AN ABSTRACT OF THE DISSERTATION OF

Lawrence Roy for the degree of Doctor of Philosophy in Computer Science presented on October 18, 2022.

Title: Communication-Efficient Secure Two-Party Computation From Minicrypt and OT

Abstract approved: _____

Mike Rosulek

Secure two-party computation (2PC) is the task of performing arbitrary calculations on secret inputs provided by two parties, while maintaining secrecy if at least one party is honest. 2PC has been applied to privacy-preserving record linkage and machine learning, in areas such as medicine where maintaining privacy is crucial. One of the most practical techniques for 2PC is garbled circuits, which is based entirely on Minicrypt assumptions — i.e. it uses only symmetric key cryptography. However, it requires an initial oblivious transfer (OT) phase. Most of these oblivious transfers can be generated with OT extension, a technique where a batch of a constant number of base OTs is extended into as many OTs as needed, again using only Minicrypt assumptions. These base OTs, however, require more expensive public key cryptography, accompanied by stronger assumptions. Together, these three stages form a commonly used 2PC protocol. It is computationally efficient and requires only minimal cryptographic assumptions. However, the protocol requires significant communication, which typically becomes a bottleneck.

We present an efficient framework for 2PC based on garbled circuits and OT extension. Our techniques trade some computational efficiency for reduced communication, lessening the communication bottleneck. Our garbling scheme supports free XOR gates and only requires $1.5\lambda + 5$ bits of communication, where λ is the security parameter. It improves over the state-of-the-art "half-gates" scheme of Zahur, Rosulek, and Evans (Eurocrypt 2015), in which XOR gates are free and AND gates cost 2λ bits. Our OT extension is the first to improve on the communication efficiency of IKNP (Ishai et al., Crypto 2003) without using additional non-Minicrypt assumptions. While IKNP requires λ bits of communication for

each OT, SoftSpokenOT only needs λ/k bits, for any k, at the expense of requiring $2^{k-1}/k$ times the computation. We also present a simple and efficient protocol for generating a batch of 128 base OTs, and describe a flaw present in several existing batch OTs.

Communication-Efficient Secure Two-Party Computation From Minicrypt and OT

by

Lawrence Roy

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented October 18, 2022
Commencement June 2023

Doctor of Philosophy dissertation of Lawrence Roy presented on October 18, 2022.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Lawrence Roy, Author

ACKNOWLEDGEMENTS

I'd like to start by thanking my advisor, Mike Rosulek, who educates with insight, clarity, and humor. He is consistently a great teacher, whether the topic is cryptography, computer science theory, writing, or the research culture. He led me down the road to cryptography while I was questioning him as we walked back to his office from his Theory of Computation class. I'll always have fond memories of his office whiteboard with notes from our latest discussion. When brainstorming research possibilities, he knows how to bring in points that help me see the bigger picture, and is always willing to listen to my ideas, no matter how crazy.

Next I'd like to thank my committee member Mike Bailey, who has been a mentor and supporter since I was 12 years old and auditing my first CS classes at Oregon State. He always took the time to help me when I stopped by his office or wrote him an email. Without his support, that of my previous advisor, Eugene Zhang, and former Vice Provost and Dean of the Graduate School, Jennifer Brown, it's unlikely that I would have been admitted directly into OSU's graduate program.

I would also like to acknowledge and thank my other committee members, Huck Bennett, Lisa Madsen, and Clayton Petsche, for making the time to help me through this process. Huck and Clayton taught me valuable skills in lattices and abstract algebra that have been very useful in my research. They also were willing to sign onto my committee at short notice to keep the process moving and to allow me to gain a minor in Mathematics. Lisa has been gracious with her time and patient with my many questions about statistics.

I have had many coauthors including Tommy Hollenberg, Yeongjin Jang, Stan Lyakhov, Ian McQuoid, and Jaspal Singh. I've enjoyed working with all of them and their contributions were invaluable.

Paul McKenney has been a mentor and friend for many years. I have enjoyed our many discussions at Beaver Barcamp, the IBM lunchroom, and in our lengthy email exchanges. He is always willing to listen to my ideas and point out where things will likely go wrong — often accompanied by a funny anecdote from his years of experience.

Another Paul needs acknowledgement here. Paul Paulson taught the first class that I audited at OSU. He welcomed me into his class and introduced me to other faculty, including Mike Bailey.

I was fortunate to receive a U.S. Department of Energy Computational Science Graduate Fellowship that supported me generously for 4 years of my PhD program. The opportunity to interact and share ideas with other fellows and DOE scientists helped broaden my outlook and shape my career. The staff at the Krell Institute, who administer the fellowships, were talented, friendly, and always helpful and well-organized.

Thank you to the fellow members of the OSUSEC CTF team, where I was able to enjoy practicing cryptanalysis, and hacking more generally.

Finally, I'd like to thank my parents for letting me follow my own educational path and working hard to find opportunities that suited me.

CONTRIBUTION OF AUTHORS

This dissertation is based on work coauthored with Ian McQuoid and Mike Rosulek [RR21; MRR21; Roy22]. Mike collaborated in designing and writing the 3/2 garbling protocol (Chapter 2). Both Mike and Ian were involved in designing and writing the batched OT (Chapter 3), and Ian additionally assisted in its benchmarking.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDICES

Page

LIST OF APPENDIX FIGURES

Figure Page

Chapter 1: Introduction

Imagine conducting a study to compare students' grades in health class with their medical records after they graduate, to test how well the class improves students' health. Since the grades are protected by FERPA and their medical records by HIPAA, it may be difficult to get both datasets in one place legally. Imagine a sale of a private company whose value depends on secret information, so neither the buyer nor the seller want to reveal their own valuation. They only want to reveal whether the buyer values the company more than the seller. Secure two-party computation (2PC)[1] is the tool that accomplishes these tasks, allowing two independent parties to learn a function of both their inputs, without revealing anything else. Essentially, 2PC allows arbitrary operations (represented as boolean circuits) to be performed on encrypted data—with different 2PC protocols using different kinds of encryption.

There are many protocols for 2PC, but nearly all are based on a few core techniques [EKR18]. First, some protocols are based on fully-homomorphic encryption (FHE) [AJL+12], a specialized encryption technique where any arbitrary computation f may be evaluated homomorphically. That is, given an encrypted input x, one may compute an encryption of $f(x)$, without any access to the decryption key. FHE tools work with a low communication bandwidth, as essentially only the encrypted inputs and outputs need to be communicated. Unfortunately, homomorphically evaluating each gate in the circuit has a high computational cost.

There are several techniques based on secret sharing, including GMW [Gol04] and Beaver triples [Bea92]. In secret sharing, the private inputs are divided into shares, and given out to the two parties. An individual share is completely uncorrelated with the secret, but the two shares together reveal it. Computing an AND gate requires interaction between the two parties, using a specialized 2PC protocol that cheaply generates new shares of the result of the AND gate. For Beaver's technique, each AND gate uses a "Beaver triple", a triple of random bits (a, b, c) satisfying $c = a \wedge b$ that is secret shared between the parties during a preprocessing phase. If the function has high circuit depth (i.e. its evaluation is

[1] And secure multi-party computation more generally.

not parallelizable), the parties will frequently have to wait for messages to arrive from each other, creating a latency bottleneck.

Instead, this work will focus on the first secure 2PC technique, garbled circuits.

1.1 Garbled Circuits

Garbled circuits (GC) were introduced by Yao in the 1980s [Yao86], and can be viewed as a compromise between FHE and secret sharing protocols. Like secret sharing, GC uses communication for each gate, but this communication is just one big message, without multiple rounds of interaction. Like FHE, GC requires computing some cryptographic operations for each gate, but in GC these operations are orders of a magnitude cheaper. They are extremely efficient because they are Minicrypt [Imp95], i.e., they use only symmetric-key operations that can be instantiated with block ciphers or hash functions (modeled as random oracles). In realistic deployments, each gate uses only a few calls to the AES block cipher, which is implemented in hardware. But the communication required by GC is significant — the parties must exchange $\Theta(\lambda)$ bits per gate, where λ is the security parameter.[2] While FHE bottlenecks on computation, and secret sharing 2PC bottlenecks on network latency, GC bottlenecks on network bandwidth.

Garbled circuits gives the two parties roles, with one becoming **the garbler** and the other **the evaluator**. For each wire W in the circuit, the garbler picks two **wire labels**: W_0 to represent a 0, and W_1 to represent a 1. The garbler will tell the evaluator exactly the wire labels corresponding to the real inputs. That is, if input I is set to 0, the evaluator will learn I_0, but not the opposite label I_1. The garbler will then garble each gate in the circuit, creating an encrypted lookup table that maps (A_i, B_j) to $C_{g(i,j)}$, where $g(i,j)$ is the truth table of the gate. This lookup table is called a **gate ciphertext**. Each gate ciphertext is $\Theta(\lambda)$ bits long, and sending them all to the evaluator is the most expensive step of the protocol. The evaluator will then step through the circuit in topological order, evaluating each gate using the corresponding gate ciphertext, and learn one wire label for each wire in the circuit. Finally, the evaluator sends the output wire labels back to the garbler, who interprets them to get the result of the 2PC.

Most improvements to garbled circuits have focused heavily on reducing their concrete size [BMR90; NPS99; KS08; PSS+09; KMR14; GLN+15]. Originally, Yao's protocol took

[2] That is, the cost of breaking the scheme must be on the order of 2^λ operations.

8λ bits for each gate. The state of the art for garbled (boolean) circuits was the *half-gates* construction of Zahur, Rosulek, and Evans [ZRE15]. In the half-gates scheme, AND gates are garbled with size 2λ bits, while XOR gates are free, requiring no communication. The half-gates paper also establishes a lower bound for the size of garbled circuits. Specifically, the authors define a model of **linear garbling** — which captured all known techniques at the time — and proved that a garbled AND gate in this model requires 2λ bits. Thus, half-gates is optimal among linear garbling schemes.

In Chapter 2 we present **3/2 garbling**, the first technique that bypasses the half gates lower bound. This result originally appeared in a paper at Crypto 2021 [RR21], in collaboration with my advisor. In our scheme, XOR gates are free, and AND gates cost only $\frac{3}{2}\lambda + 5$ bits each. For the typical case of $\lambda = 128$ this is a concrete reduction of 23% in the size of garbled circuits relative to half-gates. We also present a variant of our scheme suitable for gate hiding garbling, where any two-input truth table can be garbled for $1.5\lambda + 10$ bits while remaining hidden from the evaluator.

1.2 Oblivious Transfer

Garbled circuits only need Minicrypt assumptions, and so cannot be a stand alone 2PC protocol [IR90]. In fact, they depend on a functionality called oblivious transfer (OT). The garbler needs to give the evaluator the correct input wire labels, without learning the evaluator's inputs. This is exactly what OT provides. In OT, one party ("the receiver") gets exactly one of two messages from the other ("the sender"), without the sender learning which one. More generally, for any $0 \leq K \leq N$ there is $\binom{N}{K}$-OT (or K-out-of-N OT), where the receiver gets exactly K out of the sender's N messages. For garbled circuits, one OT is needed for each bit of the evaluator's input.

We need a batch of many OTs, perhaps millions, yet most OT protocols in the literature are described in terms of a single OT instance. Looking ahead, we will use a technique called OT extension to reduce the number of OTs needed to only λ, but this is still a sizeable batch. Obviously any single-instance OT protocol can be invoked λ times to produce λ OTs; however, this overlooks the possibility of optimizations for the batch setting.

In Chapter 3, we investigate **naïve batching**, a natural way to optimize certain 2-round OT protocols for the batch setting. This work originally appeared at Asiacrypt 2021 [MRR21], in collaboration with my advisor Mike Rosulek and my colleague Ian McQuoid. When the OT

sender is first to speak, it seems natural to reuse their protocol message for all OT instances in the batch. In Section 3.3, we show that it is *not* guaranteed to be secure. Unfortunately, improper batching (including naïve batching) has been implemented in several protocol libraries [Rin; CMR; Kel20; Sma] and appeared in several papers [CO15; HL17; CSW20]. But it is simple and cheap to fix. We constructed a batch protocol called POPF OT (Fig. 3.3), based on our non-batched OT protocol from CCS 2020 [MRR20]. It has been correctly optimized for the batch setting, yielding an efficient batched OT protocol with roughly half the communication compared to repeating the non-batched OT protocol.

1.3 Oblivious Transfer Extension

Unfortunately, POPF OT is based on public-key primitives, like all "base OT" protocols. This makes it computationally slow compared to garbled circuits. Unfortunately, OT cannot be constructed from scratch in the Minicrypt model [IR90]. Nevertheless, it is possible to generate a large number of "extended OTs" from symmetric-key primitives and *a small number of base OTs*, thanks to an idea called **OT extension** [Bea96]. With OT extension, parties can generate many OT instances where the *marginal cost* of each instance involves only cheap symmetric-key operations. This became very practical with the protocol of Ishai et al. [IKN+03] (hereafter, IKNP), which is fast enough to still be in widespread use.

INKP has similar strengths and weaknesses to garbled circuits. It is computationally very efficient because it uses only Minicrypt operations, i.e. calls to symmetric-key primitives. But it sends λ bits per extended OT, so it bottlenecks on network bandwidth in any realistic implementation. Recent works under the heading of Silent OT [BCG+18; BCG+19b; SGR+19; BCG+19a; YWL+20; CRR21] have communication complexity that grows only *logarithmically* in the number of oblivious transfers. Consequently, they are favored when communication is slow. On the other hand, IKNP has the advantage for computational cost: of the Silent OT protocols, only Silver [CRR21] uses a comparable amount of computation to IKNP. Additionally, while IKNP uses only Minicrypt assumptions, Silent OT is based on the learning parity with noise (LPN) assumption, which is not Minicrypt. Efficient instantiations depend on highly structured versions of this problem, with the most efficient protocol, Silver, owing its efficiency to a novel variant of LPN that was introduced solely for that work. Compared with a tried-and-true block cipher like AES, these assumptions are too recent to have received as much cryptanalysis.

Chapter 4 addresses IKNP's communication inefficiency with SoftSpokenOT, an OT extension protocol that was originally presented at Crypto 2022 [Roy22]. It is the first OT extension in the Minicrypt model to make an asymptotic improvement over IKNP's communication cost. For any parameter $k \geq 1$, SoftSpokenOT can implement OT extension using only λ/k bits per OT, compared to IKNP's λ bits. This is a communication–computation tradeoff, as the sender in our protocol must *generate* $\lambda \cdot 2^k/k$ pseudorandom bits, while IKNP only needs to generate 2λ bits. In practice, fast hardware implementations of AES make IKNP network bound, so when k is small (e.g. $k = 5$) this extra computation will have no effect on the overall SoftSpokenOT protocol latency. And for $k = 2$, no extra computation is required, making SoftSpokenOT a pure improvement over IKNP.

1.4 Security Model

In 2PC, one or more parties may be corrupt, and there are two models for what the corrupted parties may do. In the semi-honest (a.k.a. honest but curious) model, they report everything they see to the adversary, but otherwise follow the protocol honestly. For malicious security, the adversary may program the corrupted parties to do anything; there's no requirement to follow the protocol. We prove POPF OT secure in the stronger[3] malicious model. We have also proven malicious security for SoftSpokenOT, when it is combined with a consistency check. However, like most garbling schemes, 3/2 garbling is vulnerable to a malicious garbler, and only has semi-honest security. This means that our 2PC stack is only secure in the semi-honest model.

[3]For most protocols, semi-honest security follows easily from malicious security.

Chapter 2: 3/2 Garbling

Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event. Springer, Heidelberg, August 2021. DOI: 10.1007/978-3-030-84242-0_5

2.1 Introduction

Garbled circuits (GC) were introduced by Yao in the 1980s [Yao86] in one of the first secure two-party computation protocols. They remain the leading technique for constant-round two-party computation. Garbled circuits exclusively use extremely efficient symmetric-key operations (*e.g.*, a few calls to AES per gate of the circuit), making communication rather than computation the bottleneck in realistic deployments — the parties must exchange $O(\lambda)$ bits per gate. For that reason, most improvements to garbled circuits have focused heavily on reducing their concrete size [BMR90; NPS99; KS08; PSS+09; KMR14; GLN+15]. The current state of the art for garbled (boolean) circuits is the *half-gates* construction of Zahur, Rosulek, and Evans [ZRE15], in which AND gates are garbled with size 2λ bits, while XOR gates are free, requiring no communication.

The half-gates paper also has a lower bound for the size of a garbled AND gate. Specifically, all known techniques at the time were covered by the **linear garbling** model that the authors defined. They proved that at least 2λ bits are required to garbled an AND gate in this model. Thus, half-gates is optimal among linear garbling schemes. In response, there has been a line of work focused on finding ways around the lower bound. Several works [KKS16; BMR16; WM17] were successful in constructing an AND gate using only λ bits, using techniques outside of the linear-garbling model. However, these constructions work *only for a single AND gate in isolation,* so they do not result in any improvement to half-gates for garbling general circuits.[1] Garbling an entire *arbitrary circuit* with less than 2λ bits per AND-gate

[1] These constructions require the input labels to have a certain correlation that they do not guarantee for the gate's output labels.

remained an open problem. We discuss the linear garbling lower bound and different paths around it later in Section 2.7.

2.1.1 Our Results

We show a garbling scheme for general boolean circuits, in which XOR gates are free and AND gates cost only $1.5\lambda + 5$ bits. This is the first scheme to successfully bypass the linear-garbling lower bound for all AND gates in a circuit, not just a single isolated AND gate. For the typical case of $\lambda = 128$ this is a concrete reduction of 23% in the size of garbled circuits relative to half-gates. Our construction compares to half-gates along other dimensions as follows:

- **Hardness assumption:** All free-XOR-based garbling schemes require a function H with output length λ and satisfying a *circular correlation-robust* property. In short, this means that terms of the form $H(X \oplus \Delta)$ and $H(X \oplus \Delta) \oplus \Delta$ are indistinguishable from random, for adversarially chosen X and global, secret Δ. Our construction requires a slight generalization. First, we require H that gives outputs of length $\lambda/2$. Second, the secret Δ is split into two halves $\Delta = \Delta_L \| \Delta_R$, and we require terms like $H(X \oplus \Delta) \oplus \Delta_L$, $H(X \oplus \Delta) \oplus \Delta_L \oplus \Delta_R$, etc. to be indistinguishable from random.

- **Computation:** Our scheme requires 50% more calls to H per AND gate than half-gates (6 vs 4 for the garbler, and 3 vs 2 for the evaluators). Similar to other work, we can instantiate the necessary H using just 1 call to AES with a key that is fixed for the entire circuit. As a result, the computational cost of our scheme is comparable to prior work.

  Additionally, since we require H with only $\lambda/2$ bits of output, certain queries to H for different AND-gates can be combined into a single query to a λ-bit-output function. The effect of this optimization depends on the circuit topology but in some cases our construction can have identical or better computation to half-gates (see Section 2.6.2).

We bypass the [ZRE15] lower bound by using two techniques that are outside of its linear-garbling model. We refer to the techniques collectively as **slicing-and-dicing.**

- **Slicing:** In our construction the evaluator **slices wire labels** into halves, and uses (*possibly different!*) linear combinations to compute each half. We stress that this does

not halve the security — the hash H is still given the whole wire label with λ bits of entropy. To the best of our knowledge, this technique is novel in garbled circuits. As we demonstrate in detail later, introducing more linear combinations for the evaluator increases the linear-algebraic dimension in which the scheme operates, in a way that lets us exploit more linear-algebraic structures that prior schemes could not exploit.

- **Dicing:** The evaluator first decrypts a constant-size ciphertext containing "**control bits**", which determine the linear combinations (of input label [halves], gate ciphertexts, and H-outputs) he/she will use to compute the output label [halves]. The control bits are chosen randomly by the garbler (*i.e.*, by tossing "dice") in a particular way. Randomized control bits are outside of the linear garbling model, which requires the evaluator's linear combinations to be *fixed.* This technique first appeared in [KKS16].

We also describe a variant of our scheme that can garble *any* kind of gate (*e.g.*, XOR gates, even constant-output gates) for $1.5\lambda + 10$ bits, in a way that hides the gate's truth table from the evaluator. This improves on the state of the art for *gate-hiding* garbling, due to Rosulek [Ros17], in which each gate is garbled for $2\lambda + 8$ bits, and constant-output gates are not supported. Additionally, our gate-hiding construction is fully compatible with free-XOR, meaning that the circuit can contain both "public" XOR gates (evaluator knows that this gate is an XOR) and "private" XOR gates (only the garbler knows that this gate is an XOR), with the public ones being free.

2.1.2 Related Work

The garbled circuits technique was first introduced by Yao [Yao86], although the first complete description and security proof for Yao's protocol was given much later [LP09]. Bellare, Hoang, and Rogaway [BHR12] promoted garbled circuits from a *technique* to well-defined cryptographic *primitive* with standardized security properties, which they dubbed a **garbling scheme.** In this work, we use their framework to formally express our schemes and prove security.

The garbling scheme formalization captures many techniques, but in this work we focus on "practical" GC techniques built from symmetric-key tools (PRFs, hash functions, but not homomorphic encryption or obfuscation). In the realm of practical garbling, there have been many quantitative and qualitative improvements over the years, especially focused on

scheme	GC size (λ bits / gate) AND	XOR	calls to H per gate: garbler AND	XOR	evaluator AND	XOR	assump.
unoptimized textbook Yao	8	8	4	4	2.5	2.5	PRF
Yao + point-permute [BMR90]	4	4	4	4	1	1	PRF
$4 \to 3$ row reduction [NPS99]	3	3	4	4	1	1	PRF
$4 \to 2$ row reduction [PSS+09]	2	2	4	4	1	1	PRF
free-XOR [KS08]	3	0	4	0	1	0	CCR
fleXOR [KMR14]	2	$\{0,1,2\}$	4	$\{0,2,4\}$	1	$\{0,1,2\}$	CCR
half-gates [ZRE15]	2	0	4	0	2	0	CCR
[GLN+15]	2	1	4	3	2	1.5	PRF
ours	**1.5**	**0**	≤ 6	0	≤ 3	0	CCR

Table 2.1: Comparison of efficient garbling schemes. Gate size ignores small constant additive term (*i.e.*, "2" means $2\lambda + O(1)$ bits per gate). CCR = circular correlation robust hash function.

reducing the size of garbled circuits. These works are showcased in Table 2.1. Of particular note are the Free-XOR technique of Kolesnikov & Schneider [KS08] and the half-gates consruction [ZRE15], mentioned above. Free-XOR allows XOR gates in the circuit to be garbled with no communication, and our construction inherits this technique to achieve the same feature. The free-XOR technique requires a cryptographic hash with a property called circular correlation-resistance [CKK+12]. As mentioned above, the half-gates paper introduced a lower bound for garbling, which several works have bypassed in some limited manner. We discuss the lower bound and these related works in more detail in Section 2.7.

Several garbling schemes are tailored to support both AND and XOR gates while hiding the type of gate from the evaluator [KKS16; WM17; Ros17]. These works are compared in Table 2.2. They differ in the exact class of boolean gates they can support — all gates, all symmetric gates (satisfying $g(0,1) = g(1,0)$), or all non-constant gates.

2.2 Preliminaries

2.2.1 Circuits

We represent a circuit $f = (\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak}, \mathsf{eval})$ by choosing a topological order of the $|f|$ inputs and gates in the circuit. Let inputs be the number of inputs in the circuit, which we require to come first in the ordering. Each gate is then labeled by its index in

scheme	GC size (λ bits/gate)	calls to H per gate garbler	evaluator	supported gates	assump.
Yao + point-permute [BMR90]	4	4	1	all	PRF
$4 \to 3$ row reduction [NPS99]	3	4	1	all	PRF
[KKS16]	2	3	1	symmetric	CCR
[WM17]	2	3	1	symmetric	CCR
[Ros17]	2	4	1	non-const	PRF
ours	**1.5**	≤ 6	≤ 3	**all**	CCR

Table 2.2: Comparison of **gate-hiding** garbling schemes, where the garbled circuit leaks only the topology of the circuit and not the type of each gate. Gate size ignores small constant additive term (*i.e.*, "2" means $2\lambda + O(1)$ bits per gate). CCR = circular correlation robust hash function. "Symmetric" means all gates g with $g(0,1) = g(1,0)$. "Non-const" means all gates g except $g(a,b) = 0$ and $g(a,b) = 1$.

the order. For every gate index g in the circuit, its two input indices[2] are $\mathsf{in}_1(g)$ and $\mathsf{in}_2(g)$, where $\mathsf{in}_i(g) < g$. Each gate can be evaluated using a function $\mathsf{eval}(g)\colon \{0,1\}^2 \to \{0,1\}$. Finally, the outputs are a subset of the indices $\mathsf{outputs} \subseteq [1, |f|]$.

Garbling only hides only partial information about the circuit. What is revealed is contained in the "leakage function" $\Phi(f)$. Sometimes two gates in a circuit may both be e.g. XOR-gates, but one will publicly be XOR while the operation performed by the other gate will be hidden. To support this, each gate is associated with some leakage $\mathsf{leak}(g)$. Gates with different leakages may compute the same function, but have different rules about how much information is revealed. We then define $\Phi(f)$ to be $(\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak})$, containing the circuit topology and partial information about the gates' truth tables.

2.2.2 Garbling Schemes

We use a slightly modified version of the garbling definitions of [BHR12].

Definition 2.2.1. *A* garbling scheme *consists of four algorithms:*

- $(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$.
- $X := \mathsf{Encode}(e, x)$. *(deterministic)*

[2]We assume that all gates take two inputs. NOT gates can be merged into downstream gates — *e.g.* if x goes into a NOT gate, and then into an AND gate with another input y, this is equivalent to a single $\overline{x} \wedge y$ gate.

- $Y := \mathsf{Eval}(F, X)$. *(deterministic)*
- $y := \mathsf{Decode}(d, Y)$. *(deterministic)*

such that the following conditions hold.

Correctness: *For any circuit f and input x, if $(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$ then $f(x) = \mathsf{Decode}(d, \mathsf{Eval}(\mathsf{Encode}(e, x)))$ holds with all but negligible probability.*

Privacy with respect to leakage Φ: *There must be a simulator $\mathcal{S}$ such that for any circuit f and input x the following distributions are indistinguishable.*

$$\boxed{\begin{array}{l}(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f) \\ X := \mathsf{Encode}(e, x) \\ \text{return } (F, X, d)\end{array}} \qquad \boxed{\begin{array}{l}(F, X, d) \leftarrow \mathcal{S}(1^\lambda, \Phi(f), f(x)) \\ \text{return } (F, X, d)\end{array}}$$

Obliviousness w.r.t. leakage Φ: *There must be a simulator $\mathcal{S}$ such that for any circuit f and input x the following distributions are indistinguishable.*

$$\boxed{\begin{array}{l}(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f) \\ X := \mathsf{Encode}(e, x) \\ \text{return } (F, X)\end{array}} \qquad \boxed{\begin{array}{l}(F, X) \leftarrow \mathcal{S}(1^\lambda, \Phi(f)) \\ \text{return } (F, X)\end{array}}$$

Authenticity: *For any circuit f and input x, no PPT adversary $\mathcal{A}$ can make the following distribution output* TRUE *with non-negligible probability.*

$$\boxed{\begin{array}{l}(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f) \\ X := \mathsf{Encode}(e, x) \\ Y \leftarrow \mathcal{A}(F, d, X) \\ \text{return } \mathsf{Decode}(d, Y) \notin \{f(x), \bot\}\end{array}}$$

The definitions differ from [BHR12] in two ways. First, we change correctness to allow a negligible failure probability.[3] Secondly, we strengthen the authenticity property by giving d

[3] Most garbling schemes actually do not have perfect correctness. If an output wire has labels W_0, W_1, then d will contain both $H(W_0)$ and $H(W_1)$. Correctness is violated if $H(W_0) = H(W_1)$.

to the adversary. This stronger property is easy to achieve by simply changing what one takes as garbled output Y.

2.2.3 Circular Correlation Robust Hashes

Our construction requires a hash function H with a property called *circular correlation robustness (CCR)*. A comprehensive treatment of this property is presented in [CKK+12; GKW+20b].

The relevant definition of [GKW+20b] is *tweakable CCR* (TCCR). For a hash function H, define a related oracle $\mathcal{O}_\Delta(X, \tau, b) = H(X \oplus \Delta, \tau) \oplus b\Delta$. Then H is a TCCR if $\mathcal{O}_\Delta$ is indistinguishable from a random oracle, provided that the distinguisher never repeats a (X, τ) pair in calls to the oracle.

We modify their definition in several important ways:

- We require H to have different input and output lengths. In the original definition, the adversary used the argument $b \in \{0, 1\}$ to determine whether Δ was XOR'ed with the output of H. We generalize so that the adversary can choose a linear function of (the bits of) Δ that will be XOR'ed with the output of H. Our construction ultimately needs only 4 linear functions reflecting our slicing of wire labels in half: $L_{a,b}(\Delta_L \| \Delta_R) = a\Delta_L \oplus b\Delta_R$, for $a, b \in \{0, 1\}$.

- [GKW+20b] observe that a "full" TCCR is stronger than what is needed for garbled circuits. In order to construct a TCCR that uses only one call to an ideal permutation, they prove TCCR security against adversaries that query only on "*naturally derived*" keys. It is somewhat cumbersome to generalize "*naturally derived*" keys to our setting, where the values are sliced into pieces.

  We instead relax TCCR so that H is drawn from a *family* of hashes, and the adversary only receives the description of H after making all of its oracle queries. This relaxation suffices for garbled circuits (the garbler chooses H and reveals it only in the garbled circuit description, after all queries to H have been made), and simplifies both our definition and our proof.

Definition 2.2.2. *A family of hash functions $\mathcal{H}$, where each $H \in \mathcal{H}$ maps $\{0, 1\}^n \times \mathcal{T} \to \{0, 1\}^m$ for some set of tweaks $\mathcal{T}$, is* ***randomized tweakable circular correlation robust***

(RTCCR) *for a set of linear functions $\mathcal{L}$ from $\{0,1\}^n$ to $\{0,1\}^m$ if, for any PPTs $\mathcal{A}_1, \mathcal{A}_2$ that never repeat an oracle query to $\mathcal{O}_{H,\Delta}$ on the same (X, τ),*

$$\left| \Pr_{H,\Delta}\left[v \leftarrow \mathcal{A}_1^{H,\mathcal{O}_{H,\Delta}}; \mathcal{A}_2(v,H) = 1\right] - \Pr_{H,R}\left[v \leftarrow \mathcal{A}_1^{H,R}; \mathcal{A}_2(v,H) = 1\right] \right|$$

is negligible, where R is a random oracle and $\mathcal{O}_{H,\Delta}$ is defined as

$$\boxed{\begin{array}{c}\mathcal{O}_{H,\Delta}(X \in \{0,1\}^n, \tau \in \mathcal{T}, L \in \mathcal{L}): \\ \hline \text{return } H(X \oplus \Delta, \tau) \oplus L(\Delta)\end{array}}$$

In Appendix A.1 we show that if $F_k(X)$ is both a (plain) CCR hash for $\mathcal{L}$ when k is fixed and a PRF when k is random, and $\{(X,\tau) \mapsto X \oplus U(\tau) \mid U \in \mathcal{U}\}$ is a universal hash family,[4] then $\left\{(X,\tau) \mapsto F_k(X \oplus U(\tau)) \mid k \in \{0,1\}^\lambda, U \in \mathcal{U}\right\}$ is a secure RTCCR hash family for $\mathcal{L}$.

For our recommended instantiation, let σ be a simple function of the form $\sigma(X_L \| X_R) = \alpha X_L \| \alpha X_R$, where α is any fixed element in $\mathbb{F}_{2^{\lambda/2}} \setminus \mathbb{F}_{2^2}$. Then $\mathsf{AES}_k(X) \oplus \sigma(X)$ is both a PRF for random k, and a CCR for any fixed k (modelling AES_k as an ideal permutation). Hence we get an RTCCR of the form:

$$(X,\tau) \mapsto \mathsf{AES}_k\Big(X \oplus U(\tau)\Big) \oplus \sigma(X \oplus U(\tau))$$

U can likewise be a simple function, *e.g.*, when $|\tau| \le \lambda/2$ then we can use $U(\tau) = u_1\tau \| u_2\tau$ where u_1, u_2 are random elements of $\mathbb{F}_{2^{\lambda/2}}$.

2.3 A Linear-Algebraic View of Garbling Schemes

In this section we present a linear-algebraic perspective of garbling schemes, which is necessary to understand our construction and its novelty. This perspective is inspired by the presentation of Rosulek [Ros17], where the evaluator's behavior (in each of the 4 different gate-input combinations) defines a set of linear equations that the garbler must satisfy, and we rearrange those equations to isolate the values that are outside of the garbler's control.

[4] Equivalently, $\mathcal{U}$ is $2^{-\lambda}$-almost-XOR-universal (AXU).

2.3.1 The Basic Linear Perspective

Throughout this section, we consider an AND gate whose input wires have labels (A_0, A_1) and (B_0, B_1). We will always consider the free-XOR setting [KS08], where all wires have labels that xor to a common global Δ; *i.e.*, $A_0 \oplus A_1 = B_0 \oplus B_1 = \Delta$. Our view of garbling will always start with the circuit evaluator's perspective; hence we consider the subscripts to be public. In other words, if the evaluator holds A_i, then he knows the value i. In some works these subscripts are called "color bits" or "permute bits." The garbler secretly knows which of $\{A_0, A_1\}$ represent true and which of $\{B_0, B_1\}$ represent true.

Let's take an example of a textbook Yao garbled gate, using the point-permute technique. The garbled gate consists of 4 ciphertexts $G_{00}, \ldots, G_{11}$. When the evaluator has input labels A_i, B_j, he computes the output label by decrypting the (i, j)'th ciphertext, as $H(A_i, B_j) \oplus G_{ij}$.[5] In order to correspond to an AND gate, this evaluation expression must result in some label C (which could be either C_0 or C_1) representing (false) in 3 cases and $C \oplus \Delta$ (true) in the other. Suppose (A_1, B_0) is the case corresponding to inputs (true,true), then the garbler needs to arrange for:

$$C = H(A_0, B_0) \oplus G_{00} \qquad C \oplus \Delta = H(A_1, B_0) \oplus G_{10}$$
$$C = H(A_0, B_1) \oplus G_{01} \qquad C = H(A_1, B_1) \oplus G_{11}$$

We can rearrange these equations as follows:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C \\ G_{00} \\ G_{01} \\ G_{10} \\ G_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} H(A_0, B_0) \\ H(A_0, B_1) \\ H(A_1, B_0) \\ H(A_1, B_1) \end{bmatrix} \oplus \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{t} \Delta$$

In this equation, values that the garbler **cannot control** are on the right, and the results of the garbling process (gate ciphertexts and output labels) are on the left. The vector marked t is the truth table of the gate (when inputs are ordered by color bits), and known only to the garbler.

[5] For now, assume H is a random oracle. We ignore including the gate ID as an additional argument to H.

In order for the scheme to work, for all possible values on the right-hand side (including all choices of secret t!) the garbler must be able to solve for the variables on the left-hand side. In this case the left-hand side is under-determined so solving is easy. The garbler can simply choose random C and move it to the right-hand side. Then the matrix remaining on the left-hand side is an invertible identity matrix. Multiplying by the inverse solves for the desired values. Clearly this can be done for any t, meaning that this approach works to garble any gate (not just AND gates).

2.3.2 Row-Reduction Techniques

Row reduction refers to any technique to reduce the size of the garbled gate below 4 ciphertexts. The simplest method works by removing the ciphertext G_{00}, and simply having the evaluator take $H(A_0, B_0)$ as the output label when he has inputs A_0, B_0.

$$\begin{aligned} C &= H(A_0, B_0) \\ C &= H(A_0, B_1) \oplus G_{01} \\ C \oplus \Delta &= H(A_1, B_0) \oplus G_{10} \\ C &= H(A_1, B_1) \oplus G_{11} \end{aligned} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C \\ G_{01} \\ G_{10} \\ G_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} H(A_0, B_0) \\ H(A_0, B_1) \\ H(A_1, B_0) \\ H(A_1, B_1) \end{bmatrix} \oplus \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{t} \Delta$$

The matrix on the left is now a square matrix, and invertible. Thus for any choice of t, the garbler can solve for C and the G_{ij} values by multiplying by the inverse matrix.

2.3.3 Half-Gates

The previous example shows that decreasing the size of the garbled gate from 4 to 3 causes the matrix on the left to change from size 4×5 to 4×4. Reducing the garbled gate further (from 3 ciphertexts to 2) would cause the matrix to be 4×3, and the system of linear equations would be overdetermined! So how does the half-gates garbling scheme [ZRE15] actually achieve a 2-ciphertext AND gate?

Let us recall the gate-evaluation algorithm for the half-gates scheme, which is considerably different from all previous schemes. On inputs A_i, B_j the evaluator computes the output label as $H(A_i) \oplus H(B_j) \oplus i \cdot G_0 \oplus j(G_1 \oplus A_i)$, where G_0, G_1 are the two gate ciphertexts.

Suppose as before that A_1 and B_0 correspond to true. Then the garbler must arrange for

the following to be true:

$$\begin{aligned} C &= H(A_0) \oplus H(B_0) \\ C &= H(A_0) \oplus H(B_1) \qquad \oplus G_1 \oplus A_0 \\ C \oplus \Delta &= H(A_1) \oplus H(B_0) \oplus G_0 \\ C &= H(A_1) \oplus H(B_1) \oplus G_0 \oplus G_1 \oplus \underbrace{(A_0 \oplus \Delta)}_{A_1} \end{aligned}$$

Rearranging in our usual way, we get:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} C \\ G_0 \\ G_1 \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \oplus \underbrace{\left[\begin{array}{ccc|c} 0 & \cdots & 0 & 0 \\ \vdots & & \vdots & \vdots \\ \vdots & & \vdots & 0 \\ \vdots & & \vdots & 1 \\ 0 & \cdots & 0 & 0 \end{array}\right]}_{t} \right) \begin{bmatrix} H(A_0) \\ H(A_1) \\ H(B_0) \\ H(B_1) \\ A_0 \\ \Delta \end{bmatrix}$$

Note that Δ is used both in the truth table adjustment (t) and in the usual operations of the evaluator (implicitly, in the one case where he includes $A_1 = A_0 \oplus \Delta$ in the linear combination).

As promised, the matrix on the left is only 4×3. We cannot solve for the left-hand side by inverting this matrix as in the previous cases. Instead, the garbler takes advantage of the fact that the **matrices on both sides have the same column space.** Specifically, the columns on the left span the space of all even-parity vectors. For any choice of t containing just a single 1 (corresponding to the truth table of an AND gate), every column on the right also has even parity! Concretely, suppose the evaluator solved the first three rows of this system of linear equalities (which is possible since the first three rows on the left form an invertible matrix), then the fourth row would automatically be in equality since on both sides it is the sum of the first 3 rows.[6] One can see that this technique works only for gates whose truth table has odd parity (*e.g.*, AND gates).

Half-gates was the first garbling scheme to structure its oracle queries as $H(A_i)$ and $H(B_j)$, instead of $H(A_i, B_j)$. Our linear-algebraic perspective highlights the importance

[6] More generally, multiplying by a **left-inverse** of the matrix on the left-hand side "just works," as in the case where the matrix on the left-hand side is invertible.

of this change. For a 2-ciphertext AND gate, the matrix on the left will be 4×3, so the matrix on the right must have rank 3. An expression like $H(A_i, B_j)$ can be used by the evaluator in only one combination of inputs, leading to an identity matrix minor that has rank 4. By contrast, each $H(A_i)$ and $H(B_j)$ term is used for two input combinations, so the corresponding matrix can have rank 3.

Our linear algebraic perspective confirms and provides an explanation for a prior finding of Carmer & Rosulek [CR16]. They used a SAT solver to show that no garbling scheme (in the linear model of the half-gates paper) could achieve a 2-ciphertext AND gate, when the evaluator makes only one query to H. This reiterates the importance of half gates using $H(A), H(B)$ oracle queries to achieve a 2-ciphertext AND gate.

2.4 High-Level Overview of Our Scheme

In the previous section, we saw that it was important that the evaluator used oracle queries like $H(A_i)$ and $H(B_j)$ in the half-gates scheme. For every term of the form $H(A_i)$ there are two gate-input combinations in which the evaluator uses this term. This property led to a desirable redundancy in the matrix that relates H-queries to input combinations. Redundancies in this matrix lead to smaller garbled gates. We push this idea further using several key observations.

2.4.1 Observation #1: Get the Most out of the Oracle Queries

$H(A_i)$ and $H(B_j)$ are not the only oracle queries that can be made in two different gate-input combinations. We can also ask the evaluator to query $H(A_i \oplus B_j)$. Because of the free-XOR constraint, $A_0 \oplus B_0 = A_1 \oplus B_1$, and $A_0 \oplus B_1 = A_1 \oplus B_0$, so these queries also can each be computed in two cases of evaluation. Fig. 2.1 shows which oracle queries can be made for each gate-input combination. Can we use these extra queries to introduce even more redundancy in the relevant matrices?

2.4.2 Observation #2: Increase Dimension by Slicing Wire Labels

Our linear-algebraic perspective of garbling includes only 4 linear equations, corresponding to the 4 different gate-inputs. Having only 4 linear equations makes it difficult to take

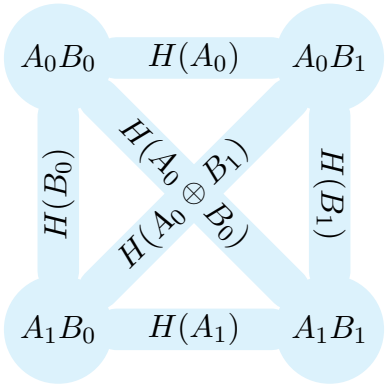


Figure 2.1: All oracle queries used by our new garbling scheme. The points represent gate-input combinations; they are labeled by the wire labels known to the evaluator. The three lines incident to a point are the three H-queries that can be made by the evaluator given those gate inputs. Half-gates only uses the outside square of oracle queries.

advantage of any new structure introduced by observation #1. Our second observation, and perhaps the key to our entire approach, is to **split each wire label into a left and right half**, and let the evaluator compute the two halves (of the output label) with different linear combinations. This results in 8 linear equations in our linear-algebraic perspective — 2 equations for each of the 4 gate-input combinations.

Consider the following proposal,

	$H(A_0)$	$H(A_1)$	$H(B_0)$	$H(B_1)$	$H(A_0 \oplus B_0)$	$H(A_0 \oplus B_1)$
(0,0) left	✓				✓	
(0,0) right			✓		✓	
(0,1) left	✓					✓
(0,1) right				✓		✓
(1,0) left		✓				✓
(1,0) right			✓			✓
(1,1) left		✓			✓	
(1,1) right				✓	✓	

(2.1)

For example, on gate-input (0,0) the evaluator will compute the left half of the output label as $H(A_0) \oplus H(A_0 \oplus B_0) \oplus \cdots$ (plus other terms, involving gate ciphertexts and input labels).

There are several important features of this table to note:

- $H(\cdot)$ is used in a linear equation to compute half of an output label, therefore $H(\cdot)$ is a function with $\lambda/2$ bits of output. Three of these half-sized hash functions are combined to encrypt the gate output.[7] However, we still will use the *entire* input wire labels as input to H — using wire-label halves as input to H would cut the effective security parameter in half.

- For an evaluator with gate-input (0,0), the values $H(A_1)$, $H(B_1)$, and $H(A_0 \oplus B_1)$ are all jointly indistinguishable from random. With that in mind, consider the linear combinations for any other gate-input. For example, in the (1,0) case the evaluator will compute the output as

$$\begin{aligned} \text{left} &= H(A_1) \oplus H(A_0 \oplus B_1) \oplus \cdots \\ \text{right} &= H(B_0) \oplus H(A_0 \oplus B_1) \oplus \cdots \end{aligned}$$

  Because $H(A_1)$ and $H(A_0 \oplus B_1)$ are pseudorandom, this makes both of these outputs *jointly* pseudorandom. The entire output of the (1,0) case is pseudorandom from the perspective of the evaluator in the (0,0) case. This is a necessary condition, since sometimes the (0,0) and (1,0) cases give different outputs. This pattern holds with respect to any pair of evaluation cases, because they will always have exactly one oracle query in common (see Fig. 2.1).

- If we interpret Eq. (2.1) as a matrix (✓=1, empty cell=0), we see that it has rank 5. This suggests that the garbling process can result in only 5 output values, where in this case each of these values is $\lambda/2$ bits. Two of the values are the halves of the output wire label C, leaving 3 values to comprise the garbled gate ciphertexts. In other words, we are on our way to a garbled gate with only $3\lambda/2$ bits, if only we can get all of the relevant linear equations to cooperate.

2.4.3 Observation #3: Randomize and Hide the Evaluator's Coefficients

Let us apply our observations so far to our linear perspective of Section 2.3. Since wire labels are divided into halves, we use notation like A_{0R} to denote the right half of A_0. Note that

[7] Hence the title: "Three Halves Make a Whole".

the free-XOR constraint applies independently to the wire label halves; *i.e.*, $A_{1R} = A_{0R} \oplus \Delta_R$ and so on.

The evaluator computes each half of the output label separately, using a linear combination of available information: oracle responses, gate ciphertexts, and the 4 (!) halves of the input labels. If we account for all 8 of the evaluator's linear equations, while using the oracle-query structure suggested in Eq. (2.1), we obtain the following system:

$$\left[\begin{array}{ccccc} 1 & 0 & ? & ? & ? \\ 0 & 1 & ? & ? & ? \\ \hline 1 & 0 & ? & ? & ? \\ 0 & 1 & ? & ? & ? \\ \hline 1 & 0 & ? & ? & ? \\ 0 & 1 & ? & ? & ? \\ \hline 1 & 0 & ? & ? & ? \\ 0 & 1 & ? & ? & ? \end{array}\right] \begin{bmatrix} C_L \\ C_R \\ G_0 \\ G_1 \\ G_2 \end{bmatrix} = \left(\left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 1 & 0 & ? & ? & ? & ? & ? & ? \\ 0 & 0 & 1 & 0 & 1 & 0 & ? & ? & ? & ? & ? & ? \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & ? & ? & ? & ? & ? & ? \\ 0 & 0 & 0 & 1 & 0 & 1 & ? & ? & ? & ? & ? & ? \\ \hline 0 & 1 & 0 & 0 & 0 & 1 & ? & ? & ? & ? & ? & ? \\ 0 & 0 & 1 & 0 & 0 & 1 & ? & ? & ? & ? & ? & ? \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & ? & ? & ? & ? & ? & ? \\ 0 & 0 & 0 & 1 & 1 & 0 & ? & ? & ? & ? & ? & ? \end{array}\right] \oplus \left[\begin{array}{c|c} 0 \cdots\cdots 0 & 0 \;\; 0 \\ \vdots & \vdots \\ & 0 \;\; 0 \\ \hline & 0 \;\; 0 \\ & 0 \;\; 0 \\ \hline & 1 \;\; 0 \\ & 0 \;\; 1 \\ \hline & 0 \;\; 0 \\ 0 \cdots\cdots 0 & \underbrace{0 \;\; 0}_{t} \end{array}\right] \right) \begin{bmatrix} H(A_0) \\ H(A_1) \\ H(B_0) \\ H(B_1) \\ H(A_0 \oplus B_0) \\ H(A_0 \oplus B_1) \\ A_{0L} \\ A_{0R} \\ B_{0L} \\ B_{0R} \\ \Delta_L \\ \Delta_R \end{bmatrix} \tag{2.2}$$

The first row represents the evaluator's linear equation to compute the *left half* C_L of the output label on input A_0, B_0, etc. Note that the truth table t now consists of 2×2 identity blocks and 2×2 zero-blocks.

For everything to work correctly, we need to replace the "?" entries, so that for every choice of t, the matrices on both sides have the same column space.

- The columns on the right-hand side (representing the H outputs) already span a space of dimension 5, so there is no choice but to extend the left-hand side matrix to a basis of that space.

- The "?" entries on the right are subject to other constraints, so that they reflect what an evaluator can actually do in each input combination. For example, on input A_0, B_1, the evaluator cannot include B_{0R} in its linear combination, it can only include $B_{1R} = B_{0R} \oplus \Delta_R$. Note that the matrix is written in terms of B_0 only.

Unfortunately, it is not possible to complete the right-hand-side matrix subject to these constraints. **For every t, there is a valid way to replace the "?" entries**, but there is

no one way that works for *all* t.

To get around this problem, we **randomize and encrypt the entries of the matrix**. To the best of our knowledge, the technique first appeared in the garbling scheme of [KKS16], and was also used in [WM17; Ros17]. The garbler will complete the matrices so that the system of equations can be solved (*i.e.*, the column spaces coincide). This causes the matrix entries to now depend on the garbler's secret t. Next, the garbler will **encrypt these matrix entries**, so that when the evaluator has input A_i, B_j, he can decrypt *only those matrix entries needed for that particular input combination* — not the entire matrix. For example, the evaluator can use A_0, B_0 to decrypt the top two rows of the matrix — just enough to determine the coefficients of the linear combinations computing the output label. Unlike other schemes, there is a step of indirection (decrypting this additional ciphertext) before the evaluator determines which linear combinations to apply — the linear combination does not depend solely on the color bits of the input labels. We call the contents of these ciphertexts **control bits**, which tell the evaluator what linear combination to apply. The control bits are of small constant size, so encrypting them adds only a constant number of bits to the garbling scheme.

The garbler completes the missing entries in the matrix by drawing them randomly from a *distribution* over matrices. The distribution depends on t, as we mentioned — however, it can be arranged that **each marginal view of the matrix is independent of t.** Since the evaluator sees only such a marginal view, not the entire matrix, the value of t is hidden.

2.5 Details: Slicing & Dicing

In this section we complete the full picture of our construction. We direct the reader to a **guide to notation/symbols** in Fig. 2.2.

Δ	free-XOR wire label offset (sometimes a vector of slices $[\Delta_L, \Delta_R]^\top$)
λ	security parameter (*e.g.*, 128)
π_k	point-permute bit of wire k: wire label W_k represents plaintext value π_k
Φ	circuit leakage function
A_0, B_0	input wire labels with color bit 0
A, B	"active" wire labels (sometimes a vector of slices $[A_L, A_R]^\top$)
C	an output wire label (sometimes a vector of slices $[C_L, C_R]^\top$)
d	dimension of basis S for control bit marginal views R_{ij}
E_k	active wire label on wire k
$\mathcal{G}$	"gate space" = column space of all matrices in (Eq. (2.3))
$\vec{G}$	vector of gate ciphertexts ($\vec{G}_k$ for the kth gate)
H, $\mathcal{H}$	RTCCR hash function, family
$\vec{H}$	vector of responses from H: $[H(A), H(A \oplus \Delta), \ldots]$ (Eq. (2.3))
$\vec{H}_0$, $\vec{H}_\Delta$	partition of $\vec{H}$ corresponding to active/inactive queries to H, respectively (in the security proof)
K	basis matrix for cokernel of gate space $\mathcal{G}$; *i.e.*, every $v \in \mathcal{G}$ satisfies $Kv = 0$
$\mathcal{L}$	set of linear functions for an RTCCR hash
M	matrix corresponding to linear combinations of H-responses (Eq. (2.3))
M_0, M_Δ	partition of M corresponding to active/inactive queries to H, respectively (in the security proof)
R	control bit matrix (Eq. (2.3)), specifying how input label slices are used in linear combinations
$\mathcal{R}(t)$	distribution over control matrices R corresponding to gate truth table t; sometimes takes gate leakage as additional argument
R'	control bit matrix R after applying a basis change in the security proof
R_p	control matrix that is always included for odd-parity gates
R_{ij}	marginal view of control matrix R, for one gate-input combination
$\bar{R}$, $\bar{R}_{ij}$	compressed representation of control matrix R, or of marginal view R_{ij}, expressed in basis S_k
$\vec{r}$	vector encoding of control matrix R, for garbling the control bits
S	basis for control bit marginal views R_{ij} (basis elements S_j); sometimes has a leakage argument
t	truth table of the gate: an 8×2 matrix composed of 2×2 identity blocks and 2×2 zero blocks
t_{ij}	the 2×2 block of t corresponding to a single gate input combination (in correctness proof)
V	matrix on LHS of the main garbling equation (Eq. (2.3)), corresponding to output label and gate ciphertexts
V_{ij}	pair of rows from V used by the evaluator when $\mathrm{lsb}(A) = i$ and $\mathrm{lsb}(B) = j$
V^{-1}	a left-inverse of V
V^{-1}_{gate}, V^{-1}_{label}	partition of V^{-1} corresponding to gate ciphertexts and output label slices, respectively (in the security proof)
W_k	wire label on wire k with color bit 0
$\vec{X}$, X_{ij}	gate output wire labels (resp. wire label) before applying control matrix (Eq. (2.10), Fig. 2.6)
x_k	plaintext value on wire k (in security proof)
$\vec{z}$	garbling/encryption of control bit matrix R / its encoding $\vec{r}$ ($\vec{z}_k$ for the kth gate)

Figure 2.2: Guide to notation.

2.5.1 Choosing the Matrices

Let us begin by filling out the question marks in Eq. (2.2). We rewrite this equation using block matrices, and we group related parts together.

$$V \begin{bmatrix} C \\ \vec{G} \end{bmatrix} = M\vec{H} \oplus \left(R \oplus \left[0 \cdots 0 \middle| t \right] \right) \begin{bmatrix} A_0 \\ B_0 \\ \Delta \end{bmatrix} \tag{2.3}$$

Here C, A_0, B_0, and Δ are two-element (column) vectors representing the two halves of these wire labels; $\vec{G}$ is the vector of gate ciphertexts; and $\vec{H} = \big[H(A_0)\ H(A_1)\ H(B_0)\ H(B_1)\ H(A_0 \oplus B_0)\ H(A_0 \oplus B_1)\big]^\top$ is the vector of H-outputs. t is the 8×2 truth table matrix, which contains a 2×2 identity matrix block for each case of the gate that should output true. We have already filled out M — it is the portion of the right-hand side matrix in Eq. (2.2) with no question marks, that operates on the hash outputs $\vec{H}$. R is called the **control matrix** because it determines which pieces of input labels are added to the output.

Choosing V. Recall that the matrices on both sides of the equation must have the same column space, and that M already spans this 5-dimensional space. Call this common column space the *gate space* $\mathcal{G}$. Then

$$\mathcal{G} = \text{colspace}(V) = \text{colspace}(M) \supseteq \text{colspace}\big(R \oplus \left[0 \cdots 0 \middle| t \right]\big).$$

It will be more convenient to represent $\mathcal{G}$ using linear constraints, rather than as the span of the columns of M. We use a matrix K as a basis for the cokernel of M, so that any vector v is in $\mathcal{G}$ if and only if $Kv = 0$. Then V must satisfy $\text{rank}(V) = 5$ and $KV = 0$.

Any K and V satisfying these constraints will suffice, and we will use the following:

$$K = \left[\begin{array}{cc|cc|cc|cc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}\right] \qquad V = \left[\begin{array}{c} 1\ 0\ 0\ 0\ 0 \\ \hline 0\ 1\ 0\ 0\ 0 \\ \hline 1\ 0\ 0\ 0\ 1 \\ \hline 0\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 0\ 1 \\ \hline 0\ 1\ 0\ 0\ 1 \\ \hline 1\ 0\ 1\ 0\ 0 \\ \hline 0\ 1\ 0\ 1\ 0 \end{array}\right]$$

Note that the columns of V corresponding to the gate ciphertexts (the 3 rightmost columns) are the same as the columns in M corresponding to hash outputs $H(A_1), H(B_1), H(A_0 \oplus B_1)$, so they are clearly in the column space of M.

Constraints on choosing R. It remains to see how we choose the control matrix R. Using our new notation, $\text{colspace}\left(R \oplus \left[0 \cdots 0 \middle| t\right]\right) \subseteq \mathcal{G}$ is equivalent to $KR = K\left[0 \cdots 0 \middle| t\right]$, so we must choose R to match Kt. Because t is composed of 2×2 zero or identity blocks, we can deduce:

$$KR = K\left[0 \cdots 0 \middle| t\right] = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & p & 0 \\ 0 & 0 & 0 & 0 & 0 & p \\ 0 & 0 & 0 & 0 & a & b \end{array}\right] \tag{2.4}$$

for some $a, b \in \{0, 1\}$, where p is the parity of the truth table. In our main construction, $p = 1$ since it only considers garbling AND gates. However, the bits a, b reveal more than the parity of the gate — they leak the position of the "1" in the truth table. Since R must depend on these a, b bits, we resort to randomizing the control matrix R to hide a, b.

We also need the control matrix to reflect linear combinations that the evaluator can actually do with the available wire labels. The linear constraints are expressed in terms of A_0, B_0, and Δ, but when the evaluator has wire label, say, A_1, he can either include it in the linear combination (adding both A_0 and Δ) or not (adding neither A_0 nor Δ) — he cannot include only one of A_0, Δ in the linear combination. This means that R must decompose

into 2×2 matrices in the following way:

$$R = \begin{bmatrix} R_{00A} & R_{00B} & 0 \\ R_{01A} & R_{01B} & R_{01B} \\ R_{10A} & R_{10B} & R_{10A} \\ R_{11A} & R_{11B} & R_{11A} \oplus R_{11B} \end{bmatrix} \tag{2.5}$$

When the evaluator holds input labels A_i, B_j, the submatrix $R_{ij} = \left[R_{ijA} \; R_{ijB}\right]$ is enough to completely determine which linear combination should be applied. We call R_{ij} the **marginal view** for that input combination. We will randomize the choice of R, subject to the constraints listed above, so that any single marginal view leaks nothing about t. That is, we want to find a distribution $\mathcal{R}(t)$ such that when $R \leftarrow \mathcal{R}(t)$, $KR = K\left[0 \cdots 0 \middle| t\right]$ with probability 1, yet for every $i, j \in \{0,1\}$, if $t \leftarrow T$ and $R \leftarrow \mathcal{R}(t)$ then t and R_{ij} are independently distributed.

Basic approach to the distribution $\mathcal{R}(t)$: We must choose R to match the p, a, b bits defined above (which depend on the truth table t). Suppose we have a distribution $\mathcal{R}_0$ with the following properties:

- If $R_\$ \leftarrow \mathcal{R}_0$ then $KR_\$ = 0$
- For all $i, j \in \{0,1\}$, if $R_\$ \leftarrow \mathcal{R}_0$ then $(R_\$)_{ij}$ (the marginal view) is uniform

and we also have fixed matrices R_p, R_a, R_b such that:

$$KR_p = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \quad KR_a = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \quad KR_b = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \tag{2.6}$$

Define $\mathcal{R}(t)$ to first sample $R_\$ \leftarrow \mathcal{R}_0$ and output $R = pR_p \oplus aR_a \oplus bR_b \oplus R_\$$. The result R will always satisfy the condition of Eq. (2.4). The randomness in $R_\$$ also causes marginal views of R_{ij} to be uniform and therefore hide p, a, b. See Appendix A.2 for details of sampling $R_\$$. Concrete values for R_p, R_a, R_b are given in Figures 2.3 and 2.4, as part of a different construction.

If $\mathcal{R}_0$ is the *uniform distribution* over all matrices satisfying $KR = 0$, then the garbler must encrypt the full marginal views R_{ij} at 8 bits per view. A more thoughtful choice of

distribution will allow the garbler to convey R_{ij} marginal views with fewer bits.

Compressing the marginal views: Each marginal view R_{ij} is a 2×4 matrix. We can "compress" these if we manage to restrict all R_{ij} to some linear subspace $S = \text{span}\{S_1, S_2, \ldots, S_d\}$ of 2×4 matrices (presumably with dimension $d < 8$), while still maintaining the other properties needed.

Let $\bar{R}_{ij}$ denote the representation of R_{ij} with respect to the basis S — *i.e.*, a vector of length d. Then the garbler can encrypt only the $\bar{R}_{ij}$'s to convey the marginal views of R. The choice of the subspace S depends on the class of truth tables that need to be hidden.

Parity-leaking gates: We performed an exhaustive computer search of low dimensional subspaces to determine how to pick the basis S for different types of gates. For even-parity gates (e.g. XOR or constant gates) we found a 2-dimensional subspace that works. Details of the $\mathcal{R}(t)$ distribution are given in Fig. 2.3. For odd-parity gates (like AND, OR) we simply use the even-parity distribution and add a public constant R_p (from Fig. 2.4) to the result. This approach works when the parity of the gate is public, since the evaluator must know to add R_p when decoding the description of their marginal view R_{ij}.

The construction for odd-parity gates is our primary construction, which would be used in most applications of garbling (in combination with free XOR gates).

Parity-hiding gates: To make the garbling scheme gate-hiding, we also need to hide the parity of the truth table. In other words, the distribution on $R_\$$ must be random enough to mask the presence (or absence) of a matrix R_p as in Eq. (2.6). The R_p in Fig. 2.4 is not in the subspace S of control matrices in Fig. 2.3. Hence, to support parity-hiding we have had to extend that subspace with two additional basis elements (the basis matrices S_1, S_2 are as in the parity-leaking case). Our parity-hiding gates require 4 (compressed) control bits per gate-input combination, corresponding to the 4-dimensional basis S. See Fig. 2.4 for details.

2.5.2 Garbling the Control Bits

So far we have glossed over the details of how the control bits actually get encrypted and sent to the evaluator. We know that there will be some $4 \times d$ ($d = 2$ for parity-leaking gates and $d = 4$ for parity hiding gates) matrix $\bar{R}$, and that the evaluator should only get to see a single row $\bar{R}_{ij}$ of $\bar{R}$ telling them what linear combination of $S_1, \ldots, S_d$ to use as control bits. The garbler can easily encrypt these values so that on input A_i, B_j the evaluator can

$$S_1 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right] \qquad S_2 = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right]$$

$$\bar{R}_a = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \bar{R}_b = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad \bar{R}_{\$} \leftarrow \text{span}\left\{ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \right\}$$

$$R_a = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \qquad R_b = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{array}\right]$$

$$R_{\$} \leftarrow \text{span}\left\{ \left[\begin{array}{cc|cc|cc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{array}\right], \left[\begin{array}{cc|cc|cc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{array}\right] \right\}$$

Figure 2.3: Control matrices for even-parity gates. The top row contains the two basis matrices for S. The bottom row shows the full control matrices (R_p is not needed for even-parity gates). The middle row shows the "compressed" representation of the control matrices, in terms of the basis $\{S_1, S_2\}$ (*i.e.*, each row expresses which linear combination of S_1, S_2 appears in the corresponding blocks of the control matrix). The reader can verify that (1) each row in $\bar{R}_{\$}$ is individually uniform; (2) $KR_{\$} = 0$; and (3) Eq. (2.6) holds.

$$S_1 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right] \quad S_2 = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right] \quad S_3 = \left[\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \quad S_4 = \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right]$$

$$\bar{R}_p = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \bar{R}_{\$} \leftarrow \text{span}\left\{ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \ldots \right\}$$

$$R_p = \left[\begin{array}{cc|cc|cc} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \qquad R_{\$} \leftarrow \text{span}\left\{ \left[\begin{array}{cc|cc|cc} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{array}\right], \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right], \ldots \right\}$$

Figure 2.4: Control matrices for gate-hiding garbling. The top row contains the basis matrices for S. The basis of Fig. 2.3 is a subset of this basis, so we can use the same R_a and R_b as Fig. 2.3. The distributions on $\bar{R}_{\$}$ and $R_{\$}$ also include the matrices from Fig. 2.3 (omitted with "..." here). The middle row gives the control matrices in terms of the new basis, while the bottom row shows them directly. The reader may verify that (1) each row of $\bar{R}_{\$}$ is individually uniform; (2) $KR_{\$} = 0$; and (3) Eq. (2.6) holds.

decrypt only $\bar{R}_{ij}$.

In order to reuse the calls to H that the evaluator already uses, it turns out that we can use our new garbling construction to *garble the control bits as well.* At first it looks like this would just give infinite recursion, as if we used something like Eq. (2.3) to garble the control bits then that garbling would need its own control bits, which would need to be garbled, and so on. In reality, the compressed control bits actually have a structure that allows us to garble them without recursive control bits.

Conceptually, we can treat the bits of $\bar{R}$ as wire labels and slice them as we do regular wire labels. Collect the bits from odd and even-indexed positions of $\bar{R}_{ij}$ into numbers $\overline{r}_{ijL}$ and $\overline{r}_{ijR} \in \mathbb{F}_{2^{d/2}}$, respectively. Define the vector

$$\vec{r} = \left[\overline{r}_{00L}\ \overline{r}_{00R}\ \overline{r}_{01L}\ \overline{r}_{01R}\ \overline{r}_{10L}\ \overline{r}_{10R}\ \overline{r}_{11L}\ \overline{r}_{11R}\right]^\top$$

We observed that for both our parity-leaking and parity-hiding constructions, this vector is always in the gate subspace $\mathcal{G}$ — *i.e.*, that $K\vec{r} = 0$. Looking at Fig. 2.3, the reader can check that this holds for any possible $\vec{r}$ (which in this case is the same as $\bar{R}$ read in row-major order). And similarly for Fig. 2.4; this time the test for $\bar{R}$ is equivalent to checking its two 4×2 blocks individually.

Since the control bits, when expressed as $\vec{r}$, are always in the gate subspace $\mathcal{G}$, they can be garbled without needing their own control bits. The garbler can compute a constant-size ciphertext $\vec{z}$ such that:

$$V\vec{z} \oplus M \operatorname{lsb}_{\frac{d}{2}}(\vec{H}) = \vec{r}, \tag{2.7}$$

where $V, M, \vec{H}$ are as in Eq. (2.3). Here we assume that every hash has been extended by an extra $d/2$ bits (or more realistically given that block ciphers have a fixed size, each wire label slice has been shrunk by $d/2$ bits to make room), and that these extra bit can be extracted with $\operatorname{lsb}_{\frac{d}{2}}$. The remainder of the hash vector, $\operatorname{msb}_{\frac{\lambda}{2}}(\vec{H})$, is used for garbling the wire labels themselves. By the same reasoning as for usual garbling, when the evaluator has input labels A_i, B_j, he can learn only the $\vec{r}_{ij}$ portions of $\vec{r}$.

We can combine Equations 2.3 and 2.7 into a single system, allowing the whole gate to

be garbled at once.

$$V\left(\vec{z}\,\middle\|\,\begin{bmatrix}C\\ \vec{G}\end{bmatrix}\right)\oplus M\vec{H}=\vec{r}\,\middle\|\left(\left(R\oplus\left[0\cdots0\middle|t\right]\right)\begin{bmatrix}A_0\\ B_0\\ \Delta\end{bmatrix}\right), \tag{2.8}$$

where $\|$ denotes element wise concatenation, so e.g. the bits of $\overline{r}_{00L}\in\mathbb{F}_{2^{d/2}}$ get concatenated with some $x\in\mathbb{F}_{2^{\lambda/2}}$ to get a value in $\mathbb{F}_{2^{(\lambda+d)/2}}$. We write the bits in little endian order, so $\mathrm{lsb}_{\frac{d}{2}}(\vec{H})\,\|\,\mathrm{msb}_{\frac{\lambda}{2}}(\vec{H})=\vec{H}$.

2.5.3 The Construction

We can now describe our garbling scheme formally. All of our different types of gates are compatible, so we describe a single unified scheme. The circuit has a leak function that indicates what information about each gate is public (which affects the cost of garbling each gate):

$\underline{\mathsf{SampleR}(t,\mathsf{leak}):}$
- $R\leftarrow\mathcal{R}(\mathsf{leak},t)$
- for $i,j\in\{0,1\}$:
  - find coeffs c s.t. $R_{ij}=\bigoplus_k c_k\,S_k(\mathsf{leak})$
  - $\overline{r}_{ijL}:=c_1\,\|\cdots\|\,c_{d-1}$ *// odd positions*
  - $\overline{r}_{ijR}:=c_2\,\|\cdots\|\,c_d$ *// even positions*
- $\vec{r}=\left[\overline{r}_{00L}\;\overline{r}_{00R}\;\overline{r}_{01L}\;\overline{r}_{01R}\;\overline{r}_{10L}\;\overline{r}_{10R}\;\overline{r}_{11L}\;\overline{r}_{11R}\right]^\top$
- if $\mathsf{leak}=\mathsf{ODD}$:
  - $R:=R\oplus R_p(\mathsf{ODD})$
- return $R,\vec{r}$

$\underline{\mathsf{Encode}((\Delta,W,\pi),x):}$
- for $k=1$ to inputs:
  - $E_k:=W_k\oplus(x_k\oplus\pi_k)\Delta$
- return E

$\underline{\mathsf{DecodeR}(\vec{r},\mathsf{leak},i,j):}$
- $\begin{bmatrix}c_1\,\|\cdots\|\,c_{d-1}\\ c_2\,\|\cdots\|\,c_d\end{bmatrix}:=\vec{r}$
- $R_{ij}:=\bigoplus_k c_k\,S_k(\mathsf{leak})$
- if $\mathsf{leak}=\mathsf{ODD}$:
  - $R_{ij}:=R_{ij}\oplus(R_p(\mathsf{ODD}))_{ij}$
- return R_{ij}

$\underline{\mathsf{Decode}((\Phi,D),E):}$
- $(\mathsf{inputs},\mathsf{outputs},\mathsf{in},\mathsf{leak}):=\Phi$
- $y:=$ empty list
- for $k\in\mathsf{outputs}$:
  - if $\exists j.\,D_k^j=H'(E_k,k)$:
    - append j to y
  - else: abort
- return y

Figure 2.5: Our garbling scheme (continued in Fig. 2.6).

$\underline{\mathsf{Garble}(1^\lambda, f)}$:

$(\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak}, \mathsf{eval}) := f$

$H \leftarrow \mathcal{H}$

$\Delta \leftarrow \begin{bmatrix} 1 \parallel \mathbb{F}_{2^{\lambda/2-1}} \\ \mathbb{F}_{2^{\lambda/2}} \end{bmatrix}$

for $k = 1$ to inputs:

$W_k \leftarrow \begin{bmatrix} 0 \parallel \mathbb{F}_{2^{\lambda/2-1}} \\ \mathbb{F}_{2^{\lambda/2}} \end{bmatrix}$

$\pi_k \leftarrow \{0,1\}$

for $k = \mathsf{inputs} + 1$ to $|f|$:

$A_0, B_0 := W_{\mathsf{in}_1(k)}, W_{\mathsf{in}_2(k)}$

$\pi_A, \pi_B := \pi_{\mathsf{in}_1(k)}, \pi_{\mathsf{in}_2(k)}$

if $\mathsf{leak}(k) = \mathsf{XOR}$:

$W_k := A_0 \oplus B_0$

$\pi_k := \pi_A \oplus \pi_B$

continue

$g := \mathsf{eval}(k)$

$t := \left[g(\pi_A, \pi_B)\ g(\pi_A, 1-\pi_B)\ g(1-\pi_A, \pi_B)\ g(1-\pi_A, 1-\pi_B)\right]^\top$

$R, \vec{r} := \mathsf{SampleR}(t, \mathsf{leak}(k))$

$$\vec{z}_k \left\| \begin{bmatrix} C \\ \vec{G}_k \end{bmatrix} := V^{-1} \left(\vec{r} \left\| \left(R \oplus \left[0 \cdots 0 \middle| t\right]\right) \begin{bmatrix} A_0 \\ B_0 \\ \Delta \end{bmatrix} \right.\right) \right.$$

$$\oplus V^{-1} M \begin{bmatrix} H(A_0, 3k-3) \\ H(A_0 \oplus \Delta, 3k-3) \\ H(B_0, 3k-2) \\ H(B_0 \oplus \Delta, 3k-2) \\ H(A_0 \oplus B_0, 3k-1) \\ H(A_0 \oplus B_0 \oplus \Delta, 3k-1) \end{bmatrix}$$

$\pi_k := \mathrm{lsb}(C)$

$W_k := C \oplus \pi_k \Delta$

for $k \in \mathsf{outputs}$, $j \in \{0,1\}$:

$D_k^j := H'(W_k \oplus (j \oplus \pi_k)\Delta, k)$

return $F = (\Phi(f), H, \vec{G}, \vec{z}),\ e = (\Delta, W, \pi),\ d = (\Phi(f), D)$

Figure 2.6: Our garbling scheme (continued from Fig. 2.5 and in Fig. 2.7). V^{-1} is a left inverse of V.

$\underline{\mathsf{Eval}(F = (\Phi, H, \vec{G}, \vec{z}), E)}$:
  $(\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak}) := \Phi$
  for $k = \mathsf{inputs} + 1$ to $|\Phi|$:
    $A, B := E_{\mathsf{in}_1(k)}, E_{\mathsf{in}_2(k)}$
    $i, j := \mathrm{lsb}(A), \mathrm{lsb}(B)$
    if $\mathsf{leak}(k) = \mathsf{XOR}$:
      $E_k := A \oplus B$
    else
      $$\vec{r} \,\|\, X_{ij} := V_{ij}\left(\vec{z}_k \,\middle\|\, \begin{bmatrix} 0 \\ \vec{G}_k \end{bmatrix}\right) \oplus \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} H(A, 3k-3) \\ H(B, 3k-2) \\ H(A \oplus B, 3k-1) \end{bmatrix}$$
      $R_{ij} := \mathsf{DecodeR}(\vec{r}, \mathsf{leak}, i, j)$
      $E_k := X_{ij} \oplus R_{ij} \begin{bmatrix} A \\ B \end{bmatrix}$
  return E

Figure 2.7: Our garbling scheme (continued from Fig. 2.6).

- EVEN: even-parity gate
- ODD: odd-parity gate
- XOR: free XOR gate
- NONE: no leakage (gate-hiding)

Because we need different control matrices depending on what kind of gate is being garbled, we use the notation $\mathcal{R}(L, t)$, for $L \in \{\mathsf{EVEN}, \mathsf{ODD}, \mathsf{NONE}\}$ to denote the appropriate distribution over control matrices. For EVEN/ODD gates, the distribution is as in Fig. 2.3 (with R_p added in the case of ODD), and for NONE the distribution is as in Fig. 2.4.

Our garbling scheme is shown in Figures 2.5 and 2.6. The garbler associates the kth wire in the circuit with a wire label W_k (and its opposite label $W_k \oplus \Delta$) and a point-and-permute bit π_k. W_k is the label with color bit $\mathrm{lsb}(W_k) = 0$ (visible to the evaluator). The label $W_k \oplus \pi_k \Delta$ is the wire label representing `false` on that wire. Equivalently, W_k is the wire label representing logical value π_k.

For each non-free gate, the garbler first samples a control matrix R and encodes its marginal views (*i.e.*, expresses each view in terms of the basis $\{S_j\}_j$). We have factored out this sampling procedure into a helper function SampleR, along with a corresponding decoding function DecodeR used by the evaluator to reconstruct its marginal view of the control matrix. One thing to note about SampleR is that in the case of a ODD gate, the

control matrices include the term R_p, but R_p is not in the subspace spanned by the basis $\{S_j\}_j$. The compressed representation of each marginal view excludes the contribution of R_p, but in these cases it is publicly known that the evaluator should compensate by manually adding R_p.

For each gate k, we have a master evaluation equation in the style of Eq. (2.8). This equation expresses constraints that must be true about that gate, but the garbler is interested in computing garbled gate ciphertexts $\vec{G}_k$, control bit ciphertexts $\vec{z}_k$, and output wire label that satisfy the constraints. As previously discussed, we can solve for these values by multipying both sides by V^{-1}, a left inverse of V. One possible choice of V^{-1} is given below:

$$V^{-1} = \left[\begin{array}{cc|cc|cc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}\right] \tag{2.9}$$

The queries to hash function H include tweaks based on the gate ID, for domain separation. Finally, for each output wire, the garbler computes hashes of the wire labels, which will be used in Decode to authenticate labels and determine their logical value (true or false). These hashes need λ bits for authenticity, so they are computed using another hash function $H'(E, k)$ with output length λ instead $\frac{\lambda+d}{2}$. It is simplest to set $H'(E,k) = \mathrm{msb}_{\frac{\lambda}{2}}(H(E, 3|f| + 2k)) \,\|\, \mathrm{msb}_{\frac{\lambda}{2}}(H(E, 3|f| + 2k + 1))$, which puts together λ bits from two evaluations of H, while avoiding any overlaps in tweaks.

The evaluator follows a similar process. Starting with the input wire labels E, it evaluates the garbled circuit one gate at a time. The invariant is that on wire k, the evaluator will hold the "active" wire label $E_k = W_k \oplus (x_k \oplus \pi_k)\Delta$, where x_k is the logical value on that wire, for the given circuit input. If A, B are the active wire labels on the input wires of this gate, then the evaluator computes terms of the form $H(A), H(B), H(A \oplus B)$ and evaluates the gate according to Eq. (2.8). The evaluator only knows enough for two rows of Eq. (2.8), depending on the color bits $i = \mathrm{lsb}(A)$, $j = \mathrm{lsb}(B)$, so we let V_{ij} be the corresponding pair of rows from V. It only evaluates the gate partially at first, in order to find the encoded control bits so that it can decode them with DecodeR and use them to finally compute the output wire label.

2.5.4 Security Proof

Theorem 2.5.1. *Let $\mathcal{H}$ be a family of hash functions, with output length $(\lambda + d)/2$ bits, that is RTCCR for $\mathcal{L} = \{L_{ab}(\Delta_L \| \Delta_R) = 0^{d/2} \| a\Delta_L \oplus b\Delta_R \mid a, b, \in \{0,1\}\}$. Then our construction (Figures 2.5 and 2.6) is a secure garbling scheme.*

Proof. We need to prove four properties of the construction.

Correctness: We need to prove an invariant: $E_k = W_k \oplus (x_k \oplus \pi_k)\Delta$ for all k, if x_k is the plaintext value on that wire. Encode chooses the inputs in this way, so at least it's true for $k \leq \mathsf{inputs}$, and it is trivially maintained for free-XOR gates. For any $v \in \text{colspace}(V) = \mathcal{G}$, we have $VV^{-1}v = v$, as there exists some u such that $v = Vu$ and $VV^{-1}Vu = Vu = v$ because V^{-1} is a left inverse of V. In Section 2.5.1 we showed that $\text{colspace}(M) = \mathcal{G}$, $\text{colspace}(R \oplus \left[0 \cdots 0 \middle| t\right]) \subseteq \mathcal{G}$, and $\vec{r} \in \mathcal{G}$, so after multiplying both sides of garbler's equation by V on the left, the VV^{-1}s will cancel, and taking a two-row piece of this equation gives the evaluator's equation. In this equation, X_{ij} is the two rows of

$$\vec{X} = C \oplus \left(R \oplus \left[0 \cdots 0 \middle| t\right]\right) \begin{bmatrix} A_0 \\ B_0 \\ \Delta \end{bmatrix}, \tag{2.10}$$

corresponding to the evaluation case i, j. The structure of R (see Eq. (2.5)) implies that the evaluator's row pair of $R[A_0^\top \; B_0^\top \; \Delta^\top]^\top$ will be $R_{ij}[A^\top \; B^\top]^\top$. Therefore

$$E_k = X_{ij} \oplus R \begin{bmatrix} A \\ B \end{bmatrix} = C \oplus t_{ij}\Delta = W_k \oplus (\mathsf{eval}(k)(\pi_A \oplus i, \pi_B \oplus j) \oplus \pi_k)\Delta,$$

which maintains this invariant because

$$i = \text{lsb}(E_{\mathsf{in}_1(k)}) = \text{lsb}\big(W_{\mathsf{in}_1(k)} \oplus (x_{\mathsf{in}_1(k)} \oplus \pi_{\mathsf{in}_1(k)})\Delta\big) = x_{\mathsf{in}_1(k)} \oplus \pi_{\mathsf{in}_1(k)},$$

and similarly for j. Finally, Decode will correctly find that $D_k^{x_k} = H'\big(W_k \oplus (x_k \oplus \pi_k)\Delta, k\big) = H'(E_k, k)$, assuming that $D_k^{x_k} \neq D_k^{1-x_k}$, which has only negligible probability of failing. Therefore it gives the correct result.

Privacy: We need to prove that generating $(\Phi, \vec{G}, \vec{z}), E, (\Phi, D)$ with Garble and Encode is indistinguishable from the output of $\mathcal{S}_{\text{priv}}$. We give a sequence of intermediate hybrids, going

$\underline{\mathcal{S}_{\text{priv}}(1^\lambda, \Phi, x)}$:
  $(\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak}) := \Phi$
  $(F, E) \leftarrow \mathcal{S}_{\text{obliv}}(1^\lambda, \Phi)$
  $E' := \mathsf{Eval}(F, E)$
  for $k \in \mathsf{outputs}$:
    $D_k^{x_k} := H'(E'_k, k)$
    $D_k^{1-x_k} \leftarrow \mathbb{F}_{2^\lambda}$
  return $F, E, (\Phi, D)$

$\underline{\mathcal{S}_{\text{obliv}}(1^\lambda, \Phi)}$:
  $(\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak}) := \Phi$
  $H \leftarrow \mathcal{H}$
  for $k = 1$ to inputs:
    $E_k \leftarrow \mathbb{F}^2_{2^{\lambda/2}}$
  for $k = \mathsf{inputs} + 1$ to $|\Phi|$:
    if $\mathsf{leak}(k) = \mathsf{XOR}$: continue
    $\vec{G}_k \leftarrow \mathbb{F}^3_{2^{\lambda/2}}$
    $\vec{z}_k \leftarrow \mathbb{F}^5_{2^{d(\mathsf{leak}(k))/2}}$
  return $(\Phi, H, \vec{G}, \vec{z}), E$

$\underline{\mathsf{Hybrid1}(1^\lambda, f, x)}$:
  $(\mathsf{inputs}, \mathsf{outputs}, \mathsf{in}, \mathsf{leak}, \mathsf{eval}) := f$
  $H \leftarrow \mathcal{H}$
  $\Delta \leftarrow \begin{bmatrix} 1 \,\|\, \mathbb{F}_{2^{\lambda/2-1}} \\ \mathbb{F}_{2^{\lambda/2}} \end{bmatrix}$
  for $k = 1$ to inputs:
    $E_k \leftarrow \mathbb{F}^2_{2^{\lambda/2}}$
  for $k = \mathsf{inputs} + 1$ to $|f|$:
    $A, B := E_{\mathsf{in}_1(k)}, E_{\mathsf{in}_2(k)}$
    $i, j := \text{lsb}(A), \text{lsb}(B)$
    $x_A, x_B := x_{\mathsf{in}_1(k)}, x_{\mathsf{in}_2(k)}$
    if $\mathsf{leak}(k) = \mathsf{XOR}$:
      $E_k := A \oplus B$
      $x_k := x_A \oplus x_B$
      continue
    $g := \mathsf{eval}(k)$
    $x_k := g(x_A, x_B)$
    $t := \begin{bmatrix} g(x_A \oplus i, x_B \oplus j) \\ g(x_A \oplus i, x_B \oplus j \oplus 1) \\ g(x_A \oplus i \oplus 1, x_B \oplus j) \\ g(x_A \oplus i \oplus 1, x_B \oplus j \oplus 1) \end{bmatrix}$
    $R, \vec{r} := \mathsf{SampleR}(t, \mathsf{leak}(k))$
    $R' := R \begin{bmatrix} 1&0&0&0&i&0 \\ 0&1&0&0&0&i \\ 0&0&1&0&j&0 \\ 0&0&0&1&0&j \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{bmatrix}$
    $\vec{H}_0 := \begin{bmatrix} H(A, 3k-3) \\ H(B, 3k-2) \\ H(A \oplus B, 3k-1) \end{bmatrix}$
    $\vec{H}_\Delta := \begin{bmatrix} H(A \oplus \Delta, 3k-3) \\ H(B \oplus \Delta, 3k-2) \\ H(A \oplus B \oplus \Delta, 3k-1) \end{bmatrix}$
    $(\vec{z}_k)_{\mathsf{bot}} \,\|\, \vec{G}_k := V^{-1}_{\mathsf{gate}} \left(\vec{r} \,\|\, \left(R' \oplus \begin{bmatrix} 0 \cdots 0 \,|\, t \end{bmatrix}\right) \begin{bmatrix} A \\ B \\ \Delta \end{bmatrix} \right)$
    $\qquad \oplus V^{-1}_{\mathsf{gate}}(M_0 \vec{H}_0 \oplus M_\Delta \vec{H}_\Delta)$
    $E_k := V_{ij} \begin{bmatrix} 0 \\ \vec{G}_k \end{bmatrix} \oplus \begin{bmatrix} 1&0&1 \\ 0&1&1 \end{bmatrix} \text{msb}_{\frac{\lambda}{2}}(\vec{H}_0) \oplus R_{ij} \begin{bmatrix} A \\ B \end{bmatrix}$
    $(\vec{z}_k)_{\mathsf{top}} := V_{ij} \begin{bmatrix} 0 \\ (\vec{z}_k)_{\mathsf{bot}} \end{bmatrix} \oplus \begin{bmatrix} 1&0&1 \\ 0&1&1 \end{bmatrix} \text{lsb}_{\frac{d}{2}}(\vec{H}_0) \oplus \vec{r}_{ij}$
  for $k \in \mathsf{outputs}$, $j \in \{0, 1\}$:
    $D_k^j := H'(E_k \oplus (j \oplus x_k)\Delta, k)$
  return $(\Phi(f), \vec{G}, \vec{z}), E, (\Phi(f), D)$

Figure 2.8: Left: simulators for privacy and obliviousness. Right: a hybrid for privacy.

from the real garbler to the simulator.

Hybrid 1: This hybrid switches from the garbler's perspective to the evaluator's perspective when garbling the circuit. Instead of keeping track of the "zero" wire label W_k for every gate, we keep track of the "active" wire label E_k, and rewrite the garbling procedure in terms of the "active" labels. This basically involves a change of variable names throughout the garbling algorithm. The changes are extensive, and given in detail in Fig. 2.8:

- Replace point-and-permute bits π_k with the equivalent expression $x_k \oplus \mathrm{lsb}(E_k)$.
- Write the control matrix part of the garbling equation in terms of active wire labels $A = E_{\mathsf{in}_1(k)}$ and $B = E_{\mathsf{in}_2(k)}$ instead of A_0 and B_0.

$$\text{replace } R \times \begin{bmatrix} A_0 \\ B_0 \\ \Delta \end{bmatrix} \text{ with equivalent } R' \times \begin{bmatrix} A \\ B \\ \Delta \end{bmatrix}.$$

  where a change of basis has been applied to R, that expresses A_0 as the appropriate linear combination of A and Δ, and expresses B_0 in terms of B and Δ.

- Partition $\vec{H}$ into two pieces:

$$\vec{H}_0 = [H(A)\ H(B)\ H(A \oplus B)]^\top$$
$$\vec{H}_\Delta = [H(A \oplus \Delta)\ H(B \oplus \Delta)\ H(A \oplus B \oplus \Delta)]^\top$$

  where again A and B are the active wire labels. Similarly partition the matrix M into M_0 and M_Δ, and replace $M \times \vec{H}$ with $(M_0\vec{H}_0 \oplus M_\Delta\vec{H}_\Delta)$.

- Note that the matrix V^{-1} has 5 rows, where the first 2 correspond to slices of the output label and the last 3 correspond to the gate ciphertexts. Denote this division of V^{-1} by V^{-1}_{label} and V^{-1}_{gate}. Instead of multiplying on the left by V^{-1} to solve for the output label and gate ciphertexts, we now multiply on the left by V^{-1}_{gate} to solve for only the gate ciphertexts. We then evaluate those gate ciphertexts with A and B to learn the (active) output label E_k. This different approach has the same result by the correctness of the scheme.

  We can similarly partition the control bit ciphertexts $\vec{z}_k = [(\vec{z}_k)_{\mathsf{top}}\ (\vec{z}_k)_{\mathsf{bot}}]$, use V^{-1}_{gate} to

compute $(\vec{z}_k)_{\mathsf{bot}}$, and then use the evaluator's computation to solve for $(\vec{z}_k)_{\mathsf{top}}$. Solving for $(\vec{z}_k)_{\mathsf{top}}$ is simplified by the first two columns of V_{ij} being the identity matrix. In this case, we solve for the missing positions using knowledge of the compressed control bits $\overline{r}_{ij}$.

All of the changes are simple variable substitutions or basis changes in the linear algebra, so this hybrid is distributed identically to the real garbling.

Hybrid 2: In this hybrid, we apply the RTCCR property of H to all oracle queries of the form $H(\cdot \oplus \Delta)$. We must show that Δ is used in a way that can be achieved by calling the oracle from the RTCCR security game.

We focus on the term

$$V_{\mathsf{gate}}^{-1} M \vec{H} = V_{\mathsf{gate}}^{-1}(M_0 \vec{H}_0 \oplus M_\Delta \vec{H}_\Delta)$$

First, consider the expression $V^{-1} \times M$, and recall that M is written in terms of the zero-labels A_0, B_0. Using the V^{-1} given in Eq. (2.9), we can compute:

$$V^{-1}M = \left[\begin{array}{cc|cc|cc} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \tag{2.11}$$

Thus $V_{\mathsf{gate}}^{-1} \times M$ will consist of the bottom three rows of Eq. (2.11).

Recall that the columns of M correspond to oracle queries $H(A_0), H(A_0 \oplus \Delta), H(B_0)$, $H(B_0 \oplus \Delta), H(A \oplus B), H(A \oplus B \oplus \Delta)$, in that order. In the current hybrid M is partitioned into M_0 (corresponding to H-queries on active labels) and M_Δ (corresponding to the other queries). In other words, M_Δ will consist of exactly one of rows $\{1, 2\}$, exactly one of rows $\{3, 4\}$, and exactly one of rows $\{5, 6\}$ from M. In all cases, the result of $V_{\mathsf{gate}}^{-1} M_\Delta$ (*i.e.*, the bottom 3 rows of $V^{-1} M_\Delta$) is the 3×3 identity matrix!

This means we can rewrite the hybrid in the following way:

$$(\vec{z}_k)_{\mathsf{bot}} \left\| \vec{G}_k := V_{\mathsf{gate}}^{-1} \left(\vec{r} \left\| \left(R' \oplus \left[0 \cdots 0 \middle| t \right] \right) \begin{bmatrix} A \\ B \\ \Delta \end{bmatrix} \right. \right) \oplus V_{\mathsf{gate}}^{-1}(M_0 \vec{H}_0 \oplus M_\Delta \vec{H}_\Delta)$$
$$= \vec{H}_\Delta \oplus [\text{linear combinations of } \Delta] \oplus \cdots$$

Since all the H-queries in $\vec{H}_\Delta$ include a Δ term, we can compute this expression with 3 suitable calls to the RTCCR oracle.[8] Finally, $D_k^{1-x_k} = H'(E_k \oplus \Delta, k)$ also uses Δ, and will become two calls to the RTCCR oracle. These transformations successfully moves all references to Δ into the RTCCR oracle.

Applying RTCCR security, it has negligible effect to replace the results of these H-queries with uniformly random values. This has the effect of making the entire expression uniform, *i.e.*:

$$(\vec{z}_k)_{\mathsf{bot}} \left\| \vec{G}_k \leftarrow \mathbb{F}_{2^{(\lambda+d)/2}}^3 \right.$$

Also, $D_k^{1-x_k}$ is now sampled uniformly at random in $\mathbb{F}_{2^\lambda}$.

Hybrid 3: After making the previous change, the only place that R is used is when we use the marginal views R_{ij} and $\vec{r}_{ij}$ to solve for the output label and for the missing pieces of the control bit ciphertexts. In Section 2.5.1 we specifically chose $\mathcal{R}$ so that this marginal views is uniform for all t and all i, j. Therefore instead of doing $R, \vec{r} \leftarrow \mathsf{SampleR}(t, \mathsf{leak}(k))$, we can simply choose uniform $\vec{r}_{ij}$ and use $\mathsf{DecodeR}$ to reconstruct R_{ij}. The change has no effect on the overall view of the adversary.

Note that after making this change, the control-bit ciphertexts $(\vec{z}_k)_{\mathsf{top}}$ become uniform since $\vec{r}_{ij}$ acts as a one-time pad.

Hybrid 4: As a result of the previous change, the hybrid no longer uses t. Additionally, t was the only place where the plaintext values x_k were used, other than in the computation of D. But D only uses plaintext values for the circuit's output wires. In other words, the entire hybrid can be computed knowing only the circuit output $f(x)$. Additionally, all garbled gate ciphertexts and control bit ciphertexts are chosen uniformly, and the active wire labels on output wires are determined by the scheme's evaluation procedure. Hence, the hybrid exactly matches what happens in $\mathcal{S}_{\text{priv}}$.

[8] Note also that the calls to H have globally distinct tweaks.

Obliviousness: Notice that $\mathcal{S}_{\text{priv}}$ calls $\mathcal{S}_{\text{obliv}}$ to generate (F, E), then samples some more random bits for decoding and returns it all. Therefore, any adversary for obliviousness could be turned into one for privacy by only looking at (F, E) and ignoring the rest.

Authenticity: The first two steps of the authenticity distribution are exactly the same as the real privacy distribution, so we can swap them for the simulated distribution $\mathcal{S}_{\text{priv}}$ in a hybrid. Then to break authenticity the adversary must cause Decode to choose $j = 1 - x_k$ for at least one output k, as otherwise it will either produce the correct answer or abort. But $D_k^{1-x_k}$ is fresh uniform randomness, so the probability that $D_k^{1-x_k} = H'(E_k, k)$ is $2^{-\lambda}$. □

2.5.5 Discussion

Concrete costs. The garbler makes 6 calls to H per non-free gate, while the evaluator makes 3 calls to H per non-free gate.

Each non-free garbled gate consists of gate ciphertexts $\vec{G}$ and encrypted control bits $\vec{z}$. There are 3 gate ciphertexts, each being $\lambda/2$ bits long. The encrypted control bits are a vector of length 5, where each component of the vector has length $d/2$ (where d is the dimension of the control matrix subspace). For the standard (parity-leaking) instantiation of our scheme, $d = 2$ and we get that the total size of a garbled gate is $1.5\lambda + 5$ bits. For the gate-hiding instantiation, $d = 4$ and we get a size of $1.5\lambda + 10$ bits.

Comparison to half-gates. We assume that calls to H are the computational bottleneck, in any implementation of both our scheme and in half-gates [ZRE15]. The following analysis therefore ignores the cost of xor'ing wire labels and bit-fiddling related to color bits and control bits.

In the time it takes to call H 12 times, half-gates generates 3 gates and sends 6λ bits (4 calls to H and 2λ bits per gate), while our scheme generates 2 gates and sends 3λ bits (6 calls to H and 1.5λ bits per gate). Thus, a CPU-bound implementation of our scheme will produce garbled output at half the rate of half-gates. We evaluated the optimized half-gates garbling algorithm from the ABY3 library [MR18], and found it capable of generating garbled output at a rate of ∼850 Mbyte/s on single core of a i7-7500U laptop processor running at 3.5GHz. Thus, we conservatively estimate that a comparable implementation of our scheme could generate garbled output at ∼400 Mbyte/s = 3.2 Gbit/s. This rate would still leave our scheme network-bound in most situations and applications of garbled circuits. When

both half-gates and our scheme are network bound, our scheme is expected to take ∼25% less time by virtue of reducing communication by 25%.

2.6 Optimizations

2.6.1 Optimizing Control Bit Encryptions

In our scheme the control bit encryptions $\vec{z}$ is a vector of length 5, where the components in that vector are each a single bit (in the case of parity-leaking gates) or 2 bits (in the case of parity-hiding gates). These ciphertexts therefore contribute 5 or 10 bits to the size of each garbled gate.

We remark that it is possible to use ideas of garbled row reduction [NPS99; PSS+09] to reduce $\vec{z}$ to a length-3 vector. This will result in these ciphertexts contributing 3 or 6 bits to the garbled gate. Such an optimization may be convenient in parity-hiding case, where the change from 10 to 6 bits allows these control bit ciphertexts to fit in a single byte.

Recall that in the security proof, we partition the control bit ciphertexts $\vec{z}$ into $(\vec{z})_{\mathsf{top}}$ (2 components) and $(\vec{z})_{\mathsf{bot}}$ (3 components). Our idea to reduce their size is to simply fix $(\vec{z})_{\mathsf{top}}$ to zeroes, so that these components do not need to be explicitly included in the garbled gate. The evaluator can act exactly as before, taking the missing values from $\vec{z}$ to be zeroes. The garbler must sample the control matrix subject to it causing $(\vec{z})_{\mathsf{top}} = 0$.

A drawback to this optimization is that it significantly complicates the security proof (and hence why we only sketch it here). When we apply the security of RTCCR in the security proof, the hybrid acts as follows:

1. It uses the $d/2$ least significant bits of the H-outputs to determine how the control bits are going to be "masked".

2. Based on these masks, it chooses a consistent control matrix R that causes the first two components of $\vec{z}$ to be 0.

3. The choice of R determines which linear combinations of wire label slices (including slices of Δ) are applied.

So the reduction to RTCCR security must first read the low bits of several $H(\cdot \oplus \Delta)$ queries before it decides which linear combination of Δ should be XOR'ed with the remaining output

of H. Of course the RTCCR oracle requires the choice of linear combination to be provided when H is called. It is indeed possible to formally account for this, but only by modeling the two parts of H's output (for masking wire label slices and for masking control bits) as separate hash functions for the purposes of the security proof.

2.6.2 Optimizing Computation

Our construction requires a RTCCR function H with output length $(\lambda + d)/2$. We propose an efficient instantiation of H which naturally results in λ-bit output, which is then truncated to $(\lambda + d)/2$. The hash produces nearly twice as many bits as needed, raising the question of whether we are "wasting" these extra bits. In fact, if we reduce the security parameter slightly so that H is derived from a $(\lambda + d)$-bit primitive, we can use these extra bits to reduce the computation cost.

Suppose H' is a [RT]CCR with $(\lambda + d)$ bits of output. Then define

$$H(X, \tau) = \begin{cases} \text{first half of } H'(X, \frac{\tau}{2}) & \tau \text{ even} \\ \text{second half of } H'(X, \frac{\tau-1}{2}) & \tau \text{ odd} \end{cases}$$

Clearly H is also a [RT]CCR with $(\lambda + d)/2$ bits of output. How can we use this H to reduce the total number of calls to the underlying H'?

When a wire with labels $(A, A \oplus \Delta)$ is used as input to an AND gate, our scheme makes calls of the form $H(A, j), H(A \oplus \Delta, j)$ where j is the ID of that AND gate. Let us slightly change how the tweaks are used. Suppose this wire with label $(A, A \oplus \Delta)$ is used as input in n different AND gates. Then those gates should make calls of the form $H(A, 0 \parallel i), H(A, 1 \parallel i), \ldots, H(A, n-1 \parallel i)$, where i is now the index of *the wire whose labels are* $(A, A \oplus \Delta)$. When H is defined as above, these queries can be computed with only $\lceil n/2 \rceil$ queries to H'.

Note that both the garbler and evaluator can take advantage of this optimization, with the garbler always requiring exactly twice as many calls to H' (if in some scenario the evaluator needs $H'(X)$ then the garbler will need $H'(X)$ and $H'(X \oplus \Delta)$). Our AND gates require calls to H of the form $H(A), H(B), H(A \oplus B)$, and so far we have discussed optimizing only the $H(A)$ and $H(B)$ queries. Similar logic can be applied to the queries of the form $H(A \oplus B)$; for example, if a circuit contains gates $a \wedge b$ and $(a \oplus b) \wedge c$, then both of those AND gates

circuit	baseline	optimized	improvement	half-gates [ZRE15]
64-bit adder	6.00	6.00	0%	4.00
64-bit division	6.00	5.75	4.1%	4.00
64-bit multiplication	6.00	4.99	16.8%	4.00
AES-128	6.00	4.31	28.2%	4.00
SHA-256	6.00	5.77	3.8%	4.00
Keccak f	6.00	4.00	33.3%	4.00

Figure 2.9: Number of calls to λ-bit H' RTCCR function (per AND gate) to garble each circuit, with and without the optimization of Section 2.6.2. Evaluating the garbled circuit costs exactly half this number of calls to H'.

will require $H(A \oplus B)$ terms that can be optimized in this way.

We explored the effect of this optimization for a selection of circuits.[9] The results are shown in Fig. 2.9. The improvement ranges from 0% to 33.3%. As a reference, our baseline construction requires 6 calls to $((\lambda + d)/2$-bit output) H to garble an AND gate, while half-gates requires 4 calls (to a λ-bit function). Interestingly, in the Keccak f-function every wire used as input to an *even number* of AND gates, so that our optimized scheme has the same computation cost as half-gates (4 calls to H' per AND gate). In principle, this optimization can result in as few as 3 calls to H' per AND gate,[10] but typical circuits do not appear to be nearly so favorable.

2.7 The Linear Garbling Lower Bound

In [ZRE15], the authors present a lower bound for garbled AND gates in a model that they call **linear garbling.** The linear garbling model considers schemes with the following properties:

- Wire labels have an associated *color bit* which must be $\{0, 1\}$.
- To evaluate the garbled gate, the evaluator makes a sequence of calls to a random oracle (that depend only on the input wire labels), and then outputs some linear combination of input labels, gate ciphertexts, and random oracle outputs. The linear combination must depend only on the color bits of the input labels.

[9] Circuits were obtained from https://homes.esat.kuleuven.be/~nsmart/MPC/

[10] This can happen, *e.g.*, when for every $a \wedge b$ gate there is a corresponding $a \vee b = \overline{\overline{a} \wedge \overline{b}}$ gate.

The bound of [ZRE15] considers only linear combinations over the field $\mathbb{F}_{2^\lambda}$, and it is unclear to what extent the results generalize to other fields.

Several works have bypassed this lower bound, and we summarize them below. All of these works show how to garble an AND gate for $\lambda + O(1)$ bits, but only a single AND gate in isolation. These constructions all require the input wire labels to satisfy a certain structure, but do not guarantee that the output labels also satisfy that structure.

- Kempka, Kikuchi, and Suzuki [KKS16] and Wang & Malluhi [WM17] both use a technique of randomizing the control bits. The evaluator decrypts a constant-size ciphertext to determine which linear combination to apply. This approach is outside of the linear garbling model, which requires that the linear combination depend only on the color bits. These works also add wire labels in $\mathbb{Z}_{2^\lambda}$ rather than XOR them (as in $\mathbb{F}_{2^\lambda}$). Apart from these similiarities, the two approaches are quite different.

- Ball, Malkin, and Rosulek [BMR16] deviate from the linear garbling model by letting each wire label have a color "trit" from $\mathbb{Z}_3$ instead of a color bit from $\mathbb{Z}_2$. There is no further "indirection" of the evaluator's linear combination — it depends only on the colors of the input labels. They also perform some linear combinations on wire labels over a field of characteristic 3.

As described earlier, we bypass the lower bound by adopting the control-bit randomization technique of [KKS16] but also introducing the wire-label-slicing technique.

2.8 Open Problems

We conclude by listing several open problems suggested by our work.

Optimality. Is 1.5λ bits optimal for garbled AND gates in a more inclusive model than the one in [ZRE15]? A natural model that excludes "heavy machinery" like fully homomorphic encryption is Minicrypt, in which all parties are computationally unbounded but have bounded access to a random oracle. Conversely, can one do better — say, $4\lambda/3$ bits per AND gate? Does it help to sacrifice compatibility with free-XOR? In our construction, free-XOR seems crucial.

Computation Cost. In Section 2.6.2 we described how to reduce the number of queries to an underlying λ-bit primitive, with an optimization that depends on topology of the circuit.

Is there a way to reduce the computation cost of our scheme (measured in number of calls to, say, a λ-bit ideal permutation), for *all* circuits?

In the best case, we can garble a circuit for only 3 (amortized) calls per AND gate, whereas all prior schemes require 4. Setting aside garbled circuit size and free-XOR compatibility, is there any scheme that can garble arbitrary circuits for less than 4 (amortized) calls to a λ-bit primitive per AND gate?

Hardness Assumption. Free-XOR garbling requires some kind of circular correlation robust assumption (see [CKK+12] for a formal statement). The state-of-the-art garbling scheme based on the minimal assumption of PRF is due to Gueron *et al.* [GLN+15], where AND gates cost 2λ and XOR gates cost λ bits. Can our new techniques be used to improve on garbling from the PRF assumption, or alternatively can the optimality of [GLN+15] be proven? Again, our construction seems to rely heavily on the free-XOR structure of wire labels, which (apparently) makes circular correlation robustness necessary.

Privacy-Free Garbling. Frederiksen *et al.* [FNO15] introduced *privacy-free* garbled circuits, in which only the authenticity property is required of the garbling scheme. The state-of-the-art privacy-free scheme is due to [ZRE15], where XOR gates are free and AND gates cost λ bits. Can our new techniques lead to a privacy-free garbling scheme with less than λ bits per AND gate (with or without free-XOR)?

Simpler Description. Is there a way to describe our construction as the clean composition of simpler components, similar to how the half-gates construction is described in terms of simpler "half gate" objects? The challenge in our scheme is the way in which left-slices and right-slices of the wire labels are used together.

Chapter 3: POPF OT

Ian McQuoid, Mike Rosulek, and Lawrence Roy. Batching base oblivious transfers. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, pages 281–310, Cham. Springer International Publishing, 2021. ISBN: 978-3-030-92078-4. URL: https://eprint.iacr.org/2021/682

3.1 Introduction

Oblivious transfer (OT) is a fundamental primitive for cryptographic protocols. It is well-known that OT cannot be constructed in a black-box way from symmetric-key primitives [IR90]. Nevertheless, thanks to an idea called **OT extension** [Bea96], it is possible to generate a large number of OTs from symmetric-key primitives and a few "base OTs". With OT extension, the cost of the base OTs can be rapidly amortized over numerous extended OT instances, because the *marginal cost* of each instance involves only cheap symmetric-key operations. Modern OT extension protocols, such as SoftSpokenOT (Chapter 4), can generate millions of OTs per second.

OT extension protocols require λ (*e.g.*, 128) base OTs, and yet most base-OT protocols in the literature are described in terms of a single OT instance. Obviously any single-instance OT protocol can be invoked λ times to produce base OTs; however, this overlooks the possibility of optimizations for the batch setting. In this work we provide a full treatment of the batch setting for recent leading OT protocols.

3.1.1 Overview of Our Results

Naïve batching is insecure. We describe a natural way to optimize certain 2-round OT protocols for the batch setting. If the OT sender is first to speak, or at least the sender's message does not depend on what the receiver says, it is natural to reuse their protocol message for all OT instances in the batch. We call this method **naïve batching**.

We show that naïve batching is not guaranteed to be secure. Not only does naïve batching fail to achieve an appropriate security notion, but it is also demonstrably unsuitable as the

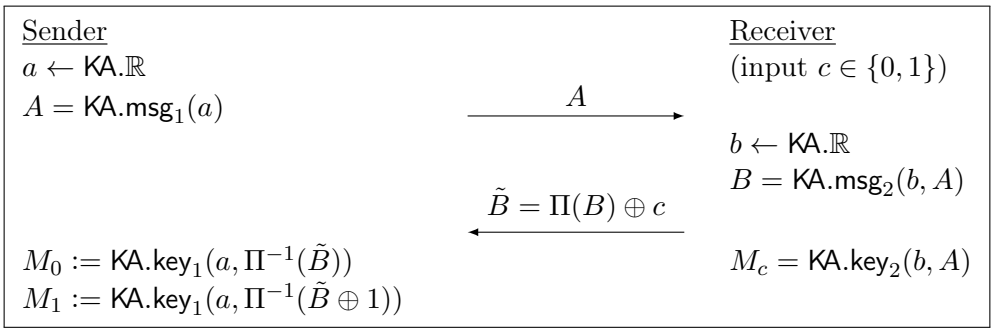


Figure 3.1: Our conceptually simple 1-of-2 random OT protocol, from instantiating [MRR20] with a new "programmable-once public function." $\Pi^{\pm}$ is an ideal permutation and KA is a 2-message key agreement whose "B-messages" are pseudorandom bit strings.

base OTs for certain OT extension protocols. Specifically, we show a serious attack on the 1-out-of-N OT extension protocol of Orrù, Orsini, and Scholl [OOS17], when its base OTs are generated with naïve batching. Unfortunately, we find improper batching (including naïve batching) implemented in several protocol libraries [Rin; CMR; Kel20; Sma] and appearing in several papers [CO15; HL17; CSW20].

Proper batching of base OTs. We then give a complete treatment of how to correctly optimize leading OT protocols for the batch setting. Fortunately, it is simple and cheap to fix naïve batching, although the complete security analysis requires care. We show how to correctly optimize the recent OT protocol of McQuoid, Rosulek, and Roy [MRR20] (hereafter, MRR) for the batch setting. As we show, the Masny-Rindal protocol [MR19] is a special case of the MRR protocol, so our analysis applies to that protocol as well. A comparison of our batched-OT/base-OT protocol to existing work is shown in Table 3.1.

Other improvements. We present several additional improvements to the OT protocol paradigm of McQuoid-Rosulek-Roy (MRR). The MRR protocol can provide 1-out-of-N random-OT, for essentially any N. Modern OT extension protocols require the base OTs to provide only 1-out-of-2 OT. Our optimizations to the MRR approach center around the special case of 1-out-of-2 OT[1] and specific properties of the batch setting.

- The MRR protocol revolves around an object called a **programmable-once public function (POPF)**. A POPF with domain $[N]$ leads to a protocol for 1-out-of-N OT.

[1]Most of our improvements also apply to 1-out-of-N OT, for polynomial N.

In introducing the concept of a POPF, MRR describe a POPF with domain $\{0,1\}^*$, which is useful in some applications but overkill for the special case of 1-out-of-2 OT.

We show several improved POPF constructions for small domains (such as $N = 2$). One particularly interesting and new POPF is in the ideal random permutation model[2] and is inspired by the Even-Mansour block cipher construction [EM93]. When we instantiate MRR with this new POPF, we obtain an endemic OT protocol that is efficient, incredibly simple to describe, and may have pedagogical value as well (Fig. 3.1).

- The MRR protocol constructs OT from a POPF and a key agreement (KA) protocol. These two components must be compatible, and in [MRR20] it was shown how to make elliptic-curve Diffie-Hellman KA compatible with POPFs, by using hash-to-curve operations or Elligator [BHK+13] encoding steps. In this work, we present an alternative approach that avoids using either of these somewhat costly operations, based on a trick due to Möller [Möl04]. Möller-DHKA also avoids curve point addition, allowing us to use Montgomery ladders to multiply, which are more efficient. Adopting the Möller technique requires doubling the length of the sender's protocol message; however, in the batch setting it is exactly this sender's message that is reused across all OT instances in the batch, so the effect of doubling its size is minimal. In our performance benchmark, we found that the Möller technique affords up to a 36% increase in efficiency when batching OTs. This allows for UC secure constructions with comparable runtime to those with standalone security. See Table 3.2.

Finally, we show how our batch OT protocol can be used as the base OTs in **2-round** OT extension.

3.2 Preliminaries

Endemic OT. We use the security definitions for universally composable OT suggested by [MR19] (ideal functionality given in Fig. 3.2), which are a convenient middle-ground between random OT and chosen-message OT. An OT protocol results in outputs r_0, r_1 for the sender and r_c for the receiver (who has choice bit c). In **endemic OT**, a corrupt party may choose their own OT outputs, and all other OT outputs are chosen uniformly by the functionality.

[2]The ideal random permutation model is like the random oracle model, except that all parties have access to a random permutation on $\{0,1\}^{2\lambda}$, *and its inverse!*

Scheme	Assumption	Setup	Flows	Exp (Send/Receive)	Com (Send/Receive)
SimplestOT [CO15]	Gap-CDH	PRO	2	1f $(m+1)$v / mf mv	$1\mathbb{G}$ / $m\mathbb{G}$
BlazingOT [CSW20]	CDH	ORO	3	1f $(m+1)$v / mf mv	$m\lambda + 1\mathbb{G}$ / $2\lambda + m\mathbb{G}$
EndemicOT [MR19]	DDH	PRO	2	$2m$f $2m$v / mf mv	$2m\mathbb{G}$ / $2m\mathbb{G}$
EndemicOT [MR19]	iDDH	PRO	1	mf $2m$v / mf mv	$m\mathbb{G}$ / $2m\mathbb{G}$
Ours (MR)	ODH	PRO	1	2fM $2m$vM / mfM mvM	$2\mathbb{G}$ / $2m\mathbb{G}$
Ours (EKE)	ODH	IC	1	2fM $2m$vM / mfM mvM	$2\mathbb{G}$ / $m\mathbb{G}$
Ours (Feistel)	ODH	PRO	1	2fM $2m$vM / mfM mvM	$2\mathbb{G}$ / $m\mathbb{G}$

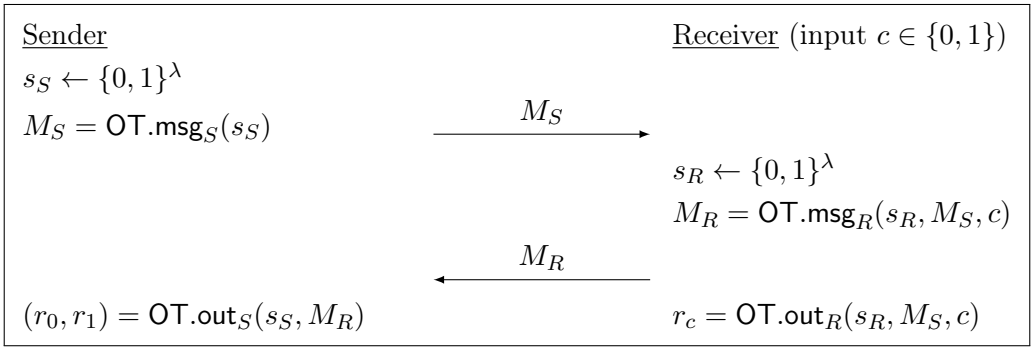
Table 3.1: Comparison of m-instance random 1-of-2 OT protocols. "Exp" denotes exponentiations (f = fixed-base, v = variable-base, fM = fixed-base Montgomery, vM = variable-base Montgomery). "Com" denotes communication ($\mathbb{G}$ = one group element). PRO = programmable random oracle; ORO = observable random oracle; IC = ideal cipher.

Hence, a corrupt sender can choose both r_0 and r_1. A corrupt receiver can choose r_c and the functionality will ensure that r_{1-c} is uniform. As shown in [MR19], OT extension protocols are secure if the base OTs satisfy this notion of endemic OT.

3.3 Problems With Naïve Batching

3.3.1 Naïve Batching

Consider any 2-round protocol for (endemic) OT, with the following syntax:



Where the four functions $\mathsf{OT}.\{\mathsf{msg}, \mathsf{out}\}_{\{S,R\}}$ are abstracted from the raw OT protocol. In such a protocol, the sender's message M_S is clearly independent of the receiver's influence. In many protocols M_S is additionally a message from a KA protocol, and it is well-known

Functionality $\mathcal{F}_{\text{batchEOT}}$

The functionality $\mathcal{F}_{\text{batchEOT}}$ is parameterized by the length of the OT strings ℓ and the number m of OTs in the batch. It interacts with two parties, a sender S and a receiver R via the following queries:

On input $(\text{READY}, (\tilde{r}_{1,0}, \tilde{r}_{1,1}, \ldots, \tilde{r}_{m,0}, \tilde{r}_{m,1}))$ from S, with $\tilde{r}_{i,c} \in \{0,1\}^\ell$:

- If S is corrupt, and there has been no previous READY command from S, then internally record $r_{i,c} = \tilde{r}_{i,c}$ for all $i \in [m]$, $c \in \{0,1\}$. Otherwise do nothing.

On input $(\text{READY}, (c_1, \ldots c_m) \in \{0,1\}^m, (\tilde{r}_1, \ldots, \tilde{r}_m))$ from R, with $\tilde{r}_i \in \{0,1\}^\ell$:

- Do nothing if there has been a previous READY query from R.
- Internally record $(c_1, \ldots c_m)$
- If R is corrupt, then internally record $r_{i,c_i} = \tilde{r}_i$ for each $i \in [m]$.

After receiving READY queries from both S and R:

- For all $i \in [m], c \in \{0,1\}$, if $r_{i,c}$ is not already defined, then sample $r_{i,c} \leftarrow \{0,1\}^\ell$.
- Output $(r_{1,c_1}, \ldots, r_{m,c_m})$ to R and $((r_{1,0}, r_{1,1}), \ldots, (r_{m,0}, r_{m,1}))$ to S.

Figure 3.2: Ideal functionality for a size m batch of endemic 1-out-of-2 oblivious transfers, $\mathcal{F}_{\text{batchEOT}}$. Adapted from the endemic OT functionality of [MR19].

that a KA message can be reused for many KA instances, in certain circumstances. These observations suggest reusing the first OT protocol message in the following way, when generating a batch of m OTs:

Sender		Receiver (inputs $\{c_i\}_{i\in[m]}$)
$s_S \leftarrow \{0,1\}^\lambda$		
$M_S = \mathsf{OT.msg}_S(s_S)$	$\xrightarrow{M_S}$	
		for $i \in [m]$:
		$s_{R,i} \leftarrow \{0,1\}^\lambda$
		$M_{R,i} = \mathsf{OT.msg}_R(s_{R,i}, M_S, c_i)$
	$\xleftarrow{M_{R,1}, \ldots, M_{R,m}}$	
for $i \in [m]$:		for $i \in [m]$:
$(r_{i,0}, r_{i,1}) = \mathsf{OT.out}_S(s_S, M_{R,i})$		$r_{i,c_i} = \mathsf{OT.out}_R(s_{R,i}, M_S, c_i)$

We call this protocol transformation **naïve batching**. All four component functions taken from the base OT protocol will be given the same (sub)session ID because they are treated as a single batch instance. They are reused in such a way that disallows for internal domain separation.

Lemma 3.3.1. *Naïve batching does **not** securely realize batch endemic OT (Fig. 3.2).*

Proof. The attack is simple: a corrupt receiver simply sends $M_{R,1} = \cdots = M_{R,m}$. As a result, the sender must compute $(r_{1,0}, r_{1,1}) = \cdots = (r_{m,0}, r_{m,1})$. There is no way for the simulator to influence the sender's output in this way in the ideal model, hence this constitutes an attack. □

Why not trivially patch this attack? The attack is for the receiver to send the same OT response for all instances. We could simply tell the sender to abort if it receives any repeated OT responses.

However, the simple attack that we have described is only the tip of the iceberg. In all of the 2-round OT protocols that we consider, a corrupt receiver can induce more complicated correlations among the OT values. For example, a receiver can act honestly in the first OT instance to learn $r_{1,0}$. Then $r_{1,1}$ is unknown to the receiver. But there is a more sophisticated strategy for the receiver to force the ratio $r_{1,1}/r_{2,0}$ to be a certain value. (The details of this strategy depend on the details of a specific base OT protocol, so we defer them to Appendix B.1.)

Based on this kind of attack, one might wish to weaken the endemic OT functionality. Why not allow the simulator to specify these kinds of correlations in the ideal model? Even this will not work, because the attack is *perfectly* indistinguishable from honest behavior by the receiver. Thus, there is simply no way for the simulator to distinguish this kind of an attack (where the receiver must learn $r_{1,1}/r_{2,0}$) vs. honest behavior (where the receiver must learn $r_{2,0}$).

For these reasons, we believe there is no way to closely capture the security of naïve batching in a UC ideal functionality.

3.3.2 Implications for OT Extension

Since the main application for batch OTs is as base OTs for OT extension, it is natural to wonder whether the simple attack above jeopardizes the security of OT extension. It has

been established that OT extension can be securely realized from base OTs with weakened security. For example, [CSW20] show that certain input-dependent aborts in the base OTs do not harm the security of OT extension.

We show that our simple attack on naïve batching indeed compromises security of some OT extension protocols. Specifically, we consider the protocol of Orrù, Orsini, and Scholl (OOS) [OOS17]. This OT extension protocol generates many instances of 1-out-of-2^t OT, where in each one the sender obtains $r_1, \ldots, r_{2^t}$ and the receiver learns only r_c, where c is an input. It will be convenient to consider c to be an element of $\{0,1\}^t$ in the natural way.

The OOS protocol is secure when the base OTs securely realize endemic batch OT; see [MR19] for details. However, it loses security when using naïve batching to generate its base OTs.

Lemma 3.3.2. *The OOS protocol [OOS17] is demonstrably insecure when its base OTs are instantiated via naïve batching.*

Proof. The complete details of OOS can be found in [OOS17]. We sketch the relevant details of their protocol here. Let Alice be the OOS sender (with no inputs), and Bob be the OOS receiver (with choice value $c_i \in \{0,1\}^t$ for the ith OT instance). The protocol proceeds as follows:

- The parties run m base OT instances, with Alice acting as receiver and Bob acting as sender. Bob obtains base-OT outputs $(k_{1,0}, k_{1,1}), \ldots, (k_{m,0}, k_{m,1})$. Alice's inputs and outputs are not relevant here.
- When extending to N OTs, Bob constructs two $N \times m$ matrices K and R as follows:
  - The jth column of K is $\mathsf{PRG}(k_{j,0}) \oplus \mathsf{PRG}(k_{j,1})$.
  - The ith row of R is $C(c_i)$ where $C : \{0,1\}^t \to \{0,1\}^m$ is a suitable binary error correcting code (the details of which are not relevant here).

  Bob sends $K \oplus R$ to Alice.

These details of OOS are enough to understand the attack. A corrupt Alice will attack the base OTs (in the role of OT receiver as above) so that all $k_{i,0}$'s are the same and all $k_{i,1}$'s are the same. As a result, every column of K is identical. In other words, every *row* of K is either 0^m or 1^m.

Then the ith row of Bob's matrix $K \oplus R$ is either $C(c_i)$ or its complement. This means that if $c, c' \in \{0,1\}^t$ are any two choices for Bob whose codewords are not bitwise complements of each other, then Alice can distinguish between Bob having choice c vs c' in each extended OT. For some choices of C, learning $C(x)$ up to complement uniquely reveals x. This attack results in almost complete leakage of Bob's private input. □

What if C is a repetition code? C is a binary error-correcting code, the simplest of which is the repetition code $C : \{0,1\} \to \{0,1\}^m$. This corresponds to the case of $t = 1$, and hence 1-out-of-2 OT extension. Specifically, instantiating OOS with a repetition code collapses it to the Keller-Orsini-Scholl 1-out-of-2 OT extension protocol (KOS) [KOS15], at least from the viewpoint of a malicious KOS sender.[3]

In this case the only two codewords are 0^m and 1^m. Since these are bitwise complements of one another, it is not clear that our attack leads to any security problems. The rows of matrix R (encoding Bob's private input) are masked by either 0^m or 1^m, depending on a bit that is unknown to Alice. We are not sure whether a more sophisticated attack on naïvely batched OTs can break KOS OT extension.

Punctured PRFs. More recent OT extensions, such as Silent OT [BCG+19b] and SoftSpokenOT (Chapter 4), do not use their base OTs directly. Instead, they use a technique called punctured OT, which builds a set of punctured PRFs out of the base OTs. Here, naïve batching can cause even more severe issues. The punctured PRFs are built out of a GGM tree, with the top two nodes' values coming from the first OT (see Fig. 4.13). Let the corrupt base OT receiver attack the first base OT associated to each punctured PRF, by sending the same M_R each time, making the corresponding OT messages match across all of these base OTs. Since the whole GGM tree is determined by these values, the two GGM trees will also agree everywhere, and so the punctured PRFs will be identical. The base OT receiver can then choose different choice bits for the other base OTs, which corresponds to picking different places to puncture each PRF. This lets them learn all the unpunctured PRFs, as they are all identical, completely breaking whatever OT extension is based on them.

[3] KOS and OOS have significantly different consistency checks, but their only purpose is to protect against a *malicious receiver*.

3.3.3 Problematic Batching Found in the Wild

Looking ahead, the fix for naïve batching is simple and essentially free (although the security analysis of the fix requires some care, as we show in the next sections). In Diffie-Hellman-based OT protocols, the OT outputs r_0, r_1 are computed by taking a (random oracle) hash of a Diffie-Hellman value. The fix is to include the OT index in that key derivation — *i.e.*, instead of $r_0 = H(sid, g^{ab})$, use $r_0 = H(sid, g^{ab}, \underline{i})$ in the ith OT instance in the batch. That way, even if all g^{ab} values are identical (or correlated strangely), the final OT values are independently random.

Given that both the attack and the fix are so simple, one may wonder whether this problem is well-known. In fact, we found problems related to OT batching in many libraries that implement malicious-secure OT extension. [4] We focus on the implications for the overall OT extension protocols, which are minor in most cases. However, the consequences would be more severe for developers that directly access the base-OT functionalities of these libraries.

- The `libote` OT extension library [Rin] implements Masny-Rindal [MR19] base OTs and applies naïve batching. The original Masny-Rindal paper considers only the single-instance setting and does not discuss security of the batch setting under naïve batching. In some configurations, the `libote` implementation of OOS indeed uses these naïvely batched base OTs, thus falling victim to our attack. Other configurations use a hybrid approach, first naïvely batching 128 base OTs, then using KOS to extend to 512 OTs, and using those 512 OTs as base for OOS. As mentioned above, we are not aware of any attacks on KOS extension related to naïve batching, but our observations merely raise some concerns about its security with naïvely batched OTs.

- The `swanky` MPC library [CMR] implements the Chou-Orlandi protocol and reuses the sender's message, but uses good domain separation in key derivation.[5] However, it allows the sender's protocol message to be reused across several batches, while the domain separation is local to the batch! In other words, parties could execute two batches of OTs, and the receiver could cause the batches to produce identical outputs, by replaying its protocol messages.

[4] We notified the maintainers of these libraries about the issues and the suggested fix. By the time of writing, all maintainers have either already fixed or planned to fix their handling of batch OTs.

[5] The authors explicitly justify their correct key derivation as a bug in the Chou-Orlandi paper, and reference the attack in which all base OTs generate identical output. See `chou_orlandi.rs`.

In this library's implementation of OT extension, they first apply the transformation in [MR19] from endemic OT to uniform-message OT on the base OTs. This prevents the receiver from forcing OT extension to operate on identical base OTs. If not for this additional step, even 1-out-of-2 OT extension would leak information across different batches. As it is, only the XOR of PRG seeds is leaked under our attack on naïve batching, which is unlikely to lead to a concrete attack.

- The `mp-spdz` [Kel20] and `scale-mamba` [Sma] library implementations of OT use naïve batching of Chou-Orlandi base OTs. These libraries implement only KOS and not OOS, and therefore we know of no concrete attack related to naïve batching against their OT extension.

We have also identified problematic handling of OT batching in several papers:

- The Chou-Orlandi OT protocol [CO15] explicitly considers the batch setting and uses naïve batching to achieve it. As such, the protocol as written is not suitable as the base OT for certain OT extensions.

- Since security flaws (unrelated to batching) were discovered in the Chou-Orlandi protocol, several works have attempted to address and repair them. Of those works, both [HL17] and [CSW20] explicitly consider the batch setting. The paper of Hauck & Loss [HL17] maintains the naïve batching of the original.

- The "Blazing OT" construction of Canetti, Sarkar, and Wang [CSW20] does not technically use naïve batching, since it introduces a joint consistency check across all instances in the batch. However, the key derivation in their base OTs does *not* include the OT index. This means that the attack in Theorem 3.3.1 has the intended effect: causing all OT instances to give identical output. The paper only considers a combined protocol with batched Chou-Orlandi base OTs and KOS OT extension, and as such we are not aware of an explicit attack on their final OT extension protocol. However, their security analysis does not seem to acknowledge the possibility of all base OTs giving identical outputs.

We found one instance of totally correct batching, in the implementation of Chou-Orlandi OT in `emp-toolkit` [WMK16], despite naïve batching being described in the paper.

3.4 Properly Batching OTs

In this section we describe how to repair naïve batching. We focus on the McQuoid-Rosulek-Roy (MRR) protocol [MRR20] since it subsumes the Masny-Rindal protocol, while the Chou-Orlandi protocol does not achieve UC security. As we saw, the main problem is that a corrupt receiver can force correlations among the OT outputs in different instances — even causing some OT values to be equal. The solution is to enforce "domain separation" among the different instances. Intuitively, parties should hash each instance's OT outputs under a random oracle, with domain separation (*i.e.*, include the index of that instance in the hash).

However, proving the security of this change requires some care. For example, we cannot prove security merely from the single-instance security of the OT protocol, since the single-instance protocol is not being used correctly. Instead, we must use some known structure of the protocol. The MRR protocol derives its outputs from its underlying KA protocol, and we require stronger properties from that KA. The KA must accept an extra "tag" argument, so that even if the KA messages are identical, the resulting keys will be different under different tags.

3.4.1 Tagged KA

A **tagged KA** is identical in syntax to a traditional KA, except that the $\mathsf{KA.key}_1$ and $\mathsf{KA.key}_2$ algorithms take an additional tag argument. Correctness is that for all $a, b \in \mathsf{KA}.\mathbb{R}$ and all tags τ:

$$\mathsf{KA.key}_1(a, \mathsf{KA.msg}_2(b, \mathsf{KA.msg}_1(a)), \tau) = \mathsf{KA.key}_2(b, \mathsf{KA.msg}_1(a), \tau)$$

Looking ahead to our batch OT protocol, we will let the tag τ be the index of the OT instance (*e.g.*, OT instance 1, 2, 3, ...).

Intuitively, we will require that KA outputs under different tags appear independently random. This should hold not only when the KA protocol messages are identical, but also when the KA messages (*e.g.*, $\mathsf{KA.msg}_2$) are correlated, since we previously observed (Section 3.3) that the adversary could induce arbitrary correlations across OT/KA instances. This definition may be of independent interest—specifically, in scenarios where KA protocol messages are reused.

Definition 3.4.1. *A tagged KA protocol is* ***tag-non-malleable*** *if a session with tag τ^* is secure, even against an eavesdropper that has oracle access to $\mathsf{KA.key}_1(a, \cdot, \cdot)$, provided*

the eavesdropper never queries the oracle on tag τ^*. Formally, the following distributions are indistinguishable, for all τ^* and every PPT $\mathcal{A}$ that never queries its oracle with second argument τ^*:

$a, b \leftarrow \mathsf{KA}.\mathbb{R}$
$M_1 = \mathsf{KA}.\mathsf{msg}_1(a)$
$M_2 = \mathsf{KA}.\mathsf{msg}_2(b, M_1)$
$K = \mathsf{KA}.\mathsf{key}_1(a, M_2, \tau^*)$
return $\mathcal{A}^{\mathsf{KA}.\mathsf{key}_1(a,\cdot,\cdot)}(M_1, M_2, K)$

$a, b \leftarrow \mathsf{KA}.\mathbb{R}$
$M_1 = \mathsf{KA}.\mathsf{msg}_1(a)$
$M_2 = \mathsf{KA}.\mathsf{msg}_2(b, M_1)$
$K \leftarrow \mathsf{KA}.\mathcal{K}$
return $\mathcal{A}^{\mathsf{KA}.\mathsf{key}_1(a,\cdot,\cdot)}(M_1, M_2, K)$

Like [MRR20], we also require the KA protocol to satisfy the following randomness property:

Definition 3.4.2. *A key agreement protocol has **strongly random responses** if the honest output of $\mathsf{KA}.\mathsf{msg}_2$ is indistinguishable from random, even to an adversary who (perhaps maliciously) generated M_1. Formally, for all polynomial time $\mathcal{A}$, the following distributions are indistinguishable:*

$(M_1, state) \leftarrow \mathcal{A}()$
$b \leftarrow \mathsf{KA}.\mathbb{R}$
$M_2 = \mathsf{KA}.\mathsf{msg}_2(b, M_1)$
return $(state, M_2)$

$(M_1, state) \leftarrow \mathcal{A}()$

$M_2 \leftarrow \mathsf{KA}.\mathcal{M}$
return $(state, M_2)$

3.4.2 Programmable-Once Public Functions

The MRR protocol uses a primitive called programmable-once public functions (POPFs). We introduce definitions for POPF here, which slightly differ from the original definitions. We have specialized the definitions for the case of 1-out-of-2 OT [6] — [MRR20] define POPFs in a way that is useful for 1-out-of-N OT (with exponential N) and also password-authenticated key exchange. In the original POPF definitions, a simulator simulated the random oracle setup in the service of a single POPF instance; in our batch setting there will be many POPF instances, thus we must adapt the definitions to explicitly allow simulation of multiple POPFs in a non-interfering way.

[6] All of the POPFs in this paper have straightforward generalizations to the 1-out-of-N case, for polynomial N, and some to exponential N as well, but we restrict ourselves to the 1-out-of-2 case for simplicity.

Definition 3.4.3. *A **1-weak random oracle** is a function $F\colon \mathcal{N} \to \mathcal{O}$ such that the following two distributions are indistinguishable,*

$x \leftarrow \mathcal{N}$	$x \leftarrow \mathcal{N}$
$y := F(x)$	$y \leftarrow \mathcal{O}$
return x, y	return x, y

when the adversary does not have access to F other than through these experiments.

Note that F is only allowed to be used once this definition. This makes it an extremely weak property — it's even satisfied by universal hashes.

Definition 3.4.4 (Syntax)**.** *A **batch 2-way programmable-once public function (batch 2-POPF)** consists of algorithms:*

- $\mathsf{Eval} : \mathcal{M} \times \{0,1\} \to \mathcal{N}$
- $\mathsf{Program} : \{0,1\} \times \mathcal{N} \to \mathcal{M}$

Both algorithms access some local setup $\mathbb{H}$ — depending on the instantiation, $\mathbb{H}$ could consist of common reference strings, random oracles, ideal ciphers, etc. All parties (adversaries) may access the setup directly as well, although it is local to a single instance of the batch 2-POPF. The setup may be stateful (e.g., the "lazy" formulation of a random oracle, which samples outputs on the fly).

A 2-POPF must also include alternative local setups, which are used in different security definitions:

- $\mathbb{H}_{\mathsf{HSim}}$ *must provide the same interface as $\mathbb{H}$ as well as an additional method* $\mathsf{HSim}\colon \mathcal{N} \times \mathcal{N} \to \mathcal{M}$.
- $\mathbb{H}_{\mathsf{Extract}}$ *must provide the same interface as $\mathbb{H}$ as well as an additional method* $\mathsf{Extract}\colon \mathcal{M} \to \{0,1\}$. $\mathsf{Extract}$ *must not modify the private state of $\mathbb{H}_{\mathsf{Extract}}$.*

We write $\mathcal{A}^{\mathbb{H}}$ to denote an algorithm $\mathcal{A}$ with oracle access to all methods provided by the setup $\mathbb{H}$.

Definition 3.4.5 (Correctness)**.** *A batch 2-POPF satisfies **correctness** if $\mathsf{Eval}(\phi, x^*) = y^*$ with all but negligible probability, whenever $\phi \leftarrow \mathsf{Program}(x^*, y^*)$.*

Definition 3.4.6 (Security)**.** *A batch 2-POPF is* ***secure*** *if it satisfies the following properties:*

1. **Indistinguishable Local Setups:** *The local setups* $\mathbb{H}$*,* $\mathbb{H}_{\mathsf{HSim}}$ *and* $\mathbb{H}_{\mathsf{Extract}}$ *all implement a common interface. The setups must be indistinguishable to an adversary that only queries on this interface. Formally, if* $\mathcal{A}$ *is a polynomial-time algorithm that only queries its setup on the interface of* $\mathbb{H}$ *then the following probabilities are negligibly close:*

$$\Pr[\mathcal{A}^{\mathbb{H}}() = 1]; \qquad \Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{HSim}}}() = 1]; \qquad \Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{Extract}}}() = 1]$$

2. **Honest Simulation:** *Any* ϕ *that is generated honestly as* $\phi \leftarrow \mathsf{Program}(x^*, y^*)$*, with* y^* *chosen uniformly, is indistinguishable from* ϕ *generated via the* HSim *algorithm of* $\mathbb{H}_{\mathsf{HSim}}$*. Since* HSim *does not have a "preferred" input* x^**, this establishes that an honestly generated* ϕ *hides the* x^* *on which it was programmed.*

   Formally, define the following functions:

$\underline{\text{REAL\_PHI}(x^* \in \{0,1\}, \mathfrak{D})}$:
$(s, y^*) \leftarrow \mathfrak{D}$
$\phi \leftarrow \mathsf{Program}(x^*, y^*)$
$r_0 := \mathsf{Eval}(\phi, 0)$
$r_1 := \mathsf{Eval}(\phi, 1)$
return s, ϕ, r_0, r_1

$\underline{\text{SIM\_PHI}(x^* \in \{0,1\}, \mathfrak{D})}$:
$(s, y^*) \leftarrow \mathfrak{D}$
$r_{x^*} := y^*$
$r_{1-x^*} \leftarrow \mathcal{N}$
$\phi \leftarrow \mathsf{HSim}(r_0, r_1)$
return s, ϕ, r_0, r_1

   Then for all polynomial time $\mathcal{A}$*,*

$$\Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{HSim}}, \text{REAL\_PHI}}() = 1] - \Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{HSim}}, \text{SIM\_PHI}}() = 1]$$

   is negligible. Here we restrict $\mathcal{A}$ *to always query with* $\mathfrak{D}$ *a distribution over* $\{0,1\}^* \times \mathcal{N}$ *such that the marginal distribution of* y^* *is indistinguishable from the uniform distribution over* $\mathcal{N}$*. The other component* s *appears for technical reasons; the reader can think of it as the coins used to sample* y^**.*

   Note that SIM_PHI *calls the* HSim *method of the local setup, and that* $\mathcal{A}$ *may even query the* HSim *method (both the real and ideal experiments use* $\mathbb{H}_{\mathsf{HSim}}$*).*

3. **Uncontrollable Outputs:** *For any* ϕ *generated by the adversary, the* Extract *method of*

$\mathbb{H}_{\mathsf{Extract}}$ can identify an input x^* such that the adversary has no control over $\mathsf{Eval}(\phi, 1-x^*)$. We say that $\mathsf{Eval}(\phi, 1-x^*)$ is beyond the adversary's control if $F(\mathsf{Eval}(\phi, 1-x^*))$ is indistinguishable from random, for any 1-weak-RO F.[7]

Formally, the following distributions must be indistinguishable for all polynomial-time $\mathcal{A}_1, \mathcal{A}_2$ and all 1-weak-RO F:

$(\phi, state) \leftarrow \mathcal{A}_1^{\mathbb{H}_{\mathsf{Extract}}}()$	$(\phi, state) \leftarrow \mathcal{A}_1^{\mathbb{H}_{\mathsf{Extract}}}()$
$x^* := \mathsf{Extract}(\phi)$	
$r := F(\mathsf{Eval}(\phi, 1-x^*))$	$r \leftarrow \mathcal{N}$
return $\mathcal{A}_2^{\mathbb{H}_{\mathsf{Extract}}}(state, r)$	return $\mathcal{A}_2^{\mathbb{H}_{\mathsf{Extract}}}(state, r)$

As above, the left distribution calls the $\mathsf{Extract}$ method of the $\mathbb{H}_{\mathsf{Extract}}$ setup, and the adversary may query this method as well. Note that $\mathcal{A}$ does not have any access to F beyond the one call provided by this experiment.

The reader may be curious why we forced y^* to be sampled inside Honest Simulation, instead of letting the adversary choose it like in [MRR20]. The answer is that otherwise an ideal cipher would not be a POPF. An adversary could have already run $\mathsf{Program}(0, y^*)$ earlier, and because for each x there is a bijection between values of y and ϕ, a call to $\mathsf{HSim}(y^*, r_1)$ would be forced to return the same ϕ as before. Ideal ciphers were used as a motivating example for POPFs in [MRR20], so this is clearly a mistake. Ideal ciphers satisfy our new definition (Section 3.5.1).

3.4.3 The Batch OT Protocol

In Fig. 3.3 we present the batch variant of the OT protocol of [MRR20]. The protocol is essentially the naïve batching of the single-instance protocol, except we use a tagged KA and use different tags for each KA output.

Theorem 3.4.7. *When instantiated with a secure batch POPF and a tag-non-malleable KA scheme (Theorem 3.4.1) with strongly random responses (Theorem 3.4.2), the OT protocol*

[7] There are 1-weak ROs whose outputs can be distinguished from random when inputs are chosen in a certain adversarial way. Hence, requiring the RO outputs to remain random is a way of requiring that these values are not chosen in an adversarial way.

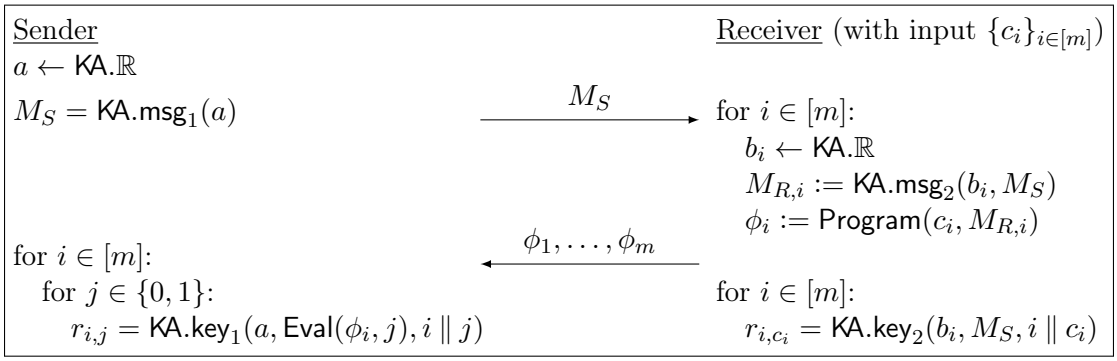


Figure 3.3: Our m-batch 1-of-2 oblivious transfer protocol.

in Fig. 3.3 is a UC secure batch endemic OT (Fig. 3.2), if the POPF's output satisfies $\mathcal{N} = \mathsf{KA}.\mathcal{M}_2$.

Proof. Correctness of the POPF and KA clearly show that the protocol is correct in the case where both parties are honest. When both parties are corrupt, the simulator has direct access to both parties and can simulate the real protocol by just running it. This leaves the two interesting cases, where one party is malicious and the other is honest. We prove each case by giving first a simulator, then a sequence of hybrids showing indistinguishability. The hybrids start from the real world and end at the ideal world: the simulator composed with an ideal batch endemic OT.

Simulator for Malicious Sender: The simulator uses $\mathbb{H}_{\mathsf{HSim}}$ instead of $\mathbb{H}$ to implement the local setup. It then waits until the sender provides its protocol message M_S. It creates fresh random values $b_{i,j} \in \mathsf{KA}.\mathbb{R}$ for $i \in [m], j \in \{0,1\}$, then computes the KA messages $M_{i,j} = \mathsf{KA.msg}_2(b_{i,j}, M_S)$. Then it chooses $\phi_i \leftarrow \mathsf{HSim}(M_{i,0}, M_{i,1})$ and sends $\phi_1, \ldots, \phi_m$ as the simulated protocol message from the honest receiver. Finally, it submits $r_{i,j} = \mathsf{KA.key}_2(b_{i,j}, M_S, i \parallel j)$ to the ideal functionality, for $i \in [m]$ and $j \in \{0,1\}$ (as the endemic OT values).

Sequence of Hybrids for Malicious Sender: Starting at the real interaction between malicious sender and honest receiver:

1. Replace local setup $\mathbb{H}$ with $\mathbb{H}_{\mathsf{HSim}}$. This change is indistinguishable by the Indistinguishable Local Setups property of the POPF.

2. Change how ϕ_i is generated:

$$\text{replace} \boxed{\begin{array}{l} b_i \leftarrow \mathsf{KA}.\mathbb{R} \\ M_{R,i} = \mathsf{KA.msg}_2(b_i, M_S) \\ \phi_i \leftarrow \mathsf{Program}(c_i, M_{R,i}) \end{array}} \text{with} \boxed{\begin{array}{l} b_i \leftarrow \mathsf{KA}.\mathbb{R} \\ M_{i,c_i} = \mathsf{KA.msg}_2(b_i, M_S) \\ M_{i,1-c_i} \leftarrow \mathsf{KA}.\mathcal{M} \\ \phi_i \leftarrow \mathsf{HSim}(M_{i,0}, M_{i,1}) \end{array}}$$

   This is indistinguishable by the Honest Simulation property. Recall that this property requires b_i, M_{i,c_i} to come from a distribution $\mathfrak{D}$ over $\{0,1\}^* \times \mathcal{N}$ where the marginal distribution of the second element is indistinguishable from uniform. This holds because KA has strongly random responses.

3. Change how $M_{i,1-c_i}$ is sampled:

$$\text{replace} \boxed{\begin{array}{l} b_i \leftarrow \mathsf{KA}.\mathbb{R} \\ M_{i,c_i} = \mathsf{KA.msg}_2(b_i, M_S) \\ M_{i,1-c_i} \leftarrow \mathsf{KA}.\mathcal{M} \\ \phi_i \leftarrow \mathsf{HSim}(M_{i,0}, M_{i,1}) \end{array}} \text{with} \boxed{\begin{array}{l} b_{i,0}, b_{i,1} \leftarrow \mathsf{KA}.\mathbb{R} \\ M_{i,0} = \mathsf{KA.msg}_2(b_{i,0}, M_S) \\ M_{i,1} = \mathsf{KA.msg}_2(b_{i,1}, M_S) \\ \phi_i \leftarrow \mathsf{HSim}(M_{i,0}, M_{i,1}) \end{array}}$$

   Later references to b_i become references to b_{i,c_i}. This is indistinguishable because KA has strongly random responses.

This final hybrid describes the ideal world. The receiver's inputs c_i are not used to simulate protocol messages to the sender; they are used only to determine which $r_{i,j} \stackrel{\text{def}}{=} \mathsf{KA.key}_2(b_{i,j}, M_S)$ the receiver takes as output. In the ideal world the simulator sends identically defined $r_{i,j}$ to the ideal functionality, which uses the receiver's c_i inputs to determine which ones to deliver as the receiver's output.

Simulator for Malicious Receiver: The simulator uses $\mathbb{H}_{\mathsf{Extract}}$ instead of $\mathbb{H}$ to implement the local setup. It generates M_S in the same way as an honest sender and sends it to the corrupted receiver. When the receiver provides $\phi_1, \ldots, \phi_m$, the simulator runs $c_i = \mathsf{Extract}(\phi_i)$ for all $i \in [m]$, and submits them to the ideal functionality. It also computes $r_{i,c_i} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, c_i), i \parallel c_i)$, and submits these to the ideal functionality as well (as the endemic OT values).

Sequence of Hybrids for Malicious Receiver:

1. Replace local setup $\mathbb{H}$ with $\mathbb{H}_{\mathsf{Extract}}$, an indistinguishable change.

2. Rearrange how $r_{i,j}$ are computed:

$$\text{replace} \boxed{\begin{array}{l}\text{for } j \in \{0,1\}\text{:} \\ \quad r_{i,j} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, j), i \,\|\, j)\end{array}}$$

$$\text{with} \boxed{\begin{array}{l}c_i \leftarrow \mathsf{Extract}(\phi_i) \\ r_{i,c_i} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, c_i), i \,\|\, c_i) \\ r_{i,1-c_i} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, 1-c_i), i \,\|\, 1-c_i)\end{array}}$$

   This is indistinguishable because running $\mathsf{Extract}$ has no effect on the local setup's internal state.

3. For each $i \in [m]$ and $j \in \{0,1\}$, create an oracle $F_{i,j} = y \mapsto \mathsf{KA.key}_1(a, y, i \,\|\, j)$. Then rewrite the computation of $r_{i,j}$ in terms of these oracles as $r_{i,j} = F_{i,j}(\mathsf{Eval}(\phi_i, j))$. In Theorem 3.4.8 we show that every oracle $F_{i,j}$ is a 1-weak random oracle.

4. Change how $r_{i,1-c_i}$ is chosen:

$$\text{replace} \boxed{\begin{array}{l}c_i \leftarrow \mathsf{Extract}(\phi_i) \\ r_{i,c_i} = F_{i,c_i}(\mathsf{Eval}(\phi_i, c_i)) \\ r_{i,1-c_i} = F_{i,c_i-1}(\mathsf{Eval}(\phi_i, 1-c_i))\end{array}} \text{with} \boxed{\begin{array}{l}c_i \leftarrow \mathsf{Extract}(\phi_i) \\ r_{i,c_i} = F_{i,c_i}(\mathsf{Eval}(\phi_i, c_i)) \\ r_{i,1-c_i} \leftarrow \mathsf{KA}.\mathcal{K}\end{array}}$$

   This change is indistinguishable by the Uncontrollable Outputs property. Since each $F_{i,j}$ is a 1-weak RO, we can apply the Uncontrollable Outputs property once for each i to make the change described here.

This final hybrid describes the ideal world. After seeing the receiver's protocol message, the simulator extracts c_i values and also computes values r_{i,c_i} which will be part of the sender's output. The other OT values in the sender's output ($r_{i,1-c_i}$) are sampled uniformly, just as in the ideal world. □

Lemma 3.4.8. *For any tag-non-malleable key agreement KA with strongly random responses, and for any set of tags $\mathcal{T}$, the following distribution outputs a key agreement message and a*

collection of $|\mathcal{T}|$ weak random oracles from $\mathsf{KA}.\mathcal{M}_2$ to $\mathsf{KA}.\mathcal{K}$.

$a \leftarrow \mathsf{KA}.\mathbb{R}$
$M_S := \mathsf{KA.msg}_1(a)$
for $\tau \in \mathcal{T}$:
  $F_\tau := x \mapsto \mathsf{KA.key}_1(a, x, \tau)$
return $M_S, \{F_\tau\}_{\tau \in \mathcal{T}}$

Proof. We need to show that every F_τ is a weak random oracle. We describe a sequence of hybrids starting from the real weak random oracle distribution and ending at random.

1. Sample the input x and compute y early, when the oracle F_τ is created, rather than when the weak RO experiment is run.

2. Instead of sampling $x \leftarrow \mathsf{KA}.\mathcal{M}_2$, sample $b \leftarrow \mathsf{KA}.\mathbb{R}$ and set $x = \mathsf{KA.msg}_2(b, M_S)$. This is indistinguishable by the strongly random responses property of KA.

3. We are now computing $y = \mathsf{KA.key}_1(a, x, \tau)$ for a random KA message x, then giving oracle access to $\mathsf{KA.key}_1(a, x', \tau')$ (from the other oracles $F_{\tau'}$), but only for $\tau' \neq \tau$. This is exactly the same as the real distribution for a tag-non-malleable KA, so it is indistinguishable to switch to the random distribution by randomly sampling $y \leftarrow \mathsf{KA}.\mathcal{K}$ instead.

4. Use strongly random responses again, to sample $x \leftarrow \mathsf{KA}.\mathcal{M}_2$ and remove b.

5. Delay the sampling of x, y until the 1-weak RO distribution is run.

□

Our protocol considers an underlying KA with sequential messages. Yet Diffie-Hellman-based KA protocols have independent messages that can be sent in any order. We call such a KA protocol **1-flow**, where $\mathsf{KA.msg}_2(b)$ is independent of M_S. When the KA is 1-flow, the OT protocol can also be made 1-flow by sending both messages in parallel.

Theorem 3.4.9. *Our OT protocol (Fig. 3.3) becomes a 1-flow UC secure batch endemic OT when KA is 1-flow.*

Proof. This theorem largely the same as Theorem 3.4.7 from the previous one, but with key changes. In the 1-flow instance, the adversary may rush the other party, requiring them to send their message first before responding. For malicious receiver the adversary already went last, but it's different for malicious sender.

When the sender is corrupt, the simulator instead generates $\phi_1, \ldots, \phi_m$ with HSim before receiving M_S, as each of the receiver's messages from the key agreement may now be sampled independently of the sender's. The hybrid proof continues as before, after replacing $\mathsf{KA.msg}_2(b, M_S)$ with $\mathsf{KA.msg}_2(b)$. □

3.5 New/Improved POPF Constructions

In this section, we describe several suitable POPF constructions for the batch OT protocol.

3.5.1 Ideal Cipher (EKE)

Our first POPF is inspired by the EKE password-authenticated key exchange protocol of Bellovin & Merritt [BM92]. POPF was created as a generalization of an ideal cipher in the EKE protocol, and it is no surprise that in fact an ideal cipher is a POPF. The full definition is in Fig. 3.4. We are not aware of prior work pointing out the connection between EKE and oblivious transfer. But it is easy to see that an ideal cipher is useful for OT: the adversary can know the trapdoor to at most one of $E^{-1}(0, \phi)$ and $E^{-1}(1, \phi)$.

The local setup $\mathbb{H}$ is simply an ideal cipher. Actually, we have defined $\mathbb{H}$ in a way that is indistinguishable from an ideal cipher — it chooses oracle responses uniformly, instead guaranteeing that each $E(x, \cdot)$ is a permutation. By a standard PRF/PRP switching lemma, the difference is indistinguishable, and this choice makes the description of $\mathbb{H}$ simpler. $\mathbb{H}_{\mathsf{HSim}}$ is similar to $\mathbb{H}$, but it programs E^{-1} so that $\mathsf{Eval}(\phi, i) = r_i$, to satisfy the honest simulation property.

In $\mathbb{H}_{\mathsf{Extract}}$, $\mathsf{Extract}(\phi)$ finds the first ideal cipher call that produced ϕ — either as the input to an E^{-1} query or the output of an E query. The idea is that once ϕ has appeared in some ideal cipher query, future forward queries to E give output ϕ only with negligible probability. Hence, all future calls that involve ϕ must be of the form $E^{-1}(\cdot, \phi)$, meaning that the adversary has no control over the outputs of these queries (which are outputs of Eval). This is precisely the property needed for a POPF.

$\mathcal{M} := \mathcal{N}$

$\underline{\mathsf{Program}(x, y)\text{:}}$
  return $E(x, y)$

$\underline{\mathsf{Eval}(\phi, x)\text{:}}$
  return $E^{-1}(x, \phi)$

$\mathbb{H}$

$T :=$ empty list
$\underline{E(x, y)\text{:}}$
  if $\exists \phi.\, (x, y, \phi) \in T$:
    return ϕ
  $\phi \leftarrow \mathcal{M}$
  append (x, y, ϕ) to T
  return ϕ
$\underline{E^{-1}(x, \phi)\text{:}}$
  if $\exists y.\, (x, y, \phi) \in T$:
    return y
  $y \leftarrow \mathcal{N}$
  append (x, y, ϕ) to T
  return y

$\mathbb{H}_{\mathsf{HSim}}$

$T := \{\}$
// E and E^{-1} are same as in $\mathbb{H}$
$\underline{\mathsf{HSim}(r_0, r_1)\text{:}}$
  if $\exists x, \phi.\, (x, r_x, \phi) \in T$:
    return $\perp$
  $\phi \leftarrow \mathcal{M}$
  append $(0, r_0, \phi)$ to T
  append $(1, r_1, \phi)$ to T
  return ϕ

$\mathbb{H}_{\mathsf{Extract}}$

$T := \{\}$
// E and E^{-1} are same as in $\mathbb{H}$
$\underline{\mathsf{Extract}(\phi)\text{:}}$
  find first $(x^*, y^*, \phi) \in T$:
    return x^*
  if none exist:
    return 0

Figure 3.4: Batch 2-POPF based on an ideal cipher.

Theorem 3.5.1. *Fig. 3.4 defines a secure and correct batch 2-POPF with all distinguisher advantages except for Uncontrollable Outputs bounded by* $O\left(\frac{q^2}{|\mathcal{N}|}\right)$*, when the adversary makes* q *ideal cipher lookups. Uncontrollable Outputs instead has advantage bounded by* $q\mathcal{A}(wRO) + O\left(\frac{q^2}{|\mathcal{N}|}\right)$*, where* $\mathcal{A}(wRO)$ *is the distinguisher advantage against the 1-weak RO* F*.*

Proof. We have deferred the security proofs for the POPF constructions to the appendix. See Appendix B.2.1. □

$\mathcal{M} := \mathcal{N} := \{0,1\}^\alpha$

$\underline{\mathsf{Program}(x, y)}$:
  return $\Pi(y) \oplus x$

$\underline{\mathsf{Eval}(\phi, x)}$:
  return $\Pi^{-1}(\phi \oplus x)$

$\mathbb{H}$

$T :=$ empty list
$\underline{\Pi(u)}$:
  if $\exists v.\, (u, v) \in T$:
    return v
  $v \leftarrow \{0,1\}^\alpha$
  append (u, v) to T
  return v
$\underline{\Pi^{-1}(v)}$:
  if $\exists u.\, (u, v) \in T$:
    return u
  $u \leftarrow \{0,1\}^\alpha$
  append (u, v) to T
  return u

$\mathbb{H}_{\mathsf{HSim}}$

$T :=$ empty list
// E and E^{-1} are same as in $\mathbb{H}$
$\underline{\mathsf{HSim}(r_0, r_1)}$:
  if $\exists x, \phi.\, (r_x, \phi \oplus x) \in T$:
    return $\perp$
  $\phi \leftarrow \{0,1\}^\alpha$
  append $(r_0, \phi \oplus 0)$ to T
  append $(r_1, \phi \oplus 1)$ to T
  return ϕ

$\mathbb{H}_{\mathsf{Extract}}$

$T :=$ empty list
// E and E^{-1} are same as in $\mathbb{H}$
$\underline{\mathsf{Extract}(\phi)}$:
  find first $(y^*, \phi \oplus x^*) \in T$:
    return x^*
  if none exist:
    return 0

Figure 3.5: Batch 2-POPF based on an ideal permutation.

3.5.2 Even-Mansour POPF

In [MRR20] the authors construct a POPF with a 2-round Feistel cipher. Intuitively, a POPF generalizes an ideal cipher, but is strictly weaker. So, while an 8-round Feistel cipher is indifferentiable from an ideal cipher, a 2-round Feistel cipher suffices for a POPF. Similarly, we suggest a POPF based on the Even-Mansour [EM93] construction. While the Even-Mansour construction is not an ideal cipher unless many rounds are added [DSS+17], a single round suffices for a POPF.

The construction (Fig. 3.5) is similar to the Ideal Cipher POPF, but with a few changes.

The local setup $\mathbb{H}$ is not an ideal cipher, but a simpler ideal random permutation. In the ideal cipher POPF, every query to the oracles included the x-value (as the key of the cipher). In this Even-Mansour POPF the value x is used only by xor'ing with the ideal permutation output — it is not directly available to the simulator (in $\mathsf{Extract}$).

To deal with this challenge, we observe that x can be inferred by the simulator given ϕ. The only situation where x is ambiguous given ϕ is when $\Pi(y_1) \oplus x_1 = \phi = \Pi(y_2) \oplus x_2$ for distinct bits x_1, x_2. This event implies $\Pi(y_1) \oplus \Pi(y_2) = x_1 \oplus x_2 = 1$, which is negligibly likely for forward queries to Π. This turns out to be enough for the simulator to extract. The construction generalizes to strings x which are significantly shorter than the ideal permutation output.

Theorem 3.5.2. *Fig. 3.5 defines a secure and correct batch 2-POPF where the distinguisher advantage is $O(q^2 2^{-\alpha})$ when the adversary makes q ideal permutation lookups, except for Uncontrollable Outputs which allows an additional advantage of $q\mathcal{A}(wRO)$.*

Proof. We have deferred this proof to Appendix B.2.2. □

3.5.3 Masny-Rindal POPF

This next POPF is inspired by the OT construction of Masny and Rindal [MR19]. Using this POPF in the context of Fig. 3.3 we see that the Masny-Rindal OT protocol for 1-out-of-2 OT[8] is then a specific instance of our protocol. The description of the POPF can be found in Fig. 3.6.

The local setup $\mathbb{H}$ consists of two random oracles H_0, H_1 whose outputs are a group $\mathbb{G}$. In the resulting OT protocol, the KA scheme must have protocol messages that reside in this group. $\mathbb{H}_{\mathsf{HSim}}$ is similar to $\mathbb{H}$, but it also tracks the values r_0, r_1 that have been given to $\mathsf{HSim}(R)$. To satisfy the honest simulation property, it further programs the random oracles H_x to be consistent:

$$\mathsf{Eval}(\phi, x) = s_x \cdot H_x(s_{1-x}) = s_x \cdot (s_x)^{-1} \cdot r_x = r_x.$$

$\mathbb{H}_{\mathsf{Extract}}$ is also very similar to $\mathbb{H}$, but it also tracks chronological order of the oracle queries. $\mathsf{Extract}(\phi)$, upon seeing $\phi = (s_0, s_1)$, checks if s_{1-x^*} was a query to the random

[8] Generalizing to 1-out-of-N for polynomial N works the same as in [MR19].

$\mathcal{M} := \mathbb{G}^2$

$\underline{\mathsf{Program}(x, y)}$:
  $s_{1-x} \leftarrow \mathbb{G}$
  $s_x = y \cdot (H_x(s_{1-x}))^{-1}$
  return (s_0, s_1)

$\underline{\mathsf{Eval}((s_0, s_1), x)}$:
  return $s_x \cdot H_x(s_{1-x})$

$\mathbb{H}$

record calls in a transcript T
$\underline{H_x(u)}$:
  if $\exists v.\, (v \leftarrow H_x(u)) \in T$:
    return v
  $v \leftarrow \mathbb{G}$
  return v

$\mathbb{H}_{\mathsf{HSim}}$

record calls in a transcript T
$U :=$ empty assoc. array
$\underline{H_x(u)}$:
  if $\exists v.\, (v \leftarrow H_x(u)) \in T$:
    return v
  if $U[x, u]$ defined:
    return $U[x, u]$
  $v \leftarrow \mathbb{G}$
  return v
$\underline{\mathsf{HSim}(r_0, r_1)}$:
  $\phi = (s_0, s_1) \leftarrow \mathcal{M}$
  $U[0, s_1] := s_0^{-1} \cdot r_0$
  $U[1, s_0] := s_1^{-1} \cdot r_1$
  return ϕ

$\mathbb{H}_{\mathsf{Extract}}$

record calls in a transcript T
// H_x is the same as in $\mathbb{H}$
$\underline{\mathsf{Extract}((s_0, s_1))}$:
  find first query $H_{x^*}(s_{1-x^*})$ in T:
    return x^*
  if none exist:
    return 0

Figure 3.6: Batch 2-POPF based on the OT construction of Masny-Rindal [MR19]. Here $H_0, H_1 : \mathbb{G} \to \mathbb{G}$ are random oracles, and $(\mathbb{G}, \cdot)$ is a group.

oracle H_{x^*}, for either $x^* \in \{0, 1\}$. $\mathsf{Extract}(\phi)$ then chooses the first query (chronologically) and returns the associated x^*, or chooses x^* arbitrarily to be 0 if neither call was made. As in the original proof in [MR19] the main idea is that for the adversary to program ϕ, they need to query on one of the two s_x values to find the other, unless the "other" is sampled independently, in which case the adversary fails to program.

Theorem 3.5.3. *Fig. 3.6 defines a secure and correct batch 2-POPF with all distinguisher advantages except for Uncontrollable Outputs bounded by* $O\left(\frac{q^2}{|\mathbb{G}|}\right)$ *when the adversary makes* q *queries to the random oracles. Uncontrollable Outputs instead has advantage bounded by*

$\mathcal{N} := \mathbb{G}$
$\mathcal{M} := \mathbb{G} \times \mathbb{F}$

$\underline{\mathsf{Program}(x, y)}$:
  $u \leftarrow \mathbb{F}$
  $t := H(x, u)^{-1} \cdot y$
  $s := u - \iota(t)x$
  return s, t

$\underline{\mathsf{Eval}((s, t), x)}$:
  return $H(x, \iota(t)x + s) \cdot t$

$\mathbb{H}$

record calls in a transcript T
$\underline{H(x, u)}$:
  if $\exists v.\, (v \leftarrow H(x, u)) \in T$:
    return v
  $v \leftarrow \mathbb{G}$
  return v

$\mathbb{H}_{\mathsf{HSim}}$

record calls in a transcript T
$U :=$ empty assoc. array
$\underline{H(x, u)}$:
  if $\exists v.\, (v \leftarrow H(x, u)) \in T$:
    return v
  if $U[x, u]$ defined:
    return $U[x, u]$
  $v \leftarrow \mathbb{G}$
  return v
$\underline{\mathsf{HSim}(r_0, r_1)}$:
  $(s, t) \leftarrow \mathbb{G} \times \mathbb{F}$
  $U[0, \iota(t)\,0 + s] := r_0 \cdot t^{-1}$
  $U[1, \iota(t)\,1 + s] := r_1 \cdot t^{-1}$
  return (s, t)

$\mathbb{H}_{\mathsf{Extract}}$

record calls in a transcript T
// H is the same as in $\mathbb{H}$
$\underline{\mathsf{Extract}((s, t))}$:
  find first query $H(x^*, \iota(t)x^* + s)$:
    return x^*
  if none exist:
    return 0

Figure 3.7: Variant of the Feistel POPF in [MRR20], where one random oracle has been replaced with multiplication in a finite field $\mathbb{F}$. ι is an injection with an efficient left inverse ι^{-1}, *i.e.*, $\forall t.\, \iota^{-1}(\iota(t)) = t$.

$\frac{q^2-q+2}{2}\mathcal{A}(wRO) + O\left(\frac{q^2}{|\mathbb{G}|}\right)$.

Proof. We have deferred this proof to Appendix B.2.3. □

3.5.4 Streamlined Feistel POPF

[MRR20] propose a POPF based on 2-round Feistel, in which the ϕ value is 3λ bits longer than the underlying value from $\mathcal{N}$. We present an alternative construction (Fig. 3.7) that improves on this when $\mathbb{G} = \mathcal{N}$ can be represented with less than 3λ bits. This is useful because elliptic curve points usually can be represented with 2λ bits.

As with [MRR20], we need $\mathcal{N}$ to be a group $\mathbb{G}$, and the local setup $\mathbb{H}$ is a hash function H mapping into $\mathbb{G}$. However, instead of a second random oracle $H'(x, T)$, we use an injection ι from $\mathbb{G}$ into a finite field $\mathbb{F}$. The hash call $H'(x, T)$ in one of the Feistel rounds is then replaced with multiplication $\iota(T)x$. ι is required to have an efficiently computable left inverse ι^{-1}.

These changes eliminate the main bad event in the security proof of [MRR20], which occurs when the adversary manages to delay making the H' query, which the simulator needs to see in order to find what T the adversary chose, until after the simulator needs to use T to program H. The simulator can now find T directly using ι^{-1}.

Theorem 3.5.4. *The streamlined Feistel POPF in Fig. 3.7 is a secure and correct batch 2-POPF. The distinguisher advantage is* $O\left(\frac{q^2}{|\mathbb{G}|}\right)$ *when the adversary makes* q *ideal permutation lookups, except for Uncontrollable Outputs which allows an additional advantage of* $\frac{q^2-q+2}{2}\mathcal{A}(wRO)$.

Proof. We have deferred this proof to Appendix B.2.4. □

The original 2-round Feistel POPF in [MRR20] also satisfies our new definitions. We omit the proof because it is substantially similar to the proof of Theorem 3.5.4, just preserving a few more ideas from [MRR20].

3.6 Suitable Key Agreement Choices

Our batched OT protocol requires a tagged KA in which the receiver's protocol messages are indistinguishable from the uniform distribution over the domain of the POPF (outputs of Eval). In this section we discuss several choices for KA, including one not considered in [MRR20] but which is well-suited to the batch setting.

The main challenge is that traditional DHKA on an elliptic curve is not enough. Under the usual encoding (the x-coordinate), points on the curve are easily distinguishable from

random strings, while it is more natural to define a POPF operating on strings. Hence, some care is involved in making the POPF and KA compatible.

3.6.1 Curve Mappings

In [MRR20], the authors suggest two ways to achieve compatibility between POPF and KA over elliptic curves.

One choice is to ensure that the KA protocol messages are uniform *bit strings.* This can be done using the Elligator technique of [BHK+13] to encode curve elements. Elligator is an injective and efficiently invertible function ι from $\{0,1\}^\lambda$ to a large subset of the elliptic curve. If some party wishes to make their KA protocol message a uniform string, they simply sample from points in the image of ι. This is achieved in practice by re-sampling a DH scalar until the resulting curve point is in $\iota(\{0,1\}^\lambda)$. If the range of ι is a large fraction of the elliptic curve, then the expected number of re-samples is small. See Fig. 3.8 for a formal description of tagged Elligator ECDHKA.

Another choice is to ensure that the POPF Eval function only outputs values on the curve. In the POPF construction of [MRR20] this can be achieved by instantiating a random oracle that gives outputs in the curve.

These techniques incur nontrivial computational overhead. The Elligator approach requires resampling each curve element some constant number of times on average. The state-of-the-art techniques for hashing-to-curve [BCI+10; FFS+10; TK17] have cost roughly 25% that of a scalar multiplication on the curve, and the POPF requires at least 2 hash-to-curve operations per party.

3.6.2 Möller Variant of ECDHKA

We now suggest a more efficient approach that is well suited for the batch setting. Before continuing, let us give a brief review of elliptic curves. For the remainder of this section, we will consider curves over prime fields with order larger than 3. Further results and descriptions can be found in Silverman [Sil09].

Definition 3.6.1. *An **elliptic curve** $E_{a,b}$ over a field $\mathbb{F}_p$ is defined by a congruence of the form $Y^2 = X^3 + aX + b$ parameterized by elements $a, b \in F_p$ such that $4a^3 + 27b^2 \neq 0$. The elements of $E_{a,b}$ are given by tuples (X, Y) satisfying the congruence along with a neutral*

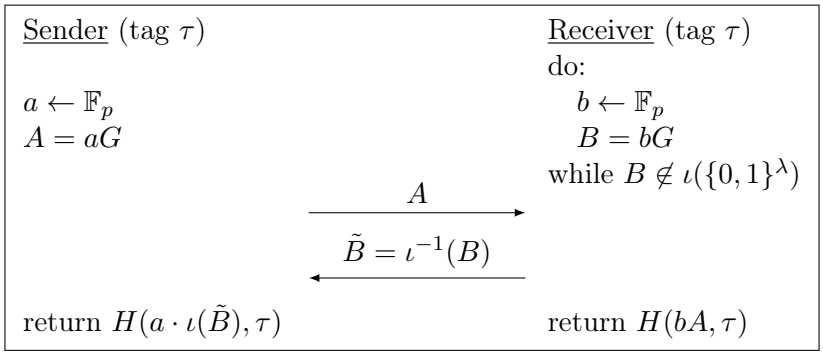


Figure 3.8: Tagged Elligator ECDHKA. G is a generator of the curve and ι is the injective Elligator mapping of [BHK+13].

*element $\mathcal{O}$, the **point at infinity**.*

We may equip this set with a group law called the chord-and-tangent law such that we arrive at a commutative group where the usual Diffie-Hellman problems are believed to be hard.

Definition 3.6.2. *Given an elliptic curve $E_{a,b}$ over a field $\mathbb{F}_p$ and $c \in \mathbb{F}_p$, we may consider the elliptic curve $E'_c : cY^2 = X^3 + aX + b$. If c is a quadratic residue in $\mathbb{F}_p$ then E' is isomorphic to E, otherwise, E' is called the (quadratic) **twist** of E.*

As a twist of a given curve is unique up to isomorphism, we may consider, singly, a primary curve and its twist curve. It follows from the definition that any $x \in \mathbb{F}_p$ is the abscissa (x-coordinate) of a point on E or of a point on the twist E'.

Lemma 3.6.3. *Let $c \neq 0$ be a quadratic non-residue in the field $\mathbb{F}_p$, and let $E_{a,b}$ be an elliptic curve over $\mathbb{F}_p$ with twist E'_c. Then for every $x \in \mathbb{F}_p$:*

1. *If $x^3 + ax + b$ is a non-zero quadratic residue, then $(x, \pm\sqrt{x^3 + ax + b})$ are points on $E_{a,b}$. Furthermore, $(x^3 + ax + b)/c$ is a quadratic non-residue and x is not the abscissa of any point on E'_c*

2. *If $x^3 + ax + b$ is a quadratic non-residue, then x is not a point on $E_{a,b}$. Furthermore, $(x^3 + ax + b)/c$ is a quadratic residue and $(x, \pm\sqrt{(x^3 + ax + b)/c})$ are points on E'_c.*

3. *If $x^3 + ax + b = 0$, then $(x, 0)$ is a point on $E_{a,b}$ and E'_c.*

This idea is of importance as for many curves and applications; only the abscissa of a point is needed. This means that we can work with bitstrings using the implicit mapping defined above.

Furthermore, there are a similar number of points on the twist as there are on the curve. If one were to toss a coin $b \leftarrow \{0,1\}$, and then sample an x-coordinate of a random curve point (if $b = 0$) or a random twist point (if $b = 1$), the result would be statistically close to the uniform distribution on the set of bitstrings.

Lemma 3.6.4 ([CFG+06, Corollary 11])**.** *Given a curve $E_{a,b}$ and its twist E'_c over $\mathbb{F}_p$, where $2^q - p < 2^{q/2}$ (i.e., p is very close to a power of 2), the following distribution is indistinguishable from the uniform distribution in $\{0,1\}^q$*

$$\mathcal{D} = \{\beta \leftarrow \{0,1\}, x_0 \leftarrow [E_{a,b}]_{abscissa}, x_1 \leftarrow [E'_c]_{abscissa} : K = x_\beta\},$$

with statistical distance

$$\delta = \frac{1}{2} \sum_{x \in \mathbb{F}_p} \left| \Pr_{K \leftarrow \mathbb{F}_{2^q}}[K = x] - \Pr_{K \leftarrow \mathcal{D}}[K = x] \right| \leq \frac{1 + \sqrt{2}}{2^{q/2}}.$$

This suggests the key agreement approach in Fig. 3.9. The receiver will sample an x-coordinate as above. The sender cannot anticipate the receiver's choice, so she prepares a DH message on both the curve and the twist, then chooses the correct one to compute the final key. Theorem 3.6.4 establishes that the receiver's KA message is statistically indistinguishable from the uniform distribution on strings.

Note that the sender sends two curve/twist elements instead just one as in standard DHKA. However, in batched OT it is exactly this sender message that is reused across all OT instances. Hence a slight increase in its size has minimal effect on the overall OT protocol's efficiency.

Similar approaches to representation have been used in the context of PAKE [BMN01], pseudo-random permutations [Kal91], authenticated key exchange [CFG+06], and by Möller [Möl04] in the context of ElGamal.

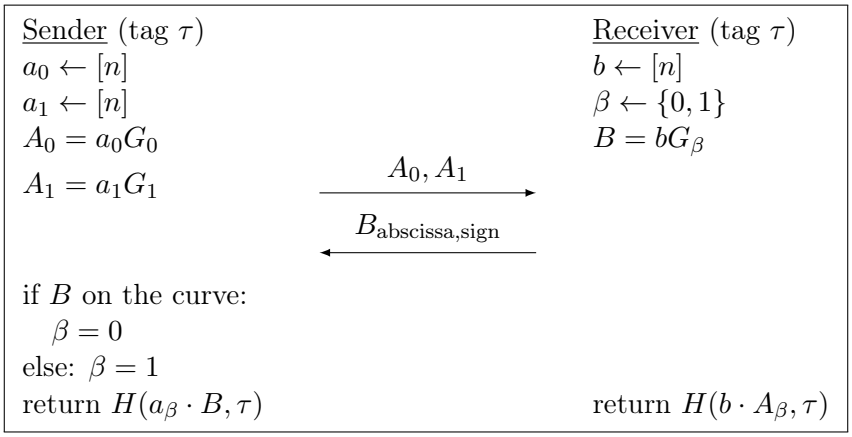


Figure 3.9: Möller tagged ECDHKA. G_0 is a generator of the curve and G_1 is a generator of its twist.

3.6.3 Curve Choice and Security

We now discuss the security of the Möller variant (tagged) KA protocol. The choice of curve must satisfy the following

- The finite field must have order at least $2^q - 2^{q/2}$.
- The curve and its twist must be cryptographically secure.
- The curve and its twist must be cyclic.

More specifically, we need a security property similar to the **oracle Diffie-Hellman (ODH)** assumption [ABR01]. That definition is as follows:

Definition 3.6.5 ([ABR01])**.** *Let $\mathbb{G}$ be a cyclic group of order n, with generator g, and let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. Then the **oracle Diffie-Hellman (ODH)** assumption holds in $\mathbb{G}$ with respect to H if the following distributions are indistinguishable, for all $\mathcal{A}$ that do not query their oracle at bg.*

$a, b \leftarrow [n]$ def $\mathcal{H}_a(X) = H(aX)$ $K = H(abg)$ return $\mathcal{A}^{\mathcal{H}_a}(ag, bg, K)$	$a, b \leftarrow [n]$ def $\mathcal{H}_a(X) = H(aX)$ $K \leftarrow \{0,1\}^\ell$ return $\mathcal{A}^{\mathcal{H}_a}(ag, bg, K)$

Our applications require a variant of ODH where the hash function H takes an additional *tag* argument:

Definition 3.6.6. *Let $\mathbb{G}$ be a cyclic group of order n, with generator g, and let $H : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. Then the* ***tagged oracle Diffie-Hellman (TODH)*** *assumption holds in $\mathbb{G}$ with respect to H if the following distributions are indistinguishable, for all tags τ^* and all $\mathcal{A}$ that do not query their oracle with second argument τ^*:*

$a, b \leftarrow [n]$	$a, b \leftarrow [n]$
def $\mathcal{H}_a(X, \tau) = H(aX, \tau)$	def $\mathcal{H}_a(X, \tau) = H(aX, \tau)$
$K = H(abg, \tau^*)$	$K \leftarrow \{0,1\}^\ell$
return $\mathcal{A}^{\mathcal{H}_a}(ag, bg, K)$	return $\mathcal{A}^{\mathcal{H}_a}(ag, bg, K)$

In [ABR01] the authors show that standard ODH is secure in the generic group model when H is a random oracle. This proof is easily adapted to the new TODH assumption as well.

Proposition 3.6.7. *Möller tagged DHKA (Fig. 3.9) satisfies tag nonmalleability (Theorem 3.4.1) if the TODH assumption holds in both the curve and its twist.*

A further small optimization is possible for Montgomery curves. The multiplication algorithm only depends on the x-coordinate of its input and is uniform for both the curve and its twist, in the sense that the usual multiplication algorithm for the curve also correctly multiplies in the twist if the input is on the twist. So if the sender in Fig. 3.9 chooses $a_0 = a_1$ then there is no need to check whether the receiver's B is on the curve or twist. Instead, the sender simply multiplies B without any checking. However, security of this optimization requires that a kind of TODH assumption hold for the curve and twist jointly (instead of separately/independently for the curve and for the twist).

3.6.3.1 Instantiation

When creating a concrete instantiation of Möller ECDHKA, we chose to use Curve25519 [Ber06]. The main reasons for this choice were:

1. The base field $\mathbb{F}_p$ is of prime order $2^{255} - 19 > 2^{255} - 2^{255/2}$.

2. Curve25519 is explicitly designed to have a twist that is as secure as the curve itself.

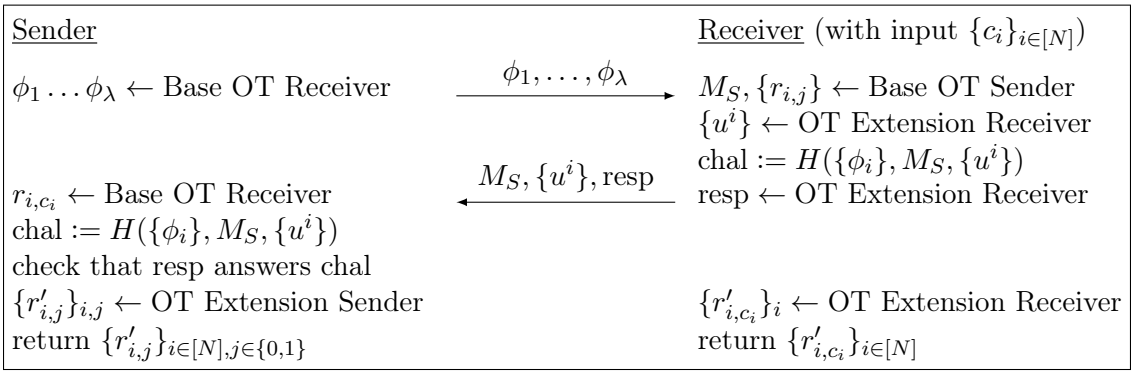


Figure 3.10: Sketch of the composition of our batch OT protocol with a maliciously secure OT extension protocol, such as OOS or SoftSpokenOT, in 2 rounds.

3. Curve25519 can take full advantage of Montgomery Ladders for scalar multiplication which allows us to use only the abscissa in computations.

4. Curve25519 and its twist have large prime subgroups of size $\#E/8$ and $\#E'_c/4$.

Curve25519 also provides additional evidence for the security of the above optimization of setting $a_0 = a_1$, because [Ber06] recommends not checking whether a given point is on the curve or twist before performing scalar multiplication. This optimization is why Curve25519 was chosen to have a secure twist, and in fact the reference implementation does not check if an elliptic curve point is on the curve. This requires a similar additional security assumption to our optimization because it uses the same key for both the curve and its twist.

3.7 2-round Endemic OT Extension

When our protocol is used for base OTs, we can achieve a 2-round Endemic OT extension protocol if the Fiat-Shamir heuristic is used. First, recall that our batch OT protocol is 1-flow when instantiated with a 1-flow KA protocol, *e.g.*, any Diffie-Hellman-based KA protocol. This gives us the flexibility to send base OT messages in any order.

Second, we summarize the 1-out-of-2 OT extension protocol of [KOS15]:

- The parties perform base OTs
- The receiver (who is base OT sender) sends data as in all IKNP-based [IKN+03]

extension protocols.

- To protect against a malicious receiver, the sender gives a random challenge
- The receiver sends a response to this challenge, which the sender checks.

We can order the messages of the base OTs so that the receiver can send their IKNP data along with their base OT sender message. Additionally, we can collapse the malicious consistency check using the Fiat-Shamir heuristic, since the sender's challenge is random. The resulting OT extension protocol is sketched in Fig. 3.10.

In related work, [CSW20] show how to use the Chou-Orlandi base OT protocol to achieve 3-round OT extension. This is inevitable since their base OTs already require 3 rounds. [BCG+19a] show a 2-round OT extension protocol based on newer "silent OT" techniques. Note however that both these papers achieve chosen message OT, while Fig. 3.10 only achieves endemic OT and would require a third round to derandomize the sender's messages.

3.8 Performance Evaluation

In this section, we will explore the concrete performance benchmarks of multiple instantiations of the protocol in Fig. 3.3.

3.8.1 Implementation Details

We implemented[9] our protocol inside the `libote` OT extension library [Rin], modifying the library to use Rijndael-256 [DR99; BÖS11] to instantiate an ideal cipher and `libsodium` [Den20] to implement elliptic curve operations. The library uses Blake2 [ANW+13] to instantiate a random oracle. We then tested the protocols on a machine running on an Intel Xeon E5-2699 v3 CPU, without assembly optimizations or multi-threading. For benchmarking, each protocol was run in a batch of 128 OTs for two settings of simulated latency and bandwidth limiting. The two settings are meant to shed light on the LAN vs WAN environments that these protocols may run in. The number of OTs to run was chosen to provide a realistic setting in the case of 128 base OTs as is common in OT extension.

We compared the following implementations:

[9] Source code is at `https://github.com/Oreko/popfot-implementation`.

- Chou-Orlandi (Simplest OT).
- Naor-Pinkas OT
- Masny-Rindal (Endemic OT), with and without reusing the sender's message. This protocol uses hash-to-curve operations.
- Our protocol instantiated with Möller's DHKA and various POPFs. We used the original Feistel POPF of MRR, as well as the POPFs presented presented in Section 3.5. Because the messages from Möller's scheme are uniformly random bit strings, our POPFs avoid the hash-to-curve operations that are needed in [MR19]. We did not evaluate the Even-Mansour POPF (Fig. 3.5) since its performance would be identical to the EKE POPF (Fig. 3.4) when Rijndael is used to instantiate both the ideal cipher and ideal permutation.
- Our protocol with traditional DHKA, and all POPF instantiations excluding EKE and Masny-Rindal. We did not implement the EKE POPF using DHKA; however, this might be possible using Elligator or a similar mapping to construct an ideal cipher on a subset of the curve points. We did not implement our protocol with Masny-Rindal POPF as it would be nearly identical to the Masny-Rindal protocol.

3.8.2 Results & Discussion

The performance benchmarks can be found in Table 3.2 for both settings.

As we would expect, when comparing the three instances of Masny-Rindal OT, each with their own improvement, we see a marked increase in efficiency. Specifically, reusing the sender's message reduced the total time spent by both parties by 18% / 11% in the low latency and high bandwidth setting / the high latency and low bandwidth setting, respectively. Moving to Möller's KA caused an additional 24% / 9% improvement, respectively, for the Masny-Rindal construction. On average, for the three protocols with both DHKA and Möller DHKA versions (Masny-Rindal and the two Feistel POPF variants) we saw an improvement of 31% / 12%, respectively, when moving to Möller's KA.

As expected, the Simplest OT protocol outperforms our instantiations for the sender since it uses fewer exponentiations in the group. One point to take note of in the evaluation data

Protocol	Security	Sender (ms)	Receiver (ms)
0.1ms latency, 10000Mbps bandwidth cap			
Simplest OT [CO15] (Sender-reuse)	standalone	35	17
Naor-Pinkas OT [NP01] (Sender-reuse)	standalone	43	34
Endemic OT [MR19] (No reuse)	UC	79	42
Endemic OT (Sender-reuse)	UC	62	37
Ours (Feistel POPF [MRR20] — DHKA)	UC	82	40
Ours (Field Feistel POPF Fig. 3.7 — DHKA)	UC	80	40
Ours (Feistel POPF — Möller DHKA)	UC	49	26
Ours (Field Feistel POPF — Möller DHKA)	UC	50	27
Ours (MR POPF Fig. 3.6 — Möller DHKA)	UC	48	27
Ours (EKE POPF Fig. 3.4 — Möller DHKA)	UC	50	25
30ms latency, 100Mbps bandwidth cap			
Simplest OT [CO15] (Sender-reuse)	standalone	105	111
Naor-Pinkas OT [NP01] (Sender-reuse)	standalone	101	107
Endemic OT [MR19] (No reuse)	UC	161	53
Endemic OT (Sender-reuse)	UC	137	53
Ours (Feistel POPF [MRR20] — DHKA)	UC	155	47
Ours (Field Feistel POPF Fig. 3.7 — DHKA)	UC	155	47
Ours (Feistel POPF — Möller DHKA)	UC	128	44
Ours (Field Feistel POPF — Möller DHKA)	UC	128	44
Ours (MR POPF Fig. 3.6 — Möller DHKA)	UC	128	44
Ours (EKE POPF Fig. 3.4 — Möller DHKA)	UC	128	44

Table 3.2: Running time to generate a batch of 128 OT instances. We report the average of 100 trials for each experiment.

is the large gap in the performance for the receiver between the Naor-Pinkas and Simplest / Blazing OT constructions and the POPF and Masny-Rindal constructions in the high latency / low bandwidth setting. This is due to the different flow requirements between the two sets of protocols. Simplest OT and Naor-Pinkas constructions all require an additional flow (or two) which, in the WAN setting, will accrue more time for the party which needs to wait. It then follows that the advantages of our protocol over Simplest OT is our UC security and round/flow complexity.

Chapter 4: SoftSpokenOT

Lawrence Roy. SoftSpokenOT: communication–computation tradeoffs in OT extension. In *CRYPTO 2022*, LNCS. Springer, Heidelberg, August 2022. URL: https://eprint.iacr.org/2022/192. To be published.

4.1 Introduction

Oblivious transfer (OT) is a basic building block of multi-party computation (MPC), and for many realistic problems, MPC protocols may require millions of OTs. [Bea96] introduced the concept of OT extension, where a small number of OTs called *base OTs* are processed to efficiently generate a much larger number of *extended OTs*. [IKN+03] (hereafter, IKNP) was the first OT extension protocol to make black-box use of its primitives, a significant improvement in efficiency. Because of its speed, it is still widely used for semi-honest OT extension.

However, IKNP has a bottleneck: communication. It transfers λ bits for every extended random OT. Recent works under the heading of Silent OT [BCG+18; BCG+19b; SGR+19; BCG+19a; YWL+20; CRR21] have improved on this. Their total communication complexity grows *logarithmically* in the number of oblivious transfers, so they are favored when communication is slow. Yet IKNP is still more computationally efficient, as only Silver [CRR21] among the Silent OT protocols has similar computational cost to IKNP. Additionally, Silent OT requires stronger assumptions than IKNP, as unlike IKNP it requires a non-Minicrypt assumption: learning parity with noise (LPN). Efficient Silent OT instantiations depend on highly structured versions of LPN, with Silver, the most efficient protocol, introducing a novel variant of LPN solely to improve its efficiency. These assumptions are too recent to have received much cryptanalysis, especially when compared to widely-used symmetric key primitives such as AES.

Improvements to IKNP also benefit a number of derived protocols. For maliciously secure OT extension, the main approach [KOS15] (hereafter, KOS) is to combine IKNP with a consistency check, although Silent OT can also achieve malicious security. [KK13] achieved $\binom{N}{1}$-OT extension by noticing that part of IKNP can be viewed as encoding the OT choice

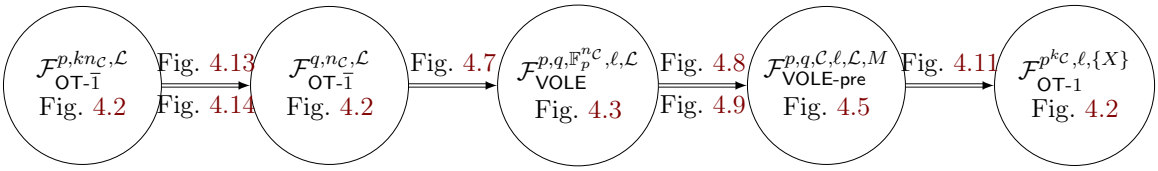


Figure 4.1: Sequence of ideal functionalities and protocols used for OT extension. Here $q = p^k$ is the size of the small field VOLE, and $\mathcal{L} = \text{Affine}(\mathbb{F}_p^{kn_C})$ is the set of allowed selective abort attacks against the base OT receiver. Protocols below the arrows are consistency checks needed for maliciously security.

bits with a repetition code. They replaced it with a more sophisticated error correcting code. [OOS17] (hereafter, OOS) and [PSS17] (hereafter, PSS) then devised more general consistency checking protocols to achieve maliciously secure $\binom{N}{1}$-OT extension. [CCG18] (hereafter, CCG) generalized OOS to work over larger fields, which have better linear codes. This allowed for fewer base OTs, but required more communication per extended OT.

4.1.1 Our Results

Our technique, SoftSpokenOT, makes an asymptotic improvement over IKNP's communication cost. It is the first OT extension to do so in the Minicrypt model. While IKNP needs λ bits of communication for each OT, SoftSpokenOT can implement $\binom{2}{1}$-OT maliciously secure extension using only λ/k bits, for any parameter $k \geq 1$. This comes at the cost of the sender *generating* $\lambda \cdot 2^k/k$ pseudorandom bits (and a similar number by the receiver), while IKNP only needs to generate 2λ bits. In practice, IKNP is network bound due to fast hardware implementations of AES, so when k is set to be appropriately small (e.g. $k = 5$) SoftSpokenOT's extra computation will have no effect on the overall protocol latency. Note that when $k = 2$, SoftSpokenOT requires no extra computation is required, making it a pure improvement over IKNP. Asymptotically, setting $k = \Theta(\log(\ell))$ generates ℓ OTs with sublinear communication $\Theta\left(\frac{\lambda \cdot \ell}{\log(\ell)}\right)$, in polynomial time.

We present a sequence of protocols (Fig. 4.1), starting with base OTs, continuing through vector oblivious linear evaluation (VOLE), and ending at OT extension. First, we present a novel silent protocol for VOLE over polynomial-sized fields, which may be of independent interest. A VOLE generates correlated randomness $(\vec{u}, \vec{v})$ and $(\Delta, \vec{w})$ where $\vec{w} - \vec{v} = \vec{u}\Delta$. Our next stepping stone is an ideal functionality that we call subspace VOLE, which produces

correlations satisfying $W - V = UG_{\mathcal{C}} \operatorname{diag}(\bar{\Delta})$. Here, $G_{\mathcal{C}}$ is the generator matrix for a linear code $\mathcal{C}$. Note that Δ-OT (a.k.a. correlated OT) is a special case of subspace VOLE, as is the correlation used by PaXoS [PRT+20]. Our Δ-OT works over any field of polynomial size, so it can encode the inputs for arithmetic garbling [BMR16]. Finally, we hash the subspace VOLE using a correlation robust (CR) hash to build random $\binom{N}{1}$, a correlation (x, m_x) and $(m_0, \ldots, m_{N-1})$ where the m_y are all random. These may be used directly, or to encode lookup tables representing multiple small-secret $\binom{2}{1}$-OTs [KK13].

We generalize OOS to construct a consistency checking protocol that achieves maliciously secure subspace VOLE, albeit with a selective abort attack. However, while proving our protocol secure, we found flaws (Section 4.4.1) in three existing works on consistency checks for OT extension. For OOS this is minor — just a flaw in their proof — and a special case of our new proof shows that OOS is still secure. We found two attacks on KOS which show that it is not always as secure as claimed, though it's still secure enough in practice. We leave to future research the problem of finding a sound proof of security for KOS. However, PSS's flaw is more severe, as we found a practical attack that can break their $\binom{256}{1}$-OT extension at $\lambda = 128$ security in time 2^{34} with probability 2^{-8}.

There is an existing work on OT extension consistency checking that we did not find to be flawed. CCG base their proof on [CDD+16]'s careful analysis of consistency checking for homomorphic commitments. CCG's check is similar to ours in that it works over any field. However, similarly to [CDD+16] and unlike CCG we use universal hashing to compress the random challenge of the consistency check. Additionally, we prove a tighter concrete security bound than either work, which halves the number of rows that must be consistency checked.

The final step, going from correlated randomness (i.e. subspace VOLE) to extended OTs, requires a CR hash function. For malicious security, a mechanism is needed to stop the receiver from causing a collision between CR hash inputs. [GKW+20b] solve this with a tweakable CR (TCR) hash, using a tweak to stop these collisions. TCR hashes are more expensive than plain CR hashes, so Endemic OT [MR19] instead prevent the receiver from controlling the base OTs, proving that it is secure to forgo tweaks in this case. However, their proof assumes stronger properties of the consistency checking protocol than are provided by real consistency checks, allowing us to find an attack on their OT extension (see Section 4.5). We follow [CT21] in using a universal hash to prevent collisions, only using the tweak to improve the concrete security of the TCR hash. We optimize their technique by sending the universal hash in parallel with our new consistency check — our proof shows that the

receiver has few remaining choices once it learns the universal hash.

We implemented SoftSpokenOT for $\binom{2}{1}$-OT in the libOTe [Rin] library. When tested with a 1Gbps bandwidth limit, our protocol has almost a 5× speedup over IKNP with $k = 5$, resulting from a 5× reduction in communication. The only case where SoftSpokenOT was suboptimal among the tested configurations was in the WAN setting, where it took second place to Silver. However, the assumptions needed by SoftSpokenOT are much more conservative than those used by Silver.

4.1.2 Technical Overview

SoftSpokenOT is a generalization of the classic oblivious transfer extension of IKNP, which at its core is based on what can be viewed as a protocol for $\mathbb{F}_2$-VOLE. This VOLE protocol starts by using a PRG to extend $\binom{2}{1}$-OT to message size ℓ. The base OT sender, P_S, gets random strings $\vec{m}_0, \vec{m}_1$ and the receiver, P_R, gets its choice bit $b \in \mathbb{F}_2$ and its chosen message $\vec{m}_b$. P_S then computes $\vec{u} = \vec{m}_0 \oplus \vec{m}_1$ and $\vec{v} = \vec{m}_1 = 0\,\vec{m}_0 \oplus 1\,\vec{m}_1$, while P_R computes $\Delta = 1 \oplus b$, and $w = \vec{m}_b = \Delta\vec{m}_0 \oplus (1 \oplus \Delta)\vec{m}_1$.[1] Then $\vec{w} \oplus \vec{v} = \Delta\vec{m}_0 \oplus \Delta\vec{m}_1 = \Delta\vec{u}$, which is a VOLE correlation: P_S gets a vector $\vec{u} \in \mathbb{F}_2^\ell$ and P_R gets a scalar $\Delta \in \mathbb{F}_2$, and they learn secret shares $\vec{v}, \vec{w}$ of the product. While $\vec{u}$ was chosen by the protocol, it possible to derandomize $\vec{u}$ to be any chosen vector. If P_S wants to use $\vec{u}'$ instead, it can send $\bar{u} = \vec{u} \oplus \vec{u}'$ to P_R, who updates its share to be $\vec{w}' = \vec{w} \oplus \Delta\bar{u}$. This preserves the VOLE correlation, $\vec{w}' \oplus \vec{v} = \Delta\vec{u} \oplus \Delta\bar{u} = \Delta\vec{u}'$, while hiding $\vec{u}'$.

The next step of the IKNP protocol is to stack λ of these $\mathbb{F}_2$-VOLEs side by side, while sending $\lambda \cdot \ell$ bits to derandomize the $\vec{u}$ vectors to all be the same. That is, for the ith VOLE, they get a correlation $W_{\cdot i} \oplus V_{\cdot i} = \Delta_i\vec{u}$, where $V_{\cdot i}$ means the ith column of a matrix V. In matrix notation, this is an outer product: $W \oplus V = \vec{u}\overleftarrow{\Delta}$, where $\overleftarrow{\Delta}$ is the row vector of all the Δ_i. Then looking at the jth row gives $W_{j\cdot} \oplus V_{j\cdot} = u_j\overleftarrow{\Delta}$, which make u_j the choice bit of a Δ-OT. That is, P_R has learned $\overleftarrow{m}_{j0} = W_{j\cdot}$ and $\overleftarrow{m}_{j1} = W_{j\cdot} \oplus \overleftarrow{\Delta}$, while P_S has its choice bit u_j and $\overleftarrow{m}_{u_j} = V_{j\cdot}$, the corresponding message. Notice that this is a correlated OT, but now the OT sender is P_R and the OT receiver is P_S — they have been reversed from what they were for the base OTs. Hashing the $\overleftarrow{m}_{jx}$ then turns them into uncorrelated OT messages.

[1] Note that this is backwards from the usual description of IKNP — it's more usual to set Δ to be the b, the index of the message known to P_R. A key insight in SoftSpokenOT is that the unknown base OT message is the most important.

SoftSpokenOT instead bases the OT extension on a $\mathbb{F}_{2^k}$-VOLE, where $\vec{u}$ is restricted to taking values in $\mathbb{F}_2$. We now only need λ/k of these VOLEs to get the λ bits per OT needed to make the hash secure. Derandomizing $\vec{u}$ for each OT then only needs λ/k *bits* per OT, as for each VOLE the elements of $\vec{u}$ are in $\mathbb{F}_2$, reducing a major bottleneck of IKNP. Instead of $\binom{2}{1}$-OT, our $\mathbb{F}_{2^k}$-VOLE is based on $\binom{2^k}{2^k-1}$-OT, which can be instantiated using a well known protocol [BGI17] based on a punctured PRF; see Section 4.6 for details.

In $\binom{2^k}{2^k-1}$-OT a random function $F\colon \mathbb{F}_{2^k} \to \mathbb{F}_2^\ell$ is known to P_S, while P_R has a random point Δ and the restriction F^* of F to $\mathbb{F}_{2^k} \setminus \{\Delta\}$. The earlier equations for the vectors $\vec{u}$, $\vec{v}$, and $\vec{w}$ were chosen to be suggestive of their generalizations:

$$\begin{aligned}
\vec{u} &= F(0) \oplus F(1) && \implies \vec{u} = \bigoplus_{x \in \mathbb{F}_{2^k}} F(x) \\
\vec{v} &= 0\,F(0) \oplus 1\,F(1) && \implies \vec{v} = \bigoplus_{x \in \mathbb{F}_{2^k}} xF(x) \\
\vec{w} &= \Delta F^*(0) \oplus (1 \oplus \Delta)F^*(1) && \implies \vec{w} = \bigoplus_{x \in \mathbb{F}_{2^k}} (x \oplus \Delta)F^*(x).
\end{aligned}$$

Notice that the formula for $\vec{w}$ multiplies $F^*(\Delta)$ by 0, which is good because $F(\Delta)$ is unknown to P_R. Therefore, $\vec{w} \oplus \vec{v} = \bigoplus_x \Delta F(x) = \Delta\vec{u}$.

Reducing communication by a factor of k comes at the expense of increasing computation by a factor of $2^k/k$. While there are now only λ/k VOLES, they each require both parties to evaluate F at every point (except the one that P_R does not know) in a finite field of size 2^k.

4.2 Preliminaries

4.2.1 Notation

We start counting at zero, and the set $[N]$ is $\{0, 1, \ldots, N-1\}$. The finite field with p elements is written as $\mathbb{F}_p$, the vector space of dimension n as $\mathbb{F}_p^n$, and set of all $m \times n$ matrices as $\mathbb{F}_p^{m\times n}$. The vectors themselves are written with an arrow, as $\vec{x}$, while matrices are capital letters M. Row vectors are written with a backwards arrow instead: $\overleftarrow{x}$. The componentwise product of vectors is $\vec{x} \odot \vec{y} = [x_0 y_0 \ \cdots \ x_{n-1} y_{n-1}]^\top$. Diagonal matrices are

notated $\mathrm{diag}(\vec{x}) = \begin{bmatrix} x_0 & & \\ & \ddots & \\ & & x_{n-1} \end{bmatrix}$, which makes $\vec{x} \odot \vec{y} = \mathrm{diag}(\vec{x})\vec{y}$. The ith row of a matrix M is $M_{i\cdot}$, while the jth column is $M_{\cdot j}$. The first r rows of M are $M_{[r]\cdot}$, and the first c columns are $M_{\cdot[c]}$.

There are two finite fields we will usually work with: the subfield $\mathbb{F}_p$, and its extension field $\mathbb{F}_q$, where $q = p^k$. Usually p will be prime, but that is not necessary. In a few places we will equivocate between $\mathbb{F}_q$, $\mathbb{F}_p^k$, and $[q]$, using the obvious bijections between them.

Linear Codes. Let $\mathcal{C}$ be a $[n_\mathcal{C}, k_\mathcal{C}, d_\mathcal{C}]$ linear code, that is, $\mathcal{C}$ is a $k_\mathcal{C}$-dimensional subspace of $\mathbb{F}_p^{n_\mathcal{C}}$ with minimum distance $d_\mathcal{C} = \min_{\vec{y} \in \mathcal{C} \setminus \{0\}} \|\vec{y}\|_0$, where $\|\vec{y}\|_0$ is the Hamming weight of $\vec{y}$. For a matrix A, we similarly let the Hamming weight $\|A\|_0$ be the number of nonzero columns of A. Let $G_\mathcal{C} \in \mathbb{F}_p^{k_\mathcal{C} \times n_\mathcal{C}}$ be the generator matrix of $\mathcal{C}$. We follow the convention that the messages and code words are row vectors, so a row vector $\vec{x}$ encodes to the codeword $\vec{x}G_\mathcal{C} \in \mathcal{C}$. The rows of the generator matrix must form a basis of $\mathcal{C}$, which can be completed into a basis $T_\mathcal{C}$ of $\mathbb{F}_p^{n_\mathcal{C}}$; that is, the first $k_\mathcal{C}$ rows of $T_\mathcal{C}$ are $G_\mathcal{C}$. Then $T_\mathcal{C}$ has an inverse $T_\mathcal{C}^{-1}$, the last $n_\mathcal{C} - k_\mathcal{C}$ columns of which form a parity check matrix for $\mathcal{C}$.

There are two specific codes that come up most frequently. The trivial code, $\mathbb{F}_p^n$, has all vectors as code words. That is, $G_{\mathbb{F}_p^n} = T_{\mathbb{F}_p^n} = \mathbb{1}_n$, where $\mathbb{1}_n$ is the $n \times n$ identity matrix. The repetition code, $\mathsf{Rep}(\mathbb{F}_p^n)$, consists of all vectors where all elements are the same. Its generator matrix is $G_{\mathsf{Rep}(\mathbb{F}_p^n)} = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}$.

Algorithms. We use pseudocode for our constructions. In many cases there will be two similar algorithms side by side (e.g. sender and receiver, or real and ideal), and we use whitespace to align matching lines. Sampling a value x uniformly at random in a set X is written as $x \xleftarrow{\$} X$.

4.2.2 Universal Hashes

We make extensive use of universal hashes [CW79], essentially as a more efficient replacement for a uniformly random matrix. We depend on the extra structure of the hash function being linear, so we give definitions specialized to that case.

Definition 4.2.1. *A family of matrices $\mathcal{R} \subseteq \mathbb{F}_q^{m \times n}$ is a* linear ϵ-almost universal family *if, for all nonzero $\vec{x} \in \mathbb{F}_q^n$, $\Pr_{R \xleftarrow{\$} \mathcal{R}}\left[R\vec{x} = 0\right] \leq \epsilon$.*

Definition 4.2.2. *A family of matrices $\mathcal{R} \subseteq \mathbb{F}_q^{m \times n}$ is* linear ϵ-almost uniform family *if, for all nonzero $\vec{x} \in \mathbb{F}_q^n$ and all $\vec{y} \in \mathbb{F}_q^m$, $\Pr_{R \xleftarrow{\$} \mathcal{R}}\left[R\vec{x} = \vec{y}\right] \leq \epsilon$.*

For characteristic 2, this is equivalent to being ϵ-almost XOR-universal. Clearly, a family that is ϵ-almost uniform is also ϵ-almost universal. In Appendix C.6.1, we prove two composition properties of universal hashes.

Proposition 4.2.3. *Let $\mathcal{R}$ and $\mathcal{R}'$ be ϵ and ϵ'-almost universal families, respectively. Then $R'R$ for $R \in \mathcal{R}, R' \in \mathcal{R}'$ is a $(\epsilon + \epsilon')$-universal family.*

Proposition 4.2.4. *Let $\mathcal{R}$ and $\mathcal{R}'$ be ϵ-almost uniform families. Then $[R\ R']$ for $R \in \mathcal{R}, R' \in \mathcal{R}'$ is a ϵ-uniform family.*

4.2.3 Ideal Functionalities

The protocols in this paper are analyzed in the Simplified UC model of [CCL15], so whenever an ideal functionality takes inputs or outputs, the adversary is implicitly notified and allowed to delay or block delivery of the message. The functionalities deal with three entities: the sender P_S, the receiver P_R, and the adversary $\mathcal{A}$. Instead of the usual event-driven style (essentially a state machine driven by the messages), we use a blocking call syntax for our ideal functionalities, where it stops and waits to receive a message. While we will not need to receive multiple messages at once, it would be consistent to use multiple parallel threads of execution, with syntax like $\boxed{\text{recv. } x \text{ from } P_S \,\|\, \text{recv. } y \text{ from } P_R}$. We omit the "operation labels" identifying the messages, instead relying on the variable names and message order to show which send corresponds to each receive. We assume the protocol messages themselves are delivered over an authenticated channel.

All of our functionalities are for different kinds of random input VOLE or OT, meaning that the protocol pseudorandomly chooses the inputs of each party. Essentially, the functionalities just generate correlated randomness. Using random VOLE or OT, the parties can still choose their inputs using derandomization, if necessary.[2] However, we cannot guarantee that a corrupted participant does not exercise partial control over the outputs of the protocols. For this reason, we use the endemic security notion of [MR19], where any corrupted participants get to choose their protocol outputs, then the remaining honest parties receive random outputs,

[2] See [MR19] for details on derandomizing OT messages.

$\mathcal{F}_{\text{OT-1}}^{N,\ell,\mathcal{L}}$

for $i \in [\ell]$:
  if P_S is corrupted:
    recv. $F_i \in \left(\{0,1\}^\lambda\right)^{[N]}$ from $\mathcal{A}$
  else:
    $F_i \xleftarrow{\$} \left(\{0,1\}^\lambda\right)^{[N]}$
  if P_R is corrupted:
    recv. $x_i^* \in [N]$ from $\mathcal{A}$
    recv. $F_i^* \in \{0,1\}^\lambda$ from $\mathcal{A}$
    $F_i(x_i^*) := F_i^*$
  else:
    $x_i^* \xleftarrow{\$} [N]$
    $F_i^* := F_i(x_i^*)$
send $\{F_i\}_{i\in[\ell]}$ to P_S
$\mathsf{Send/Abort}\left(\{x_i^*\}_{i\in[\ell]}, \{F_i^*\}_{i\in[\ell]}, \mathcal{L}\right)$

$\mathcal{F}_{\text{OT-}\bar{1}}^{N,\ell,\mathcal{L}}$

for $i \in [\ell]$:
  if P_S is corrupted:
    recv. $F_i \in \left(\{0,1\}^\lambda\right)^{[N]}$ from $\mathcal{A}$
  else:
    $F_i \xleftarrow{\$} \left(\{0,1\}^\lambda\right)^{[N]}$
  if P_R is corrupted:
    recv. $x_i^* \in [N]$ from $\mathcal{A}$
    recv. $F_i^* \in \left(\{0,1\}^\lambda\right)^{[N]\setminus\{x_i^*\}}$ from $\mathcal{A}$
    $F_i(x) := F_i^*(x), \forall x \in [N] \setminus \{x_i^*\}$
  else:
    $x_i^* \xleftarrow{\$} [N]$
    $F_i^*(x) := F_i(x), \forall x \in [N] \setminus \{x_i^*\}$
send $\{F_i\}_{i\in[\ell]}$ to P_S
$\mathsf{Send/Abort}\left(\{x_i^*\}_{i\in[\ell]}, \{F_i^*\}_{i\in[\ell]}, \mathcal{L}\right)$

Figure 4.2: Ideal functionalities for a batch of ℓ endemic OTs, with $\binom{N}{1}$-OT on the left and $\binom{N}{N-1}$-OT on the right. Differences are highlighted.

$\mathcal{F}_{\text{VOLE}}^{p,q,\mathcal{C},\ell,\mathcal{L}}$

if P_S is corrupted:
  recv. $U \in \mathbb{F}_p^{\ell\times kc}, V \in \mathbb{F}_q^{\ell\times nc}$ from $\mathcal{A}$
else:
  $U \xleftarrow{\$} \mathbb{F}_p^{\ell\times kc}, V \xleftarrow{\$} \mathbb{F}_q^{\ell\times nc}$
if P_R is corrupted:
  recv. $\vec{\Delta} \in \mathbb{F}_q^{nc}, W \in \mathbb{F}_q^{\ell\times nc}$ from $\mathcal{A}$
  $V := -UG_{\mathcal{C}}\,\mathrm{diag}(\vec{\Delta}) + W$
else:
  $\vec{\Delta} \xleftarrow{\$} \mathbb{F}_q^{nc}$
  $W := UG_{\mathcal{C}}\,\mathrm{diag}(\vec{\Delta}) + V$
send U, V to P_S
$\mathsf{Send/Abort}(\vec{\Delta}, W, \mathcal{L})$

Figure 4.3: Ideal functionality for endemic subspace VOLE. $\mathcal{C}$ is a linear code.

$\mathsf{Send/Abort}(x \in X, y \in Y, \mathcal{L} \subseteq 2^X)$:
  if P_S is malicious:
    recv. $L \in \mathcal{L}$ from P_S
    if $x \notin L$:
      send "check failed" to P_R
      abort
  send x, y to P_R

Figure 4.4: Output with leakage function. Sends x, y to P_R, after allowing P_S to do a selective abort attack on x.

subject to the correlation. One difference, however, is that in our ideal functionalities an honest OT receiver doesn't get to choose its choice bits. Instead, all protocol inputs are random for honest parties.[3]

The ideal functionalities for length ℓ batches of $\binom{N}{1}$-OTs or $\binom{N}{N-1}$-OTs are presented in Fig. 4.2. In each OT, the sender P_S gets a random function $F\colon [N] \to \{0,1\}^\lambda$, which is chosen by the adversary if P_S is corrupted. If N is exponentially large, F should be thought of as an oracle, which will only be evaluated on a subset of $[N]$. The receiver P_R gets a choice element $x^* \in [N]$, as well as F^*, which is either the one point $F(x^*)$ for $\binom{N}{1}$-OT, or the restriction of F to every other point $[N] \setminus x^*$ for $\binom{N}{N-1}$-OT. Again, if P_R is corrupted then the adversary gets to choose these values.

In Fig. 4.3, we present subspace VOLE, a generalized notion of VOLE. Instead of a correlation of vectors $\vec{w} - \vec{v} = \vec{u}\Delta$, where $\vec{u} \in \mathbb{F}_p^\ell$ and $\vec{v} \in \mathbb{F}_q^\ell$ are given to P_S, and $\vec{w} \in \mathbb{F}_q^\ell$ and $\Delta \in \mathbb{F}_q$ to P_R [BCG+18], subspace VOLE produces a correlation of matrices $W - V = UG_{\mathcal{C}}\,\mathrm{diag}(\vec{\Delta})$, where U gets multiplied by the generator matrix $G_{\mathcal{C}}$ of a linear code $\mathcal{C}$. Subspace VOLE is essentially $n_{\mathcal{C}}$ independent VOLE correlations placed side-by-side, except that the rows of U are required to be code words of $\mathcal{C}$. For $p = q = 2$, this matches the correlation generated internally by existing $\binom{N}{1}$-OT extensions.

Selective Aborts. Our base $\binom{N}{N-1}$-OT OT and subspace VOLE protocols achieve malicious security by using a consistency check to enforce honest behavior. However, the consistency checks allow a selective abort attack where P_S can confirm a guess of part of P_R's secret outputs. This is modeled in the ideal functionality using the function Send/Abort (Fig. 4.4). Let $x \in X$ be the value subject to the selective abort attack, and $y \in Y$ be the rest of P_R's output. When P_S is malicious, it can guess a subset $L \subseteq X$, and if it is correct (i.e. $x \in L$) then the protocol continues as normal. But if the guess is wrong then P_R is notified of the error, and the protocol aborts.

The subset L that P_S guesses is restricted to being a member of $\mathcal{L}$, for some set of allowed guesses $\mathcal{L} \subseteq 2^X$. It is required to be closed under intersection, and contain the whole set X. For VOLE, where X is a vector space, we also require that $L - \vec{L}_{\text{off}} \in \mathcal{L}$ when $L \in \mathcal{L}$ and $\vec{L}_{\text{off}} \in X$. We use one main set of allowed guesses, $\mathrm{Affine}(\mathbb{F}_q^n)$. It is the set of all affine subspaces of $\mathbb{F}_q^n$, i.e. all subsets that are defined by zero or more constraints of the form $a_0x_0 + \cdots + a_{n-1}x_{n-1} = a_n$, for constants $a_0, \ldots, a_n \in \mathbb{F}_q$. Since $\mathbb{F}_q$ can be viewed as the

[3] This is similar to the pseudorandom correlation generators (PCGs) used in [BCG+19b] to build Silent OT. In fact, the small field VOLE constructed in Section 4.3.1 can be viewed as a PCG.

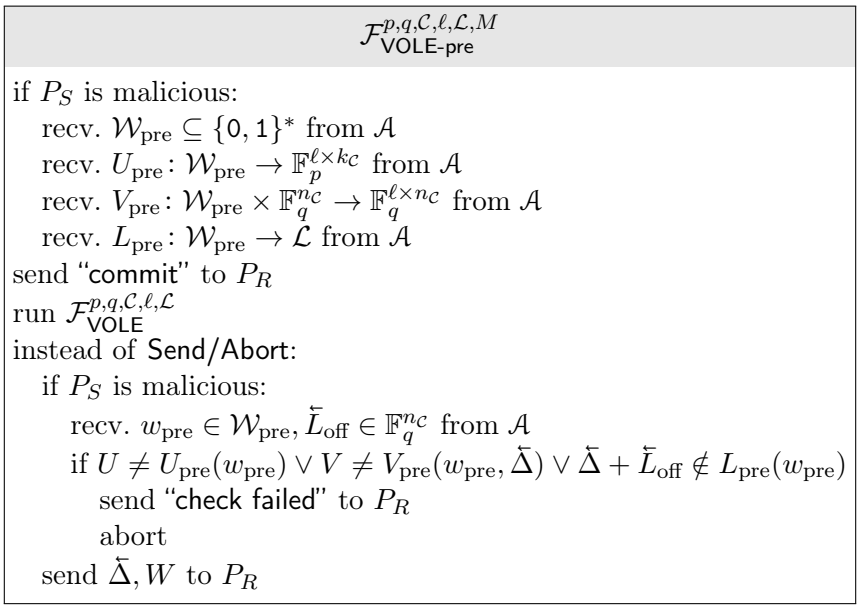


Figure 4.5: Modification of Fig. 4.3 to get an ideal functionality for subspace VOLE with a pre-commitment notification. We make two additional requirements on $\mathcal{A}$. There must be a polynomial upper bound $M \geq |\mathcal{W}_{\text{pre}}|$ on the number of input choices P_S has. And, for all $\overleftarrow{\Delta}$, $\overleftarrow{\Delta} + \overleftarrow{L}_{\text{off}} \in L_{\text{pre}}(w_{\text{pre}})$ must imply $V = V_{\text{pre}}(w_{\text{pre}}, \overleftarrow{\Delta})$, to ensure that checking V_{pre} does not make the selective abort any more powerful.

vector space $\mathbb{F}_p^k$, we have a superset relationship $\text{Affine}(\mathbb{F}_p^{nk}) \supseteq \text{Affine}(\mathbb{F}_q^n)$. There is also $\{X\}$, the trivial guess set, which only allows a malicious P_S to guess that $x \in X$. This guess is trivially true, and so leaks no information at all.

Pre-committed Inputs. Our malicious OT extension protocol uses a universal hash to stop P_R from causing collisions between two distinct extended OTs, which is sent in parallel with the VOLE consistency check for efficiency. However, the universal hash must be chosen *after* P_R (who acts as the VOLE *sender*) picks its VOLE outputs U, V and its guess L. In Fig. 4.5, we modify the VOLE functionality to notify the VOLE receiver once U, V, L are almost fixed — unfortunately, the consistency check still allows U, V, L to vary somewhat. Specifically, U may have polynomially many options (which can be computationally hard to find), L can get shifted by an offset $\overleftarrow{L}_{\text{off}}$, and V can depend on the part of $\overleftarrow{\Delta}$ that is guessed.

To address these difficulties, we identify the possible input choices with witnesses w_{pre}, and have $\mathcal{A}$ output a witness checker, i.e. an implicitly defined set $\mathcal{W}_{\text{pre}}$ of valid witnesses.

Then we require U, V, and L to be fixed in terms of w_{pre}, using functions $U_{\text{pre}}(w_{\text{pre}})$, $V_{\text{pre}}(w_{\text{pre}}, \vec{\Delta})$, and $L_{\text{pre}}(w_{\text{pre}})$. We require a polynomial upper bound $M \geq |\mathcal{W}_{\text{pre}}|$ on the number of witnesses. Additionally, so that the correctness check for V_{pre} does not leak any information, for all $\vec{\Delta}$ we require that $\vec{\Delta} + \vec{L}_{\text{off}} \in L_{\text{pre}}(w_{\text{pre}})$ implies $V = V_{\text{pre}}(w_{\text{pre}}, \vec{\Delta})$.

These changes are behind "if P_S is malicious" checks, so in the semi-honest case $\mathcal{F}_{\mathsf{VOLE}}$ is a equivalent to $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}$. For malicious security, $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}$ gives the adversary less power than $\mathcal{F}_{\mathsf{VOLE}}$ because it forces some of the choices to be made early, so any protocol for $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}$ is also a protocol for $\mathcal{F}_{\mathsf{VOLE}}$.

4.2.4 Correlation Robust Hashes

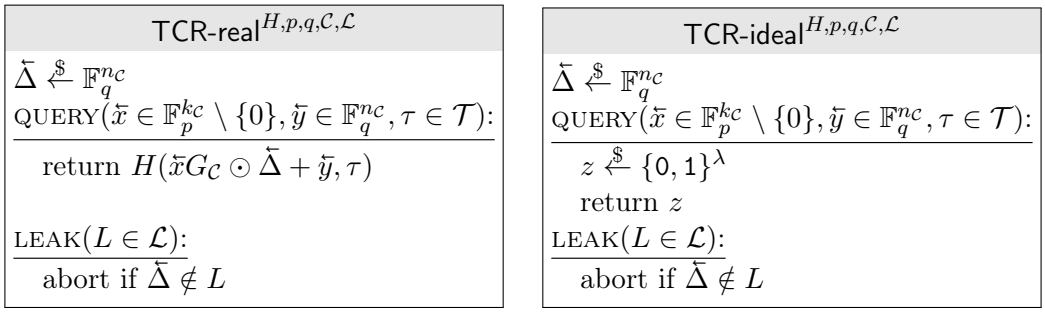
The final step of OT extension is to hash the output from the subspace VOLE. This requires a security assumption on the hash function H. We generalize the notion of a tweakable correlation robust (TCR) hash function [GKW+20b] to our setting. While this definition will most likely be used with $p = 2$ for efficiency, there are extra theoretical difficulties associated with $p > 2$.

Definition 4.2.5. *A function $H \in \mathbb{F}_q^{nc} \times \mathcal{T} \to \{0,1\}^\lambda$ is a $(p, q, \mathcal{C}, \mathcal{T}, \mathcal{L})$-*TCR hash *if the oracles given in Fig. 4.6 are indistinguishable.*[4] *Formally, for any PPT adversary $\mathcal{A}$ that does not call* QUERY *twice on the same input $(\vec{x}, \vec{y}, \tau)$,*

$$\mathsf{Adv}_{\text{TCR}} = \left| \Pr\left[\mathcal{A}^{\mathsf{TCR\text{-}real}^{H,p,q,\mathcal{C},\mathcal{L}}}() = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{TCR\text{-}ideal}^{H,p,q,\mathcal{C},\mathcal{L}}}() = 1\right] \right| \leq \text{negl}.$$

Our definition is quite similar to the TCR of [GKW+20b] in the special case where $\mathcal{C}$ is the repetition code. However, we explicitly include selective abort attacks in the TCR definition, while they require that the hash be secure for any distribution for $\vec{\Delta}$ with sufficient min-entropy. Their definition has issues when instantiated from idealized primitives such as random oracles, because, when the TCR is used for OT extension, the distribution for $\vec{\Delta}$ would have to depend on these primitives [CT21]. In the standard model, their definition is impossible to instantiate: $H(\vec{\Delta}, 0)$ must be random by TCR security, yet restricting $\vec{\Delta}$ so that the first bit of $H(\vec{\Delta}, 0)$ is zero only reduces the min-entropy by approximately one

[4] Note that we do not consider multi-instance security. In fact, there is a generic attack: given N instances, the attacker chooses an L that contains $\vec{\Delta}$ with probability $1/N$, then brute forces $\vec{\Delta}$ for instances where $\vec{\Delta} \in L$. Thus, it is N-times cheaper to brute force attack H for N instances than to target a single one.



(a) Real world. (b) Ideal world.

Figure 4.6: Oracles for TCR definition. Calls to QUERY must not be repeated on the same input.

bit and allows an efficient distinguisher. [CT21] fix the former issue with a definition TCR* that only applies to the ideal model, while ours allows the possibility of standard model constructions.

We now give two hash constructions, which we prove secure in Appendix C.1. Correlation robust hashes were inspired by random oracles (ROs), so it should be no surprise that a RO is a TCR hash.

Proposition 4.2.6. *A random oracle* $\mathsf{RO}\colon \mathbb{F}_q^{nc} \times \{0,1\}^t \to \{0,1\}^{\lambda}$ *is a* $(p, q, \mathcal{C}, \{0,1\}^t, \text{Affine}(\mathbb{F}_p^{knc}))$*-TCR hash, with distinguisher advantage at most* $\tau_{max}\left(\mathfrak{q} + \frac{1}{2}\mathfrak{q}'\right)q^{-d_C}$*. Here,* τ_{max} *is the maximum number of times* QUERY *is called with the same* τ*,* $\mathfrak{q}$ *is the number of RO queries made by the distinguisher, and* $\mathfrak{q}'$ *is the number of calls to* QUERY*.*

The next construction comes from [GKW+20a]. It is the classic $x \mapsto \pi(x) \oplus x$ permutation-based hash function, but it uses an ideal cipher so that the tweak can be the key. Changing keys in a block cipher requires recomputing the round keys, so there is a cost to changing the tweak with this method. It needs a injection ι to encode its input; when $p = 2$, ι can be the identity map.

Proposition 4.2.7. *Let* $\mathsf{Enc}\colon \{0,1\}^t \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ *be an ideal cipher, and* $\iota\colon \mathbb{F}_q^{nc} \to \{0,1\}^{\lambda}$ *be an injection. Then* $H(\vec{y}, \tau) = \mathsf{Enc}(\tau, \iota(\vec{y})) \oplus \iota(\vec{y})$ *is a* $(p, q, \mathcal{C}, \{0,1\}^t, \text{Affine}(\mathbb{F}_p^{knc}))$*-TCR hash. The distinguisher's advantage is at most* $\tau_{max}\left((2\mathfrak{q} + \frac{1}{2}\mathfrak{q}')q^{-d_C} + \frac{1}{2}\mathfrak{q}'2^{-\lambda}\right)$*, with* $\mathfrak{q}$ *and* $\mathfrak{q}'$ *as in Theorem 4.2.6.*

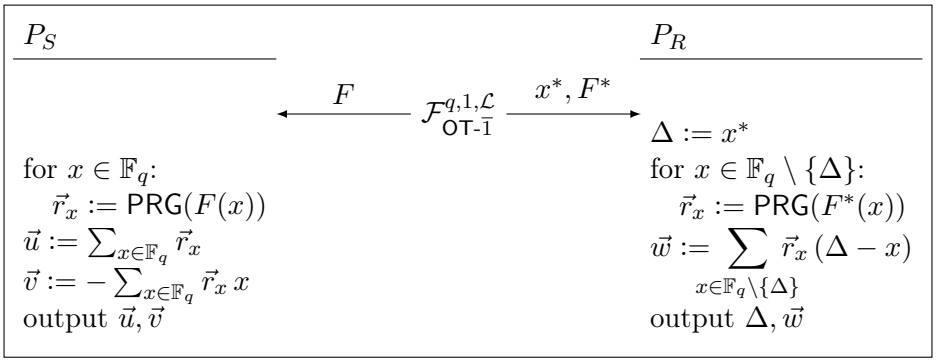


Figure 4.7: Protocol for small field VOLE. If $\mathcal{F}_{\text{OT-}\bar{1}}^{q,1,\mathcal{L}}$ instead outputs "check failed", it should be passed straight through to P_R.

4.3 VOLE

4.3.1 For Small Fields

We already presented our $\mathbb{F}_{2^k}$-VOLE in Section 4.1.2. This VOLE is generalized in Fig. 4.7 to work over any small field $\mathbb{F}_q$, specifically fields where q is only polynomially large, with $\vec{u}$ taking values in any subfield $\mathbb{F}_p$. It is based on a $\binom{q}{q-1}$-OT, and a pseudorandom generator $\mathsf{PRG}\colon \{0,1\}^\lambda \to \mathbb{F}_p^\ell$. While this is a VOLE protocol, we analyze it using our subspace VOLE definition by setting $\mathcal{C}$ to be the length one, dimension one code, i.e. $G_\mathcal{C} = [1]$. This makes U, V, and W all become column vectors and $\check{\Delta}$ become a scalar.

Theorem 4.3.1. *The VOLE given in Fig. 4.7 in the $\mathcal{F}_{\text{OT-}\bar{1}}^{q,1,\mathcal{L}}$ hybrid model securely realizes $\mathcal{F}_{\text{VOLE}}^{p,q,\mathbb{F}_p,\ell,\mathcal{L}}$, in both the semihonest and malicious models.*

Proof. The proof of correctness is simple enough. Notice that the $x = \Delta$ term of the sum for $\vec{w}$ would be multiplied by $\Delta - \Delta = 0$, so it makes no difference that it must be excluded because P_R does not know $\vec{r}_\Delta$. Therefore,

$$\vec{w} = \sum_{x\in\mathbb{F}_q\setminus\{\Delta\}} \vec{r}_x\,(\Delta - x) = \sum_{x\in\mathbb{F}_q} \vec{r}_x\,(\Delta - x) = \sum_{x\in\mathbb{F}_q} \vec{r}_x\,\Delta - \sum_{x\in\mathbb{F}_q} \vec{r}_x\, x = \vec{u}\,\Delta + \vec{v}. \tag{4.1}$$

Corrupt P_S. After receiving F from $\mathcal{A}$, the simulator will compute $\vec{u}, \vec{v}$ honestly and submit them to $\mathcal{F}_{\text{VOLE}}^{p,q,\mathbb{F}_p,\ell,\mathcal{L}}$. If P_S is malicious, it will also forward $L \in \mathcal{L}$ to the ideal functionality. In the real world, $\mathcal{F}_{\text{OT-}\bar{1}}^{q,1,\mathcal{L}}$ will generate a random $x^* = \Delta$ and send it to P_R, who will compute

$\vec{w} = \vec{u}\,\Delta + \vec{v}$ by Eq. (4.1). In the ideal world, $\mathcal{F}_{\mathsf{VOLE}}^{p,q,\mathbb{F}_p,\ell,\mathcal{L}}$ will pick Δ randomly, receive $\vec{u}, \vec{v}$ from the simulator, and compute $\vec{w} = \vec{u}\,\Delta + \vec{v}$. These are identical, implying that these two worlds are indistinguishable and that this case is secure.

Corrupt P_R. After receiving F^*, x^* from $\mathcal{A}$, the simulator will compute $\Delta = x^*$ and $\vec{w}$ honestly, and submit them to $\mathcal{F}_{\mathsf{VOLE}}^{p,q,\mathbb{F}_p,\ell,\mathcal{L}}$. We do a hybrid proof, starting from the real world and going to the ideal world.

1. In the real world, $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{q,1,\mathcal{L}}$ sets $F(x) = F^*(x)$ for $x \neq x^*$, generates $F(x^*)$ randomly, and sends them to P_S, who will compute $\vec{r}_x = \mathsf{PRG}(F(x))$ and $\vec{u}, \vec{v}$. By Eq. (4.1), $\vec{v} = \vec{w} - \vec{u}\,\Delta$.
2. Because $F(x^*)$ is only used to compute $\vec{r}_{x^*}$, the security of PRG implies that $\vec{r}_{x^*}$ can be replaced with a uniformly sampled value.
3. Instead of sampling $\vec{r}_{x^*}$ randomly, sample $\vec{u}$ uniformly at random and set $\vec{r}_{x^*} = \vec{u} - \sum_{x \neq x^*} \vec{r}_x$. This is an identical distribution.
4. We are now at the ideal world, where $\mathcal{F}_{\mathsf{VOLE}}^{p,q,\mathbb{F}_p,\ell,\mathcal{L}}$ will pick $\vec{u}$ randomly, receive $\Delta, \vec{w}$ from the simulator, and compute $\vec{v} = \vec{w} - \vec{u}\,\Delta$.

If both parties are corrupt then security is trivial, as then the simulator can just forward messages between the corrupted parties. □

Efficient Computation. Let a be a generator of $\mathbb{F}_q$ over $\mathbb{F}_p$. For computation, it's convenient to represent $\vec{v}$ as a sequence of $\mathbb{F}_p$ vectors: $\vec{v} = \vec{v}_0 + a\vec{v}_1 + \cdots + a^{k-1}\vec{v}_{k-1}$. Similarly, the index x becomes $x_0 + ax_1 + \cdots + a^{k-1}x_{k-1}$. Naïve computation of $\vec{v}$ using the sum then becomes $\vec{v}_i = \sum_x x_i \vec{r}_x$, but this would require $O(kq)$ vector additions and scalar multiplications over $\mathbb{F}_p$.

This can be improved to $O(q + \frac{q}{p} + \frac{q}{p^2} + \cdots) = O(q)$ vector additions and no scalar multiplications. For all $x' \in \mathbb{F}_q$ where $x'_0 = 0$, let $\vec{r}'_{x'} = \sum_{x_0 \in \mathbb{F}_p} \vec{r}_{x'+x_0}$, and notice that all $\vec{v}_1, \ldots, \vec{v}_{k-1}$ (and $\vec{u}$) depend only on the $\vec{r}'_{x'}$. Therefore, after computing all $\frac{q}{p}$ vectors $\vec{r}'_{x'}$, the outputs $\vec{v}_1, \ldots, \vec{v}_{k-1}$ can be found by recursion on a smaller problem size. As a byproduct, computing the $\vec{r}'_{x'}$ produces sequences of partial sums $\sum_{x_0 \leq i} \vec{r}_{x'+x_0}$, and adding all of these together then gives $\sum_{x'} \sum_{x_0} (p - x_0)\vec{r}_{x'+x_0} = \vec{v}_0$. P_R can use the same algorithm to compute $\vec{w}$ by just reordering the $\vec{r}_x$ vectors at the start, because $\sum_x \vec{r}_x(\Delta - x) = \sum_x \vec{r}_{x+\Delta}(-x)$.

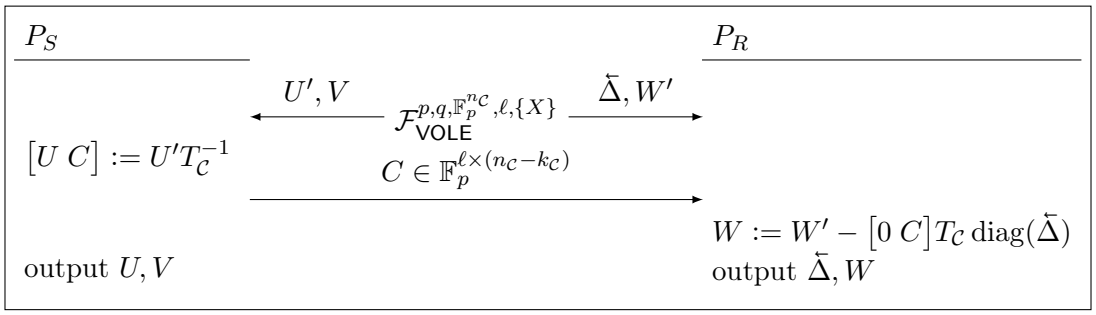


Figure 4.8: Protocol for subspace VOLE.

Concatenation. While this does not directly follow directly from the UC theorem, it should be clear that running the protocol Fig. 4.7 on a batch of n OTs will produce a batch of n VOLEs. The proof trivially generalizes. More precisely, it achieves $\mathcal{F}_{\mathsf{VOLE}}^{p,q,\mathbb{F}_p^n,\ell,\mathcal{L}}$ in the $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{q,n,\mathcal{L}}$ hybrid model, where $\mathbb{F}_p^n$ is the trivial code with $G_{\mathbb{F}_p^n} = \mathbb{1}_n$. This will be the basis for our subspace VOLE.

4.3.2 For Subspaces

For $\binom{2}{1}$-OT extension, the next step would be for P_S to send a correction to make all columns of U be identical, so that each column would use the same set of choice bits. Efficient $\binom{N}{1}$-OT extension protocols like [KK13] instead must correct the rows of U to lie in an arbitrary linear code $\mathcal{C}$, rather than the repetition code. We implement subspace VOLE to handle these more general correlations.

Our protocol for subspace VOLE is presented in Fig. 4.8. It starts out with a VOLE correlation $W' - V = U'\,\mathrm{diag}(\Delta)$. Then, P_S divides U' into parts, the message $U \in \mathbb{F}_p^{\ell\times k_\mathcal{C}}$ and the correction syndrome $C \in \mathbb{F}_p^{\ell\times n_\mathcal{C}-k_\mathcal{C}}$, sending the correction to P_R. P_R then corrects W to maintain the VOLE correlation property after P_S removes C. Unfortunately, P_S can just lie when it sends C to P_R, so the protocol only achieves semi-honest security. Since the leakage set $\mathcal{L}$ only matters for malicious security, we simplify by assuming that $\mathcal{L}$ is trivial (i.e. $\{X\}$).

Theorem 4.3.2. *The protocol in Fig. 4.8 is a semi-honest realization of* $\mathcal{F}_{\mathsf{VOLE}}^{p,q,\mathcal{C},\ell,\{X\}}$ *in the* $\mathcal{F}_{\mathsf{VOLE}}^{p,q,\mathbb{F}_p^n,\ell,\{X\}}$ *hybrid model.*

Proof. First, the protocol outputs correctly satisfy the VOLE correlation:

$$\begin{aligned} W &= W' - \big[0\ C\big] T_{\mathcal{C}} \operatorname{diag}(\vec{\Delta}) \\ &= V + U' \operatorname{diag}(\vec{\Delta}) - \big[0\ C\big] T_{\mathcal{C}} \operatorname{diag}(\vec{\Delta}) \\ &= V + \big(\big[U\ C\big] T_{\mathcal{C}} - \big[0\ C\big] T_{\mathcal{C}}\big) \operatorname{diag}(\vec{\Delta}) \\ &= V + U G_{\mathcal{C}} \operatorname{diag}(\vec{\Delta}). \end{aligned}$$

For security, notice that any $U, V, \vec{\Delta}, W$ output by the protocol and any C that the adversary eavesdrops on (because the communication is over an authenticated, but not private, channel) corresponds to a unique $U', V, \vec{\Delta}, W'$ from the underlying VOLE. Specifically, $U' = \big[U\ C\big] T_{\mathcal{C}}$ and $W' = W + \big[0\ C\big] T_{\mathcal{C}} \operatorname{diag}(\vec{\Delta})$. This implies the adversary does not learn anything new by corrupting either party, as they could already predict what that party knows. They only gain the power to program that the base VOLE's outputs for that party, but the simulator gains the corresponding power to program that party's protocol outputs to match. In more detail, $\mathcal{S}$ should receive from $\mathcal{A}$ the programed base VOLE outputs for the corrupted parties, simulate doing exactly what they would do in the protocol (while sampling a fake $C \xleftarrow{\$} \mathbb{F}_p^{\ell \times (n_{\mathcal{C}} - k_{\mathcal{C}})}$ if P_S is honest), and program the protocol outputs to be the result.

In the ideal world, $\mathcal{S}$ generates a uniformly random consistent adversary view $U, V, \vec{\Delta}, W$ (together with U' or W' if P_S or P_R was corrupted). In the real world, the underlying VOLE functionality picks $U', V, \vec{\Delta}, W'$ uniformly at random subject to the constraints of the VOLE correlation and any outputs programmed by the adversary, and then the adversary gets to see the protocol run. There is a bijection between consistent adversary views and outputs of the underlying VOLE $U', V, \vec{\Delta}, W'$, and this bijection implies that these two views are identically distributed. □

4.4 Malicious Security

Our small field VOLE construction in Section 4.3.1 was easily proved maliciously secure. It does not involve any communication, and so there are no opportunities for any of the parties to lie. However, Section 4.3.2 requires P_S to reveal part of U, allowing a malicious P_S to lie. Following KOS and OOS, we solve this by introducing a consistency check (Fig. 4.9) that is run immediately afterwards, to provide a guarantee that if P_S lies then the protocol will either abort or work properly. Then the last few rows of U, V, and W are thrown away so

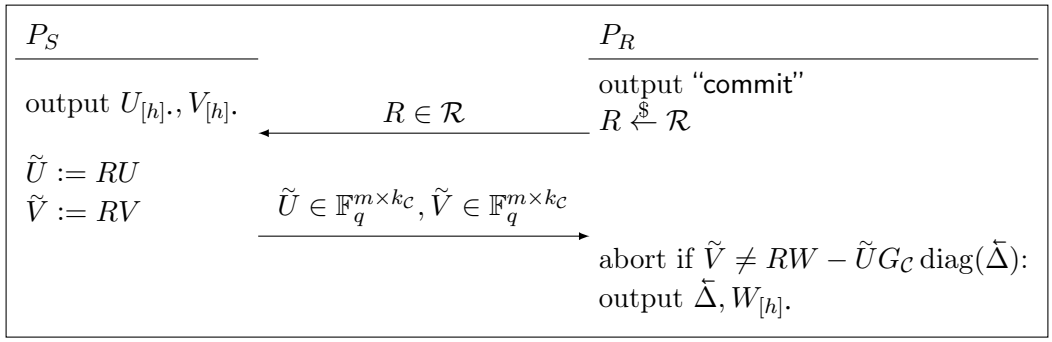


Figure 4.9: Consistency checking protocol, which should be used with Fig. 4.8. $\mathcal{R}$ must be a ϵ-universal hash family, where all $R \in \mathcal{R}$ is $\mathbb{F}_p^h$-hiding. The "abort if" means that "check failed" is output if the check fails. If instead of giving $\vec{\Delta}, W'$ to P_R, the base VOLE outputs "check failed", P_R should continue to play along with the protocol and only output "check failed" when it completes.

that the values revealed in the consistency check do not leak anything. This still allows the possibility of selective abort attacks, however.

KOS, OOS, PSS, and CCG all compute their consistency checks by multiplying each row of U with a random value — an element of an extension field for KOS or just a vector for OOS and PSS. V and W are also multiplied by random values, in a consistent way. We follow [CDD+16] in generalizing this to use linear universal hashes. Any linear ϵ-almost universal hash family $\mathcal{R} \subseteq \mathbb{F}_q^{m \times \ell}$ will work, as long as the following condition is met by every $R \in \mathcal{R}$, which guarantees that throwing away the last few rows of U is sufficient to keep the others hidden.

Definition 4.4.1. *A matrix $R \in \mathbb{F}_q^{m \times \ell}$ is* $\mathbb{F}_p^h$-hiding *if the first h inputs to R will stay hidden when the remaining inputs are secret and uniformly random. More precisely, if $\vec{x} \xleftarrow{\$} \mathbb{F}_p^\ell$ then $R\vec{x}$ must be independently random from $\vec{x}_{[h]}$*

Note that if R is $\mathbb{F}_p^h$-hiding then that it is $\mathbb{F}_q^h$-hiding, so if R is able to keep $U \in \mathbb{F}_p^{\ell \times k_\mathcal{C}}$ hidden then it will keep $V \in \mathbb{F}_q^{\ell \times n_\mathcal{C}}$ hidden as well.

Many useful universal hashes with elements in $\mathbb{F}_p$ satisfy this definition, including hashes based on polynomial evaluation or cyclic redundancy checks. That is, the last m columns of R will span the others, so R will be $\mathbb{F}_p^h$-hiding for $h = \ell - m$. However, this only works if the universal hash is over $\mathbb{F}_p$, rather than $\mathbb{F}_q$, as otherwise there won't be enough entropy in

the last m columns to completely hide the other inputs. On the other hand, using a hash over $\mathbb{F}_q$ gives better compression. For a universal hash over $\mathbb{F}_p$, the best possible ϵ is about p^{-m}, while for $\mathbb{F}_q$ it is $q^{-m} = p^{-km}$. We believe that the best approach is to compose two universal hashes, first applying a $\mathbb{F}_p^{\ell-m'}$-hiding hash $R \in \mathcal{R} \subseteq \mathbb{F}_p^{m' \times \ell}$, then further reducing the output down to m entries with a second hash $R' \in \mathcal{R}' \subseteq \mathbb{F}_q^{m \times m'}$ where $m' \geq km$. The composed hash will be $\mathbb{F}_p^{\ell-m'}$-hiding, and will still be universal by Theorem 4.2.3.

Remark 4.4.2. *P_S outputs $U_{[h]}., V_{[h]}$. in the first round, just after sending C and much before the protocol has actually completed. In applications where U will be derandomized immediately (e.g. chosen point OT extension), it is convenient to derandomize U at the same time as sending C. The protocol returning early is what allows this within the UC framework.*

Remark 4.4.3. *After sending C, P_S will not have many useful options to choose from, so the protocol notifies P_R with "commit" (as in $\mathcal{F}_{\text{VOLE-pre}}^{p,q,\mathcal{C},h,\mathcal{L},M}$) to indicate that P_S's inputs (mostly) fixed. In Section 4.5, this notification is used to send a second universal hash at the same time as R.*

4.4.1 Flaws in Existing Consistency Checks

Given the similarity of Fig. 4.9 to the KOS, PSS, and OOS consistency checks, it seems natural to adapt their proofs to the subspace VOLE consistency checking protocol. However, it turns out that all three are flawed. We first present the flaw in OOS, because it is most similar to our protocol.

4.4.1.1 Flaw in OOS's Proof.

To get the OOS consistency check, take the protocol in Fig. 4.9 and set $p = q = 2$ and $R = \begin{bmatrix} X & \mathbb{1}_m \end{bmatrix}$, where $X \xleftarrow{\$} \mathbb{F}_2^{m \times \ell - m}$ is uniformly random. There are a couple of differences, but these do not affect the consistency check proper. Our sender is their receiver and vice versa, because they are implementing OT extension and we are doing subspace VOLE. And, they send a correction C for the whole of U' at once, instead of just the syndrome, because their OT choice bits are chosen rather than random. To avoid the confusion of introducing a separate set of notations for essentially the same protocol, we ignore these differences and discuss their proof using our notation and protocol. See Appendix C.2 for a discussion using OOS's original language.

Let $[U\ \bar{C}] = U'T_{\mathcal{C}}^{-1} \oplus \begin{bmatrix}0\ C\end{bmatrix}$, so $\bar{C}$ is the error in the correction syndrome C sent by the malicious P_S. Similarly, let $\bar{U} = RU \oplus \tilde{U}$ and $\bar{V} = RV \oplus \tilde{V}$ be the errors in the consistency check messages sent by P_S. The consistency check then becomes $\bar{V} = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\tilde{\Delta})$ (see the proof of Theorem 4.4.5 for details). OOS define a set $E \subseteq [n_{\mathcal{C}}]$ of column indices i where $\left([\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\right)_{\cdot i}$ is nonzero. These are the indices i where Δ_i will have to be guessed by P_S in order to pass the consistency check. They then attempt to prove that the indices in E will be the only ones that P_S lied about. That is, their simulator tries to correct U to get P_S's real output $U^\star$, so that if $Z = [U^\star\ C]T_{\mathcal{C}} \oplus U'$ then the indices of all the nonzero columns of Z are in E. This would let $\mathcal{S}$ update V accordingly, getting $V^\star = V \oplus Z\,\mathrm{diag}(\tilde{\Delta})$, which it could find because P_S must guess Δ_i for $i \in E$.

The flaw is in their proof that $\mathcal{S}$ can (with high probability, assuming that the check passes) extract $U^\star$. Their technique is to look at $Y = [U\ \bar{C}]T_{\mathcal{C}} = U' \oplus \begin{bmatrix}0\ C\end{bmatrix}T_{\mathcal{C}}$, whose rows would be in $\mathcal{C}$ if P_S were honest, and remove the columns in E to get a punctured matrix Y_{-E}. Then they decode the rows of Y_{-E} using the punctured code $\mathcal{C}_{-E}$ to get $U^\star$, since $Y \oplus Z = U^\star G_{\mathcal{C}}$ and Z_{-E} should be 0. For this to work, they need the rows of Y_{-E} to be in $\mathcal{C}_{-E}$. They try to prove this using the following lemma.

Lemma 4.4.4 (OOS, Lem. 1)**.** *Let $\mathcal{D}$ be a linear code and $B \in \mathbb{F}_2^{\ell \times n_{\mathcal{D}}}$ be a matrix, where not all rows of B are in $\mathcal{D}$. If $X \xleftarrow{\$} \mathbb{F}_2^{m \times \ell - m}$ and $R = \begin{bmatrix}X\ \mathbb{1}_m\end{bmatrix}$, then the probability that all rows of RB are in $\mathcal{D}$ is at most 2^{-m}.*

They apply this lemma with $\mathcal{D} = \mathcal{C}_{-E}$ and $B = Y_{-E}$. Note that $RY = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}} \oplus \tilde{U}G_{\mathcal{C}}$, so $RY_{-E} = \tilde{U}G_{\mathcal{C}_{-E}}$ has all rows in $\mathcal{C}_{-E}$. They conclude that with all but negligible probability, all rows of Y_{-E} are in $\mathcal{C}_{-E}$. However, the lemma cannot be used in this way. The lemma requires that $\mathcal{D}$ and B be fixed in advance, *before* X is sampled, yet $\mathcal{C}_{-E}$ and Y_{-E} both depend on E. Recall that E is the set of nonzero columns of $[\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}$, which depends on both R directly, and on the consistency check message $\tilde{U}$ sent by P_S *after* it learns X.

While this shows that OOS's proof is wrong, we have not found any attacks that contradict their theorem statement. Additionally, a special case of our new proof (Theorem 4.4.5) shows that the OOS protocol is still secure, with statistical security only one bit less than was claimed.

4.4.1.2 Attack For PSS's Protocol.

The PSS consistency checking protocol is similar to OOS's, though they only consider Walsh–Hadamard codes, and they generate $R \xleftarrow{\$} \mathbb{F}_2^{m\times\ell}$ using a coin flipping protocol. In Lemma IV.5, they have a similar proof issue to OOS, using Corollary IV.2 on dependent values when the corollary assumes they are independent. However, we focus on a more significant problem, which we summarize here, using our own notation. See Appendix C.3 for a more detailed discussion, using their notation.

The most important difference from OOS is that PSS attempt to compress the consistency check by summing the columns of $\tilde{V}$ to get $\tilde{v} = \tilde{V}[1 \ \cdots \ 1]^\top$. The consistency check is then that $\tilde{v}$ must equal $\big(RW \oplus \tilde{U}G_{\mathcal{C}}\,\mathrm{diag}(\bar{\Delta})\big)[1 \ \cdots \ 1]^\top = RW[1 \ \cdots \ 1]^\top \oplus \tilde{U}G_{\mathcal{C}}\vec{\Delta}$. Let $\bar{C}$, $\bar{U}$, and $\bar{v}$ be defined analogously to our discussion of OOS. Then the consistency check is $\bar{v} = [\bar{U} \ \ R\bar{C}]T_{\mathcal{C}}\vec{\Delta}$. This means that a malicious receiver only needs to guess XORs of multiple bits from $\bar{\Delta}$, rather than the individual bits themselves.

We used this to create an attack against PSS. Have P_S lie about the bits in U' in length N intervals, where in the first OT it lies about the first N bits of $U'_{0,}$, and in the next OT the second N bits of $U'_{1,}$, and so on. Here, N is a parameter defining the tradeoff between computational cost and attack success rate. Then $[\bar{U} \ \ R\bar{C}]T_{\mathcal{C}}$ will have rows spanned by these N bit intervals, so $[\bar{U} \ \ R\bar{C}]T_{\mathcal{C}}\vec{\Delta}$ only depends on $\lceil \frac{n_{\mathcal{C}}}{N} \rceil$ different values: $\bigoplus_{j=0}^{N-1} \Delta_{Ni+j}$ for $i \in \left[\lceil \frac{n_{\mathcal{C}}}{N} \rceil\right]$. Therefore, the consistency check passes with probability $2^{-\lceil n_{\mathcal{C}}/N \rceil}$, even though we have lied about all $n_{\mathcal{C}}$ bits. Later, having gotten away with these lies, the hashes output by the OT extension can be brute forced to solve for each N-bit chunk of $\bar{\Delta}$ individually. This breaks the OT extension in time $\lceil \frac{n_{\mathcal{C}}}{N} \rceil 2^{N-1}$. At the $\lambda = 128$ security level, $n_{\mathcal{C}} = 256$, so by setting $N = 32$ we get an attack with success probability 2^{-8} that uses only 2^{34} hash evaluations.

4.4.1.3 Flaw in KOS's Proof.

Like with OOS, in this section we will discuss KOS by analogy with our consistency checking protocol Fig. 4.9. See Appendix C.4 for a more detailed account, using KOS's notation for their protocol.

To turn out consistentcy check into KOS's, start by fixing $p = q = 2$ and $\mathcal{C} = \mathsf{Rep}(\mathbb{F}_2^\lambda)$. Let $\mathcal{R} = \mathbb{F}_2^{\lambda\times\ell}$, which means that R is $\mathbb{F}_2^{\ell-\lambda-\sigma}$-hiding with probability at least $1 - 2^{-\sigma}$.

They use a coin flipping protocol to make sure that P_R cannot pick an R that is is not hiding. Let α a primitive element of $\mathbb{F}_{2^\lambda}$, meaning that $\{1, \alpha, \ldots, \alpha^{\lambda-1}\}$ is a basis for $\mathbb{F}_{2^\lambda}$ over $\mathbb{F}_2$. The first half of the consistency check, $\widetilde{U}$, works as normal, except that it gets encoded into a field element $u = \bigoplus_i \widetilde{U}_i.\alpha^i = \vec{\alpha}^\top \widetilde{U}$, where $\vec{\alpha} = [1, \alpha, \ldots, \alpha^{\lambda-1}]^\top$. The other half, $\widetilde{V}$, is compressed from λ^2 bits down to λ bits by turning it into a single field element $v = \bigoplus_{ij} \widetilde{V}_{ij}\alpha^{i+j} = \vec{\alpha}^\top \widetilde{V} \vec{\alpha}$. Similarly, let $w = \vec{\alpha}^\top RW\vec{\alpha}$ and $\delta = \widetilde{\Delta}\vec{\alpha}$. Then the consistency check becomes

$$\begin{aligned} v &= \vec{\alpha}^\top RW\vec{\alpha} \oplus \vec{\alpha}^\top \widetilde{U} G_{\mathcal{C}} \operatorname{diag}(\widetilde{\Delta})\vec{\alpha} \\ &= w \oplus uG_{\mathcal{C}} \operatorname{diag}(\widetilde{\Delta})\vec{\alpha} = w \oplus u[1 \ \cdots \ 1] \operatorname{diag}(\widetilde{\Delta})\vec{\alpha} = w \oplus u\delta. \end{aligned}$$

Because $\mathcal{C}$ is a repetition code, U' is supposed to be derandomized so that all columns are identical to U. Let $Y = U' \oplus \begin{bmatrix} 0 & C \end{bmatrix} T_{\mathcal{C}}$ be the derandomization of U'. Then columns i and j are called *consistent* if they imply the same values of U, i.e. if $Y_{\cdot i} = Y_{\cdot j}$. Also let S_Δ be the set of possible Δ that cause the consistency check to succeed. KOS's proof of security for malicious P_S depends entirely on their Lemma 1, which states several properties of their consistency check. Most importantly, it implies that for any u, v sent by P_S, with probability $1 - 2^{-\lambda}$ there exists $k \in \mathbb{N}$ such that $|S_\Delta| = 2^k$ and k is at most the size of the largest group of consistent columns.

KOS gave no proof for Lemma 1, instead citing the full version of their paper, which has not been made public. However, the authors of KOS were kind enough to give an unpublished draft [KOS21]. Unfortunately, their proof has a similar flaw to OOS's, because they assume that R is sampled after S_Δ is known.

Unlike OOS, we found a counterexample to show that KOS's Lemma 1 is false, which we call a collision attack. Let the malicious P_S choose C uniformly at random (so Y will also be uniformly random) but still provide an honest v during the consistency check. Because of the correction P_R applies, W will be

$$W = V \oplus (U' \oplus \begin{bmatrix} 0 & C \end{bmatrix} T_{\mathcal{C}}) \operatorname{diag}(\widetilde{\Delta}) = V \oplus Y \operatorname{diag}(\widetilde{\Delta})$$

Let $\tilde{y} = \vec{\alpha}^\top RY$. The consistency check is then

$$v = \vec{\alpha}^\top RV\vec{\alpha} \oplus \vec{\alpha}^\top RY \operatorname{diag}(\tilde{\Delta})\vec{\alpha} \oplus uG_{\mathcal{C}} \operatorname{diag}(\tilde{\Delta})\vec{\alpha}$$
$$0 = (\tilde{y} \oplus u[1 \ \cdots \ 1]) \operatorname{diag}(\tilde{\Delta})\vec{\alpha}.$$

If u is set to be some element y_i of $\tilde{y}$, the consistency check at least won't depend on Δ_i. Since Y is uniformly random, $\tilde{y}$ will be as well, so the probability of a collision among the y_i is roughly $\lambda^2 2^{-\lambda-1}$. If there is a collision $y_i = y_j$ and P_R sets $u = y_i$, then $|S_\Delta| = 2^k = 4$. This contradicts KOS's Lemma 1 because k should be at most 1 as no two columns are consistent.

In Appendix C.4.2 we present (using KOS's notation) a stronger attack against special parameters of KOS. Assuming that a certain MinRank problem always has a solution (and heuristically it should have $2^{\lambda/5}$ solutions on average), the attack succeeds in recovering Δ with probability $2^{-\frac{3}{5}\lambda}$ using $O(2^{\lambda/5})$ random oracle queries. While this is still not a practical attack, according to KOS's proof of their Theorem 1, an attack with this few random oracle queries should only succeed with probability $O(2^{-\frac{4}{5}\lambda})$.

4.4.2 Our New Proof

The biggest hurdle in the proof is the case where P_S is malicious. If P_S lies when it sends C, then it will have to guess some entries of Δ, but which entries depends on what $\widetilde{U}$ it decides to send. As with OOS's flawed proof, P_S does not have to make up its mind until after seeing R, and generally speaking universal hashes are only strong when used on data that was chosen independently of the hash. We need to find some property that only depends on C and R so that we can show that it holds (with high probability) based on C being independent of R, then use it to prove security.

The property we found was that R should preserve all the lies in C. More precisely, if $\bar{C}$ is the difference between the honest C and the one P_S sent, then $R\bar{C}$ and $\bar{C}$ should have the same row space.[5] The idea is that, if R were the identity, the consistency check would clearly ensure that whatever incorrect value C that P_S provides, it can still guess matrices U, V that make the VOLE correlation hold. Although R is not the identity matrix, the check still ensures that the VOLE correlation holds for $\widetilde{U}, \widetilde{V}$. The lie-preserving property of R then

[5]This fails if there are too many lies; however the VOLE would likely abort anyway.

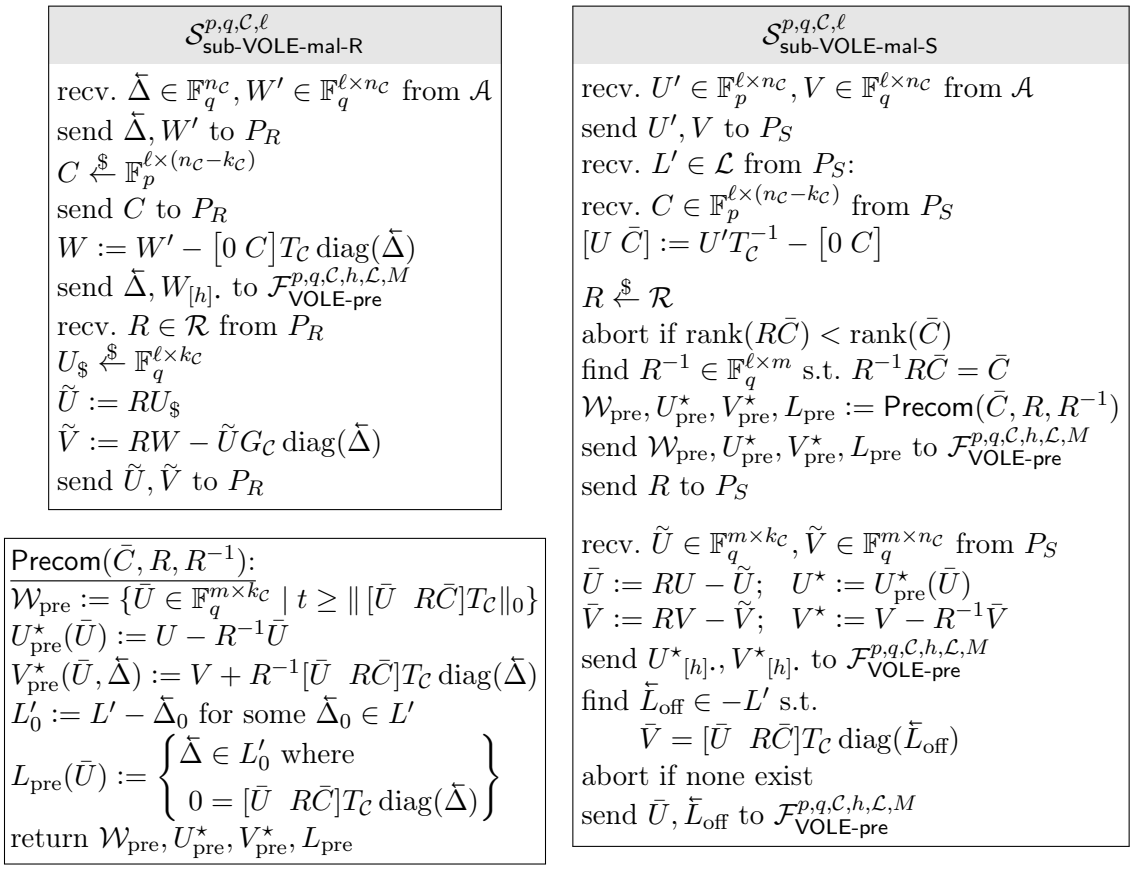


Figure 4.10: Simulators for malicious security of Fig. 4.8 combined with Fig. 4.9, for a single corrupt party. $\mathcal{S}^{p,q,\mathcal{C},\ell}_{\text{sub-VOLE-mal-R}}$ is for corrupt P_R, while $\mathcal{S}^{p,q,\mathcal{C},\ell}_{\text{sub-VOLE-mal-S}}$ is for corrupt P_S.

shows that they contain enough information to correct the whole of U and V so that they do satisfy the VOLE correlation.

The proof of [CDD+16] is based on a similar lie-preserving property, but they analyze this property independently from the consistency check. This leads to a bound of $\Theta(\sqrt{\epsilon})$ on the distinguisher's advantage. We instead consider these events together, because the distinguisher only succeeds when it violates the property *and* passes the consistency check. The product of these event's probabilities is smaller than either individual probability, so we prove a much smaller distinguisher advantage bound of $\Theta(\epsilon)$.

Theorem 4.4.5. *The subspace VOLE protocol in Fig. 4.8 combined with the consistency*

checking protocol in Fig. 4.9 is a maliciously secure implementation of $\mathcal{F}_{\text{VOLE-pre}}^{p,q,\mathcal{C},h,\mathcal{L},M}$ *if* $\mathcal{L} \supseteq \text{Affine}(\mathbb{F}_q^{n_\mathcal{C}})$*, assuming that* $\mathcal{R} \subseteq \mathbb{F}_q^{m\times\ell}$ *is a* ϵ*-almost universal family where all* R *are* $\mathbb{F}_p^h$*-hiding. The distinguisher has advantage at most* $\frac{\epsilon q}{q-1} + q^{-t-1}$*, where* $t = \frac{d_\mathcal{C}}{1+\sqrt{1+\frac{d_\mathcal{C}}{n_\mathcal{C}} - \frac{1}{n_\mathcal{C}^2}}} \geq \frac{d_\mathcal{C}}{2}$ *and* $M = n_\mathcal{C}(d_\mathcal{C} - t)$*.*

Note: when instantiated as in OOS, $\epsilon = 2^{-m}$ and $q = 2$, so our proof shows that OOS has only 1 bit less statistical security than was claimed. The q^{-t-1} term only matters for the pre-commitment property, which OOS does not consider.

Proof. There are four cases, depending on which parties are corrupted. If both parties are corrupted then the real protocol can be simulated trivially, by ignoring the ideal functionality and just passing messages between the corrupted parties. If both players are honest, the situation is very similar to the semi-honest protocol (Theorem 4.3.2). The only difference is the additional two rounds, which can be simulated by picking a random $R \in \mathcal{R}$, as well as sampling fake P_S values $U_\$ \xleftarrow{\$} \mathbb{F}_p^{\ell\times k_\mathcal{C}}$ and $V_\$ \xleftarrow{\$} \mathbb{F}_p^{\ell\times n_\mathcal{C}}$ and simulating the third round as $\tilde{U} = RU_\$, \tilde{V} = RV_\$$. Since both parties are honest, U and V are uniformly random, and so Theorem 4.4.1 guarantees that these fakes are indistinguishable from the real consistency check.

The situation is similar when only P_R is corrupted (simulator in Fig. 4.10, top left). Following the same principle as for the semi-honest protocol, $\mathcal{S}$ starts by performing the computations that an honest P_R would, while randomly sampling a fake syndrome C to send. To simulate the consistency check, after receiving R, the simulator fakes $\tilde{U}$ like in the honest–honest case, then solves for $\tilde{V}$ as the only possibility that will pass the consistency check. The real protocol and the simulation are indistinguishable because the honesty of P_S implies that the consistency check will always pass, so the formula for $\tilde{V}$ must always hold, and P_R cannot tell that $\tilde{U}$ was generated from the fake $U_\$$ because R is $\mathbb{F}_p^h$-hiding.

The most interesting case is when P_S is corrupt. We present a hybrid proof, starting with the real world, where the real protocol gets executed using the underlying ideal functionality $\mathcal{F}_{\text{VOLE}}^{p,q,\mathbb{F}_p^{n_\mathcal{C}},\ell,\mathcal{L}}$, and work towards the ideal world, where the simulator (Fig. 4.10, right) liaises between the corrupted sender and the desired ideal functionality $\mathcal{F}_{\text{VOLE-pre}}^{p,q,\mathcal{C},h,\mathcal{L},M}$.

1. Compute what P_S's honest output would be, and the difference between the honest syndrome and the one P_S provided: $[U\ \bar{C}] = U'T_\mathcal{C}^{-1} - [0\ C]$. Add a check after P_S sends $\tilde{U}$ and $\tilde{V}$, where if $\text{rank}(R\bar{C}) < \text{rank}(\bar{C})$, "check failed" is sent to P_R and the

protocol aborts. The environment's advantage for this step is the probability that this abort triggers and the protocol would not have aborted anyway. We bound this probability using the following lemma.

Lemma 4.4.6. *Let $\mathcal{R} \subseteq \mathbb{F}_q^{m\times n}$ be a linear ϵ-almost universal family, and let A be any matrix in $\mathbb{F}_q^{n\times l}$. Then, $\mathbb{E}_{R\xleftarrow{\$}\mathcal{R}}\left[q^{\operatorname{rank}(A)-\operatorname{rank}(RA)}-1\right] \le \epsilon(q^{\operatorname{rank}(A)}-1)$.*

Proof. By the rank–nullity theorem, R defines an isomorphism $\mathbb{F}_q^n/\ker(R) \cong \operatorname{colspace}(R)$. Its restriction to $\operatorname{colspace}(A)$ gives an isomorphism $\operatorname{colspace}(A)/\ker(R) \cong \operatorname{colspace}(RA)$. Therefore,

$$\begin{aligned}\operatorname{rank}(RA) &= \dim\big(\operatorname{colspace}(RA)\big)\\ &= \dim(\operatorname{colspace}(A)) - \dim(\operatorname{colspace}(A)\cap\ker(R))\\ &= \operatorname{rank}(A) - \dim(\operatorname{colspace}(A)\cap\ker(R)).\end{aligned}$$

We then want to bound the expected value of $X = q^{\dim(\operatorname{colspace}(A)\cap\ker(R))}-1 = |\operatorname{colspace}(A)\cap\ker(R)\setminus\{0\}|$. That is, X is the number of nonzero $v\in\operatorname{colspace}(A)$ such that $Rv=0$. By Theorem 4.2.1, for any particular $v\neq 0$ the probability that $Rv=0$ is at most ϵ. Since X is the sum of $|\operatorname{colspace}(A)\setminus\{0\}| = q^{\operatorname{rank}(A)}-1$ indicator random variables, we get $\mathbb{E}[X]\le\epsilon(q^{\operatorname{rank}(A)}-1)$. □

For the real protocol to not abort, $\widetilde{V} = RW - \widetilde{U}G_{\mathcal{C}}\operatorname{diag}(\vec{\Delta})$ must hold. Because P_R is uncorrupted, $\vec{\Delta}$ is sampled uniformly in $\mathbb{F}_q^{nc}$ and W' is computed as $U'\operatorname{diag}(\vec{\Delta})+V$. Therefore,

$$\begin{aligned}W = W' - \big[0\ C\big]T_{\mathcal{C}}\operatorname{diag}(\vec{\Delta}) &= (U' - \big[0\ C\big]T_{\mathcal{C}})\operatorname{diag}(\vec{\Delta}) + V\\ &= [U\ \bar{C}]T_{\mathcal{C}}\operatorname{diag}(\vec{\Delta}) + V.\end{aligned}$$

Let $\bar{U} = RU - \widetilde{U}$ and $\bar{V} = RV - \widetilde{V}$ be the differences between the honest consistency check messages and the ones sent by P_S. Then the consistency check is equivalent to $-\bar{V} = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\operatorname{diag}(\vec{\Delta})$. Next, we need to bound

$$\begin{aligned}P &= \Pr\big[\text{abort}\wedge\text{check passes}\big]\\ &= \Pr\big[\operatorname{rank}(R\bar{C}) < \operatorname{rank}(\bar{C}) \wedge -\bar{V} = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\operatorname{diag}(\vec{\Delta})\big].\end{aligned}$$

Triggering this condition requires guessing $[\bar{U} \;\; R\bar{C}]T_{\mathcal{C}} \operatorname{diag}(\vec{\Delta})$, i.e. guessing Δ_i for every nonzero column $\big([\bar{U} \;\; R\bar{C}]T_{\mathcal{C}}\big)_{\cdot i}$. Let $N = \| [\bar{U} \;\; R\bar{C}]T_{\mathcal{C}} \|_0$ be the number of these nonzero columns. A lower bound for N is $\operatorname{rank}([\bar{U} \;\; R\bar{C}]T_{\mathcal{C}})$, because every zero column does not contribute to the rank. $T_{\mathcal{C}}$ is invertible, so multiplying by it does not change the rank. Adding extra columns only increases rank, so $\operatorname{rank}([\bar{U} \;\; R\bar{C}]) \geq \operatorname{rank}(R\bar{C})$. Up until the consistency check, the behavior of P_R has been independent of $\vec{\Delta}$, and N is also independent of $\vec{\Delta}$, so $\Pr\big[\text{check} \mid N\big] \leq q^{-N}$. Let $r = \operatorname{rank}(\bar{C}) - \operatorname{rank}(R\bar{C})$, so $N \geq \operatorname{rank}(\bar{C}) - r$. Then $P \leq \mathbb{E}\big[q^{-\operatorname{rank}(\bar{C})+r} \, \mathbb{1}_{r\geq 1}\big]$, since the added abort occurs exactly when $r \geq 1$, and expectation of conditional probability is marginal probability.

Now, apply Theorem 4.4.6 to $\bar{C}$ to get $\mathbb{E}[q^r - 1] \leq \epsilon(q^{\operatorname{rank}(\bar{C})} - 1)$. If $r \geq 1$ then $\frac{q^r}{q^r-1} \leq \frac{q}{q-1}$. Multiply both sides by $q^r - 1$ to get

$$q^r \mathbb{1}_{r\geq 1} \leq \frac{q}{q-1}(q^r - 1).$$

$$P \leq \mathbb{E}\big[q^{-\operatorname{rank}(\bar{C})+r} \mathbb{1}_{r\geq 1}\big] \leq \epsilon \frac{q}{q-1} \frac{(q^{\operatorname{rank}(\bar{C})} - 1)}{q^{\operatorname{rank}(\bar{C})}} \leq \epsilon \frac{q}{q-1}$$

2. After checking that $\operatorname{rank}(R\bar{C}) = \operatorname{rank}(\bar{C})$, find $R^{-1} \in \mathbb{F}_q^{\ell \times m}$ such that $R^{-1}R\bar{C} = \bar{C}$. To do this, find the reduced row echelon forms $F = AR\bar{C}$ and $F' = B\bar{C}$ of $R\bar{C}$ and $\bar{C}$, where $A \in \mathbb{F}_q^{m\times m}$ and $B \in \mathbb{F}_p^{\ell\times\ell}$ are invertible matrices. Because they have the same rank, $R\bar{C}$ and $\bar{C}$ must have the same row space. The uniqueness of reduced row echelon forms implies that all nonzero rows of F and F' will be identical, so

$$F' = \begin{bmatrix} F \\ 0 \end{bmatrix} \quad \text{and} \quad \bar{C} = B^{-1}F' = B^{-1} \begin{bmatrix} \mathbb{1}_m \\ 0 \end{bmatrix} F = B^{-1} \begin{bmatrix} \mathbb{1}_m \\ 0 \end{bmatrix} AR\bar{C},$$

which gives a formula for R^{-1}.

Correct P_S's VOLE correlation as $U^\star = U - R^{-1}\bar{U}$ and $V^\star = V - R^{-1}\bar{V}$. Then,

assuming that the consistency check passes,

$$\begin{aligned} W &= [U\ \bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) + V \\ &= \left[(U^\star + R^{-1}\bar{U})\ \ \bar{C}\right]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) + V^\star + R^{-1}\bar{V} \\ &= U^\star G_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) + V^\star + R^{-1}\left(\left[\bar{U}\ \ R\bar{C}\right]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) + \bar{V}\right) \\ &= U^\star G_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) + V^\star. \end{aligned}$$

3. Let $\mathcal{W}_{\mathrm{pre}} = \mathbb{F}_q^{m\times k_{\mathcal{C}}}$, then run $\mathsf{Precom}(\bar{C}, R, R^{-1})$ to get the pre-commitment functions $U^\star_{\mathrm{pre}}, V^\star_{\mathrm{pre}}, L_{\mathrm{pre}}$ as in the simulator, as well as $\overleftarrow{\Delta}_0 \in L'$ and $L'_0 = L' - \overleftarrow{\Delta}_0$. Also, find some $\overleftarrow{L}_{\mathrm{off}} \in -L'$ where $\bar{V} = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{L}_{\mathrm{off}})$. Replace the underlying guess $\overleftarrow{\Delta} \in L'$ and the consistency check $-\bar{V} = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta})$ with $\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}} \in L_{\mathrm{pre}}(\bar{U})$. When such an $\overleftarrow{L}_{\mathrm{off}}$ exists, we need to show that this is equivalent to the consistency check. L'_0 is the linear subspace obtained by shifting L' to go through the origin, so $\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}} \in L'_0$ if and only if $\overleftarrow{\Delta} \in L'$ because $\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}}$ is the difference of two elements of the affine subspace L'. When $\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}} \in L'_0$, we have that $\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}} \in L_{\mathrm{pre}}(\bar{U})$ is equivalent to $0 = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}})$, which equals $[\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) + \bar{V}$. The latter being zero is the consistency check.

   We must also show that if the consistency check would pass, then a solution $\overleftarrow{L}_{\mathrm{off}}$ must exist. Assume that there exists some $\overleftarrow{\Delta}_1 \in L'$ that would pass the consistency check, i.e. $-\bar{V} = [\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}_1)$. Then $-\overleftarrow{\Delta}_1 \in -L'$ is a valid solution for $\overleftarrow{L}_{\mathrm{off}}$.

4. Factor out the sampling of Δ, the computation of $W_{[h]\cdot} = U^\star{}_{[h]\cdot}\,\mathrm{diag}(\overleftarrow{\Delta}) + V^\star{}_{[h]\cdot}$, and the selective abort attack $\overleftarrow{\Delta} + \overleftarrow{L}_{\mathrm{off}} \in L_{\mathrm{pre}}(\bar{U})$ into the ideal functionality $\mathcal{F}^{p,q,\mathcal{C},h,\mathcal{L},M}_{\mathsf{VOLE\text{-}pre}}$. The ideal functionality also includes an abort if $U^\star \neq U^\star_{\mathrm{pre}}(\bar{U})$ or $V^\star \neq V^\star_{\mathrm{pre}}(\bar{U}, \overleftarrow{\Delta})$, and we must show that neither will occur. The former cannot occur because that is exactly how $U^\star$ is calculated. For the latter, when the consistency check passes we have

$$V^\star_{\mathrm{pre}}(\bar{U}, \overleftarrow{\Delta}) = V + R^{-1}[\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\,\mathrm{diag}(\overleftarrow{\Delta}) = V - R^{-1}\bar{V} = V^\star.$$

5. We are now almost at the ideal world. We just need to change $\mathcal{W}_{\mathrm{pre}}$ to be $\{\bar{U} \in \mathbb{F}_q^{m\times k_{\mathcal{C}}} \mid t \geq \|\,[\bar{U}\ \ R\bar{C}]T_{\mathcal{C}}\|_0\}$, as in the simulator, and show that $|\mathcal{W}_{\mathrm{pre}}| \leq M$. Changing $\mathcal{W}_{\mathrm{pre}}$ is only detectable if $\bar{U} \notin \mathcal{W}_{\mathrm{pre}}$ and the consistency check still passes. Then the adversary

must guess $\| [\bar{U} \;\; R\bar{C}]T_{\mathcal{C}}\|_0 \geq t+1$ entries of $\tilde{\Delta}$, which has negligible probability q^{-t-1}. We just need to choose t to be as large as possible while keeping M small.

Finding a $\bar{U}$ such that $[\bar{U} \;\; R\bar{C}]T_{\mathcal{C}} = \bar{U}G_{\mathcal{C}} + [0 \; R\bar{C}]T_{\mathcal{C}}$ has few nonzero columns is equivalent to a bounded distance decoding problem over $\mathbb{F}_{q^m}$. That is, interpreting each column as an element of $\mathbb{F}_{q^m}$, $\bar{U}G_{\mathcal{C}}$ must be a code word close to $-[0 \; R\bar{C}]T_{\mathcal{C}}$ in Hamming weight. The simplest choice would be to set t to be the decoding radius $\lfloor \frac{d_{\mathcal{C}}-1}{2} \rfloor$ of $\mathcal{C}$, guaranteeing that there is at most a single element of $\mathcal{W}_{\text{pre}}$. To get a tighter bound, we use the Cassuto–Bruck list decoding bound [CB04], which implies $M \leq n_{\mathcal{C}}(d_{\mathcal{C}} - t)$ when $t = \frac{d_{\mathcal{C}}}{1+\sqrt{1+\frac{d_{\mathcal{C}}}{n_{\mathcal{C}}}-\frac{1}{n_{\mathcal{C}}^2}}}$. □

Optimizations. There are a couple ways that the communication complexity of Fig. 4.9 can be improved. First, if the universal hash R contains a lot of entropy, a seed $s \in \{0,1\}^\lambda$ may be sent instead, so $R = \mathsf{PRG}(s)$. The only place the randomness of R was used was to upper bound the probability that $\text{rank}(R\bar{C}) < \text{rank}(\bar{C})$. $\bar{C}$ cannot depend on s, so if using a PRG changed this probability more than negligibly then there would be an attack against the PRG.

A second optimization is to hash $\tilde{V}$ with a local random oracle Hash before sending it, because all that's needed is an equality check. The simulator (in the malicious P_S case) could then extract $\tilde{V}$ from its hash, then continue as usual. Interestingly, for concrete security it would be fine even if Hash were just an arbitrary collision resistant hash. Looking at just $\bar{C}$ and $\tilde{U}$, the simulator can see which entries of $\tilde{\Delta}$ are being guessed, though not what the guesses are. By looping through a random subset of 2^σ possible guesses (and for the usual setting of $\sigma = 40$ this is quite feasible), $\mathcal{S}$ can find the preimage of $\mathsf{Hash}(\tilde{V})$ often enough to only give the distinguisher an additional advantage of $2^{-\sigma}$.

4.5 OT Extension

Now that we have constructed subspace VOLE, it is time to go back to our original goal: OT extension. Like previous OT extensions, we hash our correlated randomness in order to get random OTs. For malicious security, our protocol (Fig. 4.11) follows [CT21] in using a universal hash to avoid collisions between extended OTs, avoiding the need for a TCR hash. However, a TCR hash allows for better concrete security (at they expense of performance) by reducing $\tau_{\max}$, which is the maximum number of queries $H(\tilde{y}, \tau)$ on the same tweak τ.

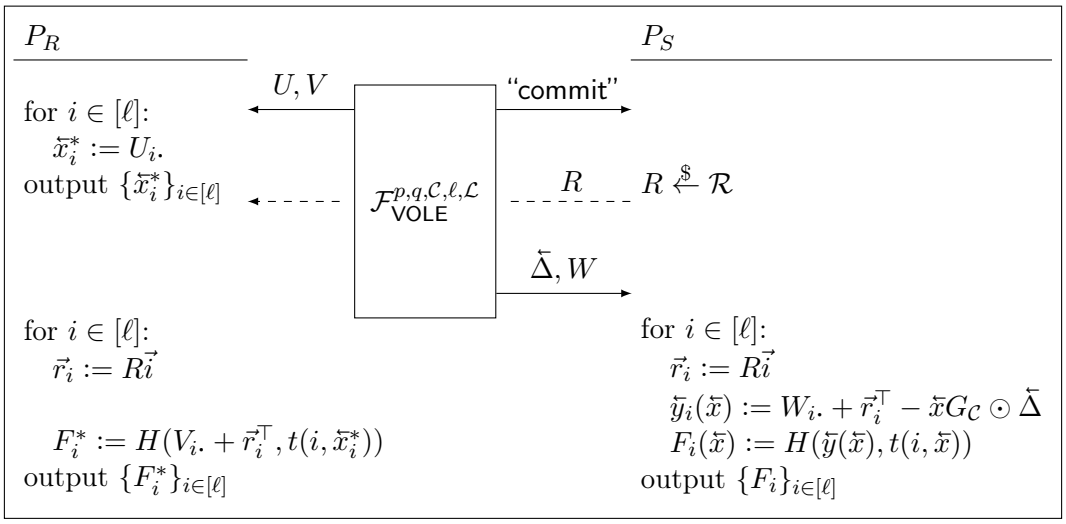


Figure 4.11: $\binom{p^{kc}}{1}$-OT extension protocol. Note that the parties for the base VOLE are swapped, with P_S (instead of P_R) getting $\vec{\Delta}$. If P_S receives "check failed" from the VOLE then the protocol is aborted immediately. For semi-honest security, the "commit" and R steps are skipped, and $\vec{r}_i := 0$.

We allow an arbitrary function $t(i, \vec{x})$ to control how many different hashes use the same tweak. Unlike [CT21], our analysis allows R to be sent in parallel with the VOLE protocol, saving a round of communication.

For generality, we allow any finite field, but we expect that $p = 2$ will be most efficient in almost all cases. We equivocate between the choices $U_i.$ in $\mathbb{F}_p^{kc}$ from the VOLE, and the choices x_i^* in $\left[p^{kc}\right]$ expected for OT. This can be thought of as writing x_i^* in base p.

Theorem 4.5.1. *The protocol in Fig. 4.11 achieves $\mathcal{F}_{OT\text{-}1}^{p^{kc},\ell,\{X\}}$ with malicious security in the $\mathcal{F}_{VOLE\text{-}pre}^{p,q,\mathcal{C},\ell,\mathcal{L},M}$ hybrid model, assuming that $H\colon \mathbb{F}_q^{nc} \times \mathcal{T} \to \{0,1\}^\lambda$ is a $(p, q, \mathcal{C}, \mathcal{T}, \mathcal{L})$-TCR hash, and $\mathcal{R} \subseteq \mathbb{F}_q^{nc \times \lceil \log_q(\ell) \rceil}$ is an ϵ-almost uniform family. The distinguisher advantage is at most $\epsilon M\ell(t_{max} - 1)/2 + \mathsf{Adv}_{\mathrm{TCR}}$, where t_{max} is the maximum number of distict OTs that can have the same tweak under t. For the TCR itself, τ_{max} will be the maximum number of evaluations $F_i(\vec{x})$ where $t(i, \vec{x})$ outputs a given tweak. For semi-honest security, $\mathcal{R}$ is unused; instead set $\epsilon = q^{-nc}$ and $M = 1$.*

Proof. See Appendix C.6.2. □

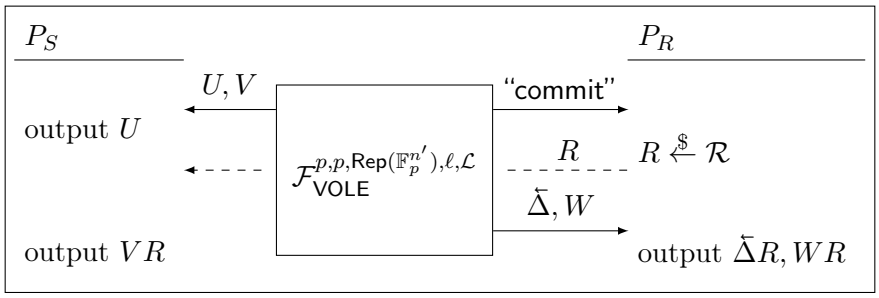


Figure 4.12: Non-leaky maliciously secure Δ-OT, a.k.a. subspace VOLE for the repetition code. If P_R receives "check failed" from the VOLE then the protocol is aborted immediately.

The Importance of Universal Hashing. For malicious security, it is *critical* that tweaking or some other mechanism is used to stop collisions in the input to H. This was noted by [GKW+20b; MR19], who show that when a malicious receiver can control its own randomness V (as we assume), they can force all H evaluations to be equal between two different extended OTs, causing two different OTs to have the same messages. However, this depends on controlling the seeds used for the underlying IKNP OT extension. Endemic OT uses this loophole, giving a protocol where P_S chooses its seeds, and claim to show that it is secure to omit the tweak in this case [MR19, Sect. 5.3]. In Appendix C.5, we give an attack against their protocol, using only the partial control over V that comes from the correction C. When the security parameter is $\lambda = 128$, it should have success probability a constant times 2^{-24} on a batch of 10^7 $\binom{2}{1}$-OTs.

4.5.1 Δ-OT

A common variant of OT extension is Δ-OT (a.k.a. correlated OT), where all OT messages follow the pattern $m_0, m_1 = m_0 \oplus \Delta$. It is useful for authenticated secret sharing and garbled circuits. More generally, over a larger field, it works as $m_x = m_0 + x\Delta$, and is useful for encoding the inputs to arithmetic garbling [BMR16].

Δ-OT works easily as a special case of subspace VOLE where $q = p$ and $\mathcal{C} = \mathsf{Rep}(\mathbb{F}_p^n)$,[6] except which party is called the sender and which the receiver is swapped, like with OT extension. However, in the malicious setting our subspace VOLE allows a selective abort

[6]Note that subspace VOLE with $q = p^k$ and $\mathcal{C} = \mathsf{Rep}(\mathbb{F}_p^n)$ can easily be turned into VOLE for $q = p$ and $\mathcal{C} = \mathsf{Rep}(\mathbb{F}_p^{kn})$, by interpreting $\mathbb{F}_q$ as a vector space over $\mathbb{F}_p$.

attack, and while for some applications (such as garbling) it may be allowed to leak a few bits for Δ, in others it may not. [BLN+15] solve this problem by multiplying the Δ-OT messages by a uniformly random rectangular matrix, throwing away some of the OT message. With high probability, any correlation among the bits of Δ is also lost, resulting in a non-leaky Δ-OT. In Fig. 4.12, we generalize this idea to use a universal hash, which can be more computationally efficient than a random matrix.

Theorem 4.5.2. *The protocol in Fig. 4.12 achieves* $\mathcal{F}_{\mathsf{VOLE}}^{p,p,\mathsf{Rep}(\mathbb{F}_p^n),\ell,\{X\}}$ *with malicious security in the* $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}^{p,p,\mathsf{Rep}(\mathbb{F}_p^{n'}),\ell,\mathrm{Affine}(\mathbb{F}_p^{n'}),M}$ *hybrid model, assuming that* $\mathcal{R} \subseteq \mathbb{F}_p^{n' \times n}$ *is a* ϵ*-almost uniform family and* $n' \geq n$*. The advantage is bounded by* $\epsilon M(p^n - 1)$*.*

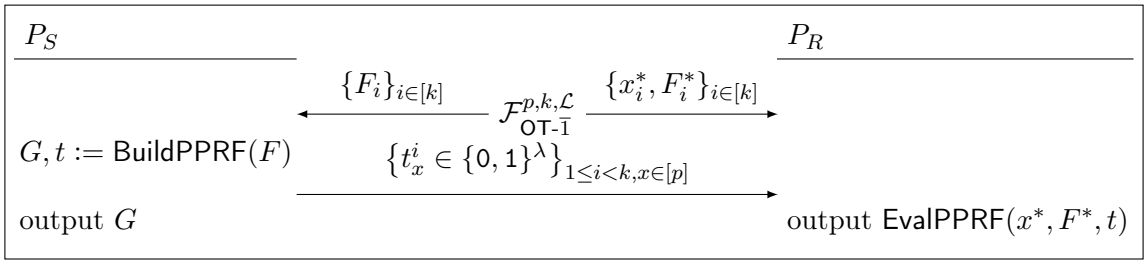
Proof. See Appendix C.6.3. □

Note that if $\mathcal{R}$ has the optimal $\epsilon = p^{-n'}$, such as when it is a uniformly random matrix, the environment's advantage is upper bounded by $Mp^{n-n'}$. Therefore, n' should be set to $n + \log_p(2)\sigma$ for security.

4.6 Base OTs

Our small field VOLE (Fig. 4.7) is based on $\binom{q}{q-1}$-OT, yet actual base OTs are generally $\binom{2}{1}$-OT. We follow [BGI17] in using a punctured PRF to efficiently construct $\binom{N}{N-1}$-OT. Our protocol (see Fig. 4.13) is based on the optimized version in [SGR+19], which generates $\binom{p^k}{p^k-1}$-OT from k $\binom{p}{p-1}$-OTs.

It depends on a $\mathsf{PRG}\colon \{0,1\}^\lambda \to (\{0,1\}^\lambda)^p$. The xth block of λ bits from this PRG is written as $\mathsf{PRG}_x(s)$. The PRG is used to create a GGM tree [GGM86]. Starting at the root of the tree, P_R gets $p-1$ of the p children from $\mathcal{F}_{\mathsf{OT}\text{-}\bar{1}}^{p,k,\mathrm{Affine}(\mathbb{F}_p^k)}$, and at every level down the tree the protocol maintains the property that P_R knows all but one of the nodes at that level. Each level i of the tree is numbered from 0 to $p^i - 1$, with the yth node in the layer containing the value s_y^i. This means that the children of node s_y^i are $s_{py+x}^{i+1} = \mathsf{PRG}_x(s_y^i)$, for $x \in [p]$. P_S computes the whole GGM tree in $\mathsf{BuildPPRF}$, finds the totals $t_x^i = \bigoplus_y s_{py+x}^{i+1}$ for each x, and uses the ith base OT to send all but one of these totals to P_R. Let y_i^* be the index of the node on the active path in layer i, i.e., the layer i node that P_R cannot learn. Then P_R will know every s_y^i except for $s_{y_i^*}^i$, so it can compute $s_{py_i^*+x}^{i+1} = t_x^i \oplus \bigoplus_{y \neq y_i^*} s_{py+x}^{i+1}$. Thus, it learns s_y^{i+1} for all $y \neq y_{i+1}^*$. At the end, this process gives $p^k - 1$ of the p^k leaf nodes



$\underline{\mathsf{BuildPPRF}(F)}$:
- for $x \in [p]$:
  - $s_x^1 := F_0(x)$
- for $i := 1$ to $k-1$:
  - for $y \in [p^i]$, $x \in [p]$:
    - $s_{py+x}^{i+1} := \mathsf{PRG}_x(s_y^i)$
  - for $x \in [p]$:
    - $t_x^i := F_i(x) \oplus \bigoplus_{y\in[p^i]} s_{py+x}^{i+1}$
- return $(y \mapsto s_y^k), t$

$\underline{\mathsf{EvalPPRF}(x^*, F^*, t)}$:
- for $x \in [p] \setminus \{x_0^*\}$:
  - $s_x^{*1} := F_0^*(x)$
- $y_1^* := x_0^*$
- for $i := 1$ to $k-1$:
  - for $y \in [p^i] \setminus \{y_i^*\}$, $x \in [p]$:
    - $s_{py+x}^{*i+1} := \mathsf{PRG}_x(s_y^{*i})$
  - for $x \in [p] \setminus \{x_i^*\}$:
    - $s_{py_i^*+x}^{*i+1} := t_x^i \oplus F_i^*(x) \oplus \bigoplus_{y\in[p^i]\setminus\{y_i^*\}} s_{py+x}^{*i+1}$
  - $y_{i+1}^* := py_i^* + x_i^*$
- return $y^*, (y \mapsto s_y^{*k-1})$

Figure 4.13: Protocol for $\binom{q}{q-1}$-OT based on $\binom{p}{p-1}$-OT, using a punctured PRF.

s_y^k.

Theorem 4.6.1. *Figure 4.13 constructs $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{q,1,\{X\}}$ out of $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{p,k,\{X\}}$, and is secure in the semi-honest model.*

Proof. See Appendix C.6.4. □

While the protocol only does a single $\binom{q}{q-1}$-OT from a batch of k $\binom{p}{p-1}$-OTs, it should be clear that a batch of n $\binom{q}{q-1}$-OT can be constructed from a batch of nk $\binom{p}{p-1}$-OTs. For $p = 2$, the base $\binom{p}{p-1}$-OTs are just $\binom{2}{1}$-OTs. For $p > 2$, they can be constructed from chosen message $\binom{p}{1}$-OT, by sending just the messages P_R is supposed to see.

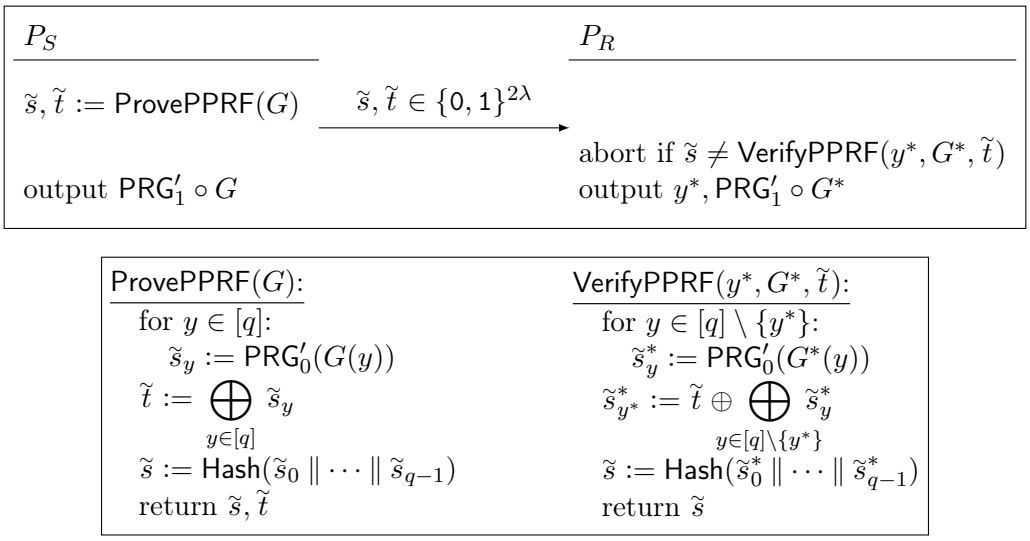


Figure 4.14: Consistency checking protocol for $\binom{q}{q-1}$-OT. This makes Fig. 4.13 maliciously secure.

4.6.1 Consistency Checking

In Fig. 4.14, we present the consistency check from the maliciously secure $\binom{2^k}{2^k-1}$-OT of [BCG+19a]. We prove a stronger property of their check, that P_S can only check guesses for the x_i^*s individually, not all of them together, which shows that any possible selective abort attack is in $\mathrm{Affine}(\mathbb{F}_p^k)$. This assumes that that PRG is collision resistant for its whole output, so there are no $s \neq s'$ such that $\mathsf{PRG}_x(s) = \mathsf{PRG}_x(s')$ for all $x \in [p]$. As in [BCG+19a], the protocol needs a second PRG, $\mathsf{PRG}' \colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda} \times \{0,1\}^\lambda$, which must be collision resistant in its first output PRG'_0. In the consistency check, P_S sends the total of all $\tilde{s}^k_y = \mathsf{PRG}'_0(s^k_y)$ so that P_R can reconstruct $\tilde{s}^k_{y^*}$. P_R then evaluates a collision resistant hash of all the $\tilde{s}^k_y$ and checks that it matches the hash from P_S. This commits P_S to a single possibility for each $\tilde{s}^k_y$.

In Appendix C.6.5 we use these assumptions to prove the following.

Proposition 4.6.2. *The selective abort attack allowed in Fig. 4.14 will always be in* $\mathcal{L} = \mathrm{Affine}(\mathbb{F}_p^k)$.

Theorem 4.6.3. *Figure 4.14 (composed with Fig. 4.13) is a maliciously secure* $\mathcal{F}_{\mathsf{OT}\text{-}\bar{1}}^{q,1,\mathrm{Affine}(\mathbb{F}_p^k)}$ *in the* $\mathcal{F}_{\mathsf{OT}\text{-}\bar{1}}^{p,k,\mathrm{Affine}(\mathbb{F}_p^k)}$ *hybrid model.*

4.7 Implementation

We implemented[7] our $\binom{2}{1}$-OT semi-honest and malicious protocols in the libOTe library [Rin], so that we could assess efficiency and parameter choices. We focus only on the case of binary fields ($p = 2$), as for this problem there is little benefit to using a larger p. First, we discuss the choices we made in instantiation.

For semi-honest security, our protocol depends on only a PRG and a TCR hash. We instantiate the TCR hash using Theorem 4.2.7, with AES as the ideal cipher. To keep $\tau_{\max}$ low, we set $t(i, \vec{x}) = \lfloor i/1024 \rfloor$, changing the tweak every 1024 OTs. We also used the hash as a PRG, evaluating it as $H(s, t(0)), H(s \oplus 1, t(1)), \ldots$ for a seed s. This allows the same AES round keys to be used across the many different PRG seeds used by OT extension, while AES-CTR would need to store many sets of round keys — too many to fit in L1 cache.

Malicious security additionally requires a universal hash for Fig. 4.9. As recommended in Section 4.4, we construct the universal hash in two stages. First, take each block of 64 bits from $\vec{x}$ and interpret it as an element of $\mathbb{F}_{2^{64}}$. These blocks become the coefficients of a polynomial over $\mathbb{F}_{2^{64}}$, which is evaluated at a random point to get $R\vec{x}$. We choose the constant term to always be zero, which makes this a uniform family (not just universal), allowing the use of Theorem 4.2.4 to sum multiple hashes together. Limiting each hash to 2^{20} blocks (each 64-bits long) before switching to the next (generated from a PRG seed) makes this a 2^{-44}-almost uniform family over $\mathbb{F}_2$. The second stage R' of the universal hash is over $\mathbb{F}_{2^k}$. It further compresses the output in $\mathbb{F}_{2^k}^{64}$ down to only $\mathbb{F}_{2^k}^{\lceil 40/k \rceil}$. We made the simple choice of a uniformly random matrix in $\mathbb{F}_{2^k}^{\lceil 40/k \rceil \times 64}$, which achieves the optimal $\epsilon = 2^{-k\lceil \frac{40}{k} \rceil}$ for a uniform family of this size.

Fig. 4.11 also needs a uniform hash, and we use multiplication over $\mathbb{F}_{2^{128}}$, multiplying each tweak by a 128-bit hash key to get a 128-bit hash. Guessing the hash would require guessing this hash key, so it is a 2^{-128}-almost uniform family.

The punctured PRF (Fig. 4.14) requires collision resistant primitives PRG, PRG', and Hash. For PRG, we assume that it is hard to find $s \neq s'$ such that $H(s, 0) = H(s', 0)$ and $H(s, 1) = H(s', 1)$, which is true in the ideal cipher model. We use Blake2 [ANW+13] for PRG'[8] and Hash.

[7] Source code is at https://github.com/ldr709/softspoken-implementation.

[8] H would also work, assuming that PRG'_0 concatenates two output blocks from H.

Protocol	Semi-honest Security					Malicious Security		
	Communication		Time (ms)			Time (ms)		
	KB	bits/OT	localhost	LAN	WAN	localhost	LAN	WAN
IKNP [IKN+03] / KOS [KOS15]	160010	128	391	1725	15525	443	1802	15662
SoftSpoken ($k = 1$)	160009	128	243	1590	15420	<u>298</u>	1637	15648
SoftSpoken ($k = 2$)	80009	64	**210**	815	7730	**255**	893	7985
SoftSpoken ($k = 3$)	53759	43	<u>223</u>	568	5208	322	677	5419
SoftSpoken ($k = 4$)	40008	32	261	<u>433</u>	3995	311	<u>530</u>	4114
SoftSpoken ($k = 5$)	32510	26	337	**348**	3271	454	**465**	3447
SoftSpoken ($k = 6$)	27509	22	471	488	2811	588	613	2985
SoftSpoken ($k = 7$)	23760	19	777	843	2380	899	966	<u>2554</u>
SoftSpoken ($k = 8$)	20008	16	1259	1314	<u>1916</u>	1293	1322	**2130**
SoftSpoken ($k = 9$)	18759	15	2302	2338	2439	2460	2457	2590
SoftSpoken ($k = 10$)	16259	13	3984	3983	4097	4126	4132	4223
Ferret [YWL+20]	2976	2.38	2156	2160	2825	2240	2242	3108
Silent (Quasi-cyclic) [BCG+19a]	**127**	**0.10**	7735	7736	8049			
Silent (Silver, weight 5) [CRR21]	<u>127</u>	<u>0.10</u>	613	613	**746**			

Table 4.1: Time and communication required to generate 10^7 OTs, averaged over 50 runs. The best entry in each column is **bolded**, and the second best is <u>underlined</u>. Communication costs for maliciously secure versions are within 10 KB of the semi-honest ones. The setup costs are included.

Semi-honest Security

Protocol	Comm.	Time (ms)					
		localhost		LAN		WAN	
	KB	P_R	P_S	P_R	P_S	P_R	P_S
IKNP [IKN+03]	4.2	27	19	32	21	94	54
SoftSpoken (k in 1–10)	8.3–9.8	27–29	28–30	32–44	33–45	86–101	127–142
Silent (Quasi-cyclic) [BCG+19a]	53.4	31	33	32	34	102	146
Silent (Silver, weight 5) [CRR21]	53.4	28	30	33	35	102	147
Ferret [YWL+20]	1166.8	65	65	70	65	552	342

Malicious Security

KOS [KOS15]	4.2	28	28	33	32	105	145
SoftSpoken (k in 1–10)	9.3–16.8	27–33	28–34	32–38	32–38	100–109	141–151
Ferret [YWL+20]	1175.3	73	73	75	73	608	553

Table 4.2: One-time setup costs for OT protocols in Table 4.1. SoftSpokenOT protocols have nearly identical setup costs, and so only a range is given.

4.7.1 Performance Comparison

In Tables 4.1 and 4.2, we present benchmarks of our implementation in both the semi-honest and malicious settings, for a variety of communication settings and parameter choices. We also compare to existing OT extensions. All results were measured on an Intel i7-7500U laptop CPU, with the sender and receiver each running on a single thread. The software was

compiled with GCC 11.1 with `-O3` and link-time optimizations enabled, and executed on Linux. In the localhost setting, there is no artificial limit on the communication between these threads, though the kernel has overhead in transferring the data, which is why our $k = 2$ is faster than $k = 1$ even in this case. We simulated communicating over a LAN by applying a latency of 1 ms and a 1 Gbps bandwidth limit. For the WAN setting, this becomes 40 ms and 100 Mbps. Base OTs were generated using POPF OT (Fig. 3.3), instantiated with the EKE POPF (Section 3.5.1).[9] The choice bits of SoftSpokenOT were derandomized immediately, as were the choice bits for Ferret, to provide the most direct comparison with IKNP and KOS. The choice bits for the Silent OTs were not derandomized, slightly biasing the comparison in their favor.

Although for $k = 1$ our protocol is the same as IKNP in the semi-honest setting, our implementation is significantly faster. This mainly comes from a new implementation of 128×128 bit transposition, based on using AVX2 to implement Eklundh's algorithm [TE76]. This gave a 6× speedup for bit transposition, which is a significant factor of IKNP's overall runtime.

In our benchmark, Silver did not perform as well as IKNP in the localhost setting, while [CRR21] found that Silver was nearly 60% faster than IKNP. We attribute this difference to using a lower quality computer, which has less memory bandwidth than the machine used for their benchmark. This is important for Silver's transposed encoding, a memory intensive operation. Compared to Silent OT, we achieve better concrete performance in the localhost and LAN settings, but the extremely low communication of Silent OT puts Silver in first place for the WAN setting. We claim another a benefit to our protocol over Silver, since SoftSpokenOT only needs fairly conservative assumptions about well-studied objects like block ciphers, while Silver depends on hardness of LPN for a novel family of codes that has yet to receive much cryptanalysis. More conservative versions of Silent OT, based on either quasi-cyclic codes [BCG+19a] or local linear codes [YWL+20], are slower than SoftSpokenOT across the tested settings.

For malicious security, we use a more efficient universal hash function compared to KOS[10], who require the additional generation of 128 bits from a PRG for every OT as part of the consistency check. We have not benchmarked maliciously secure implementations of Silent

[9] Silent OT needs more than λ base OTs, and so as an optimization it generates them using KOS, which needs only λ base OTs.

[10] The implementation of KOS in libOTe has the CR hashing flaw discussed in Section 4.5.

OT and Silver, but they likely have very similar performance to the semi-honest case.

Bibliography

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001. DOI: 10.1007/3-540-45353-9_12.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012. DOI: 10.1007/978-3-642-29011-4_29.

[ANW+13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 119–135. Springer, Heidelberg, June 2013. DOI: 10.1007/978-3-642-38980-1_8.

[BCG+18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018. DOI: 10.1145/3243734.3243868.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019. DOI: 10.1145/3319535.3354255.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019,*

Part III, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019. DOI: 10.1007/978-3-030-26954-8_16.

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010. DOI: 10.1007/978-3-642-14623-7_13.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992. DOI: 10.1007/3-540-46766-1_34.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996. DOI: 10.1145/237814.237996.

[Ber06] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006. DOI: 10.1007/11745853_14.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Heidelberg, 2017. DOI: 10.1007/978-3-319-56614-6_6.

[BHK+13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013. DOI: 10.1145/2508859.2516734.

[BHK+99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 216–233. Springer, Heidelberg, August 1999. DOI: 10.1007/3-540-48405-1_14.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012. DOI: 10.1145/2382196.2382279.

[BLN+15] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. https://eprint.iacr.org/2015/472.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992. DOI: 10.1109/RISP.1992.213269.

[BMN01] Colin Boyd, Paul Montague, and Khanh Quoc Nguyen. Elliptic curve based password authenticated key exchange protocols. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 487–501. Springer, Heidelberg, July 2001. DOI: 10.1007/3-540-47719-5_38.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016. DOI: 10.1145/2976749.2978410.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990. DOI: 10.1145/100216.100287.

[BÖS11] Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient hashing using the AES instruction set. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 507–522. Springer, Heidelberg, 2011. DOI: 10.1007/978-3-642-23951-9_33.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EU-*

ROCRYPT 2006, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, 2006. DOI: 10.1007/11761679_25.

[CB04] Yuval Cassuto and Jehoshua Bruck. A Combinatorial Bound on the List Size. Technical report, California Institute of Technology, 2004.

[CCG18] Ignacio Cascudo, René Bødker Christensen, and Jaron Skovsted Gundersen. Actively secure OT-extension from q-ary linear codes. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 333–348. Springer, Heidelberg, September 2018. DOI: 10.1007/978-3-319-98113-0_18.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015. DOI: 10.1007/978-3-662-48000-7_1.

[CDD+16] Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, Heidelberg, August 2016. DOI: 10.1007/978-3-662-53015-3_7.

[CFG+06] Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. The Twist-AUgmented technique for key exchange. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 410–426. Springer, Heidelberg, April 2006. DOI: 10.1007/11745853_27.

[CKK+12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012. DOI: 10.1007/978-3-642-28914-9_3.

[CMR] Brent Carmer, Alex J. Malozemoff, and Marc Rosen. swanky: a suite of rust libraries for secure multi-party computation. https://github.com/GaloisInc/swanky.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, Heidelberg, August 2015. DOI: 10.1007/978-3-319-22174-8_3.

[CR16] Brent Carmer and Mike Rosulek. Linicrypt: A model for practical cryptography. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 416–445. Springer, Heidelberg, August 2016. DOI: 10.1007/978-3-662-53015-3_15.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event. Springer, Heidelberg, August 2021. DOI: 10.1007/978-3-030-84252-9_17.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Blazing fast OT for three-round UC OT extension. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 299–327. Springer, Heidelberg, May 2020. DOI: 10.1007/978-3-030-45388-6_11.

[CT21] Yu Long Chen and Stefano Tessaro. Better security-efficiency trade-offs in permutation-based two-party computation. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, pages 275–304, 2021. ISBN: 978-3-030-92075-3.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[Den20] Frank Denis. The sodium cryptography library, 2020. URL: https://download.libsodium.org/doc/.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: rijndael, 1999.

[DSS+17] Yuanxi Dai, Yannick Seurin, John P. Steinberger, and Aishwarya Thiruvengadam. Indifferentiability of iterated Even-Mansour ciphers with non-idealized key-schedules: five rounds are necessary and sufficient. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*,

pages 524–555. Springer, Heidelberg, August 2017. DOI: 10.1007/978-3-319-63697-9_18.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018. ISSN: 2474-1558. DOI: 10.1561/3300000019. https://securecomputation.org/.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, November 1993. DOI: 10.1007/3-540-57332-1_17.

[FFS+10] Reza R. Farashahi, Pierre-Alain Fouque, Igor E. Shparlinski, Mehdi Tibouchi, and J. Felipe Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. Cryptology ePrint Archive, Report 2010/539, 2010. https://eprint.iacr.org/2010/539.

[FLP08] Jean-Charles Faugère, Françoise Levy-dit-Vehel, and Ludovic Perret. Cryptanalysis of minrank. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 280–296. Springer, Heidelberg, August 2008. DOI: 10.1007/978-3-540-85174-5_16.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, Heidelberg, April 2015. DOI: 10.1007/978-3-662-46803-6_7.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GKW+20a] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. Better concrete security for half-gates garbling (in the multi-instance setting). In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 793–822. Springer, Heidelberg, August 2020. DOI: 10.1007/978-3-030-56880-1_28.

[GKW+20b] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841. IEEE Computer Society Press, May 2020. DOI: 10.1109/SP40000.2020.00016.

[GLN+15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 567–578. ACM Press, October 2015. DOI: 10.1145/2810103.2813619.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004. ISBN: ISBN 0-521-83084-2 (hardback).

[HL17] Eduard Hauck and Julian Loss. Efficient and universally composable protocols for oblivious transfer from the CDH assumption. Cryptology ePrint Archive, Report 2017/1011, 2017. https://eprint.iacr.org/2017/1011.

[IKN+03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003. DOI: 10.1007/978-3-540-45146-4_9.

[Imp95] R. Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, 1995. DOI: 10.1109/SCT.1995.514853.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 8–26. Springer, Heidelberg, August 1990. DOI: 10.1007/0-387-34799-2_2.

[Kal91] Burton S. Kaliski Jr. One-way permutations on elliptic curves. *Journal of Cryptology*, 3(3):187–199, January 1991. DOI: 10.1007/BF00196911.

[Kel20] Marcel Keller. MP-SPDZ: a versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. https://eprint.iacr.org/2020/521.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013. DOI: 10.1007/978-3-642-40084-1_4.

[KKS16] Carmen Kempka, Ryo Kikuchi, and Koutarou Suzuki. How to circumvent the two-ciphertext lower bound for linear garbling schemes. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 967–997. Springer, Heidelberg, December 2016. DOI: 10.1007/978-3-662-53890-6_32.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014. DOI: 10.1007/978-3-662-44381-1_25.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015. DOI: 10.1007/978-3-662-47989-6_35.

[KOS21] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. Unpublished draft of full version, 2021.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008. DOI: 10.1007/978-3-540-70583-3_40.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009. DOI: 10.1007/s00145-008-9036-8.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228. Springer, Heidelberg, April 2015. DOI: 10.1007/978-3-662-46800-5_9.

[Möl04] Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS 2004*, volume 3193 of *LNCS*, pages 335–351. Springer, Heidelberg, September 2004. DOI: 10.1007/978-3-540-30108-0_21.

[MR18] Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 35–52. ACM Press, October 2018. DOI: 10.1145/3243734.3243760.

[MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, November 2019. DOI: 10.1145/3319535.3354210.

[MRR20] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 425–442. ACM Press, November 2020. DOI: 10.1145/3372297.3417870.

[MRR21] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Batching base oblivious transfers. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, pages 281–310, Cham. Springer International Publishing, 2021. ISBN: 978-3-030-92078-4. URL: https://eprint.iacr.org/2021/682.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, Denver, Colorado, USA. ACM, 1999. ISBN: 1-58113-176-3. DOI: 10.1145/336992.337028.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 381–396. Springer, Heidelberg, February 2017. DOI: 10.1007/978-3-319-52153-4_22.

[PRT+20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020. DOI: 10.1007/978-3-030-45724-2_25.

[PSS17] Arpita Patra, Pratik Sarkar, and Ajith Suresh. Fast actively secure OT extension for short secrets. In *NDSS 2017*. The Internet Society, 2017.

[PSS+09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009. DOI: 10.1007/978-3-642-10366-7_15.

[Rin] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[Ros17] Mike Rosulek. Improvements for gate-hiding garbled circuits. In Arpita Patra and Nigel P. Smart, editors, *INDOCRYPT 2017*, volume 10698 of *LNCS*, pages 325–345. Springer, Heidelberg, December 2017.

[Roy22] Lawrence Roy. SoftSpokenOT: communication–computation tradeoffs in OT extension. In *CRYPTO 2022*, LNCS. Springer, Heidelberg, August 2022. URL: https://eprint.iacr.org/2022/192. To be published.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event. Springer, Heidelberg, August 2021. DOI: 10.1007/978-3-030-84242-0_5.

[SGR+19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019. DOI: 10.1145/3319535.3363228.

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[Sma] Nigel Smart. SCALE-MAMBA: secure computation algorithms from LEuven, multiparty algorithms basic argot. https://homes.esat.kuleuven.be/~nsmart/SCALE/.

[TE76] R. E. Twogood and M. P. Ekstrom. An extension of Eklundh's matrix transposition algorithm and its application in digital image processing. *IEEE Transactions on Computers*, C-25(9):950–952, 1976. DOI: 10.1109/TC.1976.1674721.

[TK17] Mehdi Tibouchi and Taechan Kim. Improved elliptic curve hashing and point representation. Designs, Codes and Cryptography, 2017.

[WM17] Yongge Wang and Qutaibah m. Malluhi. Reducing garbled circuit size while preserving circuit gate privacy. Cryptology ePrint Archive, Report 2017/041, 2017. https://eprint.iacr.org/2017/041.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. DOI: 10.1109/SFCS.1986.25.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020. DOI: 10.1145/3372297.3417276.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015. DOI: 10.1007/978-3-662-46803-6_8.

APPENDICES

Appendix A: 3/2 Garbling

A.1 Randomized Tweakable Circular Correlation Robust Functions

A.1.1 Circular Correlation Robustness

We build to our final construction in two steps, the first of which is to construct a simpler circular correlation robust hash in the ideal permutation model.

Below we give the definition of circular correlation robustness from [GKW+20b], but generalized to support non-matching input/output lengths and a family of linear functions $\mathcal{L}$. The original definition corresponds to the case $n = m$ and $\mathcal{L}$ containing the identity function and the all-zeroes function.

Definition A.1.1. *Let $H : \{0,1\}^n \to \{0,1\}^m$, and let $\mathcal{L}$ be a set of linear functions from $\{0,1\}^n \to \{0,1\}^m$. Then H is **circular correlation robust for $\mathcal{L}$** if for all PPT $\mathcal{A}$,*

$$\left| \Pr_{\Delta}\left[\mathcal{A}^{\mathcal{O}_{H,\Delta}}() = 1\right] - \Pr_{R}\left[\mathcal{A}^{R}() = 1\right] \right|$$

is negligible, where $\mathcal{O}^{\mathsf{ccr}}_{H,\Delta}$ is defined as:

$$\boxed{\begin{array}{l}\underline{\mathcal{O}_{H,\Delta}(X \in \{0,1\}^n, L \in \mathcal{L}):} \\ \quad \text{return } H(X \oplus \Delta) \oplus L(\Delta)\end{array}}$$

We show how to construct such a function for $n = m$ in the ideal permutation model. Our constructions require $m \leq n$, and such a function can be obtained by simply truncating one with $n = m$.

Lemma A.1.2. *Fix a set of linear transformations $\mathcal{L}$, and let σ be any function such that:*

- *σ is linear, so that $\sigma(X \oplus Y) = \sigma(X) \oplus \sigma(Y)$*
- *$X \mapsto L(X) \oplus \sigma(X)$ is invertible for all $L \in \mathcal{L}$.*

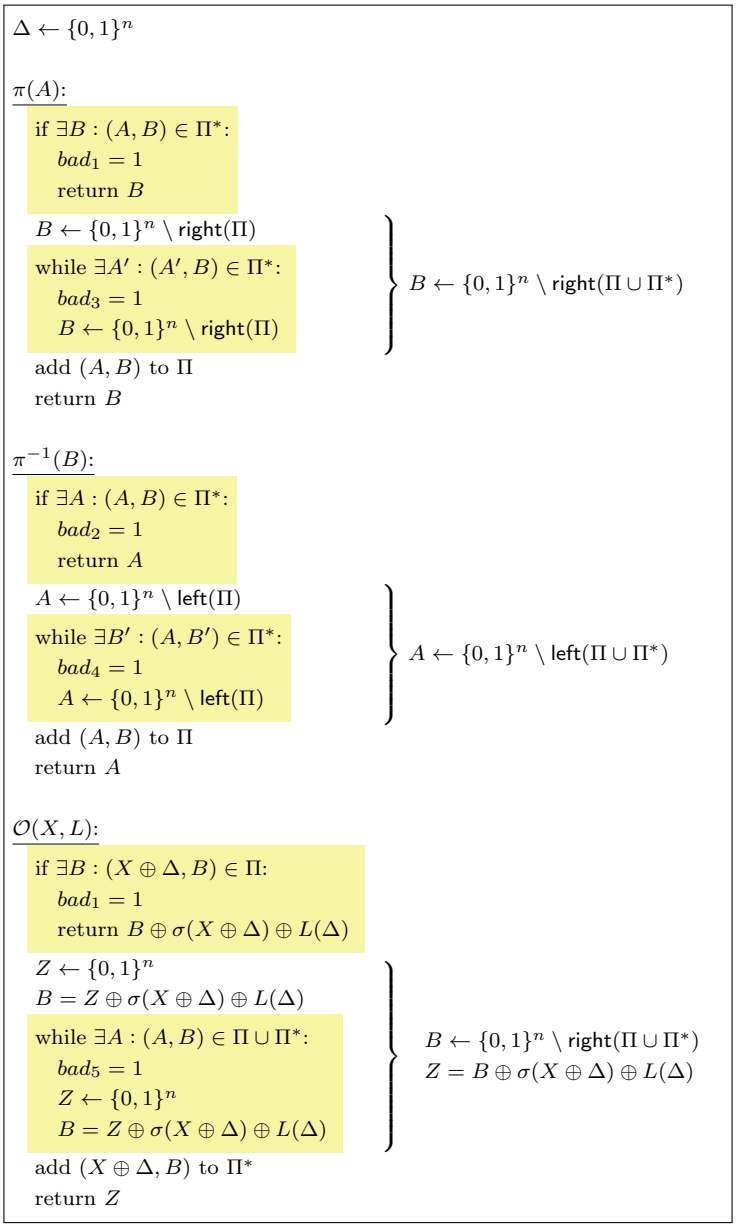
If π is an ideal permutation (all parties have oracle access to a random permutation π and its inverse π^{-1}), then

$$H(X) = \pi(X) \oplus \sigma(X)$$

is circular correlation robust for $\mathcal{L}$.

This construction is the natural generalization of the one from [GKW+20b], who consider $\mathcal{L}$ to contain only the zero-function and the identity function. In that case, our restrictions on σ amount to requiring that σ is an *orthomorphism.*

Proof. Consider the following game, in the style of Bellare & Rogaway [BR06]:



The adversary gets oracle access to these 3 oracles and finally outputs a bit. Without loss of generality, assume that the adversary does not repeat identical queries, does not query π on a previous output of π^{-1}, and does not query π^{-1} on a previous output of π — in all of these cases, the answer to the query is already known. We make the following observations:

- **Without the highlighted lines, the game matches the "ideal" CCR experiment.** The $\pi^{\pm}$ oracles instantiate an ideal permutation on the fly, in the usual way, keeping track of the input/output pairs in the set Π. $\mathsf{left}(\Pi)$ and $\mathsf{right}(\Pi)$ denote the left/right element of all tuples in Π, respectively. $\mathcal{O}$ is instantiated as an independent random function.

- **With the highlighted lines, the game matches the "real" CCR experiment.** A query of the form $\mathcal{O}(X, L) = Z$ corresponds precisely to an internal query of the form $\pi(X \oplus \Delta) = Z \oplus \sigma(X \oplus \Delta) \oplus L(\Delta)$. This game maintains a set Π^* of input/output pairs for $\pi^{\pm}$ that are defined as a result of a query to $\mathcal{O}$. The added if-statements ensure consistency when the same π input/output pair is used in both a query to $\pi^{\pm}$ and to $\mathcal{O}$. The added while-statements ensure that π always remains a permutation. Note that the added while-statements, along with the preceding lines, can be simplified as we show to the right.

- **The two games are identical-until-bad.** With or without the highlighted lines, the games execute identical statements until one of the bad_i flags is set to 1.

From [BR06], we have that the distinguishing advantage of the adversary is bounded by Pr[any bad_i is set to 1]. This probability can be calculated with respect to the ideal game (highlighted code is not executed, except to set the bad_i flags). Since the bad_i flags do not affect the execution in the ideal game, it is most convenient to consider that the adversary makes *all* queries, and only when the game is over do we inspect the execution and determine whether the bad_i flags are set. It suffices to show that each bad_i flag is set to 1 with only negligible probability.

- bad_1 is set to 1 (in either of two places) only if the adversary manages to directly query $\pi(A)$ and also $\mathcal{O}(X, L)$ where $X = A \oplus \Delta$ — the two queries can happen in either order. Note that in the ideal game variant, Δ is independent of the adversary's view. It is equivalent to choose Δ at the end of the execution, after all queries have been made, and when we are checking whether any bad_i flag was set. For a two specific queries (one to π and one to $\mathcal{O}$), the probability that they satisfy $X = A \oplus \Delta$ is $1/2^n$. Hence if the adversary makes a total of q oracle queries, bad_1 is set with probability at most $q^2/2^n$ by a simple union bound.

- bad_2 is set to 1 only if the adversary manages to query $\mathcal{O}(X, L) = Z$ and later query

$\pi^{-1}(B)$ where $Z = B \oplus \sigma(X \oplus \Delta) \oplus L(\Delta)$. As above, imagine Δ being chosen after all queries have been made. For a specific pair of queries (one to π^{-1} and one to $\mathcal{O}$), the probability that

$$\begin{aligned} Z &= B \oplus \sigma(X \oplus \Delta) \oplus L(\Delta) \\ &= B \oplus \sigma(X) \oplus \Big(\sigma(\Delta) \oplus L(\Delta)\Big) \end{aligned}$$

is $1/2^n$ since $\sigma \oplus L$ is a bijection. Hence if the adversary makes a total of q oracle queries, bad_2 is set with probability at most $q^2/2^n$ by a simple union bound.

- bad_3 is set to 1 only if B is chosen from $\mathsf{right}(\Pi^*)$. At any given time the size of $\mathsf{right}(\Pi^*)$ is bounded by q, the total number of queries made by the adversary. Hence the total probability that bad_3 gets set is bounded by $q^2/2^n$.
- bad_4 is similar to bad_3: it is set to 1 only if A is chosen from $\mathsf{left}(\Pi^*)$.
- bad_5 is set to 1 only if $B \in \mathsf{right}(\Pi \cup \Pi^*)$. Note that when Z is uniform, so is B. So as above, on each call to $\mathcal{O}$ the probability that bad_5 is set is at most $q/2^n$, and the overall probability of bad_5 being set is at most $q^2/2^n$.

Since each bad_i flag is set to 1 with negligible probability, the two games are indistinguishable, and the construction satisfies the CCR security definition. □

Instantiations. Our main construction truncates a CCR from λ to $\lambda/2$ bits. Let the input X to the CCR be λ bits and split it into two halves as $X = X_L \| X_R$. Our construction uses linear functions $L_{ab}(\Delta_L \| \Delta_R) = (a\Delta_L \oplus b\Delta_R) \| 0^{\lambda/2}$, for $a, b \in \{0, 1\}$.

Our optimization in Section 2.6.2 uses a single λ-bit CCR to derive two calls to a $\lambda/2$-bit CCR, *each with possibly different linear transformations.* This corresponds to a λ-bit CCR with linear functions $L_{abcd}(\Delta_L \| \Delta_R) = (a\Delta_L \oplus b\Delta_R) \| (c\Delta_L \oplus d\Delta_R)$, for $a, b, c, d \in \{0, 1\}$.

Our construction requires an XOR-homomorphic function σ such that $\sigma(X) \oplus L(X)$ is invertible for any L in this class. The simplest examples of such a σ is as follows: split the input X into two halves $X_L \| X_R$, then define $\sigma(X_L \| X_R) = (\alpha X_L) \| (\alpha X_R)$, where α is any fixed element in $\mathbb{F}_{2^{\lambda/2}} \setminus \mathbb{F}_{2^2}$, and the multiplication is in $\mathbb{F}_{2^{\lambda/2}}$. Then we get

$$\sigma(\Delta) \oplus L_{abcd}(\Delta) = \begin{bmatrix} \alpha \oplus a & b \\ c & \alpha \oplus d \end{bmatrix} \begin{bmatrix} \Delta_L \\ \Delta_R \end{bmatrix}$$

Since $a, b, c, d \in \{0, 1\}$, the determinant of this matrix is a degree-2 polynomial in α with binary coefficients. And since α is chosen not to be in $\mathbb{F}_{2^2}$, it is not the root of any such degree-2 polynomial. Hence the determinant of the matrix is nonzero, and $\sigma(\Delta) \oplus L_{abcd}(\Delta)$ is invertible.

A.1.2 Randomized Tweakable CCR

Lemma A.1.3. *Define* $H^*_{k,U}$ *as:*

$$H^*_{k,U}(X, \tau) = F_k(X \oplus U(\tau))$$

and denote the family of all such function as $\mathcal{H}^* = \{H_{k,U} \mid k \in \{0,1\}^\lambda, U \in \mathcal{U}\}$*. If* F *is a secure PRF, and for every (fixed)* k*,* F_k *is CCR for* $\mathcal{L}$*, and* $\{(X, \tau) \mapsto X \oplus U(\tau) \mid U \in \mathcal{U}\}$ *is a universal hash family, then* $\mathcal{H}^*$ *is a secure RTCCR hash family for* $\mathcal{L}$*.*

Proof. Consider an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ in the "real" RTCCR experiment. Its first phase $\mathcal{A}_1$ makes queries to $H^* \in \mathcal{H}^*$ and to $\mathcal{O}^{\mathsf{rtccr}}_{H^*,\Delta}$ before learning the parameters k and U.

Claim: it is with only negligible probability that $\mathcal{A}_1$ makes distinct queries (X, τ), (X', τ') to its oracles such that $X \oplus U(\tau) = X' \oplus U(\tau')$.

Proof of claim: Consider the following reduction algorithm M which has oracle access to the construction $H^*_{k,U}$, with k and U uniform. It internally runs $\mathcal{A}_1$ and chooses a random Δ. When $\mathcal{A}_1$ queries its H^* oracle, M relays that query directly to its oracle. When $\mathcal{A}_1$ queries its $\mathcal{O}$ oracle on (X, τ, L), M queries its H^* oracle and returns $H^*_{k,U}(\Delta \oplus X, \tau) \oplus L(\Delta)$.

Clearly M perfectly simulates the view of $\mathcal{A}_1$. Since F_k is a PRF and $(X, \tau) \mapsto X \oplus U(\tau)$ is a universal hash function, the construction $H^*_{k,U}(X, \tau) = F_k(X \oplus U(\tau))$ is exactly a Carter-Wegman MAC/PRF [BHK+99]. The usual security proof of Carter-Wegman establishes that it is only with negligible probability that an adversary with oracle access to the MAC causes an internal collision in the universal hash.

Now consider the following reduction algorithm M' which has an oracle for just F_k and the (plain) CCR oracle $\mathcal{O}^{\mathsf{ccr}}_{F_k,\Delta}$. It internally runs $\mathcal{A}_1$ and chooses a random $U \leftarrow \mathcal{U}$. When $\mathcal{A}_1$ queries its H^* oracle at (X, τ), M' queries its F_k oracle at $X \oplus U(\tau)$. When $\mathcal{A}_1$ queries its $\mathcal{O}^{\mathsf{rtccr}}$ oracle at (X, τ, L), M' queries its $\mathcal{O}^{\mathsf{ccr}}_{F_k,\Delta}$ oracle at $(X \oplus U(\tau), L)$. Furthermore, M' aborts if two distinct queries ever result in the same $X \oplus U(\tau)$. After $\mathcal{A}_1$ finishes, M' runs

$\mathcal{A}_2$ and gives it k and U.

Note that

$$\begin{aligned}\mathcal{O}^{\mathsf{ccr}}_{F_k,\Delta}(X \oplus U(\tau), L) &= F_k(\Delta \oplus X \oplus U(\tau)) \oplus L(\Delta) \\ &= H^*_{k,U}(X \oplus \Delta, \tau) \oplus L(\Delta) \\ &= \mathcal{O}^{\mathsf{rtccr}}_{H^*,\Delta}(X, \tau, L)\end{aligned}$$

and so, conditioned on M' not aborting, it perfectly simulates the view of $(\mathcal{A}_1, \mathcal{A}_2)$. By the argument above, M' aborts with only negligible probability. Note that M' is a valid adversary in the (plain) CCR experiment as it never repeats an argument $X \oplus U(\tau)$ to its $\mathcal{O}^{\mathsf{ccr}}_{F_k,\Delta}$ oracle. It is important that the CCR experiment allows F_k to be public, so that M' can give k to $\mathcal{A}_2$.

By the fact that F_k is CCR (for fixed k), this interaction is indistinguishable from one in which $\mathcal{O}^{\mathsf{ccr}}_{F_k,\Delta}$ is replaced with a random function. Making such a replacement causes $\mathcal{A}_1$'s $\mathcal{O}^{\mathsf{rtccr}}$ oracle to be replaced by a random function. The fact that this view for $(\mathcal{A}_1, \mathcal{A}_2)$ is indistinguishable shows that $\mathcal{H}^*$ is RTCCR. □

Instantiation A simple choice of U is multiplication (in $\mathbb{F}_{2^\lambda}$) by a random field element. Consider the map $(X, \tau) \mapsto X \oplus U(\tau) = X \oplus u \cdot \tau$ where $u \leftarrow \mathbb{F}_{2^\lambda}$. For fixed $(X, \tau) \neq (X', \tau')$, we have the following cases:

- If $\tau = \tau'$ then $X \neq X'$ so $X \oplus u\tau \neq X' \oplus u\tau'$.
- If $\tau \neq \tau'$ then $\Pr[X \oplus u\tau = X' \oplus u\tau'] = \Pr[(X \oplus X')(\tau \oplus \tau')^{-1} = u] = 1/2^\lambda$ since u is uniform.

In either case, the probability of (X, τ) and (X', τ') being a collision is negligible, so the map is a universal hash.

In the reasonable event that all tweaks are at most $\lambda/2$ bits, we can interpret τ as an element of $\mathbb{F}_{2^{\lambda/2}}$ and define $U(\tau) = (u_L\tau)\|(u_R\tau)$ where u_L, u_R are independently uniform in $\mathbb{F}_{2^{\lambda/2}}$. For $\lambda = 128$, this way of defining U involves more efficient 64-bit operations.

A.2 Randomizing the Entire Control Matrix

We used computerized linear algebra to generate a basis for the 14-dimensional linear space of matrices $R_\$$ such that $KR_\$ = 0$.

$$R_\$ \leftarrow \text{span}\left\{
\left[\begin{array}{cc|cc|cc} 1&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&1&0&1&0\\1&0&0&0&0&0\\\hline 0&0&1&0&0&0\\1&0&1&0&1&0\\\hline 1&0&0&0&1&0\\0&0&1&0&1&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&1&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&1&0&1\\0&1&0&0&0&0\\\hline 0&0&0&1&0&0\\0&1&0&1&0&1\\\hline 0&1&0&0&0&1\\0&0&0&1&0&1\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&1&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\1&0&0&0&0&0\\\hline 0&0&1&0&0&0\\0&0&1&0&0&0\\\hline 0&0&0&0&0&0\\1&0&1&0&0&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&1&0&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\0&1&0&0&0&0\\\hline 0&0&0&1&0&0\\0&0&0&1&0&0\\\hline 0&0&0&0&0&0\\0&1&0&1&0&0\end{array}\right],
\right.
$$

$$
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\1&0&0&0&0&0\\\hline 0&0&1&0&1&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\1&0&1&0&1&0\\\hline 0&0&1&0&1&0\\0&0&1&0&1&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&1&0&0&0&0\\\hline 0&0&0&1&0&1\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\0&1&0&1&0&1\\\hline 0&0&0&1&0&1\\0&0&0&1&0&1\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&1&0&0&0\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\0&0&1&0&0&0\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&1&0&0\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\0&0&0&1&0&0\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\end{array}\right],
$$

$$
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 1&0&1&0&1&0\\1&0&0&0&0&0\\\hline 0&0&1&0&0&0\\1&0&1&0&1&0\\\hline 1&0&0&0&1&0\\0&0&1&0&1&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&1&0&1&0&1\\0&1&0&0&0&0\\\hline 0&0&0&1&0&0\\0&1&0&1&0&1\\\hline 0&1&0&0&0&1\\0&0&0&1&0&1\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&1&0&1&0\\0&0&1&0&0&0\\\hline 0&0&0&0&0&0\\0&0&1&0&0&0\\\hline 0&0&0&0&0&0\\1&0&1&0&0&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&1&0&1\\0&0&0&1&0&0\\\hline 0&0&0&0&0&0\\0&0&0&1&0&0\\\hline 0&0&0&0&0&0\\0&1&0&1&0&0\end{array}\right],
$$

$$
\left.\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 1&0&0&0&1&0\\0&0&0&0&0&0\\\hline 1&0&0&0&1&0\\0&0&0&0&0&0\end{array}\right],
\left[\begin{array}{cc|cc|cc} 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\0&0&0&0&0&0\\\hline 0&1&0&0&0&1\\0&0&0&0&0&0\\\hline 0&1&0&0&0&1\\0&0&0&0&0&0\end{array}\right]
\right\}
$$

As you can see, projecting this basis unto a single evaluation case's control matrix $R_{\$ij}$ will always give a uniformly random result, so this technique is sufficient to hide the entire control matrix when using control indirection. This is easiest to see with the case $i = j = 0$ (the top two rows of each matrix) because of the particular basis we chose, which is in echelon form when the matrices are written as vectors in row-major order.

Appendix B: POPF OT

B.1 Correlation Attack on Naïvely Batched MRR-OT

Consider the following strategy for a corrupt receiver, against naïve batching of the MRR OT protocol.

- The sender first sends its (reused) KA message $A = g^a$.
- In the first instance, run honestly with choice bit $c_1 = 0$. Generate $\phi_1 = \mathsf{Program}(0, g^b)$ for known b.
- The receiver can compute the value $\tilde{B} = \mathsf{Eval}(\phi_1, 1)$. The security of the POPF is that the receiver has no control over this value (*i.e.*, doesn't know its discrete log). The sender will compute OT output from this instance $r_{1,1} = \tilde{B}^a$.
- In the second instance, set $\phi_2 = \mathsf{Program}(0, \tilde{B} \cdot g^s)$ for known s. This means that the sender will compute OT output from this instance

$$r_{2,0} = \mathsf{Eval}(\phi_2, 0)^a = (\tilde{B} g^s)^a = \tilde{B}^a g^{as} = r_{1,1} g^{as} = r_{1,1} A^s$$

Note that the receiver can indeed compute A^s, and therefore it knows the ratio $r_{2,0}/r_{1,1}$.

Note also that if s is uniform then $\tilde{B} \cdot g^s$ is distributed uniformly. From the simulator's point of view, the receiver's behavior is *identically distributed* to honest behavior — running $\mathsf{Program}(0, X)$ for a uniform group element X. Hence, even if the ideal functionality is weakened to allow the receiver to specify correlations among the OT values, in this protocol the simulator has no way of detecting which correlation is appropriate.

B.2 Security Proofs for POPFs

B.2.1 Security Proof for Ideal Cipher (EKE) POPF

First, we have that $\mathbb{H}_{\mathsf{HSim}}, \mathbb{H}, \mathbb{H}_{\mathsf{Extract}}$ are indistinguishable, because they are all identical other than HSim and $\mathsf{Extract}$. $\mathsf{Extract}$ only reads $\mathbb{H}_{\mathsf{Extract}}$'s state, but does not modify it. Correctness follows directly from the correctness property of ideal ciphers.

There is an invariant that must be maintained for the ideal cipher to continue working properly. For any x, ϕ there must not be $y_1 \neq y_2$ such that $(x, y_1, \phi) \in T$ and $(x, y_2, \phi) \in T$. Similarly, for any x, y there must not be $\phi_1 \neq \phi_2$ such that $(x, y, \phi_1) \in T$ and $(x, y, \phi_2) \in T$. If either case happened, the output of E or E^{-1} would not be well-defined. A birthday bound shows that E and E^{-1} maintain this invariant. HSim explicitly aborts if it is asked to break the invariant.

Honest Simulation: Recall that the y^* sampled by $\mathfrak{D}$ must be uniform. By a birthday bound it is unique, not overlapping any previous y in T, with all but negligible probability. Then when REAL_PHI samples $\phi \leftarrow \mathsf{Program}(x^*, y^*)$, E will choose a uniformly random ϕ, the same distribution as sampled by HSim. It will also add (x^*, y^*, ϕ) to T. A similar birthday bound allows us to assume that this ϕ is also unique. Then a $E^{-1}(1 - x^*, \phi)$ query will be freshly random, so we can equivalently sample it ahead of time in a random value r_{1-x^*} and add $(1 - x^*, r_{1-x^*}, \phi)$ to T. But this is exactly what happens in SIM_PHI, as the abort in HSim will not occur because r_0 and r_1 will be unique.

Uncontrollable Outputs: We provide a sequence of hybrids, starting from the real distribution and ending at the ideal distribution.

1. Create an empty associative array Z at the start of $\mathbb{H}_{\mathsf{Extract}}$. Inside E^{-1}, whenever y is sampled as freshly random, compute and save $Z[y] = F(y)$. Then use the precomputed value as r in Uncontrollable Outputs instead of finding it again; since $y = \mathsf{Eval}(\phi, 1 - x^*)$ is calculated $Z[y]$ must have been precomputed. This step just rearranges the order of computations, and so is indistinguishable.

2. For the first E^{-1} query, instead of finding $F(y)$, sample $Z[y]$ as a uniformly random value in $\mathcal{O}$. We would like to use the 1-weak RO's security to prove that this is indistinguishable, but it seems like we are using F multiple times, and so cannot use the security property. However, this multiple use is illusory, as only single value in Z

will ever be used and the rest will be discarded.

More concretely, if we construct a reduction from this change in the hybrid proof to the 1-weak RO security, without loss of generality we can assume that the adversary aborts if this first $Z[y]$ does not end up being used to find r. If it is not used, then the two distributions are identical anyway. Now the other computations of F are completely unused and can be removed, allowing us to reduce to 1-weak RO security.

3. Repeat Step 2 for subsequent E^{-1} queries.

4. Undo the changes in step 1. That is, delay randomly sampling the entries of Z until Uncontrollable Outputs is run, so r will be uniformly random.

The advantage is bounded by adding up the advantages of every step. All security properties used a constant number of birthday bounds, which give the adversary an advantage of $O\left(\frac{q^2}{N}\right)$. Additionally, Uncontrollable Outputs used the security of the 1-weak RO once in each E^{-1} query, for a total advantage of $q\mathcal{A}(wRO)$.

B.2.2 Security Proof for Even-Mansour POPF

The proof is very similar to that of Theorem 3.5.1, and we will only describe the differences. The invariant no longer mentions x, and just says that each u should have at most one v and vice versa. Honest Simulation works similarly to before, pre-programming the randomness that Eval will produce. It will preserve the invariant as long as it does not produce the same ϕ or $\phi \oplus 1$ as one produced previously — ignoring a single bit should not affect collision resistance.

For extraction, the only additional complication is arguing that the x^* produced by $\mathsf{Extract}$ is the only one where $\mathsf{Eval}(\phi, x^*)$ is not freshly random on its first call. The only way for it to not be random is for there to have been a previous $\Pi(u)$ call that returned $\phi \oplus x$, but this cannot happen for multiple x as we have assumed that the other bits of ϕ are unique. $\mathsf{Extract}$ checks T to find the unique call $v = \Pi(u)$ where this happened, which must have been the first, then finds x^* such that $v = \phi \oplus x^*$.

B.2.3 Security Proof for Masny-Rindal POPF

First, we have $\mathbb{H}_{\mathsf{HSim}} \approxeq \mathbb{H} \approxeq \mathbb{H}_{\mathsf{Extract}}$, because they are all identical when neither HSim nor $\mathsf{Extract}$ have been called, though some have extra bookkeeping. $\mathsf{Extract}$ only reads $\mathbb{H}_{\mathsf{Extract}}$'s state, but does not modify it. Correctness is ensured as if $\phi = (s_0, y \cdot (H_1(s_0))^{-1}) \leftarrow \mathsf{Program}(1, y)$ then $y = y \cdot (H_1(s_0))^{-1} \cdot H_1(s_0) = \mathsf{Eval}(\phi, 1)$, and similarly for $x^* = 0$.

Honest Simulation: In the generation of $\phi = (s_0, s_1)$, we have that s_{1-x^*} is uniform by construction, and furthermore since y^* is sampled from $\mathfrak{D}$ and must be uniform we have that $\mathsf{Program}$ will generate uniform s_{x^*}. s_{x^*} is distinguishable only if $H_{x^*}(s_{1-x^*})$ had previously been queried which for uniform s_{1-x^*} only happens with probability at most $\frac{q}{|\mathbb{G}|}$.

We also have uniqueness of s_0, s_1 with all but negligible probability by an application of a birthday bound over $\mathbb{G}$, so using U to program H_x will not interfere with any previous H_x queries or other HSim calls. Finally, we have that $\mathsf{Eval}(\phi, 1 - x^*) = s_{1-x^*} \cdot H_{1-x^*}(s_{x^*})$ is close to uniform (since the query $H_{1-x^*}(s_{x^*})$ will be fresh with probability $1 - \frac{q}{|\mathbb{G}|}$), so we can sample it early and save it in U. This is exactly how HSim functions in the ideal world. Finally, by the enforced consistency of HSim we have that the outputs r_0, r_1 from Eval are the values provided to HSim. This can be verified as if $\phi = (s_0, s_1) \leftarrow \mathsf{HSim}(r_0, r_1)$ then $\mathsf{Eval}(\phi, x) = s_x \cdot H_x(s_{1-x}) = s_x \cdot (s_x)^{-1} \cdot r_x = r_x$.

Uncontrollable Outputs: Similarly to the proof of Claim 4.3 in [MR19], we would like to guess a query $H_{x^*}(u^*)$. Following [MRR20], we will call this query an anchor query. The idea is that this is the query made by $\mathsf{Program}$, or however the adversary constructed ϕ. Any subsequent query $H_{1-x^*}(u)$ can be programmed to be ${u^*}^{-1} \cdot y$ to make $\mathsf{Eval}(\phi, 1 - x^*) = y$ if we guessed correctly. We will know we guessed correctly if later u^* is part of the ϕ that is input to $\mathsf{Extract}$.

However, instead of guessing a query like in [MR19], we will use a hybrid proof to get the same result. Some hybrids will make changes that are only useful if a guess is correct, but do nothing if the guess is wrong. Here is our sequence of hybrids starting with an interaction with the real protocol and ending with the ideal world.

1. Create a new associative array Z at the start of $\mathbb{H}_{\mathsf{Extract}}$. When a uniformly random value v is sampled in $H_x(u)$, look for possible anchor queries by iterating over all previous queries $H_{1-x}(u^*)$, and in each iteration, compute and save $Z[x, u^*, u] = F(u^* \cdot v)$.

2. Use the precomputed value $Z[1 - x^*, s_{1-x^*}, s_{x^*}]$ as r in the Uncontrollable Outputs

distribution, instead of finding it again with $F(\mathsf{Eval}(\phi, 1 - x^*))$, if it has already been computed.

3. For $1 \leq i \leq q$ and $1 \leq j < i$, repeat the following sequence of hybrids. That is, perform these transformations for the ith query to H and jth iteration of the loop over prior queries in H, for a total of $\frac{q(q-1)}{2}$ repetitions.

   (a) Instead of sampling $v \leftarrow \mathbb{G}$ in the ith query $H_x(u)$, use u^* from the jth iteration of the loop over possible anchor queries to sample $y \leftarrow \mathbb{G}$ and set $v = {u^*}^{-1} \cdot y$.

   (b) In the jth iteration of the ith query to H, instead of computing $Z[\phi, x] = F(y)$, sample $Z[\phi, x]$ as a uniformly random value in $\mathcal{O}$. This change is indistinguishable because F is a 1-weak RO.[1]

   (c) Undo the changes in step 3a, so v is sampled as $v \leftarrow \mathbb{G}$ again.

4. Undo the changes in step 1. That is, wait until the Uncontrollable Outputs distribution is run before sampling the entries in Z. If $Z[1 - x^*, s_{1-x^*}, s_{x^*}]$ is present, the Uncontrollable Outputs distribution now gets a uniformly random r instead of the output of F.

5. Finally, also replace r with random if it does not appear in Z. In this case, either $H_0(s_1)$ or $H_1(s_0)$ must not have been queried before, as otherwise whichever was queried first would be the anchor query and then in the second query r would be precomputed and saved in Z. Either x^* will be the query that was made, or neither were queried — either way, $H_{x^*}(s_{1-x^*})$ must not have been queried. Therefore $\mathsf{Eval}(\phi, 1 - x^*)$ must return a fresh uniformly random value, and the 1-weak RO property allows us to replace F's output with random.

For Honest Simulation, the adversary just gets a birthday bound advantage $O\left(\frac{q^2}{N}\right)$. But in Uncontrollable Outputs we use the security of 1-weak RO, which allows them an additional advantage. We use it $\frac{q(q-1)}{2}$ times in step 3b, and one additional time in step 5. Therefore, Uncontrollable Outputs allows the adversary an advantage of $\frac{q^2-q+2}{2}\mathcal{A}(wRO)$, on top of the birthday bounds.

[1] Although it appears that F is used in multiple places, only a single one is actually used in the end. See the proof for the EKE POPF for details.

B.2.4 Security Proof for Feistel POPF

All three $\mathbb{H}$s are clearly indistinguishable on their common interface, as without any HSim queries they behave identically. For correctness, notice that

$$\begin{aligned}\mathsf{Eval}(\mathsf{Program}(x^*, y^*), x^*) &= H(x^*, \iota(t)x^* + u - \iota(t)x^*) \cdot t \\ &= H(x^*, u) \cdot H(x^*, u)^{-1} \cdot y = y\end{aligned}$$

Honest Simulation: REAL_PHI chooses u uniformly randomly, so the $H(x^*, u)$ query will return fresh randomness as it has negligible probability of overlapping with any other query. Therefore $\phi = (s, t)$ will be uniformly random, the same distribution as produced by HSim. Next, we just need to prove that HSim successfully programs Eval, *i.e.*, that r_0 and r_1 will match between REAL_PHI and SIM_PHIs. It succeeds if the second if-statement in $H(x, \iota(t)x + s)$ is triggered, because then $\mathsf{Eval}(\phi, x)$ will produce $H(x, \iota(t)x + s) \cdot t = r_x \cdot t^{-1} \cdot t = r_x$.

There are two ways that it could fail: either $H(x, \iota(t)x + s)$ had already been queried before HSim was called, or $U[x, \iota(t)x + s]$ gets overwritten by another HSim query. The former case has negligible probability because there are at most q previous queries $H(x, u)$, each would cause a failure only if $s = u - \iota(t)x$, and s is chosen uniformly at random after the $H(x, u)$ query. For the latter case, notice that every $\phi = (s, t)$ defines a unique line $u = \iota(t)x + s$. A pair of such lines would have to intersect at some $x \in \{0, 1\}$ for U to be overwritten. They can only intersect for a single value of x, and since both lines are uniformly random, this x will be uniformly random in $\mathbb{F}$, so there is negligible probability of an intersection for $x \in \{0, 1\}$.[2]

Uncontrollable Outputs: We use MRR20's notion of anchor queries for this proof. An anchor query is a query made during $\mathsf{Program}$ that can be used by $\mathbb{H}_{\mathsf{Extract}}$ to identify ϕ before it is revealed by the adversary. More specifically, a query $H(x^*, u^*)$ is the anchor query if it is the first query on the line $u^* = \iota(t)x^* + s$. It is, in fact, the query that $\mathsf{Extract}$ searches for in order to find x^*. The anchor query is needed to find t early and program the subsequent H queries such that Eval outputs a random value for the weak random oracle.

[MRR20] guessed the anchor query, taking a factor q security loss, and we will do

[2]When handling exponentially large x this becomes problematic, but can be fixed by hashing x with another random oracle H' before multiplying by ι, so that the adversary would need to solve a hard preimage problem to find the x corresponding to an intersection.

something similar with hybrids. In a chain of hybrids, we guess a possible anchor query and make some changes that make progress if the guess was correct and do nothing if we are wrong. Once the anchor query $H(x^*, u^*)$ has been made, on each subsequent query $H(x, u)$ we assume it is on the same $u = \iota(t)x + s$ line and use this to find ϕ and program $H(x, u)$. Specifically, $t = \iota^{-1}\left(\frac{u-u^*}{x-x^*}\right)$ can be found from the slope of the line through the points $(x^*, u^*), (x, u)$, and $s = u - \iota(t)x$ is the u-axis intercept. If this assumption is wrong there is no harm, similarly to the anchor query.

We use the following sequence of hybrids, from the real distribution to the ideal distribution.

1. Create an empty associative array Z at the start of $\mathbb{H}_{\mathsf{Extract}}$. Inside H, whenever v is sampled as freshly random, iterate over all previous queries $H(x^*, u^*)$ for $x^* \neq x$ to look for possible anchor queries. For each such query, compute $\phi = (s, t)$ as described above. Skip to the next H-query if this ϕ came up in a previous iteration, as then it is impossible for $H(x^*, u^*)$ to be the anchor query for ϕ. Compute and save $Z[\phi, x] = F(v \cdot t)$.

2. Use the precomputed value $Z[\phi, 1 - x^*]$ as r in the Uncontrollable Outputs distribution if it is present, instead of computing it again as $F(\mathsf{Eval}(\phi, 1 - x^*))$.

3. For $1 \leq i \leq q$ and $1 \leq j < i$, repeat the following sequence of hybrids. That is, perform these transformations for the ith query to H and jth possible anchor query, for a total of at most $\frac{q(q-1)}{2}$ repetitions.

   (a) Instead of sampling $v \leftarrow \mathbb{G}$ in the ith query $H(x, u)$, use the inferred $\phi = (s, t)$ from the jth possible anchor query to sample $y \leftarrow \mathbb{G}$ and set $v = y \cdot t^{-1}$.

   (b) In the jth iteration in the ith query to H, instead of computing $Z[\phi, x] = F(y)$, sample $Z[\phi, x]$ as a uniformly random value in $\mathcal{O}$. This change is indistinguishable because F is a 1-weak RO.[3]

   (c) Undo the changes in step 3a, so v is sampled as $v \leftarrow \mathbb{G}$ again.

4. Undo the changes in step 1. That is, delay randomly sampling the entries in Z until Uncontrollable Outputs is run. If $Z[\phi, 1 - x^*]$ is present, the Uncontrollable Outputs

[3]Although it may seem that F is used multiple times, only one of these values will actually be used in the end. For more details, see the proof for the EKE POPF.

distribution now gets a uniformly random r instead of the output of F.

5. Finally, if $Z[\phi, 1 - x^*]$ is not present, replace the output r of the 1-weak RO with random. In this case $H\big(1 - x^*, \iota(t)(1 - x^*) + s\big)$ cannot have been queried before, as otherwise either the anchor query would be at $1 - x^*$ not x^*, or the anchor query $H(x^*, \iota(t)x^* + s)$ would have been made before the other query and so $Z[\phi, 1 - x^*]$ would be present, which are both contradictions. Therefore, the call to $\mathsf{Eval}(\phi, 1 - x^*)$ in the Uncontrollable Outputs distribution must return fresh randomness, and then the 1-weak RO property allows us to replace r with random.

Again, we bound the advantage by summing the advantages of each step in the hybrid proof. Excluding the birthday bounds, the only advantage the adversary gets is in Uncontrollable Outputs, when we use the 1-weak RO property. We use it $\frac{q(q-1)}{2}$ times in step 3b, once for each pair of oracle queries, because we must loop over every previous query as a possible anchor query. Finally, we use it one last time in step 5. Therefore, Uncontrollable Outputs allows the adversary an advantage of $\frac{q^2-q+2}{2}\mathcal{A}(wRO)$, on top of the birthday bounds.

Appendix C: SoftSpokenOT

C.1 Correlation Robust Hash Constructions

Proposition 4.2.6. *A random oracle* $\mathit{RO}\colon \mathbb{F}_q^{nc} \times \{0,1\}^t \to \{0,1\}^\lambda$ *is a* $(p, q, \mathcal{C}, \{0,1\}^t, \text{Affine}(\mathbb{F}_p^{knc}))$*-TCR hash, with distinguisher advantage at most* $\tau_{max}\left(\mathfrak{q} + \frac{1}{2}\mathfrak{q}'\right)q^{-d_\mathcal{C}}$*. Here,* τ_{max} *is the maximum number of times* QUERY *is called with the same* τ*,* $\mathfrak{q}$ *is the number of RO queries made by the distinguisher, and* $\mathfrak{q}'$ *is the number of calls to* QUERY*.*

Proof. We argue indistinguishability, going from the ideal oracle to the real oracle. But first, we must define two bad events, and bound the probability of their occurrence in the ideal world. For both events, we actually use a slightly modified ideal world where the abort in LEAK is delayed to the end. This makes no difference, as it will always give the same output (i.e. "abort") in the end. These bad events are:

1. Adversary runs queries $\mathsf{RO}(\tau, \vec{u})$ and $\text{QUERY}(\vec{x}, \vec{y}, \tau)$ such that $\vec{u} = \vec{x}G_\mathcal{C} \odot \vec{\Delta} + \vec{y}$.
2. Adversary runs queries $\text{QUERY}(\vec{x}, \vec{y}, \tau)$ and $\text{QUERY}(\vec{x}', \vec{y}', \tau)$ such that $\vec{x}G_\mathcal{C} \odot \vec{\Delta} + \vec{y} = \vec{x}'G_\mathcal{C} \odot \vec{\Delta} + \vec{y}'$. Equivalently, $\vec{y} - \vec{y}' = (\vec{x}' - \vec{x})G_\mathcal{C} \odot \vec{\Delta}$.

For both events, an equation of the form $\vec{y} = \vec{c} \odot \vec{\Delta}$ must hold, for some non-zero code word $\vec{c} \in \mathcal{C}$. Since $\|\vec{c}\|_0 \geq d_\mathcal{C}$, any such equation has probability at most $q^{-d_\mathcal{C}}$. There are at most $\mathfrak{q}\tau_{\max}$ suitable query pairs for the first bad event, and $\mathfrak{q}'\tau_{\max}/2$ suitable pairs for the second. Therefore, the union bound shows the probability of either event occurring is at most $\tau_{\max}\left(\mathfrak{q} + \frac{1}{2}\mathfrak{q}'\right)q^{-d_\mathcal{C}}$.

In QUERY we can replace the sampling of $z \leftarrow \{0,1\}^\lambda$ with its value in the real oracle, $\mathsf{RO}(\vec{x}G_\mathcal{C} \odot \vec{\Delta} + \vec{y}, \tau)$. Assuming that the bad events never happens, no RO query in QUERY will have the same inputs as any other RO query, from either the adversary (Event 1) or another call to QUERY (Event 2). Therefore this change is indistinguishable. We are now at the real world. □

Proposition 4.2.7. *Let* $\mathit{Enc}\colon \{0,1\}^t \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ *be an ideal cipher, and* $\iota\colon \mathbb{F}_q^{nc} \to \{0,1\}^\lambda$ *be an injection. Then* $H(\vec{y}, \tau) = \mathit{Enc}(\tau, \iota(\vec{y})) \oplus \iota(\vec{y})$ *is a* $(p, q, \mathcal{C}, \{0,1\}^t, \text{Affine}(\mathbb{F}_p^{knc}))$*-TCR hash. The distinguisher's advantage is at most* $\tau_{max}\left((2\mathfrak{q} + \frac{1}{2}\mathfrak{q}')q^{-d_\mathcal{C}} + \frac{1}{2}\mathfrak{q}'2^{-\lambda}\right)$*, with* $\mathfrak{q}$

and $\mathfrak{q}'$ *as in Theorem 4.2.6.*

Proof. We start by defining four bad events, and prove an upper bound on the probability of their occurrence when the distinguisher is given access to the oracle $\mathsf{TCR\text{-}ideal}^{H,p,q,\mathcal{C},\mathcal{L}}$. As with the RO-based TCR, we modify the ideal world slightly so as to ignore the aborts when bounding the bad events. The first two bad event are essentially the same as for the RO-based TCR hash, while the others come from Enc having an inverse. Note that if the adversary queries Enc, it is also counted as a Enc^{-1} query for the purposes of the bad events — all that matters for these events is the relation between Enc inputs and outputs.

1. Adversary queries $\mathsf{Enc}(\tau, u)$ and $\textsc{query}(\bar{x}, \bar{y}, \tau)$ such that $u = \iota(\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y})$. Equivalently, $\iota^{-1}(u)$ exists and equals $\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y}$.
2. Adversary queries $\textsc{query}(\bar{x}, \bar{y}, \tau)$ and $\textsc{query}(\bar{x}', \bar{y}', \tau)$ such that $\iota(\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y}) = \iota(\bar{x}'G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y}')$. Equivalently, $\bar{y} - \bar{y}' = (\bar{x}' - \bar{x})G_{\mathcal{C}} \odot \bar{\Delta}$.
3. Adversary queries $\mathsf{Enc}^{-1}(\tau, v)$ and $z = \textsc{query}(\bar{x}, \bar{y}, \tau)$ such that $v \oplus z = \iota(\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y})$. Equivalently, $\iota^{-1}(v \oplus z)$ exists and equals $\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y}$.
4. Adversary queries $z = \textsc{query}(\bar{x}, \bar{y}, \tau)$ and $z' = \textsc{query}(\bar{x}', \bar{y}', \tau)$ such that $z \oplus z' = \iota(\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y}) \oplus \iota(\bar{x}'G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y}')$.

Events 1–3 all require $\bar{c} \odot \bar{\Delta}$ to take a specific value, for some nonzero codeword $\bar{c} \in \mathcal{C}$. Each has probability at most q^{-dc}, for any pair of queries. Event 4 instead requires that $z \oplus z'$ take a particular value, when either z or z' will be a freshly random λ-bit string. Therefore, it has probability at most $2^{-\lambda}$, for any pair of queries. There are at most $\mathfrak{q}\tau_{\max}$ suitable query pairs for Events 1 and 3, and at most $\mathfrak{q}'\tau_{\max}/2$ suitable pairs for Events 2 and 4. Therefore, a union bound shows that probability of any bad event occurring is it most $\tau_{\max}\left((2\mathfrak{q} + \frac{1}{2}\mathfrak{q}')q^{-dc} + \frac{1}{2}\mathfrak{q}'2^{-\lambda}\right)$.

Next, we argue indistinguishability. Replacing the ideal oracle with the real oracle replaces the random value $z \leftarrow \{0,1\}^{\lambda}$ with $z = \mathsf{Enc}(\tau, u) \oplus u$, where $u = \iota(\bar{x}G_{\mathcal{C}} \odot \bar{\Delta} + \bar{y})$. The Enc call will always return fresh randomness, which will never be revealed again, because it cannot overlap with any Enc call (Event 1) or other call to QUERY (Event 2). Past Enc calls also rule out using those same value again, so $z \oplus u$ cannot be the output of another Enc call, nor can it equal $z' \oplus u'$ for another call to QUERY. But these are exactly what is ruled out by Event 3 and Event 4, respectively. Therefore, once the bad events have been excluded, the real and ideal worlds oracles identically. □

C.2 OOS Details

This section is written in using the notation of OOS. Please review that paper to familiarize yourself with the notation.

In their proof for security against a malicious receiver, OOS define a set $E \subseteq [n_{\mathcal{C}}]$ of indices of the sender's secret b. Passing the consistency check requires that the receiver guess some bits of b; E is the set of these bits. E depends on the corrections $\mathbf{u}^j$ sent by the receiver, and it also depends on the consistency check choice bits, which would be computed by an honest receiver as $\mathbf{w}^{(\ell)} = \sum_{i\in[m]} \mathbf{w}_i x_i^{(\ell)} + \mathbf{w}_{m+\ell}$. For example, assume that OOS is used for $\binom{2}{1}$-OT by setting $\mathcal{C}$ to be the repetition code, and that the receiver lies in its correction for a single one of the extended OTs by providing the first half of the correction as if its choice bit was zero, and the second half as if it were one. During the consistency check it could compute the $\mathbf{w}^{(\ell)}$ as if its choice bit were 0, or as if the choice bit were 1. In the former case, E would be the index range $\left[\frac{n_{\mathcal{C}}}{2}+1, n_{\mathcal{C}}\right]$, while in the latter it would be $\left[1, \frac{n_{\mathcal{C}}}{2}\right]$.

The flaw in OOS's proof is in the proof of Proposition 2, which states that the simulator can extract the receiver's choice bits $\mathbf{w}_i$ from the consistency check, such that the encoded choice bits $\mathcal{C}(\mathbf{w}_i)$ agree with correction the receiver sent, except at the indices in E. The proposition is quoted below.

> **Proposition 2.** If the check passes then with probability at least $1-2^{-s}-2^{-d_{\mathcal{C}}}$, $\mathcal{S}$ can extract values $\mathbf{w}_i \in \mathbb{F}_2^{k_{\mathcal{C}}}, \mathbf{e}_i \in \mathbb{F}_2^{n_{\mathcal{C}}}$, for $i \in [m]$, such that
>
> 1. $\mathbf{c}_i = \mathcal{C}(\mathbf{w}_i) + \mathbf{e}_i$
> 2. $\mathbf{e}_i[j] = 0$ for all $j \notin E$

Its proof is based on the following lemma.

> **Lemma 1.** Let $\mathcal{C}$ be a linear code of length $n_{\mathcal{C}}$, $m' = m + s$ be an integer and $\mathbf{c}_i \in \mathbb{F}_2^{n_{\mathcal{C}}}$, for $i \in [m']$, such that there exists at least one $j \in [m]$ with $\mathbf{c}_j \notin \mathcal{C}$. Then, if $x_i^{(\ell)} \xleftarrow{\$} \mathbb{F}_2$, we have that:
>
> $$\Pr\Big(\forall \ell \in [s], \sum_{i\in[m]} \mathbf{c}_i \cdot x_i^{(\ell)} + \mathbf{c}_{m+\ell} \in \mathcal{C}\Big) \leq 2^{-s}.$$
>
> *Proof.* ... □

This seems reasonable enough. The span of the $\mathbf{c}_i$ is not contained in $\mathcal{C}$, so a collection of random vectors in the span are unlikely to all be in $\mathcal{C}$. Notice that this assumes that $\mathcal{C}$ and the $\mathbf{c}_i$ are known *before* the $x_i^{(\ell)}$ are sampled, which matches the lemma statement. Now let's see how this lemma is used:

> ... For a vector $\mathbf{c} \in \mathbb{F}_2^{nc}$, define $\mathbf{c}_{-E}$ to be the vector obtained by removing the positions $j \in E$. Let $\mathcal{C}_{-E}$ be the *punctured code* consisting of the codewords $\{\mathbf{c}_{-E} : \mathbf{c} \in \mathcal{C}\}$.
>
> By definition of E, we must have $\overline{\mathbf{c}}_{-E}^{(\ell)} = 0$, and because $\mathbf{c}_*^{(\ell)} = \overline{\mathbf{c}}^{(\ell)} + \mathcal{C}(w_*^{(\ell)})$, we also have $(\mathbf{c}_*^{(\ell)})_{-E} \in \mathcal{C}_{-E}$, for every $\ell \in [s]$. Therefore, by applying Lemma 1 with the code $\mathcal{C}_{-E}$ and vectors $(\mathbf{c}_1)_{-E}, \ldots, (\mathbf{c}_{m'})_{-E}$, it holds that for every $i \in [m]$, $(\mathbf{c}_i)_{-E} \in \mathcal{C}_{-E}$, except with probability $\leq 2^{-s}$. ...

Note that the code given to Lemma 1 is not the code $\mathcal{C}$ that is fixed in advance, it is the punctured code $\mathcal{C}_{-E}$, which depends on E. Additionally, the punctured vectors $(\mathbf{c}_1)_{-E}, \ldots, (\mathbf{c}_{m'})_{-E}$ also depend on E. Here is the problem: E *depends* on the consistency check choice bits $\mathbf{w}^{(\ell)}$ that the receiver sent, *after* receiving the challenge bits $x_i^{(\ell)} \xleftarrow{\$} \mathbb{F}_2$. That is, E can depend on $x_i^{(\ell)} \xleftarrow{\$} \mathbb{F}_2$, so Lemma 1 cannot be applied here.

C.3 PSS Details

At a high level, PSS's consistency check seems the same as OOS's specialized to Walsh–Hadamard codes $\mathcal{C}_{\mathtt{WH}}^{\kappa}$, which have length κ and minimum distance $\kappa/2$. Random bits $w_i^{(l)}$ are sampled for $l \in [\mu]$ and $i \in [m + \mu]$. They choose linear combinations of the OT correlations for a consistency check:

$$\mathbf{a}^{(l)} = \bigoplus_{i=1}^{m+\mu} w_i^{(l)} \mathbf{a}_i \qquad \mathbf{b}^{(l)} = \bigoplus_{i=1}^{m+\mu} w_i^{(l)} \mathbf{b}_i \qquad \mathbf{e}^{(l)} = \bigoplus_{i=1}^{m+\mu} w_i^{(l)} \mathbf{e}_i,$$

where $\mathbf{a}^{(l)}$ is computed by the sender, while $\mathbf{b}^{(l)}$ and $\mathbf{e}^{(l)}$ are computed by the receiver. If the consistency check were $\mathbf{a}^{(l)} = \mathbf{b}^{(l)} \oplus (\mathbf{s} \odot \mathbf{e}^{(l)})$ for all l, with the $\mathbf{e}^{(l)}$ required to be code words in $\mathcal{C}_{\mathtt{WH}}^{\kappa}$, then it would work the same as OOS. However, PSS attempt to save communication

by only checking the XOR of all bits in each $\mathbf{a}^{(l)}$. That is, the PSS consistency check is

$$\bigoplus_{j=1}^{\kappa} a_j^{(l)} = \bigoplus_{j=1}^{\kappa} b_j^{(l)} \oplus \bigoplus_{j=1}^{\kappa} s_j e_j^{(l)},$$

so that $b^{(l)} = \bigoplus_{j=1}^{\kappa} b_j^{(l)}$ can be sent instead of $\mathbf{b}^{(l)}$. Unfortunately, a malicious receiver can take advantage by making guesses on XORs of several bits from $\mathbf{s}$, rather than having to guess each bit individually.

Have the receiver pick $\mathbf{e}_i$ to have an interval of 1 bits, and the rest be zeros. That is, when $i \leq \lceil \frac{\kappa}{N} \rceil$, set $e_i^j = 1$ if $N(i-1) < j \leq Ni$, and 0 otherwise, where the interval width N is a parameter of the attack. The remainder, where $i > \lceil \frac{\kappa}{N} \rceil$, are all set to be zero. For the consistency check, send all zeros for $\mathbf{e}^{(l)}$ (more precisely, send the index of the all zeros codeword in the Walsh–Hadamard code), and send the honest value for $b^{(l)}$. The sender's values $\mathbf{a}_i$ equal $\mathbf{b}_i \oplus (\mathbf{s} \odot \mathbf{e}_i)$, so the consistency check becomes

$$\begin{aligned}
\bigoplus_{j=1}^{\kappa} \bigoplus_{i=1}^{m+\mu} w_i^{(l)} a_i^j &= \bigoplus_{j=1}^{\kappa} \bigoplus_{i=1}^{m+\mu} w_i^{(l)} b_i^j \\
0 &= \bigoplus_{j=1}^{\kappa} \bigoplus_{i=1}^{m+\mu} w_i^{(l)} s^j e_i^j \\
0 &= \bigoplus_{i=1}^{\lceil \kappa/N \rceil} w_i^{(l)} \bigoplus_{j=N(i-1)+1}^{Ni} s^j.
\end{aligned}$$

Therefore, the consistency check passes if the XORs of intervals $\bigoplus_{j=N(i-1)+1}^{Ni} s^j$ are all zero. There are $\lceil \frac{\kappa}{N} \rceil$ of these intervals, so this has probability $2^{-\lceil \kappa/N \rceil}$.

If the check passes, we can now break the OT extension with only $\lceil \frac{\kappa}{N} \rceil 2^{N-1}$ hash evaluations. That is, if the sender sends the all zeros message for each of its first $\lceil \frac{\kappa}{N} \rceil$ OT results, the receiver can solve for $\mathbf{s}$ and learn every other OT message. The receiver can do this by using the hash to check guesses of $\mathbf{a}_i$. Since $\mathbf{a}_i = \mathbf{b}_i \oplus (\mathbf{s} \odot \mathbf{e}_i)$ depends on only N bits of $\mathbf{s}$, and the XOR of these bits is known to be zero, there are only 2^{N-1} possibilities to check. Repeating this for all $i \leq \lceil \frac{\kappa}{N} \rceil$ recovers $\mathbf{s}$.

Proof Flaws. How was this attack missed by PSS's proof? There are two major issues in the proof that the attack exploits. First, PSS's Lemma IV.4 proof implicitly assumes that

there will not be a linear dependency between different bits of the consistency check, and our attack causes such a linear dependency by forcing the consistency check to only depend on $\mathbf{s}$ through $\bigoplus_{j=N(i-1)+1}^{Ni} s^j$. Second, while proving that their first and second hybrids are indistinguishable they say that the sender's mask for $x_{i,j}$ is $H(i, \mathbf{b}_i \oplus (\mathbf{s} \odot (\mathbf{c}_{r_i} \oplus \mathbf{c}_j)))$, when it is really $H(i, \mathbf{a}_i \oplus (\mathbf{s} \odot \mathbf{c}_j))$. These are only equal when the receiver behaves honestly.

C.4 KOS Details

The notation of KOS is used for this section, so please review that paper if you need to.

The most important part of KOS's proof of security against a malicious receiver is the behavior of their consistency check. They give that information in the following lemma.

> **Lemma 1.** Let $S_\Delta \subseteq \mathbb{F}_2^\lambda$ be the set of all Δ for which the correlation check passes, given the view of the receiver. Except with probability $2^{-\lambda}$, there exists $k \in \mathbb{N}$ such that
>
> 1. $|S_\Delta| = 2^k$
> 2. For every $\mathbf{s} \in \{\mathbf{x}^i\}_{i\in[\lambda]}$, let $H_\mathbf{s} = \{i \in [\lambda] \mid \mathbf{s} = \mathbf{x}^i\}$. Then one of the following holds:
>   - For all $i \in H_\mathbf{s}$ and any $\Delta^{(1)}, \Delta^{(2)} \in S_\Delta$, $\Delta_i^{(1)} = \Delta_i^{(2)}$.
>   - $k \leq |H_\mathbf{s}|$, and $|\{\Delta_{H_\mathbf{s}}\}_{\Delta\in S_\Delta}| = 2^k$, where $\Delta_{H_\mathbf{s}}$ denotes the vector consisting of the bits $\{\Delta_i\}_{i\in H_\mathbf{s}}$. In other words, S_Δ restricted to the bits corresponding to $H_\mathbf{s}$ has entropy at least k.
>
> Furthermore, there exists $\hat{\mathbf{s}}$ such that $k \leq |H_{\hat{\mathbf{s}}}|$.
>
> *Proof.* See full version.

As of writing, no full version has been made public. However, the authors of KOS were kind enough to provide an unpublished draft of the full version [KOS21]. It contains an attempted proof of Lemma 1. They first observe that Lemma 1 is trivial if $S_\Delta \leq 2$, then define $\Delta^1, \Delta^2, \Delta^3$ to be three distinct elements of S_Δ. They let $\Delta' = \Delta^1 + \Delta^2$ and $\Delta'' = \Delta^1 + \Delta^3$, then derive the following equation from the consistency check.

$$0 = \sum_{j=1}^{\ell} \chi_j \cdot \underbrace{((\mathbf{x}_j * \Delta') \cdot \Delta'' - (\mathbf{x}_j * \Delta'') \cdot \Delta')}_{\tilde{x}_j}.$$

Since $\{\chi_j\}_{j\in[\ell]}$ are independently random, the equality holds with probability $2^{-\lambda}$ if not all $\{\tilde{x}_j\}_{j\in[\ell]}$ are 0 by the principle of deferred decisions. Hence, we assume that $\tilde{x}_j = 0$ for all $j \in [\ell]$.

The logic is to delay the sampling of χ_j until this sum is calculated, at which point it's clear that the output will be uniformly random in $\mathbb{F}_{2^\lambda}$ unless every $\tilde{x}_j$ is zero. However, we cannot actually delay the sampling of χ_j until then, because they are used earlier. Similarly to the problem with E in OOS, S_Δ depends on which consistency check message x the receiver sends. In an extreme case, if the receiver behaves entirely honestly, then S_Δ will be all of $\mathbb{F}_2^\lambda$, but if after looking at the χ_j they decide to send a different x instead, then they will have to guess all of Δ and so $|S_\Delta|$ will be 1. As differences between arbitrarily selected members of S_Δ, Δ' and Δ'' also depend on the χ_j, making it impossible to delay the sampling.

C.4.1 Collision Attack

Unlike OOS where we have not managed to find a counterexample to their stated Proposition 2, we have managed to find some (impractical) attacks that break KOS's Lemma 1. The simplest succeeds with probability roughly $\lambda^2 2^{-\lambda-1}$, based on getting a collision in a set of λ vectors of length λ. It works by generating $\mathbf{x}_1, \ldots, \mathbf{x}_{\ell'}$ as uniformly random elements of $\mathbb{F}_2^\lambda$, instead of the monochrome vectors that an honest receiver would generate.

Similarly to how an honest receiver would compute its consistency check message as $x = \sum_j x_j \cdot \chi_j$, for each column i let $\bar{x}_i = \sum_j x_j^i \cdot \chi_j$. For an honest receiver every $\mathbf{x}^i$ is the same, so every $\bar{x}_i$ will be exactly x. Similarly, instead of just computing $t = \sum_j \mathbf{t}_j \cdot \chi_j$ and $q = \sum_j \mathbf{q}_j \cdot \chi_j$, find $\bar{t}_i = \sum_j t_j^i \cdot \chi_j$ and $\bar{q}_i = \sum_j q_j^i \cdot \chi_j$. Since $q_j^i = t_j^i + x_j^i \cdot \Delta_i$, we have $\bar{q}_i = \bar{t}_i + \bar{x}_i \cdot \Delta_i$.

Let α be the generator of $\mathbb{F}_{2^\lambda}$ that is being used by the protocol to represent elements of $\mathbb{F}_2^\lambda$ as field elements in $\mathbb{F}_{2^\lambda}$. That is, a vector $\mathbf{v}$ becomes the field element $v_1 + \alpha \cdot v_2 + \cdots + \alpha^{\lambda-1} \cdot v_\lambda$. Then $t = \bar{t}_1 + \alpha \cdot \bar{t}_2 + \cdots + \alpha^{\lambda-1} \cdot \bar{t}_\lambda$, and similarly for q. Assume that the receiver gives the

honest value for t.[1] Then the consistency check becomes

$$t = q + x \cdot \Delta = \sum_i \bar{q}_i \cdot \alpha^{i-1} + x \cdot \Delta = \sum_i \bar{t}_i \cdot \alpha^{i-1} + \sum_i \bar{x}_i \cdot \alpha^{i-1} \cdot \Delta_i + x \cdot \Delta$$
$$0 = \sum_i \bar{x}_i \cdot \alpha^{i-1} \cdot \Delta_i + x \cdot \Delta = \sum_i (\bar{x}_i + x) \cdot \alpha^{i-1} \cdot \Delta_i.$$

Therefore, if the receiver sets $x = \bar{x}_i$ for some i then they won't have to guess Δ_i. They will only have to guess all $\Delta_{i'}$ for which $x \neq \bar{x}_{i'}$.

If there is a collision, so there are $i \neq i'$ such that $\bar{x}_i = \bar{x}_{i'}$, setting $x = \bar{x}_i$ forces $|S_\Delta| \geq 4$ and so $k \geq 2$. The $\bar{x}_i$ will all be uniformly random elements of $\mathbb{F}_{2^\lambda}$, so this happens with probability roughly $\lambda^2 2^{-\lambda-1}$. For sufficiently large ℓ each $|H_{\mathbf{x}^i}|$ will be 1, as the columns $\mathbf{x}^i$ of the matrix whose rows are $\mathbf{x}_i$ will be unique. Therefore, this attack contradicts Lemma 1.

C.4.2 Subfield Attack

At least for a special case, a slightly more practical attack seems to be possible. Assume that λ is divisible by 20, and that the minimal polynomial of α (the element being used to represent $\mathbb{F}_{2^\lambda}$) can be written in the form $P(\alpha^5)$ for some irreducible polynomial P.[2] Then $\mathbb{F}_{2^\lambda}$ has a subfield $\mathbb{F}_{2^{\lambda/5}}$ that is generated by α^5. In a subfield attack, a malicious receiver arranges the consistency check so that for any $\Delta \in S_\Delta$ (i.e. any Δ that allows the consistency check to pass) and for any $s \in \mathbb{F}_{2^{\lambda/5}}$, $s\Delta$ is also in S_Δ. This makes S_Δ be a vector space over $\mathbb{F}_{2^{\lambda/5}}$. The receiver tries to make the dimension of S_Δ over $\mathbb{F}_{2^{\lambda/5}}$ as high as possible, which seems to be 2 according to a heuristic analysis. This gives an attack that passes the consistency check with probability $2^{-\frac{3}{5}\lambda}$, and can then recover Δ using $q = 5 \cdot 2^{\lambda/5}$ queries to the random oracle, contradicting the stated bound at the end of the proof of KOS's Theorem 1, which would only allow a success probability of $O(q2^{-\lambda}) = O(2^{-\frac{4}{5}\lambda})$.

The importance of $\mathbb{F}_{2^{\lambda/5}}$ being generated by α^5 is that for any $u \in \mathbb{F}_{2^{\lambda/5}}$ and $y \in \mathbb{F}_{2^\lambda}$, we have $u * v \in \mathbb{F}_{2^{\lambda/5}}$. The proof is that for u to be in the subfield the only nonzero entries in it must be at indices of the form $5i + 1$, so that they correspond to powers of α^5. Then $u * v$ will be zero wherever u is zero, and so will also be in the subfield.

[1] There seems to be no advantage for the receiver to ever lie about t.

[2] For any λ divisible by 20, a possible P may be found by picking a element of $\mathbb{F}_{2^{\lambda/5}}$ that is not a perfect power of 5, then setting P to be its minimal polynomial. Such an element must exist because $\left|\mathbb{F}^*_{2^{\lambda/5}}\right| = 2^{\lambda/5} - 1 = 16^{\lambda/20} - 1 \equiv 0 \pmod 5$.

To make use of this fact, let the malicious receiver pick the OT corrections to make x_j^i be 1 when $5 \mid i-j$ and 0 otherwise. Also assume that the malicious receiver sends the honest value of t in its consistency check. Let $\delta_j = (\mathbf{x}_j * \Delta) \cdot \alpha^{1-j} = \sum_{i=0}^{(\lambda/5)-1} \alpha^{5i} \cdot \Delta_{5i+j} \in \mathbb{F}_{2^{\lambda/5}}$ for $1 \le j \le 5$, and let $\vec{\delta} = [\delta_1 \ \cdots \ \delta_5]^\top$. Also let $\vec{\alpha} = [1 \ \alpha \ \cdots \ \alpha^4]^\top$, so that $\vec{\alpha}^\top \vec{\delta} = \Delta$. Let $\chi'_j = \alpha^{j-1} \sum_i \chi_{5i+j}$. Write $x = \sum_{j=1}^5 x'_j \cdot \alpha^{j-1}$ and $\chi'_j = \sum_{i=1}^5 \chi'_{ji} \cdot \alpha^{i-1} = \vec{\alpha}^\top \vec{\chi}'_j$ for $x'_j, \chi'_{ji} \in \mathbb{F}_{2^{\lambda/5}}$ and $\vec{\chi}'_j = [\chi'_{j1} \ \cdots \ \chi'_{j5}]^\top$. The consistency check then becomes

$$\begin{aligned}
0 &= t + q + x \cdot \Delta \\
&= \sum_j (\mathbf{t}_j + \mathbf{q}_j) \cdot \chi_j + x \cdot \Delta \\
&= \sum_j (\mathbf{x}_j * \Delta) \cdot \chi_j + x \cdot \Delta \\
&= \sum_{j=1}^5 \sum_i \delta_j \cdot \alpha^{j-1} \cdot \chi_{5i+j} + \sum_{j=1}^5 x'_j \cdot \alpha^{j-1} \cdot \vec{\alpha}^\top \vec{\delta} \\
&= \sum_{j=1}^5 \delta_j \cdot \chi'_j + \sum_{j=1}^5 x'_j \cdot \sum_{k=1}^5 \alpha^{j+k-2} \cdot \delta_k \\
&= \vec{\alpha}^\top \left(\mathbf{X} + \sum_{j=1}^5 x'_j \cdot \mathbf{A}_j \right) \vec{\delta}, \qquad \text{(C.1)}
\end{aligned}$$

where $\mathbf{X} = [\vec{\chi}'_1 \ \cdots \vec{\chi}'_5]$ and the matrices A_j are:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & \alpha^5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad A_3 = \begin{bmatrix} 0 & 0 & 0 & \alpha^5 & 0 \\ 0 & 0 & 0 & 0 & \alpha^5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 0 & 0 & \alpha^5 & 0 & 0 \\ 0 & 0 & 0 & \alpha^5 & 0 \\ 0 & 0 & 0 & 0 & \alpha^5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad A_5 = \begin{bmatrix} 0 & \alpha^5 & 0 & 0 & 0 \\ 0 & 0 & \alpha^5 & 0 & 0 \\ 0 & 0 & 0 & \alpha^5 & 0 \\ 0 & 0 & 0 & 0 & \alpha^5 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Notice that in Eq. (C.1) both the matrix in parenthesis and the $\vec{\delta}$ on the right are in the subfield $\mathbb{F}_{2^{\lambda/5}}$. Since α generates $\mathbb{F}_{2^\lambda}$, which is a degree 5 extension of $\mathbb{F}_{2^{\lambda/5}}$, the only way for the equation to hold is if

$$\left(\mathbf{X} + \sum_{j=1}^{5} x'_j \cdot \mathbf{A}_j\right)\vec{\delta} = 0.$$

$\vec{\delta}$ will be uniformly random because Δ is, so to maximize the chance of the consistency check succeeding the receiver should minimize the rank of $\mathbf{X} + \sum_{j=1}^{5} x'_j \cdot \mathbf{A}_j$.[3] The question is, how low can it go?

We believe that for most possible $\mathbf{X}$ (which is uniformly random), the minimum rank will be 3. Unfortunately, we only have a heuristic justification for this. The number of rank ≤ 3 matrices over $\mathbb{F}_{2^{\lambda/5}}$ is at least $2^{\frac{21}{5}\lambda}$, because any matrix of the form $\begin{bmatrix}\mathbb{1}_3 \\ Y\end{bmatrix} Z$ has rank ≤ 3 for $Y \in \mathbb{F}_{2^{\lambda/5}}^{2\times 3}$ and $Z \in \mathbb{F}_{2^{\lambda/5}}^{3\times 5}$, and this representation is unique. Therefore, a uniformly random matrix will have rank ≤ 3 with probability at least $2^{\frac{21}{5}\lambda}2^{-\frac{25}{5}\lambda} = 2^{-\frac{4}{5}\lambda}$. When $\mathbf{X}$ and all the x'_j are chosen uniformly at random then $\mathbf{X} + \sum_{j=1}^{5} x'_j \cdot \mathbf{A}_j$ will be uniformly random. The solver gets to choose all of the x'_j for a total of 2^λ possibilities, making the expected number of solutions be at least $2^\lambda 2^{-\frac{4}{5}\lambda} = 2^{\lambda/5}$, so if the rank ≤ 3 matrices are relatively evenly spread over all possible $\mathbf{X}$ then a solution is very likely to exist.

Assuming that a rank 3 solution exists, the receiver should send the corresponding consistency check message x. Then $|S_\Delta| = 2^k$ for $k = \frac{2}{5}\lambda$, yet every $H_{\mathbf{s}}$ has size $\frac{1}{5}\lambda$ because $H_{\mathbf{x}_j} = \{j, 5+j, \cdots, \lambda - 5 + j\}$. This contradicts Lemma 1.

This also gives an attack on the real KOS protocol. The consistency check will succeed with probability $2^{-\frac{3}{5}\lambda}$. If it does and if the receiver gets to see both of the sender's output OT messages for the first five OTs, then it can do a brute force attack to recover Δ. Each message was protected using only $\frac{1}{5}\lambda$ bits of Δ because the Hamming weight of each $\mathbf{x}_j$ is $\frac{1}{5}\lambda$. Using $2^{\frac{1}{5}\lambda}$ queries to the random oracle, these bits can be brute forced to learn a part of Δ. Doing this five times reveals all of Δ, and then the receiver can read every OT message.

[3] See [FLP08] for an algorithm to solve the MinRank problem that runs in $O(\lambda)$ time for constant size matrices. It should be fast in practice for this small problem size.

C.5 Endemic OT Details

In this section, we describe our attack against the OT Extension Protocol With an Ideal Cipher from Endemic OT [MR19, Sect. 5.3]. The flaw results from assuming a stronger guarantee from consistency checking than protocols like KOS or OOS provide. When describing their protocol in Figure 9, they state:

> R proves in zero knowledge that
>
> $$\forall i \in [m], \exists w \in \mathbb{F}_2^{k_C} \mid 0 = b \odot (\mathbf{u}_i + \mathbf{t}_i + \mathbf{t}_{1,i} + \mathcal{C}(w))$$
>
> Note: $\mathbf{b} \in \mathbb{F}_2^{k_C}$ is distributed uniformly in the view of $\mathbf{R}$.
> For example, the proof of KOS for $N = 2$ or OOS otherwise.

This is much stronger that what consistency checking protocols provide. At best, they merely prove that $b \odot (\mathbf{u}_i + \mathbf{t}_i + \mathbf{t}_{1,i} + \mathcal{C}(w))$ was successfully guessed by R.

This opens an avenue for attack. Let R be corrupted, and behave honestly until T_0, T_1 are known. Find the closest pair of rows $\mathbf{t}_i, \mathbf{t}_j$ of T_0 in hamming distance (with $i < j$), so $D = \|\mathbf{t}_i + \mathbf{t}_j\|_0$ is as small as possible.[4] Then set $\mathbf{c}_j = \mathbf{t}_i \oplus \mathbf{t}_j$, and all other $\mathbf{c}_k = 0$. Later, for the consistency check, let $w = 0$ for each OT. The receiver will then have to guess $b \odot (\mathbf{t}_i \oplus \mathbf{t}_j)$; have it guess $\mathbf{t}_i \oplus \mathbf{t}_j$, which is correct with probability 2^{-D}. Then,

$$\mathbf{q}_j = \mathbf{c}_j \cdot \mathbf{b} \oplus \mathbf{t}_j = (\mathbf{t}_i \oplus \mathbf{t}_j) \cdot \mathbf{b} \oplus \mathbf{t}_j = \mathbf{t}_i = \mathbf{c}_i \cdot \mathbf{b} \oplus \mathbf{t}_i = \mathbf{q}_i,$$

so extended OTs i and j will agree on all evaluations, breaking the security of the OT extension.

For each u and v, $\|\mathbf{t}_u + \mathbf{t}_v\|_0$ follows a binomial distribution with parameters $n = n_{\mathcal{C}}$ and $p = \frac{1}{2}$. Given D, our attack succeeds with probability 2^{-D}. In the simplest case of $m = 2$, the success probability is then $\mathbb{E}[2^{-D}] = ((1-p) + p2^{-1})^n = \left(\frac{3}{4}\right)^n \approx 2^{-0.415n}$, which is already considerably higher than is claimed by Endemic OT. For greater m, we estimate D by finding some D_0 such that the event $D \leq D_0$ has constant probability. For any D_0, the event $D \leq D_0$ is a union of the $\binom{m}{2}$ events $\|\mathbf{t}_u + \mathbf{t}_v\|_0 \leq D_0$ for each pair $u < v$. While these events are not independent, we estimate that $\Pr[D \leq D_0]$ will be constant if $\Pr\left[\|\mathbf{t}_u + \mathbf{t}_v\|_0 \leq D_0\right] \geq \binom{m}{2}^{-1}$. We can then choose D_0 with the quantile function of the binomial distribution, and estimate

[4] This problem is important in the decoding of linear codes. See [MO15] for an efficient algorithm.

that the success probability is around 2^{-D_0}. We evaluated this for $n = 128$ and several choices of m:

m	D_0
10^6	27
10^7	24
10^8	21
10^9	18

C.6 Extra Proofs

C.6.1 Universal Hash Proofs

Proposition 4.2.3. *Let $\mathcal{R}$ and $\mathcal{R}'$ be ϵ and ϵ'-almost universal families, respectively. Then $R'R$ for $R \in \mathcal{R}, R' \in \mathcal{R}'$ is a $(\epsilon + \epsilon')$-universal family.*

Proof. Let $\vec{x} \in \mathbb{F}_q^n$ be nonzero. Except with probability ϵ, $R\vec{x} \neq 0$. If it is nonzero, then with probability at least $1 - \epsilon'$, $R'R\vec{x} \neq 0$. Therefore $R'R\vec{x} \neq 0$ with probability at least $(1 - \epsilon)(1 - \epsilon') \geq 1 - \epsilon - \epsilon'$, so $R'R$ is $(\epsilon + \epsilon')$-universal. □

Proposition 4.2.4. *Let $\mathcal{R} \subseteq \mathbb{F}_q^{m \times n}$ and $\mathcal{R}' \subseteq \mathbb{F}_q^{m \times n'}$ be ϵ-almost uniform families. Then $[R\ R']$ for $R \in \mathcal{R}, R' \in \mathcal{R}'$ is a ϵ-uniform family.*

Proof. Let $\vec{x} = \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$ be nonzero, for $\vec{x}_0 \in \mathbb{F}_q^n, \vec{x}_1 \in \mathbb{F}_q^{n'}$. Without loss of generality, assume that $\vec{x}_0$ is nonzero. For $[R\ R']\vec{x}$ to equal $\vec{y}$, we must have $R\vec{x}_0 = \vec{y} + R'\vec{x}_1$. Because R is independent of $\vec{y}$ and R', this has probability at most ϵ by Theorem 4.2.2. □

C.6.2 OT Extension Proof

Theorem 4.5.1. *The protocol in Fig. 4.11 achieves $\mathcal{F}_{\mathsf{OT}\text{-}1}^{p^k c, \ell, \{X\}}$ with malicious security in the $\mathcal{F}_{\mathsf{VOLE}\text{-}\mathsf{pre}}^{p,q,\mathcal{C},\ell,\mathcal{L},M}$ hybrid model, assuming that $H \colon \mathbb{F}_q^{nc} \times \mathcal{T} \to \{0,1\}^\lambda$ is a $(p, q, \mathcal{C}, \mathcal{T}, \mathcal{L})$-TCR hash, and $\mathcal{R} \subseteq \mathbb{F}_q^{nc \times \lceil \log_q(\ell) \rceil}$ is an ϵ-almost uniform family. The distinguisher advantage is at most $\epsilon M \ell (t_{max} - 1)/2 + \mathsf{Adv}_{\mathrm{TCR}}$, where t_{max} is the maximum number of distict OTs that can have the same tweak under t. For the TCR itself, τ_{max} will be the maximum number of evaluations $F_i(\bar{x})$ where $t(i, \bar{x})$ outputs a given tweak. For semi-honest security, $\mathcal{R}$ is unused;*

instead set $\epsilon = q^{-n_C}$ *and* $M = 1$.

Proof. First, we establish correctness, which will be used in most cases of the security proof. Since the base VOLE gives a correlation $W = UG_{\mathcal{C}} \operatorname{diag}(\vec{\Delta}) + V$,

$$F_i(\vec{x}_i^*) = H(W_{i\cdot} + \vec{r}_i^{\top} - \vec{x} G_{\mathcal{C}} \odot \vec{\Delta}, t(i, \vec{x})) = H(V_{i\cdot} + \vec{r}_i^{\top}, t(i, \vec{x}_i^*)) = F_i^*.$$

Starting with the easiest case, if both parties are corrupted then the simulator can trivially program the whole output, which will be consistent by correctness.

Corrupt P_S. At the start of the protocol, the simulator sends "commit" to P_S. For malicious security, it receives R from P_S, while for semi-honest security it generates it randomly and puts it in the transcript instead. The simulator receives $\vec{\Delta}, W$ from $\mathcal{A}$ and forwards them to P_S. It then sets $F_i(\vec{x}) = H(W_{i\cdot} + \vec{r}_i^{\top} - \vec{x} G_{\mathcal{C}} \odot \vec{\Delta}, t(i, \vec{x}))$ and sends it to $\mathcal{F}_{\mathsf{OT\text{-}1}}^{p^{k_C}, \ell, \{X\}}$, for all $i \in [\ell]$. For malicious security, it sends X to the ideal functionality as well. By correctness this works identically to the real protocol.

Corrupt P_R. This is the first case where the TCR's security is used. For malicious security, $\mathcal{S}$ first receives $\mathcal{W}_{\text{pre}}$, U_{pre}, V_{pre}, and L_{pre} from $\mathcal{A}$, then samples $R \xleftarrow{\$} \mathcal{R}$ and sends it to P_R. In either case, the simulator then receives U, V from $\mathcal{A}$ and forwards them to P_R. For malicious security, $\mathcal{S}$ then receives $w_{\text{pre}} \in \mathcal{W}_{\text{pre}}, \vec{L}_{\text{off}} \in \mathbb{F}_q^{n_C}$ from $\mathcal{A}$, generates a uniformly random $\vec{\Delta} \xleftarrow{\$} \mathbb{F}_q^{n_C}$, and aborts the protocol if the consistency check in $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}^{p,q,\mathcal{C},\ell,\mathcal{L},M}$ would fail. Finally, it sends $x_i^* = U_{i\cdot}$ and $F_i^* = H(V_{i\cdot} + \vec{r}_i^{\top}, t(i, \vec{x}_i^*))$ to the ideal functionality, for $i \in [\ell]$.

Next, we prove that the real world, where the real protocol is run in the $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}^{p,q,\mathcal{C},\ell,\mathcal{L}}$-hybrid model, is indistinguishable from the ideal world, where the simulator is given access to the ideal functionality $\mathcal{F}_{\mathsf{OT\text{-}1}}^{p^{k_C}, \ell, \{X\}}$. First, there is a bad event that we must show is unlikely. We must show that there are no two distinct OT indices $i < j$ satisfying $V_{i\cdot} + \vec{r}_i^{\top} = V_{j\cdot} + \vec{r}_j^{\top}$ that have overlapping tweaks, meaning that there exists $\vec{x}_i$ and $\vec{x}_j$ such that $t(i, \vec{x}_i) = t(j, \vec{x}_j)$. We start with the malicious case. For any such pair, $V_{i\cdot} - V_{j\cdot}^{\top} = R(\vec{j} - \vec{i})$, so if V were independent of R then this would have probability at most ϵ because $\mathcal{R}$ is a uniform hash family. Although V is allowed to depend on R, it must equal $V_{\text{pre}}(w_{\text{pre}}, \vec{\Delta})$, and w_{pre} can only be chosen from M options. There are at most $\ell(t_{\max} - 1)/2$ possible pairs of indices i, j with overlapping tweaks, so by a union bound the probability of the bad event is at most $\epsilon M \ell (t_{\max} - 1)/2$.

However, in the semi-honest case $\vec{r}_i = 0$ for all i. If V were uniformly random then we

could instead use that $V_i. = V_j.$ with probability only $q^{-n_\mathcal{C}}$. But V isn't uniformly random because P_R (who is the sender for the VOLE) is corrupted — it's chosen by the adversary. The trick is to prove security in a slightly different hybrid model, where the VOLE is required to output a V such that $V_i. \neq V_j.$ when $i \neq j$ have overlapping tweaks. Any semi-honest protocol that achieves $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}$ based on functionalities which do not explicitly depend on who is corrupt (such as a communication channel) also achieves the modified functionality. That is, in semi-honest security, corruption does not give the adversary and environment any new power other than to see more information, and the honest–honest case (where V is guaranteed to be uniformly random) still gives enough information to see whether V has distinct rows. Therefore, if when P_S is corrupt the VOLE outputs a V with repeated rows more often than random, then there is an attack against the honest–honest case of the VOLE.

Next, we present the hybrids.

1. Start from the real world, then rewrite the usage of H and $\vec{\Delta}$ into oracle calls to $\mathsf{TCR\text{-}real}^{H,p,q,\mathcal{C},\mathcal{L}}$. That is, use $\textsc{leak}(L_{\text{pre}}(w_{\text{pre}}) - \vec{L}_{\text{off}})$ to implement the selective abort, instead of checking $\vec{\Delta} + \vec{L}_{\text{off}} \in L$ directly, and change $F_i(\vec{x})$ to be computed as $\textsc{query}(\vec{x}_i^* - \vec{x}, V_i. + \vec{r}_i^\top, t(i, \vec{x}))$ where $\vec{x}_i^* = U_i.$. This is the same because

$$H(W_i. + \vec{r}_i^\top - \vec{x}G_\mathcal{C} \odot \vec{\Delta}, t(i, \vec{x})) = H((U_i. - \vec{x})G_\mathcal{C} \odot \vec{\Delta} + V_i. + \vec{r}_i^\top, t(i, \vec{x}_i^*)),$$

   by the correctness of the VOLE. This is just refactoring the computation, and so results in no observable difference for the environment.

2. Use TCR security to swap $\mathsf{TCR\text{-}real}^{H,p,q,\mathcal{C},\mathcal{L}}$ for $\mathsf{TCR\text{-}ideal}^{H,p,q,\mathcal{C},\mathcal{L}}$. This is allowed because the calls to $\textsc{query}$ are distinct, as otherwise the bad event would trigger.

3. Inline the oracles calls to $\mathsf{TCR\text{-}ideal}^{H,p,q,\mathcal{C},\mathcal{L}}$. Notice that $\vec{\Delta}$ is only used for the selective abort attack, as in the simulator. Call the process of sampling F_i and outputting it to P_S the ideal functionality, and call the rest of this hybrid the simulator. We are now at the ideal world.

Both Honest. This case has the simplest simulator. The simulator only needs to sample $R \xleftarrow{\$} \mathcal{R}$ and allow $\mathcal{A}$ to eavesdrop on it. We again use a hybrid proof, starting from the real world.

1. Let $\vec{c} \in \mathcal{C}$ be a non-zero code word. Instead of sampling $W \xleftarrow{\$} \mathbb{F}_q^{\ell \times n_\mathcal{C}}$, sample $sk \xleftarrow{\$} \mathbb{F}_p^{k_\mathcal{C}}$

and $W' \xleftarrow{\$} \mathbb{F}_q^{\ell \times n_C}$, and set $W_{i\cdot} = W'_{i\cdot} + \vec{c} \odot sk$ for all $i \in [\ell]$.

2. We now need to show that the functions $F_i^* = H(W_{i\cdot} + \vec{c} \odot sk + \vec{r}_i^\top - \vec{x} G_{\mathcal{C}} \odot \vec{\Delta}, t(i, \vec{x}))$ are uniformly random. They can be written in terms of QUERY from $\mathsf{TCR\text{-}real}^{H,p,q,\mathcal{C},\mathcal{L}}$, using sk instead of $\vec{\Delta}$ as the TCR key. There will be no duplicate queries, for the same reason as in the case of corrupt P_R. By the security of the TCR, all queries will be uniformly random.

3. Forget about $V, W, \vec{\Delta}$, which are now unused. We are now at the ideal world.

□

C.6.3 Δ-OT Extension Proof

Theorem 4.5.2. *The protocol in Fig. 4.12 achieves* $\mathcal{F}_{\mathsf{VOLE}}^{p,p,\mathsf{Rep}(\mathbb{F}_p^n),\ell,\{X\}}$ *with malicious security in the* $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}^{p,p,\mathsf{Rep}(\mathbb{F}_p^{n'}),\ell,\mathrm{Affine}(\mathbb{F}_p^{n'}),M}$ *hybrid model, assuming that* $\mathcal{R} \subseteq \mathbb{F}_p^{n' \times n}$ *is a* ϵ*-almost uniform family and* $n' \geq n$*. The advantage is bounded by* $\epsilon M(p^n - 1)$*.*

Proof. First, we need to show correctness, which is used for all cases of the proof.

$$WR = UG_{\mathsf{Rep}(\mathbb{F}_p^{n'})} \operatorname{diag}(\vec{\Delta}) R + VR = U\vec{\Delta}R + VR$$

The two cases where P_R is corrupt are trivial, as the simulator can program the protocol output to be $\vec{\Delta}R, WR$, and U is unchanged. When P_R is honest, we need to use the following lemma, which shows that the entropy in $\vec{\Delta}$ is enough to make $\vec{\Delta}R$ uniform.

Lemma C.6.1. *Let* $\mathcal{R} \subseteq \mathbb{F}_p^{n' \times n}$ *be an* ϵ*-almost uniform family and let* $A \in \mathbb{F}_p^{m \times n'}$ *be full rank, where* $m \leq n'$*. Then*

$$\Pr_{R \xleftarrow{\$} \mathcal{R}}[\operatorname{rank}(AR) < n] \leq \epsilon p^{n'-m}(p^n - 1).$$

Proof. Let $X = |\ker(AR) \setminus \{0\}| = |\ker(AR)| - 1$. The statement that $\operatorname{rank}(AR) < n$ is equivalent to $X \geq 1$. Any nonzero $\vec{x} \in \mathbb{F}_p^n$ has probability at most $\epsilon p^{n'-m}$ of being in $\ker(AR)$, because that implies $R\vec{x} \in \ker(A)$, which has size $|\ker(A)| = p^{n'-m}$ by the rank–nullity theorem. Therefore $\mathbb{E}[X] \leq \epsilon p^{n'-m}(p^n - 1)$, and the lemma follows by Markov's inequality. □

If both parties are honest then all that's needed is for R to be full rank, as then all it does is throw away part of the VOLE output. This is true except with probability $\epsilon(p^n - 1)$ by Theorem C.6.1 with $A = \mathbb{1}_{n'}$. The only interesting case is when only P_S is corrupt. The simulator first receives $\mathcal{W}_{\text{pre}}$, U_{pre}, V_{pre}, and L_{pre} from $\mathcal{A}$, then samples $R \xleftarrow{\$} \mathcal{R}$ and sends it to P_S. The simulator receives U, V from $\mathcal{A}$ and forwards them to P_S, and then receives $w_{\text{pre}} \in \mathcal{W}_{\text{pre}}, \vec{L}_{\text{off}} \in \mathbb{F}_p^{n'}$ from $\mathcal{A}$. The simulator generates a uniformly random $\vec{\Delta} \xleftarrow{\$} \mathbb{F}_p^{n'}$, and aborts the protocol if the guess in $\mathcal{F}_{\mathsf{VOLE\text{-}pre}}^{p,p,\mathsf{Rep}(\mathbb{F}_p^{n'}),\ell,\mathcal{L},M}$ would fail. Finally, it sends U, VR to the ideal functionality.

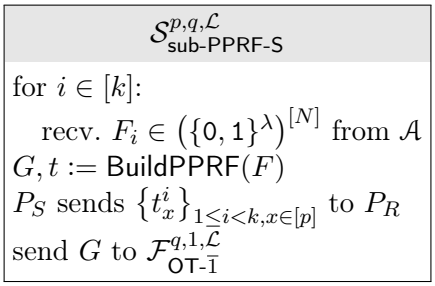
For security, we first define a bad event, and bound its probability. Let $L = L_{\text{pre}}(w_{\text{pre}}) - \vec{L}_{\text{off}}$, so the protocol aborts if $\vec{\Delta} \notin L$. Because $L \in \text{Affine}(\mathbb{F}_p^{n'})$, there exists a vector $\vec{\Delta}_0 \in \mathbb{F}_p^{n'}$ and a full rank matrix $A \in \mathbb{F}_p^{m \times n'}$ such that $L = \vec{\Delta}_0 + \text{rowspace}(A)$, where $m = \dim(L)$. Also, A is independent of $\vec{L}_{\text{off}}$, because $\vec{L}_{\text{off}}$ only shifts L. The bad event is that $\vec{\Delta} \in L$ and $\text{rank}(AR) < n$. The former has probability at most $p^{m-n'}$, since $\vec{\Delta}$ was sampled uniformly and $|L| = p^m$. If A were independent of R, the latter probability would be at most $\epsilon p^{n'-m}(p^n - 1)$ by Theorem C.6.1. Though $\mathcal{A}$ sees R before choosing L, we can still use a union bound. L_{pre} is chosen independently of R, and there are at most M possibilities for $L_{\text{pre}}(w_{\text{pre}})$, so the bad event has probability at most $\epsilon M(p^n - 1)$.

Next, the hybrid proof goes from the real world to the ideal world. The only information P_S learns about $\vec{\Delta}$ is that $\vec{\Delta} \in L$, so it is equivalent to sample a second $\Delta' \in L$ after the check, let $\Delta'' = \Delta' R$, and subsequently use Δ'' instead of ΔR. Since we are assuming the bat event does not trigger, $\text{rank}(AR)$ has full rank, so it's also equivalent to sample $\Delta'' \xleftarrow{\$} \mathbb{F}_p^n$. Split into the simulator and the ideal functionality, with Δ'' being sampled in the ideal functionality. We are now at the ideal world. □

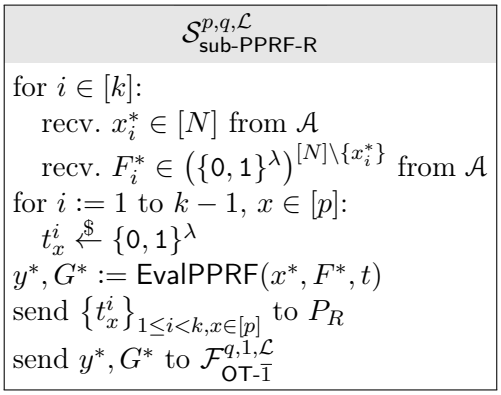
C.6.4 Semi-honest PPRF Proof

Theorem 4.6.1. *Figure 4.13 constructs $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{q,1,\{X\}}$ out of $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{p,k,\{X\}}$, and is secure in the semi-honest model.*

Proof. First, we show correctness, as it is useful for all cases of the proof. We prove by induction that $s_y^i = s_y^{*\,i}$ for all $y \in [p^i] \setminus \{y_i^*\}$. In the base case, $F_0(x) = s_x^1 = s_x^{*\,1} = F_0^*(x)$ by correctness of the first $\binom{p}{p-1}$-OT. For induction, P_S and P_R both compute s_{py+x}^{i+1} in exactly



(a) Corrupt P_S.



(b) Corrupt P_R.

Figure C.1: Simulators for semi-honest security of Fig. 4.13. In (a), P_S sending t to P_R means that the adversary is allowed to eavesdrop on a fake message from P_S to P_R.

the same way for all $y \neq y^*_i$. Then for any $x \neq x^*_i$,

$$\begin{aligned} s^{*\,i+1}_{py^*_i+x} &= t^i_x \oplus F^*_i(x) \oplus \bigoplus_{y\in[p^i]\setminus\{y^*_i\}} s^{*\,i+1}_{py+x} \\ &= F_i(x) \oplus \bigoplus_{y\in[p^i]} s^{i+1}_{py+x} \oplus F_i(x) \oplus \bigoplus_{y\in[p^i]\setminus\{y^*_i\}} s^{i+1}_{py+x} \\ &= s^{i+1}_{py^*_i+x}. \end{aligned}$$

Therefore, $s^{i+1}_y = s^{*\,i+1}_y$ for all $y \neq y^*_{i+1} = py^*_i + x^*_i$.

When both parties are honest, the simulator can generate the t^i_x uniformly at random and give them to the adversary as a fake eavesdropped message. In the hybrid proof, starting from the real world, first replace the random sampling of all $F_i(x)$ with instead sampling $t^i_x \xleftarrow{\$} \{0,1\}^\lambda$, then setting $F_i(x) = t^i_x \oplus \bigoplus_{y\in[p^i]} s^{i+1}_{py+x}$. These are the same distribution. Next, use correctness to replace $s^{*\,i}_y$ with s^i_y everywhere, so that the $F_i(x)$ are all unused and can be removed. Now the internal leaves of the GGM tree are unused, which makes the s^k_y be all the evaluations of a GGM PRF [GGM86]. Therefore, we can replace them all with uniform randomness. Finally, since the $x^*_i \in [p]$ are all uniformly random, it is equivalent to sample $y \xleftarrow{\$} \left[p^k\right]$, then let $x^*_0, \ldots, x^*_{i-1}$ be its expansion in base p, in big endian order. We are now at the ideal world.

Next, assume that only P_S is corrupt. The simulator for this case is illustrated in Fig. C.1a. By correctness, the output from P_R will be G^*, the punctured version of the G computed by a simulator. Also, y will be uniformly random for the same reason as in the honest–honest case. Therefore, the real protocol will be indistinguishable from the simulation.

Finally, we have the case where P_R is corrupt (simulator in Fig. C.1b). Going from the real world to the ideal world, we use the following hybrids.

1. In a sequence of hybrids, replace all the t_x^i and $s_{y_i^*}^i$ with uniformly random values. For $i = 1$, $s_{y_1^*}^1 = F_0(x_)^*$ is already sampled uniformly at random by the ideal functionality $\mathcal{F}_{\mathsf{OT}\text{-}\bar{1}}^{p,k,\{X\}}$. For $i > 1$, first use that $s_{y_{i-1}^*}^{i-1}$ is uniformly random to sample $s_{py_{i-1}^*+x}^i = \mathsf{PRG}_x(s_{y_{i-1}^*}^{i-1})$ uniformly at random for all $x \in [p]$. Then instead of sampling the $s_{py_{i-1}^*+x}^i$ at random, sample $t_x^i \xleftarrow{\$} \{0,1\}^\lambda$, then compute $s_{py_{i-1}^*+x}^i$ as in $\mathsf{EvalPPRF}$, for $x \neq x_{i-1}^*$. Finally, $t_{x^*}^{i-1}$ can be also be sampled randomly, because the underlying ideal functionality samples $F_{i-1}(x^*) \xleftarrow{\$} \{0,1\}^\lambda$. Continue this sequence of hybrids until $i = k$ to get the desired modifications.

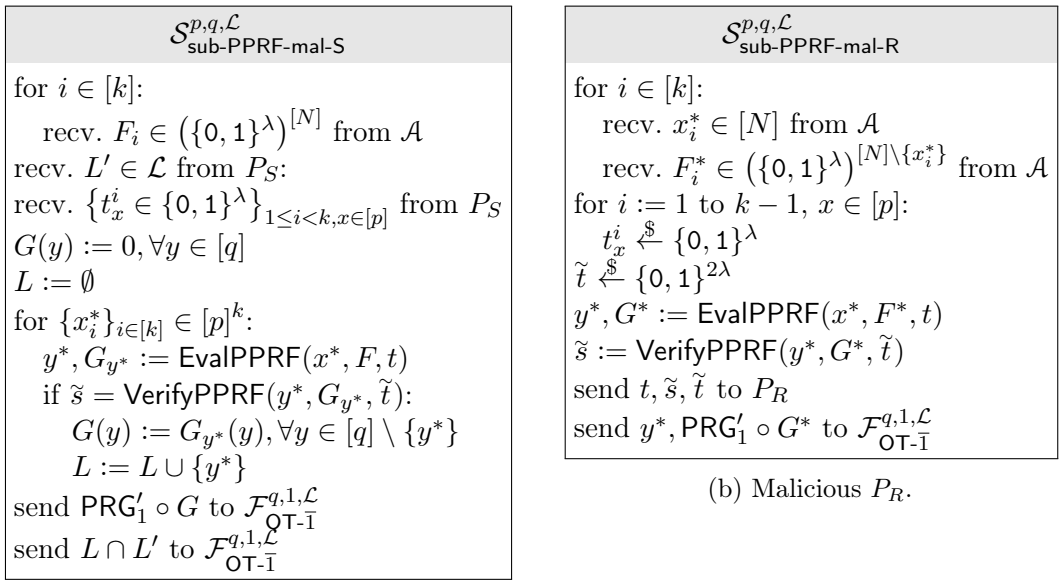
2. Notice that G^*, the restriction of P_S's output to $x \neq x^*$ is now computed in exactly the same way as $\mathsf{EvalPPRF}$, and that $G(y^*) = s_{y^*}^k$ is uniformly random. Put the computation of the former in the simulator and the latter in the desired ideal functionality $\mathcal{F}_{\mathsf{OT}\text{-}\bar{1}}^{q,1,\{X\}}$. Also notice that the t_x^i are all sampled uniformly at random and put them in the simulator. This is exactly the same as the ideal world, where the simulator talks to the ideal functionality and generates a fake transcript of the protocol.

□

C.6.5 Maliciously Secure PPRF Proofs

Proposition 4.6.2. *The selective abort attack allowed in Fig. 4.14 will always be in* $\mathcal{L} = \text{Affine}(\mathbb{F}_p^k)$. *More precisely, the* L *sent by* $\mathcal{S}_{\mathsf{sub\text{-}PPRF\text{-}mal\text{-}S}}^{p,q,\mathcal{L}}$ *(Fig. C.2) will always be in* $\text{Affine}(\mathbb{F}_p^k)$.

Proof. This is trivial if $L = \{\}$, so assume that L is not empty. The simulator will then find a preimage for $\tilde{s} = \mathsf{Hash}(\tilde{s}_0 \parallel \cdots \parallel \tilde{s}_{q-1})$. By collision resistance, every time the simulator calls $\mathsf{VerifyPPRF}(y^*, G^*, \tilde{t})$ for $y \in L$, it finds the same $\tilde{s}_y$. Next, assume that L contains at least two elements z and z', because any L containing only a single element is trivially in



(a) Malicious P_S.

(b) Malicious P_R.

Figure C.2: Simulators for malicious security of Fig. 4.14.

X. Then collision resistance of PRG'_0 implies that $G_z(y) = G_{z'}(y)$ for all $y \notin \{z, z'\}$. That is, every $G_{y^*}(y)$ will agree with $G(y)$ on every y they can compute. Use this to prove the following lemma.

Lemma C.6.2. *Let $z, z' \in L$, let $z_1, \ldots, z_k$ and $z'_1, \ldots, z'_k$ be their active paths and $w_0, \ldots, w_{k-1}$ and $w'_0, \ldots w'_{k-1}$ be their base OT choices. Let j be the first index where they differ, so that $z_i \neq z'_i$. Then for any $y^* \in [q]$ with base OT choice bits $x^*_0, \ldots, x^*_{k-1}$ and active path $y^*_1, \ldots, y^*_k$, if $x^*_i = w_i$ except when $i = j$, we have $y^* \in L$.*

Proof. The collision resistance of PRG implies that all $s^{*\,i}_y$ computed by $\mathsf{EvalPPRF}(x^*, F, t)$ that outputs either z or z' will agree. They both miss the nodes of the GGM tree on their common path $z_1, \ldots, z_{j-1}$, but every other node in the tree can be computed by at least one of them. Let s^i_y be the seeds in the GGM tree that at least one of them may compute. This implies a correctness property for the t^i_x when $i \geq j-1$: any t^i_x used by either z or z' during evaluation of the PPRF must take its correct value of $F_i(x) \oplus \bigoplus_{y\in[p^i]} s^{i+1}_{py+x}$. Otherwise z (or z') would reconstruct a different $s^{i+1}_{pz_i+x}$ during evaluation, which would not pass the

consistency check.

Since the active path of y^* agrees with z on its first $j-1$ nodes, it will find the same seeds s_y^i for $i < j$ and $y \neq y_i^*$. Additionally, y^* agrees with z after its first j nodes, so it only uses correct t_x^i for $i \geq j$. This only leaves the corrections t_x^{j-1} for the node on layer j. Because z and z' disagree on layer j, so $w_{j-1} \neq w'_{j-1'}$, every t_x^{j-1} must be correct for this layer. Therefore, the evaluation on y^* will get exactly the same seeds s_y^i for any $y \neq y_i^*$, and so G_{y^*} must agree with G_z and $G_{z'}$. Finally, $\widetilde{t}$ must be correct for there to be two evaluations z and z' that pass the consistency check, so the evaluation for y^* will correctly find all the $\widetilde{s}_y$, and so $y^* \in L$. □

In each position $i \in [k]$, either all $y^* \in L$ will have the same x_i^*, or there will be at least two different possible x_i^*. Let $L' \supseteq L$ be the set of all y^* that match with the x_i^* in the positions where all of L are the same. This allows $y^* \in L'$ to take any value at the positions where at least two x_i^* differ. These are affine constraints so $L' \in \text{Affine}(\mathbb{F}_p^k)$. We need to prove that $L' = L$.

Let $z \in L$ and $y^* \in L'$, where $z \neq y^*$. Then the first place where they differ must be a position that L' does not constrain, so there must be some $z' \in L$ that disagrees at this position. By Theorem C.6.2, there must be some $z'' \in L$ that is identical to z except at this position, where it agrees with y^* instead. Repeating this process eventually finds an element of L that is exactly the same as y^*. Therefore, $L' = L$. □

Theorem 4.6.3. *Figure 4.14 (composed with Fig. 4.13) is a maliciously secure* $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{q,1,\text{Affine}(\mathbb{F}_p^k)}$ *in the* $\mathcal{F}_{\mathsf{OT}\text{-}\overline{1}}^{p,k,\text{Affine}(\mathbb{F}_p^k)}$ *hybrid model.*

Proof. If both parties are honest then this is essentially the same as for the semi-honest case. The consistency check messages $\widetilde{s}, \widetilde{t}$ can be simulated as a hash of uniformly random values, and a uniformly random value in $\{0,1\}^{2\lambda}$. By the security of PRG', the OT outputs $\mathsf{PRG}'_1 \circ G$ will be indistinguishable from uniformly random, as will the values $\widetilde{s}_y$.

Malicious P_S. The simulator for this case is shown in Fig. C.2a. The collision resistance of Hash and PRG'_0 implies that $G_{y^*}(y) = G(y)$ for all $y \neq y^*$, when $y^* \in L$. Therefore, when the desired ideal functionality computes P_R's output, it will match what P_R would output in the real protocol, assuming that $y^* \in L$. When $y^* \notin L$, P_R never gets to see the output, so this is equivalent. Finally, Theorem 4.6.2 implies that $L \in \text{Affine}(\mathbb{F}_p^k)$, and the adversary must always provide a $L' \in \text{Affine}(\mathbb{F}_p^k)$, so $L \cap L' \in \text{Affine}(\mathbb{F}_p^k)$ because it is closed under

intersection. Therefore, the real protocol is indistinguishable from the simulated protocol using the desired ideal functionality.

Malicious P_R. Because P_R never sends any messages, malicious security is essentially the same as semi-honest security for this case. The only difference from the semi-honest protocol is the consistency check messages $\widetilde{s}, \widetilde{t}$. By the security of PRG', these will be indistinguishable from being generated from uniformly random values $\widetilde{s}_y$. Also, the OT outputs $\mathsf{PRG}'_1 \circ G$ will be indistinguishable from uniformly random. Therefore, the protocol is secure in this case. □