Cryptography against Space-Bounded Adversaries

Jiaxin Guan

a dissertation
presented to the faculty
of princeton university
in candidacy for the degree
of Doctor of Philosophy

recommended for acceptance
by the Department of
Computer Science

Adviser: Mark Zhandry

September 2023

© Copyright by Jiaxin Guan, 2023. All rights reserved.

Abstract

Traditionally in cryptography, we consider adversaries that are time-bounded by making certain computational assumptions. In this thesis, I study the scenario where the adversaries are space-bounded, i.e. the adversary can only use up to a certain amount of memory bits. Under these scenarios, we can achieve either unconditional security properties or never-before-possible results.

First, I start off with Maurer's Bounded Storage Model. It is a model where the adversary abides by a certain memory bound throughout the entire attack. Under this model, I show simple constructions of a key-agreement protocol, a commitment scheme, and an oblivious transfer protocol, all based on Raz's lower bound on parity learning. These constructions have several advantages over prior work, including enhanced correctness and an improved and optimal number of rounds.

Subsequently, I show that if we combine computational assumptions with the bounded storage model, we can achieve results that are not possible in the standard model. I define a new object named Online Obfuscation, which is analogous to a Virtual Grey-Box Obfuscation in the Bounded Storage Model, and show how to use it to construct disappearing encryption and signature schemes where the ciphertext and the signature effectively "disappear" after transmission.

Lastly, I make the observation that in the Bounded Storage Model, the memory bound on the adversary is enforced throughout the entire game. One can imagine a variant where the bound is only enforced for long-term storage, allowing the adversary to use an arbitrary amount of memory during the transmission phase. I define incompressible cryptography to capture this intuition and show constructions using randomness extractors and other cryptographic tools. Furthermore, I show that under the multi-user setting, we can still achieve desired incompressible security if we simply replace the randomness extractor with a special "multi-instance randomness extractor".

Contents

Listing of figures

Dedicated to my parents, Hong Guan and Meihong Tian.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Mark Zhandry. Mark has been an inspiring mentor, an insightful collaborator, and an incredible friend throughout my PhD journey. When I embarked on my PhD, I was merely wandering into the realm of cryptography without any set directions. I am truly thankful that Mark brought up the direction of space-bounded adversaries, which has now become the focus of this thesis. Although it was not initially one of Mark's more "established" research directions, he enthusiastically embraced this new path and explored the area together with me for the past six years.

One of the highlight memories from my PhD is this one snowy afternoon when Mark and I sat in his office, staring at the snow falling onto the gorgeous Princeton campus while contemplating a challenging problem in the Bounded Storage Model. But there have been many more afternoons like this. And it is through these countless afternoons that Mark guided me from a bewildered PhD student to a more independent researcher. Mark's guidance and support have been invaluable throughout this process. And of course, on a more practical note,

I am grateful for the financial support I received from Mark's NSF grant, which enabled me to attend various conferences and engage with fellow researchers.

I am also grateful to Princeton's Computer Science Department, which provided me with spacious offices and decent whiteboards. I enjoyed the theory lunches and seminars organized by the theory group, not just for their great catering choices, but more for all the insights they've brought to my own research. I want to extend my special thanks to Mark Braverman, Gillat Kol, Ran Raz, Matt Weinberg, and Mark Zhandry for taking their precious time to be on my committee. I also deeply appreciate the administrative support from Nicki Mahler and Mitra Kelly, whose professionalism makes a lot of the complicated tasks feel like a breeze.

I would also like to express my appreciation to all my collaborators, both on published works and ongoing research projects: James Bartusek, Dan Boneh, Pratish Datta, Alexis Korb, Fermi Ma, Hart Montgomery, Amit Sahai, Daniel Wichs, and Mark Zhandry. Working with them has been a truly delightful experience, and I highly value their insights into various cryptographic problems.

Furthermore, I extend my thanks to Xinyi Xu, Qipeng Liu, Wei Zhan, Yuping Luo, Cong Qiaoben, Bin Liu, Chang Tian, Yi Zhao, and Changshuo Liu for their companionship on this journey. During times when I felt down, stressed, or simply worried about COVID-19, their presence and conversations always managed to uplift and carry me through the hardships.

Lastly, I am deeply grateful to my parents, Hong Guan and Meihong Tian, for their unconditional love and unwavering support. They have always respected my choices and provided me with the maximum level of support possible. Without them, it would be impossible

for me to reach where I am today. Therefore, I dedicate this thesis to them, who, from my point of view, are the best parents one can possibly ask for.

1

Introduction

Time and space are the two most basic types of computational resources. *Computation time* dictates the number of steps needed to solve a problem, and *memory space* determines the amount of memory storage required in this process. In cryptography, however, when we consider adversaries against cryptographic protocols, the default is to assume only *time-bounded* adversaries, usually Probabilistic Polynomial Time (PPT) adversaries to be exact. A natural question arises here:

What happens if we consider space-bounded adversaries for cryptographic protocols?

One useful tool to study this exact question is the Bounded Storage Model proposed by Maurer[77]. In this thesis, we begin by presenting some new constructions in the Bounded Storage Model and then explore alternative models to better capture space-bounded adversaries and give concrete constructions for cryptographic protocols under these new models.

1.1 Our Contributions

1.1.1 Simple Schemes in the Bounded Storage Model

Maurer introduced the Bounded Storage Model[77] in 1992, where instead of posing constraints on the adversary's computation time, we restrict the memory space that the adversary can utilize to carry out the attacks. Amazingly, as shown by Maurer[77] and a series of following works[27,26,76,7,39,40], the Bounded Storage Model has proven to be an incredibly useful tool to obtain *unconditional* security proofs.

Typically, we prove the security of cryptographic protocols *conditioned* on certain computational problems being hard, e.g. factoring large integers or Learning with Errors (LWE). We define security in terms of security games for PPT adversaries – if the given protocol is

secure, then the adversary should be able to win the security game with only negligible probability. And then through *reduction proofs*, we show that if there exists a PPT adversary that can win the security game, then it can be efficiently converted to an adversary that solves these computational problems conjectured to be hard. Unfortunately, with the current state of complexity theory, the hardness of these problems can at best be *conjectured*. It turns out that some of these hardness assumptions, such as inverting a hash function or factoring, *will be broken* with the presence of a quantum computer by running Grover's algorithm [59] or Shor's algorithm [92]. With *unconditional* security proofs, the security of cryptographic schemes can be proven *without* relying on any hardness assumptions. This is possible in the Bounded Storage Model by having the honest parties exchange a large amount of information while the adversary has a limited amount of storage to write down only a tiny portion of such communicated information. Then using information-theoretic arguments, we elevate the adversary's lack of knowledge to the security of the scheme.

In the first part of this thesis, we present simple new constructions that are *unconditionally secure* by utilizing the Bounded Storage model. Concretely, we develop new constructions of two-party key agreement, bit commitment, and oblivious transfer in the Bounded Storage Model. In addition to simplicity, our constructions have several advantages over prior work, including an improved and optimal number of rounds and enhanced correctness. Contrary to prior works in the Bounded Storage Model [77,27,26,76,7,39,40], which typically use an argument akin to the birthday paradox, our schemes are based on Raz's lower bound for learning parities [86].

1.1.2 Disappearing Cryptography in the Bounded Storage Model

While the Bounded Storage Model has proven to be a useful tool to study space-bounded adversaries, we find that it has certain limitations too. The first observation we make is that in the Bounded Storage Model, the space constraints are enforced for the adversary *instead of* time constraints, whereas in real life, *both* computation time and memory space pose as restrictions to algorithms. So in the second part of this thesis, we address the following question:

What security notions can we achieve if we consider adversaries that are both time-bounded and space-bounded?

It turns out that by combining computational assumptions with space constraints, we can achieve security notions that are *never-before-possible*. In the second part of this thesis, we propose the notion of *disappearing cryptography*, where a component of the transmission, say a ciphertext, a digital signature, or even a program, is streamed bit by bit. The stream is so large for anyone to store in its entirety, meaning the transmission effectively "disappears" once the stream stops, while the honest parties can run encryption/decryption/signing/verification as online algorithms during the streaming. This allows for new security notions that are unachievable in the standard model. For instance, in the case of disappearing ciphertexts, we can achieve security even if the adversary is handed the *private key* after the streaming of the challenge ciphertext concludes. This is *impossible* within the standard model, as the adversary can trivially use the acquired private key to decrypt the challenge ciphertext, and hence distinguish between the challenge messages. However, with disappearing ciphertexts, even though the adversary has the private key, there is nothing for the adversary to decrypt, as the

ciphertexts have disappeared after transmission!

In this part of the thesis, we first propose the notion of online obfuscation, capturing the goal of disappearing programs in the bounded storage model. We give a negative result for Virtual Black Box (VBB) security in this model, but propose candidate constructions for a weaker security goal, namely Virtual Grey Box (VGB) security. We then demonstrate the utility of VGB online obfuscation, showing that it can be used to generate disappearing ciphertexts and signatures. All of our applications are *not* possible in the standard model of cryptography, regardless of computational assumptions used, and hence demonstrate the huge potential of combining time and space constraints for adversaries.

1.1.3 Incompressible Cryptography

Another observation we make is that in the Bounded Storage Model, the adversary needs to abide by the storage bound *throughout* the entire security game. However, in real life, short-term storage is much more achievable than long-term storage. For instance, it could be much easier to designate 1 TB of storage for 20 minutes than to keep 100 GB of data for 5 years. Although 1 TB is a larger space requirement than 100 GB, in the former case one can free up that space for other usages once the 20 minutes have passed, while in the latter case the storage, though somewhat smaller, has to be solely dedicated to this purpose for the entire 5 year period. So it seems natural to imagine a model where we only pose restrictions on the storage that is maintained over an extended period of time.

In the third part of this thesis, we modify the model for disappearing cryptography by giving the adversary an unbounded amount of storage when the transmission is happening, and bounding only the adversary's long-term storage. Here, we ask the following question:

If we give the adversary an unbounded amount of short-term storage, are there any meaning security notions that we can achieve?

We answer the question positively by proposing *Incompressible Cryptography*. *Incompressible encryption* allows us to make the ciphertext size flexibly large and ensures that an adversary learns nothing about the encrypted data, even if the decryption key later leaks, unless she stores essentially the entire ciphertext. *Incompressible signatures* can be made arbitrarily large and ensure that an adversary cannot produce a signature on *any* message, even one she has seen signed before, unless she stores one of the signatures essentially in its entirety. The notions are quite similar to that of disappearing cryptography except that the adversary is allowed an arbitrary amount of storage during the transmission phase and then *compresses* down to a smaller state for long-term storage, which we bound.

In this thesis, we give simple constructions of both incompressible public-key encryption and signatures under minimal assumptions. Furthermore, large incompressible ciphertexts (resp. signatures) can be decrypted (resp. verified) in a streaming manner with low storage. In particular, these notions further strengthen the related concepts of disappearing cryptography, the constructions of which rely on sophisticated techniques and strong, non-standard assumptions. We extend our constructions to achieve an optimal "rate", meaning the large ciphertexts (resp. signatures) can contain almost equally large messages, at the cost of stronger assumptions.

1.1.4 Multi-User Incompressible Encryption

The final observation we address in this thesis is the high communication complexity of the above schemes against space-bounded adversaries. In the Bounded Storage Model, the core

idea is to have the honest parties exchange more information than the adversary's storage, so the communication complexity is, by definition, higher than the adversary's storage bound. In the case of incompressible cryptography, though the model deviates from the Bounded Storage Model, it still requires the honest parties to exchange information that is incompressible to the adversary's bounded long-term storage. By a simple information-theoretic argument, the size of the honest party communication still needs to exceed the adversary's memory bound. So either way, to protect against adversaries with a given memory bound, each ciphertext/signature produced by the honest parties must be at least that size. This result seems inevitable, but could be quite undesirable when the message size itself is small. For instance, say we are trying to protect against adversaries with memory bounds up to 1 TB, then to send a single "Hello" message, the ciphertext needs to be terabytes in length, which makes the scheme immensely unusable. Therefore, the question we want to ask here is:

Can we better motivate these schemes against space-bounded adversaries, from a more practical point of view?

We provide an answer to this question by extending incompressible encryptions to the *multi-user* setting. Consider a state-level adversary who observes and stores large amounts of encrypted data from all users on the Internet, but does not have the capacity to store it all. Later, it may target certain "persons of interest" in order to obtain their decryption keys. We would like to guarantee that, if the adversary's storage capacity is only (say) 1% of the total encrypted data size, then even if it can later obtain the decryption keys of arbitrary users, it can only learn something about the contents of (roughly) 1% of the ciphertexts, while the rest will maintain full security. Under this setting, individual ciphertexts no longer need to be huge in size. They are aggregated with other ciphertexts from the same and other users,

and as long as the amalgamation exceeds the adversary's storage, the above security guarantee holds. In this thesis, We provide solutions in both the symmetric key and public key setting with various trade-offs in terms of computational assumptions and efficiency.

As the core technical tool, we study an information-theoretic problem which we refer to as "multi-instance randomness extraction". Suppose $X_1, \ldots, X_t$ are correlated random variables whose total joint min-entropy rate is α, but we know nothing else about their individual entropies. We choose t random and independent seeds $S_1, \ldots, S_t$ and attempt to individually extract some small amount of randomness $Y_i = \mathsf{Ext}(X_i; S_i)$ from each X_i. We'd like to say that roughly an α-fraction of the extracted outputs Y_i should be indistinguishable from uniform even given all the remaining extracted outputs and all the seeds. We show that this indeed holds for specific extractors based on Hadamard and Reed-Muller codes.

1.2 Organization

In Chapter 2, we introduce cryptographic primitives and information theoretic tools that are used in multiple chapters.

In Chapter 3, we give constructions for two-party key agreement, bit commitment, and oblivious transfer in the Bounded Storage model.

In Chapter 4, we present the concept of disappearing cryptography. We first devise the notion of an online obfuscator, and then show how a VGB online obfuscator can be used to achieve disappearing ciphertexts and signatures.

In Chapter 5, we propose incompressible cryptography and develop constructions of incompressible public-key encryption and signatures from standard assumptions. We also extend the constructions to achieve an optimal rate, with the tradeoff of stronger assumptions.

In Chapter 6, we elevate incompressible encryptions to the multi-user setting by replacing the randomness extractors in the constructions with a new "Multi-Instance Randomness Extractor".

1.3 Publications contained in this thesis

The results in this thesis are based on the following works:

- *Simple Schemes in the Bounded Storage Model*[62], with Mark Zhandry (EUROCRYPT 2019).
- *Disappearing Cryptography in the Bounded Storage Model*[63], with Mark Zhandry (TCC 2021).
- *Incompressible Cryptography*[60], with Daniel Wichs and Mark Zhandry (EUROCRYPT 2022).
- *Multi-Instance Randomness Extraction and Security against Bounded-Storage Mass Surveillance*[61], with Daniel Wichs and Mark Zhandry.

2

Preliminaries

In this chapter, we give preliminaries that are required across multiple chapters. For preliminaries that are only relevant for a specific chapter, those are given in each chapter's "Chapter Preliminaries" section.

Notation-wise, for $n \in \mathbb{N}$, we let $[n]$ denote the ordered set $\{1, 2, \ldots, n\}$. We use capital bold letters to denote a matrix $\mathbf{M}$. Lowercase bold letters denote vectors $\mathbf{v}$. Let $\mathbf{M}_{i,j}$ denote the element on the i-th row, and j-th column of $\mathbf{M}$, and $\mathbf{v}_i$ denote the i-th element of $\mathbf{v}$. For a bit-string $x \in \{0,1\}^n$, we let x_i denote the i-th bit of x. We use $\mathsf{diag}(\mathbf{M}_1, \ldots, \mathbf{M}_n)$ to denote a matrix with block diagonals $\mathbf{M}_1, \ldots, \mathbf{M}_n$.

2.1 Min-Entropy Extractor

Recall the definition for average min-entropy:

Definition 2.1.1 (Average Min-Entropy). *For two jointly distributed random variables (X, Y), the average min-entropy of X conditioned on Y is defined as*

$$H_\infty(X|Y) = -\log \mathbf{E}_{y \xleftarrow{\$} Y}[\max_x \Pr[X = x|Y = y]].$$

Lemma 2.1.1 ([43]). *For random variables X, Y where Y is supported over a set of size T, we have*

$$H_\infty(X|Y) \geq H_\infty(X, Y) - \log T \geq H_\infty(X) - \log T.$$

Definition 2.1.2 (Extractor[80]). *A function $\mathsf{Extract} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k, ϵ) strong average min-entropy extractor if, for all jointly distributed random variables (X, Y) where X takes values in $\{0,1\}^n$ and $H_\infty(X|Y) \geq k$, we have that $(U_d, \mathsf{Extract}(X; U_d), Y)$*

is ϵ-close to (s, U_m, Y), where U_d and U_m are uniformly random strings of length d and m respectively.

Remark 2.1.1. *Any strong randomness extractor is also a strong* average *min-entropy extractor, with a constant loss in ϵ.*

2.2 Functional Encryption

The concept of Functional Encryption (FE) is first raised by Sahai and Waters[91] and later formalized by Boneh, Sahai, Waters[19] and O'Neill[81].

Let λ be the security parameter. Let $\{\mathcal{C}_\lambda\}$ be a class of circuits with input space $\mathcal{X}_\lambda$ and output space $\mathcal{Y}_\lambda$. A functional encryption scheme for the circuit class $\{\mathcal{C}_\lambda\}$ is a tuple of PPT algorithms $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$ takes as input the security parameter λ, and outputs the master public key mpk and the master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$ takes as input the master secret key msk and a circuit $C \in \{\mathcal{C}_\lambda\}$, and outputs a function key sk_C.
- $\mathsf{Enc}(\mathsf{mpk}, m) \to \mathsf{ct}$ takes as input the public key mpk and a message $m \in \mathcal{X}_\lambda$, and outputs the ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \to y$ takes as input a function key sk_C and a ciphertext ct, and outputs a value $y \in \mathcal{Y}_\lambda$.

We can analogously define the "rate" of an FE scheme to be the ratio between the message length to the ciphertext length. We require correctness and security of a functional encryption scheme.

Definition 2.2.1 (Correctness)**.** *A functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be correct if for all* $C \in \{\mathcal{C}_\lambda\}$ *and* $m \in \mathcal{X}_\lambda$*:*

$$\Pr\left[y = C(m) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m) \\ y \leftarrow \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Consider the following *Semi-Adaptive Security Experiment*, $\mathsf{Dist}^{\mathsf{SemiAdpt}}_{\mathsf{FE},\mathcal{A}}(\lambda)$:

- Run $\mathsf{FE.Setup}(1^\lambda)$ to obtain $(\mathsf{mpk}, \mathsf{msk})$ and sample a random bit $b \leftarrow \{0,1\}$.
- On input 1^λ and mpk, The adversary $\mathcal{A}$ submits the challenge query consisting of two messages m_0 and m_1. It then receives $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, m_b)$.
- The adversary now submits a circuit $C \in \{\mathcal{C}_\lambda\}$ s.t. $C(m_0) = C(m_1)$, and receives $\mathsf{sk}_C \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, C)$.
- The adversary $\mathcal{A}$ outputs a guess b' for b. If $b' = b$, we say that the adversary succeeds and experiment outputs 1. Otherwise, the experiment outputs 0.

Definition 2.2.2 (Single-Key Semi-Adaptive Security)**.** *For security parameter* λ*, a functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to have single-key semi-*

adaptive security if for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\mathsf{Dist}^{\mathsf{SemiAdpt}}_{\mathsf{FE},\mathcal{A}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

We can also consider *selective* security, where the adversary only receives mpk *after* sending the challenge messages. We can also consider *many-time* semi-adaptive/selective security, where the adversary is able to adaptively query for as many sk_C as it would like, provided they all occur after the challenge query.

3

Simple Schemes in the Bounded Storage Model

3.1 Introduction

For the vast majority of cryptographic applications, security relies on the assumed hardness of certain computational problems, such as factoring large integers or inverting certain hash functions. Unfortunately, with the current state of complexity theory the hardness of these problems can only be conjectured. This means that the security of such schemes is always *conditional* on such conjectures being true.

Maurer proposes the Bounded Storage Model[77] as an alternate model for constraining the adversary; here, instead of constraining the adversary's time, the adversary's memory is bounded. Amazingly, it is actually possible to give *unconditional* proofs of security for schemes in this model. The core idea is that the honest parties exchange so much information that the adversary cannot possibly store it all. Then, schemes are cleverly devised to exploit the adversary's lack of knowledge about the scheme.

Moreover, the space bounds are only necessary when the protocol is run, and even if the adversary later gains more space the protocol remains secure. This means schemes only need to be designed with current storage capacities in mind. This is fundamentally different than the usual approach of time-bounding adversaries, where an adversary can later break the protocol if its computational abilities increase. Hence, traditional schemes must be designed with future computational abilities in mind. This is especially important in light of recent developments in quantum computing, as Grover's algorithm[59] and Shor's algorithm[92] can speed up attacks on many current cryptographic protocols. Hence, much of the communication taking place today will be revealed once quantum computers become reality.

In this chapter, we devise very simple round-optimal protocols for bit-commitment and

oblivious transfer (namely, 1 round and 2 rounds, respectively) in the Bounded Storage Model, improving 5 rounds needed in prior works. We additionally develop a new key agreement protocol with several advantages over prior works. Our results rely on Raz's recent space lower bound for learning parities[86], and in particular the simple encryption scheme based on this lower bound. Our key observation is that Raz's encryption scheme has several useful properties — including additive homomorphism and leakage resilience — that can be useful for building higher-level protocols. Our core technical contribution is a new "encrypt zero" protocol for Raz's encryption scheme, which may be of independent interest.

Our schemes are based on entirely different techniques than most of the prior literature — most of which is based on the birthday paradox — and we believe this result will therefore be a useful starting point for future work in the bounded storage model.

3.1.1 Prior Work in the Bounded Storage Model

Prior work in the Bounded Storage Model[77,27,26,76,7,39,40] typically uses something akin to the birthday paradox to achieve security against space-bounded adversaries.

In slightly more detail, the key agreement scheme of Maurer[77] works as follows. One party sends a stream of roughly n^2 random bits to the other party [*]. Each party records a random secret subset of n bits of the stream. By the birthday paradox, the two parties will have recorded one bit position in common with constant probability. They therefore share the bit positions they recorded with each other, and set their secret key to be the bit of the stream at the shared position.

[*] In most works in the Bounded Storage Model, the random bit stream is assumed to come from a trusted third party. Here we will insist on there being no trusted third party, and instead the bit stream comes from the parties themselves.

An eavesdropper first sees n^2 random bits. If the eavesdropper's storage is somewhat lower than n^2, he cannot possibly remember the entire sequence of random bits. In particular, it can be shown that the adversary has little information about the bit shared by the two honest parties. This remains true even after the parties share their bit positions. Notice that the honest parties require space n, and security holds even for adversaries with space Cn^2 for some constant C. Therefore, by tuning n so that n storage is feasible, but Cn^2 is not, one obtains the desired security.

Much of the literature on the Bounded Storage Model relies on this sort of birthday attack property. Unfortunately, this leads to several difficulties:

- The two honest parties only achieve success with constant probability. In order to achieve success with high probability, the protocol either needs to be repeated many times (thus requiring more than n^2 communication) or requires the honest users to store more than n positions (thus requiring more than n space, and making the gap between the honest users and adversaries less than quadratic).

- Remembering n random positions out of n^2 requires $O(n \log n)$ space just to record the indices. To compress the space requirements of the honest parties, the positions are actually chosen by a pairwise independent function, complicating the scheme slightly.

- The adversary has a $1/n^2$ chance of guessing the bit position shared by the two users. As such, the adversary has a non-negligible advantage in guessing the bit. To get statistical security, a randomness extraction step is applied, adding slightly to the complexity of the protocol.

- More importantly, there is very little structure to exploit with the birthday approach.

For more advanced applications such as oblivious transfer or bit commitment, the protocols end up being somewhat complicated and require several rounds.

3.1.2 Space Lower Bounds For Learning Parities

In this chapter, we exploit recent space lower bounds due to Raz[86]. Raz considers a setting where one party holds a secret key $\mathbf{k} \in \{0,1\}^n$, and streams random tuples $(\mathbf{r}_i, \mathbf{r}_i \cdot \mathbf{k})$, where $\mathbf{r}_i$ is random in $\{0,1\}^n$ and the inner product is taken mod 2. Raz asks: given these random tuples, and only limited storage (namely Cn^2 for some constant C), how hard is it to recover $\mathbf{k}$? Clearly, if $C \approx 1$, then one can store n tuples, and then recover $\mathbf{k}$ using linear algebra. But if $C \ll 1$, then the adversary has no hope of storing enough tuples to perform linear algebra.

Raz proves that, for some constant C (roughly $1/20$), then either the adversary needs an exponential (in n) number of samples, or the adversary's probability of correctly guessing $\mathbf{k}$ is exponentially small.

Raz observes that his lower bound easily leads to a secret key encryption scheme in the bounded storage model. The key will be an n-bit string $\mathbf{k}$. To encrypt a message bit b, choose a random $\mathbf{r}$, and produce the ciphertext $(\mathbf{r}, \mathbf{r} \cdot \mathbf{k} \oplus b)$. Raz's lower bound shows that after seeing fewer than exponentially many encrypted messages, an adversary with Cn^2 space has an exponentially small probability of guessing $\mathbf{k}$. This means $\mathbf{k}$ always has some min-entropy conditioned on the adversaries' view. Then using the fact that the inner product is a good extractor, we have that for any new ciphertext $\mathbf{r} \cdot \mathbf{k}$ is statistically close to random, and hence masks the message b.

3.1.3 Our Construction

In this chapter, we use Raz's scheme in order to develop simple new constructions in the Bounded Storage Model that have several advantages over prior work.

Our main observation is that Raz's encryption scheme has several attractive properties. First, it is leakage resilient: since inner products are strong extractors, the scheme remains secure even if the adversary has partial knowledge of the key, as long as the conditional min-entropy of the key is large.

Next, we note that Raz's scheme is additively homomorphic: given encryptions $(\mathbf{r}_0, \mathbf{r}_0 \cdot \mathbf{k} \oplus m_0)$ and $(\mathbf{r}_1, \mathbf{r}_1 \cdot \mathbf{k} \oplus m_1)$ of m_0, m_1, we can compute an encryption of $m_0 \oplus m_1$ by simply taking the componentwise XOR of the two ciphertexts, yielding $(\mathbf{r}_0 \oplus \mathbf{r}_1, (\mathbf{r}_0 \oplus \mathbf{r}_1) \cdot \mathbf{k} \oplus (m_0 \oplus m_1))$. This additive homomorphism will prove very useful. We can also toggle the bit being encrypted by toggling the last bit of a ciphertext.

For example, Rothblum[90] shows that any additively homomorphic secret key encryption scheme can be converted into a public key (additively homomorphic) encryption scheme. The rough idea is that the public key consists of many encryptions of zero. Then, to devise an encryption of a bit m, simply add a random subset sum of the public key ciphertexts to get a "fresh" encryption of zero, and then toggle the encrypted bit as necessary to achieve an encryption of m.

Key Agreement. In the case of Raz's scheme, the public key will end up containing $O(n)$ ciphertexts, meaning the public key is too large for the honest users to even write down. However, we can re-interpret this protocol as a *key-agreement* protocol. Here, the public key is streamed from user A to user B, who applies the additive homomorphism to construct the

fresh encryption on the fly. Now one party knows the secret key, and the other has a fresh ciphertext with a known plaintext. So the second party just sends the ciphertext back to the first party, who decrypts. The shared key is the plaintext value.

Bit Commitment. Next, we observe that the public key encryption scheme obtained above is *committing*: for any public key there is a unique secret key. Therefore, we can use the scheme to get a bit commitment scheme as follows: to commit to a bit b, the Committer simply chooses a random secret key, streams the public key to the receiver, and then sends an encryption of b. To open the commitment, the Committer simply sends the secret decryption key. The Verifier, on the other hand, constructs several fresh encryptions of 0 by reading the Committer's stream, as user B did in our key agreement protocol. Upon receiving a supposed secret key, the Verifier checks that all the encryptions do in fact decrypt to 0. If so, then it decrypts the commitment to get the committed value.

Oblivious Transfer. We can also turn this commitment scheme into an oblivious transfer protocol: the Receiver, on input b, commits to the bit b. Then the Sender, on input x_0, x_1, using the homomorphic properties of the encryption scheme, turns the encryption of b in the commitment into encryptions of $(1 - b)x_0$ and bx_1. To maintain privacy of x_{1-b}, the Sender will *re-randomize* the encryptions, again using the homomorphic properties. To re-randomize, the Sender will construct some fresh encryptions of zero, again just as user B did in our key agreement protocol. The Receiver can then decrypt these ciphertexts, which yield 0 and x_b.

Malicious Security. The commitment scheme and the oblivious transfer protocol are secure as long as the public key is generated correctly. This occurs, for example, if the randomness for the encryptions of 0 is generated and streamed by a trusted third party. This is the setting considered in much of the prior work in the bounded storage model.

On the other hand, if we do not wish to rely on a trusted third party to generate the encryption randomness, a malicious Committer can choose a public key with bad randomness, which will allow him to break the commitment, as explained below. This also would let the Receiver break the security of the oblivious transfer protocol. We therefore additionally show how to modify the constructions above to obtain security for malicious parties without relying on a trusted third party. The result is round-optimal protocols for bit-commitment and oblivious transfer without a trusted third party.

3.1.4 Additional Technical Details

The Encrypt Zero Protocol. Notice that all of our schemes have a common feature: one user has a secret key, and the other user obtains encryptions of 0. Importantly for security, these encryptions of 0 should be independent of the view of the first user.

In order to unify our schemes, we abstract the common features required with an *Encrypt Zero* protocol for Raz's encryption scheme. The goal of the protocol is to give one party, the Keeper, a random key $\mathbf{s}$, and another party, the Recorder, λ random encryptions $\{c_1, \ldots, c_\lambda\}$ of 0. Here, λ is a parameter that will be chosen based on application. Recorder security dictates that the Keeper learns nothing about the λ encryptions stored by the Recorder (aside from the fact that they encrypt 0). Keeper security requires that the min-entropy of the key $\mathbf{s}$ conditioned on the Recorder's view is $\Omega(n)$. We additionally require that the Keeper's space

is $O(n)$ (which is optimal since the Keeper must store a secret key of $O(n)$ bits), and the Recorder's space is $O(\lambda n)$ (which is also optimal, since the Recorder must store λ encryptions of $O(n)$ bits each).

Our basic protocol for Raz's scheme works as follows:

- The Keeper chooses a random key $\mathbf{k} \in \{0,1\}^n$. Let $m = O(n)$ be a parameter. The Recorder chooses a secret matrix $\mathbf{\Sigma} \in \{0,1\}^{\lambda \times m}$.

- The Keeper streams m encryptions $(\mathbf{r}_i, a_i = \mathbf{r}_i \cdot \mathbf{k} + 0)$ to the Recorder, for random $\mathbf{r}_i \in \{0,1\}^n$ and $i = 1, 2, \ldots, m$. From now on, we use the convention that "+" and "$\cdot$" are carried out mod 2.

- The Recorder maintains matrix $\mathbf{\Psi} \in \{0,1\}^{\lambda \times n}$ and column vector $\boldsymbol{\kappa} \in \{0,1\}^{\lambda}$. Each row of $(\mathbf{\Psi}|\boldsymbol{\kappa})$ will be a random subset-sum of the encryptions sent by the Keeper, with each subset-sum chosen according to $\mathbf{\Sigma}$. The matrices will be computed on the fly. So when $(\mathbf{r}_i, a_i)$ comes in, the Recorder will map $\mathbf{\Psi} \to \mathbf{\Psi} + \boldsymbol{\sigma}_i \cdot \mathbf{r}_i$, $\boldsymbol{\kappa} \to \boldsymbol{\kappa} + \boldsymbol{\sigma}_i a_i$. Here, $\boldsymbol{\sigma}_i$ is the i-th column of $\mathbf{\Sigma}$, and $\mathbf{r}_i$ is interpreted as a row vector.

- At the end of the protocol, the Keeper outputs its key $\mathbf{s} = \mathbf{k}$, and the Recorder outputs $(\mathbf{\Psi}|\boldsymbol{\kappa})$, whose rows are the ciphertexts $c_1, \ldots, c_\lambda$.

Let $\mathbf{R}$ be the matrix whose rows are the $\mathbf{r}_i$'s, and let $\mathbf{a}$ be the column vector of the a_i's. Then we have that $\mathbf{a} = \mathbf{R} \cdot \mathbf{k}$, $\mathbf{\Psi} = \mathbf{\Sigma} \cdot \mathbf{R}$, and $\boldsymbol{\kappa} = \mathbf{\Sigma} \cdot \mathbf{a} = \mathbf{\Psi} \cdot \mathbf{k}$. Hence, the rows of $(\mathbf{\Psi}|\boldsymbol{\kappa})$ are encryptions of zero, as desired.

For Keeper security, Raz's theorem directly shows that $\mathbf{k}$ has min-entropy relative to the Recorder's view. For Recorder security, notice that $\mathbf{\Sigma}$ is independent of the the Keeper's

view. Therefore, if the Keeper follows the protocol and m is slightly larger than n so that $\mathbf{R}$ is full rank with high probability, then $\mathbf{\Psi}$ is a random matrix independent of the adversary's view. Therefore the ciphertexts c_i are actually *random* encryptions of 0. Thus we get security for honest-but-curious Keepers.

KEY AGREEMENT. This protocol gives a simple key-agreement scheme. Basically, one party acts as the Keeper, and one as the Recorder. We set $\lambda = 1$. The result of the Encrypt Zero protocol is that the Recorder contains a uniformly random encryption of 0. The Recorder simply flips the bit encrypted with probability $1/2$ to get a random encryption of a random bit b, and sends the resulting ciphertext to the Keeper. The Keeper decrypts, and the shared secret key is just the resulting plaintext b.

Security of the protocol follows from the fact that after the Encrypt Zero protocol, the Keeper's key has min-entropy relative to any eavesdropper (since the eavesdropper learns no more than the Recorder). Moreover, the Keeper acts honestly, so the final ciphertext is always a fresh encryption. Finally, the encryption scheme is leakage resilient so it hides the bit b even though the adversary may have some knowledge of the key.

Notice that this scheme has *perfect* correctness, in that the two parties always arrive at a secret key. This is in contrast to the existing schemes based on the birthday paradox, where security is only statistical, and moreover this holds only if the adversary's space bounds are asymptotically smaller than n^2. In contrast, we get perfect correctness and statistical security for adversarial space bounds that are $O(n^2)$. The honest users only require $O(n)$ space.

BIT COMMITMENT. We now describe a simple bit-commitment protocol using the above Encrypt Zero protocol. Recall that in a bit-commitment scheme, there are two phases: a com-

mit phase where the Committer commits to a bit b, and a reveal or de-commit phase where the Committer reveals b and proves that b was the value committed to. After the commit phase, we want that the bit b is hidden. On the other hand, we want the commit phase to be binding, in that the Committer cannot later change the committed bit to something else.

The Committer and the Verifier will run the Encrypt Zero protocol, with Committer playing the role of Keeper and Verifier the role of Recorder. The protocol works as follows:

- Run the Encrypt Zero protocol, giving the Committer a random key $\mathbf{s}$ and the Verifier λ random encryptions c_i of 0.
- The Committer then sends an encryption of b relative to the key $\mathbf{s}$.
- To open the commitment, the Committer sends $\mathbf{s}$. The Verifier checks that $\mathbf{s}$ correctly decrypts all the c_i to 0. If so, it decrypts the final ciphertext to get b.

The security of the Encrypt Zero protocol and the leakage resilience of the encryption scheme show that this scheme is hiding. For binding, we note that an honest Committer will have no idea what encryptions c_i the Verifier has. As such, if the Committer later tries to change its committed bit by sending a malicious key $\mathbf{s}'$, $\mathbf{s}'$ will cause each ciphertext c_i to decrypt to 1 with probability 1/2. Therefore, the Committer will get caught with probability $1 - 2^{-\lambda}$.

Already, this gives a very simple protocol for bit commitment that is *non-interactive*; in contrast, the prior work of Ding *et al.*[40] required five rounds. One limitation is that we require the Committer to behave honestly during the commit phase. For example, if the Committer chooses $\mathbf{R}$ to be low rank, then the encryptions obtained by the Verifier will not be

independent of the Committer's view, and hence the Committer may be able to cheat during the de-commit phase.

To get around this, we tweak the Encrypt Zero protocol slightly to get security even against malicious Keepers. Our Enhanced Encrypt Zero protocol is as follows:

- The Keeper chooses a random key $\mathbf{k} \in \{0,1\}^n$ and an independent random secret $\mathbf{s} \in \{0,1\}^m$. We will let $m = 2n$. The Recorder chooses a secret matrix $\mathbf{\Sigma} \in \{0,1\}^{\lambda \times m}$.
- The Keeper streams random encryptions of the bits of s_i. We will write this in matrix form as $(\mathbf{R}, \mathbf{a} = \mathbf{R} \cdot \mathbf{k} + \mathbf{s})$.
- The Recorder computes $\mathbf{\Psi} = \mathbf{\Sigma} \cdot \mathbf{R}$ and $\boldsymbol{\kappa} = \mathbf{\Sigma} \cdot \mathbf{a}$.
- The Keeper then sends its key $\mathbf{k}$ *in the clear*.
- The Keeper outputs its secret $\mathbf{s}$ as the key, and the Recorder outputs $(\mathbf{\Sigma}, \boldsymbol{\kappa} - \mathbf{\Psi} \cdot \mathbf{k})$.

Notice that $\boldsymbol{\kappa} - \mathbf{\Psi} \cdot \mathbf{k} = \mathbf{\Sigma} \cdot \mathbf{s}$, a list of λ encryptions of 0 relative to the key $\mathbf{s}$, as desired. Moreover, these encryptions are random encryptions, even if $\mathbf{R}$ is chosen adversarially by the Keeper, since the Keeper has no knowledge or control over $\mathbf{\Sigma}$.

To prove the min-entropy of $\mathbf{s}$ relative to a malicious Recorder, we note that the real-or-random CPA security of the encryption scheme shows that just prior to receiving $\mathbf{k}$, the Recorder has essentially no information about $\mathbf{s}$. Then, since $\mathbf{k}$ is n bits, revealing it can only reveal n bits of $\mathbf{s}$. But $\mathbf{s}$ is a uniformly random $m = 2n$ bit string, meaning it has roughly n bits of min-entropy remaining, as desired. Thus we get both our security properties, even for malicious parties.

Our Enhanced Encrypt Zero protocol roughly doubles the communication, but otherwise maintains all the attractive properties of the original scheme: it is non-interactive and has perfect correctness.

Putting it all together, our bit commitment protocol is the following:

- To commit to a bit b, the Committer streams $\mathbf{R}$, $\mathbf{a} = \mathbf{R} \cdot \mathbf{k} + \mathbf{s}$ followed by $\mathbf{k}, \boldsymbol{\gamma}, c = \boldsymbol{\gamma} \cdot \mathbf{s} + b$ for random $\mathbf{R}, \mathbf{k}, \mathbf{s}, \boldsymbol{\gamma}$.
- The Verifier records $\boldsymbol{\Sigma}$, $\boldsymbol{\Psi} = \boldsymbol{\Sigma} \cdot \mathbf{R}$, $\boldsymbol{\kappa} = \boldsymbol{\Sigma} \cdot \mathbf{a}$ for a random choice of $\boldsymbol{\Sigma}$, and then once $\mathbf{k}$ comes in it computes $\boldsymbol{\phi} = \boldsymbol{\kappa} - \boldsymbol{\Psi} \cdot \mathbf{k} = \boldsymbol{\Sigma} \cdot \mathbf{s}$.
- To reveal the bit b, the Committer just sends $\mathbf{x} = \mathbf{s}$.
- The Verifier checks that $\boldsymbol{\phi} = \boldsymbol{\Sigma} \cdot \mathbf{x}$. If so, it computes $b' = c - \boldsymbol{\gamma} \cdot \mathbf{x}$.

Oblivious Transfer. We now turn to constructing an oblivious transfer (OT) protocol. In an OT protocol, one party, the Sender, has two input bits x_0, x_1. Another party, the Receiver, has a bit b. The Receiver would like to learn x_b without revealing b, and the Sender would like to ensure that the Receiver learns nothing about x_{1-b}.

In our protocol, the Receiver will play the role of Committer in our commitment scheme, committing to its input b. The Sender will play the role of Recorder in the Encrypt Zero protocol, setting $\lambda = 2$. The hiding property of the commitment scheme ensures that the space-bounded Sender learns nothing about the Receiver's bit b.

At the end of the Receiver's message, the Sender has an encryption $(\boldsymbol{\gamma}, c^* = \boldsymbol{\gamma} \cdot \mathbf{s} + b)$ of b with secret key $\mathbf{s}$. Additionally, it also has two encryptions of 0, namely $(\boldsymbol{\sigma}_0, c_0 = \boldsymbol{\sigma}_0 \cdot \mathbf{s})$

and $(\boldsymbol{\sigma}_1, c_1 = \boldsymbol{\sigma}_1 \cdot \mathbf{s})$ for random vectors $\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1$. Importantly, $\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1$ are independent of the Receiver's view, as they were chosen by the Sender.

The Sender will now exploit the additive homomorphism of the encryption scheme once more. In particular, it will compute encryptions of $(1-b)x_0$ and bx_1, which it will then send back to the Receiver. To compute an encryption of bx_1, it simply multiplies the ciphertext $(\boldsymbol{\gamma}, c^*)$ by x_1. Similarly, to compute an encryption of $(1-b)x_0$, it toggles c^* (to get an encryption of $1-b$) and then multiplies the entire ciphertext by x_0.

Now clearly these two ciphertexts reveal both x_0 and x_1, so the Sender cannot send them directly to the Receiver. Instead, it will *re-randomize* them by adding the two encryptions of 0. Now it obtains *fresh* encryptions of $(1-b)x_0$ and bx_1:

$$\begin{aligned}
\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma},\ c_0 + x_0(1-c^*) &= (\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma}) \cdot \mathbf{s} + \big((1-b)x_0\big) \\
\boldsymbol{\sigma}_1 + x_1\boldsymbol{\gamma},\ c_1 + x_1 c^* &= (\boldsymbol{\sigma}_1 + x_1\boldsymbol{\gamma}) \cdot \mathbf{s} + \big(bx_1\big)
\end{aligned}$$

It sends these ciphertexts to the Receiver, who then decrypts. All the Receiver learns then is $(1-b)x_0$ and bx_1. One of these plaintexts will be x_b as desired, and the other will be 0. Thus, the Receiver learns nothing about x_{1-b}.

Our protocol is round-optimal, since it involves only a single message in each direction. This improves on the best prior work of Ding *et al.*[40] requiring 5 rounds. Additionally, our protocol is much simpler than the prior work.

3.1.5 Other Related Work

A separate work by Ball *et al.* [8] shows another application of Raz's encryption scheme, where they use it to construct unconditional non-malleable codes against streaming, space-bounded tempering.

3.2 Chapter Preliminaries

Here, we recall some basic cryptographic notions, translated into the setting of the bounded storage model. In the following definitions, n will be a security parameter.

A symmetric encryption scheme is a pair of algorithms $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ with an associated key space $\mathcal{K}_n$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}_n$. Notice that the key space and ciphertext space depend on n; the message space will not depend on n. We require that:

- $\mathsf{Enc} : \mathcal{K}_n \times \mathcal{M} \to \mathcal{C}_n$ is a probabilistic polynomial time (PPT) algorithm
- $\mathsf{Dec} : \mathcal{K}_n \times \mathcal{C}_n \to \mathcal{M}$ is a deterministic polynomial time algorithm.
- Correctness: for any $k \in \mathcal{K}_n$ and any message $m \in \mathcal{M}$,

$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1.$$

Additionally, we will require a security notion. In this chapter, we will focus on the following notion.

Definition 3.2.1 (Real-or-Random-Ciphertext (RoRC) Security)**.** *Let $\mathcal{A}$ be an adversary. $\mathcal{A}$ plays the following game* $\mathsf{RoRC}_{\mathcal{A},\Pi,b}(n, q)$*:*

- *The challenger's input is a bit* $b \in \{0, 1\}$.
- *The challenger chooses a random key* $k \in \mathcal{K}_n$□
- $\mathcal{A}$ *makes* q *adaptive queries on messages* $m_1, \ldots, m_q \in \mathcal{M}$.
- *In response to each query, the challenger does the following:*
  - *If* $b = 0$, *the challenger responds with* $c_i \leftarrow \mathsf{Enc}(k, m_i)$.
  - *If* $b = 1$, *the challenger responds with a random ciphertext* $c_i \in \mathcal{C}_n$.
- *Finally,* $\mathcal{A}$ *outputs a guess* b' *for* b.

We say that Π *is* $(S(n), Q(n), \epsilon)$*-secure if for all adversaries that use at most* $S(n)$ *memory bits and* $Q(n)$ *queries (i.e.* $q \leq Q(n)$*),*

$$|\Pr[\mathsf{RoRC}_{\mathcal{A},\Pi,0}(n, q) = 1] - \Pr[\mathsf{RoRC}_{\mathcal{A},\Pi,1}(n, q) = 1]| \leq \epsilon.$$

In this chapter, a lot of the proofs are based on the Leftover Hash Lemma for Conditional Min-Entropy due to Impagliazzo, Levin, and Luby[69].

For random distributions X and Y, let $H_\infty(X|Y)$ denote the min-entropy of X conditioned on Y. Let $X \approx_\epsilon Y$ denote that the two distributions are ϵ-close, i.e. the statistical distance between these two distributions $\Delta(X, Y) \leq \epsilon$. Furthermore, let U_m denote a uniformly distributed random variable of m bits for some positive integer m.

Lemma 3.2.1 (Leftover Hash Lemma for Conditional Min-Entropy[69])**.** *Let* X, E *be a joint distribution. If* $H_\infty(X|E) \geq k$, *and* $m = k - 2\log(1/\epsilon)$, *then*

$$(H(X), H, E) \approx_{\epsilon/2} (U_m, U_d, E),$$

where m is the output length of a universal hash function H, and d is the length of the description of H.

3.3 Raz's Encryption Scheme

Our constructions of the commitment scheme and the oblivious transfer scheme are largely based on the bit encryption scheme from parity learning proposed by Raz[86]. Raz sketches how his lower bound for learning implies the security of his encryption scheme. Below we reproduce the construction of the encryption scheme, and formalize the security proof.

Construction 3.3.1 (Bit Encryption Scheme from Parity Learning)**.** *For a given security parameter n, the encryption scheme consists of a message space $\mathcal{M} = \{0,1\}$, a ciphertext space $\mathcal{C}_n = \{0,1\}^n \times \{0,1\}$, a key space $\mathcal{K}_n = \{0,1\}^n$, and a pair of algorithms $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ as specified below:*

- $\mathsf{Enc}(\mathbf{k}, m \in \mathcal{M})$*: Samples a random row vector $\mathbf{r} \leftarrow \{0,1\}^n$, computes $a = \mathbf{r} \cdot \mathbf{k} + m$, and outputs the ciphertext $c = (\mathbf{r}, a)$ as a pair.*
- $\mathsf{Dec}(\mathbf{k}, c = (\mathbf{r}, a) \in \mathcal{C}_n)$*: Computes and outputs $m' = \mathbf{r} \cdot \mathbf{k} + a$.*

To prove Real-or-Random-Ciphertext security of the above scheme, we rely on a result from Raz[86], reproduced below.

Lemma 3.3.1 ([86])**.** *For any $C < \frac{1}{20}$, there exists $\alpha > 0$, such that: for uniform $\mathbf{k} \in \{0,1\}^n$, $m \leq 2^{\alpha n}$, and algorithm $\mathcal{A}$ that takes a stream of $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, $\ldots$, $(\mathbf{x}_m, y_m)$, where $\mathbf{x}_i$ is a uniform distribution over $\{0,1\}^n$ and $y_i = \mathbf{x}_i \cdot \mathbf{k}$ for every i, under the condition that $\mathcal{A}$ uses at most Cn^2 memory bits and outputs $\tilde{\mathbf{k}} \in \{0,1\}^n$, then $Pr[\tilde{\mathbf{k}} = \mathbf{k}] \leq O(2^{-\alpha n})$.*

We also rely on the Goldreich-Levin Algorithm, reproduced below.

Lemma 3.3.2 (Goldreich-Levin Algorithm [55]). *Assume that there exists a function* $f : \{0,1\}^n \to \{0,1\}$ *s.t. for some unknown* $\mathbf{x} \in \{0,1\}^n$*, we have*

$$\Pr_{\mathbf{r}\in\{0,1\}^n}[f(\mathbf{r}) = \langle \mathbf{x}, \mathbf{r}\rangle] \geq \frac{1}{2} + \epsilon$$

for $\epsilon > 0$*.*

Then there exists an algorithm $\mathcal{GL}$ *that runs in time* $O(n^2\epsilon^{-4}\log n)$*, makes* $O(n\epsilon^{-4}\log n)$ *orcale queries into f, and outputs* $\mathbf{x}$ *with probability* $\Omega(\epsilon^2)$*.*

Instead of directly proving RoRC security of the encryption scheme, we prove Modified Real-or-Random-Ciphertext (RoRC') security, which differs from RoRC security in that for all but the last query, the challenger always responds with the valid encryption of the messsage; for the last query, the challenger responds either with a valid encryption or a random ciphertext, each with probability $1/2$. A detailed definition is given below.

Definition 3.3.1 (Modified Real-or-Random-Ciphertext (RoRC') Security). *Let* $\mathcal{A}$ *be an adversary.* $\mathcal{A}$ *plays the following game* $\mathsf{RoRC}'_{\mathcal{A},\Pi,b}(n,q)$*:*

- *The challenger's input is a bit* $b \in \{0,1\}$*.*
- *The challenger chooses a random key* $k \in \mathcal{K}_n$*.*
- $\mathcal{A}$ *makes q adaptive queries on messages* $m_1, \ldots, m_q \in \mathcal{M}$*.*
- *In response to query* m_i *with* $1 \leq i \leq q-1$*, the challenger responds with* $c_i \leftarrow \mathsf{Enc}(k, m_i)$*.*

- *In response to query m_q, the challenger does the following:*
  - *If $b = 0$, the challenger responds with $c_q \leftarrow \mathsf{Enc}(k, m_q)$.*
  - *If $b = 1$, the challenger responds with a random ciphertext $c_q \in \mathcal{C}_n$.*
- *Finally, $\mathcal{A}$ outputs a guess b' for b.*

We say that Π is $(S(n), Q(n), \epsilon)$-secure if for all adversaries that use at most $S(n)$ memory bits and $Q(n)$ queries (i.e. $q \leq Q(n)$),

$$|\Pr[\mathsf{RoRC'}_{\mathcal{A},\Pi,0}(n,q) = 1] - \Pr[\mathsf{RoRC'}_{\mathcal{A},\Pi,1}(n,q) = 1]| \leq \epsilon.$$

We now show that RoRC' security implies RoRC security.

Lemma 3.3.3. *An encryption scheme that is $(S(n), Q(n), \epsilon)$-secure under the RoRC' setting is $(S(n), Q(n), Q(n)\epsilon)$-secure under the RoRC setting.*

Proof. We prove this using a hybrid argument. For any $q \leq Q(n)$, consider the hybrid security games $H_0, H_1, \ldots, H_q$, where H_j describes the following hybrid game:

- The challenger chooses a random key $k \in \mathcal{K}_n$.
- $\mathcal{A}$ makes q adaptive queries on messages $m_1, \ldots, m_q \in \mathcal{M}$.
- In response to query m_i with $1 \leq i \leq j$, the challenger responds with $c_i \leftarrow \mathsf{Enc}(k, m_i)$.
- In response to query m_i with $j + 1 \leq i \leq q$, the challenger responds with a random ciphertext $c_i \in \mathcal{C}_n$.

Particularly, notice that H_0 corresponds to a game where the challenger always responds with random ciphertexts, and that H_q corresponds to a game where the challenger always responds with valid encryptions of the messages. In that way, the $\mathsf{RoRC}_{\mathcal{A},\Pi,b}(n,q)$ game is equivalent to distinguishing H_q from H_0.

To put this formally, let D be an arbitrary distinguisher, and $h \leftarrow H_j$ denote a randomly sampled instance of the game H_j, we have

$$\begin{aligned}
&|\Pr[\mathsf{RoRC}_{\mathcal{A},\Pi,0}(n,q)=1] - \Pr[\mathsf{RoRC}_{\mathcal{A},\Pi,1}(n,q)=1]| \\
=&\left|\Pr_{h\leftarrow H_q}[D(h)=1] - \Pr_{h\leftarrow H_0}[D(h)=1]\right|.
\end{aligned}$$

By the hybrid argument, there exists j, s.t. $0 \leq j < q$ and

$$\left|\Pr_{h\leftarrow H_q}[D(h)=1] - \Pr_{h\leftarrow H_0}[D(h)=1]\right| \leq q\left|\Pr_{h\leftarrow H_{j+1}}[D(h)=1] - \Pr_{h\leftarrow H_j}[D(h)=1]\right|.$$

To distinguish between H_{j+1} and H_j, consider the following security game $\mathsf{Dist}_{\mathcal{A},\Pi,b}(n,q,j)$:

- The challenger's input is a bit $b \in \{0,1\}$.
- The challenger chooses a random key $k \in \mathcal{K}_n$.
- $\mathcal{A}$ makes q adaptive queries on messages $m_1,\ldots,m_q \in \mathcal{M}$.
- In response to query m_i with $1 \leq i \leq j$, the challenger responds with $c_i \leftarrow \mathsf{Enc}(k,m_i)$.
- In response to query m_{j+1}, the challenger does the following:
  - If $b=0$, the challenger responds with $c_{j+1} \leftarrow \mathsf{Enc}(k,m_{j+1})$.

– If $b = 1$, the challenger responds with a random ciphertext $c_{j+1} \in \mathcal{C}_n$.

- In response to query m_i with $j+1 < i \leq q$, the challenger responds with a random ciphertext $c_i \in \mathcal{C}_n$.

- Finally, $\mathcal{A}$ outputs a guess b' for b.

This directly gives us

$$\begin{aligned}
&\left| \Pr_{b \leftarrow H_{j+1}}[D(h) = 1] - \Pr_{b \leftarrow H_j}[D(h) = 1] \right| \\
&= \left| \Pr[\mathsf{Dist}_{\mathcal{A},\Pi,0}(n,q,j) = 1] - \Pr[\mathsf{Dist}_{\mathcal{A},\Pi,1}(n,q,j) = 1] \right|.
\end{aligned}$$

Next, we show that we can use an adversary $\mathcal{A}$ for the $\mathsf{Dist}_{\mathcal{A},\Pi,b}(n,q,j)$ game to construct an adversary $\mathcal{A}'$ for the $\mathsf{RoRC}'_{\mathcal{A}',\Pi,b}(n,j+1)$ game. Notice that the only difference between $\mathsf{RoRC}'_{\mathcal{A}',\Pi,b}(n,j+1)$ and $\mathsf{Dist}_{\mathcal{A},\Pi,b}(n,q,j)$ is that $\mathsf{Dist}_{\mathcal{A},\Pi,b}(n,q,j)$ has $(q-j-1)$ extra queries at the end. An adversary $\mathcal{A}'$ for $\mathsf{RoRC}'_{\mathcal{A}',\Pi,b}(n,j+1)$ can simulate $\mathsf{Dist}_{\mathcal{A},\Pi,b}(n,q,j)$ for adversary $\mathcal{A}$ by forwarding each of $\mathcal{A}$'s first $(j+1)$ queries to the challenger in $\mathsf{RoRC}'_{\mathcal{A}',\Pi,b}(n,j+1)$, and similarly forward the responses from the challenger back to $\mathcal{A}$. For the additional $(q-j-1)$ queries in the end, $\mathcal{A}'$ can simply respond by drawing random ciphertexts from $\mathcal{C}_n$. $\mathcal{A}'$ will output whatever is output by $\mathcal{A}$.

Notice that adversary $\mathcal{A}'$ does *not* require any additional memory space besides the space used by adversary $\mathcal{A}$. All that $\mathcal{A}'$ needs to do is to forward $\mathcal{A}$'s queries and the challenger's responses, and to sample random ciphertexts from $\mathcal{C}_n$. These operations do not require $\mathcal{A}'$ to store any persistent states.

Therefore, we have

$$
\begin{aligned}
&|\Pr[\mathsf{Dist}_{\mathcal{A},\Pi,0}(n,q,j)=1]-\Pr[\mathsf{Dist}_{\mathcal{A},\Pi,1}(n,q,j)=1]| \\
\leq &\left|\Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,0}(n,j+1)=1]-\Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,1}(n,j+1)=1]\right|.
\end{aligned}
$$

Bringing all these parts together, assuming that the encryption scheme Π is $(S(n), Q(n), \epsilon)$-secure yields

$$
\begin{aligned}
&|\Pr[\mathsf{RoRC}_{\mathcal{A},\Pi,0}(n,q)=1]-\Pr[\mathsf{RoRC}_{\mathcal{A},\Pi,1}(n,q)=1]| \\
=&\left|\Pr_{h\leftarrow H_q}[D(h)=1]-\Pr_{h\leftarrow H_0}[D(h)=1]\right| \\
\leq &q\left|\Pr_{h\leftarrow H_{j+1}}[D(h)=1]-\Pr_{h\leftarrow H_j}[D(h)=1]\right| \\
=&q\,|\Pr[\mathsf{Dist}_{\mathcal{A},\Pi,0}(n,q,j)=1]-\Pr[\mathsf{Dist}_{\mathcal{A},\Pi,1}(n,q,j)=1]| \\
\leq &q\,|\Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,0}(n,j+1)=1]-\Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,1}(n,j+1)=1]| \\
\leq &q\epsilon \leq Q(n)\epsilon.
\end{aligned}
$$

Therefore, Π is $(S(n)), Q(n), Q(n)\epsilon)$-secure under the RoRC setting.

□

Theorem 3.3.1. *For any $C < \frac{1}{20}$, there exists $\alpha > 0$, s.t. the bit encryption scheme from parity learning is $(Cn^2, 2^{\alpha n}, O(2^{-\alpha n/2}))$-secure under the RoRC' setting.*

Proof. We prove this result by reducing a parity learning game to an RoRC' game.

To start off, we consider a weaker variant of the parity learning game described in Lemma

3.3.1, denoted as $\mathsf{PL}_{\mathcal{A},b}(n,q)$:

- The challenger's input is a bit $b \in \{0,1\}$.
- The challenger chooses a random $\mathbf{k} \in \{0,1\}^n$.
- The challenger streams $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, ..., $(\mathbf{x}_{q-1}, y_{q-1})$, where $\mathbf{x}_i$ is uniformly distributed over $\{0,1\}^n$ and $y_i = \mathbf{x}_i \cdot \mathbf{k}$ for all i.
- The challenger sends $(\mathbf{x}_q, y_q)$, where $\mathbf{x}_q$ is uniformly distributed over $\{0,1\}^n$ and:
  - If $b = 0$, $y_q = \mathbf{x}_q \cdot \mathbf{k}$.
  - If $b = 1$, y_q is a random bit.
- Finally, $\mathcal{A}$ outputs a guess b' for b.

We now show how we can use an adversary $\mathcal{A}$ for $\mathsf{RoRC}'_{\mathcal{A},\Pi,b}(n,q)$ to build an adversary $\mathcal{A}'$ for $\mathsf{PL}_{\mathcal{A}',b}(n,q)$. The adversary $\mathcal{A}'$ works as follows:

- Simulate for $\mathcal{A}$ an $\mathsf{RoRC}'_{\mathcal{A},\Pi,b}(n,q)$ game.
- For every query m_i submitted by $\mathcal{A}$, respond with $(\mathbf{x}_i, y_i + m_i)$ where $\mathbf{x}_i$ and y_i come from the i-th pair of the $\mathsf{PL}_{\mathcal{A}',b}(n,q)$ game.
- If the adversary $\mathcal{A}$ outputs 0, output 0. Otherwise, output 1.

This should be easily verifiable. First, notice that $\mathcal{A}'$ faithfully simulates $\mathsf{RoRC}'_{\mathcal{A},\Pi,b}(n,q)$. For $1 \leq i \leq q-1$, $\mathcal{A}$ receives $(\mathbf{x}_i, y_i + m_i) = (\mathbf{x}_i, \mathbf{x}_i \cdot \mathbf{k} + m_i)$, which is a valid encryption of m_i. Also, for the last query m_q, $\mathcal{A}$ receives either $(\mathbf{x}_q, y_q + m_q) = (\mathbf{x}_q, \mathbf{x}_q \cdot \mathbf{k} + m_q)$, i.e. a valid encryption, or $(\mathbf{x}_q, y_q + m_q)$ for a random bit y_q, i.e. a random ciphertext. Secondly,

if $\mathcal{A}$ outputs 0, that implies $(\mathbf{x}_q, y_q + m_q) = \mathsf{Enc}(\mathbf{k}, m_q) = (\mathbf{x}_q, \mathbf{x}_q \cdot \mathbf{k} + m_q)$, and hence $y_q = \mathbf{x}_q \cdot \mathbf{k}$ and $\mathcal{A}'$ should output 0. Lastly, if $\mathcal{A}$ outputs 1, we have $y_q + m_q$ being a random bit. Since m_q is fixed, we have y_q a random bit and hence $\mathcal{A}'$ should output 1.

This yields

$$\begin{aligned} &|\Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,0}(n,q) = 1] - \Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,1}(n,q) = 1]| \\ \leq &|\Pr[\mathsf{PL}_{\mathcal{A},0}(n,q) = 1] - \Pr[\mathsf{PL}_{\mathcal{A},1}(n,q) = 1]| \,. \end{aligned}$$

Let $\beta = |\Pr[\mathsf{PL}_{\mathcal{A},0}(n,q) = 1] - \Pr[\mathsf{PL}_{\mathcal{A},1}(n,q) = 1]|$. Then we have an algorithm that distinguishes between $(\mathbf{x}_q, y_q = \mathbf{x}_q \cdot \mathbf{k})$ and $(\mathbf{x}_q, y_q \leftarrow \{0,1\})$ with probability $(1+\beta)/2$, i.e. it outputs 0 if y_q is a valid inner product and 1 if it is random. This can be easily converted into an algorithm that given $\mathbf{x}_q$, outputs $\mathbf{x}_q \cdot \mathbf{k}$ with probability $(1+\beta)/2$ (simply XOR the output of the previous algorithm with y_q). Let f be the function computed by this algorithm. Then for given $\mathbf{x}_q \in \{0,1\}^n$ and unknown $\mathbf{k} \in \{0,1\}^n$, $f(\mathbf{x}_q) = \langle \mathbf{k}, \mathbf{x}_q \rangle$ with probability $(1+\beta)/2$. By applying Lemma 3.3.2, there is an algorithm that runs in time $O(n^2\beta^{-4}\log n)$ and outputs $\mathbf{k}$ with probability at least $\Omega(\beta^2)$.

Recall from Lemma 3.3.1 that for any $C < 1/20$, there is a positive α such that any potentially *computationally unbounded* algorithm that uses up to Cn^2 memory bits and has access to at most $2^{\alpha n}$ $(\mathbf{x}_i, y_i)$ pairs can output $\mathbf{k}$ with probability at most $O(2^{-\alpha n})$. Therefore, for adversaries that are space-bounded by Cn^2 bits and submit at most $2^{\alpha n}$ queries, $\Omega(\beta^2) \leq O(2^{-\alpha n})$. And hence $\beta = O(2^{-\alpha n/2})$

Therefore, for any $C < 1/20$, there is a positive α such that for all adversaries that use at

most Cn^2 memory bits and at most $2^{\alpha n}$ queries ($q \leq 2^{\alpha n}$), we have

$$|\Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,0}(n,q) = 1] - \Pr[\mathsf{RoRC}'_{\mathcal{A},\Pi,1}(n,q) = 1]| \leq \beta = O(2^{-\alpha n/2}),$$

i.e. the scheme is $(Cn^2, 2^{\alpha n}, O(2^{-\alpha n/2}))$-secure under the RoRC' setting as desired.

□

Corollary 3.3.1 (RoRC Security of the Bit Encryption Scheme from Parity Learning)**.** *For any $C < \frac{1}{20}$, there exists $\alpha > 0$, s.t. the bit encryption scheme from parity learning is $(Cn^2, 2^{\alpha n/4}, O(2^{-\alpha n/2}))$-secure under the RoRC' setting (here we further bound the number of queries to $\alpha n/4$ instead of αn). By Lemma 3.3.3, this scheme is also $(Cn^2, 2^{\alpha n/4}, 2^{\alpha n/4} \cdot O(2^{-\alpha n/2}) = O(2^{-\alpha n/4}))$-secure under the RoRC setting. Put another way, for any $C < \frac{1}{20}$, there exists $\alpha' (= \alpha/4) > 0$, s.t. the bit encryption scheme from parity learning is $(Cn^2, 2^{\alpha' n}, O(2^{-\alpha' n}))$-secure under the RoRC setting.*

3.4 Encrypt Zero Protocols

In this section, we introduce two constructions of the Encrypt Zero Protocol. They both have the same goal: to give one party, the *Keeper*, a random key $\mathbf{s}$, and the other party, known as the *Recorder*, several encryptions of 0 under the key $\mathbf{s}$. They differ in that the simple construction is only secure against honest-but-curious Keepers, while the enhanced construction is secure even against malicious Keepers.

Before we jump into the constructions, we first define an Encrypt Zero Protocol and its security properties.

An Encrypt Zero Protocol Π involves two parties, a Keeper $\mathcal{K}$ and a Recorder $\mathcal{R}$. The

protocol takes three parameters n, $m = O(n)$ and λ, and produces $(\mathbf{s}, \{c_1, c_2, \ldots, c_\lambda\}, \mathsf{trans})$, where $\mathbf{s}$ is a random key output by $\mathcal{K}$, $\{c_1, c_2, \ldots, c_\lambda\}$ is a set of ciphertexts output by $\mathcal{R}$, and trans is the transcript of their communication.

The correctness of an Encrypt Zero Protocol requires that the set of ciphertexts output by $\mathcal{R}$ are encryptions of zero under the key $\mathbf{s}$ output by $\mathcal{K}$. Put formally, we require that $\mathsf{Dec}(\mathbf{s}, c_i) = 0$ for all i.

Now, we define two desired security properties for the Encrypt Zero Protocol, namely Keeper security and Recorder security.

The security of the Keeper ensures that the Keeper's key $\mathbf{s}$ has enough min-entropy conditioned on the Recorder's view $\mathsf{view}_{\mathcal{R}}$.

Definition 3.4.1 (Keeper Security)**.** *Let the view of the Recorder be* $\mathsf{view}_{\mathcal{R}}$*, we say that a protocol* Π *is* $(S(n), h)$*-secure for the Keeper if for all Recorders* $\mathcal{R}$ *that use up to* $S(n)$ *memory bits,*

$$H_\infty(\mathbf{s}|\mathsf{view}_{\mathcal{R}}) \geq h.$$

The security of the Recorder ensures that the Keeper learns nothing about $c_1, c_2, \ldots, c_\lambda$ (except that they are encryptions of zero).

For an honest-but-curious Keeper $\mathcal{K}$, this means that given all the Keeper's randomness and the transcript produced by the protocol, it is hard to distinguish the output ciphertexts $(c_1, c_2, \ldots, c_\lambda)$ from some random ciphertexts that encrypt zero.

Definition 3.4.2 (Recorder Security with Honest-but-Curious Keeper)**.** *Let* $C = \{c_1, c_2, \ldots, c_\lambda\}$ *be the ciphertexts output by* $\mathcal{R}$ *at the end of the protocol, and* $C' = \{c'_1, c'_2, \ldots, c'_\lambda\}$ *where* $c'_i \leftarrow \mathsf{Enc}(\mathbf{s}, 0)$ *be fresh encryptions of zero under the key* $\mathbf{s}$*. Let* $\mathsf{state}_{\mathcal{K}}$ *consist of all the random*

coins used by $\mathcal{K}$ together with trans*. Given the Keeper's state* $\mathsf{state}_{\mathcal{K}}$*, the key* $\mathbf{s}$*, the protocol* Π *is* ϵ*-secure for the Recorder if for any distinguisher D,*

$$\left| \Pr_{c \leftarrow C}[D_{\mathsf{state}_{\mathcal{K}},\mathbf{s}}(c) = 1] - \Pr_{c \leftarrow C'}[D_{\mathsf{state}_{\mathcal{K}},\mathbf{s}}(c) = 1] \right| \leq \epsilon.$$

In the case of a malicious Keeper $\mathcal{K}^*$ who can have arbitrary behavior, we let $\mathsf{state}_{\mathcal{K}^*}$ be the state of $\mathcal{K}^*$ at the end of the protocol. Notice that regardless of the possible behaviors that $\mathcal{K}^*$ could have, it is constrained to the state that it has stored at the end of the protocol. It has no additional information besides what it has stored in $\mathsf{state}_{\mathcal{K}^*}$.

Definition 3.4.3 (Recorder Security with Malicious Keeper)**.** *Let* $C = \{c_1, c_2, \ldots, c_\lambda\}$ *be the ciphertexts output by* $\mathcal{R}$ *at the end of the protocol, and* $C' = \{c'_1, c'_2, \ldots, c'_\lambda\}$ *where* $c'_i \leftarrow \mathsf{Enc}(\mathbf{s}, 0)$ *be fresh encryptions of zero under the key* $\mathbf{s}$*. Given the malicious Keeper's state* $\mathsf{state}_{\mathcal{K}^*}$*, the key* $\mathbf{s}$*, the protocol* Π *is* ϵ*-secure for the Recorder if for any distinguisher D,*

$$\left| \Pr_{c \leftarrow C}[D_{\mathsf{state}_{\mathcal{K}^*},\mathbf{s}}(c) = 1] - \Pr_{c \leftarrow C'}[D_{\mathsf{state}_{\mathcal{K}^*},\mathbf{s}}(c) = 1] \right| \leq \epsilon.$$

3.4.1 Simple Encrypt Zero Protocol

Here we present the Simple Encrypt Zero Protocol, which achieves Keeper Security and Recorder security against honest-but-curious Keeper. The main idea here is simple: the Keeper will stream a sequence of ciphertexts which are encryptions of zero, and and Recorder will obtain fresh encryptions of zero by taking random subset-sums of the ciphertexts received.

Construction 3.4.1 (Simple Encrypt Zero Protocol)**.** *A Simple Encrypt Zero Protocol in-*

stance $\mathtt{EZ}(n, m, \lambda)$ *for the Keeper* $\mathcal{K}$ *and the Recorder* $\mathcal{R}$ *proceeds as follows:*

- $\mathcal{K}$ *chooses a random key* $\mathbf{k} \in \{0,1\}^n$*, and* $\mathcal{R}$ *chooses a random secret matrix* $\boldsymbol{\Sigma} \in \{0,1\}^{\lambda \times m}$*.*
- $\mathcal{K}$ *streams encryptions* $(\mathbf{r}_i, a_i = \mathbf{r}_i \cdot \mathbf{k} + 0)$ *to* $\mathcal{R}$*, for* $i = 1, 2, \ldots, m$ *and random* $\mathbf{r}_i \in \{0,1\}^n$*.*
- $\mathcal{R}$ *maintains matrix* $\boldsymbol{\Psi} \in \{0,1\}^{\lambda \times n}$ *and column vector* $\boldsymbol{\kappa} \in \{0,1\}^{\lambda}$*. Each row of* $(\boldsymbol{\Psi}|\boldsymbol{\kappa})$ *will be a random subset-sum of the encryptions sent by* $\mathcal{K}$*, with each subset-sum chosen according to* $\boldsymbol{\Sigma}$*.* $\boldsymbol{\Psi}$ *and* $\boldsymbol{\kappa}$ *will be computed on the fly. Specifically, when encryption* $(\mathbf{r}_i, a_i)$ *comes in,* $\mathcal{R}$ *will update* $\boldsymbol{\Psi}$ *to be* $\boldsymbol{\Psi} + \boldsymbol{\sigma}_i \cdot \mathbf{r}_i$ *and* $\boldsymbol{\kappa}$ *to be* $\boldsymbol{\kappa} + \boldsymbol{\sigma}_i a_i$*. Here,* $\boldsymbol{\sigma}_i$ *is the i-th column of* $\boldsymbol{\Sigma}$*, and* $\mathbf{r}_i$ *is interpreted as a row vector.*
- *At the end of the protocol,* $\mathcal{K}$ *outputs its key* $\mathbf{s} = \mathbf{k}$*, and* $\mathcal{R}$ *outputs* $(\boldsymbol{\Psi}|\boldsymbol{\kappa})$*, whose rows are the ciphertexts* $c_1, c_2, \ldots, c_\lambda$*.*

Remark 3.4.1. *For the ease of analysis, we combine all the encryptions sent together, and denote* $\mathbf{R} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \cdots \\ \mathbf{r}_m \end{bmatrix} \in \{0,1\}^{m \times n}$*, and* $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_m \end{bmatrix} \in \{0,1\}^m$*. This gives us*

$$\mathbf{a} = \mathbf{R} \cdot \mathbf{k}.$$

Correspondingly, notice that $\mathcal{R}$ *is essentially recording* $\boldsymbol{\Sigma}$*,* $\boldsymbol{\Psi} = \boldsymbol{\Sigma} \cdot \mathbf{R}$ *and* $\boldsymbol{\kappa} = \boldsymbol{\Sigma} \cdot \mathbf{a} = \boldsymbol{\Sigma} \cdot \mathbf{R} \cdot \mathbf{k} = \boldsymbol{\Psi} \cdot \mathbf{k}$*.*

It is easy to verify that the rows of $(\boldsymbol{\Psi}|\boldsymbol{\kappa})$ are encryptions of 0 under the key $\mathbf{s} = \mathbf{k}$, as they are simply sums of encryptions of 0 under $\mathbf{s}$ and by the additive homomorphism of Raz's encryption scheme they also must encrypt 0. Therefore, this construction meets the correctness requirement for an Encrypt Zero Protocol.

Next, we show that this construction achieves Keeper security and Recorder security against honest-but-curious Keepers.

Theorem 3.4.1 (Keeper Security of EZ)**.** *The Simple Encrypt Zero Protocol is* $(Cn^2, \Omega(\alpha n))$*-secure for the Keeper, for some* $C < \frac{1}{20}$ *and* α *dependent on* C*.*

Proof. This follows directly from Lemma 3.3.1. Here $\mathsf{view}_{\mathcal{R}}$ essentially contains m pairs of $(\mathbf{r}_i, a_i)$, where $a_i = \mathbf{r}_i \cdot \mathbf{s}$ for $i = 1, 2, \ldots, m$ and random $\mathbf{r}_i \leftarrow \{0,1\}^n$. For adversaries space-bounded to Cn^2 memory bits for some $C < \frac{1}{20}$ and α dependent on C, by applying Lemma 3.3.1, we get that the probability of an adversary outputting $\mathbf{s}$ is no more than $O(2^{-\alpha n})$. Hence, the average min-entropy of $\mathbf{s}$ conditioned on $\mathsf{view}_{\mathcal{R}}$ is $\Omega(\alpha n)$.

□

Theorem 3.4.2 (Recorder Security of EZ)**.** *The Simple Encrypt Zero Protocol with parameter* $m = 2n$ *and an honest-but-curious Keeper is* $O(2^{-n})$*-secure for the Recorder.*

Proof. Since the Keeper is honest and follows the protocol, $\mathbf{R}$ is a random $m \times n$ matrix. For $m = 2n$, we have $\mathbf{R}$ being a random $2n \times n$ matrix, which is full rank with probability $1 - O(2^{-n})$. Notice that if $\mathbf{R}$ is full rank, given that $\boldsymbol{\Sigma}$ is a random matrix conditioned on the Keeper's state $\mathsf{state}_{\mathcal{K}}$ and $\mathbf{s}$, $\boldsymbol{\Psi} = \boldsymbol{\Sigma} \cdot \mathbf{R}$ is also a random matrix conditioned on $\mathsf{state}_{\mathcal{K}}$ and $\mathbf{s}$.

In this way, conditioned on $\mathsf{state}_{\mathcal{K}}$ and $\mathbf{s}$, $(\boldsymbol{\Psi}|\boldsymbol{\kappa})$ contains *random* encryptions of 0. Therefore, by definition, these encryptions $\{c_1, \ldots, c_\lambda\}$ cannot be distinguished from $\{c'_1, \ldots, c'_\lambda\}$

where c'_i is a random encryption of 0. Hence, the probability of distinguishing C from C' is bounded by the probability that $\mathbf{R}$ is *not* full rank, which is $O(2^{-n})$. Thus we have

$$\left| \Pr_{c \leftarrow C}[D_{\mathsf{trans},\mathsf{s}}(c) = 1] - \Pr_{c \leftarrow C'}[D_{\mathsf{trans},\mathsf{s}}(c) = 1] \right| \leq 2O(2^{-n}) = O(2^{-n})$$

as desired.

□

Kindly notice that this simple construction of an Encrypt Zero protocol is only secure for the Recorder if the Keeper is honest. For malicious Keepers, they could, for example, generate the matrix $\mathbf{R}$ with bad randomness so that it is very likely to be low rank.

One way to tackle this is to have the random matrix $\mathbf{R}$ generated and streamed by a trusted third party, which is a common practice in much of the prior work in the bounded storage model. However, if we do not wish to rely on a trusted third party (notice that the model without a trusted third party is stronger than one with a trusted third party), we show in the following subsection how we can tweak our simple construction to have Recorder security even against malicious Keepers.

3.4.2 Enhanced Encrypt Zero Protocol

In the Enhanced Encrypt Zero Protocol construction, we tweak the simple construction slightly to account for malicious Keepers.

Construction 3.4.2 (Enhanced Encrypt Zero Protocol)**.** *An Enhanced Encrypt Zero Protocol instance* $\mathsf{EZ}^{+}(n, m, \lambda)$ *with the Keeper* $\mathcal{K}$ *and the Recorder* $\mathcal{R}$ *proceeds as follows:*

- *$\mathcal{K}$ chooses a random key $\mathbf{k} \in \{0,1\}^n$ and an independent random secret $\mathbf{s} \in \{0,1\}^m$. $\mathcal{R}$ chooses a random secret matrix $\mathbf{\Sigma} \in \{0,1\}^{\lambda \times m}$.*

- *$\mathcal{K}$ streams random encryptions of the bits in $\mathbf{s}$. Namely, in matrix form, $\mathcal{K}$ sends $(\mathbf{R}, \mathbf{a} = \mathbf{R} \cdot \mathbf{k} + \mathbf{s})$ for random $\mathbf{R} \in \{0,1\}^{m \times n}$.*

- *$\mathcal{R}$ maintains matrix $\mathbf{\Psi} = \mathbf{\Sigma} \cdot \mathbf{R}$ and column vector $\boldsymbol{\kappa} = \mathbf{\Sigma} \cdot \mathbf{a}$.*

- *$\mathcal{K}$ sends its key $\mathbf{k}$ in the clear, and $\mathcal{R}$ uses that to compute $\boldsymbol{\phi} = \boldsymbol{\kappa} - \mathbf{\Psi} \cdot \mathbf{k}$.*

- *$\mathcal{K}$ outputs $\mathbf{s}$ as its key, and $\mathcal{R}$ outputs $(\mathbf{\Sigma}|\boldsymbol{\phi})$, whose rows are the ciphertexts $c_1, c_2, \ldots, c_\lambda$.*

Notice that $\boldsymbol{\phi} = \boldsymbol{\kappa} - \mathbf{\Psi} \cdot \mathbf{k} = \mathbf{\Sigma} \cdot \mathbf{s}$, and hence the rows of $(\mathbf{\Sigma}|\boldsymbol{\phi})$ are indeed encryptions of 0 using key $\mathbf{s}$, as desired in the correctness property.

Theorem 3.4.3 (Keeper Security of EZ^+)**.** *The Simple Encrypt Zero Protocol is $(Cn^2, \Omega(n))$-secure for the Keeper, for some $C < \frac{1}{20}$ and α dependent on C.*

Proof. First, notice that before the Keeper sends over $\mathbf{k}$, the two distributions $(\mathbf{s}, \mathbf{R}, \mathbf{R} \cdot \mathbf{k} + \mathbf{s})$ and $(\mathbf{s}, \mathbf{R}, \mathbf{R} \cdot \mathbf{k} + \mathbf{s}')$ for random $\mathbf{s}' \in \{0,1\}^m$ are statistically indistinguishable, due to the RoRC security of Raz's encryption scheme.

Now, notice that in the second distribution, the probability the Recorder can guess $\mathbf{s}$ is 2^{-m}. In this case, if it later receives $\mathbf{k}$, the probability it guesses $\mathbf{s}$ is still at most 2^{n-m}, which is 2^{-n}.

Now, we use the following simple fact: suppose two distributions X, Y are ϵ-close. Then there is a procedure P which first samples $x \leftarrow X$, and then based which x it samples, it may replace x with a different sample x'. P satisfies the property that (1) its output distribution is identical to Y, and (2) the probability it re-samples is ϵ.

We use this simple fact by assigning X to $(\mathbf{s}, \mathbf{R}, \mathbf{R} \cdot \mathbf{k} + \mathbf{s}')$ for random $\mathbf{s}' \in \{0,1\}^m$ and Y to $(\mathbf{s}, \mathbf{R}, \mathbf{R} \cdot \mathbf{k} + \mathbf{s})$.

Now consider the probability of guessing $\mathbf{s}$. In the case X, we know it is 2^{-n}. So if we consider Y sampled from P, we know that the probability of guessing $\mathbf{s}$ in the non-replacing case is 2^{-n}. But the replacing case only happens with probability ϵ, meaning overall the probability of outputting $\mathbf{s}$ is at most $\epsilon + 2^{-n}$.

□

Theorem 3.4.4 (Recorder Security of EZ^+)**.** *The Enhanced Encrypt Zero Protocol with parameter $m = 2n$ and any possibly malicious Keeper $\mathcal{K}^*$ is perfectly secure for the Recorder.*

Proof. Notice that regardless of the Keeper's state $\mathsf{state}_{\mathcal{K}^*}$ (even if one of a malicious Keeper), $\square$ is always random conditioned on $\mathsf{state}_{\mathcal{K}^*}$ and $\mathbf{s}$, since it is solely sampled by the Recorder. Therefore, $(\boldsymbol{\Sigma}|\boldsymbol{\phi})$ is already random encryptions of 0 conditioned on $\mathsf{state}_{\mathcal{K}^*}$ and $\mathbf{s}$. Hence, to distinguish it from other random encryptions of 0, one can do no better than a random guess. Thus, the advantage that any distinguisher D could have in distinguishing C and C' is 0 as desired.

□

3.5 Two-Party Key-Agreement Protocol

Consider a pair of interactive PPT algorithms $\Pi = (\mathsf{A}, \mathsf{B})$. Each of A, B take n as input. We will let $(a, b, \mathsf{trans}) \leftarrow \Pi(n)$ denote the result of running the protocol on input n. Here, a is the output of A, b the output of B, and trans is the transcript of their communication.

A two-party key-agreement protocol is a protocol $\Pi = (\mathsf{A}, \mathsf{B})$ with the correctness property that $\Pr[a = b] = 1$. In this case, we will define $\hat{k} = a = b$ and write $(\hat{k}, \mathsf{trans}) \leftarrow \Pi(n)$.

Additionally, we will require eavesdropping security:

Definition 3.5.1 (Eavesdropping Security of Two-Party Key-Agreement Protocol)**.** *We say that* Π *is* $(S(n), \epsilon)$*-secure if for all adversaries* $\mathcal{A}$ *that use at most* $S(n)$ *memory bits,*

$$\begin{aligned} &|\Pr[\mathcal{A}(\hat{k}, \mathsf{trans}) = 1 : (\hat{k}, \mathsf{trans}) \leftarrow \Pi(n)] \\ &- \Pr[\mathcal{A}(k', \mathsf{trans}) = 1 : k' \leftarrow \mathcal{K}_n, (k, \mathsf{trans}) \leftarrow \Pi(n)]| \leq \epsilon. \end{aligned}$$

In this section we demonstrate how we can use the Simple Encrypt Zero Protocol to implement a two-party key-agreement protocol. For simplicity, we consider a key space of one single bit.

Construction 3.5.1 (Two-Party Key-Agreement Protocol)**.** *For two parties* $\mathcal{P}$ *and* $\mathcal{Q}$ *trying to derive a shared key* $\hat{k} \in \{0,1\}$*, they will first run a Simple Encrypt Zero Protocol* $\mathsf{EZ}(n, m, \lambda = 1)$ *with* $\mathcal{P}$ *as the Keeper and* $\mathcal{Q}$ *as the Recorder. At the end of the* EZ *protocol,* $\mathcal{P}$ *gets a key* $\mathbf{s}$*, and* $\mathcal{Q}$ *gets an encryption of* 0 *using* $\mathbf{s}$*, namely* $(\mathbf{\Psi}|\boldsymbol{\kappa})$ *(notice that* $\boldsymbol{\kappa}$ *is of dimension* $\lambda \times 1$*, and hence is a single bit here). To derive a shared key,* $\mathcal{Q}$ *sends* $\mathbf{\Psi}$ *to* $\mathcal{P}$*. The shared key is thus* $\boldsymbol{\kappa}$*, which is known to* $\mathcal{Q}$*, and is computable by* $\mathcal{P}$ *as* $\boldsymbol{\kappa} = \mathbf{\Psi} \cdot \mathbf{s}$*.*

Remark 3.5.1. *For key spaces* $\{0,1\}^d$*, we can simply tune the protocol to use* $\lambda = d$*, and that will yield a shared key* $\hat{\mathbf{k}} \in \{0,1\}^d$*.*

Theorem 3.5.1. *The two-party key-agreement protocol presented above is* $(Cn^2, O(2^{-\alpha n/2}))$*-secure against eavesdropping adversaries.*

Proof. First, by the Keeper security of the EZ protocol, for adversaries with up to Cn^2 memory bits for some $C < \frac{1}{20}$, $H_\infty(\mathbf{s}|\mathsf{view}_\mathcal{R}) \geq \Omega(\alpha n)$. Subsequently, $H_\infty(\mathbf{\Psi}, \mathbf{s}|\mathsf{view}_\mathcal{R}) \geq$

$\Omega(\alpha n)$. Let $H : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ compute the inner product. Using the fact that the inner product is a universal hash function and applying Lemma 3.2.1, we have

$$(H(\mathbf{\Psi}, \mathbf{s}), H, \mathsf{view}_{\mathcal{R}}) \approx_{\epsilon/2} (U_1, U_d, \mathsf{view}_{\mathcal{R}}),$$

where $1 + 2\log(1/\epsilon) = \Omega(\alpha n)$. Solving for ϵ yields that $\epsilon = O(2^{-\alpha n/2})$, i.e. an adversary has advantage at most $O(2^{-\alpha n/2})$ in distinguishing $H(\mathbf{\Psi}, \mathbf{s})$ and U_1. Recall that in the eavesdropping security game for Two-Party Key-Agreement Protocols, the adversary need to distinguish between actual derived keys $\hat{k} = \mathbf{\Psi} \cdot \mathbf{s}$ from random k' sampled directly from the key space $\{0,1\}$. Observe that $H(\mathbf{\Psi}, \mathbf{s}) = \mathbf{\Psi} \cdot \mathbf{s} = \hat{k}$, and k' is drawn from U_1. Therefore, we have

$$\begin{aligned} &|\Pr[\mathcal{A}(\hat{k}, \mathsf{trans}) = 1 : (\hat{k}, \mathsf{trans}) \leftarrow \Pi(n)] \\ &- \Pr[\mathcal{A}(k', \mathsf{trans}) = 1 : k' \leftarrow \mathcal{K}_n, (k, \mathsf{trans}) \leftarrow \Pi(n)]| \leq \epsilon = O(2^{-\alpha n/2}) \end{aligned}$$

as desired.

□

3.6 Bit Commitment Scheme

Let n and λ be security parameters. A bit commitment scheme Π consists of a tuple of algorithm $(\texttt{Commit}, \texttt{Reveal}, \texttt{Verify})$ for a committer $\mathcal{C}$ and a verifier $\mathcal{V}$.

- The `Commit` algorithm is run by the committer, and it takes as input the security parameter n and a bit b to be committed to. A transcript of the communication, a com-

mitter state, and a verifier state $(\mathsf{trans}, \mathsf{state}_{\mathcal{C}}, \mathsf{state}_{\mathcal{V}}) \leftarrow \texttt{Commit}(n, \lambda, b)$ is output by the `Commit` algorithm.

- The `Reveal` algorithm is also run by the committer, and it takes as input a committer state $\mathsf{state}_{\mathcal{C}}$ and a bit b'. It outputs a revealing, denoted as x, together with the committed bit b'.

- The `Verify` algorithm is run by the Verifier and takes input a verifier state $\mathsf{state}_{\mathcal{V}}$ and outputs of a `Reveal` algorithm, (x, b'). It outputs a bit u.

There are two desired security properties for a bit commitment scheme, namely hiding and binding. We will give out formal definitions below.

The hiding property of a bit commitment scheme essentially states that the committed bit b should be hidden from the Verifier given the Verifier's view after the `Commit` algorithm. Notice that the Verifier's view after the `Commit` algorithm consists of exactly trans and $\mathsf{state}_{\mathcal{V}}$. Put formally:

Definition 3.6.1 (Hiding Property of a Bit Commitment Scheme)**.** *For some given security parameters n, λ and a bit b, let $(\mathsf{trans}, \mathsf{state}_{\mathcal{C}}, \mathsf{state}_{\mathcal{V}}) \leftarrow \texttt{Commit}(n, \lambda, b)$, we say that the bit commitment scheme is $(S(n), \epsilon)$-hiding if for all Verifiers $\mathcal{V}$ with up to $S(n)$ memory bits,*

$$(b, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}) \approx_{\epsilon} (r, \mathsf{trans}, \mathsf{state}_{\mathcal{V}})$$

for random r uniformly sampled from $\{0, 1\}$.

The binding property of a bit commitment scheme essentially requires that a committer is not able to open a commitment to both 0 and 1. Notice that this applies to all committers,

who can be potentially malicious. A malicious committer $\mathcal{A}$ can run an arbitrary Commit^* procedure, which has no guarantees except that it produces some $(\mathsf{trans}, \mathsf{state}_{\mathcal{A}}, \mathsf{state}_{\mathcal{V}})$. Note that this Commit^* procedure does not necessarily commit to a bit b, so it does not take b as a parameter.

Definition 3.6.2 (Binding Property of a Bit Commitment Scheme)**.** *Let $\mathcal{A}$ be an adversary. $\mathcal{A}$ plays the following game $\mathsf{Binding}_{\mathcal{A},\Pi}(n, \lambda)$ for some given security parameters n and λ:*

- *The adversary $\mathcal{A}$ runs an arbitrary commit procedure (potentially malicious) $\mathtt{Commit}^*(n, \lambda)$ with an honest Verifier $\mathcal{V}$ and produces $(\mathsf{trans}, \mathsf{state}_{\mathcal{A}}, \mathsf{state}_{\mathcal{V}})$.*
- *The adversary produces $(x_0, 0)$ and $(x_1, 1)$.*
- *The game outputs 1 if both $\mathtt{Verify}(\mathsf{state}_{\mathcal{V}}, (x_0, 0))$ and $\mathtt{Verify}(\mathsf{state}_{\mathcal{V}}, (x_1, 1))$ output 1, and 0 otherwise.*

We say that Π is ϵ-binding if for all adversary $\mathcal{A}$

$$\Pr[\mathsf{Binding}_{\mathcal{A},\Pi}(n, \lambda) = 1] \leq \epsilon.$$

Now we present the construction for a bit commitment scheme using the Enhanced Encrypt Zero Protocol.

Construction 3.6.1 (Bit Commitment Scheme from Parity Learning)**.** *For security parameters n, λ and committer input bit b, we construct the bit commitment scheme by specifying each of the $(\mathtt{Commit}, \mathtt{Reveal}, \mathtt{Verify})$ algorithms.*

- $\mathtt{Commit}(n, b)$*: Runs the Enhanced Encrypt Zero Protocol $\mathtt{EZ}^+(n, 2n, \lambda)$ with $\mathcal{C}$ as the Keeper and $\mathcal{V}$ as the Recorder. Set* trans *to be the transcript of the $\mathtt{EZ}^+$ protocol, $\mathsf{state}_{\mathcal{C}}$ to*

be the output of $\mathcal{C}$ after the $\mathtt{EZ}^+$ *protocol, i.e. a secret key* $\mathbf{s}$*, and* $\mathsf{state}_{\mathcal{V}}$ *to be the output of* $\mathcal{V}$ *after the* $\mathtt{EZ}^+$ *protocol, namely* $(\boldsymbol{\Sigma}|\boldsymbol{\phi})$*, which contains multiple encryptions of* 0 *under the key* $\mathbf{s}$*. Additionally, samples random* $\boldsymbol{\gamma} \in \{0,1\}^{2n}$*, and sends* $(\boldsymbol{\gamma}, c = \boldsymbol{\gamma} \cdot \mathbf{s} + b)$ *to the Verifier (notice that this also gets appended to* trans*).*

- $\mathtt{Reveal}(\mathsf{state}_{\mathcal{C}}, b')$*: Outputs* $(\mathbf{x}, b') = (\mathbf{s}, b')$.
- $\mathtt{Verify}(\mathsf{state}_{\mathcal{V}}, \mathbf{x}, b')$*: Checks that* $\boldsymbol{\phi} = \boldsymbol{\Sigma} \cdot \mathbf{x}$*, and that* $c = \boldsymbol{\gamma} \cdot \mathbf{x} + b'$*. If any of the checks fail, output* 0*; otherwise, output* 1.

Theorem 3.6.1. *The bit commitment construction above is* $(Cn^2, O(2^{-n/2}))$*-hiding for some* $C < 1/20$.

Proof. First, by the Keeper security of the $\mathtt{EZ}^+$ protocol, for adversaries with up to Cn^2 memory bits for some $C < \frac{1}{20}$, $H_\infty(\mathbf{s}|\mathsf{view}_{\mathcal{V}}) \geq \Omega(n)$. Recall that $\mathsf{view}_{\mathcal{V}}$ is exactly $(\mathsf{trans}, \mathsf{state}_{\mathcal{V}})$. Subsequently, $H_\infty(\boldsymbol{\gamma}, \mathbf{s}|\mathsf{trans}, \mathsf{state}_{\mathcal{V}}) \geq \Omega(n)$. Let $H : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ compute the inner product. Using the fact that the inner product is a universal hash function and applying Lemma 3.2.1, we have

$$(H(\boldsymbol{\gamma}, \mathbf{s}), H, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}) \approx_{\epsilon/2} (U_1, U_d, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}),$$

where $1 + 2\log(1/\epsilon) = \Omega(n)$. Furthermore, we have

$$(H(\boldsymbol{\gamma}, \mathbf{s}) + c, H, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}) \approx_{\epsilon/2} (U_1 + c, U_d, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}),$$

Solving for ϵ yields that $\epsilon = O(2^{-n/2})$, i.e. an adversary has advantage at most $O(2^{-n/2})$ in distinguishing $H(\boldsymbol{\gamma}, \mathbf{s}) + c$ and $U_1 + c$. Notice that $H(\boldsymbol{\gamma}, \mathbf{s}) + c = \boldsymbol{\gamma} \cdot \mathbf{s} + c = b$, and that

$U_1 + c$ is yet another uniformly random bit $r \leftarrow \{0,1\}$. Therefore, we have

$$(b, H, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}) \approx_{\epsilon/2} (r, U_d, \mathsf{trans}, \mathsf{state}_{\mathcal{V}})$$

for $\epsilon = O(2^{-n/2})$ and r a uniformly random bit. Thus, by

$$(b, \mathsf{trans}, \mathsf{state}_{\mathcal{V}}) \approx_{\epsilon'} (r, \mathsf{trans}, \mathsf{state}_{\mathcal{V}})$$

for $\epsilon' = \frac{1}{2}O(2^{-n/2}) = O(2^{-n/2})$ and r a uniformly random bit, we have shown that the bit commitment scheme presented above is $(Cn^2, O(2^{-n/2}))$-hiding as desired.

□

Theorem 3.6.2. *The bit commitment scheme presented above is* $(2^{-\lambda})$*-binding.*

Proof. We show that the scheme is statistically binding by arguing that the probability that an adversary can win the Binding game is no more than $\frac{1}{2^{\lambda}}$.

Notice that in order for the adversary to win the game, the adversary need to output $(\mathbf{x}_0, 0)$ and $(\mathbf{x}_1, 1)$ that both pass the Verify algorithm. Recall that the Verify Algortihm checks for two things:

- $c = \boldsymbol{\gamma} \cdot \mathbf{x}_0 + 0$ and $c = \boldsymbol{\gamma} \cdot \mathbf{x}_1 + 1$ where c and $\boldsymbol{\gamma}$ are part of the transcipt trans and are stored in the Verifier's state $\mathsf{state}_{\mathcal{V}}$. This leads to that $\boldsymbol{\gamma} \cdot \mathbf{x}_0 \neq \boldsymbol{\gamma} \cdot \mathbf{x}_1$ and hence $\mathbf{x}_0 \neq \mathbf{x}_1$.
- $\boldsymbol{\phi} = \boldsymbol{\Sigma} \cdot \mathbf{x}_0 = \boldsymbol{\Sigma} \cdot \mathbf{x}_1$ where $\boldsymbol{\Sigma}$ and $\boldsymbol{\phi}$ are sampled and computed by the Verfier and stored in $\mathsf{state}_{\mathcal{V}}$. Notice this leads to $\boldsymbol{\Sigma} \cdot (\mathbf{x}_0 - \mathbf{x}_1) = 0$.

Now let $\mathbf{x}' = \mathbf{x}_0 - \mathbf{x}_1$. From $\mathbf{x}_0 \neq \mathbf{x}_1$, we know that $\mathbf{x}' \neq 0$. Therefore, we need to find a non-trivial root for the equation $\mathbf{\Sigma} \cdot \mathbf{x}' = 0$. Recall that by the Recorder's perfect security of the $\mathtt{EZ}^+$ protocol, the matrix $\mathbf{\Sigma}$ stored in $\mathsf{state}_{\mathcal{V}}$ is random conditioned on the Committer's view. For each row of $\mathbf{\Sigma}$, denoted as $\mathbf{\Sigma}_i$ for the i-th row, the probability that $\mathbf{\Sigma}_i \cdot \mathbf{x}' = 0$ is no more than a random guess, i.e. $\frac{1}{2}$. Since to pass the `Verify` algorithm requires $\mathbf{\Sigma} \cdot \mathbf{x}' = 0$, i.e. $\mathbf{\Sigma}_i \cdot \mathbf{x}' = 0$ for all $i = 1, 2, \ldots, \lambda$, and recall that the rows of $\mathbf{\Sigma}$ are independent, the probability that the adversary can find such a $\mathbf{x}'$ is no more than $(\frac{1}{2})^\lambda = \frac{1}{2^\lambda}$.

□

3.7 Oblivious Transfer Protocol

In an oblivious transfer (OT) protocol, one party, the Sender $\mathcal{S}$, has two input bits x_0, x_1, and the other party, known as the Receiver $\mathcal{R}'$ (not to be confused with the Recorder $\mathcal{R}$ in the Encrypt Zero Protocols), has an input bit b. After some communication between the two parties, $\mathcal{R}'$ outputs x_b. The OT protocol requires two security properties, namely Sender security and Receiver security. Sender security dictates that $\mathcal{R}'$ should have no information about x_{1-b}, and Receiver security requires that $\mathcal{S}$ has no information about b.

Before we proceed to our construction of an OT protocol, we first formally define these two security properties.

The security of the Sender ensures that an adversarial Receiver can learn about at most one of x_0 and x_1. In other words, there always exists a b' s.t. the Receiver has no information about $x_{b'}$. Put formally:

Definition 3.7.1 (Sender Security). *An OT protocol is said to be ϵ-secure for the Sender if there*

exists some b' s.t. for any arbitrary distinguisher D and Receiver's view $\mathsf{view}_{\mathcal{R}'}$,

$$\left|\Pr[D_{\mathsf{view}_{\mathcal{R}'}}(x_{b'}) = 1] - \Pr[D_{\mathsf{view}_{\mathcal{R}'}}(r) = 1]\right| \leq \epsilon$$

for a uniformly random bit r.

The security of the Receiver requires that the sender $\mathcal{S}$ has no information about b. In other words, given the view of the Sender, one should not be able to distinguish between b and a random bit r. Put formally:

Definition 3.7.2 (Receiver Security)**.** *Let* $\mathsf{view}_{\mathcal{S}}$ *denote the view of the sender, the OT protocol* Π *is said to be* $(S(n), \epsilon)$*-secure for the Receiver if for all possible Senders that use up to* $S(n)$ *memory bits,*

$$(b, \mathsf{view}_{\mathcal{S}}) \approx_{\epsilon} (r, \mathsf{view}_{\mathcal{S}}),$$

where r is a uniformly random bit.

Now we give out our construction of the OT protocol.

The key idea is that the Receiver will send a commitment of its bit b to the Sender. And the Sender therefore uses the additive homomorphism of Raz's encryption scheme to compute the encryptions of $(1-b)x_0$ and bx_1. The Sender further re-randomizes these two ciphertexts by adding fresh encryptions of zero before sending them to the Receiver. The Receiver decrypts these two ciphertexts and obtains 0 and x_b as desired.

Construction 3.7.1 (Oblivious Transfer Protocol from Parity Learning)**.** *For given security parameter n, a Sender $\mathcal{S}$ and a receiver $\mathcal{R}'$:*

- *Run an Enhanced Encrypt Zero Protocol* $\mathtt{EZ}^+(n, 2n, \lambda = 2)$ *with* $\mathcal{R}$*'as the Keeper and* $\mathcal{S}$ *as the Recorder. At the end of the protocol,* $\mathcal{R}'$ *has as output a secret key* $\mathbf{s}$*, and* $\mathcal{S}$ *has output* $(\boldsymbol{\Sigma}|\boldsymbol{\phi})$*, which consists of two encryptions of* 0 *under the key* $\mathbf{s}$*. Additionally,* $\mathcal{R}'$ *samples random* $\boldsymbol{\gamma} \in \{0,1\}^{2n}$*, and sends* $(\boldsymbol{\gamma}, c = \boldsymbol{\gamma} \cdot \mathbf{s} + b)$ *to the Sender. Kindly notice that in this step the Receiver* $\mathcal{R}'$ *is actually just executing* $\mathtt{Commit}(n, b)$*.*

- *For Sender* $\mathcal{S}$*, let* $\boldsymbol{\sigma}_0$*,* $\boldsymbol{\sigma}_1$ *be the first and second row of* $\boldsymbol{\Sigma}$*, and* ϕ_0*,* ϕ_1 *be the two elements in* $\boldsymbol{\phi}$*. Notice that* $\phi_0 = \boldsymbol{\sigma}_0 \cdot \mathbf{s}$ *and* $\phi_1 = \boldsymbol{\sigma}_1 \cdot \mathbf{s}$*. The Sender then sends to the Receiver two ciphertexts:*

$$\begin{aligned} \boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma},\ \phi_0 + x_0(1-c) &= (\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma}) \cdot \mathbf{s} + \big((1-b)x_0\big) \\ \boldsymbol{\sigma}_1 + x_1\boldsymbol{\gamma},\ \phi_1 + x_1 c &= (\boldsymbol{\sigma}_1 + x_1\boldsymbol{\gamma}) \cdot \mathbf{s} + \big(bx_1\big). \end{aligned}$$

- $\mathcal{R}'$ *decrypts both ciphertexts that it has received using the key* $\mathbf{s}$*, and learns* $(1-b)x_0$ *and* bx_1*. Notice that one of these two values will be* x_b *as desired and gets output by* $\mathcal{R}'$*.*

We then proceed to prove desired security properties for the above construction of the OT protocol.

Theorem 3.7.1. *The OT protocol described above is perfectly secure for the Sender.*

Proof. We show that right after the first part of the protocol where $\mathcal{R}'$ executes $\mathtt{Commit}(n, b)$, there is a fixed $b' = c + \boldsymbol{\gamma} \cdot \mathbf{s} + 1$ such that the Receiver will have no information about $x_{b'}$. Notice that this does not break Receiver security, since although b' is fixed, $\mathcal{S}$ has no way to compute b' as $\mathbf{s}$ is only known to the Receiver $\mathcal{R}'$.

If $b' = c + \boldsymbol{\gamma} \cdot \mathbf{s} + 1 = 0$, we show that the Receiver has no information about x_0, i.e. x_0 is random given the Receiver's view. Notice that we have $1 - c = \boldsymbol{\gamma} \cdot \mathbf{s}$. And hence the two ciphertext that the Receiver receives are

$$\begin{aligned} \boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma},\ \phi_0 + x_0(1-c) &= (\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma}) \cdot \mathbf{s} \\ \boldsymbol{\sigma}_1 + x_1\boldsymbol{\gamma},\ \phi_1 + x_1 c &= (\boldsymbol{\sigma}_1 + x_1\boldsymbol{\gamma}) \cdot \mathbf{s} + x_1. \end{aligned}$$

The only source that the Receiver might be able to gather information about x_0 is from the first ciphertext. However, since $\boldsymbol{\sigma}_0$ is uniformly random given the Receiver's view, $\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma}$ is also uniformly random given the Receiver's view, i.e., it does not give any additional information to the Receiver. The Receiver also gets no information from $(\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma}) \cdot \mathbf{s}$, as this value can be easily simulated by the Receiver since it knows both $\boldsymbol{\sigma}_0 + x_0\boldsymbol{\gamma}$ and $\mathbf{s}$. Therefore, x_0 is random given the Receiver's view.

If $b' = c + \boldsymbol{\gamma} \cdot \mathbf{s} + 1 = 1$, by a similar argument, we have that x_1 is random given the Receiver's view. Bringing these parts together, we have shown that for $b' = c + \boldsymbol{\gamma} \cdot \mathbf{s} + 1$, $x_{b'}$ *is* random conditioned on the Receiver's view, i.e.

$$\left|\Pr[D_{\mathsf{view}_{\mathcal{R}'}}(x_{b'}) = 1] - \Pr[D_{\mathsf{view}_{\mathcal{R}'}}(r) = 1]\right| = 0.$$

Thus, the OT protocol above is perfectly secure for the Sender as desired.

□

Theorem 3.7.2. *The OT protocol described above is* $(Cn^2, O(2^{-n/2}))$*-secure for the Receiver, for some* $C < \frac{1}{20}$.

Proof. The proof for this is extremely straightforward. As observed above, the receiver $\mathcal{R}'$ is exactly executing $\texttt{Commit}(n, b)$, i.e. it is committing the bit b to the Sender, who is playing the role of the Verifier in the commitment scheme. Hence, by the $(Cn^2, O(2^{-n/2}))$-hiding property of the commitment scheme, we have that for all possible Sender $\mathcal{S}$ that uses at most Cn^2 memory bits,

$$(b, \mathsf{trans}, \mathsf{state}_{\mathcal{S}}) \approx_{\epsilon} (r, \mathsf{trans}, \mathsf{state}_{\mathcal{S}})$$

for $\epsilon = O(2^{-n/2}))$ and a uniformly random bit r. Notice that $\mathsf{view}_{\mathcal{S}}$ is actually just $(\mathsf{trans}, \mathsf{state}_{\mathcal{S}})$. Therefore, the above equation can be rewritten as

$$(b, \mathsf{view}_{\mathcal{S}}) \approx_{\epsilon} (r, \mathsf{view}_{\mathcal{S}}).$$

This is the exact definition for (Cn^2, ϵ)-Receiver-security. Therefore, the OT protocol above is $(Cn^2, O(2^{-n/2}))$-secure for the Receiver as desired.

□

4

Disappearing Cryptography in the Bounded Storage Model

4.1 Introduction

The bounded storage model[77] leverages bounds on the adversary's storage ability to enable secure applications. A typical bounded storage model scheme will involve transmitting more information than what the adversary can possibly store. One approach is then to use some small piece of the transmission to perform, say, a one-time pad or other tasks. Since the adversary cannot record the entire transmission, they most likely will not be able to recover the small piece that is used, preventing attacks. Other approaches, say those based on taking parities[85,62], are also possible. In any case, the honest users' space requirements are always much less than the adversary's storage bound; usually, if the honest parties have space N, the adversary is assumed to have space up to roughly $O(N^2)$.

The bounded storage model has mostly been used to give protocols with information-theoretic, unconditional, and everlasting security; in contrast, the usual time-bounded adversary model generally requires making computational assumptions.

A critical feature of the bounded storage model is that the large transmission cannot be entirely stored by the adversary. This large transmission is then subsequently used in such a way that whatever space-limited information the adversary managed to record about the transmission will become useless. In this way, the large transmission is ephemeral, effectively disappearing immediately after it is sent.

Most work in the bounded storage model uses this disappearing communication as a tool to achieve information-theoretic security for primitives such as key agreement, commitments, or oblivious transfer, for which computational assumptions are necessary in the standard model. However, apart from insisting on statistical security, the security goals are

typically the same as standard-model schemes.

The goal of this chapter, in contrast, is to use such "disappearing" communication to realize never-before-possible security goals, especially those that are *impossible* in the standard model.

4.1.1 Motivating Examples

Example 1: Deniable Encryption. Deniable encryption [28] concerns the following scenario: Alice has the secret key sk for a public key encryption scheme. At some point, Bob sends a ciphertext ct encrypting message m to Alice. Charlie observes the ciphertext ct.

Later, Charlie obtains the ability to force that Alice reveals sk (say, through a warrant), so that he can decrypt ct and learn the message m. Alice wants to maintain the privacy of the message m in this scenario, so she reveals a fake decryption key sk', such that decrypting ct with sk' will result in a fake message m'. This version of deniable encryption is called *receiver* deniable encryption.

Unfortunately, as shown in Bendlin *et al.* [15], such receiver deniable encryption is impossible for "normal" encryption where the ciphertext is just a single (concise) transmission from Bob to Alice*. Prior works [28,30] therefore consider a more general notion of encryption that involves back-and-forth communication between the parties.

In this chapter, we consider a different solution: what if the ciphertext is so large that it cannot be recorded by Charlie? Alice also cannot store the ciphertext in its entirety, but she will be able to decrypt it live using her secret key. Charlie, who does not know the secret key, will be unable to decrypt during the transmission. Then we may hope that, even if Alice

*The deniable encryption literature often refers to such a scheme as having two-messages, as they consider the transmission of the public key from Alice to Bob as the first message.

subsequently reveals the *true* secret key sk, that Charlie will not be able to learn the message m since he no longer has access to ct. Such a scheme would immediately be deniable: Alice can claim that ct encrypted any arbitrary message m', and Charlie would have no way to verify whether or not she was telling the truth. Relative to the solution in prior work, such a scheme would then require only one-way communication, but at the expense of greatly increased communication in order to ensure that Charlie cannot record all of ct. Such a scheme might make sense in a setting where Bob is unable to receive incoming communication.

Example 2: Second-hand Secret Keys. Consider an encrypted broadcast service where a user may buy a decoder box which decrypts broadcasts. The content distributor wants to enforce that for each decoder box, only one individual at a time can decrypt broadcasts. Specifically, the content distributor is concerned about the following attack: Alice has a decoder box, and uses it to decrypt a broadcast live at broadcast time. Then, post broadcast, she gives the box to Bob. Bob has previously stored the encrypted broadcast, and then feeds it into the decoder box to receive the broadcast. The result is that two individuals are able to use one box to decrypt the broadcast.

Our solution, again, is to imagine the ciphertexts being so long that they cannot be stored. As such, Alice's decoder box will be completely useless to Bob after the broadcast occurs.

Example 3: Non-interactive Security Against Replay Attacks. Consider a scenario where instructions are being broadcast from a command center to a number of recipients. Suppose that the recipients are embedded devices with limited capabilities; in particular, they cannot keep long-term state. We are concerned that an attacker may try to issue malicious instructions to the recipients.

The natural solution is to authenticate the instructions, say by signing them. However, this still opens up the possibility of a replay attack, where the adversary eavesdrops on some signed instruction, and then later on sends the same instruction a second time, causing some adverse behavior.

In the classical model with stateless recipients, the only way to prevent replay attacks is with an interactive protocol, since a stateless recipient cannot distinguish the command center's original message and signature from the adversary's replay. In a broadcast scenario, interacting with each recipient may be impractical. Moreover, interaction requires the recipients themselves to send messages, which may be infeasible, especially if the recipients are low-power embedded devices.

As before, our idea is to have the signatures on the instructions be so large that the adversary cannot record them in their entirety. The recipients can nonetheless validate the signatures, but an adversary will be unable to ever generate a valid signature, even after witnessing many authenticated instructions from the command center. The result is non-interactive security against replay attacks.

Example 4: Software Subscription. The traditional software model involves the software company sending the software to users, who then run the software for themselves. Software-as-a-Service, instead, has the software company centrally host the software, which the users run remotely. The centralized model allows for subscription-based software services—where the user can only have access to the program by making recurring payments—that are impossible in the traditional software model.

On the other hand, software-as-a-service requires the user to send their inputs to the software company. While many technologies exist to protect the user data, this model inherently

requires interaction with the users.

We instead imagine the company sends its software to the users, but the transmissions are so large that the users cannot record the entire program. Nevertheless, the users have the ability to run the program entirely locally during the transmission, and do not have to send any information to the software company. Then, once the transmission ends, the user will be unable to further run the program.

Example 5: Overcoming Impossibility Results for Obfuscation. Program obfuscation is a form of intellectual property protection whereby a program is transformed so that (1) all implementation details are hidden, but (2) the program can still be run by the recipient.

Virtual Black Box (VBB) obfuscation, as defined by Barak *et al.*[9], is the ideal form of obfuscation: it informally says that having the obfuscated code is "no better than" having black box access to the functionality. Unfortunately, Barak *et al.*show that such VBB obfuscation is impossible. The counter-example works by essentially running the program on its own description, something that is not possible just given oracle access. As a consequence, other weaker notions have been used, including indistinguishability obfuscation (iO) and differing inputs obfuscation[9], as well as virtual grey box obfuscation (VGBO)[18]. These notions have proven tremendously useful for cryptographic applications, where special-purpose programs are designed to be compatible with the notion of obfuscation used. However, for securing intellectual property inside general programs, these weaker notions offer only limited guarantees.

Our model for transmitting programs above may appear to give hope for circumventing this impossibility. Namely, if the obfuscated program is so large that it cannot be recorded in

its entirety, then maybe it also becomes impossible to run the program on its own description.

4.1.2 Our Results

In this chapter, we explore the setting of disappearing cryptography, giving both negative and positives results.

Online Obfuscation. First, we propose a concrete notion of online obfuscation, which is streamed to the recipient. We then explore what kinds of security guarantees we can hope for, motivated by Examples 4 and 5 above.

First, we demonstrate that VBB obfuscation is still impossible in most settings, assuming the hardness of the Learning With Errors (LWE) problem. The proof closely follows the Barak *et al.* proof in the case of circuits, but shows that it can be adapted to work on online obfuscation. Thus we show that Example 5 is not possible.

This still leaves open the hope that online obfuscation can yield something interesting that is not possible classically. We next define a useful notion of online obfuscation, motivated by the goal of classically-impossible tasks. Towards that end, we note that differing inputs obfuscation is known to be a problematic definition[48] in the standard model. We also observe that indistinguishability obfuscation offers no advantages in the streaming setting over the classical setting. We therefore settle on a notion of virtual grey box (VGB) obfuscation for online obfuscation. We formulate a definition of VGB obfuscation which allows the recipient to evaluate the program while it is being transmitted, but then lose access to the program after the transmission completes.

We give two candidate constructions of VGB online obfuscation, based on different ideas. We leave as an open question constructing a provably secure scheme.

Applications of Online Obfuscation. Next we turn to applications, establishing VGB online obfuscation as a central tool in the study of disappearing cryptography, and providing techniques for its use. We show how to use VGB online obfuscation to realize each of the Examples 1-3.

Specifically, assuming VGB online obfuscation (and other comparatively mild computational assumptions), we define and construct the following:

- Public key encryption with *disappearing ciphertext security* in the bounded storage model. Here, ciphertexts are streamed to the recipient, and message secrecy holds against adversaries with bounded storage*, *even if the adversary later learns the secret key*. This immediately solves Examples 1 and 2.

- We generalize to functional encryption with disappearing ciphertext security, which combines the disappearing security notion above with the expressive functionality of functional encryption. This allows, for example, to combine the advantages of disappearing ciphertext security with traditional functional encryption security goals of fine-grained access control.

- Digital signatures with *disappearing signature security*, where signatures are streamed, and the recipient loses the ability to verify signatures after the stream is complete. This solves Example 3.

In the following, we expand and explain our results in more detail.

*We also require the usual polynomial *time* constraint of the adversary.

4.1.3 Defining Obfuscation in the Bounded Storage Model

We first study obfuscation in the bounded storage model. We specifically imagine that obfuscated programs are too large to store, but can be streamed and run in low space while receiving the stream.

Negative Result for VBB Obfuscation. Our first result is that, virtual black box (VBB) security remains impossible, even for this model. Recall that VBB security requires that anything which can be efficiently learned from the obfuscated code can be efficiently learned given just oracle access to the function. We follow the Barak *et al.*[9] impossibility, but take care to show that it still works for online obfuscation.

The Barak *et al.*impossibility works roughly as follows. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a *fully homomorphic* encryption scheme. Choose random values α, β, γ as well as keys sk, pk for Enc, and consider the following program:

$$P(x) = \begin{cases} \mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \alpha) & \text{if } x = 0 \\ \beta & \text{if } x = \alpha \\ \gamma & \text{if } \mathsf{Dec}(\mathsf{sk}, x) = \beta \\ \bot & \text{otherwise} \end{cases}$$

An attacker with black box access to this program can learn pk and an encryption of α. But to learn anything about β, they need to query on α; by the security of Enc, this is impossible. Thus, the attacker cannot learn anything about γ.

On the other hand, an attacker with (perhaps obfuscated) *code* for P can homomorphi-

cally apply P to $\mathsf{Enc}(\mathsf{pk}, \alpha)$ to get $\mathsf{Enc}(\mathsf{pk}, \beta)$. Then they can feed $\mathsf{Enc}(\mathsf{pk}, \beta)$ into the program to learn γ.

For online obfuscation, we show that this works, provided the attacker has access to three sequential streams of the program. In the first stream the attacker evaluates on $\circ$ to learn $\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \alpha)$. In the second stream, it uses its evaluation procedure and the program stream to homomorphically evaluate P on $\mathsf{Enc}(\mathsf{pk}, \alpha)$, learning $\mathsf{Enc}(\mathsf{pk}, \beta)$. Finally, in the third stream it runs P on $\mathsf{Enc}(\mathsf{pk}, \beta)$ to learn γ.

The only challenging part is the second stream. Here, we use the evaluation procedure for the online obfuscation. Specifically, the evaluation procedure maintains a state, which is updated as each bit of the stream comes in. We run the evaluation algorithm homomorphically on the input $\mathsf{Enc}(\mathsf{pk}, \alpha)$, by maintaining an encrypted state, which we update homomorphically.

We then explain how to remove the final stream using Compute-and-Compare obfuscation [58,97], a technique used toward a similar goal in Ananth and La Placa [6]. The first stream can also be removed in an auxiliary input setting, which is needed for most interesting applications. Thus, in the auxiliary input setting we obtain an impossibility even for a single stream. The full proof is given in Section 4.3.

Defining Online Obfuscation. Above, we only considered the standard notions of security, but for online obfuscation. We now seek to formulate a definition which captures the goal of having the obfuscated program "disappear" after the stream is complete. Concretely, we want that, after the stream is complete, it is impossible to evaluate the program on any "new" inputs.

Our formalization of this is roughly as follows: we imagine the attacker gets the program

stream, and then later learns some additional information. We ask that any such attacker can be simulated by an oracle algorithm. This algorithm makes queries to the program, and then receives the same additional information the original adversary received. Importantly, after the additional information comes in, the simulator can no longer query the program any more.

Some care is needed with the definition. VBB security, which requires the simulator to be computationally bounded, is impossible for the reasons discussed above. Indistinguishability obfuscation (iO) allows for a computationally unbounded simulator and thus avoids the impossibility*. While iO is useful in the standard model, we observe that there is little added utility to considering iO in the online model. Indeed, an unbounded simulator can query the entire function on all inputs during the query phase, and thus has no need to make additional queries after receiving the additional information.

We therefore settle on a virtual *grey* box (VGB) notion of security[18], where the simulator is computationally unbounded, but can only make a polynomial number of queries. The computationally unbounded simulator then receives the additional information, but can make no more queries. Our full definition is in Section 4.2.

We note that it may be possible to also consider a version of differing inputs obfuscation (diO) in our setting, but there is evidence that diO may be impossible[49]. So we therefore stick to VGB obfuscation.

*Concretely, it can break Enc to learn α.

4.1.4 Applications

Before giving our candidate constructions of VGB online obfuscation, we discuss applications.

Disappearing Ciphertext Security. We first demonstrate how to use online obfuscation to construct public key encryption where ciphertexts effectively disappear after being transmitted. Concretely, we define a version of public key encryption where the attacker gets to learn the secret key *after* the ciphertext is transmitted. We require that the attacker nevertheless fails to learn anything about the message.

Our first attempt is the following, which essentially uses an online obfuscator as a witness encryption scheme[50]: the public key pk is set, say, to be the output of a one-way function f on the secret key sk. To encrypt a message m to pk, generate an online obfuscation of the program $P(\mathsf{sk}')$ which outputs m if and only if $f(\mathsf{sk}') = \mathsf{pk}$. Decryption just evaluates the program on the secret key.

For security, we note that, by the one-wayness of f, an attacker who just knows pk and sees the ciphertext cannot evaluate the ciphertext program on any input that will reveal m. Hence, m presumably remains hidden. Moreover, even if the attacker learns sk after seeing the ciphertext, it should not help the attacker learn m, since the attacker no longer has access to the program stream.

Formalizing this intuition, however, leads to difficulties. Suppose we have an adversary $\mathcal{A}$ for the encryption scheme. We would like to use $\mathcal{A}$ to reach a contradiction. To do so, we invoke the security of the online obfuscator to arrive at a simulator $\mathcal{S}$ that can only query the ciphertext program, but does not have access to the program stream. Unfortunately, this sim-

ulator is computationally unbounded, meaning it can invert f to recover sk at the beginning of the experiment, and then query the program on sk.

Our solution is to replace f with a lossy function [83], which is a function with two modes: an injective mode (where f is injective) and a lossy mode (where the image of f is small). The security requirement is that the two modes are indistinguishable.

We start with f being in the injective mode. In the proof, we first switch the ciphertext program to output m if and only if $\mathsf{sk}' = \mathsf{sk}$; by the injectivity of f this change does not affect the functionality of the program. Hence, the simulator cannot detect the change (even though it can invert f and learn sk for itself), meaning the adversary cannot detect the change either.

In the next step, we switch f to being lossy, which cannot be detected by a computationally bounded attacker. We next change the ciphertext program again, this time to never output m. This only affects the program's behavior on a single point sk. But notice that for lossy f, sk is statistically hidden from the attacker, who only knows pk when the ciphertext is being streamed. This means the simulator, despite being computationally unbounded, will be unable to query on sk, meaning the simulator cannot detect the change. This holds true even though the simulator later learns sk, since at this point it can no longer query the ciphertext program. Since indistinguishability holds relative to the simulator, it also holds for the original attacker. The full construction and proof are given in Section 4.4.

Extension to Functional Encryption. We can also extend disappearing ciphertext security to functional encryption. Functional encryption allows users to obtain secret keys for functions g, which allow them to learn $g(m)$ from ciphertext encrypting m. The usual requirement for functional encryption is that an attacker, who has secret keys for functions g_i

such that $g_i(m_0) = g_i(m_1)$ for all i, cannot distinguish encryptions of m_0 from encryptions of m_1.

In Section 4.6, we consider a similar notion, but where the requirement that $g_i(m_0) = g_i(m_1)$ only holds for secret keys in possession when the ciphertext is communicated. Even if the attacker later obtains a secret key for a function g such that $g(m_0) \neq g(m_1)$, indistinguishability will still hold. Analogous to the case of plain public key encryption, this captures the intuition that the ciphertext disappears, becoming unavailable once the transmission ends.

We show how to combine standard-model functional encryption with online VGB obfuscation to obtain functional encryption with such disappearing ciphertext security. The basic idea is as follows. To encrypt a message m, first compute an encryption c of m under the standard-model functional encryption scheme. Then compute an online obfuscation of the program which takes as input the secret key sk_g for a function g, and decrypts c using sk_g, the result being $g(m)$.

This construction seems like it should work, but getting the proof to go through using computationally unbounded simulators is again non-trivial. We show how to modify the sketch above to get security to go through.

Disappearing Signatures. We next turn to constructing disappearing signatures, signatures that are large streams that can be verified online, but then the signature disappears after the transmission ends. We formalize this notion by modifying the usual chosen message security game to require that the attacker (who does not know the signing key) cannot produce a signature on *any* message, even messages that it previously saw signatures for.

We show how to construct such signatures in Section 4.5, using online obfuscation. An

additional building block we need is a *prefix puncturable signature*. This is a signature scheme where, given the signing key sk, it is possible to produce a "punctured" signing key sk_{x^*} which can sign any message of the form (x, m) such that $x \neq x^*$. We require that, even given sk_{x^*}, no message of the form (x^*, m) can be signed. Such prefix puncturable signatures can be built from standard tools[13].

We construct a signature scheme with disappearing signatures by setting the signature on a message m to be an online obfuscation of the following program P. P has sk hardcoded, and on input x outputs a signature on (x, m). To verify, simply run the streamed program on a random prefix to obtain a signature, and then verify the obtained signature.

We then prove that an attacker cannot produce a valid signature stream on any message, even messages for which it already received signature streams. For simplicity, consider the case where the attacker gets to see a signature on a single message m. Let x^* be the prefix that the verifier will use to test the adversary's forgery. Note that x^* is information-theoretically hidden to the adversary at the time it produces its forgery. We will switch to having the signature program for m that rejects the prefix x^*. Since the program no longer needs to sign the prefix x^*, it can use the punctured key sk_{x^*} to sign instead. The only point where the program output changes is on x^*. The simulator will be unable to query on x^* (since it is information-theoretically hidden), meaning the simulator, and hence the original adversary, cannot detect this change.

Now we rely on the security of the puncturable signature to conclude that the adversary's forgery program cannot output a signature on any message of the form (x^*, m), since the entire view of the attacker is simulated with the punctured key sk_{x^*}. But such a signature is exactly what the verifier expects to see; hence the verifier will reject the adversary's program.

4.1.5 Constructing Online Obfuscation

We finally turn to giving two candidate constructions of online obfuscation. We unfortunately do not know how to prove the security of either construction, which we leave as an interesting open problem. However, we discuss why the constructions are presumably resistant to attacks.

Construction 1: Large Matrix Branching Programs. Our first construction is based on standard-model obfuscation techniques, starting from Garg *et al.*[47]. As in Garg *et al.*[47], we first convert an NC^1 circuit into a matrix branching program using Barrington's theorem[12]. In Garg *et al.*[47], the program is then "re-randomized" following Kilian[74] by left and right multiplying the various branching program components with random matrices, such that the randomization cancels out when evaluating the program. We instead first pad the matrices to be very large, namely so large that honest users can record a single column, but the adversary cannot write down the entire matrix. We then re-randomize the large padded matrix.

We show that, if the matrix components are streamed in the correct order, honest users can evaluate the program in low space. However, since the program is too large to write down, malicious users will presumably be unable to evaluate the program once the stream concludes.

We note that in the standard model, re-randomizing the branching program is not enough to guarantee security. Indeed, linear algebra attacks on the program matrices are possible, as well as "mixed-input" attacks where multiple reads of the same input bit are set to different values. Garg *et al.*[47] and follow-up works block these attacks by placing the branching

program matrices "in the exponent" of a cryptographic multilinear map.

In our setting, the large matrices presumably prevent linear algebra attacks. Moreover, we show how to block mixed-input attacks by choosing the matrix padding to have a special structure. While we are unable to prove the security of our multilinear-map-less scheme, we conjecture that it nevertheless remains secure. The result is a plausible VGB online obfuscator for NC^1 circuits. Details are given in Section 4.7.

Construction 2: Time-stamping. Our second construction is based on time-stamping[78] in the bounded storage model. Here, a large stream is sent. Anyone listening can use the stream to compute a time-stamp on any message. However, once the stream concludes, it will be impossible to time-tamp a "new" message. The concrete security notion guarantees a fixed (polynomial-sized) upper bound on the total number of stamped messages any adversary can produce.

Our construction uses time-stamping, together with standard-model obfuscation. To obfuscate a program P, first send the random stream. Then, compute a standard-model obfuscation of the program P', which takes as input x and a time-stamp for x, verifies the time-stamp, and then runs P if the stamp is valid.

Assuming the standard model obfuscation is has VGB security, this construction *should* be an online obfuscation with VGB security. The intuition is to start with a VGB simulator for the standard-model scheme. This simulator is allowed to make queries at any time after the obfuscation of P' is generated, even after receiving the additional information. However, the only useful queries to P' are on inputs with valid time-stamps. The intuition is that, by the security of the time-stamping scheme, it should be information-theoretically possible to determine all the time-stamped messages that the adversary could possibly produce once the

stream concludes. The simulator will determine the possible queries, and make each of them while it has access to the program. All future queries by the simulator will then be rejected.

Unfortunately, we do not know how to actually rigorously prove that this construction works. The difficulty is justifying that we can actually anticipate all valid time-stamps that may be produced. We therefore leave formalizing the above intuition as an interesting open question.

4.1.6 Related Work

Time-stamping in the bounded storage model[78], as discussed above, is perhaps the first application of the bounded storage model beyond achieving information-theoretic security. We note, however, that non-interactive time-stamping was recently achieved in the standard model using appropriate computational assumptions[75].

Dziembowski[45] consider a notion of forward-secure storage, which is very similar to our notion of disappearing ciphertext security for encryption. A key difference is that their work only considers the secret key case, and it is unclear how to adapt their constructions to the public key setting.

Our notion of disappearing ciphertext security can be seen as achieving a notion of *forward* security, where a key revealed does not affect the security of prior sessions. Forward security has been studied in numerous standard-model contexts (e.g. [38]). However, standard-model constructions of forward security (non-interactive) encryption such as[29] always involve updating the secret keys. Our construction does not require the secret key to be updated.

4.2 Defining Obfuscation in the Bounded Storage Model

In this section we will formally define online obfuscation ($o\mathcal{O}$) and its corresponding security notions, but before we start, we will first introduce an idea called a *stream*.

A *stream* $s_{\gg}$ is a sequence of bits sent from one party to another. Generally, we require that the length of the stream, denoted as $|s_{\gg}|$, to be greater than the memory bound of the users and adversaries. This means that a properly constructed stream can *not* be stored in its entirety. However, algorithms or programs can still take a stream as an input. This means that the algorithm or program would operate in an online manner - it actively listens to the stream as the bits come in, and performs the computation simultaneously. We denote a variable as a stream by putting a "$\gg$" in the subscript.

Definition 4.2.1 (Online Obfuscator)**.** *Let λ, n be security parameters. An online obfuscator $o\mathcal{O}$ for a circuit class $\{\mathcal{C}_\lambda\}$ consists of a pair of uniform PPT machines* $(\mathsf{Obf}, \mathsf{Eval})$ *that satisfy the following conditions:*

- Obf *takes as input a circuit $C \in \mathcal{C}_\lambda$, uses up to $O(n)$ memory bits, and produces a stream $s_{\gg} \leftarrow \mathsf{Obf}(C)$.*
- Eval *takes as input a stream $s_{\gg}$ and an input x, uses up to $O(n)$ memory bits, and outputs $y \leftarrow \mathsf{Eval}(s_{\gg}, x)$.*
- *For all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that*

$$\Pr\left[C(x) = y : s_{\gg} \leftarrow \mathsf{Obf}(C), y \leftarrow \mathsf{Eval}(s_{\gg}, x)\right] = 1.$$

To define security for an online obfuscator $o\mathcal{O} = (\mathsf{Obf}, \mathsf{Eval})$, consider the following two experiments:

1. $\mathsf{Exp}_{\mathcal{A},\mathsf{ch},o\mathcal{O}}(C \in \mathcal{C}_\lambda, k)$:

    - The experiment consists of an arbitrary number of rounds. At each round, one of the following two scenarios happens:
        - At an *interaction round*, the adversary $\mathcal{A}$ interacts arbitrarily with the challenger ch.
        - At a *stream round*, the adversary $\mathcal{A}$ receives a fresh stream* of the obfuscated circuit $s_{\gg} \leftarrow \mathsf{Obf}(C)$. The challenger ch will receive a special tag notifying it that a streaming has happened.
    - The challenger ch may choose to terminate the experiment at any time by outputting a bit $b \in \{0,1\}$, and b will be the output of the program.
    - Whenever the number of stream rounds is greater than k, the challenger ch immediately outputs 0 and terminates the experiment.

2. $\mathsf{Exp}_{\mathcal{S},\mathsf{ch},o\mathcal{O}}(C \in \mathcal{C}_\lambda, k, q)$:

    - The experiment consists of an arbitrary number of rounds:
        - At an *interaction round*, the simulator $\mathcal{S}$ interacts arbitrarily with the challenger ch.

*Notice that a fresh stream is sampled every time, so that no single stream is sent repeatedly.

– At a *stream round*, the simulator $\mathcal{S}$ may send up to q adaptive oracle queries to the circuit C and receive corresponding responses. The challenger ch will receive a special tag notifying it that a streaming has happened.

- The challenger ch may choose to terminate the experiment at any time by outputting a bit $b \in \{0,1\}$, and b will be the output of the program.
- Whenever the number of stream rounds is greater than k, the challenger ch immediately outputs 0 and terminates the experiment.

Definition 4.2.2 (k-time Virtual Grey-Box (VGB) Security)**.** *Let λ, n be security parameters. Let k be a fixed positive integer. For an online obfuscator $o\mathcal{O}$ to satisfy k-time Virtual Grey-Box security, we require that there exists a memory bound $S(n)$, such that for any challenger ch, and any adversary $\mathcal{A}$ that uses up to $S(n)$ memory bits, there exists a computationally unbounded simulator $\mathcal{S}$ s.t. for all circuits $C \in \mathcal{C}_\lambda$:*

$$|\Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{ch},o\mathcal{O}}(C,k)=1] - \Pr[\mathsf{Exp}_{\mathcal{S},\mathsf{ch},o\mathcal{O}}(C,k,q)=1]| \leq \mathsf{negl}(\lambda),$$

where $q = \mathsf{poly}(\lambda)$.

The definitions for Indistinguishability Obfuscation (iO) security and Virtual Black-Box (VBB) security are obtained analogously by applying minor changes to the VGB security definition.

Remark 4.2.1 (k-time iO Security)**.** *We modify Definition 4.2.2 to allow $q = \mathsf{superpoly}(\lambda)$ to obtain the definition for k-time iO Security.*

Remark 4.2.2 (k-time VBB Security)**.** *We modify Definition 4.2.2 to restrict $\mathcal{S}$ to be a PPT simulator to obtain the definition for k-time VBB Security. We show in Section 4.3 that online obfuscators with VBB security do not exist.*

Remark 4.2.3 (1-time VBB/VGB/iO Security)**.** *Under the special case where $k = 1$, we obtain the definitions for 1-time VBB/VGB/iO security correspondingly.*

Remark 4.2.4 (Unbounded VBB/VGB/iO Security)**.** *Under the special case where $k = \mathsf{superpoly}(\lambda)$, we obtain the definitions for unbounded VBB/VGB/iO security correspondingly.*

4.3 Impossibility of VBB Online Obfuscation

In this section, we show that online obfuscation with VBB security does not exist in the Bounded Storage Model if fully homomorphic encryptions and obfuscation of multi-bit compute-and-compare programs exist. Note that both of these primitives can be built from the assumption that the Learning With Errors (LWE) problem is hard.

4.3.1 Fully Homomorphic Encryption

A Fully Homomorphic Encryption (FHE) scheme is a public key encryption scheme with an additional Eval procedure that allows arbitrary computations on the ciphertexts.

Definition 4.3.1 (Fully Homomorphic Encryption)**.** *Let λ be the security parameter. A fully homomorphic encryption scheme for circuit class $\{\mathcal{C}_\lambda\}$ is a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ with the following syntax.*

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ *takes as input the security parameter λ, and outputs a public key* pk *and a secret key* sk.

- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$ *takes as input the public key* pk *and a message* $m \in \{0,1\}^*$*, and outputs a ciphertext* ct*.*
- $\mathsf{Eval}(C, \mathsf{ct}) \to \mathsf{ct}'$ *takes as input a circuit* $C \in \mathcal{C}_\lambda$ *and a ciphertext* ct*, and outputs an evaluated ciphertext* ct'*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$ *takes as input a private key* sk *and a ciphertext* ct*, and outputs a decrypted message* m*.*

In addition to the usual PKE correctness and security requirements (which don't involve Eval at all), we require correctness of homomorphic evaluations.

Definition 4.3.2 (Correctness of Homomorphic Evaluations)**.** *A fully homomorphic encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is correct for homomorphic evaluations if for all messages* m *and circuits* $C \in \mathcal{C}_\lambda$*,*

$$\Pr\left[y = C(m) : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ \mathsf{ct}' \leftarrow \mathsf{Eval}(C, \mathsf{ct}) \\ y \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Gentry, Sahai and Waters[52] have shown how to construct such an FHE scheme assuming the hardness of the LWE problem.

4.3.2 Obfuscation of Compute-and-Compare Programs

The idea of compute-and-compare programs was first raised by Wichs and Zirdelis[97] in 2017. Around the same time, the work of Goyal, Koppula and Waters[58] essentially shows the same

result which they named "lockable obfuscation", with some slight differences in presentation and focus. Here, we will use the notion of multi-bit compute-and-compare programs from Wichs and Zirdelis[97].

Definition 4.3.3 (Multi-Bit Compute-and-Compare Program)**.** *Given a function* $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$*, a target value* $y \in \{0,1\}^{\ell_{out}}$*, and a message* $z \in \{0,1\}^{\ell_{msg}}$*, a multi-bit compute-and-compare program* P *is defined as follows:*

$$P_{f,y,z}(x) = \begin{cases} z & \text{if } f(x) = y \\ \\ \perp & \text{otherwise} \end{cases}.$$

Wichs and Zirdelis[97] have shown that obfuscation of multi-bit compute-and-compare programs exists, assuming the hardness of the LWE problem.

Lemma 4.3.1 ([97])**.** *If the LWE problem is hard, then there exists an obfuscator* $(\mathsf{Obf}, \mathsf{Eval})$ *for multi-bit compute-and-compare programs such that:*

- *For any multi-bit compute-and-compare program* P *and input* x*,*

$$\Pr\left[P(x) = w : \mathcal{P} \leftarrow \mathsf{Obf}(P), w \leftarrow \mathsf{Eval}(\mathcal{P}, x)\right] = 1.$$

- *For any multi-bit compute-and-compare program* P *with size parameters* ℓ_{in}, ℓ_{out}*, and* ℓ_{msg}*, if the target value* y *for* P *is chosen uniformly at random, then there exists a (non-uniform) PPT simulator* $\mathcal{S}$*, such that*

$$\mathsf{Obf}(P) \overset{c}{\approx} \mathcal{S}(\ell_{in}, \ell_{out}, \ell_{msg}).$$

4.3.3 Proof of VBB Impossibility

Theorem 4.3.1. *If fully homomorphic encryptions and obfuscation of multi-bit compute-and-compare programs exist, then online obfuscators with VBB security do not exist.*

Proof. Let $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a secure FHE scheme. First, we run $\mathsf{FHE.Gen}(1^\lambda)$ to obtain $(\mathsf{pk}, \mathsf{sk})$, and sample uniformly at random $\alpha, \beta, \gamma \in \{0,1\}^\lambda$. Let Q be a multi-bit compute-and-compare program where f is the FHE decryption function $\mathsf{FHE.Dec}$ with the secret key sk hardcoded in, and $y = \beta$ and $z = \gamma$. Notice that we have:

$$Q_{\beta,\gamma}(x) = \begin{cases} \gamma & \text{if } \mathsf{FHE.Dec}_{\mathsf{sk}}(x) = \beta \\ \\ \bot & \text{otherwise} \end{cases}.$$

Let $\mathcal{Q}$ be the obfuscated version of Q and define the program P as follows:

$$P_{\alpha,\beta}(x) = \begin{cases} \mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha), \mathcal{Q} & \text{if } x = 0 \\ \\ \beta & \text{if } x = \alpha \\ \\ \bot & \text{otherwise} \end{cases}.$$

We assume that there exists an online obfuscator $o\mathcal{O}$ with 2-time VBB security, and consider the following adversary $\mathcal{A}$ for the experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{ch},o\mathcal{O}}(P, k = 2)$:

- In the first stream round, $\mathcal{A}$ receives a stream $s_{\gg} \leftarrow o\mathcal{O}.\mathsf{Obf}(P)$ and then computes $o\mathcal{O}.\mathsf{Eval}(s_{\gg}, 0)$, obtaining $\mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)$ and $\mathcal{Q}$*.

*If there is an interaction round before the first stream round, during which the challenger sends $\mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)$ and $\mathcal{Q}$ to $\mathcal{A}$ as auxiliary input, then we can build a similar $\mathcal{A}$ for the experiment with $k = 1$, breaking the 1-time VBB security with auxiliary input.

- In the second stream round, $\mathcal{A}$ receives another stream $s'_{\gg} \leftarrow o\mathcal{O}.\mathsf{Obf}(P)$. $\mathcal{A}$ homomorphically evaluates P on ciphertext $\mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)$ by computing

$$\begin{aligned}\mathsf{FHE.Eval}\big(o\mathcal{O}.\mathsf{Eval}(s'_{\gg}, \cdot), \mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)\big) &= \mathsf{FHE.Eval}\big(P, \mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)\big) \\ &= \mathsf{FHE.Enc}_{\mathsf{pk}}\big(P(\alpha)\big) \\ &= \mathsf{FHE.Enc}_{\mathsf{pk}}(\beta).\end{aligned}$$

- In the next interaction round, $\mathcal{A}$ runs the program $\mathcal{Q}$ on input $\mathsf{FHE.Enc}_{\mathsf{pk}}(\beta)$ to obtain γ. Then $\mathcal{A}$ sends γ to the challenger.

VBB security of the online obfuscator requires that there exists a computationally bounded simulator $\mathcal{S}$ for the experiment $\mathsf{Exp}_{\mathcal{S},\mathsf{ch},o\mathcal{O}}(P, k = 2, q = \mathsf{poly}(\lambda))$. Given the security of the FHE scheme and that $\mathcal{S}$ is only allowed $q = \mathsf{poly}(\lambda)$ number of oracle queries to the program P, with overwhelming probability $\mathcal{S}$ can obtain only $\mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)$ and $\mathcal{Q}$ in the stream rounds. Notice that $\mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)$ does not depend on γ at all, and for the computationally bounded $\mathcal{S}$, by lemma 4.3.1, $\mathcal{Q}$ is indistinguishable from a simulator that has no knowledge of γ. Hence, the probability that $\mathcal{S}$ can send γ to the challenger is negligible, as opposed to $\mathcal{A}$, who always sends γ successfully. Therefore, a challenger can easily distinguish between the two experiments, thus breaking the 2-time VBB security of the online obfuscator.

□

4.4 Public Key Encryption with Disappearing Ciphertext Security

4.4.1 Definition

We will start by defining a security notion for public key encryption that we name *Disappearing Ciphertext Security*.

Essentially, it captures the security game where the adversary is given the private key after all of its queries but before it outputs a guess for the bit b. In traditional models, this definition does not make much sense, as the adversary can simply store the query responses, and then later use the received private key to decrypt. However, in the bounded storage model, the adversary cannot possibly store the ciphertexts, so even if the adversary is handed the private key afterwards, it cannot possibly use it to decrypt anything.

Put formally, for security parameters λ and n, a public key encryption scheme in the bounded storage model is a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ that each uses up to $O(n)$ memory bits. The syntax is identical to that of a classical PKE, except that now the ciphertexts are streams $\mathsf{ct}_{\gg}$. For the security definition, consider the following experiment:

Disappearing Ciphertext Security Experiment $\mathsf{Dist}^{\mathsf{DisCt}}_{\mathcal{A},\Pi}(\lambda, n)$:

- Run $\mathsf{Gen}(1^\lambda, 1^n)$ to obtain keys $(\mathsf{pk}, \mathsf{sk})$.
- Sample a uniform bit $b \in \{0, 1\}$.
- The adversary $\mathcal{A}$ is given the public key pk.
- The adversary $\mathcal{A}$ submits two messages m_0 and m_1, and receives $\mathsf{Enc}(\mathsf{pk}, m_b)$, which is a stream.

- The adversary $\mathcal{A}$ is given the private key sk.
- The adversary $\mathcal{A}$ outputs a guess b' for b. If $b' = b$, we say that the adversary succeeds and the output of the experiment is 1. Otherwise, the experiment outputs 0.

Using this experiment, we are now able to formally define disappearing ciphertext security.

Definition 4.4.1 (Disappearing Ciphertext Security)**.** *Let λ, n be security parameters. A public key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has disappearing ciphertext security under memory bound $S(n)$ if for all PPT adversaries $\mathcal{A}$ that use at most $S(n)$ memory bits:*

$$\Pr\left[\mathsf{Dist}^{\mathsf{DisCt}}_{\mathcal{A},\Pi}(\lambda, n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Now we will show how to use online obfuscation to construct a public key encryption scheme with disappearing ciphertext security. One important tool that we will take advantage of is lossy functions, which we will introduce in the following.

4.4.2 Lossy Function

Lossy functions are a subset of Lossy Trapdoor Functions due to Peikert and Waters[83] that do not require the existence of a trapdoor for the injective mode. To put formally:

Definition 4.4.2 (Lossy Function)**.** *Let λ be the security parameter. For $\ell(\lambda) = \mathsf{poly}(\lambda)$ and $k(\lambda) \leq \ell(\lambda)$ (k is referred to as the "lossiness"), a collection of (ℓ, k)-lossy functions is given by a tuple of PPT algorithms (S, F) with the following properties. As short-hands, we have $S_{inj}(\cdot)$ denote $S(\cdot, 1)$ and $S_{lossy}(\cdot)$ denote $S(\cdot, 0)$.*

- ***Easy to sample an injective function**: S_{inj} outputs a function index s, and $F(s,\cdot)$ computes an injective (deterministic) function $f_s(\cdot)$ over the domain $\{0,1\}^\ell$.*

- ***Easy to sample a lossy function**: S_{lossy} outputs a function index s, and $F(s,\cdot)$ computes a (deterministic) function $f_s(\cdot)$ over the domain $\{0,1\}^\ell$ whose image has size at most $2^{\ell-k}$.*

- ***Hard to distinguish injective mode from lossy mode**: Let X_λ be the distribution of s sampled from S_{inj}, and let Y_λ be the distribution of s sampled from S_{lossy}, the two distributions should be computationally indistinguishable, i.e. $\{X_\lambda\} \overset{c}{\approx} \{Y_\lambda\}$.*

4.4.3 Construction

Here we present our construction of a PKE scheme with disappearing ciphertext security, using online obfuscation and lossy function as building blocks.

Construction 4.4.1. *Let λ, n be the security parameters. Let $\mathsf{LF} = (S, F)$ be a collection of (ℓ, k)-lossy functions, and $o\mathcal{O} = (\mathsf{Obf}, \mathsf{Eval})$ an online obfuscator with 1-time VGB security under $S(n)$ memory bound. The construction $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^n)$*: Sample an injective function index f_s from S_{inj}, and a uniform $\mathsf{sk} \leftarrow \{0,1\}^\ell$. Compute $y = F(s, \mathsf{sk}) = f_s(\mathsf{sk})$, and set $\mathsf{pk} = (s, y)$. Output $(\mathsf{pk}, \mathsf{sk})$.*

- $\mathsf{Enc}(\mathsf{pk}, m)$*: Construct the program $P_{f_s,y,m}$ as follows:*

$$P_{f_s,y,m}(x) = \begin{cases} m & \text{if } f_s(x) = y \\ \bot & \text{otherwise} \end{cases}.$$

Obfuscate the above program to obtain a stream $\mathsf{ct}_{\gg} \leftarrow \mathsf{Obf}(P_{f,y,m})$. *The ciphertext is simply the stream* $\mathsf{ct}_{\gg}$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\gg})$*: Simply evaluate the streamed obfuscation using* sk *as input. An honest execution yields* $\mathsf{Eval}(\mathsf{ct}_{\gg}, \mathsf{sk}) = P_{f,y,m}(\mathsf{sk}) = m$ *as desired.*

4.4.4 Proof of Security

Now we show that if LF is a collection of (ℓ, k)-lossy functions with a lossiness $k = \mathsf{poly}(\lambda)$, and $o\mathcal{O}$ is an online obfuscator with 1-time VGB security under $S(n)$ memory bound, then the above construction has disappearing ciphertext security under $S(n)$ memory bound.

We organize our proof into a sequence of hybrids. In the very first hybrid, the adversary plays the disappearing ciphertext security game $\mathsf{Dist}^{\mathsf{DisCt}}_{\mathcal{A},\Pi}(\lambda, n)$ where b is fixed to be 0. Then we gradually modify the hybrids to reach the case where $b = 1$. We show that all pairs of adjacent hybrids are indistinguishable from each other, and therefore by a hybrid argument the adversary cannot distinguish between $b = 0$ and $b = 1$. This then directly shows disappearing ciphertext security.

Sequence of Hybrids

- H_0: The adversary plays the original disappearing ciphertext security game $\mathsf{Dist}^{\mathsf{DisCt}}_{\mathcal{A},\Pi}(\lambda, n)$ where $b = 0$, i.e. it always receives $\mathsf{Enc}(\mathsf{pk}, m_0)$.

- H_1: The same as H_0, except that in $\mathsf{Enc}(\mathsf{pk}, m_b)$, we replace P_{f,y,m_b} with P'_{sk,m_b} such

that

$$P'_{\mathsf{sk},m_b}(x) = \begin{cases} m_b & \text{if } x = \mathsf{sk} \\ \\ \bot & \text{otherwise} \end{cases}.$$

So now instead of checking the secret key by checking its image in the injective function, the program now directly checks for sk.

- H_2: The same as H_1, except that instead of sampling f_s from S_{inj}, we now use $f_{s'}$ sampled from S_{lossy}.
- H_3: The same as H_2, except that now we set $b = 1$ instead of 0.
- H_4: Switch back to using injective f_s instead of the lossy $f_{s'}$.
- H_5: Switch back to using the original program P_{f_s,y,m_b} instead of P'_{sk,m_b}.

Proof of Hybrid Arguments

Lemma 4.4.1. *If the online obfuscator $o\mathcal{O}$ has 1-time VGB security under memory bound $S(n)$, then no (potentially computationally unbounded) adversary that uses up to $S(n)$ memory bits can distinguish between H_0 and H_1 with non-negligible probability.*

Proof. This step actually only relies on *indistinguishability obfuscation* security of the obfuscator $o\mathcal{O}$, which is implied by its online VGB security. Notice that the only difference between H_0 and H_1 is the program P_{f_s,y,m_b} and P'_{sk,m_b} being obfuscated. Now notice that if f_s is injective, and that $y = f_s(\mathsf{sk})$, then $f_s(x) = y$ is equivalent to $x = \mathsf{sk}$. Hence, P_{f_s,y,m_b} and P'_{sk,m_b} have the exact same functionality, i.e. on the same input x, their outputs $P_{f_s,y,m_b}(x)$ and $P'_{\mathsf{sk},m_b}(x)$ are always the same. Then by the VGB (or even iO) security under memory bound

$S(n)$, no adversary under memory bound $S(n)$ should not be able to distinguish between the obfuscations of these two programs with non-negligible probability.

□

Lemma 4.4.2. *If* LF *is a collection of* (ℓ, k)*-lossy functions, then no PPT adversary can distinguish between* H_1 *and* H_2 *with non-negligible probability.*

Proof. This step is quite straightforward. Notice that the only difference between H_1 and H_2 is that an injective f_s is sampled in H_1 while a lossy $f_{s'}$ is sampled in H_2. Therefore, the only way an adversary can distinguish between H_1 and H_2 is by directly distinguishing f_s from $f_{s'}$, which contradicts with the security of the lossy function that it is hard to distinguish injective mode from lossy mode.

Put formally, we show how one can use an adversary $\mathcal{A}$ that distinguishes H_1 from H_2 to construct an adversary $\mathcal{A}'$ that distinguishes between the injective mode and the lossy mode of the lossy function.

$\mathcal{A}'$ receives a distribution X of function indices s sampled from either S_{inj} or S_{lossy} and it needs to tell which mode the distribution is sampled from. $\mathcal{A}'$ would run $\mathsf{Gen}(1^\lambda, 1^n)$, except that now sample s directly from the distribution X. Then $\mathcal{A}'$ simulates the rest of the disappearing ciphertext security game for $\mathcal{A}$ by playing the role of the challenger with fixed $b = 0$. At the end of the game, $\mathcal{A}$ should be able to tell if it is in H_1 or H_2. If $\mathcal{A}$ says it is in H_1, $\mathcal{A}'$ claims that X is sampled from S_{inj}, and if $\mathcal{A}$ says it is in H_2, $\mathcal{A}'$ claims that X is sampled from S_{lossy}.

Notice that if X is sampled from S_{inj}, then the view of $\mathcal{A}$ is identical to the one in H_1, and if X is sampled from S_{lossy}, the view of $\mathcal{A}$ is identical to the one in H_2. Therefore, if $\mathcal{A}$ succeeds in distinguishing H_1 from H_2, $\mathcal{A}'$ succeeds in distinguishing between the injective mode and

the lossy mode.

□

Lemma 4.4.3. *If the online obfuscator $o\mathcal{O}$ has 1-time VGB security under memory bound $S(n)$, and the lossiness k of LF is $\mathsf{poly}(\lambda)$, then no (potentially computationally unbounded) adversary under memory bound $S(n)$ can distinguish between H_2 and H_3 with non-negligible probability.*

Proof. First, we show that if the lossiness $k = \mathsf{poly}(\lambda)$, the secret key sk is information theoretically hidden from the adversary before it is sent. Recall that $y = F(s', \mathsf{sk}) = f_{s'}(\mathsf{sk})$ where $f_{s'}$ is a lossy function. For $f_{s'}$, the size of the domain is 2^{ℓ}, while the size of the range is only $2^{\ell-k}$. This implies that for an image y of a random input, the number of possible pre-images is at least $2^{k/2}$, except with probability at most $2^{-k/2}$. Now if the lossiness k is $\mathsf{poly}(\lambda)$, the number of possible pre-images is exponential, except with negligible probability. Given that the only constraint on sk is uniformly random conditioned on being a pre-image of y, it is information theoretically unpredictable from the adversary.

Now since sk is information theoretically hidden, the program P' is essentially a point function on a random point. And the only difference between H_2 and H_3 is the output of the point function. If an adversary is able to distinguish between H_2 and H_3, this means that the adversary is able to distinguish the output of an obfuscated point function without even knowing the point. This directly presents an adversary $\mathcal{A}$ for the 1-time VGB security game. In experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{ch},o\mathcal{O}}$, the adversary $\mathcal{A}$ is always able to obtain the output of an obfuscated point function. However, in game $\mathsf{Exp}_{\mathcal{S},\mathsf{ch},o\mathcal{O}}$, the simulator $\mathcal{S}$ is only allowed to make $q = \mathsf{poly}(\lambda)$ number of oracle queries to the point function. The probability that the simulator is able to obtain the output is only $q/2^{k/2} = \mathsf{poly}(\lambda)/2^{\mathsf{poly}(\lambda)} = \mathsf{negl}(\lambda)$. Therefore, the challenger can easily tell if it is interacting with the adversary $\mathcal{A}$ or the simulator $\mathcal{S}$, which

contradicts with the 1-time VGB security of the online obfuscator.

□

Lemma 4.4.4. *If* LF *is a collection of* (ℓ, k)*-lossy functions, then no PPT adversary can distinguish between* H_3 *and* H_4 *with non-negligible probability.*

The proof of this lemma follows analogously from the one of Lemma 4.4.2.

Lemma 4.4.5. *If the online obfuscator* $o\mathcal{O}$ *has* 1*-time VGB security under memory bound* $S(n)$*, then no (potentially computationally unbounded) adversary that uses up to* $S(n)$ *memory bits can distinguish between* H_4 *and* H_5 *with non-negligible probability.*

The proof of this lemma follows analogously from the one of Lemma 4.4.1.

Theorem 4.4.1. *If* LF *is a collection of* (ℓ, k)*-lossy functions with lossiness* $k = \mathsf{poly}(\lambda)$*, and* $o\mathcal{O}$ *is an online obfuscation with* 1*-time VGB security under* $S(n)$ *memory bound, then Construction 4.4.1 has disappearing ciphertext security under* $S(n)$ *memory bound.*

Proof. The lemmas above show a sequence of a polynomial number of hybrid experiments where no PPT adversary with $S(n)$ memory bound can distinguish one from the next with non-negligible probability. Notice that the first hybrid H_0 corresponds to the disappearing ciphertext security game where $b = 0$, and the last hybrid H_5 corresponds to one where $b = 1$. The security of the indistinguishability game follows.

□

4.5 Disappearing Signature Scheme

4.5.1 Definition

In this section, we define a public-key signature scheme in the bounded storage model which we call *Disappearing Signatures*. The idea is that we make the signatures be streams such that one can only verify them on the fly, and cannot possibly store them. The security game requirement is also different. Traditionally, for an adversary to win the signature forgery game, the adversary would need to produce a signature on a fresh new message. However, in the disappearing signature scheme, the adversary can win even by producing a signature on a message that it has previously queried. The catch here is that even though the message might have been queried by the adversary before, the adversary has no way to store the valid signature on the message due to its sheer size.

Put formally, for security parameters λ and n, a disappearing signature scheme consists of a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ that each uses up to $O(n)$ memory bits. The syntax is identical to that of a classical public key signature scheme, except that now the signatures are streams $\sigma_{\gg}$. For the security definition, consider the following experiment:

Signature Forgery Experiment $\mathsf{SigForge}_{\mathcal{A},\Pi}(\lambda, n)$:

- Run $\mathsf{Gen}(1^\lambda, 1^n)$ to obtain keys $(\mathsf{pk}, \mathsf{sk})$.
- The adversary $\mathcal{A}$ is given the public key pk.
- For $q = \mathsf{poly}(\lambda)$ rounds, the adversary $\mathcal{A}$ submits a message m, and receives $\sigma_{\gg} \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$, which is a stream.

- The adversary $\mathcal{A}$ outputs m' and streams a signature $\sigma'_{\gg}$. The output of the experiment is $\mathsf{Ver}(\mathsf{pk}, m', \sigma'_{\gg})$.

Notice that traditionally, we would require m' to be distinct from the messages m's queried before, but here we have no such requirement. With this experiment in mind, we now define the security requirement for a disappearing signature scheme.

Definition 4.5.1. *Let λ, n be security parameters. A disappearing signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is secure under memory bound $S(n)$, if for all PPT adversaries $\mathcal{A}$ that use up to $S(n)$ memory bits,*

$$\Pr\left[\mathsf{SigForge}_{\mathcal{A},\Pi}(\lambda, n) = 1\right] \leq \mathsf{negl}(\lambda).$$

To construct such a disappearing signature scheme, one tool that we will use alongside online obfuscation is a *prefix puncturable signature*.

4.5.2 Prefix Puncturable Signature

A *prefix puncturable signature* is similar to a regular public key signature scheme that works for messages of the form (x, m), where x is called the *prefix*. Additionally, it has a puncturing procedure Punc that takes as input the secret key sk and a prefix x^*, and outputs a punctured secret key sk_{x^*}. sk_{x^*} allows one to sign any message of the form (x, m) with $x \neq x^*$. The security requirement is that, given sk_{x^*}, one cannot produce a signature on any message of the form (x^*, m).

To put formally, in addition to the usual correctness and security requirements of a signature scheme, we also have a correctness requirement and a security requirement for the punctured key.

Definition 4.5.2 (Correctness of the Punctured Key)**.** *Let λ be the security parameter. We require that for all $m \in \{0,1\}^*$ and $x, x^* \in \{0,1\}^\lambda$ s.t. $x \neq x^*$:*

$$\Pr\left[\sigma = \sigma' : \begin{array}{c} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sig}(\mathsf{sk},(x,m)) \\ \mathsf{sk}_{x^*} \leftarrow \mathsf{Punc}(\mathsf{sk},x^*) \\ \sigma' \leftarrow \mathsf{Sig}(\mathsf{sk}_{x^*},(x,m)) \end{array}\right] = 1.$$

Definition 4.5.3 (Security of the Punctured Key)**.** *Let λ be the security parameter. We require that for all $x^* \in \{0,1\}^\lambda$ and $m \in \{0,1\}^*$, for all PPT adversaries $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{Ver}(\mathsf{pk},(x^*,m),\sigma) = 1 : \begin{array}{c} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{sk}_{x^*} \leftarrow \mathsf{Punc}(\mathsf{sk},x^*) \\ \sigma \leftarrow \mathcal{A}(\mathsf{sk}_{x^*},\mathsf{pk},(x^*,m)) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Bellare and Fuchsbauer [13] have shown that such a signature scheme can be built from any one-way function.

4.5.3 Construction

We now present our construction of the disappearing signature scheme.

Construction 4.5.1. *Let λ, n be the security parameters. Let $\mathsf{PPS} = (\mathsf{Gen},\mathsf{Sig},\mathsf{Ver},\mathsf{Punc})$ be a prefix puncturable signature scheme, and $o\mathcal{O} = (\mathsf{Obf},\mathsf{Eval})$ be an online obfuscator with 1-time VGB security under $S(n)$ memory bound. The construction $\Pi = (\mathsf{Gen},\mathsf{Sig},\mathsf{Ver})$ works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^n)$: *Run* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PPS.Gen}(1^\lambda)$*, and output* $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Sig}(\mathsf{sk}, m)$: *Construct the program P as follows:*

$$P_{\mathsf{sk},m}(x) = \mathsf{PPS.Sig}(\mathsf{sk}, (x, m)).$$

  Obfuscate the above program to obtain a stream $\sigma_{\gg} \leftarrow \mathsf{Obf}(P)$. The signature is simply the stream $\sigma_{\gg}$.

- $\mathsf{Ver}(\mathsf{pk}, m, \sigma_{\gg})$: *Sample a random prefix $x^* \in \{0,1\}^\lambda$, and evaluate the streamed obfuscated program using x^* as input. This yields*

$$\sigma^* = \mathsf{Eval}(\sigma_{\gg}, x^*) = \mathsf{PPS.Sig}(\mathsf{sk}, (x^*, m)).$$

  Then, output $\mathsf{PPS.Ver}(\mathsf{pk}, (x^*, m), \sigma^*)$ *as the result.*

The correctness of the construction comes directly from the correctness of the underlying prefix puncturable signature scheme.

4.5.4 Proof of Security

Theorem 4.5.1. *If* PPS *is a correct and secure prefix puncturable signature scheme, and $\mathcal{O}$ is an online obfuscator with* 1*-time VGB security under $S(n)$ memory bound, then Construction 4.5.1 is secure under $S(n)$ memory bound.*

Proof. We prove the security of Construction 4.5.1 through a sequence of hybrids.

Recall what happens in the signature forgery game H_0. At the end of the game, the adversary $\mathcal{A}$ outputs a message m' and a signature $\sigma'_{\gg}$. We verify it by sampling a random $x^* \in$

$\{0,1\}^\lambda$, obtain $\sigma^* = \mathsf{Eval}(\sigma'_{\gg}, x^*) = \mathsf{PPS.Sig}(\mathsf{sk}, (x^*, m'))$, and then output $\mathsf{PPS.Ver}(\mathsf{pk}, (x^*, m'), \sigma^*)$.

Now imagine that in H_1, we sample $x^* \in \{0,1\}^\lambda$ at the very beginning of the game. We also obtain a punctured key $\mathsf{sk}_{x^*} \leftarrow \mathsf{PPS.Punc}(\mathsf{sk}, x^*)$, which we don't use yet. H_1 should be indistinguishable from H_0 for any adversary, since x^* and sk_{x^*} are never sent to the adversary.

Then we move to H_2, where we modify the way the signature is generated in response to the adversary's *last* query. Now instead of sending the obfuscation of the program P, we send the obfuscation of the following program P':

$$P'_{\mathsf{sk}_{x^*}, m, x^*}(x) = \begin{cases} \mathsf{PPS.Sig}(\mathsf{sk}_{x^*}, (x, m)) & \text{if } x \neq x^* \\ \\ \bot & \text{otherwise} \end{cases}.$$

Notice that this program rejects the input x^*, but produces a valid signature on all other inputs. The only point where P and P' differ is on input x^*.

Note that before the obfuscation of P' is streamed back to the adversary, x^* is information theoretically hidden. Therefore, to distinguish between H_1 and H_2, the adversary needs to distinguish between two obfuscated programs which differ on a single input that is information theoretically hidden. By the same argument as in Lemma 4.4.2, this would break the 1-time VGB security of the underlying online obfuscator. Therefore, no adversary with up to $S(n)$ memory bits can distinguish between H_1 and H_2 with non-negligible probability.

Now we repeat the process to modify our response to the adversary's second-to-last query and obtain H_3, all the way until we reach H_{q+1}, where now all the signatures streamed to the adversary use P' instead of P. Since here we have a sequence of a polynomial number of hybrids that no adversary with a $S(n)$ memory bound can distinguish one from the next with

non-negligible probability, no adversary with a $S(n)$ memory bound can distinguish H_{q+1} from H_0. Notice that in H_0, the adversary plays the original security game. However, in H_{q+1}, all the responses to the queries use P' instead of P.

Now notice that the entire game H_{q+1} can be simulated using only the punctured secret key sk_{x^*}. If an adversary is able to win this game, then we can use this adversary to obtain a signature on (x^*, m) for some m, even if we only have sk_{x^*}. This directly contradicts with the security of the underlying prefix puncturable signature scheme. Therefore, no PPT adversary $\mathcal{A}$ with $S(n)$ memory bound can win any of H_0, H_1, ..., H_{q-1}. Since H_0 is the original signature forgery experiment, we conclude that Construction 4.5.1 is secure.

□

4.6 Functional Encryption

4.6.1 Definition

First we recall the adaptive security definition for functional encryption, which utilizes the following experiment:

Functional Encryption Security Experiment $\mathsf{Dist}^{\mathsf{FE}}_{\mathcal{A},\Pi}(\lambda)$:

- Run $\mathsf{Setup}(1^\lambda)$ to obtain keys $(\mathsf{mpk}, \mathsf{msk})$ and sample a uniform bit $b \in \{0, 1\}$.
- The adversary $\mathcal{A}$ is given the master public key mpk.
- For a polynomial number of rounds, the adversary submits a circuit $C \in \{\mathcal{C}_\lambda\}$, and receives $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$.

- The adversary $\mathcal{A}$ submits the challenge query consisting of 2 messages m_0 and m_1 s.t. $C(m_0) = C(m_1)$ for any circuit C that has been queried before, and receives $\mathsf{Enc}(\mathsf{mpk}, m_b)$.

- For a polynomial number of rounds, the adversary submits a circuit $C \in \{\mathcal{C}_\lambda\}$ s.t. $C(m_0) = C(m_1)$, and receives $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$.

- The adversary $\mathcal{A}$ outputs a guess b' for b. If $b' = b$, we say that the adversary succeeds and the output of the experiment is 1. Otherwise, the experiment outputs 0.

Definition 4.6.1 (Adaptive Security)**.** *A functional encryption scheme* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be secure if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\left[\mathsf{Dist}^{\mathsf{FE}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Now we discuss how we define functional encryption in the bounded storage model. As we have seen in the PKE with disappearing ciphertext security construction, the core idea here is similar: we now produce ciphertexts that are *streams*.

Concretely, for security parameters λ and n, a functional encryption scheme in the bounded storage model consists of a tuple of PPT algorithms $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ that each uses up to $O(n)$ memory bits. The rest of the syntax is identical to that of the classical FE scheme, except that now the ciphertexts $\mathsf{ct}_{\gg}$ are streams. The correctness requirement remains unchanged apart from the syntax change, but the security definition would need to be supplemented with a memory bound for the adversary and a slightly different security experiment $\mathsf{Dist}^{\mathsf{FE\text{-}BSM}}_{\mathcal{A},\Pi}$. $\mathsf{Dist}^{\mathsf{FE\text{-}BSM}}_{\mathcal{A},\Pi}$ is identical (apart from syntax changes) to $\mathsf{Dist}^{\mathsf{FE}}_{\mathcal{A},\Pi}$ except

that for function key queries submitted after the challenge query, we no longer require that $C(m_0) = C(m_1)$.

Definition 4.6.2 (Adaptive Security in the Bounded Storage Model)**.** *A functional encryption scheme* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be secure under memory bound* $S(n)$ *if for all PPT adversaries* $\mathcal{A}$ *that use at most* $S(n)$ *memory bits:*

$$\Pr\left[\mathsf{Dist}^{\mathsf{FE\text{-}BSM}}_{\mathcal{A},\Pi}(\lambda, n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

With these definitions in mind, we now present how one can construct a secure functional encryption scheme in the bounded storage model using online obfuscation. The construction will also be based on three classical cryptographic primitives: a Non-Interactive Zero Knowledge (NIZK) proof system, a secure classical functional encryption scheme, and a Pseudo-Random Function (PRF).

4.6.2 Construction

Construction 4.6.1. *Let* λ, n *be the security parameters. Let* $\mathsf{NIZK} = (\mathcal{P}, \mathcal{V})$ *be a non-interactive zero knowledge proof system,* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *a functional encryption scheme,* $\mathsf{PRF} : \{0,1\}^w \times \{0,1\}^* \to \{0,1\}^w$ *a pseudorandom function for* $w = \mathsf{poly}(\lambda)$*, and* $o\mathcal{O} = (\mathsf{Obf}, \mathsf{Eval})$ *an online obfuscator with* 1*-time VGB security under memory bound* $S(n)$*. We construct the functional encryption scheme* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as follows:*

- $\mathsf{Setup}(1^\lambda, 1^n)$*: Sample* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$*. Sample the common reference string* crs *for the NIZK system. Output* $(\mathsf{mpk}, \mathsf{crs})$ *as the overall public key. Output* msk

as the master secret key.

- $\mathsf{KeyGen}(\mathsf{msk}, C)$*: Sample random* $x, y \in \{0,1\}^w$*. Consider the following function:*

$$F_{C,x,y}(m,k) = \begin{cases} C(m) & \text{if } k = \bot \text{ or } \mathsf{PRF}(k,(C,y)) \neq x \\ \bot & \text{otherwise} \end{cases}.$$

Compute $\mathsf{sk}_F \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, F_{C,x,y})$*. Also, produce a NIZK proof* π *that* sk_F *is correctly generated, i.e. the tuple* $(\mathsf{mpk}, C, x, y, \mathsf{sk}_F)$ *is in the language*

$$\mathcal{L}_{\mathsf{mpk},C,x,y,\mathsf{sk}_F} := \left\{ (\mathsf{mpk}, C, x, y, \mathsf{sk}_F) \middle| \begin{array}{l} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda) \\ \mathsf{sk}_F \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, F_{C,x,y}) \end{array} \right\}.$$

Output the function key as $\mathsf{sk}_C = (C, x, y, \mathsf{sk}_F, \pi)$*.*

- $\mathsf{Enc}((\mathsf{mpk}, \mathsf{crs}), m)$*: Compute* $c \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (m, \bot))$*. Then consider the following program that takes as input a function key* $\mathsf{sk}_C = (C, x, y, \mathsf{sk}_F, \pi)$*:*

$$P_{c,\mathsf{mpk},\mathsf{crs}}(\mathsf{sk}_C) = \begin{cases} \mathsf{FE.Dec}(\mathsf{sk}_F, c) & \text{if } \mathsf{NIZK}.\mathcal{V}(\mathsf{crs}, (\mathsf{mpk}, C, x, y, \mathsf{sk}_F), \pi) = 1 \\ \bot & \text{otherwise} \end{cases}.$$

Obfuscate the above program to obtain a stream $\mathsf{ct}_{\gg} \leftarrow \mathsf{Obf}(P)$*. The ciphertext is simply the stream* $\mathsf{ct}_{\gg}$*.*

- $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_{\gg})$: *Simply output* $\mathsf{Eval}(\mathsf{ct}_{\gg}, \mathsf{sk}_C)$*.*

It should be easy to verify that an honest execution yields

$$P_{c,\mathsf{mpk},\mathsf{crs}}(C, x, y, \mathsf{sk}_F, \pi) = \mathsf{FE.Dec}(\mathsf{sk}_F, c) = F_{C,x,y}(m, \perp) = C(m)$$

as desired.

4.6.3 Proof of Security

We prove the security of Construction 4.6.1 via a sequence of hybrid experiments.

Sequence of Hybrids

- H_0: The adversary plays the functional encryption game $\mathsf{Dist}^{\mathsf{FE\text{-}BSM}}_{\mathcal{A},\Pi}(\lambda, n)$ where $b = 0$, i.e. it always receives $\mathsf{Enc}(\mathsf{mpk}, m_0)$.

- H_1: The same as H_0, except that when answering the challenge query by the adversary, we sample a random key $k \in \{0,1\}^w$. Notice that we don't change anything in the response to the challenge query yet. For any function key query that happens after the challenge query, instead of sampling $x \in \{0,1\}^w$ randomly, we set $x = \mathsf{PRF}(k, (C, y))$, where C is the circuit being queried on by the adversary.

- H_2: The same as H_1, except that when answering the challenge query, we compute $c' \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (m_b, k))$ instead of $c \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (m_b, \perp))$.

- H_3: The same as H_2, except that now the crs and the proof π of the NIZK system are generated by the NIZK simulator.

- H_4: The same as H_3, except that now we set $b = 1$ instead of 0.

- H_5: Switch back to the original method of generating crs and the proof π for the NIZK system.
- H_6: Switch back to use c instead of c'.
- H_7: Switch back to sampling random x for the function key queries that happen after the challenge query.

Proof of Hybrid Arguments

Lemma 4.6.1. *If* PRF *is a secure pseudorandom function, then no PPT adversary can distinguish between* H_0 *and* H_1 *with non-negligible probability.*

Proof. Notice that only difference between H_0 and H_1 is that instead of sampling a random x, x is computed as $\mathsf{PRF}(k, (C, y))$ where k is unknown to the adversary. The indistinguishability between H_0 and H_1 comes directly from the pseudorandomness of the underlying PRF.

Concretely, we show how on can use an adversary $\mathcal{A}$ that distinguishes H_0 from H_1 to construct an adversary $\mathcal{A}'$ that distinguishes the underlying PRF from a truly random function. When $\mathcal{A}'$ is given a function f in question, $\mathcal{A}'$ would simulate for $\mathcal{A}$ the functional encryption security game $\mathsf{Dist}^{\mathsf{FE\text{-}BSM}}_{\mathcal{A},\Pi}$ with $b = 0$. The only difference is that once after $\mathcal{A}$ has sent the challenge query, in the following function key queries, $\mathcal{A}'$ would sample a random y, and compute the x's as $x = f(C, y)$. Notice that in the case where f is a PRF, we would have $x = \mathsf{PRF}(k, (C, y))$, whereas if f is a truly random function, we would end up having a uniformly random x. Notice that these two cases exactly corresponds to H_1 and H_0, respectively. If $\mathcal{A}$ determines that it is in H_0, $\mathcal{A}'$ outputs that the function f is a truly random function. Otherwise, $\mathcal{A}'$ claims that the function f is a pseudorandom function. If

$\mathcal{A}$ succeeds with a non-negligible probability, $\mathcal{A}'$ succeeds with non-negligible probability as well.

□

Lemma 4.6.2. *If the* NIZK *system is statistically sound,* PRF *is a secure pseudorandom function against non-uniform attackers, and the online obfuscator oO has* 1*-time VGB security under memory bound $S(n)$, then no PPT adversary with memory bound $S(n)$ can distinguish between H_1 and H_2 with non-negligible probability.*

Proof. The difference between H_1 and H_2 is that we now use c' instead of c. However, notice that c and c' are never used directly, but only hardcoded into the program P. Therefore, the only way that an adversary can distinguish between H_1 and H_2 is by distinguishing the two obfuscated programs. Let P be the program obfuscated in H_1 that has c hardcoded and P' be the program obfuscated in H_2 with c' hardcoded. Let us consider how P and P' differ in functionality.

Notice that $\mathsf{NIZK}.\mathcal{V}(\mathsf{crs}, (\mathsf{mpk}, C, x, y, \mathsf{sk}_F), \pi)$ does not depend on c or c', so P and P' will always fall into the same branch. Without loss of generality, here we consider the non-trivial branch, where the NIZK proof verifies correctly and the program outputs $\mathsf{FE}.\mathsf{Dec}(\mathsf{sk}_F, c)$. Since the NIZK proof checks out and that the NIZK system has statistical soundness, we have that sk_F is a correctly generated function key. Therefore, the program P outputs $\mathsf{FE}.\mathsf{Dec}(sk_F, c) = F_{C,x,y}(m, \perp)$, and the program P' outputs $F_{C,x,y}(m, k)$. Notice that $F_{C,x,y}(m, \perp)$ always yields $C(m)$, and that $F_{C,x,y}(m, k)$ yields $C(m)$ unless $\mathsf{PRF}(k, (C, y)) = x$. In other words, P and P' always have the same output except for on inputs where $\mathsf{PRF}(k, (C, y)) = x$.

Now recall that as the obfuscated program is being streamed, k has just been freshly sampled and not used anywhere else. Therefore, k is information theoretically hidden from the

adversary. Since PRF is a pseudorandom function against non-uniform attackers, the value of $\mathsf{PRF}(k, (C, y))$ should also be information theoretically hidden from the adversary. Now that we have P and P' differing only on inputs that are information theoretically hidden, by a similar argument as in Lemma 4.4.2, by the 1-time VGB security of the online obfuscator, any PPT adversary under memory bound $S(n)$ should not be able to distinguish between the obfuscations of P and P' with non-negligible probability. Consequently, no PPT adversary with memory bound $S(n)$ can distinguish between H_1 and H_2 with non-negligible probability.

□

Lemma 4.6.3. *If the* NIZK *system is zero-knowledge, then no PPT adversary can distinguish between* H_2 *and* H_3 *with non-negligible probability.*

This lemma follows directly from the definition of zero-knowledgeness for NIZK.

Lemma 4.6.4. *If the underlying functional encryption scheme* FE *is secure, then no PPT adversary can distinguish between* H_3 *and* H_4 *with non-negligible probability.*

Proof. The only difference between H_3 and H_4 is that a different value of c is computed. In H_3, $c \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (m_0, k))$, while in H_4, $c \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (m_1, k))$. We show that if an adversary $\mathcal{A}$ can distinguish between H_3 and H_4, then there is an adversary $\mathcal{A}'$ for the $\mathsf{Dist}^{\mathsf{FE}}_{\mathcal{A}',\Pi}$ game that uses $\mathcal{A}$ as a subroutine:

- When $\mathcal{A}'$ receives the public key mpk from the challenger, use the NIZK simulator to sample the crs, and send $(\mathsf{mpk}, \mathsf{crs})$ to $\mathcal{A}$.
- Whenever $\mathcal{A}$ submits a function key query for circuit C before the challenge query, $\mathcal{A}'$ samples random x, y, and sends $F_{C,x,y}$ to the challenger. In response, $\mathcal{A}'$ receives sk_F.

$\mathcal{A}'$ then runs the NIZK simulator to produce the proof π. $\mathcal{A}'$ sends $(C, x, y, \mathsf{sk}_F, \pi)$ back to $\mathcal{A}$.

- When $\mathcal{A}$ submits a challenge query with m_0 and m_1, $\mathcal{A}'$ samples k and sends (m_0, k) and (m_1, k) as its own challenge query to the challenger. When $\mathcal{A}'$ receives the ciphertext c, $\mathcal{A}'$ constructs $P_{c,\mathsf{mpk},\mathsf{crs}}$ and sends the obfuscation of P back to $\mathcal{A}$.
- For function key queries received after the challenge query, follow the same procedure as above, except that now use $x = \mathsf{PRF}(k, (C, y))$.
- If $\mathcal{A}$ says that it is in H_3, output 0. Otherwise, output 1.

We verify that the $\mathsf{Dist}^{\mathsf{FE}}_{\mathcal{A}',\Pi}$ game that $\mathcal{A}'$ plays is valid: (1) For all the function key queries F that are sent before the challenge query, either $F_{C,x,y}(m_0, k) = C(m_0) = C(m_1) = F_{C,x,y}(m_1, k)$, or $F_{C,x,y}(m_0, k) = F_{C,x,y}(m_1, k) = \perp$. (2) For all function key queries F that are sent after the challenge query, $F_{C,x,y}(m_0, k) = F_{C,x,y}(m_1, k) = \perp$.

Notice that $\mathcal{A}'$ simulates the exact game for $\mathcal{A}$ where it needs to distinguish between H_3 and H_4. So if $\mathcal{A}$ succeeds with non-negligible probability, $\mathcal{A}'$ also succeeds with non-negligible probability, which contradicts with the security of the underlying FE scheme.

Thus, by the security of the underlying FE scheme, no PPT adversary can distinguish between H_3 and H_4 with non-negligible probability.

□

Lemma 4.6.5. *If the* NIZK *system is zero-knowledge, then no PPT adversary can distinguish between H_4 and H_5 with non-negligible probability.*

This lemma follows directly from the definition of zero-knowledgeness for NIZK.

Lemma 4.6.6. *If the* NIZK *system is statistically sound,* PRF *is a secure pseudorandom function against non-uniform attackers, and the online obfuscator $o\mathcal{O}$ has* 1*-time VGB security under memory bound $S(n)$, then no PPT adversary with memory bound $S(n)$ can distinguish between H_5 and H_6 with non-negligible probability.*

The proof of this lemma follows analogously from the one of lemma 4.6.2.

Lemma 4.6.7. *If* PRF *is a secure pseudorandom function, then no PPT adversary can distinguish between H_6 and H_7 with non-negligible probability.*

The proof of this lemma follows analogously from the one of lemma 4.6.1.

Theorem 4.6.1. *If* NIZK *is zero-knowledge and statistically sound,* PRF *is a secure pseudorandom function against non-uniform attackers,* FE *is a secure functional encryption scheme, and the online obfuscator $o\mathcal{O}$ has* 1*-time VGB security under $S(n)$ memory bound, then Construction 4.6.1 is secure under $S(n)$ memory bound.*

Proof. The lemmas above show a sequence of a polynomial number of hybrid experiments where no PPT adversary with $S(n)$ memory bound can distinguish one from the next with non-negligible probability. Notice that the first hybrid H_0 corresponds to the functional encryption security game where $b = 0$, and the last hybrid H_7 corresponds to one where $b = 1$. The security of the construction follows.

□

4.7 Candidate Construction 1

4.7.1 Matrix Branching Programs

A *matrix branching program* BP of length h, width w, and input length ℓ consists of an input selection function $\mathsf{inp} : [h] \to [\ell]$, $2h$ matrices $\{\mathbf{M}_{i,b} \in \{0,1\}^{w \times w}\}_{i \in [h]; b \in \{0,1\}}$, a left bookend that is a row matrix $\mathbf{s} \in \{0,1\}^{1 \times w}$, and a right bookend that is a column matrix $\mathbf{t} \in \{0,1\}^{w \times 1}$. BP is evaluated on input $x \in \{0,1\}^{\ell}$ by computing $\mathsf{BP}(x) = \mathbf{s} \left(\prod_{i \in [h]} \mathbf{M}_{i, x_{\mathsf{inp}(i)}} \right) \mathbf{t}$.

We say that a family of matrix branching programs are *input-oblivious* if all programs in the family share the same parameters h, w, ℓ, and the input selection function inp.

Lemma 4.7.1 (Barrington's Theorem [12]). *For a circuit C of depth d where each gate takes at most 2 inputs, we can construct a corresponding matrix branching program BP with width 5 and $h = 4^d$.*

4.7.2 The Basic Framework

Here we present the basic framework of an online obfuscator based on matrix branching programs. Our framework will be parameterized by a randomized procedure $\mathsf{Convert}$, which takes as input a log-depth circuit C and width w, and produces a branching program of length $h = \mathsf{poly}(\lambda)$ and width w. w will be chosen so that the honest parties only need $O(w)$ space to evaluate the program as it is streamed, while security is maintained even if the adversary has up to Cw^2 space, for some small constant C.

Since the branching program BP will be too large for a space bounded obfuscator to write down, we will assume that there is a local, space-efficient way to compute each entry of the branching program, given the circuit C and the random coins of $\mathsf{Convert}$.

Note that Barrington's theorem implies, for log-depth circuits, that $h = \mathsf{poly}(\lambda)$ and that w can be taken as small as 5. Convert can be thought of as some procedure to expand the width to match the desired space requirements, and also enforce other security properties, as discussed in Section 4.7.3, where we discuss our particular instantiation of the framework.

Our basic framework actually consists of three schemes. As we will demonstrate, the three schemes have equivalent security, under the assumed existence of a pseudorandom function. The first scheme is much simpler, highlights the main idea of our construction, and allows us to more easily explore security. The downside of the first scheme is that the obfuscator requires significant space, namely more than the adversary. We therefore present two additional schemes with equivalent security, where the final scheme allows the obfuscator to run in space $O(w)$, while having equivalent security to the original scheme.

Construction with Kilian Randomization.

We start with the first and simpler scheme, denoted $\mathcal{O}_{\text{Kil}}$, that uses randomization due to Kilian[74] to construct a matrix branching program BP' as follows.

Sample random invertible matrices $\mathbf{R}_i \in \{0,1\}^{w \times w}$ for $i = 0, 1, \ldots, h$. Compute $\mathbf{M}'_{i,b} = \mathbf{R}_{i-1}^{-1} \mathbf{M}_{i,b} \mathbf{R}_i$ for $i \in [h]$ and $b \in \{0,1\}$. Additionally, compute new bookends $\mathbf{s}' = \mathbf{s} \cdot \mathbf{R}_0$, and $\mathbf{t}' = \mathbf{R}_h^{-1} \cdot \mathbf{t}$. The new randomized matrix branching program is now $\mathsf{BP}' = (\mathsf{inp}, \{\mathbf{M}'_{i,b}\}_{i \in [h]; b \in \{0,1\}}, \mathbf{s}', \mathbf{t}')$. Notice that when we compute $\mathsf{BP}'(x)$, these random matrices will cancel each other out and hence the output of the program should be unchanged.

Now to turn BP' into an online obfuscator, all we need to do is to properly stream the branching program. Here we specify the order that the matrices will be streamed:

$$\mathbf{s}', \mathbf{M}'_{1,0}, \mathbf{M}'_{1,1}, \mathbf{M}'_{2,0}, \mathbf{M}'_{2,1}, \ldots, \mathbf{M}'_{h,0}, \mathbf{M}'_{h,1}, \mathbf{t}'.$$

When streaming a matrix $\mathbf{M}$, we require that the matrix $\mathbf{M}$ is streamed column by column, i.e. we start by sending the first column of $\mathbf{M}$, followed by the second column, then the third, so on and so forth.

Now let's take a look at how to evaluate the obfuscated program, i.e. the matrix branching program sent over the stream. Notice that we would need to do this using only space linear to w.

To evaluate the program, we will keep a row matrix $\mathbf{v} \in \{0,1\}^{1\times w}$ as our partial result. When the streaming begins, we will set $\mathbf{v} = \mathbf{s}'$ received over the stream.

For $i \in [h]$, we will compute $b = x_{\mathsf{inp}(i)}$ and listen to the stream of $\mathbf{M}'_{i,b}$. Let the columns of $\mathbf{M}'_{i,b}$ be $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_w$. Since $\mathbf{M}'_{i,b}$ is streamed column by column, we will receive on the stream $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_w$. As the columns are being streamed, we will compute an updated partial result $\mathbf{v}' = (v_1, v_2, \ldots, v_w)$ on the fly. As we receive $\mathbf{c}_j$ for $j \in [w]$, we would compute $v_j = \mathbf{v} \cdot \mathbf{c}_j$. After all the columns of $\mathbf{M}'_{i,b}$ have been streamed and that $\mathbf{v}'$ has been fully computed, we set $\mathbf{v} = \mathbf{v}'$.

In the end after we receive $\mathbf{t}'$, we output $\mathsf{BP}'(x) = \mathbf{v} \cdot \mathbf{t}'$.

Notice that throughout the evaluation process, we use at most $2w$ memory bits, which is linear to w.

However, one issue with this construction is that running the obfuscator requires computing products of matrices of size $w \times w$, and this inherently requires $O(w^2)$ space. Next up, we will show how we can use pseudorandom functions (PRFs) to help us carry out the randomization process using only space linear to w.

Construction with Elementary Random Row and Column Operations.

We will now give an alternate construction based on elementary row operations $\mathcal{O}_{\text{ER}}$, which will improve on the space requirements of the obfuscator. Namely, the obfuscator will still have a large source of randomness, which we will assume can be queried many times. However, other than the randomness, the only additional space that is required will be $O(w)$.

Since we are working mod 2, there is no scaling, so the only elementary row operations are (1) $\mathbf{B}_{i,j}$ which adds row j to row i, and (2) $\mathbf{C}_{i,j}$ which swaps rows i, j. $\mathbf{B}_{i,j}, \mathbf{C}_{i,j}$ are also represented as matrices, obtained by performing the relevant row operation to the identity matrix. Notice that $\mathbf{C}_{i,j} = \mathbf{B}_{i,j} \cdot \mathbf{B}_{j,i} \cdot \mathbf{B}_{i,j}$. Therefore, we consider just the $\mathbf{B}_{i,j}$. Also notice that $\mathbf{B}_{i,j}^{-1} = \mathbf{B}_{i,j}$ since we are working mod 2. Finally, note that $\mathbf{B}_{i,j}$ corresponds to the *column* operation which adds column i into column j. It will be convenient to let $\mathbf{B}_{i,i}$ to denote the identity matrix.

$\mathcal{O}_{\text{ER}}$ will sample the Kilian randomizing matrices $\mathbf{R}$ from $\mathcal{O}_{\text{Kil}}$ by sampling a sequence $\mathbf{B}^{(1)}, \ldots, \mathbf{B}^{(\tau)}$ of row operations, and setting $\mathbf{R} = \prod_{t=1}^{\tau} \mathbf{B}^{(t)}$ and $\mathbf{R}^{-1} = \prod_{t=\tau}^{1} \mathbf{B}^{(t)}$. Note that each $\mathbf{B}$ matrix is specified by a pair $(i, j) \in [w], i \neq j$. For each matrix $\mathbf{R}_i$, we generate such a sequence. We will explain how to sample the row operations shortly. First, we explain, given query access to the $\mathbf{B}$'s (or really, the (i, j) pairs), how to compute the obfuscated program stream.

We need to explain how to construct and stream BP'. To generate the bookend vector $\mathbf{s}' = \mathbf{s} \cdot \mathbf{R}_0$, start with $\mathbf{s}' = \mathbf{s}$, write $\mathbf{R}_0$ as $\prod_{t=1}^{\tau} \mathbf{B}^{(0,t)}$, interpret each of the $\mathbf{B}^{(0,t)}$ as a column operation, and apply the appropriate column operation to $\mathbf{s}'$ in order from $t = 1, \ldots, \tau$. To generate the other bookend vector $\mathbf{t}' = \mathbf{R}_b \cdot \mathbf{t}$, we start with $\mathbf{t}' = \mathbf{t}$, write $\mathbf{R}_b^{-1}$ as $\prod_{t=\tau}^{1} \mathbf{B}^{(b,t)}$, interpret each of the $\mathbf{B}^{(b,t)}$ as a row operation, and apply the appropriate row operation to $\mathbf{t}'$.

Both operations clearly take only space $O(w)$, in addition to the storage requirements for the $\mathbf{B}$ matrices.

For the $\mathbf{M}'_{i,b} = \mathbf{R}_{i-1}^{-1}\mathbf{M}_{i,b}\mathbf{R}_i$, more care is needed. First, we need a sub-routine which, for input α, computes $\mathbf{r}_\alpha$, the α-th row of $\mathbf{M}_{i,b} \cdot \mathbf{R}_i$. This sub-routine works almost exactly the same as our computation of $\mathbf{s}'$ above. The β-th entry of $\mathbf{r}_\alpha$ gives the entry (α, β) of $\mathbf{M}_{i,b} \cdot \mathbf{R}_i$. We can thus compute $\mathbf{c}_\beta$, the β-th column of $\mathbf{M}_{i,b} \cdot \mathbf{R}_i$, element by element.

To compute an entry (α, β) of $\mathbf{M}'_{i,b}$, we first compute the corresponding column $\mathbf{c}_\beta$. We then compute $\mathbf{c}'_\beta = \mathbf{R}_{i-1}^{-1} \cdot \mathbf{c}_\beta$, analogous to how we computed $\mathbf{t}'$. Then we output entry α of $\mathbf{c}'_\beta$.

Now we explain how to sample the sequence $\mathbf{B}^{(1)}, \ldots, \mathbf{B}^{(\tau)}$. We will use the following lemma:

Lemma 4.7.2. *There exist constants C_0, C_1 such that the following is true. For every w, there exists a sequence of integers $d_1, \ldots, d_\tau$ and distributions $D_1, \ldots, D_\tau$, $\tau \leq C_0 w, d_t \leq C_1 w$, where each D_t is a distribution over a sequence of d_t of the $\mathbf{B}$ matrices. The guarantee is that if the sequences $\mathbf{B}^{(t,1)}, \cdots, \mathbf{B}^{(t,d_t)}$ are sampled from D_t (each sequence independently), then $\mathbf{R} = \prod_{t=1}^{\tau} \prod_{i=1}^{d_t} \mathbf{B}^{(t,i)}$ is distributed identically to a uniform random $\mathbf{R}$ mod 2, conditioned on $\mathbf{R}$ being invertible.*

Proof. The proof follows ideas from Randall[84].

The base case $w = 1$ is trivial: the only invertible matrix mod 2 is 1. So we set $\tau = 0$ in this case.

We now assume the lemma holds true for $w - 1$. Thus, there is a sequence of $C_0(w-1)$ distributions over sequences of $C_1(w-1)$ row operations generating a random $(w-1) \times (w-1)$ matrix $\mathbf{R}'$. We will construct $\mathbf{R}$ from $\mathbf{R}'$ as follows.

- First let

$$\mathbf{R}_0 = \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{R}' \end{pmatrix} .$$

- Next, construct $\mathbf{R}_1$, which fills in the zeros of the first row with uniform random bits:

$$\mathbf{R}_1 = \begin{pmatrix} 1 & \mathbf{x} \\ 0 & \mathbf{R}' \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{x} \\ 0 & \mathbf{I} \end{pmatrix} \cdot \mathbf{R}_0$$

for a random row vector $\mathbf{x}$. Note that the matrix $\begin{pmatrix} 1 & \mathbf{x} \\ 0 & \mathbf{I} \end{pmatrix}$ can be constructed from a sequence of $w-1$ row operations. Also note that $\mathbf{R}_1$ is a uniformly random matrix, conditioned on the first column being 10^{w-1} and the matrix being invertible. This follows from the fact that, for matrices with the given first column, having determinant 1 (the only invertible possibility mod 2) is equivalent to having $\det(\mathbf{R}_0) = 1$. Thus, a random invertible matrix with the given first column is identical to choosing a random $\mathbf{x}$, and then choosing a random invertible $\mathbf{R}_0$.

- Next, sample a random non-zero column vector $\mathbf{y} \in \{0,1\}^w \setminus 0^w$. Let $\mathbf{C}$ be any invertible matrix such that $\mathbf{y} = \mathbf{C} \cdot 10^{w-1}$. As explained by[84], $\mathbf{C}$ is actually a bijection between the set of invertible matrices whose first column is $\mathbf{y}$ and the set of invertible matrices whose first column is 10^{w-1}.

  Thus, setting $\mathbf{R}_2 = \mathbf{C} \cdot \mathbf{R}_1$ will result in a uniformly random matrix $\mathbf{R}_2$.

  Note that $\mathbf{C}$ can be taken to be constructed from a sequence of $w+2$ of the $\mathbf{B}$ matrices: 3 to swap the first column with some non-zero position of $\mathbf{y}$, and then $w-1$ additional

ones to fill in the remaining positions of $\mathbf{y}$.

Thus, we can take $d_t \leq w + 2$, and we have that $C_0 w = \tau \leq 2 + C_0(w-1)$. We can take $\tau = 2w$ to solve the recurrence. This completes the proof. □

Thus, we will use Lemma 4.7.2 to construct the distributions D_t, and then sample the matrices $\mathbf{B}^{(t,i)}$ from D_t. Lemma 4.7.2 shows that the $\mathbf{R}$ matrices, and hence the view of the adversary, are indistinguishable.

Eliminating Space with PRFs.

We now turn to the final construction, which eliminates all but $O(w)$ from the obfuscator's space requirements.

$\mathcal{O}_{\text{PRF}}$ will work exactly as $\mathcal{O}_{\text{ER}}$, except that instead of sampling truly random samples from D_t, it will do the following. For each $\mathbf{R}$ matrix, it will sample a uniformly random key k for a pseudorandom function PRF. Then matrix sequence $\mathbf{B}^{(t,i)}$ will be computed as $D_t(\cdot; \mathsf{PRF}(k,t))$. That is, it will use $\mathsf{PRF}(k,t)$ as the random coins needed by D_t. In this way, it can generate the $\mathbf{B}^{(t,i)}$ matrices on the fly, without having to store them. Since each sequence of $\mathbf{B}$ matrices has size at most $O(w)$, it can generate the matrices space efficiently. By the security of the PRF, the following is immediate:

Lemma 4.7.3. *For any choice of* Convert*, assuming* PRF *is a secure PRF and $\mathcal{O}_{Kil}$ is k-time VGB secure when using* Convert*, then $\mathcal{O}_{PRF}$ is k-time VGB secure when using* Convert*.*

Thus, it suffices to analyze $\mathcal{O}_{\text{Kil}}$ for a given choice of algorithm Convert; then we can instantiate $\mathcal{O}_{\text{PRF}}$ with Convert, and be guaranteed that security will carry over.

4.7.3 Instantiating Convert

Now we will discuss how we specifically instantiate Convert, constructing the branching program BP for a circuit C that we plug into our framework.

To motivate our construction, we recall that Barrington's theorem [12] plus Kilian randomization[74] already provides *some* very mild security: given the matrices corresponding to an evaluation on any chosen input x (which selects one matrix from each matrix pair), the set of matrices information-theoretically hides the entire program, save for the output of the program on x.

This one-time security, however, is clearly not sufficient for full security. For starters, the adversary can perform *mixed-input* attacks, where it selects a single matrix from each pair, but for multiple reads of the same input, it chooses different matrices. This allows the attacker to treat the branching program as a read-once branching program. It may be that, by evaluating on such inputs, the adversary learns useful information about the program.

Another problem is linear-algebraic attacks. The rank of each matrix is preserved under Kilian randomization. Assuming all matrices are full-rank (which is true of Barrington's construction), the eigenvalues of $\mathbf{M}_{i,0} \cdot \mathbf{M}_{i,1}^{-1}$ are preserved under Kilian randomization.

In branching program obfuscation starting from[47], multilinear maps are used to block these attacks. In our setting, we will instead use the storage bounds on the attacker. First, we observe that Raz[85] essentially shows that linear-algebraic attacks are impossible if the attacker cannot even record the matrices being streamed. While we do not know how to apply Raz's result to analyze our scheme, we conjecture that for appropriately chosen matrices, it will be impossible to do linear-algebraic attacks.

The next main problem is input consistency. To accomplish this, we will do the follow-

ing. We will first run Barrington's theorem to get a branching program consisting of 5×5 matrices. We will then construct an "input consistency check" branching program, and glue the two programs together.

As a starting point, we will construct a *read-once* matrix branching program BP_1 (one that reads each input bit exactly once) that outputs 0 on an all-zero or all-one input string, and outputs 1 on all other inputs. Looking forward, we will insert this program into the various reads of a single input bit: any honest evaluation will cause the branching program to output 0, whereas an evaluation that mixes different reads of this bit will cause the program to output 1.

Matrix Branching Program BP_1:

- The width, the length, and the input length of the branching program are all L.
- inp is the identity function, i.e. $\mathbf{M}_{i,b}$ reads x_i as input.
- For $i \in [L]$, $\mathbf{M}_{i,0} = \mathbf{I}_L$ where $\mathbf{I}_L$ is the $L \times L$ identity matrix. $\mathbf{M}_{i,1}$ is the $L \times L$ permutation matrix representing shifting by 1. Specifically,

$$\mathbf{M}_{i,1} = \begin{pmatrix} 0^{(L-1)\times 1} & \mathbf{I}_{L-1} \\ 1 & 0^{1\times(L-1)} \end{pmatrix}.$$

- The left bookend is $\mathbf{s} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix}$ and the right bookend is $\mathbf{t} = \begin{pmatrix} 0 & 1 & 1 & \cdots & 1 \end{pmatrix}^T$.

We now briefly justify why BP_1 works as desired. Let $0 \leq w \leq L$ be the Hamming weight of the input x. Notice that when evaluating $\mathsf{BP}_1(x)$, the number of $\mathbf{M}_{i,1}$ matrices chosen is exactly w, and the rest of the chosen matrices are all $\mathbf{M}_{i,0}$, the identity matrix. Therefore, the product of all the $\mathbf{M}$ matrices is equivalent to a permutation matrix representing shifting by w. When this product is left-multiplied by $\mathbf{s} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix}$, we get a resulting row matrix $\mathbf{s}'$ that is equivalent to $\mathbf{s}$ right-shifted by w. Notice that $\mathbf{s}'$ has a single 1 at position $(w \mod L) + 1$. When multiplying $\mathbf{s}'$ by the right bookend $\mathbf{t}$, the result will always be 1, unless $(w \mod L) + 1 = 1$. The only w values that satisfy $(w \mod L) + 1 = 1$ are $w = 0$ and $w = L$, which correspond to $x = 0^L$ and $x = 1^L$ respectively. Hence BP_1 gives us the desired functionality.

Next up, we will expand BP_1 to a read-once matrix branching program BP_2 with the following functionality: for a set S of input bits, BP_2 outputs 0 if and only if all the input bits within S are identical (the input bits outside of S can be arbitrary). This is accomplished by simply setting the matrices for the inputs in S to be from BP_1, while the matrices for all other inputs are just identity matrices.

Next, we describe a simple method of taking the "AND" of two matrix branching programs with the same length, input length and input function. Given matrix branching programs $\mathsf{BP}^A = (\mathsf{inp}, \{\mathbf{M}^A_{i,b}\}_{i\in[h];b\in\{0,1\}}, \mathbf{s}^A, \mathbf{t}^A)$ and $\mathsf{BP}^B = (\mathsf{inp}, \{\mathbf{M}^B_{i,b}\}_{i\in[h];b\in\{0,1\}}, \mathbf{s}^B, \mathbf{t}^B)$ with length h and input length ℓ, we construct a new brancing program BP^C such that $\mathsf{BP}^C = \mathsf{BP}^A(x) \cdot \mathsf{BP}^B(x)$ for all inputs x:

Constructing $\mathsf{BP}^C = \mathsf{AND}(\mathsf{BP}^A, \mathsf{BP}^B)$:

- The length, the input length, and the input function of BP^C are also h, ℓ and inp, respectively. The width of BP^C is $w_C = w_A \cdot w_B$, where w_A and w_B are the widths of BP^A and BP^B, respectively.
- For all $i \in [h]$ and $b \in \{0,1\}$, compute $\mathbf{M}^C_{i,b} = \mathbf{M}^A_{i,b} \otimes \mathbf{M}^B_{i,b}$ where $\otimes$ denotes the matrix tensor product (Kronecker product). Notice that the widths of $\mathbf{M}^A_{i,b}$, $\mathbf{M}^B_{i,b}$, and $\mathbf{M}^C_{i,b}$ are w_A, w_B, and $w_A w_B$ as desired.
- The left bookend is $\mathbf{s}^C = \mathbf{s}^A \otimes \mathbf{s}^B$, and the right bookend is $\mathbf{t}^C = \mathbf{t}^A \otimes \mathbf{t}^B$.

Using the mixed-product property of matrix tensor products, it should be easy to verify that $\mathsf{BP}^C(x) = \mathsf{BP}^A(x) \cdot \mathsf{BP}^B(x)$ as desired.

Next, let BP_* be a random read-once matrix branching program with input length L and width $m = \mathsf{poly}(\lambda)$. We can sample such a branching program by uniformly sampling each of its matrices and bookends.*

We will assume that the program computed by BP_* gives a pseudorandom function. This is, unfortunately not strictly possible: write $x = (x_1, x_2)$ for two contiguous chunks of input bits x_1, x_2. Then the matrix $\left(\mathsf{BP}_*(x_1, x_2)\right)_{x_1 \in X_1, x_2 \in X_2}$ for any sets X_1, X_2 will have rank at most m. By setting X_1, X_2 to be larger than m, one can distinguish this matrix consisting of outputs of BP_* from a uniformly random one. The good news is that this attack requires a large amount of space, namely m^2. If the attacker's space is limited to be somewhat less than

*When this is later put through the basic framework, we would need to generate these random matrices using a PRF. This would allow us to reconstruct it at a later point.

m^2, this plausibly leads to a pseudorandom function. We leave justifying this conjecture as an interesting open question.

Now consider the branching program $\mathsf{BP}_3 = \mathsf{AND}(\mathsf{BP}_2, \mathsf{BP}_*)$. Notice that BP_3 has width nm and is equal to o on inputs x where $\forall i, j \in S, x_i = x_j$, and is equal to $\mathsf{BP}_*(x)$ on all other x.

With these tools in hand, we are now ready to show how to enforce input consistency on an existing matrix branching program.

Given a matrix branching program $\mathsf{BP} = (\mathsf{inp}, \{\mathbf{M}_{i,b}\}_{i\in[h];b\in\{0,1\}}, \mathbf{s}, \mathbf{t})$ with length h, width w and input length ℓ, we construct the branching program BP' as follow:

Input Consistent Branching Program BP':

- BP' has the same length h, input length ℓ, and input function inp. The width is now $w + mh$ where $m = \mathsf{poly}(\lambda)$.
- For all $j \in [\ell]$, let S_j be the set of all reads of x_j, i.e. $S_j = \{i | i \in [h], \mathsf{inp}(i) = j\}$. Construct the branching program $\mathsf{BP}_2^{(j)}$ using the BP_2 construction with input length h and $S = S_j$. Overwrite the input function of $\mathsf{BP}_2^{(j)}$ with inp so that it now takes $x \in \{0,1\}^\ell$ as input. Notice that $\mathsf{BP}_2^{(j)}(x) = 0$ if and only if all reads of the j-th bit of x are identical.

  Sample a fresh random matrix branching program $\mathsf{BP}_*^{(j)}$ with length h, width m, input length ℓ and input function inp. Compute $\mathsf{BP}_3^{(j)} = \mathsf{AND}(\mathsf{BP}_2^{(j)}, \mathsf{BP}_*^{(j)})$. Denote the matrices in $\mathsf{BP}_3^{(j)}$ as $\{\mathbf{M}_{i,b}^{(j)}\}_{i\in[h];b\in\{0,1\}}$, and the bookends as $\mathbf{s}^{(j)}, \mathbf{t}^{(j)}$.
- For all $i \in [h]$, and $b \in \{0,1\}$, construct the matrix $\mathbf{M}'_{i,b}$ by adding all the $\mathbf{M}_{i,b}^{(j)}$'s

to the diagonal as $\mathbf{M}'_{i,b} = \mathsf{diag}(\mathbf{M}_{i,b}, \mathbf{M}^{(1)}_{i,b}, \ldots, \mathbf{M}^{(\ell)}_{i,b})$. Notice that the width of $\mathbf{M}'_{i,b}$ is $w + \sum_{j\in[\ell]} m|S_j| = w + mh$.

- The left bookend is now $\mathbf{s}' = \begin{pmatrix} \mathbf{s} & \mathbf{s}^{(1)} & \mathbf{s}^{(2)} & \cdots & \mathbf{s}^{(\ell)} \end{pmatrix}$ and the right bookend is now $\mathbf{t}' = \begin{pmatrix} \mathbf{t}^T & \left(\mathbf{t}^{(1)}\right)^T & \left(\mathbf{t}^{(2)}\right)^T & \cdots & \left(\mathbf{t}^{(\ell)}\right)^T \end{pmatrix}^T$.

Notice that we have

$$\mathsf{BP}'(x) = \mathsf{BP}(x) + \sum_{j\in[\ell]} \mathsf{BP}^{(j)}_3(x) = \mathsf{BP}(x) + \sum_{j\in[\ell]} \mathsf{BP}^{(j)}_2(x)\mathsf{BP}^{(j)}_*(x).$$

If all reads of the input x are consistent, then we have $\mathsf{BP}^{(j)}_2(x) = 0$ for all j, and the program outputs the original output $\mathsf{BP}'(x) = \mathsf{BP}(x)$.

If the reads of the input x are not consistent, then $\mathsf{BP}^{(j)}_2(x) = 1$ for some j, and consequently $\mathsf{BP}^{(j)}_*(x)$ will be added to the program output. By our conjecture that $\mathsf{BP}^{(j)}_*(x)$ acts as a PRF to space-bounded attackers, we thus add a pseudorandom value to $\mathsf{BP}(x)$, hiding its value. Thus, we presumably force input consistency. BP' will be the output of Convert, which we then plug into our framework.

4.8 Candidate Construction 2

Now we present the second candidate construction from digital time-stamping and standard-model obfuscation. The concept of a digital time-stamp was first introduced by Haber and Stornetta[66], and since then we have seen various instantiations of digital time-stamping sys-

tems. One construction of particular interest is by Moran, Shaltiel and Ta-Shma[78], where they construct a non-interactive time-stamping scheme in the bounded storage model. This will be what we base our construction on.

Definition 4.8.1 (Non-Interactive Digital Time-stamp in the Bounded Storage Model)**.** *Let λ, n be the security parameters. A non-interactive digital time-stamp scheme in the bounded storage model with stamp length $\ell = O(n)$ consists of a tuple of PPT algorithms $\Pi = (\mathsf{Stream}, \mathsf{Stamp}, \mathsf{Ver})$ that each uses up to $O(n)$ memory bits:*

- $\mathsf{Stream}(1^\lambda, 1^n) \to (s_{\gg}, k)$ *takes as input security parameters λ, n and outputs a stream $s_{\gg}$ and a short sketch k of the stream.*
- $\mathsf{Stamp}(s_{\gg}, x) \to \sigma$ *takes as input the stream $s_{\gg}$ and an input $x \in \{0,1\}^*$, and outputs a stamp $\sigma \in \{0,1\}^\ell$.*
- $\mathsf{Ver}(k, x, \sigma) \to 0/1$ *takes as input the sketch k, an input $x \in \{0,1\}^*$ and a stamp σ and outputs a single bit 0 or 1.*

We require correctness and security of the digital time-stamp scheme.

Definition 4.8.2 (Correctness)**.** *We require that for all $x \in \{0,1\}^*$, we have*

$$\Pr\left[\mathsf{Ver}(k, x, \sigma) = 1 : (s_{\gg}, k) \leftarrow \mathsf{Stream}(1^\lambda, 1^n), \sigma \leftarrow \mathsf{Stamp}(s_{\gg}, x)\right] = 1.$$

For security, we ideally want that an adversary cannot produce a valid time-stamp on an input x that the adversary did not run Stamp on. Instead, [78] notice that an adversary with $S(n)$ memory bits can store at most $S(n)/\ell$ time-stamps, and therefore define security as upper bounding the number of time-stamps an adversary can produce. While not the

same as the ideal goal, it at least implies the adversary cannot produce arbitrary time-stamped messages.

Definition 4.8.3 (Security)**.** *We require that for all adversary $\mathcal{A}$ that uses up to $S(n)$ memory bits, we have*

$$\Pr\left[\forall(x,\sigma)\in M, \mathsf{Ver}(k,x,\sigma)=1 \;\middle|\; \begin{array}{c}(s_{\gg},k)\leftarrow \mathsf{Stream}(1^\lambda)\\ M\leftarrow \mathcal{A}^{\mathsf{Stamp}(\cdot)}(s_{\gg})\\ |M|>\frac{S(n)}{\ell}\\ \forall(x_1,\sigma_1),(x_2,\sigma_2)\in M, x_1\neq x_2\end{array}\right]\leq \mathsf{negl}(\lambda).$$

Now we show how we can use such a digital time-stamping scheme to construct an online obfuscator.

Construction 4.8.1. *Let λ, n be the security parameters. Let* TSP *be a digital time-stamping scheme in the bounded storage model. Let* $\mathsf{VGB}=(\mathsf{Obf},\mathsf{Eval})$ *be a classical VGB obfuscator for all circuits. We construct our online obfuscator for the circuit class $\{\mathcal{C}_\lambda\}$ as follows:*

- $\mathsf{Obf}(C)$*: Run* $\mathsf{TSP.Stream}(1^\lambda,1^n)$ *to stream $s_{\gg}$ and obtain the sketch k. Consider the following program $P_{C,k}$:*

$$P_{C,k}(x,\sigma)=\begin{cases}C(x) & \text{if } \mathsf{TSP.Ver}(k,x,\sigma)=1\\ \bot & \text{otherwise}\end{cases}.$$

  Let $\mathcal{P}\leftarrow \mathsf{VGB.Obf}(P_{C,k})$ be the standard-model *VGB obfuscation of $P_{C,k}$. The obfuscated program is simply the stream $s_{\gg}$ followed by $\mathcal{P}$.*

- $\mathsf{Eval}((s_{\gg}, \mathcal{P}), x)$: *To evaluate the obfuscated program, first compute* $\sigma \leftarrow \mathsf{TSP.Stamp}(s_{\gg}, x)$ *when* $s_{\gg}$ *is being streamed. Then the output is simply* $\mathsf{VGB.Eval}(\mathcal{P}, (x, \sigma))$.

Correctness is straightforward. One detail is that, using the time-stamping protocol of[78], the sketch k, and thus $P_{C,k}$ will be of size $O(n)$ bits. Thus, we need to use an obfuscator such that VGB.Obf only expands the program by a constant factor. We conjecture that the constant-overhead construction of[5] will work here. Alternatively, one can use branching-program based obfuscation directly from multilinear maps, for example[47] and follow-ups.[18] even gives evidence that these constructions may be VGB secure. The difficulty is that the constructions blow up the input program by a polynomial factor, and therefore cannot be written down. However, as they have the form of a branching program, they can be streamed much the same way as we stream Candidate Construction 1. We therefore conjecture that some instantiation of VGB.Obf will lead to a secure online VGB obfuscator that can also be streamed in low space. We leave proving or disproving this conjecture as an open question.

5

Incompressible Cryptography

5.1 Introduction

Security breaches are ubiquitous. Therefore, it is natural to wonder: will encrypted messages remain secure, even if the secret decryption key is later leaked? Forward secrecy deals exactly with this problem, but requires either multi-round protocols or key updates, both of which may be undesirable in many scenarios. And in the usual time-bounded adversary model, unfortunately, such limitations are inherent: an adversary can simply store the ciphertext and wait for the secret key to leak, at which point it can easily decrypt.

In this chapter we ask: can we force a would-be "save-it-for-later adversary" to actually store the ciphertext in its entirety, for the entire length of time it is waiting for the secret key to leak? At a minimum such storage may be inconvenient, and for very large files or long time frames, it may be prohibitively costly. Even for short messages, one may artificially increase the ciphertext size, hopefully forcing the adversary to use much more storage than message length. We may therefore hope that such an *incompressible encryption scheme* maintains the privacy of messages even if the secret key is later revealed.

Remark 5.1.1. *For an illustrative example, an individual with a gigabit internet connection can transmit ∼10TB per day, potentially much more than their own storage. Of course many entities will have 10TB or even vastly more, but an incompressible scheme would force them to devote 10TB to storing a particular ciphertext for potentially years until the key is revealed. Across millions or billions of people, even powerful adversaries like state actors would only be able to devote such storage to a small fraction of victims.*

Unfortunately, traditional public key encryption schemes are not incompressible; an adversary may be able to store only a short digest of the ciphertext and still obtain non-trivial

information about the plaintext once the secret key is leaked. For example, for efficiency reasons, hybrid encryption is typically used in the public key setting, where the encryption of a message m may look like:

$$(\mathsf{Enc}(\mathsf{pk}, s) \ , \ G(s) \oplus m) \ .$$

Here, s is a short seed, and G is a pseudorandom generator used to stretch the random seed into a pseudorandom pad for the message m. A save-it-for-later adversary need not store the entire ciphertext; instead, they can store just $\mathsf{Enc}(\mathsf{pk}, s)$ as well as, say, the first few bits of $G(s) \oplus m$. Once the secret key is revealed, they can learn s and then recover the first few bits of m. This may already be enough to compromise the secrecy of m. Such an attack is especially problematic if we wanted to artificially increase the ciphertext size by simply padding the message and appending dummy bits, since then the first few bits of m would contain the entire secret plaintext.

The compressibility issue is not limited to the scheme above: we could replace $G(s) \oplus m$ with a different efficient symmetric key encryption scheme such as CBC-mode encryption, and essentially the same attack would work. The same goes for bit encryption as well.

Incompressible public key encryption instead requires that if the adversary stores anything much smaller than the ciphertext, the adversary learns absolutely nothing about the message, even if the secret key later leaks.

Remark 5.1.2. *We note that plain public key encryption does have* some *incompressibility properties. In particular, it is impossible, in a plain public key encryption scheme, for the adversary to significantly compress the ciphertext and later be able to reconstruct the original ciphertext. However, this guarantee implies nothing about the privacy of the underlying message should the key leak.*

Incompressible Signatures. A canonical application of signatures is to prevent man-in-the-middle attacks: by authenticating each message with a signature, one is assured that the messages were not tampered with. However, a man-in-the-middle can always *delay* sending an authenticated message, by storing it for later. The only way to block such attacks in the usual time-bounded adversary model is to use multi-round protocols, rely on synchronized clocks and timeouts, or have the recipients keep state, all of which may be undesirable. We therefore also consider the case of incompressible *signatures*, which force such a delaying adversary to actually store the entire signature for the duration of the delay.

In slightly more detail, in the case of plain signatures, a forgery is a signature on any *new* message, one the adversary did not previously see signed. The reason only new signed messages are considered forgeries is because an adversary can simply store a valid signature it sees, and later reproduce it. An *incompressible* signature, essentially, requires that an adversary who produces a valid signature on an existing message must have actually stored a string almost as large as the signature. By making the signatures long, we may hope to make it prohibitively costly to maintain such storage. As in the case of encryption, existing signature schemes do not appear to offer incompressible security; indeed, it is usually desired that signatures are very short.

Feature: Low-storage for streaming honest users. Given that communication will be inconveniently large for the adversary to store, a desirable feature of incompressible ciphertexts and signatures is that they can be sent and received with low storage requirements for the honest users. In such a setting, the honest users would never store the entire ciphertext or signature, but instead generate, send, and process the communication bit-by-bit in a streaming fashion.

Feature: High rate. With incompressible ciphertexts and signatures, communication is set to be deliberately large. If the messages themselves are also large, it may be costly to further blow up the communication in order to achieve incompressibility. Therefore, a desirable feature is to have the rate—the ratio of the maximum message length to the communication size—be as close to 1 as possible. In this way, for very large messages, there is little communication overhead to make the communication incompressible.

5.1.1 Prior Work

Dziembowski[45] constructed information-theoretically secure symmetric-key incompressible encryption (referred to as forward-secure encryption) via randomness extractors. The focus of our work is on public-key encryption and signature schemes, which inherently cannot be information-theoretically secure.*

Also, notice that the notion of incompressible cryptography is very similar to *disappearing* public key encryption and digital signatures introduced in the previous chapter, except with an important distinction: disappearing cryptography assume both honest and malicious parties operate as space-bounded streaming algorithms throughout their operation. Honest users are assumed to have a somewhat lower storage bound than the adversary's.

In terms of the functionality requirement for honest users, disappearing cryptography corresponds to the low-storage streaming variant of incompressible cryptography. However, in terms of the security requirement, disappearing cryptography is somewhat weaker, since

*The symmetric-key scheme of[45] also only offers one-time security. However, a simple hybrid argument shows that this implies many-time security, where the adversary can compress each of many ciphertexts *separately* and later sees the secret key. However, it inherently does not offer any security if the adversary can *jointly* compress many ciphertexts, even if the compressed value is much smaller than a single ciphertext! In contrast, public-key incompressible encryption automatically ensures security in such setting via a simple hybrid argument.

it restricts the adversary to also be space-bounded throughout its entire operation, and observe the ciphertexts/signatures produced by the cryptosystem in a streaming manner. On the other hand, incompressible cryptography allows the adversary to observe each ciphertext/signature in its entirety and compute on it using an unrestricted amount of local memory, but then store some small compressed version of it afterwards. Some disappearing schemes may be insecure in the incompressible threat model: for example, one of the disappearing ciphertext schemes from the previous chapter could potentially even be based on *symmetric key* cryptography, despite being a public key primitive.* Yet public key incompressible ciphertexts easily imply public key encryption, which is believed to be stronger than symmetric key cryptography[70].

In summary, incompressible cryptography with low-storage streaming is also disappearing, but the reverse direction does not hold.

5.1.2 Our Results

We give new positive results for incompressible cryptography:

- Under the minimal assumption of standard-model public key encryption, we construct a simple incompressible public key encryption scheme. The scheme supports streaming with constant storage, independent of the ciphertext size. As a special case, we achieve provably secure disappearing ciphertexts with optimal honest-user storage and under mild assumptions, significantly improving on disappearing cryptography from the previous chapter. The ciphertext size is $|c| = |S| + |m| \times \mathsf{poly}(\lambda)$, where $|S|$ is the adversary's storage, $|m|$ the message size, and λ the security parameter.

*It's not hard to see that one-way functions, and therefore symmetric key cryptography, are implied by disappearing ciphertexts, since the secret key can be information-theoretically recovered from the public key.

- Under the minimal assumption of one-way functions, we construct incompressible signatures. Our scheme supports streaming with constant storage, independent of the signature size. Thus we also achieve provably secure disappearing signatures under minimal assumptions, again significantly improving on disappearing cryptography from the previous chapter. The total communication (message length plus signature size) is $|S| + |m| + \mathsf{poly}(\lambda)$.

- Under standard-model indistinguishability obfuscation (iO), we construct "rate 1" incompressible public-key encryption, where $|c| = |S| + \mathsf{poly}(\lambda)$ and the message length can be as large as roughly $|S|$. In particular, for very large messages, the ciphertext size is roughly the same as the message size.

  The public keys of our scheme are small, but the secret keys in this scheme are at least as large as the message, which we explain is potentially inherent amongst provably-secure high-rate schemes.

  Along the way, we give the first rate-1 construction of functional encryption for circuits, where $|c| = |m| + \mathsf{poly}(\lambda)$.

- We consider a notion of "rate-1" incompressible signatures, where the total communication is only $|S| + \mathsf{poly}(\lambda)$, and the message can be as large as roughly $|S|$. Note that the signature by itself must have size at least $|S|$ for incompressibility (since m may be compressible), and so if we separately send the message and signature, the total communication would be at least $|S| + |m|$, which is not rate 1. Instead, we just send a signature and require the message to be efficiently extractible from the signature.

  We show that rate-1 incompressible signatures are *equivalent* to incompressible encod-

ings, defined by Moran and Wichs[79]. By relying on the positive results of Moran and Wichs[79], we obtain such signatures under either the Decisional Composite Residuosity (DCR) or Learning With Errors (LWE) assumption, in either the CRS or random oracle model. The random oracle version supports low-space streaming, as does the CRS model if we assume the (large) CRS is streamed. On the other hand, by relying on the negative results of Moran and Wichs[79], we conclude that a provably secure rate-1 construction in the standard model is unlikely.

5.1.3 Other Related Work

Big-Key Cryptography in the Bounded Retrieval Model. The study of big-key cryptography in the Bounded Retrieval Model (BRM) has evolved through a series of works[44,37,31,4,3,14]. The high-level difference is that in the BRM, the secret keys are made large to prevent exfiltration, while the communication (e.g., ciphertexts, signatures) are kept small. Incompressible cryptography is the reverse: we make the communication large to prevent an adversary from being able to remember it in its entirety, while the secret key is ideally small. On a technical level, while there are some high-level similarities such as relying on a combination of computational and information-theoretic techniques, the concrete schemes are quite different.

Symmetric Cryptography with Memory-Bounded Adversaries. There has been various studies into the symmetric-key setting where the adversaries are memory-bounded. For instance, the work by Rivest[89] introduces *all-or-nothing encryption*, a symmetric-key encryption scheme such that only knowing some individual bits of the ciphertext reveals no information about the message, even if the adversary is later given the secret key. This is similar to the forward-secure encryption due to Dziembowski[45], except that in forward-secure

encryption, the adversary is allowed to compute an arbitrary function (with a small-sized output) of the ciphertext, instead of only knowing a few individual bits of it. So all-or-nothing encryption can be thought of as disappearing encryption in the symmetric-key setting, whereas forward-secure encryption is closer to the symmetric-key version of incompressible encryption. The work by Zaverucha[98] further extends the idea of all-or-nothing encryption, constructing a password-based encryption scheme. Building on this, the work by Biryukov and Khovratovich[17] constructs memory-hard encryption by combining the idea from Zaverucha[98] together with an external memory-hard function, which allows for high memory bounds even with a small block size. All of these prior works are in the symmetric-key setting, and it is not obvious how to extend them to the public-key setting as we study in this chapter.

Rate-1 Incompressible Encryption from Standard Assumptions. In a later followup work, Branco, Döttling, and Dujmovic[25] constructed rate-1 incompressible encryption with CCA security from programmable hash proof systems (HPS), plain-model incompressible encodings[79] and a pseudorandom generator (PRG). These primitives can be realized from, e.g. the DDH and additionally the DCR or the LWE assumptions,

5.1.4 Technical Overview

Incompressible Encryption. We first consider incompressible public key encryption. The syntax is identical to that of standard-model encryption, but the security game is different:

1. The challenger first gives the adversary the public key.

2. The adversary then produces two messages m_0, m_1.

3. The challenger encrypts one of the two messages, as the ciphertext c.

4. Now the adversary produces a state s of size somewhat smaller than c.

5. The challenger then reveals the secret key.

6. The adversary, given only the small state s but also the secret key, now makes a guess for which message was encrypted.

Note that, except for the size of the state s being bounded between Steps 4 and 6, the size of the adversary's storage is unbounded. It is also easy to see that this definition implies standard semantic security of public-key encryption.

Remark 5.1.3. *Note that this security definition is quite similar to that of disappearing public key encryption from the previous chapter with two distinctions. Firstly, in the disappearing encryption security experiment, there is no Step 4 as above. Instead, the adversary is bounded by some space* throughout the entire experiment*. Additionally, functionality wise, disappearing encryption requires the protocol to be executable by honest parties with some space bound lower than the adversary's storage. In our setting, we do not consider this to be an inherent requirement, but rather a desirable feature that some of our schemes satisfy. As we will see in Remark 5.1.4, this feature is incompatible with rate-1 schemes, and hence we will drop it in that setting.*

Our Solution. We give a construction of incompressible encryption in Section 5.3, under the minimal assumption of generic public key encryption.

We describe our solution using *functional* encryption (FE), which is a form of public key encryption where the secret key holder can give out function secret keys for functions

f; a function secret key allows for learning $f(m)$ but nothing else about the message. For our application, we only need a very special case of single-key functional encryption, which we instantiate with a simple and potentially practical construction from generic public key encryption scheme. Our incompressible encryption scheme works as follows:

- The public key is just the public key for the underlying FE scheme. The secret key is a function secret key for the function f_v defined as

$$f_v(s,b) = \begin{cases} s & \text{if } b = 0 \\ s \oplus v & \text{if } b = 1 \end{cases}$$

  where the value v is chosen uniformly at random and hard-coded into f_v. Here, s, v are reasonably short strings, whose length will be discussed shortly.

- To encrypt m, choose a random s, and compute $c \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (s,0))$ as an encryption of $(s,0)$ under the FE scheme. Then choose a large random string R. Interpret s as the pair (s',t), where t is a string of length equal to the message length, and s' is the seed for a strong extractor. Then compute $z = \mathsf{Extract}(R;s') \oplus t \oplus m$. The final ciphertext is (c, R, z).

- To decrypt, use the FE secret key to recover $s = (s',t)$ from c. Then recover $m = z \oplus \mathsf{Extract}(R;s') \oplus t$.

We can generate and transmit the string R in a streaming fashion. We can then use an online extractor[93] so that $\mathsf{Extract}(R;s')$ can be computed without having to store R in its entirety. Note that R is the only "big" component of the ciphertext, so encryption and decryption therefore require small space.

We prove security through a hybrid argument. First, we use FE security to switch to c being generated as $c \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (s \oplus v, 1))$. Since this c decrypts equivalently under the secret key, this change is indistinguishable.

We then observe that the string $u = s \oplus v$ being encrypted under the FE scheme, as well as the string z included in the final ciphertext, are both just uniformly random strings. We can therefore delay the generation of the secret key and v until the very end of the experiment. Now we think of the adversary's state (as well as some other small values needed to complete the simulation) as a leakage on the large random string R. Since the adversary's storage is required to be small compared to R, R has min-entropy conditioned on this leakage. This means we can invoke the randomness guarantee of the randomness extractor to replace $\mathsf{Extract}(R; s')$ with a uniform random string. At this point, m is one-time-padded with a uniform string, and therefore information-theoretically hidden.

We explain how to instantiate the functional encryption scheme. Since the adversary only ever sees a single secret key, we can build such a functional encryption scheme generically from public key encryption, using garbled circuit techniques[57]. On the other hand, our functional encryption scheme only needs to support an extremely simple linear function. We show a very simple and potentially practical solution from any public key encryption scheme.

Remark 5.1.4. *We note that our scheme has a less-than-ideal rate, since the ciphertext size is at least as large as the adversary's storage* plus *the length of the message. Low rates, however, are inherent to schemes supporting low-storage streaming. Indeed, the storage requirements of the honest users must be at least as large as the message, and in the high-rate case this means the honest users must be capable of storing the entire ciphertext. This remains true* even if the message itself is streamed bit-by-bit*, which can be seen as follows: by incompressibility, the decrypter*

cannot start outputting message bits until essentially the entire stream has been sent. Otherwise, an attacker can store a short prefix of the ciphertext, and then when it gets the secret key mimic the decrypter until it outputs the first message bit. Now, at the point right before the decrypter outputs the first message bit, the entire contents of the message must be information-theoretically contained within the remaining communication (which is short) and the decrypter's state, since the decrypter ultimately outputs the whole message. Thus the decrypter's state must be almost as large as the message.

A RATE-1 SOLUTION. We now discuss how we achieve a rate-1 scheme, using indistinguishability obfuscation. This is our most complicated construction, and we only give a brief overview here with the full construction in Section 5.4.

The central difficulty in achieving a rate-1 scheme is that we cannot guarantee a ciphertext with large information-theoretic entropy. Indeed, the ciphertext must be almost as small as the message, so there is little room for added entropy on top of the message. But the message itself, while large, many not have much entropy. Therefore, our approach of using randomness extraction to extract a random string from the ciphertext will not work naively.

Our solution, very roughly, is to have the large random value in the *secret key*. Using a delicate argument, we switch to a hybrid where the ciphertext is just an encryption of large randomness R, and the secret key contains the message, masked by a string extracted from R. Now we can mimic the low-rate case, arguing that given the small state produced by the adversary, R still has min-entropy. Thus, the message m is information-theoretically hidden.

The result is that we achieve an incompressible encryption scheme whose rate matches the rate of the underlying functional encryption scheme. Unlike the low-rate case, our FE scheme appears to need the full power of FE for circuits, since it will be evaluating crypto-

graphic primitives such as PRGs and extractors. Unfortunately, all existing FE schemes for general circuits, even using iO, have poor rate. For example, if we look at the original iO scheme of Garg *et al.*[47], the ciphertext contains *two* plain public key encryption encryptions of the message, *plus* a NIZK proof of consistency. The result is that the rate is certainly at most 1/3. Another construction due to Boyle *et al.*[22] sets the ciphertext to be an obfuscated program containing the message; since known obfuscation schemes incur a large blowup, the scheme is not rate-1.

We give a novel rate-1 FE scheme (with many key security), by building on ideas from Boneh and Zhandry[21]. They build an object called private linear broadcast encryption (PLBE), which can be seen as a special case of FE for simple comparison functionalities. However, their approach readily generalizes to more complex functionalities. The problem with their construction is that their proof incurs a security loss proportional to the domain size. In their case, the domain is polynomial and this is not a problem. But in our case, the domain is the message space, which is exponential. One may hope to use complexity leveraging, but this would require setting the security parameter to be at least as large as the message. However, this will not give a rate-1 scheme since the ciphertext is larger than the message by an additive factor linear in the security parameter.

We therefore devise new techniques for proving security with just a polynomial loss, even for large messages, thus giving the first rate-1 FE scheme for general circuits, from iO and one-way functions. Details in Section 5.7.

Remark 5.1.5. *We note that the final construction of rate-1 incompressible encryption has very short public keys, but large secret keys. We therefore leave as an interesting open question devising a scheme that also has short secret keys. However, achieving such a scheme with provable security*

under standard assumptions appears hard. Indeed, cryptographic assumptions typically make no restrictions on the adversary's storage. The issue is that the message itself may have little entropy, and so to prove that a ciphertext is incompressible it seems the computational assumptions will be used to transition to a hybrid where the ciphertext has nearly full entropy (indeed, this is how our proof works). But this transition happens without space bounds, meaning the reduction actually is capable of decrypting the ciphertext and recovering the message once the key is revealed. Yet in this hybrid the ciphertext was "used up" in order to make it high-entropy, and it seems the only place left to embed the message is the secret key (again, this is how our proof works). If the message is large, it therefore seems the secret key must be large as well. We believe this intuition can be formalized as a black-box separation result, similarly to analogous results of[96]*, but we leave this for future work.*

Incompressible Signatures. An incompressible signature scheme is defined by the following experiment:

1. The challenger first gives the adversary the public key.
2. The adversary makes repeated signing queries on arbitrary messages. In response, the challenger produces a signature on the message.
3. After observing many signatures, the adversary must produce a small state s of size somewhat smaller than a single signature.
4. Next, the adversary, is given the small state s, and wins if it produces a valid signature on *any* message, potentially even one used in a prior signing query.

Note that, except for the size of the state s being bounded between Steps 3 and 4, the size of the adversary's storage is unbounded.

Remark 5.1.6. *This definition is also quite similar to that of disappearing signature from the previous chapter except for two differences. For disappearing signatures, the security experiment does not have Step 3 as above, and instead requires the adversary to be bounded by some space* throughout the entire experiment. *Functionality wise, disappearing signature requires the scheme can be run by honest parties with a space bound somewhat lower that the adversary's storage, whereas we don't require that for incompressible signatures.*

Our Solution. We give a very simple construction of incompressible signatures in Section 5.5. To sign m, first choose a large uniformly random string R, and then compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \ (R, m) \ )$, where Sign is a standard-model signature scheme. The overall signature is then (R, σ). Verification is straightforward.

Both signing and verification can be evaluated in a low-space streaming fashion, provided Sign can be evaluated as such. One can always assume this property of Sign: first hash the message using a streaming-friendly hash function such as Merkle-Damgård, and then sign the hash. Since the hash is small and computing the hash requires low-space, the overall signing algorithm is low space.

For security, consider an adversary which produces a small state s somewhat smaller than the length of R. Since R is random, it will be infeasible for the adversary to re-produce R in Step 4. Therefore, any valid signature must have an R different than any of the messages previously signed. But this then violates the standard unforgeability of Sign.

A RATE-1 SOLUTION. In Section 5.6, we modify the above construction to get a rate-1 solution. We note that "rate" here has to be defined carefully. In the above solution, the signature size is independent of the message size, and so it seems that the signature has good rate. However, communication will involve both the signature *and* the message, and so the total length of the communication will be significantly larger than the message. We therefore want that the *total communication* length is only slightly longer than the message being signed.

On the other hand, if the message is very long, one may naturally wonder whether we can just sign the message using any standard-model signature scheme, and have the resulting communication be rate-1. However, a long message may in fact be compressible. What we want is to achieve rate-1 total communication, and incompressibility, even if the message may be compressed.

We therefore define a rate-1 incompressible signature as an incompressible signature where the signature is only slightly longer than the message, and where there is a procedure to extract the message from the signature. In this way, all that needs to be sent is the signature itself, and therefore the total communication remains roughly the same as the message.

EQUIVALENCE TO INCOMPRESSIBLE ENCODINGS. We next demonstrate that incompressible signatures are equivalent to incompressible encodings[79]. These are public encoding schemes where the encoding encodes a message into a codeword c that is only slightly longer than the message. From c, the original message can be recovered using a decoding procedure. For security, the adversary then receives the codeword as well as the message, tries to compress the codeword into a small storage s. Then the adversary, given *s and the message*, tries to recover the exact codeword c.

A rate-1 incompressible signature (with small public keys) gives an incompressible encod-

ing: to encode a message, simply generate a new public/secret key pair, and sign the message. The codeword c is then the public key together with the signature. Decoding and security follow readily from the message extraction procedure and security of the incompressible signature.

In the other direction, to sign a message, first incompressibly encode the message and then sign the result using a standard-model signature scheme. The final signature is the codeword together with the standard-model signature. Extraction follows from the decoding procedure. If the incompressible encoding supports low-space streaming, so does the signature scheme. For security, since the adversary cannot produce the original codeword that was signed due to the security of the incompressible encoding, they must produce some other codeword. But a valid signature would also contain a standard-model signature on this new codeword, violating the security of the signature scheme.

Moran and Wichs[79] instantiate incompressible encodings under either the Decisional Composite Residuosity (DCR) or Learning With Errors (LWE) assumptions, in either the CRS or random oracle models. We observe that their incompressible encodings simply break the message into blocks of length $\mathsf{poly}(\lambda)$ and encode each block separately; as such they can be easily streamed in low space, though the CRS-based scheme would need the CRS to be streamed as well. We obtain the incompressible signatures under the same assumptions in the same models, with low-space streaming.

We also note that we can have the signer generate the CRS and include it in the public key, giving a standard-model incompressible encoding scheme with large public keys. Note that such a scheme is not immediately equivalent to incompressible encodings, since the codeword contains the public key, and would therefore be too large.

On the other hand,[79] show that a CRS or random oracle is somewhat necessary, by giving a black box separation relative to falsifiable assumptions in the standard model. Due to our equivalence, this implies such a black box impossibility for incompressible signatures in the standard model as well.

5.1.5 Acknowledgements

We would like to thank Ji Luo, Chenzhi Zhu, and the audience members of the CMU CyLab Crypto Seminar for pointing out an issue regarding Definition 5.5.1 for incompressible signatures and helpful discussions on that matter.

5.2 Chapter Preliminaries

Digital Signatures. We generalize the syntax of a signature scheme, which will ultimately be necessary to achieve a meaningful high "rate". Instead of producing a signature that is sent along side the message, we would implicitly embed or *encode* the message into the signature. The signature is then all that is sent to the receiver, from which the message can be decoded and verified. Any standard signature scheme can readily be viewed in our generalized syntax by just calling (m, σ) the "signature."

A public key signature scheme for message space $\{0,1\}^{L_m}$ and signature space $\{0,1\}^{L_\sigma}$ is a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ such that:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$ samples a verification key vk, and a signing key sk.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$ takes as input the signing key sk and a message m, and computes a signature σ *that implicitly contains the message* m.

- $\mathsf{Ver}(\mathsf{vk}, \sigma) \to m/\bot$ takes as input the verification key vk and a signature σ, and outputs either the message m or $\bot$. Outputting m means that the signature verifies, and outputting $\bot$ means that the signature is invalid.

Definition 5.2.1 (Correctness). *For all $\lambda \in \mathbb{N}$ and message $m \in \{0,1\}^{L_m}$, let $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, then we have $\Pr[\mathsf{Ver}(\mathsf{vk}, \mathsf{Sign}(\mathsf{sk}, m)) = m] \geq 1 - \mathsf{negl}(\lambda)$.*

We modify the security experiment slightly by asking the adversary to output a signature σ instead of a message-signature pair, and the adversary wins the game if and only if $\mathsf{Ver}(\mathsf{vk}, \sigma) \notin \{\bot, m_1, \ldots, m_q\}$ where m_i's are the previously queried messages. The "rate" of the signature scheme is defined to be L_m/L_σ.

5.3 Incompressible Encryption: Our Basic Construction

Here we show how to construct an incompressible public key encryption scheme with low "rate", i.e. the ratio of the message size to the ciphertext size. First, we define what it means for a public key encryption scheme to be *incompressible*.

5.3.1 Definition

We give the definition of incompressible encryption, which is based on the similar definition of disappearing encryption[63]. For security parameters λ and S, an incompressible public key encryption scheme with message space $\{0,1\}^{L_m}$ and ciphertext space $\{0,1\}^{L_{\mathsf{ct}}}$ is a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Remark 5.3.1. *For the original disappearing PKE defined in[63], it is additionally required that* Gen, Enc, *and* Dec *can be run in space $N \ll L_{\mathsf{ct}}$. Here, we will consider schemes that have both large and small space.*

The rest of the syntax of an incompressible PKE scheme is identical to that of a classical PKE scheme. The "rate" of the PKE scheme is simply L_m/L_{ct}.

For the security definition, consider the following indistinguishability experiment for an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

Incompressible Encryption Security Experiment $\mathsf{Dist}^{\mathsf{IncomEnc}}_{\mathcal{A},\Pi}(\lambda)$:

1. The adversary $\mathcal{A}_1$, on input 1^λ, outputs a space bound 1^S.
2. Run $\mathsf{Gen}(1^\lambda, 1^S)$ to obtain keys $(\mathsf{pk}, \mathsf{sk})$.
3. Sample a uniform bit $b \in \{0,1\}$.
4. The adversary is then provided the public key pk and submits an auxiliary input aux.
5. The adversary replies with the challenge query consisting of two messages m_0 and m_1, receives $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
6. $\mathcal{A}_1$ produces a state st of size at most S.
7. The adversary $\mathcal{A}_2$ is given the tuple $(\mathsf{pk}, \mathsf{sk}, \mathsf{aux}, \mathsf{st})$ and outputs a guess b' for b. If $b' = b$, we say that the adversary succeeds and the output of the experiment is 1. Otherwise, the experiment outputs 0.

Definition 5.3.1 (Incompressible Encryption Security)**.** *For security parameters λ and S, a public key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has incompressible encryption security if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:*

$$\Pr\left[\mathsf{Dist}^{\mathsf{IncomEnc}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Notice that allowing the adversary to submit and later receive the auxiliary input aux is equivalent to allowing $\mathcal{A}_1, \mathcal{A}_2$ to just have shared randomness at the beginning of the experiment and that, in the non-uniform setting, the definition would be the same without aux since $\mathcal{A}_1, \mathcal{A}_2$ are deterministic w.l.o.g.

Remark 5.3.2. *The original Disappearing Ciphertext Security*[63] *has a very similar security notion, except that the adversary has a space bound of S* throughout the entire experiment*, and that the ciphertext is a long stream sent bit by bit. Notice that our definition of Incompressible Encryption Security is a strictly stronger security definition than Disappearing Ciphertext Security.*

5.3.2 Construction

Construction 5.3.1. *Given* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *a single-key selectively secure functional encryption scheme with a rate of* ρ_{FE} *and a strong average min-entropy extractor* $\mathsf{Extract} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{L_m}$*, with* $d = \mathsf{poly}(\lambda)$ *and* $n = S + \mathsf{poly}(\lambda)$ *the construction* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: First, obtain* $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$*. Then, generate the secret key for the following function* f_v *with a hardcoded* $v \in \{0,1\}^{d+L_m}$*:*

$$f_v(s' = (s,t), \mathsf{flag}) = \begin{cases} s' & \text{if } \mathsf{flag} = 0 \\ s' \oplus v & \text{if } \mathsf{flag} = 1 \end{cases}.$$

  Output $\mathsf{pk} = \mathsf{FE.mpk}$ *and* $\mathsf{sk} = \mathsf{FE.sk}_{f_v} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, f_v)$*.*

- $\mathsf{Enc}(\mathsf{pk}, m)$*: Sample a random tuple* $s' = (s,t)$ *where* $s \in \{0,1\}^d$ *is used as a seed*

for the extractor and $t \in \{0,1\}^{L_m}$ *is used as a one-time pad. The ciphertext consists of three parts:* $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (s', 0))$*, a long randomness* $R \in \{0,1\}^n$*, and* $z = \mathsf{Extract}(R; s) \oplus t \oplus m$*.*

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct} = (\mathsf{FE.ct}, R, z))$*: First, obtain* $s' \leftarrow \mathsf{FE.Dec}(\mathsf{FE.sk}_{f_v}, \mathsf{FE.ct})$*, and then use the seed* s *to compute* $\mathsf{Extract}(R; s) \oplus z \oplus t$ *to recover* m*.*

Note that if Extract is an *online* extractor[93], then encryption and decryption can be run in a low-space streaming fashion, by first sending FE.ct, then streaming R, and then sending z. The rate of this construction is

$$\frac{L_m}{L_{\mathsf{ct}}} = L_m \left(\frac{d + L_m + 1}{\rho_{\mathsf{FE}}} + n + L_m \right)^{-1} = \frac{1}{(1/\rho_{\mathsf{FE}} + 1) + S/L_m} - o(1).$$

Theorem 5.3.1. *Assuming the existence of a functional encryption scheme with single-key selective security and a rate of* $1/\mathsf{poly}(\lambda)$*, and a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, there exists an incompressible PKE with ciphertext size* $S + L_m + \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda) L_m$*, public key size* $\mathsf{poly}(\lambda)$ *and secret key size* $\mathsf{poly}(\lambda)$*. It supports streaming decryption using* $L_m + \mathsf{poly}(\lambda)$ *bits of memory.*

5.3.3 Proof of Security

We organize our proof of security into a sequence of hybrids.

Sequence of Hybrids

- H_0: The original incompressible encryption security experiment $\mathsf{Dist}^{\mathsf{IncomEnc}}_{\mathcal{A},\Pi}$, where the bit b in the experiment is fixed to be 0.

- H_1: In step 5, instead of computing $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (s', 0))$, compute $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (s' \oplus v, 1))$.

- H_2: In step 2, only sample $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. In step 5, after receiving the challenge query, sample uniformly random $z \in \{0,1\}^{L_m}$, $u \in \{0,1\}^{d+L_m}$, $R \in \{0,1\}^n$ and send back $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (u, 1))$, R, and z as the ciphertext. In step 7, sample a uniformly random $s \in \{0,1\}^d$, and compute $t = \mathsf{Extract}(R; s) \oplus z \oplus m_0$, and $v = s' \oplus u$ where s' is the tuple (s, t). Use this v to compute $\mathsf{sk} = \mathsf{FE.sk}_{f_v} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, f_v)$.

- H_3: In step 7, sample a uniformly random $r \in \{0,1\}^{L_m}$ and compute $t = r \oplus z \oplus m_0$ instead.

- H_4: Swap the bit b in the security experiment to be 1 instead of 0.

- H_5: Switch back to the case where $t = \mathsf{Extract}(R; s) \oplus z \oplus m_1$.

- H_6: Switch back to the case where we produce sk in step 2 instead of step 5.

- H_7: Switch the FE ciphertext back to the real one $\mathsf{FE.Enc}(\mathsf{FE.mpk}, (s', 0))$. Notice here we're at the original incompressible encryption security experiment, where the bit b is fixed to be 1.

Proof of Hybrid Arguments

Lemma 5.3.1. *If the functional encryption scheme* FE *has single-key selective security, then no PPT adversary can distinguish between* H_0 *and* H_1 *(respectively* H_6 *and* H_7*) with non-negligible probability.*

Proof. Here we will prove the case for H_0 and H_1. The case for H_6 and H_7 follows analogously. This is by a simple reduction to the single-key selective security of the functional encryption scheme. If an adversary $\mathcal{A}$ is able to distinguish between H_0 and H_1, we show how to construct an adversary $\mathcal{A}'$ that breaks security of the functional encryption scheme FE. The only difference between H_0 and H_1 is that in H_0 the adversary receives an encryption of $(s', 0)$, while in H_1 the adversary receives an encryption of $(s' \oplus v, 1)$. But notice that $f_v(s', 0) = s' = f_v(s' \oplus v, 1)$, so the adversary $\mathcal{A}$ is able to distinguish between two FE ciphertexts that have the same functional output on function f_v, for which it has a secret key. This directly breaks the underlying functional encryption security. Concretely, $\mathcal{A}'$ works as follows by using $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as a subroutine:

- On input 1^λ, sample uniform values s' and v, and submit the challenge query $\mathsf{FE}.m_0 = (s', 0)$ and $\mathsf{FE}.m_1 = (s' \oplus v, 1)$ to the challenger. Receive $\mathsf{FE.mpk}$ and $\mathsf{FE.ct}$ in response.
- Send 1^λ to $\mathcal{A}_1$ and receive 1^S.
- Send $\mathsf{FE.mpk}$ to $\mathcal{A}_1$, receive aux and the challenge query m_0 and m_1, and respond with $\mathsf{FE.ct}$, R and z, where R is a random string of length $S + \mathsf{poly}(\lambda)$, and $z = \mathsf{Extract}(R; s) \oplus t \oplus m_0$. The adversary $\mathcal{A}_1$ produces a state st. Notice that the only component that's different for H_0 and H_1 is $\mathsf{FE.ct}$, and it does not depend on the challenge query from $\mathcal{A}_1$. R and z remain unchanged.
- Send f_v to the challenger and receive $\mathsf{FE.sk}_{f_v}$. Forward $\mathsf{sk} = \mathsf{FE.sk}_{f_v}$ to $\mathcal{A}_2$ together with $(\mathsf{FE.mpk}, \mathsf{aux}, \mathsf{st})$.
- If $\mathcal{A}_2$ outputs that it is in H_0, output 0. Otherwise, output 1.

It is straightforward to verify that if $\mathcal{A}$ wins the game, $\mathcal{A}'$ wins as well. □

Lemma 5.3.2. *No adversary can distinguish between H_1 and H_2 (respectively H_5 and H_6) with non-negligible probability.*

Proof. We prove the case for H_1 and H_2, the case for H_5 and H_6 follows analogously. Since pk does not depend on sk, and sk is not used until in step 7, now instead of fixing f_v (and thus $\mathsf{sk} = \mathsf{FE.sk}_{f_v}$) in step 2, we sample it lazily in step 7. Our new sampling procedure in H_2 makes the following two changes to H_1: First, in H_1, we sample a uniform t and compute $z = \mathsf{Extract}(R; s) \oplus t \oplus m_0$, while in H_2, we sample a uniform z and compute $t = \mathsf{Extract}(R; s) \oplus z \oplus m_0$. This is just a change of variables, and gives two identical distributions. Second, in H_1 we sample a uniform v and encrypt $u = v \oplus s'$, while in H_2 we encrypt a uniform u and compute $v = u \oplus s'$. Again, these are identical distributions. Thus, no adversary can distinguish between H_1 and H_2 with non-negligible probability. □

Lemma 5.3.3. *If the extractor* $\mathsf{Extract}$ *is a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, then no adversary that produces a state* st *of size at most S can distinguish between H_2 and H_3 (resp. H_4 and H_5) with non-negligible probability.*

Proof. We prove the case for H_2 and H_3. The other case follows naturally.

Here let the random variables $X = R$, and $Y = (\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{aux}, u, z)$ and $Z = \mathsf{st}$. By Lemma 2.1.1, we have

$$H_\infty(X|Y,Z) \geq \min_y H_\infty(X|Y=y,Z) \geq \min_y H_\infty(X|Y=y) - S = \mathsf{poly}(\lambda).$$

The last equality above follows since $X = R$ is a uniformly random string, independent of Y, of length $S + \mathsf{poly}(\lambda)$. By extractor security, no adversary can distinguish $(s, \mathsf{Extract}(R; s), Y, Z)$

from (s, U_{L_m}, Y, Z) except with $\mathsf{negl}(\lambda)$ probability. Since we now sample $u \leftarrow U_{L_m}$, no adversary can now distinguish between $t = \mathsf{Extract}(R; s) \oplus z \oplus m_0$ and $t = u \oplus z \oplus m_0$, i.e. H_2 and H_3. □

Lemma 5.3.4. *No adversary can distinguish H_3 from H_4 with non-zero probability.*

Proof. Notice that the only difference between H_3 and H_4 is that in H_3 we have $t = r \oplus z \oplus m_0$ while in H_4 we have $t = r \oplus z \oplus m_1$, where r is uniformly random. Thus t is uniformly random in both cases, and H_3 and H_4 are identical. □

Theorem 5.3.2. *If* FE *is a functional encryption scheme with single-key selective security, and* $\mathsf{Extract}$ *is a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, then Construction 5.3.1 has incompressible encryption security.*

Proof. The lemmas above show a sequence of hybrids where no PPT adversary that produces a state with size at most S can distinguish one from the next with non-negligible probability. The first hybrid H_0 corresponds to the incompressible encryption security game where $b = 0$, and the last one H_7 corresponds to the case where $b = 1$. The security of the indistinguishability game follows. □

5.3.4 Instantiating our FE

We now give a simple construction of functional encryption for our needed functionality. Recall that our functions f_v have the form $f_v(s, \mathsf{flag}) = s \oplus (\mathsf{flag} \cdot v)$.

Construction 5.3.2. *Let* $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *be a public key encryption scheme. Our scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for message length* $n + 1$ *is defined as:*

- $\mathsf{Setup}(1^\lambda)$: *For* $i \in \{1,\ldots,n\}, b \in \{0,1\}$, *run* $(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) \leftarrow \mathsf{Gen}'(1^\lambda)$. *Output* $(\mathsf{mpk} = (\mathsf{pk}_{i,b})_{i,b}\ ,\ \mathsf{msk} = (\mathsf{sk}_{i,b})_{i,b})$.
- $\mathsf{KeyGen}(\mathsf{msk}, f_v) = (\mathsf{sk}_{i,v_i})_i$.
- $\mathsf{Enc}(\mathsf{mpk}, (s, \mathsf{flag}))$: *For* $i \in \{1,\ldots,n\}, b \in \{0,1\}$, *compute* $c_{i,b} = \mathsf{Enc}'(\mathsf{pk}_{i,b},\ s_i \oplus (\mathsf{flag} \cdot b)\ )$. *Output* $c = (c_{i,b})_{i,b}$.
- $\mathsf{Dec}(\mathsf{sk}_{f_v}, c)$: *Output* $x = x_1 x_2 \cdots x_n$ *where* $x_i = \mathsf{Dec}'(\mathsf{sk}_{i,v_i}, c_{i,v_i})$

For correctness, note that $x_i = s_i \oplus (\mathsf{flag} \cdot v_i)$, and therefore $x = s \oplus (\mathsf{flag} \cdot v) = f_v(s, \mathsf{flag})$. Note that the rate of this scheme is $1/\mathsf{poly}(\lambda)$. Thus the overall rate of our incompressible encryption scheme is $1/\mathsf{poly}(\lambda)$.

Theorem 5.3.3. *If* $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *is a CPA secure public key encryption scheme, then Construction 5.3.2 is single key semi-adaptively secure for the functions* f_v.

Proof. Consider a single key semi-adaptive adversary for Construction 5.3.2. Let $m_0 = (s_0, \mathsf{flag}_0), m_1 = (s_1, \mathsf{flag}_1)$ be the challenge messages. For a fixed flag bit, f_v is injective. Therefore, if $m_0 \neq m_1$, it must be that $\mathsf{flag}_0 \neq \mathsf{flag}_1$. Then if the adversary's secret key query is on f_v, we must have $v = s_0 \oplus s_1$. Thus the two possibilities for the challenge ciphertext are the same for c_{i,v_i}, but encrypt opposite bits in $c_{i,1-v_i}$. Since the adversary never gets to see the secret keys $\mathsf{sk}_{i,1-v_i}$, a simple hybrid argument shows that flipping these bits is indistinguishable. □

Corollary 5.3.1. *Assuming the existence of a CPA secure public key encryption scheme and a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, there exists an incompressible PKE with ciphertext size* $S + L_m + \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda)L_m$, *public key size* $\mathsf{poly}(\lambda)$ *and secret key size* $\mathsf{poly}(\lambda)$. *Furthermore, it supports streaming decryption using* $L_m + \mathsf{poly}(\lambda)$ *bits of memory.*

5.4 Rate-1 Incompressible Encryption

Here, we construct incompressible encryption with an optimal rate of $1 - o(1)$, i.e. the message length is (almost) the same as the ciphertext length.

5.4.1 Construction

For our construction, we require a functional encryption scheme with single-key semi-adaptive security and a rate of 1, a strong average min-entropy extractor, and a secure pseudorandom generator (PRG). Our construction works as follows.

Construction 5.4.1. *Given* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *a rate-1 functional encryption scheme satisfying single-key semi-adaptive security,* $\mathsf{Extract} : \{0,1\}^{L_m} \times \{0,1\}^d \to \{0,1\}^n$ *a strong average min-entropy extractor where* $d, n = \mathsf{poly}(\lambda)$*, and* $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^{L_m}$ *a secure PRG, the construction* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: First, obtain* $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$*. Then, generate the secret key for the following function* $f_{v,s}$ *with a hardcoded large random pad* $v \in \{0,1\}^{L_m}$ *and a small extractor seed* $s \in \{0,1\}^d$*:*

$$f_{v,s}(x, \mathsf{flag}) = \begin{cases} x & \text{if } \mathsf{flag} = 0 \\ \mathsf{PRG}(\mathsf{Extract}(x; s)) \oplus v & \text{if } \mathsf{flag} = 1 \end{cases}.$$

  Output $\mathsf{pk} = \mathsf{FE.mpk}$ *and* $\mathsf{sk} = \mathsf{FE.sk}_{f_{v,s}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, f_{v,s})$*. Set* $L_m = S + \mathsf{poly}(\lambda)$*.*

- $\mathsf{Enc}(\mathsf{pk}, m)$*: The ciphertext is simply an encryption of* $(m, 0)$ *using the underlying* FE *scheme, i.e.* $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (m, 0))$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Decryption corresponds to* FE *decryption. The output is* $\mathsf{FE.Dec}(\mathsf{FE.sk}_{f_{v,s}}, \mathsf{ct}) = f_{v,s}(m, 0) = m$ *as desired.*

Let ρ_{FE} be the rate of FE. Then the ciphertext size is $(L_m + 1)/\rho_{\mathsf{FE}}$ and the rate of our incompressible encryption scheme is $\rho_{\Pi} = \rho_{\mathsf{FE}}/(1 + L_m^{-1})$. If $\rho_{\mathsf{FE}} = 1 - o(1)$, then $\rho_{\Pi} = 1 - o(1)$ as well.

Theorem 5.4.1. *Assuming the existence of a functional encryption scheme with single-key semi-adaptive security and a rate of* $1 - o(1)$*, and a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, there exists an incompressible PKE with message size of up to* $S - \mathsf{poly}(\lambda)$*, ciphertext size* $S + \mathsf{poly}(\lambda)$*, public key size* $\mathsf{poly}(\lambda)$ *and secret key size* $\mathsf{poly}(S, \lambda)$.

5.4.2 Proof of Security

We organize our proof of security into a sequence of hybrids.

Sequence of Hybrids

- H_0: The original incompressible encryption security experiment $\mathsf{Dist}^{\mathsf{IncomEnc}}_{\mathcal{A},\Pi}$, where the bit b in the experiment is fixed to be 0.

- H_1: Instead of fixing v and s in step 2 of the security experiment, lazily sample v and s in step 7 where we need to provide sk. Also, instead of sampling v directly, first sample a uniformly random $u \in \{0,1\}^{L_m}$, and then compute $v = u \oplus m_0$.

- H_2: We further modify how we sample v. Now instead of sampling a random u, we sample a random PRG key $k \in \{0,1\}^n$, and set $v = \mathsf{PRG}(k) \oplus m_0$.
- H_3: We once more modify how we sample v. We now sample a long randomness $R \in \{0,1\}^{L_m}$ and use that to compute $v = \mathsf{PRG}(\mathsf{Extract}(R;s)) \oplus m_0$.
- H_4: In step 5, set the ciphertext to be $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (R,1))$.
- H_5: In step 7, revert to computing $v = \mathsf{PRG}(k) \oplus m_0$ for a uniform k.
- H_6: In step 7, revert to computing $v = u \oplus m_0$ for a uniform u.
- H_7: Switch the bit b of the experiment from 0 to 1.
- H_8: In step 7, sample v as $\mathsf{PRG}(k) \oplus m_1$.
- H_9: In step 7, sample v as $\mathsf{PRG}(\mathsf{Extract}(R;s)) \oplus m_1$.
- H_{10}: In step 5, change the ciphertext back to $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (m_1,0))$.
- H_{11}: In step 7, sample v as $\mathsf{PRG}(k) \oplus m_1$.
- H_{12}: In step 7, sample v as $u \oplus m_1$.
- H_{13}: Sample a uniform v back at the beginning of the experiment in step 2. Notice that now we're back at the original incompressible encryption security experiment, where the bit b is fixed to be 1.

Lemma 5.4.1. *No adversary can distinguish between H_0 and H_1 (respectively H_{12} and H_{13}) with non-negligible probability.*

Proof. We prove the case for H_0 and H_1. The case for H_{12} and H_{13} follows analogously. Notice that pk does not depend on sk, and sk is the only value that depends on v and s, but it is not used until in step 7. So we can sample v and s lazily in step 7 instead of fixing it as early as in step 2.

Sampling a uniformly random u and XORing it with m_0 is equivalent to using u as a one-time pad. By the statistical security of OTP, H_0 and H_1 are also statistically indistinguishable. □

Lemma 5.4.2. *If the underlying* PRG *is a secure pseudorandom generator, then no PPT adversary can distinguish between H_1 and H_2 (as well as H_5 and H_6, H_7 and H_8, H_{11} and H_{12}) with non-negligible probability.*

Proof. Here we prove the case for H_1 and H_2. The other three cases follow naturally. In H_1, we have $v = u \oplus m_0$ with uniformly random u, and in H_2, we have $v = \mathsf{PRG}(k) \oplus m_0$ with a uniformly random PRG key k. Since the key k is random and not used anywhere else, by PRG security, the PRG output should be computationally indistinguishable from a uniform distribution. This directly completes the proof. □

Lemma 5.4.3. *If the underlying* $\mathsf{Extract}$ *is a $(L_m, \mathsf{negl}(\lambda))$ average min-entropy extractor, then no adversary can distinguish between H_2 and H_3 (respectively H_{10} and H_{11}) with non-negligible probability.*

Proof. We prove the case for H_2 and H_3. The other case follows.

The randomness R is freshly sampled and not used anywhere else, and hence have full L_m average min-entropy conditioned on the other variables. Therefore, we can easily invoke the extractor security and that gives us $\mathsf{Extract}(R; s)$ is statistically close to a uniform k, and hence also H_2 and H_3. □

Lemma 5.4.4. *If the underlying* FE *is a functional encryption scheme with single-key semi-adaptive game-based security, then no PPT adversary can distinguish between H_3 and H_4 (respectively H_9 and H_{10}) with non-negligible probability.*

Proof. We will prove the case for H_3 and H_4. The other one follows analogously. Notice that the only difference between H_3 and H_4 is the message being encrypted by the underlying FE scheme. In H_3, we use the FE scheme to encrypt $(m, 0)$, while in H_4, we encrypt $(R, 1)$. Notice that

$$f_{v,s}(R, 1) = \mathsf{PRG}(\mathsf{Extract}(R; s)) \oplus \mathsf{PRG}(\mathsf{Extract}(R; s)) \oplus m = m = f_{v,s}(m, 0).$$

So we have two ciphertexts with the same functionality under the function $f_{v,s}$. By the single-key semi-adaptive security of the FE scheme, they should be computationally indistinguishable.

More concretely, assume that there exists an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that distinguishes between H_3 and H_4, we show how to construct an adversary $\mathcal{A}'$ that wins the semi-adaptive security game of the FE scheme. By using $\mathcal{A}$ as a subroutine, $\mathcal{A}'$ works as follows:

- Receive 1^λ and $\mathsf{FE.mpk}$ from the challenger, send 1^λ to $\mathcal{A}_1$, receive 1^S and set $L_m = S + \mathsf{poly}(\lambda)$.

- Send $\mathsf{FE.mpk}$ to $\mathcal{A}_1$ and receive aux and the challenge query m_0 and m_1.
- Sample a uniformly random $R \in \{0,1\}^{L_m}$, and submit the challenge query $\mathsf{FE}.m_0 = (m_0, 0)$ and $\mathsf{FE}.m_1 = (R, 1)$ to the challenger. Receive $\mathsf{FE.ct}$ in response and forward it to $\mathcal{A}_1$. $\mathcal{A}_1$ produces a state st.
- Sample random seed $s \in \{0,1\}^d$, compute $v = \mathsf{PRG}(\mathsf{Extract}(R; s)) \oplus m_0$, and send $f_{v,s}$ to the challenger. Receive in response $\mathsf{FE.sk}_{f_{v,s}}$, and forward it to $\mathcal{A}_2$ together with $(\mathsf{FE.mpk}, \mathsf{aux}, \mathsf{st})$.
- If $\mathcal{A}_2$ outputs that it is in H_3, output 0. Otherwise, output 1.

It should be easy to verify that if $\mathcal{A}$ wins, $\mathcal{A}'$ also wins. □

Lemma 5.4.5. *If the underlying* $\mathsf{Extract}$ *is a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, then no adversary that uses a state of size at most S can distinguish between* H_4 *and* H_5 *(respectively* H_8 *and* H_9*) with non-negligible probability.*

Proof. Here we prove the case for H_4 and H_5. The other case follows analogously.

Here let the random variables $X = R$, and $Y = (\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{aux})$ and $Z = \mathsf{st}$. By Lemma 2.1.1, we have

$$H_\infty(X|Y,Z) \geq \min_y H_\infty(X|Y=y,Z) \geq \min_y H_\infty(X|Y=y) - S = \mathsf{poly}(\lambda).$$

The last equation follows from that $X = R$ is a uniformly random string of length $L_m = S + \mathsf{poly}(\lambda)$. Therefore, by extractor security, no adversary can distinguish $(s, \mathsf{Extract}(R; s), Y, Z)$

from (s, U_n, Y, Z) except with $\mathsf{negl}(\lambda)$ probability. And since we now sample $k \leftarrow U_n$, no adversary can now distinguish between $v = \mathsf{PRG}(\mathsf{Extract}(R; s)) \oplus m_0$ and $v = \mathsf{PRG}(k) \oplus m_0$, i.e. H_4 and H_5. □

Lemma 5.4.6. *No adversary can distinguish between H_6 and H_7 with non-negligible probability.*

Proof. The only difference between H_6 and H_7 is that in H_6 we have $v = u \oplus m_0$ and in H_7 we have $v = u \oplus m_1$, where u is uniformly random. This is just a one time pad encryption with a uniformly sampled key. By OTP security, H_6 and H_7 are statistically indistinguishable. □

Theorem 5.4.2. *If* FE *has single-key semi-adaptive security,* Extract *is a* $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$ *average min-entropy extractor, and* PRG *is a secure PRG, then Construction 5.4.1 has incompressible encryption security.*

Proof. The lemmas above show a sequence of a polynomial number of hybrid experiments where no PPT adversary that produces a state with size at most S can distinguish one from the next with non-negligible probability. The first hybrid H_0 corresponds to the incompressible encryption security game where $b = 0$, and the last one H_{13} corresponds to the case where $b = 1$. The security of the indistinguishability game follows. □

5.5 Incompressible Signatures: Our Basic Construction

5.5.1 Definition

Here we give the definition of *incompressible* signatures. An incompressible signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ takes an additional space parameter S, and in addition to the standard

model signature security (where the adversary has unbounded space throughout the game), we also require *incompressible signature security* that utilizes the following experiment for adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

Signature Forgery Experiment $\mathsf{SigForge}^{\mathsf{IncomSig}}_{\mathcal{A},\Pi}(\lambda)$:

- The adversary $\mathcal{A}_1$, on input 1^λ, outputs a space bound 1^S.
- Run $\mathsf{Gen}(1^\lambda, 1^S)$ to obtain keys $(\mathsf{vk}, \mathsf{sk})$.
- The adversary $\mathcal{A}_1$ is given the public key vk, and submits an auxiliary input aux.
- For $q = \mathsf{poly}(\lambda)$ rounds, $\mathcal{A}_1$ submits a message m, and receives $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$. At the end of the last round, $\mathcal{A}_1$ produces a state st of size at most S.
- The adversary $\mathcal{A}_2$ is given the public key vk, the state st, and the auxiliary input aux, and outputs a signature σ'. If $\mathsf{Ver}(\mathsf{vk}, \sigma')$ outputs $\perp$, output 0. Otherwise, output 1.

Notice that traditionally, we would require $\mathsf{Ver}(\mathsf{vk}, \sigma')$ to be distinct from the messages m's queried before, but here we have no such requirement. Also, notice that allowing the adversary to submit and later receive the auxiliary input aux is equivalent to allowing $\mathcal{A}_1, \mathcal{A}_2$ to just have shared randomness at the beginning of the experiment and that, in the non-uniform setting, the definition would be the same without aux since $\mathcal{A}_1, \mathcal{A}_2$ are deterministic w.l.o.g. With this experiment in mind, we now define the additional security requirement for an incompressible signature scheme.

Definition 5.5.1 (Incompressible Signature Security)**.** *For security parameters λ and S, an incompressible signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ has incompressible signature security, if*

for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*:*

$$\Pr\left[\mathsf{SigForge}^{\mathsf{IncomSig}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

5.5.2 Construction

We present a very simple construction from classical public key signature schemes.

Construction 5.5.1. *Let* λ*,* S *be security parameters. Given* $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *a classical public key signature scheme with message space* $\{0,1\}^{n+L_m}$ *where* $n = S + \mathsf{poly}(\lambda)$ *and rate* ρ'*, we construct an incompressible signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Run* $\mathsf{Sig.Gen}(1^\lambda)$ *to obtain* $(\mathsf{Sig.vk}, \mathsf{Sig.sk})$*. Output* $\mathsf{vk} = \mathsf{Sig.vk}$ *and* $\mathsf{sk} = \mathsf{Sig.sk}$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: Sample randomness* $R \in \{0,1\}^n$*, and output* $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, (R, m))$*.*
- $\mathsf{Ver}(\mathsf{vk}, \sigma)$*: Run* $M \leftarrow \mathsf{Sig.Ver}(\mathsf{Sig.vk}, \sigma)$*. If* $M = \bot$*, output* $\bot$*. Otherwise, if* $M = (R, m)$*, output* m*.*

Sig can be computed in an low-space streaming fashion, since we can hash the message in low space first using Merkle-Damgård. Then Construction 5.6.1 can readily be computed with low space streaming. The rate of this construction is

$$\frac{L_m}{L_\sigma} = \frac{L_m}{(S + L_m)/\rho'} = \rho'(1 + S/L_m)^{-1}.$$

5.5.3 Proof of Security

Theorem 5.5.1. *Assuming the existence of a secure public key signature scheme with rate ρ', there exists an incompressible signature scheme with signature size $\rho'(S + L_m + \mathsf{poly}(\lambda))$, public key size $\mathsf{poly}(\lambda)$ and secret key size $\mathsf{poly}(\lambda)$. Furthermore, it supports streaming computation using $\mathsf{poly}(\lambda)$ bits of memory.*

Proof. We show this through a reduction proof. Concretely, we show how one can use an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks the incompressible signature security as a subroutine to build an adversary $\mathcal{A}'$ the breaks the underlying classical Sig scheme. The adversary $\mathcal{A}'$ works as follows:

- Send 1^λ to $\mathcal{A}_1$, receive 1^S, and set $n = S + \mathsf{poly}(\lambda)$.
- Receive vk from the challenger, forward it to $\mathcal{A}_1$, and receive aux from $\mathcal{A}_1$.
- For each signing query m_i made by $\mathcal{A}_1$, sample a random $R_i \in \{0,1\}^n$ and make a query (R_i, m_i) to the challenger. Receive back σ_i and forward it directly to $\mathcal{A}_1$.
- When $\mathcal{A}_1$ produces a state st, send vk, st and aux to $\mathcal{A}_2$. Output what $\mathcal{A}_2$ outputs as σ'.

Notice that if $\mathcal{A}$ wins, that means $\mathsf{Ver}(\mathsf{vk}, \sigma') = (R', m') \neq \bot$. If $m' \notin \{m_i\}_i$, then (R', m') is a pair not queried before by $\mathcal{A}'$, and thus $\mathcal{A}'$ wins the game. If $m' = m_j$ for some j, then we argue that with overwhelming probability $R' \neq R_j$, and hence $\mathcal{A}'$ wins as well. Indeed this is true since

$$H_\infty(R_j | \mathsf{st}, \mathsf{vk}, \{m_i\}_i) \geq S + \mathsf{poly}(\lambda) - S = \mathsf{poly}(\lambda).$$

Therefore R_j is unpredictable conditioned on $\mathcal{A}_2$'s view, so the probability of $\mathcal{A}_2$ producing some $R' = R_j$ is negligible. □

5.6 Rate-1 Incompressible Signatures

5.6.1 Incompressible Encoding

Moran and Wichs[79] give the definition for incompressible encodings and show construction based on either the Decisional Composite Residuosity (DCR) or Learning With Errors (LWE) assumptions, in either the random oracle model or the CRS model. We modify the definition slightly to better accommodate the syntax in this chapter.

Definition 5.6.1 (Incompressible Encodings[79])**.** *Let λ be security parameters. An incompressible encoding scheme for message space $\{0,1\}^{L_m}$ and codeword space $\{0,1\}^{L_c}$ is a pair of PPT algorithms $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ that utilizes the following syntax:*

- $\mathsf{Enc}(1^\lambda, m) \to c$ *on input the security parameter and a message, outputs a codeword c.*
- $\mathsf{Dec}(c) \to m$ *on input a codeword, outputs the decoded message m.*

The "rate" of the incompressible encoding is L_m/L_c.*

We additionally require correctness and S-incompressibility†:

Definition 5.6.2 (Correctness)**.** *For all $\lambda \in \mathbb{N}$ and $m \in \mathcal{M}$,* $\Pr[\mathsf{Dec}(\mathsf{Enc}(1^\lambda, m)) = m] \geq 1 - \mathsf{negl}(\lambda)$.

Next, consider the following experiment for adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

Codeword Compression Experiment $\mathsf{Comp}^{\mathsf{IncomCode}}_{\mathcal{A},\mathsf{Code}}(\lambda, S)$:

*This is equivalent to the α-expansion property as defined in [79] for $\alpha = L_c/L_m$.
†This is equivalent to β-incompressibility as defined in [79] for $\beta = S$.

- On input 1^λ, the adversary $\mathcal{A}_1$ submits a message m and auxiliary input aux. It receives $c \leftarrow \mathsf{Enc}(1^\lambda, m)$, and produces a state st of size at most S.
- The adversary $\mathcal{A}_2$ is given the state st, the message m, and the auxiliary information aux; it produces a codeword c'. Output 1 if and only if $c' = c$.

Definition 5.6.3 (S-Incompressibility)**.** *For security parameter λ, we require that for all PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:*

$$\Pr\left[\mathsf{Comp}^{\mathsf{IncomCode}}_{\mathcal{A},\mathsf{Code}}(\lambda, S) = 1\right] \leq \mathsf{negl}(\lambda).$$

5.6.2 Construction

Now we show how we modify Construction 5.5.1 to get an incompressible signature scheme with a rate of 1. Essentially we can think of the procedure of attaching a long random string in Construction 5.5.1 as a form of an incompressible encoding with a poor rate. Here we just need to replace it with an incompressible encoding with a rate of 1.

Construction 5.6.1. *Let λ, S be security parameters. Given $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ a classical signature scheme with rate 1, and $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ an incompressible encoding scheme with rate 1 and S-incompressibility, we construct an incompressible signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Run* $\mathsf{Sig.Gen}(1^\lambda)$ *to obtain* $(\mathsf{Sig.vk}, \mathsf{Sig.sk})$*. Output* $\mathsf{vk} = \mathsf{Sig.vk}$ *and* $\mathsf{sk} = \mathsf{Sig.sk}$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: First compute the codeword* $c \leftarrow \mathsf{Code.Enc}(1^\lambda, m)$*, and then compute* $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, c)$*.*

- $\mathsf{Ver}(\mathsf{vk}, \sigma)$: *Run* $c \leftarrow \mathsf{Sig.Ver}(\mathsf{Sig.vk}, \sigma)$. *If* $c = \bot$, *output* $\bot$. *Otherwise, output* $m \leftarrow \mathsf{Code.Dec}(c)$.

The rate of our scheme is the product of the rates of the incompressible encoding and standard-model signature scheme. We can construct a classical signature scheme with rate $1 - o(1)$ from any one-way function by hashing the message using a universal one-way hash function, and then signing the hash value. Our incompressible signatures therefore have rate $1 - o(1)$, in the CRS or random oracle model.

Theorem 5.6.1. *Assuming the existence of a secure public key signature scheme with rate* $1 - o(1)$ *and an incompressible encoding scheme [in the CRS/RO model] with rate* $1 - o(1)$*, there exists an incompressible signature scheme [in the CRS/RO model] with rate* $1 - o(1)$*, public key size* $\mathsf{poly}(\lambda)$ *and secret key size* $\mathsf{poly}(\lambda)$*. Furthermore, it supports streaming computation using* $\mathsf{poly}(\lambda)$ *bits of memory, assuming that the incompressible encoding scheme does as well [either in the random oracle model, or with the streaming of the CRS in the CRS model]. If the incompressible encoding scheme in the CRS model is only selectively secure, then so is the resulting incompressible signature scheme.*

Proof. Assume towards contradiction that there exists an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that wins the incompressible signature game. Let $\{m_i\}_i$ be the message queries made by $\mathcal{A}_1$, $\{\sigma_i\}_i$ the responses, and $\{c_i = \mathsf{Sig.Ver}(\mathsf{vk}, \sigma_i)\}_i$. Let σ' be $\mathcal{A}_2$'s forgery, and $c' = \mathsf{Sig.Ver}(\mathsf{vk}, \sigma')$.

Let p be the probability that $\mathcal{A}$ wins and $c' \notin \{c_i\}_i$. The security of the standard-model signature scheme immediately implies that p is negligible: simply devise a new adversary $\mathcal{A}'$ that is the same as $\mathcal{A}$, except that it encodes every message m_i into $c_i \leftarrow \mathsf{Code.Enc}(1^\lambda, m_i)$ before making a signing query.

Let r be the probability that $\mathcal{A}$ wins and $c' \in \{c_i\}$. The security of the incompressible encoding implies that r is negligible: we construct a new adversary $\mathcal{A}''$ which sets up the standard-model signature for itself and simulates the entire view of $\mathcal{A}$. The exception is that it guesses a random i^*, and forwards the m_{i^*} as it's challenge message; when it receives c_{i^*} from the encoding challenge, it computes $\sigma_{i^*} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, c_{i^*})$. With probability r/q, this adversary is able to reproduce c_{i^*}, despite compressing it. Here, q is the number of queries made.

We therefore have that $\mathcal{A}$ wins with probability $p + r$, which is negligible. □

5.6.3 Equivalence to Incompressible Encoding

Lastly, we quickly show that incompressible signatures are equivalent to incompressible encodings (plus one-way functions) by showing how to construct an incompressible encoding scheme from an incompressible signature scheme.

Construction 5.6.2. *Let λ be a security parameter. Given $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ an incompressible signature scheme with rate 1 and small verification keys, we construct an incompressible encoding scheme $\Pi = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ as follows:*

- $\mathsf{Enc}(1^\lambda, m)$*: Sample* $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda, 1^S)$*, and then compute* $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, m)$*. Output* $c = (\mathsf{Sig.vk}, \sigma)$.
- $\mathsf{Dec}(c = (\mathsf{Sig.vk}, \sigma))$*: Simply output* $m \leftarrow \mathsf{Sig.Ver}(\mathsf{Sig.vk}, \sigma)$.

The codeword length is the signature length (equal to message length if Sig has rate 1) plus the length of the verification length. Hence the rate is 1 if the verification keys are short.

Correctness follows directly from the correctness of the signature scheme. Security also follows directly: if an adversary using a state st of size at most S is able to produce $c' = c$, then it has also produced a valid signature σ and hence wins the incompressible signature security game. Therefore, by Construction 5.6.1 and 5.6.2, incompressible signatures and incompressible encodings (plus one-way functions) are equivalent.

5.7 Constructing Rate-1 Functional Encryption

Here, we build rate-1 functional encryption (FE). For our application, we only need one key security. However, our construction satisfies many-key security, though we need indistingishability obfuscation (iO). We leave it as an open question whether such high-rate *single key* FE can be built from standard tools.

Our construction is based on the techniques of Boneh and Zhandry[21], who build from iO something called private linear broadcast encryption, which is a special case of general FE. A number of issues arise in generalizing their construction to general functions, which we demonstrate how to handle.

5.7.1 Building Blocks

Definition 5.7.1 (Indistinguishability Obfuscation[9])**.** *An* indistinguiability obfuscator $i\mathcal{O}$ *for a circuit class* $\{\mathcal{C}_\lambda\}$ *is a PPT uniform algorithm satisfying the following conditions:*

- **Functionality***: For any* $C \in \mathcal{C}_\lambda$*, then with probability 1 over the choice of* $C' \leftarrow i\mathcal{O}(1^\lambda, C)$*,* $C'(x) = C(x)$ *for all inputs* x*.*

- **Security**: *For all pairs of PPT adversaries* (S, D)*, if there exists a negligible function* α *such that*

$$\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow S(\lambda)] > 1 - \alpha(\lambda)$$

then there exists a negligible function β *such that*

$$\left| \Pr[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1] - \Pr[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1] \right| < \beta(\lambda)$$

When $\mathcal{C}_\lambda$ is the class of all polynomial-size circuits, we simply call $i\mathcal{O}$ an indistinguishability obfuscator. There are several known ways to construct indistinguishability obfuscation:

- Garg et al.[47] build the first candidate obfuscation from cryptographic multilinear maps.
- Provably from novel strong circularity assumptions[24,51,95]
- Provably from "standard" assumptions[71]: (sub-exponentially secure) LWE, LPN over fields, bilinear maps, and constant-locality PRGs

Definition 5.7.2 (Puncturable PRF[20,73,23])**.** *A* puncturable PRF *with domain* $\mathcal{X}_\lambda$ *and range* $\mathcal{Y}_\lambda$ *is a pair* $(\mathsf{Gen}, \mathsf{Punc})$ *where:*

- $\mathsf{Gen}(1^\lambda)$ *outputs an efficiently computable function* $\mathsf{PRF} : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$
- $\mathsf{Punc}(\mathsf{PRF}, x)$ *takes as input a function* PRF *and an input* $x \in \mathcal{X}_\lambda$*, and outputs a "punctured" function* $\mathsf{PRF}^{\overline{x}}$*.*

- **Correctness**: *With probability 1 over the choice of* $\mathsf{PRF} \leftarrow \mathsf{Gen}(1^\lambda)$,

$$\mathsf{PRF}^{\bar{x}}(x') = \begin{cases} \mathsf{PRF}(x') & \text{if } x' \neq x \\ \bot & \text{if } x' = x \end{cases}$$

- **Security**: *For all* $x \in \mathcal{X}_\lambda$, $(\mathsf{PRF}^{\bar{x}}, \mathsf{PRF}(x))$ *is computationally indistinguishable from* $(\mathsf{PRF}^{\bar{x}}, y)$, *where* $\mathsf{PRF} \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $y \leftarrow \mathcal{Y}_\lambda$.

Such puncturable PRFs can be built from any one-way function [54].

We now give a new definition of a type of signature scheme with a *single-point binding* (SPB) property. This allows, given a message m, for generating a fake verification key together with a signature on m. The fake verification key and signature should be indistinguishable from the honest case. Yet there are no signatures on messages other than m relative to the fake verification key. [21] implicitly constructs such signatures from iO and one-way functions, but with a logarithmic message space, which was good enough for their special-purpose FE scheme. In our case, we need to handle very large exponential message spaces. The problem with [21]'s approach is that the security loss is proportional to the message space; to compensate requires assuming (sub)exponential hardness, and also setting the security parameter to be larger than the message length. This results in the signature size being polynomial in the message size, resulting in a low-rate FE scheme. SPB signatures avoid the exponential loss, so we can keep the security parameter small, resulting in a rate-1 FE scheme.

Definition 5.7.3. *A* single-point binding (SPB) signature *is a quadruple of algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{GenBind})$ *where* $\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}$ *satisfy the usual properties of a signature scheme. Additionally, we have the following:*

- $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$ *takes as input a message* m*, and produces a verification key* vk *and signature* σ*.*

- *For any messages* m *and with overwhelming probability over the choice of* $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$*,* $\mathsf{Ver}(\mathsf{vk}, \sigma') \in \{m, \perp\}$ *for any* σ'*. That is, there is no message* $m' \neq m$ *such that there is a valid signature of* m' *relative to* vk*.*

- *For any* m*,* $\mathsf{GenBind}(1^\lambda, m)$ *and* $(\mathsf{vk}, \mathsf{Sign}(\mathsf{sk}, m))$ *are indistinguishable, where* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*. Note that this property implies that* $\mathsf{Ver}(\mathsf{vk}, \sigma)$ *accepts and output* m*, when* $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$*.*

We explain how to construct SPB signatures in Section 5.7.3, either from leveled FHE (and hence LWE), or from iO and one-way functions.

Our Rate-1 FE Scheme. We now give our rate-1 FE scheme:

Construction 5.7.1. *Let* $i\mathcal{O}$ *be an indistinguishability obfuscator,* Gen *be a PRF,* $(\mathsf{Gen}', \mathsf{Sig}, \mathsf{Ver})$ *a signature scheme, and* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}, \mathsf{PRG}' : \{0,1\}^\lambda \to \{0,1\}^{L_m}$ *be a PRG.*

- $\mathsf{Setup}(1^\lambda)$*: Sample* $\mathsf{PRF} \leftarrow \mathsf{Gen}(1^\lambda)$*. Set* $\mathsf{msk} = \mathsf{PRF}$ *and* $\mathsf{mpk} = i\mathcal{O}(1^\lambda, P_{\mathsf{Enc}})$*, where* P_{Enc} *is the program given in Figure 5.1.*

- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: output* $\mathsf{sk}_f \leftarrow i\mathcal{O}(1^\lambda, P_{\mathsf{Dec},f})$*, where* $P_{\mathsf{Dec},f}$ *is the program given in Figure 5.2.*

- $\mathsf{Enc}(\mathsf{mpk}, m)$*: Choose a random* r*, and evaluate* $(t, v) \leftarrow \mathsf{mpk}(r)$*. Then parse* $v = (w, u)$*. Set* $c = \mathsf{PRG}'(w) \oplus m$*. Next run* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}'(1^\lambda; u)$*, using* u *as the random coins for* Gen'*. Compute* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, c)$*. Output* (t, σ)*.*

- $\mathsf{Dec}(sk_f,\ (t,\sigma)\ ) = \mathsf{sk}_f(t,\sigma)$

Figure 5.1: The program P_{Enc}.

Inputs: r
Constants: PRF

1. $t \leftarrow \mathsf{PRG}(r)$.
2. $v \leftarrow \mathsf{PRF}(t)$.
3. Output (t, v).

Figure 5.2: The program $P_{\mathsf{Dec},f}$.

Inputs: t, σ
Constants: PRF

1. $(w, u) \leftarrow \mathsf{PRF}(t)$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}'(1^\lambda; u)$.
3. $c \leftarrow \mathsf{Ver}(\mathsf{vk}, \sigma)$. If $c = \perp$, abort and output $\perp$.
4. Output $f(\mathsf{PRG}'(w) \oplus c)$.

Correctness follows immediately from the correctness of the various components. Notice that the ciphertext size is $L_m + \mathsf{poly}(\lambda)$, provided the signature scheme outputs short signatures. Therefore, construction 5.7.1 has rate $1 - o(1)$.

Provided the random coins for $(\mathsf{Gen}', \mathsf{Sign}, \mathsf{Ver})$ are independent of the message length, P_{Enc} has size $\mathsf{poly}(\lambda)$, independent of the message length. If $\mathsf{Gen}', \mathsf{Sign}$ can be evaluated in a low-space streaming fashion, then so can Enc.

5.7.2 Proof of Security

Sequence of Hybrids

- H_0: This is the FE security experiment, where the bit b in the experiment is fixed to be 0. Note that in this hybrid, the challenge ciphertext is generated as (t^*, σ^*), where $r^* \leftarrow \{0,1\}^\lambda, t^* \leftarrow \mathsf{PRG}(r^*), (w^*, u^*) \leftarrow \mathsf{PRF}(t^*), x^* \leftarrow \mathsf{PRG}'(w^*), c^* \leftarrow x^* \oplus m_0$, $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}'(1^\lambda; u^*)$, and $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{sk}^*, c^*)$.

- H_1: This is identical to H_0, except that we now generate t^* uniformly at random: $t^* \leftarrow \{0,1\}^{2\lambda}$.

- H_2: This is the same as H_1, except that we change the way we generate $\mathsf{mpk}, \mathsf{sk}_f$. First compute $\mathsf{PRF}^{\overline{t^*}} \leftarrow \mathsf{Punc}(\mathsf{PRF}, t^*)$, $(w^*, u^*) \leftarrow \mathsf{PRF}(t^*)$. Then let $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}'(1^\lambda; u^*)$ and $x^* = \mathsf{PRG}(w^*)$. We now compute $\mathsf{mpk} \leftarrow i\mathcal{O}(1^\lambda, P^{\mathtt{punc}}_{\mathsf{Enc}})$ and answer secret key queries with $\mathsf{sk}_f \leftarrow i\mathcal{O}(1^\lambda, P^{\mathtt{punc}}_{\mathsf{Dec}})$. Here, $P^{\mathtt{punc}}_{\mathsf{Enc}}$ and $P^{\mathtt{punc}}_{\mathsf{Dec},f}$ are the programs in Figures 5.3 and 5.4

- H_3: This is identical to H_2, except that now we generate w^*, u^* uniformly at random, instead of $(w^*, u^*) \leftarrow \mathsf{PRF}(t^*)$.

- H_4: This is identical to H_3 except that we now generate x^* uniformly at random instead of $x^* \leftarrow \mathsf{PRG}(w^*)$.

- H_5: This is identical to H_4, except for the following changes:

  - We generate c^* uniformly at random at the beginning of the experiment.
  - After the challenge query, we generate $x^* = c^* \oplus m_0$. Note that x^* is the only place m_0 enters the experiment.

- H_6: This is identical to H_5, except now we generate $(\mathsf{vk}^*, \sigma^*) \leftarrow \mathsf{GenBind}(1^\lambda, c^*)$.

- H_7 through H_{13}: Hybrid H_{7+i} is identical to H_{6-i}, except that m_0 is replaced with m_1. Thus H_{13} is the FE security experiment where b is fixed to be 1.

Inputs: $m; r$

Constants: $\mathsf{PRF}^{\overline{t^*}}$, t^*

1. $t \leftarrow \mathsf{PRG}(r)$. If $t = t^*$, immediately abort and output $\perp$.
2. $v \leftarrow \mathsf{PRF}^{\overline{t^*}}(t)$.
3. Output (t, v).

Figure 5.3: The program $P^{\mathtt{punc}}_{\mathsf{Enc}}$. Differences from P_{Enc} highlighted in yellow.

Inputs: t, σ

Constants: $\mathsf{PRF}_1^{\overline{t^*}}, \mathsf{PRF}_2^{\overline{t^*}}$, t^*, x^*, vk^*

1. If $t \neq t^*$, skip to Step 2. If $t = t^*$, run $c \leftarrow \mathsf{Ver}(\mathsf{vk}^*, \sigma)$; if $c = \perp$, abort and output $\perp$, otherwise abort and output $f(x^* \oplus c)$.
2. $(w, u) \leftarrow \mathsf{PRF}^{\overline{t^*}}(t)$
3. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}'(1^\lambda; u)$.
4. $c \leftarrow \mathsf{Ver}(\mathsf{vk}, c, \sigma)$. If $c = \perp$, abort and output $\perp$.
5. Output $f(\mathsf{PRG}(w) \oplus c)$.

Figure 5.4: The program $P^{\mathtt{punc}}_{\mathsf{Dec},f}$. Differences from $P_{\mathsf{Enc},f}$ highlighted in yellow.

Proofs of Hybrid Steps

Lemma 5.7.1. *If* PRG *is a secure PRG, then no PPT adversary can distinguish between H_0 and H_1 (respectively H_{12} and H_{13}) except with negligible probability.*

Proof. The only difference between the hybrids is how we generate t^*; in H_0 it is pseudorandom and in H_1 it is uniformly random. Thus indistinguishability follows immediately from the security of PRG. □

Lemma 5.7.2. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, then no PPT adversary can distinguish between H_1 and H_2 (respectively H_{11} and H_{12}) except with negligible probability.*

Proof. Note that, with overwhelming probability, the uniformly random t^* is not in the sparse image of PRG. Thus, with overwhelming probability, the abort step in $P^{\mathtt{punc}}_{\mathsf{Enc}}$ is never triggered. On all $t \neq t^*$, PRF and $\mathsf{PRF}^{\overline{t^*}}$ behave identically. Thus, P_{Enc} and $P^{\mathtt{punc}}_{\mathsf{Enc}}$ have identical functionalities. Thus their obfuscations are indistinguishable. Likewise, $P_{\mathsf{Dec},f}$, on input (t^*, c, σ), would compute $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}'(1^\lambda; u^*)$, which would exactly output $(\mathsf{vk}^*, \mathsf{sk}^*)$. Provided the signature accepted, it would output $f(m)$ where $m = \mathsf{PRG}(w^*) \oplus c$. Thus $P_{\mathsf{Dec},f}$ and $P_{\mathsf{Dec},f}$ behave identically on all inputs of this form. On inputs (t, c, σ) with $t \neq t^*$, the programs trivially behave identically. Thus they are identical on all inputs, and their obfuscations are indistinguishable. □

Lemma 5.7.3. *If $(\mathsf{Gen}', \mathsf{Punc})$ is a secure puncturable PRF, then no PPT adversary can distinguish between H_3 and H_4 (respectively H_{10} and H_{11}) except with negligible probability.*

Proof. The only difference between these hybrids is that w^*, u^* switch from being outputs of $\mathsf{PRF}(t^*)$ to being uniformly random. But since the rest of the experiment can be simulated using only $\mathsf{PRF}^{\overline{t^*}}$, security follows immediately from punctured PRF security. □

Lemma 5.7.4. *If PRG' is a secure PRG, then no PPT adversary can distinguish H_3 and H_4 (respectively H_9 and H_{10}) except with negligible probability.*

Proof. The only difference between the hybrids is that we switch from x^* being pseudorandomly generated from a random w^* to x^* being uniform. Indistinguishability follows immediately from the security of PRG'. □

Lemma 5.7.5. *H_4 and H_5 (respectively H_8 and H_9) are identically distributed.*

Proof. Since x^* is uniform, so is $c^* = x^* \oplus m_b$. In both hybrids, we choose c^* or x^* randomly, and solve for the other. Thus the distributions are identical. □

Lemma 5.7.6. *If* $(\mathsf{Gen}', \mathsf{Sign}, \mathsf{Ver}, \mathsf{GenBind})$ *is a secure SPB signature scheme, then no PPT adversary can distinguish between* H_5 *and* H_6 *(respectively* H_7 *and* H_8*) except with negligible probability.*

Proof. Note that neither hybrid requires sk^*, and the only difference is how we generate vk^*, σ^*: in H_5 (respectively H_8) vk^* is generated from Gen' using fresh random coins u^* and σ^* is the signature on c^*, whereas in H_6 (respectively H_7), $(\mathsf{vk}^*, \sigma^*)$ is generated as $\mathsf{GenBind}(1^\lambda, c^*)$. Indistinguishability follows immediately from the security of the signature scheme. □

Lemma 5.7.7. *If iO is a secure indistinguishability obfuscator, then no PPT adversary can distinguish between* H_6 *and* H_7*.*

Proof. The only difference between the two hybrids is whether $x^* = c^* \oplus m_0$ or $x^* = c^* \oplus m_1$, and the only place x^* enters the experiment is in $P^{punc}_{\mathsf{Dec},f}$. Moreover, x^* only affects the output $f(x^* \oplus c)$, and only in the event that the input (t, c, σ) satisfies $t = t^*$ and $\mathsf{Ver}(\mathsf{vk}^*, c, \sigma)$ accepts.

By the single-point binding of vk^*, all inputs (t^*, c, σ) reject, except for $c = c^*$. But in the case $c = c^*$, we have that $f(x^* \oplus c) = f(m_b)$. The FE security experiment guarantees that $f(m_0) = f(m_1)$. Thus the programs $P^{punc}_{\mathsf{Dec},f}$ have identical functionality in both hybrids, and so their obfuscations are indistinguishable. □

Theorem 5.7.1. *If iO is a secure indistinguishability obfuscator,* $\mathsf{PRG}, \mathsf{PRG}'$ *are secure PRGs,* $(\mathsf{Gen}', \mathsf{Sign}, \mathsf{Ver}, \mathsf{GenBind})$ *is a secure SPB signature, and* $(\mathsf{Gen}, \mathsf{Punc})$ *is a secure puncturable pseudorandom function, then Construction 5.7.1 is a secure functional encryption scheme.*

5.7.3 Constructing SPB Signatures

We now show how to construct single-point binding signatures.

A low-rate construction. We first describe a simple low-rate construction. This construction is not good enough for our purposes, as our FE scheme inherits the rate of the signature scheme. But we will later show how to compile any low-rate construction into a high-rate construction.

Our construction is just Lamport one-time signatures, where the underling one-way function is replaced with a PRG:

Construction 5.7.2. *Let* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a PRG. Then for a desired message length n, our construction works as follows:*

- $\mathsf{Gen}(1^\lambda)$*: for* $i \in \{1,\ldots,n\}, b \in \{0,1\}$*, sample* $\mathsf{sk}_{i,b} \leftarrow \{0,1\}^\lambda$ *and set* $\mathsf{vk}_{i,b} = \mathsf{PRG}(\mathsf{sk}_{i,b})$*. Output* $(\mathsf{vk} = (\mathsf{vk}_{i,b})_{i,b}\ ,\ \mathsf{sk} = (\mathsf{sk}_{i,b})_{i,b})$.

- $\mathsf{Sign}(\mathsf{sk}, m)$*: output* $\sigma = (m,\ (\mathsf{sk}_{i,m_i})_i\ )$

- $\mathsf{Ver}(\mathsf{vk}, \sigma)$*: Extract m from* σ*. For each* $i \in \{1,\ldots,n\}$*, check that* $\mathsf{PRG}(\mathsf{sk}_{i,m_i}) = \mathsf{pk}_{i,m_i}$*. If all checks pass, output m. Otherwise output* $\perp$.

- $\mathsf{GenBind}(1^\lambda, m)$*: for each* $i \in \{1,\ldots,n\}$*, sample* $\mathsf{sk}_{i,m_i} \leftarrow \{0,1\}^\lambda$ *and set* $\mathsf{vk}_{i,m_i} = \mathsf{PRG}(\mathsf{sk}_{i,m_i})$*. Then sample* $\mathsf{vk}_{i,1-m_i} \leftarrow \{0,1\}^{2\lambda}$ *uniformly. Output* $(\mathsf{vk} = (\mathsf{vk}_{i,b})_{i,b}\ ,\ \sigma = (m,\ (\mathsf{sk}_{i,m_i})_i)\ )$.

In other words, to bind to a message, simply replace all the public key components that do not correspond to the message with uniform randomness.

Binding follows from the fact that, with overwhelming probability $\mathsf{vk}_{i,1-m_i}$ in binding mode will have no pre-images. Since any message other than m must differ from m on *some* bit i, such messages will not have any signatures. Security follows immediately from the pseudorandomness of PRG.

The problem with this signature scheme is that its rate is poor: the signature on a message is a multiplicative $\mathsf{poly}(\lambda)$ factor larger than the message itself

FROM LOW-RATE TO HIGH-RATE USING SPB HASHES. We now describe a new object, related to somewhere statistically binding (SSB) hashing[68], which we call single-point binding (SPB) hashing.

Definition 5.7.4. *A* single-point binding (SPB) hash function *is a triple of algorithms* $(\mathsf{Gen}, H, \mathsf{GenBind})$ *where:*

- $\mathsf{Gen}(1^\lambda)$ *produces a hashing key* hk.
- $H(\mathsf{hk}, m)$ *deterministically produces a hash* h*, with* $|h| \ll |m|$.
- $\mathsf{GenBind}(1^\lambda, m^*)$ *takes as input a message* m^**, and produces a hashing key* hk *with the property that, with overwhelming probability over the choice of* $\mathsf{hk} \leftarrow \mathsf{GenBind}(1^\lambda, m^*)$*, for any* $m \neq m^*$, $H(\mathsf{hk}, m) \neq H(\mathsf{hk}, m^*)$.
- *For any message* m^*, $(m^*, \mathsf{Gen}(1^\lambda))$ *is computationally indistinguishable from* $(m^*, \mathsf{GenBind}(1^\lambda, m^*))$.

We now use a SPB hash to improve the rate of an SPB signature. The construction is the usual hash-and-sign signature scheme: to sign a message m, simply compute the signature

$\sigma \leftarrow H(\mathsf{hk}, m)$, and output (m, σ). If the underlying signature is an SPB signature, then $\mathsf{GenBind}$ for the new signature simply binds hk to m, and then binds vk to $H(\mathsf{hk}, m)$.

If H hashes to a size that is independent of the message, then the resulting signature has rate 1, regardless of the rate of the original signature.

Constructing SPB Hashing. It remains to construct an SPB hash function.

We first briefly note that such hash functions can be build from fully homomorphic encryption (FHE), following essentially the same construction of somewhere statistically binding hashing from[68]. The hashing key is normally the encryption of a random string r of length equal to the message. To hash a message m, homomorphically compute an encryption of b, the result of comparing m with r. To bind the hashing key to m, simply encrypt the message m. FHE security immediately implies security. For binding, the message m will then hash to an encryption of 1, whereas any other message will hash to an encryption of 0. By the correctness of the FHE scheme, encryptions of 0 and 1 must be disjoint.

Next, we explain how to get a construction from iO and one-way functions. Since we are already using iO and one-way functions to build our FE scheme, these assumptions are redundant.

Construction 5.7.3. *Let $i\mathcal{O}$ be an indistinguishability obfuscator,* Gen' *the generation algorithm for a PRF, and* PRG *a pseudorandom generator.*

- $\mathsf{Gen}(1^\lambda)$*: Sample* $\mathsf{PRF} \leftarrow \mathsf{Gen}'(1^\lambda)$*, and output* $\mathsf{hk} = i\mathcal{O}(1^\lambda, P_{\mathtt{hash}})$*, where* $P_{\mathtt{hash}}$ *is the program given in Figure 5.5.*

- $H(\mathsf{hk}, m) = \mathsf{hk}(m)$

- $\mathsf{GenBind}(1^\lambda, m^*)$: *Sample* $\mathsf{PRF} \leftarrow \mathsf{Gen}'(1^\lambda)$, *compute* $\mathsf{PRF}^{\overline{m^*}} \leftarrow \mathsf{Punc}(\mathsf{PRF}, m^*)$, *and choose a random string* x^*. *Output* $\mathsf{hk} = i\mathcal{O}(1^\lambda, P^{\mathtt{bind}}_{tthash})$, *where* $P^{\mathtt{bind}}_{tthash}$ *is the program given in Figure 5.6.*

Remark 5.7.1. *If we want our rate-1 incompressible encryption to have encryption be computable in low space given a stream of m, then we need our rate-1 FE encryption to be likewise be computable in low space given a message stream. This in turn means we need to be able to evaluate* $H(\mathsf{hk}, m)$ *in low space given the message stream, and given the random coins used to construct* hk. *We can always assume the random coins are small. In our construction, we cannot compute* hk *itself in low space, since it is a large obfuscated program. However, we can nevertheless compute* $H(\mathsf{hk}, m) = \mathsf{PRG}(\mathsf{PRF}(m))$ *in low-space, provided* PRF *has small keys and can be evalauted on a message stream in low space (the output of* PRF *is small, so* PRG *can easily be computed once we have* $\mathsf{PRF}(m)$. *Most PRFs have this property, including the puncturable PRF from one-way functions due to*[53]. *This gives us the desired rate-1 incompressible encryption with low-space encryption.*

Inputs: m
Constants: PRF

1. Output $\mathsf{PRG}(\mathsf{PRF}(m))$

Figure 5.5: The program $P_{\mathtt{hash}}$.

For binding, note that the random x^* outputted on input m^* is, with overwhelming probability, outside the range of PRG. But all inputs $m \neq m^*$ must output points in the range of PRG. Thus, there are no collisions with m^*.

For security, use the following sequence of hybrids:

Inputs: m

Constants: $\mathsf{PRF}^{\overline{m^*}}$, m^*, x^*

1. If $m = m^*$, output x^*. Otherwise,
2. Output $\mathsf{PRG}(\mathsf{PRF}(m))$

Figure 5.6: The program $P^{\mathtt{bind}}_{\mathtt{hash}}$. Differences from $P_{\mathtt{hash}}$ are highlighted in yellow.

- H_0: this is the case where $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda)$.
- H_1: here, we generate $\mathsf{hk} = i\mathcal{O}(P^{\mathtt{bind}}_{\mathtt{hash}})$, except that x^* is set to $\mathsf{PRG}(\mathsf{PRF}(m^*))$. Note that this x^* is exactly the output of $P_{\mathtt{hash}}(m^*)$. Hence in this case $P^{\mathtt{bind}}_{\mathtt{hash}}$ and $P_{\mathtt{hash}}$ have identical functionalities. Indistinguishability follows from iO.
- H_2: here we generate $x^* = \mathsf{PRG}(s^*)$ for a uniform random value s^*. The only difference from H_1 is that we replace $\mathsf{PRF}(m^*)$ with s^*. But since only the punctured PRF $\mathsf{PRF}^{\overline{m^*}}$ is needed, this change follows from punctured PRF security.
- H_3: here, we generate $\mathsf{hk} \leftarrow \mathsf{GenBind}(1^\lambda, m^*)$. The only difference from H_2 is that we replace $x^* = \mathsf{PRG}(s^*)$ with a uniformly random x^*. Since s^* is uniformly random, this follows immediately from the pseudorandomness of PRG.

6

Multi-User Incompressible Encryption

6.1 Introduction

Bounded-Storage Mass Surveillance. We consider a scenario where a powerful (e.g., state-level) adversary wants to perform mass surveillance of the population. Even if the population uses encryption to secure all communication, the adversary can collect large amounts of encrypted data from the users (e.g., by monitoring encrypted traffic on the Internet). The data is encrypted and hence the adversary does not learn anything about its contents when it is collected. However, the adversary may store this data for the future. Later, it may identify various "persons of interest" and perform expensive targeted attacks to get their secret keys (e.g., by remote hacking or by physically compromising their devices). We will assume the adversary is capable of eventually getting any secret key of any user of its choosing. Can we still achieve any meaningful notion of security against such mass-surveillance?

One option is to rely on cryptosystems having *forward secrecy*[64], which exactly addresses the problem of maintaining security even if the secret key is later compromised. Unfortunately, forward-secure encryption schemes inherently require either multi-round interaction between the sender and receiver or for the receiver to perform key updates, both of which can be impractical or impossible in many natural scenarios. Without these, it may seem that no reasonable security is possible – if the adversary collects all the ciphertexts and later can get any secret key, clearly it can also get any plaintext!

In this chapter, we restrict the adversary to have *bounded storage*, which is much smaller than the total of size of all the encrypted data it can observe. This is a reasonable assumption since the total communication of an entire population is likely huge.* As a running example

*Global annual Internet traffic has long surpassed 1 zettabyte (10^{21} bytes)[11], while *total* world-wide data-center storage is only a couple zettabytes in 2022[36].

throughout the introduction, we will assume that the adversary's storage capacity is 1% of the total encrypted data size. We allow the adversary to observe all the encrypted data simultaneously and then compress it in some arbitrary way to fit within its storage budget. Later, the adversary can get any secret key of any user of its choosing, and eventually it may even get all the keys of all the users. What kind of security guarantees can we provide in this setting?

Clearly, the adversary can simply store 1% of the ciphertexts and discard the remaining 99%, which will allow it to later compromise the security of 1% of the users by getting their secret keys. While one may pessimistically see this as a significant privacy violation already, we optimistically regard this as a potentially reasonable privacy outcome that's vastly preferable to the adversary being able to compromise all the users. For example, if the adversary later chooses a random user and wants to learn something about their data, it will only be able to do so with 1% probability, even if it can get their secret key. But can we argue that this is the best that the adversary can do? In particular, we'd like to say that, no mater what compression strategy the adversary employs, it will be unable to learn anything about the contents of 99% of the ciphertexts, even if it later gets all the secret keys. Unfortunately, this is not generically true. For example, the adversary could store the first 1% of the bits of every ciphertext. If the encryption scheme is (e.g.,) the one-time pad, then an adversary who later learns the secret keys would later be able to learn the first 1% of every encrypted message of every user, which may provide a pretty good idea of the overall message contents. In fact, it can get even worse than this. If the encryption scheme is fully homomorphic, the adversary can individually compress each ciphertext into a small evaluated ciphertext encrypting some arbitrary predicate of the data (e.g., was the message insulting of the supreme leader), and therefore learn the outcome of this predicate about the encrypted data of every user. Even worse, if the

encryption scheme is multi-key fully homomorphic, the adversary can derive a compressed ciphertext that encrypts the output of a joint computation over all the data of all the users, as long as the output is sufficiently small. Thus, in general, an adversary whose storage capacity is only 1%, may still be able to learn some partial information about the encrypted messages of a 100% of the users. The question is then, whether or not it is indeed possible to guarantee only 1% of users are compromised, and if so to actually design such a scheme.

Connection to Incompressible Cryptography. Encryption schemes that offer protection against bounded-storage mass surveillance can be seen as a generalization of *incompressible encryption* [45,60,25] to the setting of multiple ciphertexts. To clarify the distinction, we refer to the earlier notion of incompressible encryption as *individually incompressible* and our new notion as *multi-incompressible*.

In an *individually incompressible encryption* scheme, we can make the size of a ciphertext flexibly large, and potentially huge (e.g., many gigabytes). An adversary observes a single ciphertext, but cannot store it in its entirety and can instead only store some compressed version of it. Security dictates that even if the adversary later gets the user's secret key, it cannot learn anything about the encrypted message. The work of Dziembowski [45] gave a construction of one-time symmetric-key encryption with information-theoretic security in this setting, and the previous chapter in this thesis showed how to achieve public-key encryption in this setting, under the minimal assumption that standard public-key encryption exists. The previous chapter, along with Branco *et al.* [25], also constructed such public-key encryption schemes having rate 1, meaning that the size of the message can be almost as large as the ciphertext size, and the latter work even showed how to do so under specific but standard public-key assumptions.

In our new notion of *multi-incompressible encryption*, we also have the flexibility to make the ciphertext size arbitrarily large. But now the adversary observes a large number of ciphertexts from many users and compresses them down to something that's roughly an α-fraction of the size of all the original ciphertexts, for some α. In particular, the adversary's storage may be much larger than a single ciphertext. Later the adversary gets all the secret keys, and we want to say that the adversary can only learn something about a (roughly) α-fraction of the messages, but cannot learn anything about the rest.

Our new notion of multi-incompressibility implies individual incompressibility. In particular, in the case of a single ciphertext, unless the adversary stores essentially all of it (i.e., $\alpha \approx 1$), it cannot learn anything about the encrypted message (= 100% of the messages). But our notion is significantly more general. For example, individual incompressibility does not even offer any guarantees if an adversary can take even 2 ciphertexts and compress them down to the size of 1, while multi-incompressibility ensures that one of the messages stays secure.

Formalizing multi-incompressibility is tricky: the natural indistinguishability-based approach would be to insist that the encryptions of two lists of messages are indistinguishable. But unlike individually incompressible encryption, in our setting the adversary can always learn *something*, namely the messages contained in ciphertexts it chose to store. We therefore need a fine-grained notion which captures that some messages to be learned, but other messages remain completely hidden. We give details on our solution below.

Extracting randomness against correlated sources. Before getting to our results, we discuss randomness extraction, which is a central tool in all existing constructions of incompressible encryption. A randomness extractor Ext takes as input a source of imperfect randomness X and uses it to distill out some (nearly) uniformly random string Y. Here,

we consider seeded extractors, which use a public uniformly random seed S as a catalyst to extract $Y = \mathsf{Ext}(X; S)$, such that Y should be (nearly) uniform even conditioned on the seed S.

While randomness extraction is very well studied, it is most often in the *single-use* case, where a single string $Y = \mathsf{Ext}(X; S)$ is extracted from a single source X having sufficient entropy. Here we ask: what if many strings $Y_i = \mathsf{Ext}(X_i; S_i)$ are extracted from multiple sources X_i respectively (using independent random seeds S_i), but where the sources X_i may be arbitrarily correlated? What guarantees can be made? We consider the case where we only know that the total joint entropy of all the sources is high, but we know nothing else about their individual entropies; indeed some of the sources may have no entropy at all! In this case, clearly not all of the extracted values Y_i can be uniform, and some may even be entirely deterministic. One may nevertheless hope that *some* of the extracted values remain uniform, where the fraction of uniform values roughly correlates to combined total entropy rate of all the sources. To the best of our knowledge, randomness extraction in this setting has not been studied before.

6.1.1 Our Results.

Formalizing Multi-user Incompressible Encryption. We first provide definitions for multi-user incompressible encryption. We depart from the indistinguishability-based definitions of the prior work on incompressible cryptography[45,60,25], and instead give a *simulation*-based definition. Essentially, the definition says that anything that an adversary can learn by taking many ciphertexts of different users, compressing them down sufficiently, and later getting all the secret keys, can be simulated by a simulator that can only ask to see some

small fraction of the plaintexts but does not learn anything about the remaining ones. In the single-instance case, this definition implies indistinguishability-based security, but appears stronger. Nevertheless, existing constructions and proofs are readily adapted to satisfy simulation security. The distinction becomes more important in the multi-user setting, however, where simulation security allows us to naturally define what it means for some messages to be revealed and some to remain hidden.

Multi-Instance Randomness Extractors. As our main technical tool, we explore a new kind of extractor that we call a multi-instance randomness extractor, which aims to solve the extraction problem outlined above. Syntactically, this is a standard extractor $Y = \mathsf{Ext}(X; S)$ that takes as input a source X and a seed S and outputs some short randomness Y. However, we now imagine that the extractor is applied separately to t correlated sources X_i, with each invocation using an independent seed S_i, to derive extracted values $Y_i = \mathsf{Ext}(X_i; S_i)$. The only guarantee on the sources is that the total joint min-entropy of $X = (X_1, \ldots, X_t)$ is sufficiently high. Any individual source X_i, however, may actually be deterministic (have 0 entropy), in which case the corresponding extracted value Y_i is of course not random. However, provided the total min-enropy rate of X is high, it is guaranteed that *many* of the t extracted values are statistically-close uniform. Ideally, if the joint min-entropy rate of X is α, we would hope that roughly αt of the extracted values are uniform.

Formalizing the above requires some care. For example, it may be the case that X is chosen by selecting a random index $i^* \leftarrow [t]$, setting X_{i^*} to be all 0's, and choosing the remaining block X_j for $j \neq i^*$ uniformly at random. In that case X has a very high entropy rate, but for any fixed index i, the min-entropy of X_i is small (at most $\log t$ since with polynomial probability $1/t$ the value of X_i is all 0's), and not enough to extract even 1 bit with negligible bias.

Therefore, we cannot argue that $Y_i = \mathsf{Ext}(X_i; S_i)$ is close to uniform for any particular index i! Instead, we allow the set of indices i, for which Y_i is close to uniform, itself be a random variable correlated with X. (See Definition 6.3.1.)

We show constructions of multi-instance randomness extractors nearing the optimal number of uniform extracted values. In particular, we show that if the joint min-entropy rate of $X = (X_1, \ldots, X_t)$ is α then there exists some random variable I_X denoting a subset of $\approx \alpha \cdot t$ indices in $[t]$ such that nobody can distinguish between seeing all the extracted values $Y_i = \mathsf{Ext}(X_i; S_i)$ versus replacing all the Y_i for $i \in I_X$ by uniform, even given all the seeds S_i. (See Corollary 6.3.1.) Our constructions are based on Hadamard codes (long seed) and Reed-Muller codes (short seed). While the constructions themselves are standard, our analysis is novel, leveraging the list-decodability of the codes, plus a property we identify called *hinting*. Hinting roughly means that the values of $\{\mathsf{Ext}(x; S_i)\}_i$ on some particular exponentially large set of pairwise independent seeds S_i can be compressed into a single small hint, of size much smaller than x. This hinting property is a crucial feature in the *local* list-decoding algorithms for these codes, but appears not to have been separately formalized/utilized as a design goal for an extractor.*

Applications. We then show that multi-instance randomness extraction can be used essentially as a drop-in replacement for standard randomness extractors in prior constructions of individual incompressible encryption, lifting them to multi-incompressible encryption.

*The work of Aggarwal *et al.* [1] studied a notion of extractors for "Somewhere Honest Entropic Look Ahead" (SHELA) sources. The notions are largely different and unrelated. In particular: (i) in our work X is an arbitrary source of sufficient entropy while Aggarwal *et al.* [1] places additional restrictions, (ii) we use a seeded extractor while Aggarwal *et al.* [1] wants a deterministic extractor, (iii) we apply the seeded extractor separately on each X_i while Aggarwal *et al.* [1] applies it jointly on the entire X, (iv) we guarantee that a large fraction of extracted outputs is uniform even if the adversary sees the rest, while in Aggarwal *et al.* [1] the adversary cannot see the rest.

As concrete applications, we obtain multi-incompressible encryption in a variety of settings:

- A symmetric key scheme with information-theoretic security, by replacing the extractor in Dziembowski[45].
- A "rate-1" symmetric key scheme, meaning the ciphertext is only slightly larger than the message. Here, we assume either decisional composite residuosity (DCR) or learning with errors (LWE), matching[25]*.
- A public key scheme, assuming any ordinary public key encryption scheme, matching the construction 5.3.1 from the previous chapter.
- A rate-1 public key scheme, under the same assumptions as Branco *et al.*[25]†. The scheme has large public keys.
- A rate-1 public key scheme that additionally has succinct public keys, assuming general functional encryption, matching construction 5.4.1 from the previous chapter.

In all cases, we guarantee that if the adversary's storage is an α fraction of the total size of all the ciphertexts, then it can only learn something about a $\beta \approx \alpha$ fraction of the encrypted messages. We can make $\beta = \alpha - 1/p(\lambda)$ for any polynomial p in the security parameter λ, by choosing a sufficiently large ciphertext size.

Multiple ciphertexts per user. Prior work, in addition to only considering a single user, also only considers a single ciphertext per user. Perhaps surprisingly, security does not

*One subtlety is that, for all of our rate-1 constructions, we need a PRG secure against *non-uniform* adversaries, whereas the prior work could have used a PRG against uniform adversaries.

†Branco *et al.*[25] explores CCA security, but in this chapter for simplicity we focus only on CPA security.

compose, and indeed for any fixed secret key size, we explain that simulation security for unbounded messages is impossible.

We therefore develop schemes for achieving a bounded number of ciphertexts per user. We show how to modify each of the constructions above to achieve multi-ciphertext security under the same assumptions.

The Random Oracle Model. We show how to construct symmetric key multi-user incompressible encryption with an unbounded number of ciphertexts per user and also essentially optimal secret key and ciphertext sizes, from random oracles. This shows that public key tools are potentially not inherent to rate-1 symmetric incompressible encryption.

6.1.2 Concurrent Work

A concurrent and independent work of Dinur *et al.*[41] (Section 6.2) considers an extraction problem that turns out to be equivalent to our notion of *Multi-Instance Randomness Extractor*. They study this problem in a completely different context of differential-privacy lower bounds. They show that (in our language) universal hash functions are "multi-instance randomness extractors" with good parameters, similar to the ones in our work. While conceptually similar, the results are technically incomparable since we show our result for hinting extractors while they show it for universal hash functions. One advantage of our result is that we show how to construct hinting extractors with short seeds, while universal hash functions inherently require a long seed. Their proof is completely different from the one in our paper.

The fact that multi-instance randomness extractors have applications in different contexts, as demonstrated in our work and Dinur *et al.*[41], further justifies them as a fundamen-

tal primitive of independent interest. We believe that having two completely different techniques/approaches to this problem is both interesting and valuable.

6.1.3 Our Techniques: Multi-Instance Randomness Extraction

We discuss how to construct a multi-instance randomness extractor Ext. Recall, we want to show that, if the joint min-entropy rate of $X = (X_1, \ldots, X_t)$ is α then there exists some random variable I_X denoting a subset of $\approx \alpha \cdot t$ indices in $[t]$ such that the distribution $(S_i, Y_i = \mathsf{Ext}(X_i; S_i))_{i \in [t]}$ is statistically indistinguishable from $(S_i, Z_i)_{i \in [t]}$ where Z_i is uniformly random for $i \in I_X$ and $Z_i = Y_i$ otherwise.

A failed approach. A natural approach would be to try to show that every standard seeded extractor is also a "multi-instance randomness extractor". As a first step, we would show that there is some random variable I_X denoting a large subset of $[t]$ such that the values X_i for $i \in I_X$ have large min-entropy conditioned on $i \in I_X$. Indeed, such results are known; see for example the "block-entropy lemma" of Dodis, Quach, and Wichs[42] (also Dziembowski[46] and Damgård *et al.*[34]). In fact, one can even show a slightly stronger statement that the random variables X_i for $i \in I_X$ have high min-entropy even conditioned on all past blocks $X_1, \ldots, X_{i-1}$. However, it cannot be true that X_i has high min-entropy conditioned on *all* other blocks past and future (for example, think of X being uniform subject to $\bigoplus_{i=1}^{t} X_i = 0$). Unfortunately, this prevents us for using the block-entropy lemma to analyze multi-instance extraction, where the adversary sees some extracted outputs from all the blocks.* It remains as a fascinating open problem whether every standard seeded extractor is

*This strategy would allow us to only prove a very weak version of multi-instance extraction when the number of blocks t is sufficiently small. In this case we can afford to lose the t extracted output bits from the entropy of *each* block. However, in our setting, we think of the number of blocks t as huge, much larger than

also a multi-instance randomness extractor or if there is some counterexample.*

OUR APPROACH. We are able to show that particular seeded extractors Ext based on Hadamard or Reed-Muller codes are good multi-instance randomness extractors. For concreteness, let us consider the Hadamard extractor $\mathsf{Ext}(x; s) = \langle x, s \rangle$.† Our proof proceeds in 3 steps:

Step 1: Switch quantifiers. We need to show that there *exists* some random variable I_X such that *every* statistical distinguisher fails to distinguish between the two distributions $(S_i, Y_i)_{i\in[t]}$ and $(S_i, Z_i)_{i\in[t]}$. We can use von Neumann's minimax theorem to switch the order quantifiers.‡ Therefore, it suffices to show that for every (randomized) statistical distinguisher D there is some random variable I_X such that D fails to distinguish the above distributions.

Step 2: Define I_X. For any fixed $x = (x_1, \ldots, x_t)$ we define the set I_x to consist of indices $i \in [t]$ such that D fails to distinguish between the hybrid distributions $(\{S_j\}_{j\in[t]}, Z_1, \ldots, Z_{i-1}, Y_i, \ldots, Y_t)$ versus $(\{S_j\}_{j\in[t]}, Z_1, \ldots, Z_i, Y_{i+1}, \ldots, Y_t)$, where in both distributions we condition on $X = x$. In other words, these are the indices where we can replace the next extracted output by random and fool the distinguisher. We then define the random variable I_X that chooses the correct set I_x according to X. It is easy to show via a simple hybrid argument that with this definition of I_X it is indeed true that D fails to distinguish $(S_i, Y_i)_{i\in[t]}$ and $(S_i, Z_i)_{i\in[t]}$.

Step 3: Argue that I_X is large. We still need to show that I_X is a large set, containing $\approx \alpha \cdot t$

the size/entropy of each individual block.

*We were initially convinced that the general result does hold and invested much effort trying to prove it via some variant of the above approach without success. We also mentioned the problem to several experts in the field who had a similar initial reaction, but were not able to come up with a proof.

†For the sake of exposition, here we only show the case where the extractor output is a single bit. In section 6.3, we construct extractors with multiple-bit outputs.

‡Think of the above as a 2 player game where one player chooses I_X, the other chooses the distinguisher and the payout is the distinguishing advantage; the minimax theorem says that the value of the game is the same no matter which order the players go in.

indices. To do so, we show that if I_X were small (with non negligible probability) then we could "guess" X with sufficiently high probability that would contradict X having high min-entropy. In particular, we provide a guessing strategy such that for any x for which I_x is small, our strategy has a sufficiently high chance of guessing x. First, we guess the small set $I_x \subseteq [t]$ as well as all of the blocks x_i for $i \in I_x$ uniformly at random. For the rest of the blocks $i \notin I_x$, we come up with a guessing strategy that does significantly better than guessing randomly. We rely on the fact that distinguishing implies predicting, to convert the distinguisher D into a predictor P such that for all $i \notin I_x$ we have: $P(S_i, \{S_j, \mathsf{Ext}(x_j; S_j)\}_{j\in[t]\setminus\{i\}}) = \mathsf{Ext}(x_i; S_i)$ with probability significantly better than $1/2$. Now we would like to use the fact that the Hadamard code $(\mathsf{Ext}(x; s) = \langle x, s\rangle)_s$ is list-decodable to argue that we can use such predictor P to derive a small list of possibilities for x. Unfortunately, there is a problem with this argument. To call the predictor, the predictor requires an auxiliary input, namely $\mathsf{aux}_i = \{S_j, \mathsf{Ext}(x_j; S_j)\}_{j\in[t]\setminus\{i\}}$. Supplying the aux_i in turn requires knowing at least t bits about x. We could hope to guess a good choice of aux_i, but there may be a different good choice for each $i \in [t]$, and therefore we would need to guess a fresh t bits of information about x just to recover each block x_i, which when $|x_i| < t$ is worse than the trivial approach of guessing x_i directly! Instead, we use a trick inspired by the proof of the Goldreich-Levin theorem. For each block $j \in [t]$, we guess the values of $b^{(k)} := \langle x_j, S_j^{(k)}\rangle$ for a very small "base set" of Q random seeds $S_j^{(1)}, \ldots, S_j^{(Q)}$. We can then expand this small "base set" of seeds into an exponentially larger "expanded set" of $2^Q - 1$ seeds $S_j^{(K)} := \sum_{k\in K} S_j^{(k)}$ for $K \subseteq [Q] \setminus \emptyset$, and derive guesses for $b^{(K)} := \langle x_j, S_j^{(K)}\rangle$ by setting $b^{(K)} = \sum_{k\in K} b^{(k)}$. By linearity, the expanded set of guesses is correct if the base set is correct, and moreover the expanded sets of seeds $(S_j^{(K)})_K$ are pairwise independent for different sets K. Therefore, for each set K, we can derive the corresponding

$\mathsf{aux}_i^{(K)}$. We can now apply Chebyshev's bound to argue that if for each i we take the majority value for $P(S_i, \mathsf{aux}_i^{(K)})$ across all $2^Q - 1$ sets K, it is likely equal to $\mathsf{Ext}(x_i; S_i)$ with probability significantly better than $1/2$. Notice that we got our saving by only guessing Qt bits about $x = (x_1, \ldots, x_t)$ for some small value Q (roughly $\log(1/\varepsilon)$ if we want indistinguishability ε) and were able to use these guesses to recover all the blocks x_i for $i \notin I_x$.

Generalizing. We generalize the above analysis for the Hadamard extractor to any extractor that is list-decodable and has a "hinting" property as discussed above. In particular, this also allows us to use a Reed-Muller based extractor construction with a much smaller seed and longer output length.

6.1.4 Our Techniques: Multi-Incompressible Encryption

We then move to considering incompressible encryption in the multi-user setting.

Definition. We propose a simulation-based security definition for multi-incompressible encryption. Roughly, the simulator first needs to simulate all the ciphertexts for all the instances *without* seeing any of the message queries, corresponding to the fact that at this point the adversary can't learn anything about any of the messages. To model the adversary then learning the secret keys, we add a second phase where the simulator can query for a *subset* of the messages, and then must simulate *all* the private keys. We require that no *space-bounded* distinguisher can distinguish between the receiving real encryptions/real private keys vs receiving simulated encryptions/keys. The number of messages the simulator can query will be related to the storage bound of the distinguisher.

Upgrading to multi-incompressible encryption using multi-instance randomness extraction. All prior standard-model constructions of individual incompressible encryption[45,60,25] utilize a randomness extractor. For example, Dziembowski[45] gives the following simple construction of a symmetric key incompressible encryption scheme:

- The secret key k is parsed as (s, k') where s is a seed for a randomness extractor, and k' is another random key.
- To encrypt a message m, choose a large random string R, and output $c = (R, d = \mathsf{Ext}(R; s) \oplus k' \oplus m)$.

The intuition for (individual) incompressible security is that an adversary that cannot store essentially all of c can in particular not store all of R, meaning R has min-entropy conditioned on the adversary's state. The extraction guarantee then shows that $\mathsf{Ext}(R; s)$ can be replaced with a random string, thus masking the message m.

We demonstrate that our multi-instance randomness extractors can be used as a drop-in replacement for ordinary random extractors in all prior constructions of individual incompressible encryption, upgrading them to multi-incompressible encryption. In the case of Dziembowski[45], this is almost an immediate consequence of our multi-instance randomness extractor definition. Our simulator works by first choosing random s for each user, and sets the ciphertexts of each user to be random strings. Then it obtains from the multi-instance randomness extractor guarantee the set of indices i where Y_i is close to uniform. For these indices, it sets k' to be a uniform random string. This correctly simulates the secret keys for these i.

For i where Y_i is not uniform, the simulator then queries for messages for these i. It

programs k' as $k' = d \oplus \mathsf{Ext}(R; s) \oplus m$; decryption under such k' will correctly yield m. Thus, we correctly simulate the view of the adversary, demonstrating multi-incompressible security.

Remark 6.1.1. *The set of indicies where Y_i is uniform will in general not be efficiently computable, and multi-instance randomness extraction only implies that the set of indices exist. Since our simulator must know these indices, our simulator is therefore inefficient. In general, an inefficient simulator seems inherent in the standard model, since the adversary's state could be scrambled in a way that hides which ciphertexts it is storing.*

We proceed to show that various constructions from the previous chapter and the work by Branco *et al.*[25] are also secure in the multi-user setting, when plugging in multi-instance randomness extractors. In all cases, the proof is essentially identical to the original single-user counterpart, except that the crucial step involving extraction is replaced with the multi-instance randomness extraction guarantee. We thus obtain a variety of parameter size/security assumption trade-offs, essentially matching what is known for the single-user setting.

One small issue that comes up is that, once we have invoked the multi-instance randomness extractor, the simulation is inefficient. This presents a problem in some of the security proofs, specifically in the "rate-1" setting where messages can be almost as large as ciphertexts. In the existing proofs in this setting, there is a computational hybrid step that comes *after* applying the extractor. Naively, this hybrid step would seem to be invalid since the reduction now has to be inefficient. We show, however, that the reduction can be made efficient as long as it is *non-uniform*, essentially having the choice of indices (and maybe some other quantities) provided as non-uniform advice. As long as the underlying primitive for these post-extraction hybrids has non-uniform security, the security proof follows.

Multiple ciphertexts per user. We also consider the setting where there may be multiple ciphertexts per user, which has not been considered previously.

It is not hard to see that having an *unbounded* number of ciphertexts per user is impossible in the standard model. This is because the simulator has to simulate everything but the secret key without knowing the message. Then, for the ciphertexts stored by the adversary, the simulator queries for the underlying messages and must generate the secret key so that those ciphertexts decrypt to the given messages. By incompressiblity, this means the secret key length must be at least as large as the number of messages.

We instead consider the case of bounded ciphertexts per user. For a stateful encryption scheme, it is trivial to upgrade a scheme supporting one ciphertext per user into one supporting many: simply have the secret key be a list of one-time secret keys. In the symmetric key setting, this can be made stateless by utilizing k-wise independent hash functions.

In the public key setting, achieving a stateless construction requires more work, and we do not believe there is a simple generic construction. We show instead how to modify all the existing constructions to achieve multiple ciphertexts per user. Along the way, we show an interesting combinatorial approach to generically lifting non-committing encryption to the many-time setting without sacrificing ciphertext rate.

Random Oracle Model. In Section 6.7, we finally turn to constructions in the random oracle model. We give a construction of symmetric key incompressible encryption with optimal key and ciphertext length, achieving security for an unbounded number of users and unbounded number of ciphertexts per user. As explained above, this is only possible because our simulator utilizes the random oracle: the incompressibility argument no longer applies since the simulator can covertly set the messages by programming random oracle queries.

The construction is essentially a 2-round unbalanced Feistel network.

We also show that standard hybrid encryption lifts essentially any random oracle-based symmetric key incompressible encryption to a public key scheme, assuming only general public key encryption. This significantly generalizes a construction of Branco *et al.*[25]. Note, however, that as observed by Branco *et al.*[25], the security of the scheme in the standard model is problematic: they show that if the PKE scheme is instantiated with fully homomorphic encryption, then there is a simple efficient attack that completely violates incompressible security. This gives a very natural random oracle uninstantiability result. In particular, all prior random oracle uninstantiabilities require a contrived instantiation of some building block*, whereas this uninstantiability only requires instantiating hybrid encryption with fully homomorphic encryption.

Remark 6.1.2. *Note that the underlying symmetric key scheme in Branco* et al.[25] *uses ideal ciphers instead of random oracles. Thus, their uninstantiability is only for the ideal cipher model.*[25] *claims the counterexample applies to random oracles, since random oracles and ideal ciphers are supposedly equivalent*[67]*. However, this is incorrect, as the equivalence only holds in the "single stage" setting*[88]*. Importantly, incompressible encryption is not a single stage game, owing to the space bound on the adversary's storage between receiving the ciphertexts and receiving the secret keys. In the more general multi-stage setting encompassing incompressible encryption, the equivalence of ideal ciphers and random oracles is open. Our generalized construction fixes this issue by directly designing our symmetric key scheme from random oracles.*

*For example, even the "natural" uninstnatiability of Fiat-Shamir[56] requires a contrived proof system.

Lemma 6.2.1 (Johnson Bound, Theorem 3.1 of[65]). *Let $\mathcal{C} \subseteq \Sigma^n$ with $|\Sigma| = q$ be any q-ary error-correcting code with relative distance $p_0 = 1 - (1+\rho)\frac{1}{q}$ for $\rho > 0$, meaning that for any two distinct values $x, y \in \mathcal{C}$, the Hamming distance between x, y is at least $p_0 \cdot n$. Then for any $\delta > \sqrt{\rho(q-1)}$ there exists some $L \leq \frac{(q-1)^2}{\delta^2 - \rho(q-1)}$ such that the code is $(p_1 = (1-(1+\delta)\frac{1}{q}), L)$-list decodable, meaning that for any $y \in \Sigma_q^n$ there exist at most L codewords $x \in \mathcal{C}$ that are within Hamming distance $p_1 n$ of y.*

Lemma 6.2.2 (Distinguishing implies Predicting). *For any randomized function $D : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ there exists some randomized function $P : \{0,1\}^n \to \{0,1\}^m$ such that for any jointly distributed random variables (A, B) over $\{0,1\}^n \times \{0,1\}^m$:*

$$\textit{if } \Pr[D(A,B)=1] - \Pr[D(A,U_m)=1] \geq \varepsilon \textit{ then } \Pr[P(A)=B] \geq \frac{1}{2^m}(1+\varepsilon).$$

Proof. Define $P(a)$ as follows. Sample a random $b_0 \leftarrow \{0,1\}^m$, if $D(a, b_0) = 1$ output b_0 else sample a fresh $b_1 \leftarrow \{0,1\}^m$ and output b_1.

Define $p = \Pr[D(A, U_m) = 1]$. Let B_0, B_1 be independent random variables that are uniform over $\{0,1\}^m$ corresponding to the strings b_0, b_1 . Then we have

$$\begin{aligned}
\Pr[P(A)=B] &= \Pr[D(A,B_0)=1 \wedge B_0 = B] + \Pr[D(A,B_0)=0 \wedge B_1 = B] \\
&= \Pr[B_0 = B]\Pr[D(A,B)=1] + \Pr[D(A,B_0)=0]\Pr[B_1=B] \\
&= \frac{1}{2^m}(\varepsilon + p) + (1-p)\frac{1}{2^m} = \frac{1}{2^m}(1+\varepsilon).
\end{aligned}$$

□

6.2.1 Incompressible Symmetric-Key Encryption (SKE)[45]

Similar to an incompressible PKE scheme discussed in Chapter 5, one can also imagine an analogous incompressible *symmetric* key encryption (SKE) scheme. This object has been studied earlier by Dziembowski under the name *forward-secure storage*[45]. The syntax of an incompressible SKE follows a standard SKE scheme. The "rate" is also defined the same as the ratio of the message length to the ciphertext length. The security of an incompressible SKE can be analogously defined through the following experiment $\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A},\Pi}(\lambda)$:

1. The adversary $\mathcal{A}_1$ takes 1^λ, and outputs a space bound 1^S.
2. Run $\mathsf{Gen}(1^\lambda, 1^S)$ to obtain the key k.
3. Sample a uniform bit $b \in \{0,1\}$.
4. The adversary submits an auxiliary input aux.
5. The adversary submits the challenge query consisting of two messages m_0 and m_1, and receives $\mathsf{ct} \leftarrow \mathsf{Enc}(k, m_b)$.
6. $\mathcal{A}_1$ produces a state st of size at most S.
7. The adversary $\mathcal{A}_2$ is given the tuple $(k, \mathsf{aux}, \mathsf{st})$ and outputs a guess b' for b. If $b' = b$, we say that the adversary succeeds and the output of the experiment is 1. Otherwise, the experiment outputs 0.

Definition 6.2.1 (Incompressible SKE Security)**.** *Let λ and S be security parameters. A symmetric key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to have incompressible SKE security*

if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*:*

$$\Pr\left[\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

6.3 Multi-Instance Randomness Extraction

6.3.1 Defining Multi-Instance Extraction

Definition 6.3.1 (Multi-Instance Randomness Extraction)**.** *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is* $(t, \alpha, \beta, \varepsilon)$-multi-instance extracting *if the following holds. Let* $X = (X_1, \ldots, X_t)$ *be any random variable consisting of blocks* $X_i \in \{0,1\}^n$ *such that* $H_\infty(X) \geq \alpha \cdot tn$*. Then, there exists some random variable* I_X *jointly distributed with* X*, such that* I_X *is supported over sets* $\mathcal{I} \subseteq [t]$ *of size* $|\mathcal{I}| \geq \beta \cdot t$ *and:*

$$(S_1, \ldots, S_t, \mathsf{Ext}(X_1; S_1), \ldots, \mathsf{Ext}(X_t; S_t)) \approx_\varepsilon (S_1, \ldots, S_t, Z_1, \ldots, Z_t)$$

where $S_i \in \{0,1\}^d$ *are uniformly random and independent seeds, and* $Z_i \in \{0,1\}^m$ *is sampled independently and uniformly random for* $i \in I_X$ *while* $Z_i = \mathsf{Ext}(X_i; S_i)$ *for* $i \notin I_X$*.*

In other words, the above definition says that if we use a "multi-instance extracting" extractor with independent seeds to individually extract from t correlated blocks that have a joint entropy-rate of α, then seeing all the extracted outputs is indistinguishable from replacing some carefully chosen β-fraction by uniform.

6.3.2 Hinting Extractors

Definition 6.3.2 (Hinting Extractor)**.** *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a* (δ, L, h, Q)*-hinting extractor if it satisfies the following:*

- *List Decodable: If we think of* $\mathsf{ECC}(x) = (\mathsf{Ext}(x;s))_{s\in\{0,1\}^d}$ *as a* $(2^d, n)_{\Sigma=\{0,1\}^m}$ *error-correcting code over the alphabet* $\Sigma = \{0,1\}^m$*, then the code is* $(p = 1-(1+\delta)2^{-m}, L)$*-list decodable, meaning that for any* $y \in \Sigma^{2^d}$*, the number of codewords that are within Hamming distance* $p \cdot 2^d$ *of* y *is at most* L*.*

- *Pairwise-Independent Hint: There exists some functions* $\mathsf{hint} : \{0,1\}^n \times \{0,1\}^\tau \to \{0,1\}^h$*, along with* rec_0 *and* rec_1 *such that:*

  - *For all* $x \in \{0,1\}^n, r \in \{0,1\}^\tau$*, if we define* $\sigma = \mathsf{hint}(x;r)$*,* $\{s_1, \ldots, s_Q\} = \mathsf{rec}_0(r)$*, and* $\{y_1, \ldots, y_Q\} = \mathsf{rec}_1(\sigma, r)$*, then* $\mathsf{Ext}(x; s_i) = y_i$ *for all* $i \in [Q]$*.*

  - *Over a uniformly random* $r \leftarrow \{0,1\}^\tau$*, the* Q *seeds* $\{s_1, \ldots, s_Q\} = \mathsf{rec}_0(r)$*, are individually uniform over* $\{0,1\}^d$ *and pairwise independent.*

Intuitively, the pairwise-independent hint property says that there is a small (size h) hint about x that allows us to compute $\mathsf{Ext}(x; s_i)$ for a large (size Q) set of pairwise independent seeds s_i. We generally want Q to be exponential in h.

The list-decoding property, on the other hand, is closely related to the standard definition of strong randomness extractors. Namely, if Ext is a (k, ε)-extractor then it is also $(p = 1 - (1+\delta)2^{-m}, 2^k)$-list decodable for $\delta = \varepsilon \cdot 2^m$, and conversely, if it is $(p = 1-(1+\delta)2^{-m}, 2^k)$-list deocdable then it is a $(k + m + \log(1/\delta), \delta)$-extractor (see Proposition 6.25 in[94]).

Construction 1: Hadamard. Define $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ via $\mathsf{Ext}(x;s) = \langle x, s\rangle$, where we interpret x, s as elements of $\mathbb{F}_{2^m}^{\hat{n}}$ for $\hat{n} := n/m$ and all the operations are over $\mathbb{F}_{2^m}$. The seed length is $d = n$ bits and the output length is m bits.

Lemma 6.3.1. *The above* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ *is a* (δ, L, h, Q)*-hinting extractor for any* $h, \delta > 0$ *with* $Q \geq 2^{h-m}$ *and* $L \leq 2^{2m}/\delta^2$.

Proof. The list-decoding bounds on δ, L come from the Johnson bound (Lemma 6.2.1) with $q = 2^m$, $\rho = 0$. For pairwise-independent hints, let $\hat{h} = h/m$ and define $\mathsf{hint}(x;R)$ to parse $R \in \mathbb{F}_{2^m}^{\hat{h} \times \hat{n}}$ and output $\sigma = R \cdot x^\top$, which has bit-size h. Let $\mathcal{V} \subseteq \mathbb{F}_{2^m}^{\hat{h}}$ be a set of vectors such that any two distinct vectors $v_1 \neq v_2 \in \mathcal{V}$ are linearly independent. Such a set $\mathcal{V}$ exists of size $Q = (2^m)^{\hat{h}-1} + (2^m)^{\hat{h}-2} + \cdots + 2^m + 1 \geq 2^{h-m}$, e.g., by letting $\mathcal{V}$ be the set of all non-zero vectors whose left-most non-zero entry is a 1. Define $\mathsf{rec}_0(R)$ so that it outputs $\{s_v = v \cdot R\}_{v \in \mathcal{V}}$. Correspondingly, $\mathsf{rec}_1(\sigma, R)$ outputs $\{y_v = \langle v, \sigma\rangle\}_{v \in \mathcal{V}}$. It's easy to see that the seeds s_v are individually uniform and pairwise independent, since for any linearly-independent $v_1 \neq v_2 \in \mathcal{V}$ and the value $s_{v_1} = v_1 R$ and $s_{v_2} = v_2 R$ are random and independent over a random choice of the matrix R. Moreover for all seeds s_v we have

$$\mathsf{Ext}(x, s_v) = \langle s_v, x\rangle = v \cdot R \cdot x^\top = \langle v, \sigma\rangle = y_v.$$

□

Construction 2: Hadamard ∘ Reed-Muller. Define $\mathsf{Ext}(f; s = (s_1, s_2)) = \langle f(s_1), s_2\rangle$, where $f \in \mathbb{F}_{2^w}^{\binom{\ell+g}{g}}$ is interpreted as a ℓ-variate polynomial of total degree g over some field

of size $2^w > g$, and $s_1 \in \mathbb{F}_{2^w}^{\ell}$ is interpreted as an input to the polynomial (this is Reed-Muler).* Then $y = f(s_1)$ and s_2 are interpreted as a values in $\mathbb{F}_{2^m}^{w/m}$ and the inner-product $\langle y, s_2 \rangle$ is computed over $\mathbb{F}_{2^m}$ (this is Hadamard). So overall, in bits, the input length is $n = w \cdot \binom{\ell+g}{g}$, the seed length is $d = w(\ell+1)$ and the output length is m. This code has relative distance $1 - (\frac{1}{2^m} + \frac{g}{2^w}) = 1 - \frac{1}{2^m}(1 + \frac{g}{2^{w-m}})$.

Lemma 6.3.2. *For any w, ℓ, g, m, δ such that $2^w > g$ and m divides w, if we set $n = w \cdot \binom{\ell+g}{g}$, $d = w(\ell+1)$ then the above* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a (δ, L, h, Q)-hinting extractor with $\delta = \sqrt{g 2^{2m}/2^w}$, $L = \frac{2^{2m}}{\delta^2 - g 2^{2m}/2^w}$, $h = w \cdot (g+1)$, $Q = 2^w$.*

In particular, for any n, m, w such that m divides w, we can set $\ell = g = \log n$ to get an (δ, L, h, Q)-hinting extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *with $d = O(w \log n)$, $\delta = 2^{m + \log\log n - w/2}$, $h = O(w \log n)$ and $Q = 2^w$.*

Proof. The list-decoding bounds on δ, L come from the Johnson bound (Lemma 6.2.1) with $q = 2^m$, $\rho = \frac{g}{2^{w-m}}$. On the other hand, for pairwise-independent hints, we can define $\mathsf{hint}(f; r)$ as follows. Parse $r = (r^0, r^1, s_1^1, \ldots, s_1^Q)$ with $r^0, r^1 \in \mathbb{F}_{2^w}^{\ell}$ and $s_1^i \in \mathbb{F}_{2^m}^{w/m}$. Let $\hat{f}(i) = f(r^0 + i \cdot r^1)$ be a univariate polynomial of degree g and define the hint $\sigma = \hat{f}$ to be the description of this polynomial. Define $\{s_i = (s_0^i, s_1^i)) = \mathsf{rec}_0(r)$ for $i \in \mathbb{F}_{2^w}$ by setting $s_0^i = r^0 + i \cdot r^1$. Define $\{y_i\} = \mathsf{rec}_1(\sigma, r)$ via $y_i = \langle \hat{f}(i), s_1^i \rangle$. It is easy to check correctness and pairwise independence follows from the fact that the values $s_0^i = r^0 + i \cdot r^1$ are pairwise independent over the randomness r^0, r^1. □

*Since the the input to the extractor is interpreted as a polynomial, we will denote it by f rather than the usual x to simplify notation.

6.3.3 Hinting-Extractors are Multi-Instance-Extracting

Lemma 6.3.3 (Multi-Instance-Extraction Lemma)**.** *Let* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a* (δ, L, h, Q)*-hinting extractor. Then, for any* $t, \alpha > 0$ *such that* $Q \geq 2t\frac{2^{2m}}{\delta^2}$*, it is also* $(t, \alpha, \beta, \varepsilon)$*-multi-instance extracting with* $\varepsilon = 6t\delta$ *and* $\beta = \alpha - \frac{\log L + h + \log t + \log(1/\varepsilon) + 3}{n}$.

Proof. Our proof follows a sequence of steps.

Step 0: Relax the Size Requirement. We modify the statement of the lemma as follows. Instead of requiring that $|I_X| \geq \beta \cdot t$ holds with probability 1, we relax this to requiring that $\Pr[|I_X| < \beta \cdot t] \leq \varepsilon/4$. On the other hand, we strengthen the requirement on statistical indisitnguishability from ε to $\varepsilon/2$:

$$(S_1, \ldots, S_t, \mathsf{Ext}(X_1; S_1), \ldots, \mathsf{Ext}(X_1; S_t)) \approx_{\varepsilon/2} (S_1, \ldots, S_t, Z_1, \ldots, Z_t).$$

This modified variant of the lemma implies the original.

To see this, notice that we can replace the set I_X that satisfies the modified variant with I'_X which is defined as $I'_X := I_X$ when $|I_X| \geq \beta t$ and $I'_X := \{1, \ldots, \beta t\}$ else. The set I'_X then satisfies the original variant. In particular, we can prove the indisintinguishability guarantee of the original lemma via a hybrid argument: replace I'_X by I_X ($\varepsilon/4$ statistical distance), switch from the left distribution to right distribution ($\varepsilon/2$ statistical distance), replace I_X back by I'_X ($\varepsilon/4$ statistical distance) for a total distance of ε.

Step 1: Change quantifiers. We need to prove that: *for all* X *with* $H_\infty(X) \geq \alpha \cdot tn$*, there* ***exists*** *some random variable* $I_X \subseteq [t]$ *with* $\Pr[|I_X| < \beta t] \leq \varepsilon/4$ *such that* ***for all***

(inefficient) distinguishers D:

$$\Pr[D(S_1, \ldots, S_t, Y_1, \ldots, Y_t) = 1] - \Pr[D(S_1, \ldots, S_t, Z_1, \ldots, Z_t) = 1] \leq \varepsilon/2 \qquad (6.1)$$

where we define $Y_i = \mathsf{Ext}(X_i; S_i)$*, and the random variables* Z_i *are defined as in the Lemma.* By the min-max theorem, we can switch the order of the last two quantifiers. In particular, it suffices to prove that: *for all X with* $H_\infty(X) \geq \alpha \cdot tn$ *and* ***for all*** *(inefficient, randomized) distinguishers D there* ***exists*** *some random variable* $I_X \subseteq [t]$ *with* $\Pr[|I_X| < \beta t] \leq \varepsilon/4$ *such that equation (6.1) holds.*

We can apply min-max because a distribution over inefficient distinguishers D is the same as a single randomized inefficient distinguisher D and a distribution over random variables I_X is the same as a single random variable I_X.

Step 2: Define I_X. Fix a (inefficient/randomized) distinguisher D.

For any fixed value $x \in \{0,1\}^{n \cdot t}$, we define a set $I_x \subseteq [t]$ iteratively as follows. Start with $I_x := \emptyset$. For $i = 1, \ldots, t$ add i to I_x if

$$\left(\begin{array}{ll} & \Pr[D(S_1, \ldots, S_t, Z_1^x \ldots, Z_{i-1}^x, Y_i^x, Y_{i+1}^x, \ldots, Y_t^x) = 1] \\ - & \Pr[D(S_1, \ldots, S_t, Z_1^x, \ldots, Z_{i-1}^x, U_m, Y_{1+1}^x, \ldots, Y_t^x) = 1] \end{array} \right) \leq 3\delta \qquad (6.2)$$

where S_i is uniform over $\{0,1\}^d$, $Y_j^x = \mathsf{Ext}(x_j; S_j)$ and for $j < i$ we define Z_j^x to be uniformly random over $\{0,1\}^m$ for $j \in I_x$, while $Z_j^x = Y_j^x$ for $j \notin I_x$. Note that $Y_i^x = (Y_i | X = x)$ and $Z_i^x = (Z_i | X = x)$.

Define I_X to be the random variable over the above sets I_x where x is chosen according to

X. With the above definition, equation 6.1 holds since:

$$
\begin{aligned}
&\Pr[D(S_1,\ldots,S_t,Y_1,\ldots,Y_t)=1]-\Pr[D(S_1,\ldots,S_t,Z_1,\ldots,Z_t)=1]\\
&=\mathbb{E}_{x\leftarrow X}\Pr[D(S_1,\ldots,S_t,Y_1,\ldots,Y_t)=1|X=x]-\Pr[[D(S_1,\ldots,S_t,Z_1,\ldots,Z_t)=1|X=x]\\
&=\mathbb{E}_{x\leftarrow X}\Pr[D(S_1,\ldots,S_t,Y_1^x,\ldots,Y_t^x)=1]-\Pr[D(S_1,\ldots,S_t,Z_1^x,\ldots,Z_t^x)=1]\\
&=\mathbb{E}_{x\leftarrow X}\sum_{i\in[t]}\underbrace{\left(\begin{array}{rl} & \Pr[D(S_1,\ldots,S_t,Z_1^x,\ldots,Z_{i-1}^x,Y_i^x,Y_{i+1}^x,\ldots,Y_t^x)=1]\\ - & \Pr[D(S_1,\ldots,S_t,Z_1^x,\ldots,Z_{i-1}^x,Z_i^x,Y_{i+1}^x,\ldots,Y_t^x)=1]\end{array}\right)}_{(*)}\\
&\leq 3t\delta=\varepsilon/2
\end{aligned}
$$

The last line follows since, for any x and any $i\in[t]$, if $i\notin I_x$ then $Y_i^x=Z_i^x$ and therefore $(*)=0$, and if $i\in I_x$ then $(*)\leq 3\delta$ by the way we defined I_x in equation (6.2).

Step 3: Argue I_X is large. We are left to show that

$$\Pr[|I_X|<\beta\cdot t]\leq\varepsilon/4. \tag{6.3}$$

We do this via a proof by contradiction. Assume otherwise that (6.3) does not hold. Then we show that we can guess X with high probability, which contradicts the fact that X has high min-entropy. In particular, we define a randomized function $\mathsf{guess}()$ such that, for any x for which $|I_x|<\beta\cdot t$, we have:

$$\Pr_{\hat{x}\leftarrow\mathsf{guess}()}[\hat{x}=x]\geq\frac{1}{4}\left(t^{\beta t+1}2^{bt}L^t2^{\beta tn}\right)^{-1}. \tag{6.4}$$

Then, assuming (6.3) does not hold, we have

$$\begin{aligned}\Pr_{\hat{x}\leftarrow\mathsf{guess}(),x\leftarrow X}[\hat{x}=x] &\geq \Pr_{x\leftarrow X}[|I_x|<\beta t]\Pr_{\hat{x}\leftarrow\mathsf{guess}(),x\leftarrow X}[\hat{x}=x \mid |I_x|<\beta t]\\ &\geq \frac{\varepsilon}{16}\left(t^{\beta t+1}2^{bt}L^t 2^{\beta tn}\right)^{-1}.\end{aligned}$$

which contradicts $H_\infty(X)\geq \alpha tn$.

Before defining the function $\mathsf{guess}()$, we note that by the definition of I_x in equation (6.2) and the"distinguishing implies predicting" lemma (Lemma 6.2.2), there exist some predictors P_i (depending only on D), such that, for all $x\in\{0,1\}^n$ and $i\notin I_x$, we have:

$$\Pr[P_i(S_1,\ldots,S_t,Z_1^x,\ldots,Z_{i-1}^x,Y_{i+1}^x,\ldots,Y_t^x)=Y_i^x]\geq\frac{1}{2^m}(1+3\delta) \tag{6.5}$$

The guessing strategy. We define $\mathsf{guess}()$ using these predictors P_i as follows:

1. Sample values $r_1,\ldots,r_t$ with $r_i\leftarrow\{0,1\}^\tau$.
2. Sample a set $\hat{I}_x\subseteq[t]$ of size $|\hat{I}_x|\leq\beta t$ uniformly at random.
3. Sample values $\hat{\sigma}_i\leftarrow\{0,1\}^b$ for $i\notin\hat{I}_x$ uniformly at random.
4. Sample values $\hat{x}_i\leftarrow\{0,1\}^n$ for $i\in\hat{I}_x$ uniformly at random.
5. Let $\{s_i^1,\ldots,s_i^Q\}=\mathsf{rec}_0(r_i)$, and $\{y_i^1,\ldots,y_i^Q\}=\mathsf{rec}_1(\hat{\sigma}_i,r_i)$.
6. Use all of the above values to define, for each $i\notin\hat{I}_x$, a randomized function $\hat{P}_i(s)$

which chooses a random $j^* \leftarrow [Q]$ and outputs:

$$\hat{P}_i(s) = P_i(s_1^{j^*}, \ldots, s_{i-1}^{j^*}, s, s_{i+1}^{j^*}, \ldots, s_t^{j^*}, z_1^{j^*}, \ldots, z_{i-1}^{j^*}, y_{i+1}^{j^*}, \ldots, y_t^{j^*})$$

where $z_i^{j^*} := y_i^{j^*}$ if $i \notin \hat{I}_x$ and $z_i^{j^*} \leftarrow \{0,1\}^m$ if $i \in \hat{I}_x$.

7. For each $i \notin \hat{I}_x$, define $\mathsf{cw}_i \in \Sigma^{2^d}$ by setting $\mathsf{cw}_i[s] \leftarrow \hat{P}_i(s)$, where $\Sigma = \{0,1\}^m$. Let $\mathcal{X}_i$ be the list of at most L values $\tilde{x}_i$ such that the Hamming distance between $\mathsf{ECC}(\tilde{x}_i)$ and cw_i is at most $(1+\delta)2^d$, as in Definition 6.3.2.

8. For each $i \notin \hat{I}_x$, sample $\hat{x}_i \leftarrow \mathcal{X}_i$.

9. Output $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_t)$.

Fix any x such that $|I_x| < \beta t$ and let us analyze $\Pr_{\hat{x} \leftarrow \mathsf{guess}()}[\hat{x} = x]$.

Event E_0. Let E_0 be the event that $\hat{I}_x = I_x$ and, for all $i \in I_x$: $\hat{x}_i = x_i$ and $\hat{\sigma}_i = \mathsf{hint}(x_i, r_i)$. Then $\Pr[E_0] \geq \left(t^{\beta t+1} 2^{bt} 2^{\beta t n}\right)^{-1}$. Let us condition on E_0 occurring for the rest of the analysis. In this case, we can replace all the "hatted" values $\hat{I}_x, \hat{\sigma}_i, \hat{x}_i$ with their "unhatted" counterparts $I_x, \sigma_i = \mathsf{hint}(x_i, r_i), x_i$ and we have $y_i^j = \mathsf{Ext}(x_i; s_i^j)$. Furthermore, since the "hatted" values were chosen uniformly at random, E_0 is independent of the choice of $r_1, \ldots, r_t$ and of all the "unhatted" values above; therefore conditioning on E_0 does not change their distribution.

Event E_1. Now, for any fixed choice of $r_1, \ldots, r_t$, define the corresponding procedure $\hat{P}_i$ to be "good" if

$$\Pr_{s \leftarrow \{0,1\}^d}[\hat{P}_i(s) = \mathsf{Ext}(x_i; s)] \geq (1+2\delta)\frac{1}{2^m},$$

where the probability is over the choice of $s \leftarrow \{0,1\}^d$ and the internal randomness of $\hat{P}_i$ (i.e., the choice of the index $j^* \leftarrow [Q]$ and the values $z_i^{j^*} \leftarrow \{0,1\}^m$ for $i \in I_x$). Let E_1 be the event that for all $i \notin I_x$ we have $\hat{P}_i$ is good, where the event is over the choice of $r_1, \ldots, r_t$. Define random variables V_i^j over the choice of $r_1, \ldots, r_t$ where

$$\begin{aligned} V_i^j &= \Pr_{s \leftarrow \{0,1\}^d}[\hat{P}_i(s) = \mathsf{Ext}(x_i; s) \mid j^* = j] \\ &= \Pr_{s \leftarrow \{0,1\}^d}[P_i(s_1^j, \ldots, s_{i-1}^j, s, s_{i+1}^j, \ldots, s_t^j, z_1^j, \ldots, z_{i-1}^j, y_{i+1}^j, \ldots, y_t^j) = \mathsf{Ext}(x_i; s)]. \end{aligned}$$

and $V_i := \sum_{j \in Q} V_i^j$. Then $\hat{P}_i$ is good iff $V_i \geq Q(1+2\delta)\frac{1}{2^m}$. By equation (6.5), we have $E[V_i] = \sum_j E[V_i^j] \geq Q(1+3\delta)\frac{1}{2^m}$. Furthermore, for any fixed i, the variables V_i^j are pairwise independent by Definition 6.3.2 and the fact that V_i^j only depends on s_i^j. Therefore $Var[V_i] = \sum_j Var[V_i^j] \leq Q$. We can apply the Chebyshev inequality to get:

$$\begin{aligned} \Pr[E_1|E_0] &\geq 1 - \Pr\left[\exists i \notin I_x \ : \ V_i < Q(1+2\delta)\frac{1}{2^m}\right] \\ &\geq 1 - \sum_{i \notin I_x} \Pr\left[V_i < Q(1+2\delta)\frac{1}{2^m}\right] \\ &\geq 1 - \sum_{i \notin I_x} \Pr\left[|V_i - E[V_i]| > Q\delta\frac{1}{2^m}\right] \geq 1 - t\frac{2^{2m}}{\delta^2 Q} \geq \frac{1}{2} \end{aligned}$$

Event E_2. Now fix any choice of the values in steps (1)-(6) such that E_0, E_1 hold. Let cw_i be the values sampled in step 7. Define the event E_2 to hold if for all $i \notin I_x$ the value cw_i agrees with $\mathsf{ECC}(x_i)$ is at least $(1+\delta)2^{d-m}$ coordinates, where the probability is only over the internal randomness used to sample the components $\mathsf{cw}_i(s) \leftarrow \hat{P}_i(s)$. We can define random variables W_i^s which are 1 if $\mathsf{cw}_i(s) = \mathsf{Ext}(x_i; s)$ and 0 otherwise. These variables

are mutually independent (since each invocation of $\hat{P}_i$ uses fresh internal randomness) and $E[\sum_s W_i^s] = 2^d \Pr_s[\hat{P}_i(s) = \mathsf{Ext}(x_i; s)] \geq (1+2\delta)2^{d-m}$. Therefore, by the Chernoff bound:

$$\begin{aligned}
\Pr[E_2 | E_1 \wedge E_0] &= 1 - \Pr[\exists i \notin I_x \ : \ \sum_s W_i^s \leq (1+\delta)2^{d-m}] \\
&\geq 1 - \sum_{i \notin I_x} \Pr[\sum_s W_i^s \leq (1+\delta)2^{d-m}] \\
&\geq 1 - t \cdot e^{-\delta^2 2^{d-m}/8} \geq \frac{1}{2}
\end{aligned}$$

Event E_3. Finally, fix any choice of the values in steps (1)-(7) such that E_0, E_1, E_2 hold. Let E_3 be the event that for each $i \notin \hat{I}_x$ if $\hat{x}_i \leftarrow \mathcal{X}_i$ is the value sampled in step (8) then $\hat{x}_i = x_i$. Then $\Pr[E_3 | E_2 \wedge E_1 \wedge E_0] \geq \left(\frac{1}{L}\right)^t$. Therefore, our guess is correct if E_0, E_1, E_2, E_3 all occur, which gives us the bound in equation (6.4). □

Corollary 6.3.1. *For any $n, m, t, \varepsilon > 0, \alpha > 0$, there exist extractors $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ that are $(t, \alpha, \beta, \varepsilon)$-multi-instance extracting with either:*

1. *seed length $d = n$ and $\beta = \alpha - \frac{O(m + \log t + \log(1/\varepsilon))}{n}$, or*
2. *seed length $d = O((\log n)(m + \log\log n + \log t + \log(1/\varepsilon)))$ and $\beta = \alpha - \frac{O(d)}{n}$.*

In particular, letting λ denote the security parameter, for any input length $n = \omega(\lambda \log \lambda)$ with $n < 2^\lambda$, for number of blocks $t < 2^\lambda$, any entropy rate $\alpha > 0$, there exists an extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with output length $m = \lambda$ and seed length $d = O(\lambda \log n)$, which is a $(t, \alpha, \beta, \epsilon = 2^{-\lambda})$-multi-instance randomness extractor with $\beta = \alpha - o(1)$. In other words, the fraction of extracted values that can be replaced by uniform is nearly α.

Utilizing multi-instance randomness extractors, we can now explore the multi-user setting for incompressible encryptions. But first, we need to formally define what it means for an incompressible PKE or SKE scheme to be multi-user secure.

We propose a simulation-based security definition. Roughly, the simulator first needs to simulate all the ciphertexts for all the instances *without* seeing any of the message queries. So far, this is akin to the standard semantic security notion for encryption. But we need to now model the fact that the adversary can store ciphertexts for later decryption, at which point it has all the private keys. We therefore add a second phase where the simulator can query for a *subset* of the messages, and then must simulate *all* the private keys. We require that no space-bounded distinguisher can distinguish between receiving real encryptions/real private keys vs receiving simulated encryptions/keys. The number of messages the simulator can query is related to the storage bound of the distinguisher.

Put formally, let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme, to define simulation-based incompressible ciphertext security for the multiple-instance setting, consider the following two experiments:

- In the real mode experiment, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ interacts with the challenger $\mathcal{C}$, who has knowledge of all the adversary's challenge messages.

  Real Mode $\mathsf{ExpReal}^{\Pi}_{\mathcal{C},\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}(\lambda, \eta, \ell, S)$:

  1. For $i \in [\eta]$, the challenger $\mathcal{C}$ runs $\mathsf{Gen}(1^\lambda, 1^S)$ to sample $(\mathsf{pk}_i, \mathsf{sk}_i)$.
  2. The challenger $\mathcal{C}$ sends all the pk_i's to $\mathcal{A}_1$.

3. For each $i \in [\eta]$, $\mathcal{A}_1$ can produce up to ℓ message queries $\{m_{i,j}\}_{j\in[\ell]}$. The adversary submits all of the message queries *in one single batch* $\{m_{i,j}\}_{i,j}$ and receives $\{\mathsf{ct}_{i,j}\}_{i,j}$ where $\mathsf{ct}_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, m_{i,j})$.
4. $\mathcal{A}_1$ produces a state st of size at most S.
5. On input of st, $\{m_{i,j}\}_{i,j}$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- In the ideal mode experiment, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ interacts with a simulator $\mathcal{S}$, which needs to simulate the view of the adversary with no/partial knowledge of the challenge messages.

  Ideal Mode $\mathsf{ExpIdeal}^{\Pi}_{\mathcal{S},\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}(\lambda, \eta, \ell, q, S)$:

  1. For $i \in [\eta]$, the simulator $\mathcal{S}$ samples pk_i.
  2. The simulator $\mathcal{S}$ sends all the pk_i's to $\mathcal{A}_1$.
  3. For each $i \in [\eta]$, and $j \in [\ell]$, $\mathcal{A}_1$ produces $m_{i,j}$. All of the queries $\{m_{i,j}\}_{i,j}$ are submitted in one batch and the simulator $\mathcal{S}$ produces $\{\mathsf{ct}_{i,j}\}_{i,j}$ *without* seeing $\{m_{i,j}\}_{i,j}$.
  4. $\mathcal{A}_1$ produces a state st of size at most S.
  5. The simulator now submits up to q number of (i, j) index pairs, and receives the corresponding messages $m_{i,j}$'s. Then $\mathcal{S}$ simulates all the secret keys sk_i's.
  6. On input of st, $\{m_{i,j}\}_{i,j}$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

  Notice that the simulator needs to simulate the ciphertexts first without knowing the corresponding messages, and then sample the secret keys so that the ciphertexts appear appropriate under the given messages.

Definition 6.4.1 (Multi-Instance Simulation-Based CPA Security)**.** *For security parameters* $\lambda, \eta(\lambda), \ell(\lambda), q(\lambda)$ *and* $S(\lambda)$*, a public key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* (η, ℓ, q, S)-MULT-SIM-CPA *secure if for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a simulator* $\mathcal{S}$ *such that:*

$$\left|\Pr\left[\mathsf{ExpReal}^{\Pi}_{\mathcal{C},\mathcal{A}}(\lambda, \eta, \ell, S) = 1\right] - \Pr\left[\mathsf{ExpIdeal}^{\Pi}_{\mathcal{S},\mathcal{A}}(\lambda, \eta, \ell, q, S) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Remark 6.4.1. *If* $\ell = 1$*, we say that the scheme has only single-ciphertext-per-user security. For* $\ell > 1$*, we say that the scheme has multi-ciphertext-per-user security.*

Remark 6.4.2. *Notice that by replacing the underlying PKE scheme with a Symmetric Key Encryption (SKE) scheme and modifying corresponding syntaxes (sample only* sk*'s instead of* $(\mathsf{pk}, \mathsf{sk})$ *pairs, and remove step* 2 *of the experiments where the adversary receives the* pk*'s), we can also get a* MULT-SIM-CPA *security definition for SKE schemes.*

6.5 Symmetric Key Incompressible Encryption

In this section, we explore the multi-user security of incompressible SKEs, both in the low-rate setting and the rate-1 setting. We also present a generic lifting technique to obtain an SKE with multi-ciphertext-per-user security from an SKE with single-ciphertext-per-user security.

6.5.1 Low Rate Incompressible SKE

For low rate incompressible SKE, it follows almost immediately from multi-instance randomness extractors that the forward-secure storage by Dziembowski[45] is MULT-SIM-CPA secure

(by using multi-instance randomness extractors as the "BSM function" and using One Time Pad (OTP) as the underlying SKE primitive).

First, let us recall the construction by Dziembowski[45], with the multi-instance randomness extractors and OTP plugged in.

Construction 6.5.1 (Forward-Secure Storage[45])**.** *Let λ and S be security parameters. Given $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ a $(t, \alpha, \beta, \epsilon)$-multi-instance randomness extractor as defined in Definition 6.3.1 where the seed length $d = \mathsf{poly}(\lambda)$, output length $w = \mathsf{poly}(\lambda)$ and $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$, the construction $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for message space $\{0,1\}^w$ works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Sample a seed $s \leftarrow \{0,1\}^d$ for the randomness extractor, and a key $k' \leftarrow \{0,1\}^w$. Output $k = (s, k')$.*
- $\mathsf{Enc}(k, m)$*: To encrypt a message m, first parse $k = (s, k')$ and sample a long randomness $R \leftarrow \{0,1\}^n$. Compute the ciphertext as $\mathsf{ct} = (R, \mathsf{ct}' = \mathsf{Ext}(R; s) \oplus k' \oplus m)$.*
- $\mathsf{Dec}(k, \mathsf{ct})$*: First, parse $\mathsf{ct} = (R, \mathsf{ct}')$ and $k = (s, k')$. Then compute $m = \mathsf{Ext}(R; s) \oplus k' \oplus \mathsf{ct}'$.*

Correctness is straightforward. Construction 6.5.1 is also MULT-SIM-CPA secure. Essentially, the simulator simply sends ct_i's as uniformly random strings. Then when the simulator sends the keys, it would use the simulator for the multi-instance randomness extractor to get the index subset $I \subset [\eta]$, and for $i \in I$, simply send k_i as a uniformly random string. For $i \notin I$, it samples the extractor seed s_i and then compute $k'_i = m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus \mathsf{ct}'_i$. Notice that for $i \notin I$, $\mathsf{ct}'_i = m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus k'_i$, and for $i \in I$, $\mathsf{ct}'_i = m_i \oplus u_i \oplus k'_i$ where

u_i is a w-bit uniform string. This is now just the definition of multi-instance randomness extractors.

We prove below the MULT-SIM-CPA security of Construction 6.5.1 formally through a sequence of hybrids.

In hybrid 0, we start with the ideal mode experiment $\mathsf{ExpIdeal}^{\Pi}_{\mathcal{S},\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}$ with a specific simulator plugged in, and through the sequence of hybrids, we gradually move towards the real mode experiment $\mathsf{ExpReal}^{\Pi}_{\mathcal{S},\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}$.

Sequence of Hybrids

- Hybrid H_0: The ideal mode experiment $\mathsf{ExpIdeal}^{\Pi}_{C,\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}(t, 1, (1-\beta)t, S)$.

    1. For each $i \in [t]$, $\mathcal{A}_1$ produces m_i. All of the queries $\{m_i\}_i$ are submitted in a single batch and *not* available to the simulator $\mathcal{S}$. $\mathcal{S}$ samples uniformly random ciphertexts $\mathsf{ct}_i = (R_i, \mathsf{ct}'_i)$, and hence is able to produce $\{\mathsf{ct}_i\}_i$ *without* seeing $\{m_i\}_i$.
    2. $\mathcal{A}_1$ produces a state st of size at most S.
    3. The simulator $\mathcal{S}$ runs the simulator for the multi-instance randomness extractor to get a set of indices $I \subseteq [t]$ with $|I| \geq \beta t$. The simulator now submits the set $[t] \backslash I$, and receives the corresponding messages $\{m_i\}_{i \notin I}$. Then $\mathcal{S}$ simulates all the keys k_i's. For $i \in I$, sample a uniform $k_i \leftarrow \{0,1\}^w$. For $i \notin I$, sample a uniform seed s_i, and compute $k_i = (s_i, m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus \mathsf{ct}'_i)$.
    4. On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_1: The same as H_0, except that in step 3, for $i \in I$, sample a uniform s_i and compute $k_i = (s_i, m_i \oplus u_i \oplus \mathsf{ct}'_i)$, where u_i is a uniformly random w-bit string.

- Hybrid H_2: The same as H_1, except that in step 3, for all i, sample a uniform seed s_i, and compute $k_i = (s_i, m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus \mathsf{ct}'_i)$. Notice that the game is now identical to the real mode experiment $\mathsf{ExpReal}^{\Pi}_{\mathcal{C},\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}$, where we send the adversary faithful encryptions of the message queries.

Proof of Hybrid Arguments

Lemma 6.5.1. *No adversary can distinguish between H_0 and H_1 with non-negligible probability.*

Proof. Notice that the only difference between H_0 and H_1 is that in H_0, for $i \in I$, we sample a uniform s_i and a uniform k'_i, and in H_1, we sample a uniform s_i and compute k'_i as $m_i \oplus u_i \oplus \mathsf{ct}'_i$, where u_i is a uniform w-bit string. This is just an One Time Pad (OTP) encryption of $m_i \oplus \mathsf{ct}'_i$, and hence should be indistinguishable from a uniformly random k'_i by the information-theoretic security of OTP. □

Lemma 6.5.2. *If $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ is a $(t, \alpha, \beta, \epsilon)$-multi-instance randomness extractor with $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$, then no adversary can distinguish between H_1 and H_2 with non-negligible probability.*

Proof. First, notice the difference between H_1 and H_2. In H_2, for all i, we have $k_i = (s_i, m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus \mathsf{ct}'_i)$. In H_1, for $i \in I$, $k_i = (s_i, m_i \oplus u_i \oplus \mathsf{ct}'_i)$. For $i \notin I$, $k_i = (s_i, m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus \mathsf{ct}'_i)$ is the same.

Notice that each R_i is a uniformly random n-bit string independent of m_i. So by lemma 2.1.1, $H_\infty(\{R_i\}_i|\mathsf{st}, \{m_i\}_i) = H_\infty(\{R_i\}_i|\mathsf{st}) \geq nt - n(1-\alpha)t = \alpha \cdot tn$, i.e. $\{R_i\}_i$ has at least $\alpha \cdot tn$ bits of min-entropy conditioned on the adversary's view. And recall that I is the set of indices output by the multi-instance randomness extractor simulator. We can invoke the property of the multi-instance randomness extractor, and hence have

$$(s_1, \ldots, s_t, \mathsf{Ext}(R_1; s_1), \ldots, \mathsf{Ext}(R_t; s_t)) \approx_\epsilon (s_1, \ldots, s_t, Z_1, \ldots, Z_t),$$

where $Z_i = u_i$ for all $i \in I$, and $Z_i = \mathsf{Ext}(R_i; s_i)$ for all $i \notin I$. Notice that in H_1, we equivalently have $k_i = (s_i, m_i \oplus Z_i \oplus \mathsf{ct}'_i)$, and in H_2, we have $k_i = (s_i, m_i \oplus \mathsf{Ext}(R_i; s_i) \oplus \mathsf{ct}'_i)$. The only difference is that in H_1 we have the Z_i's instead of the $\mathsf{Ext}(R_i; s_i)$'s in H_2, and these are indistinguishable by the extractor property. Hence, no adversary can distinguish between H_1 and H_2 with non-negligible probability. □

Theorem 6.5.1. *Let λ, S be security parameters. If $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ is a $(t, \alpha, \beta, \epsilon)$-multi-instance randomness extractor with $d, w = \mathsf{poly}(\lambda)$ and $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$, then Construction 6.5.1 is $(t, 1, (1-\beta)t, S)$-*MULT-SIM-CPA *secure.*

Proof. The lemmas above show a sequence of a polynomial number of hybrid experiments where no adversary can distinguish one from the next with non-negligible probability. Notice that the first hybrid H_0 corresponds to the ideal mode experiment of multi-user security, and the last hybrid H_2 corresponds to the real mode one. The simulation-based security follows. □

Remark 6.5.1. *While* MULT-SIM-CPA *security only requires that no* PPT *adversaries can distinguish between the real mode and the ideal mode experiments, what we have proved for con-*

struction 6.5.1 here is that it is actually MULT-SIM-CPA *secure against* all (potentially computationally unbounded) *adversaries, and hence is information theoretically* MULT-SIM-CPA *secure.*

6.5.2 Rate-1 Incompressible SKE

Branco, Döttling and Dujmovic[25] construct rate-1 incompressible SKE from HILL-Entropic Encodings[79], extractors and PRGs. We show that by replacing the extractors with multi-instance randomness extractors and slightly modifying the scheme, we get MULT-SIM-CPA security.

First, we recall the definitions and security requirements of a HILL-Entropic Encoding scheme[79].

Definition 6.5.1 (HILL-Entropic Encoding[79])**.** *Let λ be the security parameter. An (α, β)-HILL-Entropic Encoding in the common random string setting is a pair of PPT algorithms* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *that works as follows:*

- $\mathsf{Enc}_{\mathsf{crs}}(1^\lambda, m) \to c$*: On input the common random string* crs*, the security parameter, and a message, outputs a codeword c.*
- $\mathsf{Dec}_{\mathsf{crs}}(c) \to m$*: On input the common random string and a codeword, outputs the decoded message m.*

It satisfies the following properties.

Correctness. For all $\lambda \in \mathbb{N}$ and $m \in \{0,1\}^*$, $\Pr[\mathsf{Dec}_{\mathsf{crs}}(\mathsf{Enc}_{\mathsf{crs}}(1^\lambda, m)) = m] \geq 1 - \mathsf{negl}(\lambda)$.

α-**Expansion.** For all $\lambda, k \in \mathbb{N}$ and for all $m \in \{0,1\}^k$, $|\mathsf{Enc}_{\mathsf{crs}}(1^\lambda, m)| \leq \alpha(\lambda, k)$.

β-**HILL-Entropy.** There exists a simulator algorithm SimEnc such that for all polynomial $k = k(\lambda)$ and any ensemble of messages $m = \{m_\lambda\}$ of length $k(\lambda)$, consider the following real mode experiment:

- $\mathsf{crs} \leftarrow \{0,1\}^{t(\lambda,k)}$
- $c \leftarrow \mathsf{Enc}_{\mathsf{crs}}(1^\lambda, m_\lambda)$

and let CRS, C denote the random variables for the corresponding values in the real mode experiment. Also consider the following simulated experiment:

- $(\mathsf{crs}', c') \leftarrow \mathsf{SimEnc}(1^\lambda, m_\lambda)$

and let CRS', C' be the corresponding random variables in the simulated experiment. We require that $(\mathsf{CRS}, C) \approx_c (\mathsf{CRS}', C')$ and that $H_\infty(C'|\mathsf{CRS}') \geq \beta(\lambda, k)$.

Moran and Wichs[79] show that we can construct HILL-Entropic Encodings in the CRS model from either the Decisional Composite Residuosity (DCR) assumption[82,35] or the Learning with Errors (LWE) problem[87]. Their construction achieves $\alpha(\lambda, k) = k(1 + o(1)) + \mathsf{poly}(\lambda)$ and $\beta(\lambda, k) = k(1 - o(1)) - \mathsf{poly}(\lambda)$, which we call a "good" HILL-entropic encoding.

Now we reproduce the construction from [25] with the multi-instance randomness extractors and some other minor changes (highlighted below).

Construction 6.5.2 ([25])**.** *Let λ and S be security parameters. Given* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ *a $(t, \alpha, \beta, \epsilon)$-multi-instance randomness extractor as defined in Definition 6.3.1 where*

the seed length $d = \mathsf{poly}(\lambda)$, $w = \mathsf{poly}(\lambda)$ *and* $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$, $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *a "good"* (α', β')*-HILL-Entropic Encoding scheme, and* $\mathsf{PRG} : \{0,1\}^w \to \{0,1\}^n$ *a pseudorandom generator secure against non-uniform adversaries, the construction* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for message space* $\{0,1\}^n$ *works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Sample a seed* $s \leftarrow \{0,1\}^d$ *for the randomness extractor, a common random string* $\mathsf{crs} \in \{0,1\}^{\mathsf{poly}(\lambda,n)}$ *for the HILL-Entropic Encoding, and a random pad* $r \leftarrow \{0,1\}^n$*. Output* $k = (s, r, \mathsf{crs})$.

- $\mathsf{Enc}(k, m)$*: To encrypt a message* m*, first parse* $k = (s, r, \mathsf{crs})$ *and sample a random PRG seed* $s' \leftarrow \{0,1\}^w$*. Compute* $c_1 = \mathsf{Code.Enc}_{\mathsf{crs}}(1^\lambda, \mathsf{PRG}(s') \oplus r \oplus m)$ *and* $c_2 = s' \oplus \mathsf{Ext}(c_1, s)$*. The final ciphertext is* $\mathsf{ct} = (c_1, c_2)$.

- $\mathsf{Dec}(k, \mathsf{ct})$*: First, parse* $\mathsf{ct} = (c_1, c_2)$ *and* $k = (s, r, \mathsf{crs})$*. Then compute* $s' = \mathsf{Ext}(c_1; s) \oplus c_2$ *and obtain* $m = \mathsf{Code.Dec}_{\mathsf{crs}}(c_1) \oplus \mathsf{PRG}(s') \oplus r$.

Correctness follows from the original construction and should be easy to verify. Notice that by the α'-expansion of the "good" HILL-entropic encoding, the ciphertexts have length $(1 + o(1))n + w + \mathsf{poly}(\lambda) = (1 + o(1))n + \mathsf{poly}(\lambda)$ (the $\mathsf{poly}(\lambda)$ part is independent of n), while the messages have length n. Hence the scheme achieves an optimal rate of 1 ($(1 - o(1))$ to be exact). The keys are bit longer though, having size $d + n + \mathsf{poly}(\lambda, n) = n + \mathsf{poly}(\lambda, n)$. Furthermore, Moran and Wichs[79] show that the CRS needs to be at least as long as the message being encoded. Thus the key has length at least $2n + \mathsf{poly}(\lambda)$.

We prove security of Construction 6.5.2 through a sequence of hybrids.

Sequence of Hybrids

- Hybrid H_0:
  - Run the adversary $\mathcal{A}_1$ to receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$:
    * Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
    * Sample $r_i \leftarrow \{0,1\}^n$ uniformly at random.
    * Sample $s'_i \leftarrow \{0,1\}^w$ uniformly at random.
    * Sample crs_i uniformly at random.
    * Let $c_{1,i} \leftarrow \mathsf{Code.Enc}_{\mathsf{crs}_i}(1^\lambda, \mathsf{PRG}(s'_i) \oplus r_i \oplus m_i)$.
    * Let $c_{2,i} \leftarrow s'_i \oplus \mathsf{Ext}(c_{1,i}; s_i)$.
    * Let $\mathsf{ct}_i = (c_{1,i}, c_{2,i})$.
  - Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
  - Let $\{k_i\}_i = \{(s_i, r_i, \mathsf{crs}_i)\}_i$.
  - On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.
- Hybrid H_1:
  - Run the adversary $\mathcal{A}_1$ to receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$:
    * Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
    * Sample $r_i \leftarrow \{0,1\}^n$ uniformly at random.

* Sample $s'_i \leftarrow \{0,1\}^w$ uniformly at random.
* Let $(\mathsf{crs}_i, c_{1,i}) \leftarrow \mathsf{SimEnc}(1^\lambda, \mathsf{PRG}(s'_i) \oplus r_i \oplus m_i)$.
* Let $c_{2,i} \leftarrow s'_i \oplus \mathsf{Ext}(c_{1,i}; s_i)$.
* Let $\mathsf{ct}_i = (c_{1,i}, c_{2,i})$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– Let $\{k_i\}_i = \{(s_i, r_i, \mathsf{crs}_i)\}_i$.

– On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

• Hybrid H_2:

– Run the adversary $\mathcal{A}_1$ to receive $\{m_i\}_i$ for $i \in [t]$.

– For each $i \in [t]$:

* Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
* Sample $u_i \leftarrow \{0,1\}^n$ uniformly at random.
* Sample $s'_i \leftarrow \{0,1\}^w$ uniformly at random.
* Let $(\mathsf{crs}_i, c_{1,i}) \leftarrow \mathsf{SimEnc}(1^\lambda, u_i)$.
* Let $c_{2,i} \leftarrow s'_i \oplus \mathsf{Ext}(c_{1,i}; s_i)$.
* Let $\mathsf{ct}_i = (c_{1,i}, c_{2,i})$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– For each $i \in [t]$:

* Let $r_i = u_i \oplus \mathsf{PRG}(s'_i) \oplus m_i$.
* Let $k_i = (s_i, r_i, \mathsf{crs}_i)$.

- On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_3:

  - Run the adversary $\mathcal{A}_1$ to receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$:
    - Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
    - Sample $u_i \leftarrow \{0,1\}^n$ uniformly at random.
    - Let $(\mathsf{crs}_i, c_{1,i}) \leftarrow \mathsf{SimEnc}(1^\lambda, u_i)$.
    - Sample $c_{2,i} \leftarrow \{0,1\}^w$ uniformly at random.
    - Let $\mathsf{ct}_i = (c_{1,i}, c_{2,i})$.
  - Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
  - For each $i \in [t]$:
    - Let $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus \mathsf{Ext}(c_{1,i}; s_i)) \oplus m_i$.
    - Let $k_i = (s_i, r_i, \mathsf{crs}_i)$.
  - On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_4:

  - Run the adversary $\mathcal{A}_1$ to receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$:
    - Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
    - Sample $u_i \leftarrow \{0,1\}^n$ uniformly at random.

* Let $(\mathsf{crs}_i, c_{1,i}) \leftarrow \mathsf{SimEnc}(1^\lambda, u_i)$.
* Sample $c_{2,i} \leftarrow \{0,1\}^w$ uniformly at random.
* Let $\mathsf{ct}_i = (c_{1,i}, c_{2,i})$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– Run the simulator for the multi-instance randomness extractor to get a set of indices $I \subseteq [t]$ with $|I| \geq \beta t$. For each $i \in [t]$:

* If $i \in I$, let $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus v_i) \oplus m_i$ where v_i is a uniformly sampled w-bit string.
* If $i \notin I$, let $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus \mathsf{Ext}(c_{1,i}; s_i)) \oplus m_i$.
* Let $k_i = (s_i, r_i, \mathsf{crs}_i)$.

– On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_5:

– Run the adversary $\mathcal{A}_1$ to receive $\{m_i\}_i$ for $i \in [t]$. Discard $\{m_i\}_i$ without looking at it.

– For each $i \in [t]$:

* Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
* Sample $u_i \leftarrow \{0,1\}^n$ uniformly at random.
* Let $(\mathsf{crs}_i, c_{1,i}) \leftarrow \mathsf{SimEnc}(1^\lambda, u_i)$.
* Sample $c_{2,i} \leftarrow \{0,1\}^w$ uniformly at random.
* Let $\mathsf{ct}_i = (c_{1,i}, c_{2,i})$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– Run the simulator for the multi-instance randomness extractor to get a set of indices $I \subseteq [t]$ with $|I| \geq \beta t$. Submit the set $[t] \backslash I$, and receive the corresponding messages $\{m_i\}_{i \notin I}$. For each $i \in [t]$:

  * If $i \in I$, sample a uniform $r_i \leftarrow \{0,1\}^n$.
  * If $i \notin I$, let $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus \mathsf{Ext}(c_{1,i}; s_i)) \oplus m_i$.
  * Let $k_i = (s_i, r_i, \mathsf{crs}_i)$.

– On input of st, $\{m_i\}_i$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

Proof of Hybrid Arguments

Lemma 6.5.3. *If* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *has* β'*-HILL-entropy, then no PPT adversary can distinguish between* H_0 *and* H_1 *with non-negligible probability.*

Proof. The only difference between H_0 and H_1 is that in H_0, crs_i is sampled uniformly random and $c_{1,i} \leftarrow \mathsf{Code.Enc}_{\mathsf{crs}_i}(1^\lambda, \mathsf{PRG}(s'_i) \oplus r_i \oplus m_i)$, while in H_1, we get $(\mathsf{crs}_i, c_{1,i}) \leftarrow \mathsf{SimEnc}(1^\lambda, \mathsf{PRG}(s'_i) \oplus r_i \oplus m_i)$. By the β'-HILL-entropy, the crs_i and $c_{1,i}$ in H_0 are computationally indistinguishable from the ones in H_1. Hence, no PPT adversary can distinguish between H_0 and H_1 with non-negligible probability. □

Lemma 6.5.4. *No adversary can distinguish between* H_1 *and* H_2 *with non-negligible probability.*

Proof. Here we are just changing the ways the variables are sampled. In H_1, we sample a uniform r_i and compute $u_i = \mathsf{PRG}(s'_i) \oplus r_i \oplus m_i$, while in H_2, we sample a uniform u_i, and then compute $r_i = \mathsf{PRG}(s'_i) \oplus u_i \oplus m_i$. These two ways of sampling are equivalent, and hence no adversary can distinguish between H_1 and H_2 with non-negligible probability. □

Lemma 6.5.5. *No adversary can distinguish between H_2 and H_3 with non-negligible probability.*

Proof. This step is similar to the previous one, another change of variables. In H_2, we sample a uniform s'_i, and compute $c_{2,i} = s'_i \oplus \mathsf{Ext}(c_{1,i}; s_i)$, while in H_2, we sample a uniform $c_{2,i}$ and compute $s'_i = c_{2,i} \oplus \mathsf{Ext}(c_{1,i}; s_i)$. These are equivalent and hence no adversary can distinguish. □

Lemma 6.5.6. *If* $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *is a "good" HILL-entropic encoding with β'-HILL-entropy, and* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ *is a $(t, \alpha, \beta, \epsilon)$-multi-instance randomness extractor with $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$, then no adversary can distinguish between H_3 and H_4 with non-negligible probability.*

Proof. By the β'-HILL-entropy and the goodness of the encoding scheme, $H_\infty(c_{1,i}|\mathsf{crs}_i) \geq \beta'(\lambda, n) = n(1-o(1)) - \mathsf{poly}(\lambda)$. With all the $c_{1,i}$'s combined, we have $H_\infty(\{c_{1,i}\}_i|\{\mathsf{crs}_i\}_i) \geq tn(1-o(1)) - t\mathsf{poly}(\lambda)$. Then, by the fact that $c_{1,i}$'s are sampled independent of the m_i's and lemma 2.1.1, $H_\infty(\{c_{1,i}\}_i|\{\mathsf{crs}_i\}_i, \{m_i\}_i, \mathsf{st}) = H_\infty(\{c_{1,i}\}_i|\{\mathsf{crs}_i\}_i, \mathsf{st}) \geq tn(1-o(1)) - (1-\alpha)nt = \alpha \cdot tn$. Therefore, we can invoke the multi-instance randomness extraction property and have

$$(s_1, \ldots, s_t, \mathsf{Ext}(c_{1,1}; s_1), \ldots, \mathsf{Ext}(c_{1,t}; s_t)) \approx_\epsilon (s_1, \ldots, s_t, Z_1, \ldots, Z_t),$$

where $Z_i = v_i$ for all $i \in I$, and $Z_i = \mathsf{Ext}(c_{1,i}; s_i)$ for all $i \notin I$. Notice that in H_3, we have $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus \mathsf{Ext}(c_{1,i}; s_i)) \oplus m_i$, and in H_4, we equivalently have $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus Z_i) \oplus m_i$. The only difference is that in H_4 we have the Z_i's instead of the $\mathsf{Ext}(c_{1,i}; s_i)$'s in H_3,

and these are indistinguishable by the extractor property. Hence, no adversary can distinguish between H_3 and H_4 with non-negligible probability. □

Lemma 6.5.7. *If* PRG *is a pseudorandom generator secure against non-uniform adversaries, then no PPT adversary can distinguish between H_4 and H_5 with non-negligible probability.*

Proof. First, notice that in H_4, for $i \in I$, we compute $r_i = u_i \oplus \mathsf{PRG}(c_{2,i} \oplus v_i) \oplus m_i$, where v_i is a uniformly random string. This is equivalent as $r_i = u_i \oplus \mathsf{PRG}(v_i') \oplus m_i$ where v_i' is a uniformly random string. So here we are running the PRG on a uniformly random seed. Although we do need to run the *inefficient* simulator for the multi-instance randomness extractor earlier, we can still replace the PRG output with random if the PRG is secure *against non-uniform adversaries*. Hence we have $r_i = u_i \oplus u_i' \oplus m_i$ where u_i' is a uniformly random n-bit string, and this is just equivalent as having a uniformly random r_i, which is the exact case in H_5. Therefore, no PPT adversary can distinguish between H_4 and H_5 with non-negligible probability. □

Theorem 6.5.2. *If* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ *is a* $(t, \alpha, \beta, \epsilon)$*-multi-instance randomness extractor with* $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$, $\mathsf{Code} = (\mathsf{Enc}, \mathsf{Dec})$ *is a "good" HILL-entropic encoding with* β'*-HILL-entropy, and* PRG *is a pseudorandom generator secure against non-uniform adversaries, then Construction 6.5.2 is* $(t, 1, (1-\beta)t, S)$-MULT-SIM-CPA *secure.*

Proof. The lemmas above show a sequence of a polynomial number of hybrid experiments where no PPT adversary can distinguish one from the next with non-negligible probability. Notice that the first hybrid H_0 corresponds to the real mode experiment of multi-user security, and the last hybrid H_5 corresponds to the ideal mode one. The simulation-based security follows. □

6.5.3 Dealing with Multiple Messages per User

Above we have showed MULT-SIM-CPA security for SKE schemes where the number of messages per user ℓ is equal to 1. Here, we show how we can generically lift a SKE scheme with single-message-per-user MULT-SIM-CPA security to multiple-messages-per-user MULT-SIM-CPA security.

Construction 6.5.3. *Let λ, S be security parameters. Given $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ a $(\eta, 1, q, S)$-MULT-SIM-CPA secure SKE with key space $\{0,1\}^n$ [*] and $\mathcal{F}$ a class of ℓ-wise independent functions with range $\{0,1\}^n$, we construct $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows.*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Sample a random function $f \leftarrow \mathcal{F}$. Output $k = f$.*
- $\mathsf{Enc}(k = f, m)$ *: Sample a short random string r with $|r| = \mathsf{polylog}(\ell)$, compute $k' = f(r)$, and get $c \leftarrow \mathsf{SKE.Enc}(k', m)$. Output $\mathsf{ct} = (r, c)$.*
- $\mathsf{Dec}(k = f, \mathsf{ct} = (r, c))$ *: Compute $k' = f(r)$, and output $m \leftarrow \mathsf{SKE.Dec}(k', c)$.*

Correctness should be easy to verify given the correctness of the underlying SKE scheme and the deterministic property of the ℓ-wise independent functions.

Lemma 6.5.8. *If SKE is a $(\eta, 1, q, S)$-MULT-SIM-CPA secure SKE with key space $\{0,1\}^n$ and $\mathcal{F}$ is a class of ℓ-wise independent functions with range $\{0,1\}^n$, then Construction 6.5.3 is $(\eta/\ell, \ell, q, S - \eta \cdot \mathsf{polylog}(\ell))$-MULT-SIM-CPA secure.*

Proof. We prove this through a reduction. We show that if there is an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks the $(\eta/\ell, \ell, q, S - \eta \cdot \mathsf{polylog}(\ell))$-MULT-SIM-CPA security of Π, then we can

[*] Here we assume SKE's keys are uniformly random n-bit strings. This is without loss of generality since we can always take the key to be the random coins for Gen.

construct an adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ that breaks the $(\eta, 1, q, S)$-MULT-SIM-CPA security of SKE. $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ works as follows:

- $\mathcal{A}'_1$: First, run $\mathcal{A}_1$ to obtain a list of message queries $\{m_{i,j}\}_{i\in[\eta/\ell],j\in[\ell]}$. Then, let $m'_i = m_{(i/\ell)+1,((i-1) \bmod \ell)+1}$ for $i \in [\eta]$. Notice that here we are essentially flattening the list of messages. Submit the list $\{m'_i\}_{i\in[\eta]}$ and receive $\{\mathsf{ct}'_i\}_{i\in[\eta]}$. Reconstruct $\mathsf{ct}_{i,j} = (r_{i,j}, \mathsf{ct}'_{(i-1)\cdot\ell+j})$ for $i \in [\eta/\ell]$ and $j \in [\ell]$, where $r_{i,j}$ is a uniformly random string sampled from $\{0,1\}^{\mathsf{polylog}(\ell)}$. Notice that the $r_{i,j}$'s have no collisions under the same i with overwhelming probability. Send the list of ciphertexts $\{\mathsf{ct}_{i,j}\}_{i,j}$ back to $\mathcal{A}_1$ and receive a state st. Output the state $\mathsf{st}' = (\mathsf{st}, \{r_{i,j}\}_{i,j})$. The size of the state is $|\mathsf{st}| + \eta \cdot \mathsf{polylog}(\ell) \leq S - \eta \cdot \mathsf{polylog}(\ell) + \eta \cdot \mathsf{polylog}(\ell) = S$.

- $\mathcal{A}'_2$: First receive $\mathsf{st}' = (\mathsf{st}, \{r_{i,j}\}_{i,j}), \{m'_i\}_{i\in[\eta]}, \{k'_i\}_{i\in[\eta]}$ from the challenger / simulator. Reorganize $m_{i,j} = m'_{(i-1)\cdot\ell+j}$ for $i \in [\eta/\ell]$ and $j \in [\ell]$. Construct k_i as an ℓ-wise independent function f_i s.t. for all $i \in [\eta/\ell]$ and $j \in [\ell], f_i(r_{i,j}) = k'_{(i-1)\cdot\ell+j}$. Send $\mathsf{st}, \{m_{i,j}\}_{i\in[\eta/\ell],j\in[\ell]}, \{k_i = f_i\}_{i\in[\eta/\ell]}$ to $\mathcal{A}_2$ and receive a bit b. Output b.

Notice that $\mathcal{A}'$ perfectly simulates the view for $\mathcal{A}$. If $\mathcal{A}$ says it is in the real mode, this means the ciphertexts are faithful encryptions of the message queries, and hence $\mathcal{A}'$ should be in the real mode as well, and vice versa. Therefore, construction 6.5.3 is $(\eta/\ell, \ell, q, S - \eta \cdot \mathsf{polylog}(\ell))$-MULT-SIM-CPA secure. □

6.6 Public Key Incompressible Encryption

Here we explore multi-user security of incompressible Public Key Encryptions (PKEs), considering constructions from [60,25]. Unlike the SKE setting, where we can generically lift single-

ciphertext-per-user security to multi-ciphertext-per-user security, here we show how to obtain multi-ciphertext security by modifying each construction specifically.

6.6.1 Low Rate Incompressible PKE

For low rate incompressible PKE, we show that the construction from [60] is MULT-SIM-CPA secure by plugging in the multi-instance randomness extractor. Then, we upgrade the construction to have multi-ciphertext-per-user security by upgrading the functionality of the underlying functional encryption scheme.

Construction by [60].

We recall the low rate incompressible PKE construction by [60], with the multi-instance randomness extractor plugged in.

Construction 6.6.1 ([60]). *Given* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *a single-key selectively secure functional encryption scheme and a* $(t, \alpha, \beta, \epsilon)$*-multi-instance randomness extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$*, with* $d = \mathsf{poly}(\lambda)$*,* $w = \mathsf{poly}(\lambda)$ *and* $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$*, the construction* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\{0,1\}^w$ *works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: First, obtain* $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$*. Then, generate the secret key for the following function* f_v *with a hardcoded* $v \in \{0,1\}^{d+w}$*:*

$$f_v(s' = (s, \mathsf{pad}), \mathsf{flag}) = \begin{cases} s' & \text{if } \mathsf{flag} = 0 \\ s' \oplus v & \text{if } \mathsf{flag} = 1 \end{cases}.$$

  Output $\mathsf{pk} = \mathsf{FE.mpk}$ *and* $\mathsf{sk} = \mathsf{FE.sk}_{f_v} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, f_v)$.

- $\mathsf{Enc}(\mathsf{pk}, m)$*: Sample a random tuple* $s' = (s, \mathsf{pad})$ *where* $s \in \{0,1\}^d$ *is used as a seed for the extractor and* $\mathsf{pad} \in \{0,1\}^w$ *is used as a one-time pad. The ciphertext consists of three parts:* $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (s', 0))$*, a long randomness* $R \in \{0,1\}^n$*, and* $z = \mathsf{Ext}(R; s) \oplus \mathsf{pad} \oplus m$*.*

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct} = (\mathsf{FE.ct}, R, z))$*: First, obtain* $s' \leftarrow \mathsf{FE.Dec}(\mathsf{FE.sk}_{f_v}, \mathsf{FE.ct})$*, and then use the seed* s *to compute* $\mathsf{Ext}(R; s) \oplus z \oplus \mathsf{pad}$ *to recover* m*.*

The correctness follows from the original construction.

Theorem 6.6.1. *If* FE *is a single-key selectively secure functional encryption scheme and* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ *is a* $(t, \alpha, \beta, \epsilon)$*-multi-instance randomness extractor with* $d, w = \mathsf{poly}(\lambda)$ *and* $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$*, then Construction 6.6.1 is* $(t, 1, (1-\beta)t, S)$-MULT-SIM-CPA *secure.*

We prove Theorem 6.6.1 through a sequence of hybrids, starting with H_0 being the real mode experiment and ending with H_3 being the ideal mode experiment. The proofs of the hybrid arguments are identical to those from[60] (except for the extractor step, which is analogous to the proof of Lemma 6.5.2), so we will not reproduce them here and instead point the reader to the original[60] paper.

Sequence of Hybrids

- Hybrid H_0:
  - For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$ and sample a uniform random $v_i \leftarrow \{0,1\}^{d+w}$. Set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ and $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i})$.

- Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.
- For each $i \in [t]$:
  * Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
  * Sample $\mathsf{pad}_i \leftarrow \{0,1\}^w$ uniformly at random.
  * Let $s'_i = (s_i, \mathsf{pad}_i)$.
  * Let $\mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (s'_i, 0))$.
  * Sample $R_i \leftarrow \{0,1\}^n$ uniformly at random.
  * Let $z_i = \mathsf{Ext}(R_i; s_i) \oplus \mathsf{pad}_i \oplus m_i$.
  * Let $\mathsf{ct}_i = (\mathsf{FE.ct}_i, R_i, z_i)$.
- Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
- On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_1:
  - For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$ and sample a uniform random $v_i \leftarrow \{0,1\}^{d+w}$. Set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ and $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i})$.
  - Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$:
    * Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
    * Sample $\mathsf{pad}_i \leftarrow \{0,1\}^w$ uniformly at random.
    * Let $s'_i = (s_i, \mathsf{pad}_i)$.

* Let $\mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (s'_i \oplus v_i, 1))$.
* Sample $R_i \leftarrow \{0,1\}^n$ uniformly at random.
* Let $z_i = \mathsf{Ext}(R_i; s_i) \oplus \mathsf{pad}_i \oplus m_i$.
* Let $\mathsf{ct}_i = (\mathsf{FE.ct}_i, R_i, z_i)$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_2:

– For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.

Set only $\mathsf{pk}_i = \mathsf{FE.mpk}_i$.

– Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.

– For each $i \in [t]$:

* Sample $u_i \leftarrow \{0,1\}^{d+w}$ uniformly at random.
* Let $\mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (u_i, 1))$.
* Sample $R_i \leftarrow \{0,1\}^n$ uniformly at random.
* Sample $z_i \leftarrow \{0,1\}^w$ uniformly at random.
* Let $\mathsf{ct}_i = (\mathsf{FE.ct}_i, R_i, z_i)$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– For each $i \in [t]$:

* Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
* Let $\mathsf{pad}_i = \mathsf{Ext}(R_i; s_i) \oplus z_i \oplus m_i$.

* Let $s'_i = (s_i, \mathsf{pad}_i)$ and compute $v_i = s'_i \oplus u_i$.
* Obtain $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i})$.

– On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_3:

– For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Set only $\mathsf{pk}_i = \mathsf{FE.mpk}_i$.

– Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$. Discard $\{m_i\}_i$ without looking at it.

– For each $i \in [t]$:

* Sample $u_i \leftarrow \{0,1\}^{d+w}$ uniformly at random.
* Let $\mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (u_i, 1))$.
* Sample $R_i \leftarrow \{0,1\}^n$ uniformly at random.
* Sample $z_i \leftarrow \{0,1\}^w$ uniformly at random.
* Let $\mathsf{ct}_i = (\mathsf{FE.ct}_i, R_i, z_i)$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– Run the simulator for the multi-instance randomness extractor to get a set of indices $I \subseteq [t]$ with $|I| \geq \beta t$. Submit the set $[t] \backslash I$, and receive the corresponding messages $\{m_i\}_{i \notin I}$. For each $i \in [t]$:

* Sample $s_i \leftarrow \{0,1\}^d$ uniformly at random.
* If $i \in I$, sample $\mathsf{pad}_i \leftarrow \{0,1\}^w$ uniformly at random.
* If $i \notin I$, let $\mathsf{pad}_i = \mathsf{Ext}(R_i; s_i) \oplus z_i \oplus m_i$.

* Let $s'_i = (s_i, \mathsf{pad}_i)$ and compute $v_i = s'_i \oplus u_i$.
* Obtain $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i})$.

– On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

Upgrading to Multiple Ciphertexts Per User.

Additionally, We show that the constructions from [60] can be upgraded to have multi-ciphertext-per-user security. Essentially, all we need is to upgrade the functionality of the underlying functional encryption scheme to work for a slightly more generalized class of functions. We will need functions $f_{\{v_i\}_i}(s, \mathsf{flag}) = s \oplus v_{\mathsf{flag}}$ for hard coded values $v_1, \ldots, v_\ell$ and a special v_0 being the all 0 string. Notice that the original GWZ construction [60] can be viewed as using functions that are a special case where $\ell = 1$. We show how to construct FE schemes for such $f_{\{v_i\}_i}$ functions from plain PKE below. With this new class of functions, we can achieve $(t, \ell, (1-\beta)\ell t, \mathcal{S})$-MULT-SIM-CPA security. In the hybrid proof where we replace $\mathsf{FE.Enc}(\mathsf{FE.mpk}, (s', 0))$ with $\mathsf{FE.Enc}(\mathsf{FE.mpk}, (s' \oplus v, 1))$, now for the j-th message query for the i-th user where $i \in [t]$ and $j \in [\ell]$, we replace $\mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (s'_{i,j}, 0))$ with $\mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (s'_{i,j} \oplus v_{i,j}, j))$. The rest of the hybrid proof follows analogously.

Instantiating FE for $f_{\{v_i\}_i}$ Functions.

Here we show how to construct the FE scheme for the new class of functions that we need to upgrade construction 6.6.1 to have multi-ciphertext-per-user security. We only need plain PKE for the construction. Recall that our functions $f_{\{v_i\}_i}$ have the form $f_{\{v_i\}_i}(s, \mathsf{flag}) = s \oplus v_{\mathsf{flag}}$, where $\mathsf{flag} \in \{0, 1, \ldots, \ell\}$.

Construction 6.6.2. *Let* $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *be a public key encryption scheme. Our scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for a single message bit s is defined as:*

- $\mathsf{Setup}(1^\lambda)$*: For* $i \in \{0, 1, \ldots, n\}, b \in \{0, 1\}$*, run* $(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) \leftarrow \mathsf{Gen}'(1^\lambda)$*. Output* $(\mathsf{mpk} = \{\mathsf{pk}_{i,b}\}_{i,b}\ ,\ \mathsf{msk} = \{\mathsf{sk}_{i,b}\}_{i,b})$.

- $\mathsf{KeyGen}(\mathsf{msk}, f_{\{v_i\}_i}) = \{\mathsf{sk}_{i,v_i}\}_i$*. Notice that we hardcode* $v_0 = 0$.

- $\mathsf{Enc}(\mathsf{mpk}, (s, \mathsf{flag}))$*: Sample uniformly random bits* $s^{(0)}, s^{(1)}, \ldots, s^{(n)}$ *s.t.* $s^{(0)} \oplus s^{(1)} \oplus \cdots \oplus s^{(n)} = s$*. For* $i \in \{0, 1, \ldots, n\} \setminus \{\mathsf{flag}\}, b \in \{0, 1\}$*, compute* $c_{i,b} = \mathsf{Enc}'(\mathsf{pk}_{i,b}, s^{(i)})$*. For* $b \in \{0, 1\}$*, compute* $c_{\mathsf{flag},b} = \mathsf{Enc}'(\mathsf{pk}_{i,b}, s^{(\mathsf{flag})} \oplus b)$*. Output* $c = (c_{i,b})_{i,b}$.

- $\mathsf{Dec}(\mathsf{sk}_{\{v_i\}_i}, c)$*: Output* $x = x^{(0)} \oplus x^{(1)} \oplus \cdots \oplus x^{(n)}$ *where* $x^{(i)} = \mathsf{Dec}'(\mathsf{sk}_{i,v_i}, c_{i,v_i})$

For correctness, note that for $i \neq \mathsf{flag}$, $x^{(i)} = s^{(i)}$, and that $x^{(\mathsf{flag})} = s^{(\mathsf{flag})} \oplus v_{\mathsf{flag}}$, therefore $x = s^{(0)} \oplus s^{(1)} \oplus \cdots \oplus s^{(n)} \oplus v_{\mathsf{flag}} = s \oplus v_{\mathsf{flag}}$.

Lemma 6.6.1. *If* $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *is a CPA secure public key encryption scheme, then Construction 6.6.2 is single key semi-adaptively secure for the functions* $f_{\{v_i\}_i}$.

Proof. Consider a single key semi-adaptive adversary for Construction 6.6.2. Let $m_0 = (s_0, \mathsf{flag}_0), m_1 = (s_1, \mathsf{flag}_1)$ be the challenge messages. For a fixed flag, $f_{\{v_i\}_i}$ is injective. Therefore, if $m_0 \neq m_1$, it must be that $\mathsf{flag}_0 \neq \mathsf{flag}_1$. Then if the adversary's secret key query is on $f_{\{v_i\}_i}$, we must have $s_0 \oplus v_{\mathsf{flag}_0} = s_1 \oplus v_{\mathsf{flag}_1}$. Therefore the c_{i,v_i}'s always encrypt an instance of a secret share of the same value $s_0 \oplus v_{\mathsf{flag}_0} = s_1 \oplus v_{\mathsf{flag}_1}$. Hence, for $i \notin \{\mathsf{flag}_0, \mathsf{flag}_1\}$, $\hat{b} \in \{0, 1\}$, $c_{i,\hat{b}}$'s follow the same distribution in both cases and do not depend on the challenge bit b. The only dependence on the challenge bit b is that $c_{\mathsf{flag}_b,0}$ always encrypts the opposite bit that $c_{\mathsf{flag}_b,1}$ encrypts, whereas $c_{\mathsf{flag}_{1-b},0}$ and $c_{\mathsf{flag}_{1-b},1}$ always encrypt the same bit.

However, since the adversary never gets to see the secret key $\mathsf{sk}_{\mathsf{flag}_b, 1-v_{\mathsf{flag}_b}}$, a simple hybrid argument shows that flipping the challenge bit is indistinguishable. □

6.6.2 Rate-1 Incompressible PKE

For rate-1 incompressible PKE, we first show that we can easily plug in the multi-instance randomness extractor to the construction by Guan, Wichs and Zhandry[60]. We also provide a generalization on the construction by Branco, Döttling and Dujmovic[25] using a Key Encapsulation Mechanism (KEM) with a special *non-committing* property. For both constructions, we show how to adapt them to allow for multi-ciphertext-per-user security.

Construction by Guan et al.[60].

We first reproduce the rate-1 PKE construction from[60], with the multi-instance randomness extractors plugged in.

Construction 6.6.3 ([60]). *Given* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *a rate-1 functional encryption scheme satisfying single-key semi-adaptive security,* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ *a* $(t, \alpha, \beta, \epsilon)$*-multi-instance randomness extractor with* $d, w = \mathsf{poly}(\lambda), n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$ *and* $\mathsf{PRG} : \{0,1\}^w \to \{0,1\}^n$ *a secure PRG against non-uniform adversaries, the construction* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for message space* $\{0,1\}^n$ *works as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: First, obtain* $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$*. Then, generate the secret key for the following function* $f_{v,s}$ *with a hardcoded large random pad* $v \in \{0,1\}^n$

and a small extractor seed $s \in \{0,1\}^d$*:*

$$f_{v,s}(x, \mathsf{flag}) = \begin{cases} x & \text{if } \mathsf{flag} = 0 \\ \mathsf{PRG}(\mathsf{Extract}(x; s)) \oplus v & \text{if } \mathsf{flag} = 1 \end{cases}.$$

Output $\mathsf{pk} = \mathsf{FE.mpk}$ *and* $\mathsf{sk} = \mathsf{FE.sk}_{f_{v,s}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, f_{v,s})$.

- $\mathsf{Enc}(\mathsf{pk}, m)$*: The ciphertext is simply an encryption of* $(m, 0)$ *using the underlying* FE *scheme, i.e.* $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (m, 0))$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Decryption also corresponds to* FE *decryption. The output is* $\mathsf{FE.Dec}(\mathsf{FE.sk}_{f_{v,s}}, \mathsf{ct}) = f_{v,s}(m, 0) = m$ *as desired.*

Correctness easily follows from the original construction. The rate of the construction is the rate of the underlying FE multiplied by $\frac{n}{n+1}$. If the FE has rate $(1-o(1))$, the construction has rate $(1 - o(1))$ as desired.

Theorem 6.6.2. *If* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a single-key semi-adaptively secure functional encryption scheme,* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^w$ *is a* $(t, \alpha, \beta, \epsilon)$*-multi-instance randomness extractor, with* $d, w = \mathsf{poly}(\lambda)$ *and* $n = \frac{S}{(1-\alpha)t} + \mathsf{poly}(\lambda)$*, and* $\mathsf{PRG} : \{0,1\}^w \to \{0,1\}^n$ *is a PRG secure against non-uniform adversaries, then Construction 6.6.3 is* $(t, 1, (1-\beta)t, S)$-MULT-SIM-CPA *secure.*

We prove Theorem 6.6.2 through a sequence of hybrids, starting with H_0 being the real mode experiment where we play the role of the challenger and ending with H_6 being the ideal mode experiment where we play the role of the simulator. For the proofs of each hybrid argument, see the original[60] paper, since they are identical except for the extractor step

(analogous to Lemma 6.5.2) and the PRG against non-uniform attackers step (analogous to Lemma 6.5.7).

Sequence of Hybrids

- Hybrid H_0:
  - For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$ and sample uniformly random $v_i \leftarrow \{0,1\}^n$ and $s_i \leftarrow \{0,1\}^d$. Set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ and $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.
  - Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$, let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (m_i, 0))$.
  - Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
  - On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.
- Hybrid H_1:
  - For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
    Only set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ for now.
  - Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$, let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (m_i, 0))$.
  - Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
  - For each $i \in [t]$:
    - Sample a uniformly random $s_i \leftarrow \{0,1\}^d$.

* Sample a uniformly random $u_i \leftarrow \{0,1\}^n$, and let $v_i = u_i \oplus m_i$.
* Let $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.

– On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_2:

– For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Only set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ for now.

– Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.

– For each $i \in [t]$, let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (m_i, 0))$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– For each $i \in [t]$:

* Sample a uniformly random $s_i \leftarrow \{0,1\}^d$.
* Sample a uniformly random PRG key $k_i \leftarrow \{0,1\}^w$, and let $v_i = \mathsf{PRG}(k_i) \oplus m_i$.
* Let $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.

– On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_3:

– For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Only set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ for now.

– Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.

– For each $i \in [t]$, sample a uniformly random $R_i \leftarrow \{0,1\}^n$, and let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (m_i, 0))$.

- Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
- For each $i \in [t]$:
  * Sample a uniformly random $s_i \leftarrow \{0,1\}^d$.
  * Let $k_i = \mathsf{Ext}(R_i; s_i)$, and let $v_i = \mathsf{PRG}(k_i) \oplus m_i$.
  * Let $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.
- On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_4:
  - For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Only set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ for now.
  - Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.
  - For each $i \in [t]$, sample a uniformly random $R_i \leftarrow \{0,1\}^n$, and let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (R_i, 1))$.
  - Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
  - For each $i \in [t]$:
    * Sample a uniformly random $s_i \leftarrow \{0,1\}^d$.
    * Let $k_i = \mathsf{Ext}(R_i; s_i)$, and let $v_i = \mathsf{PRG}(k_i) \oplus m_i$.
    * Let $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.
  - On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_5:

- For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Only set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ for now.
- Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$.
- For each $i \in [t]$, sample a uniformly random $R_i \leftarrow \{0,1\}^n$, and let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (R_i, 0))$.
- Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.
- Run the simulator for the multi-instance randomness extractor to get a set of indices $I \subseteq [t]$ with $|I| \geq \beta t$. For each $i \in [t]$:
  - Sample a uniformly random $s_i \leftarrow \{0,1\}^d$.
  - If $i \in I$, sample a uniformly random PRG key $k_i \leftarrow \{0,1\}^w$, and let $v_i = \mathsf{PRG}(k_i) \oplus m_i$.
  - If $i \notin I$, let $k_i = \mathsf{Ext}(R_i; s_i)$, and let $v_i = \mathsf{PRG}(k_i) \oplus m_i$.
  - Let $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.
- On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

- Hybrid H_6:
  - For each $i \in [t]$, obtain $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Only set $\mathsf{pk}_i = \mathsf{FE.mpk}_i$ for now.
  - Send $\{\mathsf{pk}_i\}_i$ to the adversary $\mathcal{A}_1$ and receive $\{m_i\}_i$ for $i \in [t]$. Discard $\{m_i\}_i$ without looking at it.
  - For each $i \in [t]$, sample a uniformly random $R_i \leftarrow \{0,1\}^n$, and let $\mathsf{ct}_i = \mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, (R_i, 0))$.

– Send $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ and receive a state st.

– Run the simulator for the multi-instance randomness extractor to get a set of indices $I \subseteq [t]$ with $|I| \geq \beta t$. Submit the set $[t] \setminus I$, and receive the corresponding messages $\{m_i\}_{i \notin I}$. For each $i \in [t]$:

  * Sample a uniformly random $s_i \leftarrow \{0,1\}^d$.
  * If $i \in I$, sample a uniformly random $v_i \leftarrow \{0,1\}^n$.
  * If $i \notin I$, let $k_i = \mathsf{Ext}(R_i; s_i)$, and let $v_i = \mathsf{PRG}(k_i) \oplus m_i$.
  * Let $\mathsf{sk}_i = \mathsf{FE.sk}_{f_{v_i,s_i}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, f_{v_i,s_i})$.

– On input of st, $\{m_i\}_i$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

Upgrading to Multiple Ciphertexts Per User.

Upgrading Construction 6.6.3 to multi-ciphertext-per-user security is rather straightforward. Since the construction already requires a full functionality FE scheme, we just modify the class of functions that the underlying FE scheme uses, without introducing any new assumptions. Specifically, we use the class of functions $f_{\{v_j\}_j,\{s_j\}_j}$ with hard-coded values $v_j \in \{0,1\}^n$ and $s_j \in \{0,1\}^d$ for $j \in [\ell]$ that behaves as follows:

$$f_{\{v_j\}_j,\{s_j\}_j}(x, \mathsf{flag}) = \begin{cases} x & \text{if } \mathsf{flag} = 0 \\ \mathsf{PRG}(\mathsf{Extract}(x; s_{\mathsf{flag}})) \oplus v_{\mathsf{flag}} & \text{if } \mathsf{flag} \in [\ell] \end{cases}.$$

This gives us $(t, \ell, (1-\alpha)\ell t, S)$-MULT-SIM-CPA security. Notice that this modification does slightly harm the rate of the scheme, since the flag is now $\log(\ell)$ bits instead of one bit, but asymptotically the rate is still $(1 - o(1))$.

The hybrid proof works analogously to that of Theorem 6.6.2, except that in the hybrid proof where we swap the FE encryption of $(m, 0)$ to $(R, 1)$, we now swap from $(m_{i,j}, 0)$ to $(R_{i,j}, j)$ for the j-th ciphertext from the i-th user.

Generalization of Construction by Branco et al.[25].

Branco et al.[25] show how to lift a rate-1 incompressible SKE scheme to a rate-1 incompressible PKE scheme using a Key Encapsulation Mechanism[33] built from programmable Hash Proof Systems (HPS)[32,72]. Their construction satisfy CCA2 security. We show that if we are to relax the security notion to only CPA security, all we need for the lifting is a Key Encapsulation Mechanism with a *non-committing* property, defined as follows.

Definition 6.6.1 (Key Encapsulation Mechanism[33])**.** *Let λ be the security parameters, a Key Encapsulation Mechanism (KEM) is a tuple of algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ that works as follow:*

- $\mathsf{KeyGen}(1^\lambda, 1^{\mathcal{L}_k}) \to (\mathsf{pk}, \mathsf{sk})$*: The key generation algorithm takes as input the security parameter and the desired symmetric key length $\mathcal{L}_k$, outputs a pair of public key and private key* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encap}(\mathsf{pk}) \to (k, c)$*: The encapsulation algorithm takes the public key* pk*, produces a symmetric key $k \in \{0,1\}^{\mathcal{L}_k}$, and a header c that encapsulates k.*
- $\mathsf{Decap}(\mathsf{sk}, c) \to k$*: The decapsulation algorithm takes as input the private key* sk *and a header c, and decapsulates the header to get the symmetric key k.*

We require *correctness* of the KEM.

Definition 6.6.2 (Correctness)**.** *A key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ *is said to be correct if:*

$$\Pr\left[k' = k : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\mathcal{L}_k}) \\ (k, c) \leftarrow \mathsf{Encap}(\mathsf{pk}) \\ k' \leftarrow \mathsf{Decap}(\mathsf{sk}, c) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Definition 6.6.3 (Non-Committing)**.** *A key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ *is said to be* non-committing *if there exists a pair of simulator algorithm* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that* $\mathsf{Sim}_1(1^\lambda, 1^{\mathcal{L}_k})$ *outputs a simulated public key* pk'*, a header* c' *and a state* st *with* $|\mathsf{st}| = \mathsf{poly}(\lambda, \mathcal{L}_k)$*, and for any given target key* $k' \in \{0,1\}^{\mathcal{L}_k}$*,* $\mathsf{Sim}_2(\mathsf{st}, k')$ *outputs the random coins* r^{KeyGen} *and* r^{Encap}*. We require that if we run the key generation and encapsulation algorithm using these random coins, we will get the desired* pk', c'*, and* k'*, i.e.:*

$$\Pr\left[\begin{array}{l} \mathsf{pk}' = \mathsf{pk} \\ k' = k \\ c' = c \end{array} : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\mathcal{L}_k}; r^{\mathsf{KeyGen}}) \\ (k, c) \leftarrow \mathsf{Encap}(\mathsf{pk}; r^{\mathsf{Encap}}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Kindly notice that by the correctness property, $\mathsf{Decap}(\mathsf{sk}, c') \to k'$*.*

This *non-committing* property allows us to commit to a public key and header first, but then later able to reveal it as an encapsulation of an arbitrary symmetric key in the key space. And it will be impossible to distinguish the simulated public key and header from the ones we get from faithfully running KeyGen and Encap.

Using this non-committing KEM, we are able to construct rate-1 incompressible PKE from rate-1 incompressible SKE, with multi-user security in mind. This is a generalization

of the construction by[25].

Construction 6.6.4 (Generalization of[25])**.** *Let λ, S be security parameters. Given* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ *a non-commiting KEM and* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *a rate-1 incompressible SKE for message space* $\{0,1\}^n$*, we construct rate-1 incompressible PKE* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for message space* $\{0,1\}^n$ *as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: First, run* $\mathsf{SKE.Gen}(1^\lambda, 1^S)$ *to determine the required symmetric key length* $\mathcal{L}_k$ *under security parameters* λ, S*. Then run* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.KeyGen}(1^\lambda, 1^{\mathcal{L}_k})$ *and output* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: First, run* $(k, c_0) \leftarrow \mathsf{KEM.Encap}(\mathsf{pk})$ *to sample a symmetric key* k*, and encapsulate it into a header* c_0*. Then compute* $c_1 \leftarrow \mathsf{SKE.Enc}(k, m)$*. The ciphertext is the tuple* (c_0, c_1)*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct} = (c_0, c_1))$*: First, decapsulate* c_0 *using* sk *to obtain* $k \leftarrow \mathsf{KEM.Decap}(\mathsf{sk}, c_0)$*, and then use* k *to decrypt* c_1 *and get* $m \leftarrow \mathsf{SKE.Dec}(k, c_1)$*.*

Correctness follows from the correctness of the underlying incompressible SKE and the KEM scheme. In terms of the rate, to achieve a rate-1 incompressible PKE, we would require the KEM to produce "short" headers, i.e. $|c_0| = \mathsf{poly}(\lambda)$ independent of $\mathcal{L}_k$ (notice that $\mathcal{L}_k = \mathsf{poly}(\lambda, n)$ and needs to be at least as large as n). We can build such KEMs using various efficient encapsulation techniques[10,2,16]. With the short header and an incompressible SKE with rate $(1 - o(1))$, the ciphertext length is $n/(1 - o(1)) + \mathsf{poly}(\lambda)$, yielding an ideal rate of $(1 - o(1))$ for the construction. However, these KEMs require long public keys, as opposed to the short public keys in Construction 6.6.3.

For security, we prove that if the underlying SKE has MULT-SIM-CPA security, then Construction 6.6.4 has MULT-SIM-CPA security as well.

Theorem 6.6.3. *If* KEM *is a non-commiting KEM, and* SKE *is a* $(\eta, 1, q, S)$-MULT-SIM-CPA *secure SKE with message space* $\{0,1\}^n$*, then Construction 6.6.4 is* $(\eta, 1, q, S - \eta \cdot \mathsf{poly}(\lambda, n))$-MULT-SIM-CPA *secure.*

Proof. We prove this through a reduction. We show that if there is an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks the $(\eta, 1, q, S - \eta \cdot \mathsf{poly}(\lambda, n))$-MULT-SIM-CPA security of Π, then we can construct an adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ that breaks the $(\eta, 1, q, S)$-MULT-SIM-CPA security of SKE. $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ works as follows:

- $\mathcal{A}'_1$: Use the security parameters λ, S to determine the key length $\mathcal{L}_k$ for the underlying SKE*. For each $i \in [\eta]$, obtain $(\mathsf{pk}_i, c_{0,i}, \mathsf{KEM.st}_i) \leftarrow \mathsf{KEM.Sim}_1(1^\lambda, 1^{\mathcal{L}_k})$. Send $\{\mathsf{pk}_i\}_i$ to $\mathcal{A}_1$ to get a list of message queries $\{m_i\}_i$. Then, forward the list $\{m_i\}_i$ to the challenger / simulator and receive a list of ciphertexts $\{\mathsf{ct}'_i\}_i$. Construct $\mathsf{ct}_i = (c_{0,i}, \mathsf{ct}'_i)$, and send all $\{\mathsf{ct}_i\}_i$ to $\mathcal{A}_1$ to receive a state st. Output the state $\mathsf{st}' = (\mathsf{st}, \{\mathsf{KEM.st}_i\}_i)$. The size of the state is $|\mathsf{st}| + \eta \cdot \mathsf{poly}(\lambda, \mathcal{L}_k) \leq S - \eta \cdot \mathsf{poly}(\lambda, n) + \eta \cdot \mathsf{poly}(\lambda, n) = S$.

- $\mathcal{A}'_2$: First receive $\mathsf{st}' = (\mathsf{st}, \{\mathsf{KEM.st}_i\}_i), \{m_i\}_i, \{k_i\}_i$ from the challenger / simulator. For each $i \in [\eta]$, run $(r_i^{\mathsf{KeyGen}}, r_i^{\mathsf{Encap}}) \leftarrow \mathsf{KEM.Sim}_2(\mathsf{KEM.st}_i, k_i)$, and $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\lambda, 1^{\mathcal{L}_k}; r_i^{\mathsf{KeyGen}})$. Notice that pk_i matches the pk_i produced previously by $\mathcal{A}'_1$ due to the non-committing property of the KEM. Send $\mathsf{st}, \{m_i\}_i, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$ to $\mathcal{A}_2$ and receive a bit b. Output b.

*For the ease of syntax, we imagine the security parameters to be part of the public parameters always accessible to the adversary.

Notice that $\mathcal{A}'$ perfectly simulates the view for $\mathcal{A}$. If $\mathcal{A}$ says it is in the real mode interacting with the challenger, this means the ciphertexts ct_i's are faithful encryptions of the message queries m_i's, i.e. $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_i) = \mathsf{SKE.Dec}(\mathsf{KEM.Decap}(\mathsf{sk}_i, c_{0,i}), \mathsf{ct}'_i) = m_i$ for all $i \in [\eta]$. This implies that $\mathsf{SKE.Dec}(k_i, \mathsf{ct}'_i) = m_i$, and hence $\mathcal{A}'$ is also in the real mode. The converse also holds true. Therefore, construction 6.6.4 is $(\eta, 1, q, S - \eta \cdot \mathsf{poly}(\lambda, n))$-MULT-SIM-CPA secure. □

Upgrading to Multiple Ciphertexts Per User.

Next we show how to upgrade Construction 6.6.4 to have multi-ciphertext-per-user security. All we need is to upgrade the KEM to be ℓ-*strongly non-committing*, defined as below.

Definition 6.6.4 (ℓ-Strongly Non-Committing)**.** *A key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ *is said to be* ℓ-strongly non-committing *if there exists a pair of simulator algorithm* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that* $\mathsf{Sim}_1(1^\lambda, 1^{\mathcal{L}_k})$ *outputs a simulated public key* pk'*, a set of simulated headers* $\mathcal{C}' = \{c'_1, c'_2, \ldots, c'_\ell\}$ *and a state* st *with* $|\mathsf{st}| = \mathsf{poly}(\lambda, \mathcal{L}_k, \ell)$*, and for any given set of target keys* $\mathcal{K}' = \{k'_1, k'_2, \ldots, k'_\ell\}$ *where* $k'_i \in \{0,1\}^{\mathcal{L}_k}$ *for all* $i \in [\ell]$*,* $\mathsf{Sim}_2(\mathsf{st}, \mathcal{K}')$ *outputs a set of random coin pairs* $\{(r_i^{\mathsf{KeyGen}}, r_i^{\mathsf{Encap}})\}_{i\in[\ell]}$*. We require that if we run the key generation and encapsulation algorithm using the i-th pair of these random coins, we will get the desired* pk'*,* c'_i*, and* k'_i*, i.e. for all* $i \in [\ell]$*:*

$$\Pr\left[\begin{array}{l|c} \mathsf{pk}' = \mathsf{pk} & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\mathcal{L}_k}; r_i^{\mathsf{KeyGen}}) \\ k'_i = k & : \\ c'_i = c & (k, c) \leftarrow \mathsf{Encap}(\mathsf{pk}; r_i^{\mathsf{Encap}}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Kindly notice that by the correctness property, $\mathsf{Decap}(\mathsf{sk}, c'_i) \to k'_i$.

We show how to construct ℓ-strongly non-committing KEMs by composing plain non-committing KEMs below.

To get multi-ciphertext security, we simply plug in the ℓ-strongly non-committing KEM in place of the plain non-committing KEM in construction 6.6.4. The resulting construction has $(\eta/\ell, \ell, q, S - \eta \cdot \mathsf{poly}(\lambda, n, \ell))$-MULT-SIM-CPA security. The security proof follows analogous from that of Theorem 6.6.3.

Instantiating ℓ-Strongly Non-Committing KEM.

We give a simple construction of ℓ-strongly non-committing KEM by composing 2ℓ plain non-committing KEMs.

Construction 6.6.5. *Let* $\mathsf{KEM}_1, \mathsf{KEM}_2, \ldots, \mathsf{KEM}_n$ *be* $n = 2\ell$ *instances of non-committing KEMs, we construct an* ℓ*-strongly non-committing KEM* $\Pi = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ *as follows:*

- $\mathsf{KeyGen}(1^\lambda, 1^{\mathcal{L}_k})$*: For each* $i \in [n]$*, run* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM}_i.\mathsf{KeyGen}(1^\lambda, 1^{\mathcal{L}_k})$*. Publish* $\mathsf{pk} = \{\mathsf{pk}_i\}_i$ *and* $\mathsf{sk} = \{\mathsf{sk}_i\}_i$*.*
- $\mathsf{Encap}(\mathsf{pk})$*: First sample a random subset* $I \subseteq [n]$*. Then for all* $i \in I$*, get* $(k_i, c_i) \leftarrow \mathsf{KEM}_i.\mathsf{Encap}(\mathsf{pk}_i)$*. Output* $k = \bigoplus_{i \in I} k_i$*, and* $c = (I, \{c_i\}_i)$*.*
- $\mathsf{Decap}(\mathsf{sk}, c)$*: First parse* $c = (I, \{c_i\}_i)$*. Then for all* $i \in I$*, get* $k_i \leftarrow \mathsf{KEM}_i.\mathsf{Decap}(\mathsf{sk}_i, c_i)$*. Output* $k = \bigoplus_{i \in I} k_i$*.*

Correctness is trivial given the correctness of the underlying KEMs. The public key, private key and header sizes all blow up by a factor of n.

Lemma 6.6.2. *If* $\mathsf{KEM}_1, \mathsf{KEM}_2, \ldots, \mathsf{KEM}_n$ *are non-committing KEMs, then construction 6.6.5 is* ℓ*-strongly non-committing.*

Proof. We show how to construct the pair of simulator algorithms $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ for Π:

- $\mathsf{Sim}_1(1^\lambda, 1^{\mathcal{L}_k})$: For all $i \in [n]$, get $(\mathsf{pk}'_i, c'_i, \mathsf{st}_i) \leftarrow \mathsf{KEM}_i.\mathsf{Sim}_1(1^\lambda, 1^{\mathcal{L}_k})$. For all $j \in [\ell]$, sample a random subset $I_j \subseteq [n]$, and let $\hat{c}_j = (I_j, \{c'_i\}_{i \in I_j})$. Output $\mathsf{pk}' = \{\mathsf{pk}'_i\}_{i \in [n]}, \mathcal{C}' = \{\hat{c}_j\}_{j \in [\ell]}$, and $\mathsf{st} = (\{I_j\}_{j \in [\ell]}, \{\mathsf{st}_i\}_{i \in [n]})$.

- $\mathsf{Sim}_2(\mathsf{st}, \mathcal{K}' = \{k'_j\}_{j \in [\ell]})$: First, parse $\{I_j\}_{j \in [\ell]}$ as a $\ell \times n$ bit matrix $\mathbf{M}$. $\mathbf{M}_{\alpha,\beta} = 1$ if and only if $\beta \in I_\alpha$. Solve for the vector $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ where each $\mathbf{v}_i \in \{0,1\}^{\mathcal{L}_k}$ such that

  $$\mathbf{M} \cdot \mathbf{v}^\top = (k'_1, k'_2, \ldots, k'_\ell)^\top.$$

  Assume for now that there exists a satisfying solution for $\mathbf{v}$. Notice that this means for all $j \in [\ell], k'_j = \bigoplus_{i \in I_j} \mathbf{v}_i$, i.e. $\mathbf{v}$ gives an assignment of keys for $\mathsf{KEM}_1, \mathsf{KEM}_2, \ldots, \mathsf{KEM}_n$ that satisfies the target key set $\mathcal{K}'$ for Π. Now we just to run $(r_i^{\mathsf{KeyGen}}, r_i^{\mathsf{Encap}}) \leftarrow \mathsf{KEM}_i.\mathsf{Sim}_2(\mathsf{st}_i, \mathbf{v}_j)$ for all $i \in [n]$. Output $\{(r_i^{\mathsf{KeyGen}}, r_i^{\mathsf{Encap}})\}_{i \in [n]}$.

By the non-committing property of the underlying KEMs, it is easy to see that these random coins yield the simulated public keys, headers and the target keys.

The only remaining thing to show is that $\mathbf{M} \cdot \mathbf{v}^\top = (k'_1, k'_2, \ldots, k'_\ell)^\top$ has a satisfying solution for $\mathbf{v}$. Notice that $\mathbf{v}$ has at least one solution if $\mathbf{M}$ has rank ℓ. Having rank ℓ essentially says that all the rows of $\mathbf{M}$ are linearly independent. This ensures that the linear equation system generated by $\mathbf{M} \cdot \mathbf{v}^\top = (k'_1, k'_2, \ldots, k'_\ell)^\top$ is consistent. Notice having rank ℓ also means that $\mathbf{M}$ has at least ℓ non-zero columns. This gives us a consistent linear equation system with ℓ equations and at least ℓ variables, which is guaranteed to have a (not necessarily

unique) solution. Notice that if we choose $n = 2\ell$, then $\mathbf{M}$ is an $\ell \times 2\ell$ matrix, which has full rank (rank ℓ) with overwhelming probability $1 - O(2^{-\ell})$. □

6.7 Incompressible Encryption in the Random Oracle Model

6.7.1 Rate-1 Incompressible SKE from Random Oracles

We show how to build rate-1 incompressible SKE in the random oracle model.

Construction 6.7.1. *Let λ, S be security parameters. Given $G : \{0,1\}^{\mathsf{poly}(\lambda)} \times \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}^n, H : \{0,1\}^{\mathsf{poly}(\lambda)} \times \{0,1\}^n \to \{0,1\}^{\mathsf{poly}(\lambda)}$ two hash functions modelled as random oracles, we construct $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for message space $\{0,1\}^n$ as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Sample a uniformly random key $k \in \{0,1\}^{\mathsf{poly}(\lambda)}$. Output k.*
- $\mathsf{Enc}(k, m)$*: First, choose a random $r \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$. Let $d = G(k, r) \oplus m$. Then let $c = H(k, d) \oplus r$. Output $\mathsf{ct} = (c, d)$.*
- $\mathsf{Dec}(k, \mathsf{ct} = (c, d))$*: First, Compute $r = H(k, d) \oplus c$, and then $m = G(k, r) \oplus d$.*

Correctness is easy to verify given that G and H are deterministic. The ciphertext has length $|c| + |d| = n + \mathsf{poly}(\lambda)$, which gives an ideal rate of $(1 - o(1))$. The secret key size is $\mathsf{poly}(\lambda)$, which is also optimal.

The construction has $(2^\lambda, 2^\lambda, \frac{S}{n}, S)$-MULT-SIM-CPA security. Notice that this security holds for an unbounded (exponential) number of ciphertexts per user.

Theorem 6.7.1. *If G, H are hash functions modelled as random oracles, then construction 6.7.1 is $(2^\lambda, 2^\lambda, \frac{S}{n}, S)$-MULT-SIM-CPA secure.*

Proof. We prove this by limiting the adversary's queries to the random oracles G and H through several steps. First, recall the challenger's behavior in the real mode experiment:

- For $i \in [2^\lambda]$, sample a uniform $k_i \in \{0,1\}^{\mathsf{poly}(\lambda)}$.
- Receive a list of message queries $\{m_{i,j}\}_{i,j\in[2^\lambda]}$ from $\mathcal{A}_1$.
- For each $i, j \in [2^\lambda]$:
  - Sample a uniformly random $r_{i,j} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
  - Let $d_{i,j} = G(k_i, r_{i,j}) \oplus m_{i,j}$.
  - Let $c_{i,j} = H(k_i, d_{i,j}) \oplus r_{i,j}$.
  - Let $\mathsf{ct}_{i,j} = (c_{i,j}, d_{i,j})$.
- Send $\{\mathsf{ct}_{i,j}\}_{i,j}$ to $\mathcal{A}_1$ and receive a state st of size at most S.
- On input of st, $\{m_{i,j}\}_{i,j}$, $\{k_i\}_i$, $\mathcal{A}_2$ outputs a bit 1/0.

Now we limit the adversary's queries to G and H.

1. Notice that $\mathcal{A}_1$ can never query the random oracles G or H using some k_i, since the k_i's remain hidden from $\mathcal{A}_1$. The probability of $\mathcal{A}_1$ guessing a k_i correctly is exponentially small.

2. $\mathcal{A}_2$ can only query $G(k_i, r_{i,j})$ after querying $H(k_i, d_{i,j})$. This is because if $\mathcal{A}_2$ has not queried $H(k_i, d_{i,j})$ yet, then $r_{i,j} = H(k_i, d_{i,j}) \oplus c_{i,j}$ is just a uniformly random λ-bit string to the adversary, and the probability of guessing it correctly is exponentially small.

3. $\mathcal{A}_2$ can make at most S/n queries to $H(k_i, d_{i,j})$ with different (i,j) pairs. Notice that the probability of guessing a $d_{i,j}$ correctly is exponentially small. So in order to successfully query $H(k_i, d_{i,j})$, $d_{i,j}$ must be stored in st. But each $d_{i,j}$ is n bits, and $|\mathsf{st}| \leq S$, so $\mathcal{A}_2$ can recover at most S/n such $d_{i,j}$'s and hence make at most S/n valid queries to H.

With these limitations in mind, the simulator for the ideal mode experiment works as follow:

- Receive a list of message queries $\{m_{i,j}\}_{i,j\in[2^\lambda]}$ from $\mathcal{A}_1$. Discard without looking at it.
- For each $i,j \in [2^\lambda]$:
  - Sample a uniformly random $c_{i,j} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
  - Sample a uniformly random $d_{i,j} \leftarrow \{0,1\}^n$.
  - Let $\mathsf{ct}_{i,j} = (c_{i,j}, d_{i,j})$.
- Send $\{\mathsf{ct}_{i,j}\}_{i,j}$ to $\mathcal{A}_1$ and receive a state st of size at most S.
- Sample uniformly random keys $k_i \in \{0,1\}^{\mathsf{poly}(\lambda)}$
- $\mathcal{A}_2$ receives st, $\{m_{i,j}\}_{i,j}$, $\{k_i\}_i$.
- Whenever $\mathcal{A}_2$ queries the random oracle on $H(k_i, d_{i,j})$, submit a query for message $m_{i,j}$. There will be at most S/n such queries. Program $H(k_i, d_{i,j})$ to output a uniformly random string $r'_{i,j} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$, and program $G(k_i, r'_{i,j} \oplus c_{i,j}) = m_{i,j} \oplus d_{i,j}$.
- $\mathcal{A}_2$ outputs a bit 1/0.

Notice that this simulator queries a subset of messages that has size at most S/n. It is easy to see that a PPT adversary cannot distinguish between the challenger and the simulator. For a pair of index (i,j) that $\mathcal{A}$ has queried $H(k_i, d_{i,j})$, we have $\mathsf{ct}_{i,j} = (c_{i,j}, d_{i,j}) = (H(k_i, d_{i,j}) \oplus r_{i,j}, G(k_i, r_{i,j}) \oplus m_{i,j})$ is just a faithful encryption of $m_{i,j}$, which is the same thing the challenger in the real mode would output. For a pair of index (i,j) that $\mathcal{A}$ has *not* queried $H(k_i, d_{i,j})$, then by limitation 2 above, $\mathcal{A}$ has also not queried $G(k_i, r_{i,j})$. Here, $m_{i,j}$ is essentially masked with a random string, so the adversary cannot tell between an encryption of $m_{i,j}$ and a random ciphertext, i.e. the challenger output and the simulator output. □

6.7.2 Rate-1 Incompressible PKE from Random Oracles

We then show how to construct rate-1 incompressible PKE from random oracles, plain PKE, and rate-1 incompressible SKE. The construction is essentially a hybrid mode PKE with random oracles plugged in. Notice that this construction can be viewed as a generalization of Construction 5 in Section 7.1 of[25].

Construction 6.7.2. *Let λ, S be security parameters. Given $\mathsf{PKE}' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ a plain PKE scheme with many-time CPA security, $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ a rate-1 incompressible SKE with $(2^\lambda, 1, q, S)$-MULT-SIM-CPA security, message space $\{0,1\}^n$ and key space $\{0,1\}^{\mathcal{L}_k}$, and $H : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}^{\mathcal{L}_k}$ a hash function modelled as a random oracle, we construct $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for message space $\{0,1\}^n$ as follows:*

- $\mathsf{Gen}(1^\lambda, 1^S)$*: Run* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE}'.\mathsf{Gen}'(1^\lambda)$. *Output* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$*: Sample a short random* $r \in \{0,1\}^{\mathsf{poly}(\lambda)}$. *Compute* $c \leftarrow \mathsf{PKE}'.\mathsf{Enc}'(\mathsf{pk}, r)$ *and* $d \leftarrow \mathsf{SKE}.\mathsf{Enc}(H(r), m)$. *Output* $\mathsf{ct} = (c, d)$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct} = (c, d))$*: Get* $r \leftarrow \mathsf{PKE}'.\mathsf{Dec}'(\mathsf{sk}, c)$*, and output* $m \leftarrow \mathsf{SKE}.\mathsf{Dec}(H(r), d)$.

It is easy to see that given the correctness of PKE' and SKE and that H is deterministic, this construction is correct. The ciphertexts have length $|c| + |d| = n + \mathsf{poly}(\lambda)$, yielding an ideal rate of $(1 - o(1))$. The public key and the private key both have size $\mathsf{poly}(\lambda)$, which is optimal.

We show that the construction has $(2^\lambda, 2^\lambda, q, S)$-$\mathsf{MULT\text{-}SIM\text{-}CPA}$ security.

Theorem 6.7.2. *If* PKE' *has many-time CPA security,* SKE *has* $(2^\lambda, 1, q, S)$-$\mathsf{MULT\text{-}SIM\text{-}CPA}$ *security, and H is a hash function modelled as a random oracle, then construction 6.7.2 is* $(2^\lambda, 2^\lambda, q, S)$-$\mathsf{MULT\text{-}SIM\text{-}CPA}$ *secure.*

Proof. We show how to construct the simulator for the ideal mode experiment by using the simulator for the underlying incompressible SKE.

- For $i \in [2^\lambda]$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PKE}'.\mathsf{Gen}'(1^\lambda)$.
- Receive a list of message queries $\{m_{i,j}\}_{i,j \in [2^\lambda]}$ from $\mathcal{A}_1$. Discard without looking at it.
- Run the simulation for the incompressible SKE to obtain a list of ciphertexts $\{d_{i,j}\}_{i,j}$ for $i, j \in [2^\lambda]$.
- For each $i, j \in [2^\lambda]$:
  - Sample a uniformly random $r_{i,j} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
  - Let $c_{i,j} \leftarrow \mathsf{PKE}'.\mathsf{Enc}'(\mathsf{pk}_i, r_{i,j})$.
  - Let $\mathsf{ct}_{i,j} = (c_{i,j}, d_{i,j})$.

- Send $\{\mathsf{ct}_{i,j}\}_{i,j}$ to $\mathcal{A}_1$ and receive a state st of size at most S. Forward the state st to the simulator for the incompressible SKE.

- Run the incompressible SKE simulator to obtain the simulated symmetric keys $\{k_{i,j}\}_{i,j}$, and reprogram the random oracle H to output $H(r_{i,j}) = k_{i,j}$. In the process, if the SKE simulator queries for message $m_{i,j}$, also query for $m_{i,j}$. Notice that there will be at most q such queries.

- $\mathcal{A}_2$ receives st, $\{m_{i,j}\}_{i,j}$, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_i$ and outputs a bit 1/0.

The security of the underlying PKE' ensures that the reprogramming of H is undetectable to the adversary. This is because for $\mathcal{A}_1$, $r_{i,j}$'s are encrypted under PKE', and the PKE' private keys remain hidden to $\mathcal{A}_1$. Therefore, $\mathcal{A}_1$ is not able to query H on any of the $r_{i,j}$'s before the reprogramming happens, and hence is not able to detect it.

By the property of the incompressible SKE simulator, the rest is easy to see that the simulator constructed above is indistinguishable from a real mode challenger. □

Remark 6.7.1. *By using construction 6.7.1 as the incompressible SKE scheme in construction 6.7.2, we would get a rate-*1*, random oracle based, incompressible PKE scheme for message space* $\{0,1\}^n$ *that has* $(2^\lambda, 2^\lambda, \frac{S}{n}, S)$-MULT-SIM-CPA *security.*

References

[1] Aggarwal, D., Obremski, M., Ribeiro, J. L., Siniscalchi, L., & Visconti, I. (2020). How to extract useful randomness from unreliable sources. In A. Canteaut & Y. Ishai (Eds.), *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS* (pp. 343–372).: Springer, Heidelberg.

[2] Albrecht, M., Cid, C., Paterson, K. G., Tjhai, C. J., & Tomlinson, M. (2019). Ntskem. *NIST submissions*, 2, 4–13.

[3] Alwen, J., Dodis, Y., Naor, M., Segev, G., Walfish, S., & Wichs, D. (2010). Public-key encryption in the bounded-retrieval model. In H. Gilbert (Ed.), *EUROCRYPT 2010*, volume 6110 of *LNCS* (pp. 113–134).: Springer, Heidelberg.

[4] Alwen, J., Dodis, Y., & Wichs, D. (2009). Leakage-resilient public-key cryptography in the bounded-retrieval model. In S. Halevi (Ed.), *CRYPTO 2009*, volume 5677 of *LNCS* (pp. 36–54).: Springer, Heidelberg.

[5] Ananth, P., Jain, A., & Sahai, A. (2017). Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In J. Katz & H. Shacham (Eds.), *CRYPTO 2017, Part II*, volume 10402 of *LNCS* (pp. 252–279).: Springer, Heidelberg.

[6] Ananth, P. & Placa, R. L. L. (2020). Secure software leasing.

[7] Aumann, Y. & Rabin, M. O. (1999). Information theoretically secure communication in the limited storage space model. In M. J. Wiener (Ed.), *CRYPTO'99*, volume 1666 of *LNCS* (pp. 65–79).: Springer, Heidelberg.

[8] Ball, M., Dachman-Soled, D., Kulkarni, M., & Malkin, T. (2018). Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In J. B. Nielsen & V. Rijmen (Eds.), *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS* (pp. 618–650).: Springer, Heidelberg.

[9] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. P., & Yang, K. (2001). On the (im)possibility of obfuscating programs. In J. Kilian (Ed.), *CRYPTO 2001*, volume 2139 of *LNCS* (pp. 1–18).: Springer, Heidelberg.

[10] Bardet, M., Barelli, E., Blazy, O., Torres, R. C., Couvreur, A., Gaborit, P., Otmani, A., Sendrier, N., & Tillich, J.-P. (2017). Big quake binary goppa quasi–cyclic key encapsulation. *NIST submissions*.

[11] Barnett Jr., T. (2016). The zettabyte era officially begins (how much is that?). https://blogs.cisco.com/sp/the-zettabyte-era-officially-begins-how-much-is-that.

[12] Barrington, D. A. M. (1986). Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *18th ACM STOC* (pp. 1–5).: ACM Press.

[13] Bellare, M. & Fuchsbauer, G. (2014). Policy-based signatures. In H. Krawczyk (Ed.), *PKC 2014*, volume 8383 of *LNCS* (pp. 520–537).: Springer, Heidelberg.

[14] Bellare, M., Kane, D., & Rogaway, P. (2016). Big-key symmetric encryption: Resisting key exfiltration. In M. Robshaw & J. Katz (Eds.), *CRYPTO 2016, Part I*, volume 9814 of *LNCS* (pp. 373–402).: Springer, Heidelberg.

[15] Bendlin, R., Nielsen, J. B., Nordholt, P. S., & Orlandi, C. (2011). Lower and upper bounds for deniable public-key encryption. In D. H. Lee & X. Wang (Eds.), *ASIACRYPT 2011*, volume 7073 of *LNCS* (pp. 125–142).: Springer, Heidelberg.

[16] Bernstein, D. J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., et al. (2017). Classic mceliece: conservative code-based cryptography. *NIST submissions*.

[17] Biryukov, A. & Khovratovich, D. (2016). Egalitarian computing. In T. Holz & S. Savage (Eds.), *USENIX Security 2016* (pp. 315–326).: USENIX Association.

[18] Bitansky, N., Canetti, R., Kalai, Y. T., & Paneth, O. (2014). On virtual grey box obfuscation for general circuits. In J. A. Garay & R. Gennaro (Eds.), *CRYPTO 2014, Part II*, volume 8617 of *LNCS* (pp. 108–125).: Springer, Heidelberg.

[19] Boneh, D., Sahai, A., & Waters, B. (2011). Functional encryption: Definitions and challenges. In Y. Ishai (Ed.), *TCC 2011*, volume 6597 of *LNCS* (pp. 253–273).: Springer, Heidelberg.

[20] Boneh, D. & Waters, B. (2013). Constrained pseudorandom functions and their applications. In K. Sako & P. Sarkar (Eds.), *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS* (pp. 280–300).: Springer, Heidelberg.

[21] Boneh, D. & Zhandry, M. (2014). Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In J. A. Garay & R. Gennaro (Eds.), *CRYPTO 2014, Part I*, volume 8616 of *LNCS* (pp. 480–499).: Springer, Heidelberg.

[22] Boyle, E., Chung, K.-M., & Pass, R. (2014a). On extractability obfuscation. In Y. Lindell (Ed.), *TCC 2014*, volume 8349 of *LNCS* (pp. 52–73).: Springer, Heidelberg.

[23] Boyle, E., Goldwasser, S., & Ivan, I. (2014b). Functional signatures and pseudorandom functions. In H. Krawczyk (Ed.), *PKC 2014*, volume 8383 of *LNCS* (pp. 501–519).: Springer, Heidelberg.

[24] Brakerski, Z., Döttling, N., Garg, S., & Malavolta, G. (2020). Candidate iO from homomorphic encryption schemes. In A. Canteaut & Y. Ishai (Eds.), *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS* (pp. 79–109).: Springer, Heidelberg.

[25] Branco, P., Döttling, N., & Dujmovic, J. (2022). Rate-1 incompressible encryption from standard assumptions. In E. Kiltz & V. Vaikuntanathan (Eds.), *TCC 2022, Part II*, volume 13748 of *LNCS* (pp. 33–69).: Springer, Heidelberg.

[26] Cachin, C., Crépeau, C., & Marcil, J. (1998). Oblivious transfer with a memory-bounded receiver. In *39th FOCS* (pp. 493–502).: IEEE Computer Society Press.

[27] Cachin, C. & Maurer, U. M. (1997). Unconditional security against memory-bounded adversaries. In B. S. Kaliski Jr. (Ed.), *CRYPTO'97*, volume 1294 of *LNCS* (pp. 292–306).: Springer, Heidelberg.

[28] Canetti, R., Dwork, C., Naor, M., & Ostrovsky, R. (1997). Deniable encryption. In B. S. Kaliski Jr. (Ed.), *CRYPTO'97*, volume 1294 of *LNCS* (pp. 90–104).: Springer, Heidelberg.

[29] Canetti, R., Halevi, S., & Katz, J. (2003). A forward-secure public-key encryption scheme. In E. Biham (Ed.), *EUROCRYPT 2003*, volume 2656 of *LNCS* (pp. 255–271).: Springer, Heidelberg.

[30] Canetti, R., Park, S., & Poburinnaya, O. (2020). Fully deniable interactive encryption. In D. Micciancio & T. Ristenpart (Eds.), *CRYPTO 2020, Part I*, volume 12170 of *LNCS* (pp. 807–835).: Springer, Heidelberg.

[31] Cash, D., Ding, Y. Z., Dodis, Y., Lee, W., Lipton, R. J., & Walfish, S. (2007). Intrusion-resilient key exchange in the bounded retrieval model. In S. P. Vadhan (Ed.), *TCC 2007*, volume 4392 of *LNCS* (pp. 479–498).: Springer, Heidelberg.

[32] Cramer, R. & Shoup, V. (2002). Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In L. R. Knudsen (Ed.), *EUROCRYPT 2002*, volume 2332 of *LNCS* (pp. 45–64).: Springer, Heidelberg.

[33] Cramer, R. & Shoup, V. (2003). Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1), 167–226.

[34] Damgård, I., Fehr, S., Renner, R., Salvail, L., & Schaffner, C. (2007). A tight high-order entropic quantum uncertainty relation with applications. In A. Menezes (Ed.), *CRYPTO 2007*, volume 4622 of *LNCS* (pp. 360–378).: Springer, Heidelberg.

[35] Damgård, I. & Jurik, M. (2001). A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In K. Kim (Ed.), *PKC 2001*, volume 1992 of *LNCS* (pp. 119–136).: Springer, Heidelberg.

[36] Department, S. R. (2018). Data center storage capacity worldwide from 2016 to 2021, by segment. https://www.statista.com/statistics/638593/worldwide-data-center-storage-capacity-cloud-vs-traditional/.

[37] Di Crescenzo, G., Lipton, R. J., & Walfish, S. (2006). Perfectly secure password protocols in the bounded retrieval model. In S. Halevi & T. Rabin (Eds.), *TCC 2006*, volume 3876 of *LNCS* (pp. 225–244).: Springer, Heidelberg.

[38] Diffie, W., van Oorschot, P. C., & Wiener, M. J. (1992). Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2), 107–125.

[39] Ding, Y. Z. (2001). Oblivious transfer in the bounded storage model. In J. Kilian (Ed.), *CRYPTO 2001*, volume 2139 of *LNCS* (pp. 155–170).: Springer, Heidelberg.

[40] Ding, Y. Z., Harnik, D., Rosen, A., & Shaltiel, R. (2004). Constant-round oblivious transfer in the bounded storage model. In M. Naor (Ed.), *TCC 2004*, volume 2951 of *LNCS* (pp. 446–472).: Springer, Heidelberg.

[41] Dinur, I., Stemmer, U., Woodruff, D. P., & Zhou, S. (2023). On differential privacy and adaptive data analysis with bounded space. Cryptology ePrint Archive, Report 2023/171. https://eprint.iacr.org/2023/171.

[42] Dodis, Y., Quach, W., & Wichs, D. (2022). Authentication in the bounded storage model. In O. Dunkelman & S. Dziembowski (Eds.), *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS* (pp. 737–766).: Springer, Heidelberg.

[43] Dodis, Y., Reyzin, L., & Smith, A. (2004). Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin & J. Camenisch (Eds.), *EUROCRYPT 2004*, volume 3027 of *LNCS* (pp. 523–540).: Springer, Heidelberg.

[44] Dziembowski, S. (2006a). Intrusion-resilience via the bounded-storage model. In S. Halevi & T. Rabin (Eds.), *TCC 2006*, volume 3876 of *LNCS* (pp. 207–224).: Springer, Heidelberg.

[45] Dziembowski, S. (2006b). On forward-secure storage (extended abstract). In C. Dwork (Ed.), *CRYPTO 2006*, volume 4117 of *LNCS* (pp. 251–270).: Springer, Heidelberg.

[46] Dziembowski, S., Kazana, T., & Zdanowicz, M. (2018). Quasi chain rule for min-entropy. *Information Processing Letters*, 134, 62–66.

[47] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., & Waters, B. (2013a). Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS* (pp. 40–49).: IEEE Computer Society Press.

[48] Garg, S., Gentry, C., Halevi, S., Sahai, A., & Waters, B. (2013b). Attribute-based encryption for circuits from multilinear maps. In R. Canetti & J. A. Garay (Eds.), *CRYPTO 2013, Part II*, volume 8043 of *LNCS* (pp. 479–499).: Springer, Heidelberg.

[49] Garg, S., Gentry, C., Halevi, S., & Wichs, D. (2014). On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In J. A. Garay & R. Gennaro (Eds.), *CRYPTO 2014, Part I*, volume 8616 of *LNCS* (pp. 518–535).: Springer, Heidelberg.

[50] Garg, S., Gentry, C., Sahai, A., & Waters, B. (2013c). Witness encryption and its applications. In D. Boneh, T. Roughgarden, & J. Feigenbaum (Eds.), *45th ACM STOC* (pp. 467–476).: ACM Press.

[51] Gay, R. & Pass, R. (2021). Indistinguishability obfuscation from circular security. In S. Khuller & V. V. Williams (Eds.), *53rd ACM STOC* (pp. 736–749).: ACM Press.

[52] Gentry, C., Sahai, A., & Waters, B. (2013). Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R.

Canetti & J. A. Garay (Eds.), *CRYPTO 2013, Part I*, volume 8042 of *LNCS* (pp. 75–92).: Springer, Heidelberg.

[53] Goldreich, O., Goldwasser, S., & Micali, S. (1984). On the cryptographic applications of random functions. In G. R. Blakley & D. Chaum (Eds.), *CRYPTO'84*, volume 196 of *LNCS* (pp. 276–288).: Springer, Heidelberg.

[54] Goldreich, O., Goldwasser, S., & Micali, S. (1986). How to construct random functions. *Journal of the ACM*, 33(4), 792–807.

[55] Goldreich, O. & Levin, L. A. (1989). A hard-core predicate for all one-way functions. In *21st ACM STOC* (pp. 25–32).: ACM Press.

[56] Goldwasser, S. & Kalai, Y. T. (2003). On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS* (pp. 102–115).: IEEE Computer Society Press.

[57] Gorbunov, S., Vaikuntanathan, V., & Wee, H. (2012). Functional encryption with bounded collusions via multi-party computation. In R. Safavi-Naini & R. Canetti (Eds.), *CRYPTO 2012*, volume 7417 of *LNCS* (pp. 162–179).: Springer, Heidelberg.

[58] Goyal, R., Koppula, V., & Waters, B. (2017). Lockable obfuscation. In C. Umans (Ed.), *58th FOCS* (pp. 612–621).: IEEE Computer Society Press.

[59] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *28th ACM STOC* (pp. 212–219).: ACM Press.

[60] Guan, J., Wichs, D., & Zhandry, M. (2022). Incompressible cryptography. In O. Dunkelman & S. Dziembowski (Eds.), *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS* (pp. 700–730).: Springer, Heidelberg.

[61] Guan, J., Wichs, D., & Zhandry, M. (2023). Somewhere randomness extraction and security against bounded-storage mass surveillance. Cryptology ePrint Archive, Paper 2023/409. https://eprint.iacr.org/2023/409.

[62] Guan, J. & Zhandry, M. (2019). Simple schemes in the bounded storage model. In Y. Ishai & V. Rijmen (Eds.), *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS* (pp. 500–524).: Springer, Heidelberg.

[63] Guan, J. & Zhandry, M. (2021). Disappearing cryptography in the bounded storage model. In K. Nissim & B. Waters (Eds.), *TCC 2021, Part II*, volume 13043 of *LNCS* (pp. 365–396).: Springer, Heidelberg.

[64] Günther, C. G. (1990). An identity-based key-exchange protocol. In J.-J. Quisquater & J. Vandewalle (Eds.), *EUROCRYPT'89*, volume 434 of *LNCS* (pp. 29–37).: Springer, Heidelberg.

[65] Guruswami, V. (2004). *List decoding of error-correcting codes: winning thesis of the 2002 ACM doctoral dissertation competition*, volume 3282. Springer Science & Business Media.

[66] Haber, S. & Stornetta, W. S. (1991). How to time-stamp a digital document. In A. J. Menezes & S. A. Vanstone (Eds.), *CRYPTO'90*, volume 537 of *LNCS* (pp. 437–455).: Springer, Heidelberg.

[67] Holenstein, T., Künzler, R., & Tessaro, S. (2011). The equivalence of the random oracle model and the ideal cipher model, revisited. In L. Fortnow & S. P. Vadhan (Eds.), *43rd ACM STOC* (pp. 89–98).: ACM Press.

[68] Hubacek, P. & Wichs, D. (2015). On the communication complexity of secure function evaluation with long output. In T. Roughgarden (Ed.), *ITCS 2015* (pp. 163–172).: ACM.

[69] Impagliazzo, R., Levin, L. A., & Luby, M. (1989). Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC* (pp. 12–24).: ACM Press.

[70] Impagliazzo, R. & Rudich, S. (1990). Limits on the provable consequences of one-way permutations. In S. Goldwasser (Ed.), *CRYPTO'88*, volume 403 of *LNCS* (pp. 8–26).: Springer, Heidelberg.

[71] Jain, A., Lin, H., & Sahai, A. (2021). Indistinguishability obfuscation from well-founded assumptions. In S. Khuller & V. V. Williams (Eds.), *53rd ACM STOC* (pp. 60–73).: ACM Press.

[72] Kalai, Y. T. (2005). Smooth projective hashing and two-message oblivious transfer. In R. Cramer (Ed.), *EUROCRYPT 2005*, volume 3494 of *LNCS* (pp. 78–95).: Springer, Heidelberg.

[73] Kiayias, A., Papadopoulos, S., Triandopoulos, N., & Zacharias, T. (2013). Delegatable pseudorandom functions and applications. In A.-R. Sadeghi, V. D. Gligor, & M. Yung (Eds.), *ACM CCS 2013* (pp. 669–684).: ACM Press.

[74] Kilian, J. (1988). Founding cryptography on oblivious transfer. In *20th ACM STOC* (pp. 20–31).: ACM Press.

[75] Landerreche, E., Stevens, M., & Schaffner, C. (2019). Non-interactive cryptographic timestamping based on verifiable delay functions. Cryptology ePrint Archive, Report 2019/197. https://eprint.iacr.org/2019/197.

[76] Lu, C.-J. (2002). Hyper-encryption against space-bounded adversaries from on-line strong extractors. In M. Yung (Ed.), *CRYPTO 2002*, volume 2442 of *LNCS* (pp. 257–271).: Springer, Heidelberg.

[77] Maurer, U. M. (1992). Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1), 53–66.

[78] Moran, T., Shaltiel, R., & Ta-Shma, A. (2004). Non-interactive timestamping in the bounded storage model. In M. Franklin (Ed.), *CRYPTO 2004*, volume 3152 of *LNCS* (pp. 460–476).: Springer, Heidelberg.

[79] Moran, T. & Wichs, D. (2020). Incompressible encodings. In D. Micciancio & T. Ristenpart (Eds.), *CRYPTO 2020, Part I*, volume 12170 of *LNCS* (pp. 494–523).: Springer, Heidelberg.

[80] Nisan, N. (1990). Psuedorandom generators for space-bounded computation. In *22nd ACM STOC* (pp. 204–212).: ACM Press.

[81] O'Neill, A. (2010). Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556. https://eprint.iacr.org/2010/556.

[82] Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In J. Stern (Ed.), *EUROCRYPT'99*, volume 1592 of *LNCS* (pp. 223–238).: Springer, Heidelberg.

[83] Peikert, C. & Waters, B. (2008). Lossy trapdoor functions and their applications. In R. E. Ladner & C. Dwork (Eds.), *40th ACM STOC* (pp. 187–196).: ACM Press.

[84] Randall, D. (1993). Efficient generation of random nonsingular matrices. *Random Structures & Algorithms*, 4.

[85] Raz, R. (2016). Fast learning requires good memory: A time-space lower bound for parity learning. In I. Dinur (Ed.), *57th FOCS* (pp. 266–275).: IEEE Computer Society Press.

[86] Raz, R. (2017). A time-space lower bound for a large class of learning problems. In C. Umans (Ed.), *58th FOCS* (pp. 732–742).: IEEE Computer Society Press.

[87] Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow & R. Fagin (Eds.), *37th ACM STOC* (pp. 84–93).: ACM Press.

[88] Ristenpart, T., Shacham, H., & Shrimpton, T. (2011). Careful with composition: Limitations of the indifferentiability framework. In K. G. Paterson (Ed.), *EUROCRYPT 2011*, volume 6632 of *LNCS* (pp. 487–506).: Springer, Heidelberg.

[89] Rivest, R. L. (1997). All-or-nothing encryption and the package transform. In E. Biham (Ed.), *FSE'97*, volume 1267 of *LNCS* (pp. 210–218).: Springer, Heidelberg.

[90] Rothblum, R. (2011). Homomorphic encryption: From private-key to public-key. In Y. Ishai (Ed.), *TCC 2011*, volume 6597 of *LNCS* (pp. 219–234).: Springer, Heidelberg.

[91] Sahai, A. & Waters, B. R. (2005). Fuzzy identity-based encryption. In R. Cramer (Ed.), *EUROCRYPT 2005*, volume 3494 of *LNCS* (pp. 457–473).: Springer, Heidelberg.

[92] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS* (pp. 124–134).: IEEE Computer Society Press.

[93] Vadhan, S. P. (2003). On constructing locally computable extractors and cryptosystems in the bounded storage model. In D. Boneh (Ed.), *CRYPTO 2003*, volume 2729 of *LNCS* (pp. 61–77).: Springer, Heidelberg.

[94] Vadhan, S. P. et al. (2012). Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3), 1–336.

[95] Wee, H. & Wichs, D. (2020). Candidate obfuscation via oblivious LWE sampling. Cryptology ePrint Archive, Report 2020/1042. https://eprint.iacr.org/2020/1042.

[96] Wichs, D. (2013). Barriers in cryptography with weak, correlated and leaky sources. In R. D. Kleinberg (Ed.), *ITCS 2013* (pp. 111–126).: ACM.

[97] Wichs, D. & Zirdelis, G. (2017). Obfuscating compute-and-compare programs under LWE. In C. Umans (Ed.), *58th FOCS* (pp. 600–611).: IEEE Computer Society Press.

[98] Zaverucha, G. (2015). Stronger password-based encryption using all-or-nothing transforms.