Disappearing Cryptography in the Bounded Storage Model

Jiaxin Guan* and Mark Zhandry**

Princeton University and NTT Research, USA

Abstract. In this work, we study *disappearing cryptography* in the bounded storage model. Here, 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.

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 VBB security in this model, but propose candidate constructions for a weaker security goal, namely 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.

1 Introduction

The bounded storage model [Mau92] 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 [Raz16, GZ19], 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.

This Work: Disappearing Cryptography. A critical feature of the bounded storage model is that the large transmission cannot be entirely stored by the adversary.

* jiaxin@guan.io
** mzhandry@cs.princeton.edu

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

1.1 Motivating Examples

Example 1: Deniable Encryption. Deniable encryption [CDNO97] 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 [BNNO11], such receiver deniable encryption is impossible for "normal" encryption where the ciphertext is just a single (concise) transmission from Bob to Alice[1]. Prior works [CDNO97, CPP20] therefore consider a more general notion of encryption that involves back-and-forth communication between the parties.

In this work, 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 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.

[1] 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.

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. [BGI+01], 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 [BGI+01], as well as virtual grey box obfuscation (VGBO) [BCKP14]. 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.

1.2 Our Results

In this work, 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 [GGH+13b] 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[2], *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.

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. [BGI+01] impossibility, but take care to show that it still works for online obfuscation.

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

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 homomorphically 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 0 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 [GKW17, WZ17], a technique used toward a similar goal in [AP20]. 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.

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[3]. 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 [BCKP14], 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 3.

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 [GGHW14]. So we therefore stick to VGB obfuscation.

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 [GGSW13]: 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 simulator is computationally unbounded, meaning it can invert f to recover sk at the beginning of the experiment, and then query the program on sk.

[3] Concretely, it can break Enc to learn α.

Our solution is to replace f with a lossy function [PW08], 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 5.

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 7, 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 modi-

fying 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 6, 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 [BF14].

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.

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 [GGH$^+$13a]. As in [GGH$^+$13a], we first convert an NC^1 circuit into a matrix branching program using Barrington's theorem [Bar86]. In [GGH$^+$13a], the program is then "re-randomized" following Kilian [Kil88] 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. [GGH$^+$13a] 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 8.

Construction 2: Time-stamping. Our second construction is based on time-stamping [MST04] 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.

1.6 Related Work

Time-stamping in the bounded storage model [MST04], 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 [LSS19].

Dziembowski [Dzi06] 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. [DvW92]). However, standard-model constructions of forward security (non-interactive) encryption such as [CHK03] always involve updating the secret keys. Our construction does not require the secret key to be updated.

2 Preliminaries

Different sections of this paper rely on different cryptographic primitives. To minimize the page-turning effort of our reader, we will introduce the related notions and definitions separately in each section. Here we will just state the notations that are used throughout this paper.

We use capital bold letters to denote a matrix $\mathbf{M}$. Lowercase bold letters denote vectors $\mathbf{v}$. For $n \in \mathbb{N}$ we let $[n]$ denote the ordered set $\{1, 2, \ldots, n\}$. 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 $\mathbf{M}_1, \ldots, \mathbf{M}_n$.

3 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 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[4] 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.
     • 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 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 $q = \mathsf{poly}(\lambda)$*.*

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

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 1 (k-time iO Security). We modify Definition 2 to allow $q = \mathsf{superpoly}(\lambda)$ to obtain the definition for k-time iO Security.

Remark 2 (k-time VBB Security). We modify Definition 2 to restrict $\mathcal{S}$ to be a PPT simulator to obtain the definition for k-time VBB Security. We show in Section 4 that online obfuscators with VBB security do not exist.

Remark 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 (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 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.

4.1 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 3 (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 (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 [GSW13] have shown how to construct such an FHE scheme assuming the hardness of the LWE problem.

4.2 Obfuscation of Compute-and-Compare Programs

The idea of compute-and-compare programs was first raised by Wichs and Zirdelis [WZ17] in 2017. Around the same time, the work of Goyal, Koppula and Waters [GKW17] 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 [WZ17].

Definition 5 (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 [WZ17] have shown that obfuscation of multi-bit compute-and-compare programs exists, assuming the hardness of the LWE problem.

Lemma 1 ([WZ17]). *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 Proof of VBB Impossibility

Theorem 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 $s_{\gg} \leftarrow o\mathcal{O}.\mathsf{Obf}(P)$. $\mathcal{A}$ computes $o\mathcal{O}.\mathsf{Eval}(s_{\gg}, 0)$, obtaining $\mathsf{FHE.Enc}_{\mathsf{pk}}(\alpha)$ and $\mathcal{Q}$[5].
- 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

[5] 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.

bounded $\mathcal{S}$, by lemma 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. □

5 Public Key Encryption with Disappearing Ciphertext Security

5.1 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 6 (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.

5.2 Lossy Function

Lossy functions are a subset of Lossy Trapdoor Functions due to Peikert and Waters [PW08] that do not require the existence of a trapdoor for the injective mode. To put formally:

Definition 7 (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. $\{X_\lambda\} \overset{c}{\approx} \{Y_\lambda\}$.*

5.3 Construction

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

Construction 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:

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

  Obfuscate the above program to obtain a stream $\mathsf{ct}_{\gg} \leftarrow \mathsf{Obf}(P_{f_s,y,m})$. 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 $\mathsf{Eval}(\mathsf{ct}_{\gg}, \mathsf{sk}) = P_{f_s,y,m}(\mathsf{sk}) = m$ as desired.

5.4 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_s,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 2. *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 3. *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. *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 5. *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 3.

Lemma 6. *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 2.

Theorem 2. *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 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. □

6 Disappearing Signature Scheme

6.1 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 $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ 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 $\sigma_{\gg} \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, 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 8. *Let λ, n be security parameters. A disappearing signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ 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*.

6.2 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 9 (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. $x \neq x^*$:*

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

Definition 10 (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 [BF14] have shown that such a signature scheme can be built from any one-way function.

6.3 Construction

We now present our construction of the disappearing signature scheme.

Construction 2. Let λ, n be the security parameters. Let $\mathsf{PPS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Punc})$ 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 $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ works as follows:

- $\mathsf{Gen}(1^\lambda, 1^n)$: Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PPS.Gen}(1^\lambda)$, and output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: Construct the program P as follows:

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

  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

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

  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.

6.4 Proof of Security

Theorem 3. *If* PPS *is a correct and secure prefix puncturable signature scheme, and* $o\mathcal{O}$ *is an online obfuscator with* 1*-time VGB security under* $S(n)$ *memory bound, then Construction 2 is secure under* $S(n)$ *memory bound.*

Proof. We prove the security of Construction 2 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 $\sigma^* = \mathsf{Eval}(\sigma'_{\gg}, x^*) = \mathsf{PPS.Sign}(\mathsf{sk}, (x^*, m'))$, 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':

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

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 3, 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 2 is secure. □

7 Functional Encryption

7.1 Definition

The concept of Functional Encryption (FE) is first raised by Sahai and Waters [SW05] and later formalized by Boneh, Sahai, Waters [BSW11] and O'Neill [O'N10]. Here we review the syntax and security definition of functional encryption and how they would translate to the bounded storage model.

Syntax of Functional Encryption. 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 $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{msk})$ takes as input the security parameter λ, and outputs the public key pk 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.
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$ takes as input the public key pk 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 require correctness and security of a functional encryption scheme.

Definition 11 (Correctness). *A functional encryption scheme* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be correct if for all* $C \in \{\mathcal{C}_\lambda\}$ *and* $m \in \mathcal{X}_\lambda$*:*

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

For the security definition, consider the following experiment:

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

– Run $\mathsf{Setup}(1^\lambda)$ to obtain keys $(\mathsf{pk}, \mathsf{msk})$ and sample a uniform bit $b \in \{0, 1\}$.
– The adversary $\mathcal{A}$ is given the public key pk.
– 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 $\mathsf{Enc}(\mathsf{pk}, m_b)$.
– 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 12 (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 these definitions would need to be modified for defining 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 13 (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).

7.2 Construction

Construction 3. 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 $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$. Sample the common reference string crs for the NIZK system. Output $(\mathsf{pk}, \mathsf{crs})$ 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 $(\mathsf{pk}, C, x, y, \mathsf{sk}_F)$ is in the language

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

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

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

  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

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

as desired.

7.3 Proof of Security

We prove the security of Construction 3 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 $\mathsf{Enc}(\mathsf{pk}, m_0)$.
- 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 $c' \leftarrow \mathsf{FE.Enc}(\mathsf{pk}, (m_b, k))$ instead of $c \leftarrow \mathsf{FE.Enc}(\mathsf{pk}, (m_b, \bot))$.
- 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 7. *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 8. *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_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 $\mathsf{NIZK}.\mathcal{V}(\mathsf{crs}, (\mathsf{pk}, C, x, y, \mathsf{sk}_F), \pi)$ 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 $\mathsf{FE.Dec}(\mathsf{sk}_F, c)$. 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 $\mathsf{FE.Dec}(sk_F, c) = F_{C,x,y}(m, \bot)$, and the program P' outputs $F_{C,x,y}(m, k)$. Notice that $F_{C,x,y}(m, \bot)$ 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 3, 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 9. *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 10. *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, $c \leftarrow \mathsf{FE.Enc}(\mathsf{pk}, (m_0, k))$, while in H_4, $c \leftarrow \mathsf{FE.Enc}(\mathsf{pk}, (m_1, k))$.

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 pk from the challenger, use the NIZK simulator to sample the crs, and send $(\mathsf{pk}, \mathsf{crs})$ 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 $P_{c,\mathsf{pk},\mathsf{crs}}$ 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 11. *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 12. *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 8.

Lemma 13. *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 7.

Theorem 4. *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 3 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. □

8 Candidate Construction 1

8.1 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 14 (Barrington's Theorem [Bar86]). *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 $h = 4^d$.*

8.2 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 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 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 8.3, 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 [Kil88] 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 $\mathbf{t}' = \mathbf{R}_h \cdot \mathbf{t}$, we start with $\mathbf{t}' = \mathbf{t}$, write $\mathbf{R}_h^{-1}$ as $\prod_{t=\tau}^{1} \mathbf{B}^{(h,t)}$, interpret each of the $\mathbf{B}^{(h,t)}$ 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 subroutine which, for input α, computes $\mathbf{r}_\alpha$, the α-th row of $\mathbf{M}_{i,b} \cdot \mathbf{R}_i$. This subroutine 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 15. *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 [Ran93].

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 [Ran93], $\mathbf{C}$ 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 15 to construct the distributions D_t, and then sample the matrices $\mathbf{B}^{(t,i)}$ from D_t. Lemma 15 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.

$\mathcal{O}_{\mathrm{PRF}}$ will work exactly as $\mathcal{O}_{\mathrm{ER}}$, 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 16. *For any choice of* $\mathsf{Convert}$*, assuming* PRF *is a secure PRF and* $\mathcal{O}_{Kil}$ *is* k*-time VGB secure when using* $\mathsf{Convert}$*, then* $\mathcal{O}_{PRF}$ *is* k*-time VGB secure when using* $\mathsf{Convert}$*.*

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

8.3 Instantiating Convert

Now we will discuss how we specifically instantiate $\mathsf{Convert}$, 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 [Bar86] plus Kilian randomization [Kil88] 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 [GGH+13a], 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 [Raz16] 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 following. 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.[6]

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 $\big(\mathsf{BP}_*(x_1, x_2)\big)_{x_1 \in X_1, x_2 \in X_2}$ 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 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 0 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 $m = \mathsf{poly}(\lambda)$.

[6] 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.

- 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 $\mathbf{M}'_{i,b} = \mathsf{diag}(\mathbf{M}_{i,b}, \mathbf{M}_{i,b}^{(1)}, \ldots, \mathbf{M}_{i,b}^{(\ell)})$. 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 $\mathbf{s}' = \left(\mathbf{s}\ \mathbf{s}^{(1)}\ \mathbf{s}^{(2)}\ \cdots\ \mathbf{s}^{(\ell)}\right)$ and the right bookend is now $\mathbf{t}' = \left(\mathbf{t}^T\ \left(\mathbf{t}^{(1)}\right)^T\ \left(\mathbf{t}^{(2)}\right)^T \cdots\ \left(\mathbf{t}^{(\ell)}\right)^T\right)^T$.

Notice that we have

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

If all reads of the input x are consistent, then we have $\mathsf{BP}_2^{(j)}(x) = 0$ 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 $\mathsf{BP}_2^{(j)}(x) = 1$ 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 $\mathsf{Convert}$, which we then plug into our framework.

9 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 [HS91], and since then we have seen various instantiations of digital time-stamping systems. One construction of particular interest is by Moran, Shaltiel and Ta-Shma [MST04], where they construct a non-interactive time-stamping scheme in the bounded storage model. This will be what we base our construction on.

Definition 14 (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 15 (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, [MST04] 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 16 (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. 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 [MST04], the sketch k, and thus $P_{C,k}$ will be of size $O(n)$ bits. Thus, we need to use an obfuscator such that $\mathsf{VGB.Obf}$ only expands the program by a constant factor. We conjecture that the constant-overhead construction of [AJS17] will work here. Alternatively, one can use branching-program based obfuscation directly from multilinear maps, for example [GGH$^+$13a] and follow-ups. [BCKP14] 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 $\mathsf{VGB.Obf}$ 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.

References

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 252–279. Springer, Heidelberg, August 2017.

[AP20] Prabhanjan Ananth and Rolando L. La Placa. Secure software leasing, 2020.

[Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *18th ACM STOC*, pages 1–5. ACM Press, May 1986.

[BCKP14] Nir Bitansky, Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On virtual grey box obfuscation for general circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2014.

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 520–537. Springer, Heidelberg, March 2014.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BNNO11] Rikke Bendlin, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Lower and upper bounds for deniable public-key encryption. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2011.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 90–104. Springer, Heidelberg, August 1997.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Heidelberg, May 2003.

[CPP20] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. Fully deniable interactive encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 807–835. Springer, Heidelberg, August 2020.

[DvW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, June 1992.

[Dzi06] Stefan Dziembowski. On forward-secure storage (extended abstract). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 251–270. Springer, Heidelberg, August 2006.

[GGH+13a] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 479–499. Springer, Heidelberg, August 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Heidelberg, August 2014.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, October 2017.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[GZ19] Jiaxin Guan and Mark Zhandary. Simple schemes in the bounded storage model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 500–524. Springer, Heidelberg, May 2019.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 437–455. Springer, Heidelberg, August 1991.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

[LSS19] Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. Cryptology ePrint Archive, Report 2019/197, 2019. https://eprint.iacr.org/2019/197.

[Mau92] Ueli M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, January 1992.

[MST04] Tal Moran, Ronen Shaltiel, and Amnon Ta-Shma. Non-interactive timestamping in the bounded storage model. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 460–476. Springer, Heidelberg, August 2004.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008.

[Ran93] Dana Randall. Efficient generation of random nonsingular matrices. *Random Structures & Algorithms*, 4, 01 1993.

[Raz16] Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. In Irit Dinur, editor, *57th FOCS*, pages 266–275. IEEE Computer Society Press, October 2016.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th FOCS*, pages 600–611. IEEE Computer Society Press, October 2017.