from $Game_{0}^{\mathsf{GJKW}}$ to $Game_{bad}^{\mathsf{GJKW}}$ with the following probabilities:

$$\begin{aligned}&\Pr\left[Game_{0}^{\mathsf{GJKW}}(B, H, G).\mathsf{run}() : \mathsf{res} \wedge y \neq h^{\mathsf{sk}}\right] \\ &\quad = \Pr\left[Game_{bad}^{\mathsf{GJKW}}(B, H, G).\mathsf{run}() : \mathsf{res} \wedge y \neq h^{\mathsf{sk}} \wedge \mathsf{CountingA.cG} \leq (q_{\mathcal{G}} + 1)\right].\end{aligned}$$

$Game_{0}^{\mathsf{GJKW}}$ and $Game_{bad}^{\mathsf{GJKW}}$ are the same except for some differences but with different oracles, and the difference does not change the adversary's behaviour. It states that the probabilities of these two games are equivalent, except the variable (CountingA) is initialised to zero in the $Game_{bad}^{\mathsf{GJKW}}$, and adversary calls the CountingA of the oracles. This can easily be proved in EasyCrypt by just program equivalences. Here, the counter bounds the probability that the adversary forges via the unsoundness of the NIZK proof. Now, the Difference Lemma 1 is applied to capture the failure event as the flag *bad* is set. The difference in probability of any event between two games is upper bounded by the probability of the occurrence of the failure event *bad* in one of them. The following lemma shows that the $\frac{(q_{\mathcal{G}}+1)}{p}$ bounds the adversary's probability of winning with $y \neq h^{\mathsf{sk}}$.

$Game_0(\mathcal{H}, \mathcal{G}, \mathcal{S})$

1 : $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$;
2 : $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$;
3 : $(\mathsf{m}, \sigma) \leftarrow \mathcal{A}(O).forge(\mathsf{pk})$;
4 : $(y, \pi, b) \leftarrow \sigma$;
5 : $(c, s) \leftarrow \pi$;
6 : $h \leftarrow \mathcal{H}.get(\mathsf{m}, \mathsf{b})$
7 : $\mathsf{c}' \leftarrow \mathcal{G}.get(\mathsf{h}, y, g^s \cdot \mathsf{pk}^{-c}, h^s \cdot y^{-c}, m)$
8 : **return** $c = c' \wedge \neg m \in \mathcal{Q}_\mathcal{S}$;

$Game_{bad}(\mathcal{H}, \mathcal{G}, \mathcal{S})$

1 : $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$
2 : $\mathsf{pk} \leftarrow g^{\mathsf{sk}}; b \leftarrow_\$ \{0, 1\}$
3 : $h \leftarrow \mathcal{H}.get(\mathsf{m}, \mathsf{b})$
4 : $\mathsf{y} \leftarrow \mathsf{h}^{\mathsf{sk}}$;
5 : Failure event
$\mathsf{c} \leftarrow_\$ \mathbb{F}_q$;
$\mathrm{bad} \leftarrow \mathrm{bad} \vee (y \neq h^{\mathsf{sk}} \wedge (a \cdot g^{\mathsf{sk}c})^{\log_g h} = b \cdot y^c)$
$\mathcal{G}[(\mathsf{h}, h^{sk}, g^r, h^r, m)] \leftarrow c$;
6 : $\mathrm{bad} \leftarrow \mathrm{false}$;
7 : $\mathrm{CountingA} \leftarrow 0$;
8 : $(\mathsf{m}, \sigma) \leftarrow \mathcal{A}(\mathrm{CountingA}(A, O)).forge(\mathsf{pk})$;
9 : $(y, \pi, b) \leftarrow \sigma$;
10 : $(c, s) \leftarrow \pi$;
11 : $h \leftarrow \mathcal{H}.get(\mathsf{m}, \mathsf{b})$
12 : $\mathsf{c}' \leftarrow \mathrm{CountingA}(A, O).(\mathsf{h}, y, g^s \mathsf{pk}^{-c}, h^s y^{-c}, m)$
13 : **return** $c = c' \wedge \neg m \in \mathcal{Q}_\mathcal{S}$;

Figure 5.6: $Game_0^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$ and $Game_{bad}^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$

Lemma 16 (bad)**.** *For any forger* $\mathcal{F}$*,*

$$\Pr\left[Game_{bad}^{GJKW}(B, H, G).\mathsf{run}()\&\mathsf{m} : \mathsf{res} \wedge y \neq h^{\mathsf{sk}}\right] \leq \frac{(q_\mathcal{G} + 1)}{p}.$$

Proof. Now, one can bound the probability of the failure event *bad*. The boolean flag *bad* is set by the signing query if the tuple it uses as an input to oracle $\mathcal{G}$ coincides with one of the inputs on which $\mathcal{G}$ has already been queried. One can see that both games are equivalent as long as *bad* is false. Firstly, the counter *countingA* is added before

proving the bound on the failure event.

$$\Pr\left[Game_{bad}^{\mathsf{GJKW}}(B,H,G).\mathsf{run}()\&\mathsf{m} : \mathsf{res} \wedge y \neq h^{\mathsf{sk}} \wedge \mathsf{CountingA.cG} \leq (q_{\mathcal{G}}+1)\right]$$
$$\leq \Pr\left[Game_{bad}^{\mathsf{GJKW}}(B,H,G).\mathsf{run}()\&\mathsf{m} : \mathsf{bad} \wedge \mathsf{CountingA.cG} \leq (q_{\mathcal{G}}+1)\right]$$

The probability that the forger queries and gets a value c is at most $1/p$. Since $\mathcal{G}$ query was made by the forger, there are at most $(q_{\mathcal{G}}+1)$ such queries. So, the probability for any forger $\mathcal{F}$ that makes at most $q_{\mathcal{G}}$ queries to her $\mathcal{G}$ oracle is at least $\frac{(q_{\mathcal{G}}+1)}{p}$. □

Failure Event Lemma (fel tactic)

When the current goal's conclusion has the form of the above probability, then one can use the advanced tactic called Failure Event Lemma (*fel*) to reason about the failure event in EasyCrypt. Here, the $Game_{Bad}^{\mathsf{GJKW}}$ has a boolean variable *bad*, which is the bad event. So, one can use the following rules in EasyCrypt;

fel

init The first argument is the numbers of the lines in the procedure that initialises the failure event lemma.

ctr The second argument is the counter where the program variables have the expression type integer, and the counter counts the queries to the oracle.

stepub The third argument is a function of the counter's current value when the query is made and provides the probability that a specific query triggers the failure event *bad*.

bound The fourth argument becomes the maximum value of the counter that is allowed to be made by the adversary.

bad The fifth argument is an expression of a boolean type: the failure event.

conds The sixth argument is a list of *procedures*, where boolean type involves program variables and the parameters of *procedures*.

invr The last argument is an optional invariant on program variables.

The following Listing 5.7 for the *fel* tactic from EasyCrypt can help to understand better the above rules.

Listing 5.7: fel tactic in EasyCrypt

```
fel 9 CountingA.cG
     (fun _=> inv p%r)
     (qG + 1)
     Game0_bad.bad
     [Game0_bad(Bt.LRO, Ht.RO, Gt.RO).O.sign: false ]=> //.
```

Listing 5.8: $Game0_{bad}^{\mathsf{GJKW}}(\mathcal{F})$ in EasyCrypt

```
local module Game0_bad (B : Bt.RO) (H : Ht.RO) (G : Gt.RO) = {
  var sk : zmod
  var qs : msg fset
  var bad : bool         (* The bad is set *)
  var  m : msg
  var  y : group
  var  c : zmod
  var  s : zmod
  var  b : bool
  var  h : group
  module O = {
    proc h = H.get
    proc g(h, y, a, b, m) = {
      var c;
      c <$ dZp;
      if ((h, y, a, b, m) \notin Gt.RO.m) {
        bad <- bad \/ (   y <> h^sk
              /\ (a * gg^sk^c)^logge h = b * y^c);
      Gt.RO.m.[(h, y, a, b, m)] <- c;
    }
    return oget Gt.RO.m.[h, y, a, b, m];
  }
  proc sign(m) = {
    var b, h, r, c;
    qs <- qs `|` fset1 m;
    b <@ B.get(m);
    r <@ Rt.RO.get(m);
    h <@ H.get(m, b);
    c <@ G.get(h, h^sk, gg^r, h^r, m);
    return (h^sk, (c, c * sk + r), b);
    }
  }
  proc run() = {
    var pk, sig, pi, c';
    H.init();          (* 1 counting no. of lines for fel tactic*)
    G.init();              (* 2 *)
    B.init();              (* 3 *)
    Rt.RO.init();          (* 4 *)
    qs <- fset0;           (* 5 *)
    bad <- false;          (* 6 *)
    sk <$ dZp;             (* 7 *)
    pk <- gg ^ sk;         (* 8 *)
    CountingA.cG  <- O;    (* 9 *)
    (m, sig) <@ A(CountingA(A,O).Go).forge(pk);
```

```
    (y, pi, b) <- sig;
      (c, s) <- pi;
        h  <@ H.get(m, b);
        c' <@ CountingA(A,O).Go.g(h, y, gg^s * pk^-c, h^s * y^-c, m);
        B.sample(m);
  return c = c' /\ !m \in qs;
  }
 proc distinguish = run (* Fit the RO distinguisher interface *)
}.
```

Here, the number of lines is nine, as highlighted in Listing 5.8. The *CountingA.cG* is the counter, causing the probability of the *bad* event ($Game0 - bad.bad$) happening by the query is the inverse of p with $(q_{\mathcal{G}} + 1)$ is at most value allowed to be made by the adversary. The procedure Sign is set to *false* as it has not been called, and one can say that the oracle never triggers the bad event. The sign does not do anything when the counter reaches the bound $(q_{\mathcal{G}} + 1)$.

2nd Simplification

In the second simplification, on point 2, the $Game_0^{\mathsf{GJKW}}$ further splits into two statements, A $(\hat{b} = b_{\hat{m}})$ and B $(\hat{b} \neq b_{\hat{m}})$ (as shown in Figure 5.4). These two statements can be seen in the rest of the proofs. The intention behind the second simplification is to remove the cases where the adversary has to guess the value of $\hat{b}$ that can be used during the forgeries.

At branch 2, the arrow points at the $Game_0^{\mathsf{GJKW}}$ with events such as $(res \wedge \hat{y} = \hat{h}^{\mathsf{sk}})$. Here, one must know the value of $\hat{b}$, which the adversary gives as part of the forgery. The adversary wins the game with two probabilities; if she guesses the value of $\hat{b}$ correctly, then $\hat{b} = b_{\hat{m}}$ (go to branch A), the value is $\hat{b} \neq b_{\hat{m}}$ (go to branch B).

From $Game_0^{\mathsf{GJKW}}$ to $Game\prime_0^{\mathsf{GJKW}}$

In branch A), the value of $\hat{b}$ does not exist in $Game_0^{\mathsf{GJKW}}$; therefore, the $Game\prime_0^{\mathsf{GJKW}}$ ensures that the value exists. This game is inserted because sampling $\hat{b}$ corresponds to the adversary's forger, which later allows her to decide whether she has guessed it or not. This game can be proved easily with $Game_0^{\mathsf{GJKW}}$, with just equivalences for all possible events.

If the adversary manages to forge and wins the game with forgery such that $\hat{b} = b_{\hat{m}}$ (guessed correctly), then the probability is 1/2. So, the probability (A with red colour) in the Diagram 5.4 is the probability that the adversary guessed it correctly, the probability that the adversary made this event accurate, and when the value is $\hat{b} \neq b_{\hat{m}}$.

Lemma 17 (Unpredictability)**.**

$$\Pr\left[Game_0^{GJKW}(B,H,G).\mathsf{run}()\&\mathsf{m} : (\mathsf{res} \wedge y = h^{\mathsf{sk}})\right]$$
$$\leq 2 \cdot \Pr\left[Game_0^{GJKW}(B,H,G).\mathsf{run}()\&\mathsf{m} : (\mathsf{res} \wedge y = h^{\mathsf{sk}} \wedge \hat{b} \neq b_{\hat{m}})\right]$$

To reason in EasyCrypt, $Game\prime_0^{\mathsf{GJKW}}$ samples the value b, which corresponds to the $Game0.m$ in the procedure *run* (see Listing 5.9). The reason for sampling b is that adversary has made a query with value $Game0.m$ as input to the signing oracle. Therefore, the $Game0.m$ is in the list, and the adversary cannot win. If the adversary has $Game0.m$, it must be that it is not in the map $(Bt.RO.m)$; otherwise, the forgery $Game0.m$ would not be fresh. The $Game\prime_0^{\mathsf{GJKW}}$ runs the $Game_0^{\mathsf{GJKW}}$, and it makes sure to sample b, and from the adversary's point of view, $Game\prime_0^{\mathsf{GJKW}}$ is equivalent to $Game_0^{\mathsf{GJKW}}$. However, suppose the value b that the adversary gives differs as part of the signature. In that case, the probability of getting the value the adversary has at most $1/2$ of finding the forger, which was seen in the above probabilities.

Listing 5.9: $Game\prime_0^{\mathsf{GJKW}}(\mathcal{H},\mathcal{G},\mathcal{S})$ in EasyCrypt

```
local module Game0' = {
  module O = Game0(Bt.RO, Ht.RO, Gt.RO).O
  proc inner() = {
    var pk, sig, pi, c';
    Ht.RO.init();
    Gt.RO.init();
    Bt.RO.init();
    Rt.RO.init();
    Game0.qs <- fset0;
    Game0.sk <$ dZp;
    pk <- gg^Game0.sk;
    (*** we take the m (Game0.m) that the adversary
         gave us as a part of the forgery ***)
    (Game0.m, sig) <@ A(O).forge(pk);
    (Game0.y, pi, Game0.b) <- sig;
    (Game0.c, Game0.s) <- pi;
    Game0.h <@ Ht.RO.get(Game0.m, Game0.b);
    c' <@ Gt.RO.get(Game0.h, Game0.y,
            gg^Game0.s * pk^-Game0.c,
            Game0.h^Game0.s * Game0.y^-Game0.c, Game0.m);
    return Game0.c = c' /\ !Game0.m \in Game0.qs;
  }
  proc run() = {
    var r;
    r <@ inner();
    (*** samples the bit that have been used
         for the forgery ***)
    Bt.RO.sample(Game0.m);
```

```
    return r;
  }
}.
```

5.5 Formal Steps

The techniques of a single proof base to build subsequent formal steps (refactorisation, embedding, simulation and reduction) from the machine-checked formal proofs of EDL and CM signature schemes can be applied here. Note that the steps can reduce the proof obligation to obligations on oracles, and the single proof base functions as a distinguisher between the different sets of oracles in games. However, it requires a few gymnastics to manipulate the proofs to use the techniques here. So, to begin with, in the formal steps, $Game_1^{\mathsf{GJKW}}$ is applied to $Game_0^{\mathsf{GJKW}}$ in branch B. In $Game_1^{\mathsf{GJKW}}$, pre-computation to all oracles is added in three sub-steps, explained later. One can prove that $Game_1^{\mathsf{GJKW}}$ is syntactically equivalent to $Game_0^{\mathsf{GJKW}}$ (as shown in Listing 5.10).

Listing 5.10: Game equivalences in EasyCrypt

```
local equiv Game0_Game1_b:
  Game0(Bt.RO, Ht.LRO, Gt.LRO).run ~
  Game1(CR_b, Bt.RO, Rt.RO, Ht.LRO, Gt.LRO).run:
  ={glob A}
    ==>    ={glob A, res}
        /\ ={h, y, s, c, m, b, qs, sk}(Game0, Game1)
        /\ Bt.RO.m.[Game0.m]{1} = Some Game1.bm{2}.
proof.
```

Once the equivalences between $Game_0^{\mathsf{GJKW}}$ and $Game_1^{\mathsf{GJKW}}$ are proved, the transition from $Game_0^{\mathsf{GJKW}}$ to the final game begins, which tells if the adversary wins the game ($Shim(O_1)^{\mathsf{GJKW}}$), then the simulator succeeds in breaking the CDH challenge (see Figure 5.1). There are minor changes made during the transitions of the games in which their probabilities can easily be proved with equivalences, explained next. The transitions of these games are explained in the three steps:

1. the refactorisation occurs in the $Game_1^{\mathsf{GJKW}}$;
2. the embedding and simulation occur in the game $Shim(O_0)^{\mathsf{GJKW}}$;
3. the reduction step ends in the game $Shim(O_1)^{\mathsf{GJKW}}$.

Step 1: Refactorisation ($Game_1^{\mathsf{GJKW}}(\mathcal{F})$ and $Shim(O_0)^{\mathsf{GJKW}}(\mathcal{F})$)

The refactoring step in GJKW is more complicated than in prior schemes (EDL and CM). The reason is that it later supports the easier embedding and simulation steps occurring in $Game_1^{\mathsf{GJKW}}$ and $Shim(O_0)^{\mathsf{GJKW}}$. In the pen and paper proof of the GJKW's scheme, they introduced the *compute-relevant* queries steps (see Figure 5.1), which now can be seen here. So, one can insert the *compute-relevant* queries to all oracles that the adversary can call and start pre-computing the signatures. The *compute-relevant* queries are introduced into sub-steps, which are next.

3 Sub-Steps

Every time the adversary queries which contains message m, the signature computation on a message is expected in three sub-steps. In the first sub-step, it refactors and samples b, and r and ignores the result with Lazy random oracle. In the second step, samples b, r and h, observe the result of b and r with Eager random oracle and ignores the results of h. In the third step, samples c and the value of h is extracted to use as input to the $\mathcal{G}$ random oracle. The reason for samplings at a different level supports unwanted samplings that are not needed and can only insert the samplings if the values are not used and are sampled (see Listing 5.11).

Listing 5.11: Transformation of samplings in three sub-steps

```
(** Sub-step 1: Anticipate sampling of b and r **)
local module CR_b (B : Bt.RO) (R : Rt.RO) (H : Ht.RO) (G : Gt.RO) = {
  proc compute_relevant(m) = {
    B.sample(m);
    R.sample(m);
  }
}.
(** Sub-step 2: anticipate sampling of h **)
local module CR_h (B : Bt.RO) (R : Rt.RO) (H : Ht.RO) (G : Gt.RO) = {
  proc compute_relevant(m) = {
    var b, r;
    b <@ B.get(m);
    r <@ R.get(m);
    H.sample(m, b);
  }
}.
(** Sub-step 3: anticipate sampling of c **)
local module CR_g (B : Bt.RO) (R : Rt.RO) (H : Ht.RO) (G : Gt.RO) = {
  proc compute_relevant(m) = {
    var b, r, h;
    b <@ B.get(m);
    r <@ R.get(m);
    h <@ H.get(m, b);
    G.sample(h, h ^ Game1.sk, gg^r, h^r, m);
```

```
  }
}.
```

Earlier, it was mentioned that one of the differences between this scheme and the EDL or CM is the *compute-relevant* queries steps. For a message m, the queries to $\mathcal{H}(\mathsf{m}, \star)$, $\mathcal{G}(\star, \star, \star, \star, \mathsf{m})$, and $Sign_{\mathsf{sk}}(\mathsf{m})$ are relevant for message m. The *compute-relevant* oracle is inserted in $Game_1^{\mathsf{GJKW}}(\mathcal{F})$ (see O, in the l.h.s. of Figure 5.7) whenever the adversary asks for a signature or hash from $\mathcal{H}$ or $\mathcal{G}$. The $Game_1^{\mathsf{GJKW}}(\mathcal{F})$ is parametrised by *CompRel* and all random oracles (B, R, H, G). The *CompRel* is a type module parametrised by random oracles (B, R, H, G), and a module of this type implements the *compute-relevant* queries.

[$Shim(O_0)^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, \mathcal{S})$**]**

The transition from $Game_0^{\mathsf{GJKW}}$ to the CDH adversary A happens with small hops in $Shim(O_0)^{\mathsf{GJKW}}$ and $Shim(O_1)^{\mathsf{GJKW}}$. The game $Shim(O_0)^{\mathsf{GJKW}}$ is similar to $Game_0^{\mathsf{GJKW}}$, except the procedure *Sign* in $Game_0^{\mathsf{GJKW}}$ is slightly different (see Figure 5.7 to see the difference). The $Game_0^{\mathsf{GJKW}}(\mathcal{F})$ computes the signature after the *computed-relevant* oracle, whereas in the $Shim(O_0)^{\mathsf{GJKW}}$, the message is searched in the map.

In EasyCrypt, the game $Shim(O_0)^{\mathsf{GJKW}}$ is defined in Listing 5.12, where the game takes *Oracles* as a parameter (see Listing 5.13).

Listing 5.12: Shim takes module Oracles as a parameter

```
local module Shim (O : Oracles) = {
  var sk : zmod
  var qs : msg fset
  var ms : (msg, (group * (zmod * zmod) * bool)) fmap
  var  m : msg
  var  y : group
  var  c : zmod
  var  s : zmod
  var  b : bool
  var  h : group
  var bm : bool
  var gb : group
  module O' = {
    proc h(m, b) = {
      var r;
      O.compute_related(m);
      r <@ O.h(m, b);
      return r;
    }
    proc g(h, y, a, b, m) = {
      var r;
      O.compute_related(m);
      r <@ O.g(h, y, a, b, m);
      return r;
    }
```

```
    proc sign(m) = {
      O.compute_related(m);
      qs <- qs `|` fset1 m;
      return oget ms.[m];
    }
  }
  proc run() = {
    var pk, sig, pi, c';
    O.init();
    Bt.RO.init();
    Rt.RO.init();
    qs <- fset0;
    ms <- empty;
    gb <$ dgp;
    sk <$ dZp;
    pk <- gg ^ sk;
    (m, sig) <@ A(O').forge(pk);
    (y, pi, b) <- sig;
    (c, s) <- pi;
    h <@ O.h(m, b);
    c' <@ O.g(h, y, gg ^ s * pk ^ -c, h ^ s * y ^ -c, m);
    bm <@ Bt.RO.get(m);
    return c = c' /\ !m \in qs
            /\ y = h ^ sk /\ bm <> b;
  }
  proc distinguish = run (* Fit the RO distinguisher interface *)
}.
```

Listing 5.13: Shim takes module Oracles as a parameter

```
module type Oracles  = {
  proc init() : unit
  proc h(m : msg, b : bool) : group
  proc g(h : group, y : group, a : group, b : group, m : msg) : zmod
  proc compute_related(m : msg) : unit
}.
```

The pen and paper proof of *compute-relevant* 5.1 queries has conditional statements, making states stateless instead of stateful. By removing the *if* statements in the machine-checked formalisation, intermediate games are easier to implement in EasyCrypt.

Next, the embedding and simulation step occurs in the game $Shim(O_1)^{\mathsf{GJKW}}$, as mentioned earlier.

Step 2: Embedding and Simulation ($Shim(O_1)^{\mathbf{GJKW}}(\mathcal{H}, \mathcal{G}, CompRel)$)

The steps of embedding and simulation occur simultaneously for GJKW's proof, unlike EDL and CM's proofs which occurred separately. In the machine-checked formalisation

$Game_0^{\mathsf{GJKW}}(\mathcal{H},\mathcal{G},\mathcal{S})$

$b_m \leftarrow \text{false}$
$\mathsf{sk} \leftarrow_\$ \mathbb{F}_q$
$\mathsf{b} \leftarrow_\$ \{0,1\}$
$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
$y \leftarrow h^{\mathsf{sk}}$
$(\mathsf{m},\sigma) \leftarrow \mathcal{A}(O).forge(\mathsf{pk})$
$\sigma \leftarrow (y,\pi,b)$
$\pi \leftarrow (c,s)$
$\mathsf{h} \leftarrow \mathcal{H}.get(m,b)$
$\mathsf{A} \leftarrow g^s \mathsf{pk}^{-c}$
$\mathsf{B} \leftarrow \mathsf{h}^s y^{-c}$
$\mathsf{c\prime} \leftarrow \mathcal{G}.get(\mathsf{h},y,\mathsf{A},\mathsf{B},m)$
$b_m \leftarrow B.get(\mathsf{m})$
$\mathsf{win} \leftarrow c = \mathsf{c\prime} \wedge \neg \mathsf{m} \in \mathcal{Q}_{\mathcal{S}}$

$Shim(O_0)^{\mathsf{GJKW}}(\mathcal{H},\mathcal{G},\mathcal{S})$

$ms : (\mathsf{m},(\sigma = (y,\pi = (c,s),b)))fmap$
$b_m \leftarrow \text{false}$
$\mathsf{sk} \leftarrow_\$ \mathbb{F}_q;$
$\mathsf{b} \leftarrow_\$ \{0,1\}$
$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
$\mathsf{g_b} \leftarrow g^d$
$(m,\sigma) \leftarrow \mathcal{A}(O\prime).forge(\mathsf{pk})$
$\sigma \leftarrow (y,\pi,b)$
$\pi \leftarrow (c,s)$
$\mathsf{h} \leftarrow O.h(m,b)$
$\mathsf{A} \leftarrow g^s \mathsf{pk}^{-c}$
$\mathsf{B} \leftarrow \mathsf{h}^s y^{-c}$
$\mathsf{c\prime} \leftarrow O.g(\mathsf{h},y,\mathsf{A},\mathsf{B},m)$
$b_m \leftarrow Bt.RO.get(\mathsf{m})$
$\mathsf{win} \leftarrow c = \mathsf{c\prime} \wedge \neg \tilde{\mathsf{m}} \in \mathcal{Q}_{\mathcal{S}}$
$\wedge (y = h^{\mathsf{sk}}) \wedge (b_m \neq b)$

O

Oracle $\mathcal{H}$
  compute-relevant queries
  $h \leftarrow \mathcal{H}(\mathsf{m},b)$
Oracle $\mathcal{G}$
  compute-relevant queries
  $c \leftarrow \mathcal{G}(h,y,A,B,\mathsf{m})$
Oracle Sign
  compute-relevant queries
  $b \leftarrow B.get(\mathsf{m})$
  $r \leftarrow R.get(\mathsf{m})$
  $h \leftarrow H.get(\mathsf{m},b)$
  $c \leftarrow G.get(h,h^{\mathsf{sk}},g^r,h^r,\mathsf{m})$
  return $(h^{\mathsf{sk}},(c,(c\cdot\mathsf{sk}+r)),b)$

$O\prime$

Oracle $\mathcal{H}$
  compute-relevant queries
  $h \leftarrow \mathcal{H}(\mathsf{m},b)$
Oracle $\mathcal{G}$
  compute-relevant queries
  $c \leftarrow \mathcal{G}(h,y,A,B,\mathsf{m})$
Oracle Sign
  compute-relevant queries
  return $ms.[\mathsf{m}]$

Figure 5.7: The difference b/w $Game_0^{\mathsf{GJKW}}$ and $Shim(O_0)^{\mathsf{GJKW}}$

of GJKW, there is no failure event to account for now, and the embedding step becomes failure-event-free. The two steps occur in the game $Shim(O_1)^{\mathsf{GJKW}}(\mathcal{F})$ (as shown in r.h.s. of the Figure 5.8). The CDH challenge is embedded into the responses given to $\mathcal{H}$ queries, and then the *compute-relevant* queries do the simulation.

[$Shim(O_1)^{\mathbf{GJKW}}(\mathcal{H}, \mathcal{G}, CompRel)$**]**

The Figure 5.8 shows that the red texts represent the changes between them in two consecutive games, and the following section describes how they are equivalent.

The game $Shim(O_1)^{\mathsf{GJKW}}$ modifies the oracle $\mathcal{H}$ to embed the CDH challenge in the log of forger queries. The oracle $\mathcal{H}$ is revealed to the forger, which then can compute its output as $g_b g^d$ for uniformly sampled d, which can be kept alongside the oracle's response. Also, logging d in $\mathcal{H}$ does not affect the adversary's viewpoint as computing both $g_b g^d$and g^d generate the uniform distribution over $\mathbb{G}$ when d is uniform in $\mathbb{F}_q$ and d is not used. Note that the failure event *bad* was already accounted for in the previous game, which occurred when the query to oracle $\mathcal{G}$ made by the signing oracle was not fresh. So, here in this game, the oracle $\mathcal{G}$ is now set to value c.

The simulation step happens in the *compute-relevant*, now in the oracle Sign. The *compute-relevant* simulates the proof of discrete logarithm equality, which relies on the property of zero-knowledge without using the secret key. The modification of the oracle Sign, which samples a fresh value of h, does not change the behaviour whether it appeared in $\mathcal{H}$ or in the signing query, and the adversary can not distinguish this change. This is because both oracles are already programmed.

In EasyCrypt, the oracles of two games, $Shim(O_0)^{\mathsf{GJKW}}$ and $Shim(O_1)^{\mathsf{GJKW}}$, are proved with *equivalences* of lemmas, keeping the above statements in mind. The above statements capture the behaviour of the two games and are written as invariants in the pre and post conditions of the equivalence lemmas. The lemmas of oracles for equivalences are not delved into discussions. Thus, the probability of these two games can be written as: $\Pr[Shim(O_0).\mathsf{run}()@\ \&\mathrm{m}: \mathrm{res}] = \Pr[Shim(O1).\mathsf{run}()@\ \&\mathrm{m}: \mathrm{res}]$

Step 3: Reduction

Oracles were now simulated without using the secret key, and a CDH challenge was embedded. At this final stage, if the forger $\mathcal{F}$ wins the $(Shim(O_1)^{\mathsf{GJKW}})$ and shows a successful forgery $(m, (\sigma = y, (\pi = (c, s))$ then the adversary from Figure 5.3 succeeds in solving its given instance.

$Shim(O_0)^{\mathsf{GJKW}}_{\mathcal{H},\mathcal{G},\mathcal{S}}$

$ms : (\mathsf{m}, (\sigma = (y, \pi = (c, s), b)))fmap$

1 : $b_m \leftarrow$ false ;$\mathsf{sk} \leftarrow\!\!\$\ \mathbb{F}_q$;
2 : $\mathsf{b} \leftarrow\!\!\$\ \{0,1\}$
3 : $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
4 : $\mathsf{g_b} \leftarrow g^d$
5 : $(m, \sigma) \leftarrow \mathcal{A}(O\prime).forge(\mathsf{pk})here?$
6 : $\sigma \leftarrow (y, \pi, b)$
7 : $\pi \leftarrow (c, s)$
8 : $\mathsf{h} \leftarrow O.h(m, b)$
9 : $\mathsf{A} \leftarrow g^s \mathsf{pk}^{-c}$
10 : $\mathsf{B} \leftarrow \mathsf{h}^s y^{-c}$
11 : $\mathsf{c}\prime \leftarrow O.g(\mathsf{h}, y, \mathsf{A}, \mathsf{B}, m)$
12 : $b_m \leftarrow Bt.RO.get(\mathsf{m})$
13 : $\mathsf{win} \leftarrow c = \mathsf{c}\prime \wedge \neg\tilde{\mathsf{m}} \in \mathcal{Q}_{\mathcal{S}}$
$\wedge\ (y = h^{\mathsf{sk}}) \wedge (b_m \neq b)$

$Shim(O_1)^{\mathsf{GJKW}}(\mathcal{H}, \mathcal{G}, CompRel)$

$h_m : (\mathsf{m} \cdot \mathrm{bool}, \text{h-resp})fmap$
$gm : ((group, group, group, group, msg), zmod)fmap$

1 : $b_m \leftarrow$ false
2 : $b \leftarrow Bt.RO.get(m)$
3 : $\mathsf{pk} \leftarrow g^{Shim.\mathsf{sk}}$
4 : $\mathsf{g_b} \leftarrow g^d$??
5 : $(m, \sigma) \leftarrow \mathcal{A}(O\prime).forge(\mathsf{pk})here?$
6 : $\sigma \leftarrow (y, \pi, b)$
7 : $\pi \leftarrow (c, s)$
8 : Wrapper $O\prime$
9 : $\mathsf{h} \leftarrow g^d$? do we need this now?
10 : $\mathsf{y} \leftarrow \mathsf{pk}^d$
11 : $\mathsf{A} \leftarrow g^s \mathsf{pk}^{-c}$
12 : $\mathsf{B} \leftarrow \mathsf{h}^s y^{-c}$
13 : $\mathsf{c}\prime \leftarrow O.g(\mathsf{h}, y, \mathsf{A}, \mathsf{B}, m)$
14 : $\mathsf{win} \leftarrow c = \mathsf{c}\prime \wedge \neg\tilde{\mathsf{m}} \in \mathcal{Q}_{\mathcal{S}}$
$\wedge\ (y = h^{\mathsf{sk}}) \wedge (b_m \neq b)$

$O\prime$

1 : Oracle $\mathcal{H}$
compute-related queries
$h \leftarrow \mathcal{H}(\mathsf{m}, b)$

2 : Oracle $\mathcal{G}$
compute-related queries
$c \leftarrow \mathcal{G}(h, y, A, B, \mathsf{m})$

3 : Oracle Sign
compute-related queries
return $ms.[\mathsf{m}]$

$O\prime$

1 : Oracle $\mathcal{H}(x)$
compute-related queries
$y \leftarrow\!\!\$\ \mathbb{F}_q$
if $(x \notin hm)$
$hm[x] \leftarrow h$
return $hm[x]$

2 : Oracle $\mathcal{G}(x)$
compute-related queries
$y \leftarrow\!\!\$\ \mathbb{F}_q$
if $(x \notin gm)$
$gm[x] \leftarrow c$
return $gm[x]$

3 : compute-related (CompRel)
if $m \notin$ Shim.ms
$b \leftarrow Bt.RO.get(m)$
$s \leftarrow Rt.RO.get(m)$
$\mathsf{d} \leftarrow\!\!\$\ \mathbb{F}_q$
$h_m.[(m, b)] \leftarrow$ SHQuery d
$h \leftarrow g^d$
$c \leftarrow\!\!\$\ \mathbb{F}_q$
$\mathsf{pk} \leftarrow g^{Shim.ms}$
$y \leftarrow \mathsf{pk}^d$
$\mathsf{A} \leftarrow g^s \mathsf{pk}^{-c}$
$\mathsf{B} \leftarrow \mathsf{h}^s y^{-c}$
$\mathsf{c}\prime \leftarrow O.g(\mathsf{h}, y, \mathsf{A}, \mathsf{B}, m)$
$Shim.ms.[\mathsf{m}] \leftarrow (y, (c, s), b)$

Figure 5.8: The difference b/w $Shim(O_0)^{\mathsf{GJKW}}$ and $Shim(O_1)^{\mathsf{GJKW}}$

Then the following lemma states that the probability that the adversary wins the game $Shim(O_1)^{\mathsf{GJKW}}$) is bounded by the probability of the simulation that solves the CDH challenge.

$$\Pr[Shim(01).run()@\ \&m{:}res] \leq \Pr[\mathsf{CDH}(\text{Simulator(A).run() @ \&m: res})]\,.$$

Finally, one can bound the probability of the EUF-CMA of GJKW is bounded by the sum of the probabilities of lemmas 16 and 17.

$$\begin{aligned}\Pr[\mathsf{EUF\text{-}CMA}(\mathcal{H},\mathcal{G},\mathsf{GJKW},\mathcal{A}).\text{run() @ \&m: res}] \\ \leq \frac{(q_{\mathcal{G}}+1)}{p} \\ + 2\cdot\Pr[\mathsf{CDH}(Simulator(\mathcal{A})).\text{run() @ \&m: res}]\,.\end{aligned}$$

5.6 Reflecting on the proof pattern

From the formalisation of EDL and CM, we identified a proof pattern or a proof structure which we claim as a contribution. The proof pattern is general enough to be applied to similar signature schemes. The research investigated that there is a similar signature scheme by Goh, Jarecki, Katz and Wang (GJKW) [6], whose security is also based on a tight discrete logarithm. So, this chapter aimed to formalise the GJKW's scheme by applying the same proof pattern and technique we used for EDL and CM schemes. To our knowledge, this is the first machine-checked formalisation of GJKW's scheme using the EasyCrypt proof assistant. However, the formalisation of GJKW's scheme required some work before applying the proof pattern.

For the formal verification of the scheme, the proof goes with two sections of simplifications with events $(res \wedge \hat{y} \neq \hat{h}^{\mathsf{sk}})$ and $(res \wedge \hat{y} = \hat{h}^{\mathsf{sk}})$ at points 1 and 2, respectively, and both simplifications help eliminate other unnecessary events and cases. The first simplification follows that if the forger $\mathcal{F}$ outputs a valid forgery where $\hat{y} \neq \hat{h}^{\mathsf{sk}}$, $Game_0$ is bounded by the *bad* event in $Game_{bad}$, where it encounters the failure event. The probability of occurring such an event is at most $\frac{(q_{\mathcal{G}}+1)}{p}$. Unlike the EDL and CM schemes, the GJKW scheme only encounters one failure event, and this is because, in the first simplification, it ignores forgeries where forgeries rely on unsoundness proof of non-interactive zero-knowledge (NIZK) for now. In addition, the transition from $Game_0$ to $Game_{bad}$ allows to bound its probability by simply counting the number of

queries and identifying a neat bound on the adversary. In the second simplification of GJKW's proof, the forger $\mathcal{F}$ outputs a valid forgery where $\hat{y} = \hat{h}^{\mathsf{sk}}$, then the proof further splits into sections A and B. Before heading with further splits, one should add the probability of events$(res \wedge \hat{y} \neq \hat{h}^{\mathsf{sk}})$ and $(res \wedge \hat{y} = \hat{h}^{\mathsf{sk}})$, bounded by the probability of event res in $Game_0$. The second simplification removes the cases where the adversary guesses the value of $\hat{b}$, which is used for her forgery. In contrast to the pen-and-paper proof of the GJKW's scheme, the machine-checked proof has the value of $b_{\hat{m}}$, which is not independent of the view of the forger, if she made any query to the random oracle on message $\hat{m}$. In response to $\mathcal{H}$ queries with $\hat{m}$ as a message, it samples $b_{\hat{m}}$, branches the value sampled before, and the value is returned to the adversary from the oracle. Therefore, it depends on the value sampled for $b_{\hat{m}}$. So, after bounding the event, the forger $\mathcal{F}$ on point 2 had not requested a signature on the message earlier, and the value of $b_{\hat{m}}$ can or cannot be viewed by the $\mathcal{F}$, which gives them further splits of A and B. If the value of $b_{\hat{m}}$ is independent of the view of the forger, then the probability for event $(res \wedge \hat{y} = \hat{h}^{\mathsf{sk}} \wedge \hat{b} = b_{\hat{m}})$ is at most $1/2$. When the value is $\hat{b} \neq b_{\hat{m}}$ with event $(res \wedge \hat{y} = \hat{h}^{\mathsf{sk}} \wedge \hat{b} \neq b_{\hat{m}})$, the proof of GJKW utilises the **four-step proof structure** (refactor, embedding, simulation and reduction) of EDL and CM schemes. Unlike the EDL and CM schemes, the first step of refactorisation in GJKW has complex proof steps. However, later, it helps to have significantly more straightforward steps for embedding and simulation. The *compute-relevant* queries from the pen and paper proof of GJKW are introduced in the refactorisation step. The embedding and simulation steps occur at the same time in this proof, where the CDH challenge is embedded into random oracle H and simulates the zero-knowledge proof. Finally, the reduction step shows that if the forger wins the game, then the simulator succeeds in breaking the CDH challenge. Unlike the formal verification of the prior schemes, the GJKW scheme had only one failure event in the first simplification section. The reason for bounding the failure event earlier is that the rest of the proof becomes simpler to bound and reasoning outside evolves slightly easier. During the formal development of the scheme, the first simplification removes the if-and-else statements from the games. The signing oracle of $Game_0$ makes the sampling operation for b and r instead of sampling in distributions. So, instead of sampling a fresh value every time one gets the same message m, one can retrieve the value sampled before. This way, the proof first de-randomises if the message is signed, which returns all signatures. Here, it ensures that b and r are produced pseudorandomly from m, and the rest is deterministic computation. Despite the fact that the scheme was thought to be very similar to the EDL and CM schemes, pushing the game transformations to a successful conclusion took an immense amount of work, and the effort involved in its ideation was not scalable as it took eight months to produce almost two thousand lines of code in EasyCrypt.

5.7 Lessons Learned

During the formal development of the GJKW's scheme in EasyCrypt, the initial machine-checked formalisation suffered a little with complicated *if-and-else* statements in the pen and paper proof. Goh et al. [6] introduced additional steps called *relevant queries* before answering the query. The adversary performs the steps whenever the first relevant query for some message is made. For a message m, the queries to $\mathcal{H}(\mathsf{m}, \star)$, $\mathcal{G}(\star, \star, \star, \star, \mathsf{m})$, and $Sign_{\mathsf{sk}}(\mathsf{m})$ are relevant for the message m. In the *relevant query* steps, firstly, the random oracle produces b_m, which maps m to b_m. Secondly, the internal map is modified whenever the queries to $\mathcal{H}$ are made to (m, b_m); it outputs the $h_m = \mathcal{H}(\mathsf{m}, b_m) = g^{\gamma_m}$, where $\gamma_m \in \mathbb{Z}_q$ is A, and stores $\mathsf{m}, b_m, \gamma_m$. Lastly, instead of computing the $y = h^{\mathsf{sk}}$, it computes $y = \mathsf{pk}^{\gamma_m}$ and simulates non-interactive zero-knowledge proof. The values c and s are distributed by the honest verifier zero-knowledge property of the proof system and compute $A = g^s pk^{-c}$, and $B = h_m^{\mathsf{sk}} \cdot y^{-c}$ ($y = \mathsf{pk}^{\gamma_m} = (g^{\mathsf{sk}})^{\gamma_m} = h_m^{\mathsf{sk}}$). Setting $\sigma_m = (y, c, s, b_m)$, defining $\mathcal{H}(h_m, y, A, B, \mathsf{m}) = c$ and stores $(sig, \mathsf{m}, \sigma_m)$. So, if the relevant query is asked, then the adversary always has a valid signature for any message for which a relevant query has been asked.

Immediate goals include further unifying all the sections and sub-sections with sub-steps from the Diagram 5.4 and merging them as a formal Shim.

Chapter 6

Conclusion

Modern cryptography has extended to elaborated functionalities that can only be achieved by being implemented in the cryptographic systems interacting with other constructions. As a result of functionality expansions, the errors in the cryptographic proofs are mostly oversight; in fact, the proofs have become challenging to check due to being lengthy and complicated. The key issue is not just to check whether the cryptographic proofs are error-free but also to verify if the cryptographic systems function well; therefore, it has become one of the main challenges many cryptographers face. In 2005, Halevi [7] advocated designing a tool-supported framework to help cryptographers build and verify their cryptographic proofs. So this led to the development of machine-checked tools which ensure the correctness of the proofs and provide confidence and high assurance in the security proofs.

This thesis utilises the EasyCrypt framework to report on the first machine-checked formal verification of three digital signature schemes, i.e., Goh and Jarecki [16] (EDL), Chevallier-Mames [17] (CM) and Goh, Jarecki, Katz and Wang [6] (GJKW).[1] The security of all three signature schemes is based on tight reductions to the CDH problem. The machine-checked proofs of the schemes are achieved using the code-based game-playing technique that Shoup [15] advocated for verification. Chapter 4 explained in detail that achieving the machine-checked proofs for both EDL and CM scheme; both schemes were reformulated and showed that the statements of the security reductions are concrete and constructive. The formalisation effort of EDL and CM schemes sought similarities and factors them out. The EDL and CM schemes were formally defined with three intermediate games, which bridge the gap between the EUF-CMA game and the CDH game. The thesis gave a deep insight into these intermediate games as an

[1] EDL: https://gitlab.com/fdupress/edl/-/tree/master/edl
CM: https://gitlab.com/fdupress/edl/-/tree/master/cm
GJKW: https://gitlab.com/fdupress/edl/-/tree/master/gjkw

instance of a common game called *Shim*. The formalisation showed that any two successive games differ only in the oracles provided to the *Shim*. This research provides the first comprehensive use of the game *Shim*, which allowed only to focus reasoning on the parts of the proofs that change between the successive games and helps reduce the boilerplate proof amount. The insight gained from the formalisation of both schemes was further generalised and outlined in a generic three-step proof structure (embedding, simulation and reduction). In the generic three-step proof, the embedding relied on the underlying challenge into random oracle answers; the simulation relied on the zero-knowledge proof, and the reduction relied on the soundness of the zero-knowledge proof and underlying CDH problem. The proof extracted in such a pattern could be reusable and can be applied to similar signature schemes whose security is based on a tight discrete logarithm.

Moreover, the EDL and CM schemes relied on freshness to finish the reduction; however, the GJKW scheme does not rely on freshness to push the reduction. The latter case is because when an adversary makes a query to the $\mathcal{H}$ oracle with a message $\hat{m}$, the value is inserted in the map as if the signing oracle had queried it with a message $\hat{m}$. So, essentially, the adversary can never request the signatures, so it is unlikely that the adversary has guessed it. The research contributes to the understanding of formal verification of digital signature schemes based on reducing a tight discrete logarithm.

Goh and Jarecki [16] present the EDL scheme, which is existentially unforgeable under a chosen-message attack (EUF-CMA), in the random oracle model. The scheme is constructed over a group where the CDH problem is hard. Consequently, the scheme gives a tighter reduction than most other discrete logarithm-based signature schemes that support shorter signatures. The reason for choosing this scheme was to start the PhD research with some simple scheme for the formalisation, using the EasyCrypt. Another reason for choosing the scheme was that the security proof of the scheme does not make use of the Forking Lemma [1], and EasyCrypt is unable to use the lemma yet. In fact, the scheme holds a Schnorr proof and is used as a zero-knowledge proof of discrete logarithm equality rather than as a proof of knowledge. So, for those reasons, the research began to formalise the first machine-checked proofs of Goh and Jarecki's scheme [16] (EDL) in EasyCrypt proof assistant. While developing the machine-checked proofs, the research found some minor mistakes in the original proof of Goh and Jarecki's scheme. They overlooked the mistake, leading the thesis to some imprecise security bounds. One can find the details in the section 4.3. Instead, the thesis proved a very close bound, similar to the EDL scheme given by Chevallier-Mames [17], with minor changes to address the formal details.

The second machine-checked formal verification of the scheme was by Chevallier-Mames [17], and the security reduction is also tightly related to the CDH problem.

Chevallier-Mames [17] claims that his signature is efficient, and the results are smaller than the EDL signature scheme. He also corrected the EDL scheme and presented it with his scheme [17]. The insights gained from the first formalism assisted in formalising the second scheme, which was the proof of the Chevallier-Mames (CM) in EasyCrypt. The difference in this formalisation is that the message is not included in the random oracle query, which served as the second base for the proof of discrete logarithm equality. Instead, the message is included as input to the challenge-generating random oracle query, similar to the standard Schnorr signatures. So this difference supports the security of the scheme but without additional randomness. The thesis shows that this scheme has an almost similar underlying proof structure to machine-checked proofs of the EDL scheme.

For the formal development of the CM scheme, the EDL proof was replayed over the CM scheme. The formal Shim was reused with minor tweaks and showed that proof differs only in the probability bounding and reduction step. The thesis shows that the proof carries over similar objects as the similarities were sought out and attempted to factor the proofs out. There is an exception in the reduction step, which had an additional case to be considered. The formalisation of both schemes allows identifying the proof principles that could be adapted to other existing or new signature schemes to obtain new proofs of tight security.

The thesis has argued earlier that insight gained from the generalisation of EDL and CM schemes has potential. The thesis claimed that the proof structure extracted from the generalisation could be applied to similar signature schemes of tight security and a similar signature is the scheme of Goh, Jarecki, Katz and Wang [6]. So, this research formalised the third scheme, which is the scheme of Goh et al. [6] (GJKW) using the EasyCrypt. The machine-checked proofs of the scheme have some additional steps, and that is because the pen and paper proofs of the scheme are slightly different from the EDL and CM schemes, but the GJKW delivers the same security, that is, the tight discrete logarithm-based reductions. The main difference in the pen and paper proof of the GJKW's scheme is that they have shortened the size of the scheme to improve the EDL signature scheme and removed the random salt. For that, the signer has a hidden, random *selector bit*, called b_m related to each message the signer signs. To improve the efficiency, Goh et al.[6] suggested avoiding having the signer maintain the record of all (m, b_m) pairs. One could have the signer producing b_m and the value r as *deterministic pseudorandom* functions of the message m. The result is the same way the signature is produced each time a particular message is signed. The pen and paper proof of the GJKW's scheme has additional steps called *compute-relevant* queries, where the adversary performs three steps whenever the first relevant query for the message is made. After spending a lot of effort and time, the thesis simplified the proofs into

two sections for the machine-checked security proofs of the GJKW scheme. The purpose of the two sections of the simplifications is to eliminate unnecessary events and cases. During the formal verification of the scheme, the security proofs also benefited from the proof steps structure (refactor, embedding, simulation and reduction) of the EDL and CM schemes. Despite the fact that the scheme was thought to be very similar to the EDL and CM schemes, pushing the game transformations to a successful conclusion took an immense amount of work, and the effort involved in its ideation was not scalable.

The formalisation of all three schemes had its challenges, and transforming the formalism into EasyCrypt was another work. The initial version of the formal proofs for EDL and CM schemes took longer than GJKW, but once the knowledge was gained for the machine-checked proofs, it became more manageable but still challenging to achieve a successful conclusion.

6.1 Future Work

At the beginning of this research, after formalising EDL and CM schemes, it was suggested that a more immediate extension would be to consider the schemes by Katz and Wang [63] and Goh et al. [6] (GJKW), which also benefit from tight discrete logarithm-based reductions. The research immediately started to formalise the later scheme. As it was to extend the proof schema and leverage the scheme's technique of unpredictability as opposed to full randomness—of the second base h would extend the class of signature schemes whose security could be machine-checked at a minimal cost. The formalisation of the GJKW scheme proves that similar techniques can be reused.

Immediate goals include unifying the GJKW's scheme, which has sections and subsections from bounding the failure event to computing relevant query steps and the formal proof structure (refactoring, embedding, simulating and reduction). The game transformations are laid out already for the scheme, and it would be advantageous to develop a common game, just like Shim, which we had for the EDL and CM schemes. Just like those two schemes, the intermediate games can be written as instances of a common game. Then one can easily have proof-steps to introduce a general approach to prettify this scheme. Further work in this area helps extract the abstract game transformations and security statements required to obtain generic proofs from the machine-checked proofs of all three signature schemes and be able to instantiate them all. Extracting such abstracts is a gap in EasyCrypt proof assistant. The new insights gained from this research may assist in filling this gap in EasyCrypt proof assistant. The comparisons of the security proofs with EasyCrypt proofs are guidelines and manuals to start the formalisation of similar signature schemes. The overview of the proofs with

visual representations guides the naive user to formalise the complicated cryptographic constructions. As mentioned in the chapter, simplifying and dividing into subsections can help formalise complicated and complex signature schemes.

In the pen and paper proof of Goh et al. [6], they presented two schemes whose securities are tightly related to the Diffie-Hellman problems. Their first scheme was based on the hardness of the Computational Diffie-Hellman (CDH) problem, and the second was based on the hardness of the Decisional Diffie-Hellman (DDH) problem. It was impossible to formalise the later scheme due to time constraints; therefore, the scheme based on DDH is untouched. Further research could usefully explore how to formalise the second scheme of GJKW, using the EasyCrypt. The thesis has provided a deeper insight into the machine-checked formalisation of the discrete-based signature schemes equipped with tight proofs; therefore, it would require less effort to formalise the second scheme (based on DDH) of GJKW using the EasyCrypt proof assistant.

One of the more significant findings to emerge from all the machine-checked proofs are that this research identified the same proof schema, which is applied three times. Although the pen and paper proofs of the three schemes (EDL, CM and GJKW) are all different, the principles that make them secure are the same. The formal verification of all three schemes benefited from the approach of the proof schema. This approach will prove helpful in expanding the understanding of how formal verification of related signature schemes benefits from tight discrete logarithm-based reductions.

The original EDL scheme was proposed by Chaum and Pedersen [18] for use in a tamper-resistant wallet. Signature computation was meant to be split, with the host producing the value h and a proof of its well-formedness, and the wallet produce the rest of the signature. Direct Anonymous Attestation schemes [121] are very similar in their structure. They are widely deployed in Trusted Platform Modules, and ECC-DAA [122] is now part of a FIDO alliance standard (as ECDAA). This makes DAA schemes excellent targets for formalization on the back of the results presented here.

This research has discussed why the above schemes are formalised in EasyCrypt. The thesis has provided a deeper insight into the machine-checked formal verification of the digital signature schemes based on the reduction of the discrete logarithm. The process of learning and adopting the machine-checking proofs was challenging in the first 12-18 months of the research work. However, the joy of proving the goals in EasyCrypt was undefinable. A slight limitation of this research is not having the full knowledge of cryptography, which is why this research did not dig deeper into the cryptography to develop new verified signature schemes in EasyCrypt. Nonetheless, the main contribution was learning and being skilful in the machine-checked formal verification field, using the proof assistant. Further research might explore using different proof assistants and

gaining knowledge about being a cryptographer as these two aspects go hand in hand.

The generalisation discussed in chapter 4 has noted that the results and associated proof artefacts can open new directions to push formalisation into. As the security proofs of both EDL and CM schemes have similar objectives and the formalisation effort of this research sought out their similarities and factored them out, which can be reusable or some parts can be replayed to other similar constructions. The proof pattern extracted from the generalisation could lay a building block for the formalisation effort.

Appendix A

The following figure displays the original pen and paper definitions of EDL [16], CM [17] and GJKW [6] signature schemes.

EDL Scheme	CM Scheme	GJKW Scheme
$\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$	$\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$	$\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$
$y \leftarrow g^{\mathsf{sk}}$	$y \leftarrow g^{\mathsf{sk}}$	$y_1 \leftarrow g^{\mathsf{sk}}$
$r \leftarrow_\$ \{0,1\}^{n_r}$	$k \leftarrow_\$ \mathbb{Z}_q$	b-bit
$h \leftarrow \mathcal{H}(\mathsf{m}, r)$	$h \leftarrow \mathcal{H}(u)$	$h \leftarrow \mathcal{H}'(b, m)$
$z \leftarrow h^{\mathsf{sk}}$	$z \leftarrow h^{\mathsf{sk}}$	$y_2 \leftarrow h^{\mathsf{sk}}$
$k \leftarrow_\$ \mathbb{Z}_q$		$r \leftarrow_\$ \mathbb{Z}_q$
$u \leftarrow g^k$	$u \leftarrow g^k$	$A \leftarrow g^r$
$v \leftarrow h^k$	$v \leftarrow h^k$	$B \leftarrow h^r$
$c \leftarrow \mathcal{H}'(g, h, y, z, u, v)$	$c \leftarrow \mathcal{G}(m, g, h, y, z, u, v)$	$c \leftarrow \mathcal{H}(h, y_2, A, B, m)$
$s \leftarrow k + c \cdot \mathsf{sk}$	$s \leftarrow k + c \cdot \mathsf{sk}$	$s \leftarrow c \cdot \mathsf{sk} + r$
		$\pi \leftarrow (c, s)$
$\sigma \leftarrow (z, r, s, c)$	$\sigma \leftarrow (z, s, c)$	$\sigma \leftarrow (y_2, \pi, b)$

Figure A.1: EDL, CM and GJKW schemes from their original paper

Bibliography

[1] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *Advances in Cryptology – EUROCRYPT'96*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Saragossa, Spain: Springer, Heidelberg, Germany, May 12–16, 1996, pp. 387–398.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology – CRYPTO'84*, ser. Lecture Notes in Computer Science, G. R. Blakley and D. Chaum, Eds., vol. 196. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 1984, pp. 10–18.

[3] W. M. Daley and R. G. Kammer, "Digital signature standard (dss)," BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, Tech. Rep., 2000.

[4] C. F. Kerry and P. D. Gallagher, "Digital signature standard (dss)," *FIPS PUB*, pp. 186–4, 2013.

[5] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[6] E.-J. Goh, S. Jarecki, J. Katz, and N. Wang, "Efficient signature schemes with tight reductions to the Diffie-Hellman problems," *Journal of Cryptology*, vol. 20, no. 4, pp. 493–514, Oct. 2007.

[7] S. Halevi, "A plausible approach to computer-aided cryptographic proofs," Cryptology ePrint Archive, Report 2005/181, 2005, http://eprint.iacr.org/2005/181.

[8] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 14–18, 2011, pp. 71–90.

[9] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, "Easycrypt: A tutorial," in *Foundations of security analysis and design vii*. Springer, 2013, pp. 146–166.

[10] G. Barthe, B. Grégoire, and S. Zanella-Béguelin, "Formal certification of code-based cryptographic proofs," *SIGPLAN Not.*, vol. 44, no. 1, p. 90–101, Jan. 2009. [Online]. Available: https://doi.org/10.1145/1594834.1480894

[11] A. Petcher and G. Morrisett, "The foundational cryptography framework," in *International Conference on Principles of Security and Trust*. Springer, 2015, pp. 53–72.

[12] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, "Easycrypt: A tutorial," *Foundations of security analysis and design vii*, pp. 146–166, 2013.

[13] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Annual Cryptology Conference*. Springer, 2011, pp. 71–90.

[14] G. Huet, G. Kahn, and C. Paulin-Mohring, "The coq proof assistant a tutorial," *Rapport Technique*, vol. 178, 1997.

[15] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, 2004, http://eprint.iacr.org/2004/332.

[16] E.-J. Goh and S. Jarecki, "A signature scheme as secure as the Diffie-Hellman problem," in *Advances in Cryptology – EUROCRYPT 2003*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 2656. Warsaw, Poland: Springer, Heidelberg, Germany, May 4–8, 2003, pp. 401–415.

[17] B. Chevallier-Mames, "An efficient CDH-based signature scheme with a tight security reduction," in *Advances in Cryptology – CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 14–18, 2005, pp. 511–526.

[18] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology – CRYPTO'92*, ser. Lecture Notes in Computer Science, E. F. Brickell, Ed., vol. 740. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 16–20, 1993, pp. 89–105.

[19] M. Jakobsson and C. Schnorr, "Efficient oblivious proofs of correct exponentiation," in *Secure Information Networks: Communications and Multimedia Security,*

IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium, 1999, pp. 71–86.

[20] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, Jan. 1991.

[21] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[22] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[23] F. Guo, W. Susilo, and Y. Mu, *Introduction to security reduction*. Springer, 2018.

[24] H. Elkamchouchi, K. Elshenawy, and H. Shaban, "Extended rsa cryptosystem and digital signature schemes in the domain of gaussian integers," in *The 8th International Conference on Communication Systems, 2002. ICCS 2002.*, vol. 1. IEEE, 2002, pp. 91–95.

[25] J. N. Bos and D. Chaum, "Provably unforgeable signatures," in *Annual International Cryptology Conference*. Springer, 1992, pp. 1–14.

[26] F. Guo and W. Susilo, "Optimal tightness for chain-based unique signatures," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 553–583.

[27] D. Bleichenbacher and U. M. Maurer, "Optimal tree-based one-time digital signature schemes," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 1996, pp. 361–374.

[28] S. Micali and L. Reyzin, "Improving the exact security of digital signature schemes," Cryptology ePrint Archive, Report 1999/020, 1999, http://eprint.iacr.org/1999/020.

[29] G. Hotz, "Console hacking 2010-ps3 epic fail," in *27th Chaos Communications Congress*, 2010.

[30] S. Goldwasser, S. Micali, and R. L. Rivest, "A" paradoxical" solution to the signature problem," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 265–284.

[31] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS 93: 1st Conference on Computer and*

Communications Security, D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds. Fairfax, Virginia, USA: ACM Press, Nov. 3–5, 1993, pp. 62–73.

[32] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469–472, 1985.

[33] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *Journal of Cryptology*, vol. 13, no. 3, pp. 361–396, Jun. 2000.

[34] M. Girault, "An identity-based identification scheme based on discrete logarithms modulo a composite number (rump session)," in *Advances in Cryptology – EUROCRYPT'90*, ser. Lecture Notes in Computer Science, I. Damgård, Ed., vol. 473. Aarhus, Denmark: Springer, Heidelberg, Germany, May 21–24, 1991, pp. 481–486.

[35] G. Poupard and J. Stern, "Security analysis of a practical "on the fly" authentication and signature generation," in *Advances in Cryptology – EUROCRYPT'98*, ser. Lecture Notes in Computer Science, K. Nyberg, Ed., vol. 1403. Espoo, Finland: Springer, Heidelberg, Germany, May 31 – Jun. 4, 1998, pp. 422–436.

[36] H. C. Van Tilborg and S. Jajodia, *Encyclopedia of cryptography and security*. Springer Science & Business Media, 2014.

[37] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[38] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[39] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, and R. Urian, "One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation," in *2017 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2017, pp. 901–920.

[40] M. Bellare, J. Kilian, and P. Rogaway, "The security of cipher block chaining," in *Advances in Cryptology – CRYPTO'94*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 839. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 21–25, 1994, pp. 341–358.

[41] H. Patel, "A pure block chain based decentralized exchange," Cryptology ePrint Archive, Report 2014/1005, 2014, http://eprint.iacr.org/2014/1005.

[42] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications (extended abstract)," in *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 2–4, 1988, pp. 103–112.

[43] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, *EasyCrypt: A Tutorial*. Cham: Springer International Publishing, 2014, pp. 146–166. [Online]. Available: https://doi.org/10.1007/978-3-319-10082-1_6

[44] F. Dupressoir and S. Zain, "Machine-checking unforgeability proofs for signature schemes with tight reductions to the computational diffie-hellman problem," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE, 2021, pp. 1–15.

[45] J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols," in *International Conference on Security in Communication Networks*. Springer, 2002, pp. 268–289.

[46] ——, "Signature schemes and anonymous credentials from bilinear maps," in *Annual international cryptology conference*. Springer, 2004, pp. 56–72.

[47] R. Cramer and I. Damgård, "New generation of secure and practical rsa-based signatures," in *Annual International Cryptology Conference*. Springer, 1996, pp. 173–185.

[48] C. Dwork and M. Naor, "An efficient existentially unforgeable signature scheme and its applications," in *Annual International Cryptology Conference*. Springer, 1994, pp. 234–246.

[49] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.

[50] M. Girault, "Self-certified public keys," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 490–497.

[51] L. C. Guillou and J.-J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," in *Workshop on the theory and application of of cryptographic techniques*. Springer, 1988, pp. 123–128.

[52] G. Poupard and J. Stern, "Security analysis of a practical "on the fly" authentication and signature generation," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 422–436.

[53] F. Zhang, R. Safavi-Naini, and W. Susilo, "An efficient signature scheme from bilinear pairings and its applications," in *International workshop on public key cryptography*. Springer, 2004, pp. 277–290.

[54] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[55] R. Cramer and V. Shoup, "Signature schemes based on the strong rsa assumption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 161–185, 2000.

[56] R. Gennaro, S. Halevi, and T. Rabin, "Secure hash-and-sign signatures without the random oracle," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 123–139.

[57] M. Bellare and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rabin," in *Advances in Cryptology – EUROCRYPT'96*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Saragossa, Spain: Springer, Heidelberg, Germany, May 12–16, 1996, pp. 399–416.

[58] D. Boneh, "The decision Diffie-Hellman problem," in *Third Algorithmic Number Theory Symposium (ANTS)*, ser. Lecture Notes in Computer Science, vol. 1423. Springer, Heidelberg, Germany, 1998, invited paper.

[59] U. M. Maurer and S. Wolf, "The diffie–hellman protocol," *Designs, Codes and Cryptography*, vol. 19, no. 2, pp. 147–171, 2000.

[60] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *Advances in Cryptology – EUROCRYPT'97*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Konstanz, Germany: Springer, Heidelberg, Germany, May 11–15, 1997, pp. 256–266.

[61] W. M. Daley and R. G. Kammer, "Digital signature standard (dss)," BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, Tech. Rep., 2000.

[62] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[63] J. Katz and N. Wang, "Efficiency improvements for signature schemes with tight security reductions," in *ACM CCS 2003: 10th Conference on Computer and Communications Security*, S. Jajodia, V. Atluri, and T. Jaeger, Eds. Washington, DC, USA: ACM Press, Oct. 27–30, 2003, pp. 155–164.

[64] A. Joux and K. Nguyen, "Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups," Cryptology ePrint Archive, Report 2001/003, 2001, http://eprint.iacr.org/2001/003.

[65] V. Shoup, "OAEP reconsidered," in *Advances in Cryptology – CRYPTO 2001*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 2139. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 19–23, 2001, pp. 239–259.

[66] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *Advances in Cryptology – EUROCRYPT'94*, ser. Lecture Notes in Computer Science, A. D. Santis, Ed., vol. 950. Perugia, Italy: Springer, Heidelberg, Germany, May 9–12, 1995, pp. 92–111.

[67] M. Bellare, "Practice-oriented provable-security (invited lecture)," in *ISW'97: 1st International Workshop on Information Security*, ser. Lecture Notes in Computer Science, E. Okamoto, G. I. Davida, and M. Mambo, Eds., vol. 1396. Tatsunokuchi, Japan: Springer, Heidelberg, Germany, Sep. 17–19, 1998, pp. 221–231.

[68] M. Bellare and P. Rogaway, "Code-based game-playing proofs and the security of triple encryption," Cryptology ePrint Archive, Report 2004/331, 2004, http://eprint.iacr.org/2004/331.

[69] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd Annual Symposium on Foundations of Computer Science*. Las Vegas, NV, USA: IEEE Computer Society Press, Oct. 14–17, 2001, pp. 136–145.

[70] U. Maurer, "Constructive cryptography - a primer (invited paper)," in *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, R. Sion, Ed., vol. 6052. Tenerife, Canary Islands, Spain: Springer, Heidelberg, Germany, Jan. 25–28, 2010, p. 1.

[71] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[72] A. C.-C. Yao, "Theory and applications of trapdoor functions (extended abstract)," in *23rd Annual Symposium on Foundations of Computer Science*. Chicago, Illinois: IEEE Computer Society Press, Nov. 3–5, 1982, pp. 80–91.

[73] J. Kilian and P. Rogaway, "How to protect DES against exhaustive key search (an analysis of DESX)," *Journal of Cryptology*, vol. 14, no. 1, pp. 17–35, Jan. 2001.

[74] J. B. Almeida, M. Barbosa, E. Bangerter, G. Barthe, S. Krenn, and S. Zanella Béguelin, "Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols," in *ACM CCS 2012: 19th Conference on Computer and*

Communications Security, T. Yu, G. Danezis, and V. D. Gligor, Eds. Raleigh, NC, USA: ACM Press, Oct. 16–18, 2012, pp. 488–500.

[75] G. Barthe, D. Pointcheval, and S. Zanella-Béguelin, "Verified security of redundancy-free encryption from rabin and RSA," Cryptology ePrint Archive, Report 2012/308, 2012, http://eprint.iacr.org/2012/308.

[76] M. Backes, G. Barthe, M. Berg, B. Grégoire, C. Kunz, M. Skoruppa, and S. Z. Béguelin, "Verified security of merkle-damgård," in *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE, 2012, pp. 354–368.

[77] B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay, "Computationally sound mechanized proofs for basic and public-key kerberos," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, 2008, pp. 87–99.

[78] M. Backes, M. Maffei, and D. Unruh, "Computationally sound verification of source code," in *ACM CCS 2010: 17th Conference on Computer and Communications Security*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. Chicago, Illinois, USA: ACM Press, Oct. 4–8, 2010, pp. 387–398.

[79] G. Barthe, E. Fagerholm, D. Fiore, J. C. Mitchell, A. Scedrov, and B. Schmidt, "Automated analysis of cryptographic assumptions in generic group models," *Journal of Cryptology*, vol. 32, no. 2, pp. 324–360, Apr. 2019.

[80] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *2006 IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society Press, May 21–24, 2006, pp. 140–154.

[81] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "CryptHOL: Game-based proofs in higher-order logic," Cryptology ePrint Archive, Report 2017/753, 2017, http://eprint.iacr.org/2017/753.

[82] G. Barthe, B. Grégoire, R. Janvier, and S. Zanella Béguelin, "Formal certification of code-based cryptographic proofs," Cryptology ePrint Archive, Report 2007/314, 2007, http://eprint.iacr.org/2007/314.

[83] D. Cadé and B. Blanchet, "From computationally-proved protocol specifications to implementations and application to ssh." *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 4, no. 1, pp. 4–31, 2013.

[84] B. Blanchet, "Symbolic and computational mechanized verification of the arinc823 avionic protocols," in *2017 IEEE 30th computer security foundations symposium (CSF)*. IEEE, 2017, pp. 68–82.

[85] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2017, pp. 435–450.

[86] A. Lochbihler, S. R. Sefidgar, D. Basin, and U. Maurer, "Formalizing constructive cryptography using crypthol," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019, pp. 152–15 214.

[87] G. Barthe, B. Grégoire, and B. Schmidt, "Automated proofs of pairing-based cryptography," in *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, I. Ray, N. Li, and C. Kruegel, Eds. Denver, CO, USA: ACM Press, Oct. 12–16, 2015, pp. 1156–1168.

[88] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella-Béguelin, "Proving the tls handshake secure (as it is)," in *Annual Cryptology Conference*. Springer, 2014, pp. 235–255.

[89] G. Barthe, F. Dupressoir, P.-A. Fouque, B. Grégoire, M. Tibouchi, and J.-C. Zapalowicz, "Making rsa–pss provably secure against non-random faults," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 206–222.

[90] R. Canetti, A. Stoughton, and M. Varia, "Easyuc: Using easycrypt to mechanize proofs of universally composable security," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019, pp. 167–16 716.

[91] V. Cortier, C. C. Dragan, F. Dupressoir, and B. Warinschi, "Machine-checked proofs for electronic voting: privacy and verifiability for belenios," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 298–312.

[92] A. Hülsing, M. Meijers, and P.-Y. Strub, "Formal verification of saber's public-key encryption scheme in easycrypt," *Cryptology ePrint Archive*, 2022.

[93] G. Barthe, B. Grégoire, J. Hsu, and P.-Y. Strub, "Coupling proofs are probabilistic product programs," *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 161–174, 2017.

[94] J.-S. Coron, "Optimal security proofs for PSS and other signature schemes," in *Advances in Cryptology – EUROCRYPT 2002*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed., vol. 2332. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 28 – May 2, 2002, pp. 272–287.

[95] C. Bader, T. Jager, Y. Li, and S. Schäge, "On the impossibility of tight cryptographic reductions," in *Advances in Cryptology – EUROCRYPT 2016, Part II*, ser. Lecture Notes in Computer Science, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Vienna, Austria: Springer, Heidelberg, Germany, May 8–12, 2016, pp. 273–304.

[96] G. Barthe, F. Dupressoir, P.-A. Fouque, B. Grégoire, M. Tibouchi, and J.-C. Zapalowicz, "Making RSA-PSS provably secure against non-random faults," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, ser. Lecture Notes in Computer Science, L. Batina and M. Robshaw, Eds., vol. 8731. Busan, South Korea: Springer, Heidelberg, Germany, Sep. 23–26, 2014, pp. 206–222.

[97] J.-S. Coron and A. Mandal, "PSS is secure against random fault attacks," in *Advances in Cryptology – ASIACRYPT 2009*, ser. Lecture Notes in Computer Science, M. Matsui, Ed., vol. 5912. Tokyo, Japan: Springer, Heidelberg, Germany, Dec. 6–10, 2009, pp. 653–666.

[98] A. El Kaafarani, S. Katsumata, and F. Pintore, "Lossy CSI-FiSh: Efficient signature scheme with tight reduction to decisional CSIDH-512," in *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, ser. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2020, pp. 157–186.

[99] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: An efficient post-quantum commutative group action," in *Advances in Cryptology – ASIACRYPT 2018, Part III*, ser. Lecture Notes in Computer Science, T. Peyrin and S. Galbraith, Eds., vol. 11274. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, Dec. 2–6, 2018, pp. 395–427.

[100] W. Beullens, T. Kleinjung, and F. Vercauteren, "CSI-FiSh: Efficient isogeny based signatures through class group computations," in *Advances in Cryptology – ASIACRYPT 2019, Part I*, ser. Lecture Notes in Computer Science, S. D. Galbraith and S. Moriai, Eds., vol. 11921. Kobe, Japan: Springer, Heidelberg, Germany, Dec. 8–12, 2019, pp. 227–247.

[101] E. Kiltz, V. Lyubashevsky, and C. Schaffner, "A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model," in *Advances in Cryptology – EUROCRYPT 2018, Part III*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10822. Tel Aviv, Israel: Springer, Heidelberg, Germany, Apr. 29 – May 3, 2018, pp. 552–586.

[102] D. Unruh, "Post-quantum verification of Fujisaki-Okamoto," in *Advances in Cryptology – ASIACRYPT 2020, Part I*, ser. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, Dec. 2020, pp. 321–352.

[103] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt, "Mind the gap: Modular machine-checked proofs of one-round key exchange protocols," in *Advances in Cryptology – EUROCRYPT 2015, Part II*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 26–30, 2015, pp. 689–718.

[104] J. B. Almeida, M. Barbosa, G. Barthe, M. Campagna, E. Cohen, B. Grégoire, V. Pereira, B. Portela, P.-Y. Strub, and S. Tasiran, "A machine-checked proof of security for AWS key management service," in *ACM CCS 2019: 26th Conference on Computer and Communications Security*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM Press, Nov. 11–15, 2019, pp. 63–78.

[105] N. Swamy, C. Hritcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J.-K. Zinzindohoué, and S. Zanella-Béguelin, "Dependent types and multi-monadic effects in F*," in *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, Jan. 2016, pp. 256–270. [Online]. Available: https://www.fstar-lang.org/papers/mumon/

[106] B. Blanchet, "A computationally sound mechanized prover for security protocols," *IEEE Trans. Dependable Secur. Comput.*, vol. 5, no. 4, pp. 193–207, 2008. [Online]. Available: https://doi.org/10.1109/TDSC.2007.1005

[107] B. Lipp, B. Blanchet, and K. Bhargavan, "A mechanised cryptographic proof of the wireguard virtual private network protocol," in *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 231–246. [Online]. Available: https://doi.org/10.1109/EuroSP.2019.00026

[108] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 22–26, 2017, pp. 483–502.

[109] B. Blanchet, "Composition theorems for CryptoVerif and application to TLS 1.3," 2018, pp. 16–30.

[110] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational

approach," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 435–450. [Online]. Available: https://doi.org/10.1109/EuroSP.2017.38

[111] B. Blanchet, "Automatically verified mechanized proof of one-encryption key exchange," 2012, pp. 325–339.

[112] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp, and D. Riepel, "Analysing the HPKE standard," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1499, 2020. [Online]. Available: https://eprint.iacr.org/2020/1499

[113] J. B. Almeida, M. Barbosa, G. Barthe, and F. Dupressoir, "Certified computer-aided cryptography: efficient provably secure machine code from high-level implementations," in *ACM CCS 2013: 20th Conference on Computer and Communications Security*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. Berlin, Germany: ACM Press, Nov. 4–8, 2013, pp. 1217–1230.

[114] ——, "Verifiable side-channel security of cryptographic implementations: Constant-time MEE-CBC," in *Fast Software Encryption – FSE 2016*, ser. Lecture Notes in Computer Science, T. Peyrin, Ed., vol. 9783. Bochum, Germany: Springer, Heidelberg, Germany, Mar. 20–23, 2016, pp. 163–184.

[115] G. Barthe, B. Grégoire, C. Jacomme, S. Kremer, and P.-Y. Strub, "Symbolic methods in computational cryptography proofs," 2019, pp. 136–151.

[116] J.-S. Coron, "On the exact security of full domain hash," in *Advances in Cryptology – CRYPTO 2000*, ser. Lecture Notes in Computer Science, M. Bellare, Ed., vol. 1880. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 20–24, 2000, pp. 229–235.

[117] D. Hofheinz and E. Kiltz, "Programmable hash functions and their applications," in *Advances in Cryptology – CRYPTO 2008*, ser. Lecture Notes in Computer Science, D. Wagner, Ed., vol. 5157. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 17–21, 2008, pp. 21–38.

[118] B. Chevallier-Mames, "A pairing-free signature scheme from correlation intractable hash function and strong diffie-hellman assumption," in *Cryptographers' Track at the RSA Conference*. Springer, 2022, pp. 26–48.

[119] M. Jakobsson and C. P. Schnorr, "Efficient oblivious proofs of correct exponentiation," in *Secure Information Networks*. Springer, 1999, pp. 71–84.

[120] B. Chevallier-Mames, "An efficient cdh-based signature scheme with a tight security reduction," in *Annual International Cryptology Conference*. Springer, 2005, pp. 511–526.

[121] E. F. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *ACM CCS 2004: 11th Conference on Computer and Communications Security*, V. Atluri, B. Pfitzmann, and P. McDaniel, Eds. Washington, DC, USA: ACM Press, Oct. 25–29, 2004, pp. 132–145.

[122] L. Chen, D. Page, and N. P. Smart, "On the design and implementation of an efficient daa scheme," in *Smart Card Research and Advanced Application*, D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 223–237.