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 too 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 achieve information-theoretic, unconditional, and everlasting security; in contrast, the usual time-bounded adversary model generally requires 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. This large transmission is then subsequently used in such a way that whatever

[*] jiaxin@guan.io

[**] mzhandry@princeton.edu

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 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.

Remark 1. The usual bounded storage model as defined in [Mau92] envisions a trusted third party broadcasting a large stream of uniformly random bits, which is assumed to be too large to store. All other communication remains short. In this work, we operate in a slightly different setting where there is no trusted third party, but the large streams are instead generated by the users themselves. Additionally, we allow the stream of bits to be structured. However, we emphasize that we still require all parties to be low space.

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

[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.

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, or Alice is unable to broadcast.

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 several users trying to share a single decoder box. During broadcast time, each user records the encrypted broadcast individually. Then they pass the decoder box around to the various users, allowing them to decrypt their locally-stored broadcast one at a time. Of course, once one user decrypts the broadcast, they can simply send the decrypted contents to the other users. We imagine, however, that the contents are very large, and it is easier to send the decoder box than to transmit the large decrypted contents.

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 low-power embedded devices with limited capabilities; in particular, they cannot keep long-term state nor transmit outgoing messages. 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 for weak devices.

As before, our idea is to have the signatures 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, hosts the software centrally and allows the users to 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 with traditional software.

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, without sending 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$^+$01], is the ideal form of obfuscation: it informally says that having the obfuscated code is "no better than" having black box access to the functionality. Unfortunately, Barak et al. show that such VBB obfuscation is impossible. The counter-example works by essentially running the program on its own description, something that is not possible just given oracle access. As a consequence, other weaker notions have been used, including indistinguishability obfuscation (iO), differing inputs obfuscation [BGI$^+$01], and 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 weaker obfuscation notions. However, for securing intellectual property, 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 positive 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.

We demonstrate that, under the Learning With Errors (LWE) assumption, VBB obfuscation is still impossible. The proof closely follows Barak et al.'s proof for circuits, adapting it for online obfuscation. This rules out Example 5.

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. 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 appears to offer 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 loses access to the program after the transmission completes.

We give two candidate VGB online obfuscators based on very different ideas, and leave a provable secure scheme as an interesting open question.

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 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. We show 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

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

efficiently learned given just oracle access. We follow the Barak et al. [BGI$^+$01] impossibility, but take care to show that it still works for online obfuscation.

The idea is that, the version of Barak et al.'s impossibility that works for circuit obfuscation already has to contend with the fact that circuits cannot be evaluated on themselves, since a circuit is almost always larger than its input size. In order to get an impossibility result for online obfuscation, we show that their attack works in low storage. The full proof is given in the full version.

One issue that comes up in the naive adaptation of Barak et al.'s attack is that it requires the obfuscation to be streamed multiple times. We explain how to make the attack work with just a single stream using Compute-and-Compare obfuscation [GKW17, WZ17], following ideas from [AP20].

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, which avoids the impossibility. While iO is immensely 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[3].

We therefore give 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.

[3] The usual way indistinguishability approach to defining iO does not use a simulator, but is equivalent in the standard model to the simulation definition. In the online model, the indistinguishability and simulation models may not be equivalent. Nevertheless, the indistinguishability version of iO still appears to offer no advantages in the online setting, since in this version the adversary knows the programs in the clear from the very beginning.

1.4 Applications

Before giving our candidate online obfuscation schemes, 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 say that a public key encryption scheme has *disappearing ciphertext security* if the contents of a ciphertext remain hidden, even if the attacker subsequently learns the secret key.

Our first attempt is to use 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, the key difficulty is that we cannot switch to a hybrid where the secret key does not exist, as would be used to prove the standard CPA security of the scheme using witness encryption. After all, the adversary eventually sees the secret key, so it must always exist!

Toward a proof, 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.

Security would be trivial with online obfuscation with VBB security. However, difficulties arise with trying to formalize this intuition with our notion of VGB security. 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.

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. Lossy functions can be build under various standard assumptions such as DDH or LWE.

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 streamed. This means the simulator, despite being computationally unbounded, will be unable to query on sk, and thus 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 result is the following, proved in Section 4:

Theorem 1 (Informal). *Assuming the existence of VGB online obfuscation and lossy functions, there exists a public key encryption scheme with disappearing ciphertext security.*

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 a 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 6, we consider a disappearing ciphertext security variant, where the requirement that $g_i(m_0) = g_i(m_1)$ only holds for secret keys in possession when the ciphertext is transmitted. 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. In Section 6, we show how to modify the sketch above to get security to go through, yielding the following:

Theorem 3 (Informal). *Assuming the existence of VGB online obfuscation, NIZKs, non-uniform secure PRFs, and* standard-model *functional encryption, there exists a functional encryption scheme with disappearing ciphertext security.*

Disappearing Signatures. We next turn to constructing disappearing signatures, signatures that are large streams that can be verified online, but then the signature disappears after the transmission ends. We formalize this notion by modifying the usual chosen message security notion to give *disappearing signature security*, where 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 disappearing signatures in Section 5, using online obfuscation. An additional building block we need is a *prefix puncturable signature.* This is a 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 $x \neq x^*$, but sk_{x^*} is incapable of signing messages of the form (x^*, m). 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 reject 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. The result is the following theorem:

Theorem 2 (Informal). *Assuming the existence of VGB online obfuscation and one-way functions, there exists a disappearing signature scheme.*

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 space proportional to N, the height of the matrices. However, recording even a single matrix from the program requires space N^2, and so for adversaries with space somewhat less than N^2, it may be reasonable to conjecture that the program "disappears" after 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, since an adversary with space somewhat less than N^2 will be unable to even record a single matrix from the program. Moreover, we show how to block mixed-input attacks by choosing the matrix padding to have a special structure, which is inspired by the classical obfuscation techniques. 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 7.

Remark 2. The re-randomization of $N \times N$ matrices samples random $N \times N$ matrices, and must compute their inverses. Inverting a random $N \times N$ matrix is impossible with space $o(N^2)$, a consequence of [Raz16]. Our basic construction thus has the sender use $O(N^2)$ space, while the receiver requires only $O(N)$ space. We show, however, how to reduce the space requirements of the sender to $O(N)$ by generating the re-randomization matrices and their inverses using PRFs. The resulting low-sender-space obfuscation scheme is secure, provided the basic construction is a secure (with large sender space) online obfuscation, and the PRF is secure. Details are given in Section 7.2.

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-stamp 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 generate and send a random stream for time-stamping. Afterward, compute and send a standard-model obfuscation of the program P', which takes as input x together with a time-stamp, verifies the time-stamp is valid for x, and then runs P if and only if the stamp is valid.

The intuition for security is that we can invoke the standard-model security of P' to get a simulator S' which just makes black box queries to P'. We then use the security of the time-stamping protocol to conclude that the accepting queries from the simulator, which are those containing valid time stamps, must have been "known" when the time-stamping stream was sent. For any inputs derived from new information sent after the stream concludes, the adversary will not be able to produce a valid time stamp, and thus P' will reject any such inputs. The result is that S' should be simulatable just by making queries to P, and these queries are all made prior to receiving any additional post-stream information.

Unfortunately, turning the above intuition into a full proof appears challenging. One issue is that the obfuscation of P' serves as a verification oracle for checking the validity of time stamps. Existing time stamping security notions offer no guarantees in the presence of a verification oracle, and we do not know if the existing constructions are secure in this setting.

If we were to assume the time-stamping protocol secure even with verification queries, there are still potential problems, mostly revolving around formalizing that the simulator "knows" its input when the time-stamping stream is sent. Indeed, to prove security we need to convert our simulator S' into a simulator S which makes all of its queries by the time the stream concludes, before receiving any additional information. The above intuition would show that S' "knew" these inputs before the stream concludes, but perhaps the inputs (and their time stamps) were hidden inside of the code of S' and only revealed later, after more information is received.

We conjecture that such an S can nevertheless be constructed from S'. The idea is to have S run S' until the time-stamping stream concludes. Then S will try to extract the queries from the state of S' by simulating many possible executions of the remaining security experiment for S' and collecting the queries S' makes to P'. It then uses its assumed time-stamping verification oracle to check which queries have valid time stamps. Since S' can only know a polynomial number of valid time stamps, it seems S should eventually collect all of them. Then it can make these queries to its own oracle for P, and run S' one more time using the answers to P. Unfortunately, formalizing this idea appears tricky, and we leave it as a direction for future work.

1.6 Related Work, Discussion, and Future Directions

Never-before-possible results. The bounded storage model is most often used to eliminate computational assumptions. 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].

Our work shows that there is potentially a rich landscape of applications which leverage the bounded storage model to give results that are *impossible* in the standard model. Our particular applications can all be seen as achieving versions 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 secure (non-interactive) encryption such as [CHK03] always involve updating the secret keys. Our constructions do not require the secret key to be updated. We note that Dziembowski [Dzi06] considers 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. A natural direction for future work is to explore other potential areas besides forward security which may be impossible classically but are achievable in the bounded storage model.

Obfuscation in the bounded storage model. We also initiate the study of obfuscation in the bounded storage model. Just as standard-model obfuscation has proven to be a central tool in the study of standard-model cryptography, our work demonstrates online obfuscation is analogously a central tool in the study of disappearing cryptography. Just as standard-model obfuscation schemes started out as conjectures, with security gradually improved culminating with [JLS20], we hope that future work will improve the status of our candidates.

Besides achieving never-before-possible applications, one advantage of our setting is that we may be able to leverage the bounded storage model to achieve security under milder assumptions than is known for obfuscation in the standard model. Indeed, online obfuscation could plausibly exist *information-theoretically*, and our first construction could plausibly be an instantiation[4]. This gives hope that security can actually be proved unconditionally, without requiring the strong algebraic assumptions needed in the standard model. We leave exploring such information-theoretic security as a fascinating open question.

The quadratic gap. All prior information-theoretic results in the bounded storage model achieve at best an adversary storage that is quadratic in the honest users' storage. Our first candidate construction of an online obfuscator, being plausibly information-theoretic, inherits this quadratic gap. While some negative results are known [DM04], it remains open whether this quadratic "gap" is necessary. While our constructions are probably impractical due to the reliance on obfuscation techniques, such a quadratic gap may be meaningful in practice: for example, if the honest users' storage is 16GB, then security would be maintained against adversaries with $\sim$5ZB, which is on the order of the total data center storage capacity world-wide in 2021 [Mli21]. On the other hand, using computational assumptions, it is possible to get an improved "gap" for time-stamping, and our second construction built from time-stamping can similarly be obtained with an arbitrarily-large polynomial gap.

Other Computational Models. It is possible to achieve classically-impossible results using either hardware assumptions (e.g. [GKR08]) or non-classical laws of physics such as quantum mechanics (e.g. [BB84]). However, as far as we are

[4] The basic large-sender-space version would be purely information-theoretic, whereas the version with low sender space requires only the information-theoretic conjecture together with the existence of one-way functions.

aware, none of these models besides the bounded storage model allows for sending messages that effectively disappear after the transmission is over.

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*. It is similar to the publicly-accessible random string as in [Mau92], but now it is created and sent by one of the parties, and it does not need to be random.

A *stream* $s_{\gg}$ is a long sequence of bits sent sequentially 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[5]. 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, reading the bits one-by-one. This means that the algorithm or program would operate in an online manner - as the streaming happens, it actively reads the stream bit by bit, performs the computation simultaneously, and produces the output in one pass. Since the outputs of such algorithms or programs could have significantly smaller sizes than the stream, while $s_{\gg}$ itself is too large to write down, the short outputs can be reasonably stored. 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*

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

[5] Notice that generating such a stream could still be done using a low memory bound.

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

1. $\mathsf{ExpAdv}_{\mathcal{A},\mathsf{ch},o\mathcal{O}}(C \in \mathcal{C}_\lambda, k)$:
   - 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[6] 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. $\mathsf{ExpSim}_{\mathcal{S},\mathsf{ch},o\mathcal{O}}(C \in \mathcal{C}_\lambda, k, q)$:
   - 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.

The purpose of the interaction round is to allow the challenger to obtain auxiliary information about the circuit C, such as an accepting input. The key feature is that this auxiliary information can be obtained *after* seeing the obfuscated stream, at which point the stream effectively disappears and the adversary can no longer query the program.

We note that in the stream round, we allow the simulator to make adaptive queries. One could also imagine a stronger variant where the simulator can only send a single round of *non-adaptive* queries to the circuit in the stream round. We focus on the weaker version since it suffices for our applications and our VBB impossibility already applies in this setting.

[6] Notice that a fresh stream is sampled every time, so that no single stream is sent repeatedly. One could also imagine a stronger version where the same stream is sent repeatedly, but to achieve that the randomness used must be small. It has also been shown that for learning parities, even just two-pass learning, where the same stream is repeated only once more, has a weaker time-space lower bound than the one-pass one [GRT19] ($\Omega(n^{1.5})$ vs. $\Omega(n^2)$). Therefore, applications are far less plausible in the setting where the same stream is repeated many more times.

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 under memory bound $S(n)$, we require 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 $C \in \mathcal{C}_\lambda$:*

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

where $q = \mathsf{poly}(\lambda)$[7]*.*

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

Remark 4 (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 the full version of the paper that online obfuscators with VBB security do not exist.

Remark 5 (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 6 (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 Public Key Encryption with Disappearing Ciphertext Security

4.1 Definition

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

Essentially, it captures the security game where the adversary is given the private key after all of its queries but before it outputs a guess for the bit b. In traditional models, this definition does not make much sense, as the adversary can simply store the query responses, and then later use the received private

[7] A space $S(n)$ attacker can always run the honest evaluation procedure $S(n)/O(n)$ times in parallel on different inputs, thereby evaluating the program on $S(n)/O(n)$ different points. Thus, the number of queries q the simulator makes must be at least this quantity. One could imagine an alternative definition that sets q to be *exactly* this value. We instead opt for a weaker notion where the simulator is allowed to make an arbitrarily large polynomial number of queries in order to simulate, potentially much larger than S.

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 $\mathsf{ct}_{\gg} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$, 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 3 (Disappearing Ciphertext Security). *Let λ, n be security parameters. A public key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has disappearing ciphertext security under memory bound $S(n)$ if for all PPT adversaries $\mathcal{A}$ that use at most $S(n)$ memory bits:*

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

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

4.2 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 4 (Lossy Function). *Let λ be the security parameter. For $\ell(\lambda) = \mathsf{poly}(\lambda)$ and $k(\lambda) \leq \ell(\lambda)$ (k is referred to as the "lossiness") , a collection of (ℓ, k)-lossy functions is given by a tuple of PPT algorithms (S, F) with the following properties. As short-hands, we have $S_{inj}(\cdot)$ denote $S(\cdot, 1)$ and $S_{lossy}(\cdot)$ denote $S(\cdot, 0)$.*

– ***Easy to sample an injective function**: S_{inj} outputs a function index s, and $F(s,\cdot)$ computes an injective (deterministic) function $f_s(\cdot)$ over the domain $\{0,1\}^{\ell}$.*
– ***Easy to sample a lossy function**: S_{lossy} outputs a function index s, and $F(s,\cdot)$ computes a (deterministic) function $f_s(\cdot)$ over the domain $\{0,1\}^{\ell}$ whose image has size at most $2^{\ell-k}$.*
– ***Hard to distinguish injective mode from lossy mode**: Let X_λ be the distribution of s sampled from S_{inj}, and let Y_λ be the distribution of s sampled from S_{lossy}, the two distributions should be computationally indistinguishable, i.e. $\{X_\lambda\} \overset{c}{\approx} \{Y_\lambda\}$.*

4.3 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.

4.4 Proof of Security

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

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

Sequence of Hybrids

- H_0: The adversary plays the original disappearing ciphertext security game $\mathsf{Dist}^{\mathsf{DisCt}}_{\mathcal{A},\Pi}(\lambda, n)$ where $b = 0$, i.e. it always receives $\mathsf{Enc}(\mathsf{pk}, m_0)$.
- H_1: The same as H_0, except that in $\mathsf{Enc}(\mathsf{pk}, m_b)$, we replace P_{f_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}.

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

For the proofs of the hybrid arguments and the Theorem, please refer to the full version of the paper.

5 Disappearing Signature Scheme

5.1 Definition

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

Put formally, for security parameters λ and n, a disappearing signature scheme consists of a tuple of PPT algorithms $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ 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 $\sigma_{\gg}$. In addition to the standard model signature security (where the adversary has unbounded space), we also require *disappearing signature security* that utilizes 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 additional security requirement for a disappearing signature scheme.

Definition 5 (Disappearing Signature Security). *Let λ, n be security parameters. A disappearing signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ has disappearing signature security 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.*

5.2 Prefix Puncturable Signature

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

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

Definition 6 (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 7 (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 a puncturable signature can be built from any one-way function using certificates, though their basic construction does not satisfy the strong correctness we require: their punctured key yields valid signatures, but not necessarily identical signatures. Nevertheless, it is straightforward to modify the ideas to yield a scheme with the desired correctness. Our modified scheme for prefix puncturable signature can be found in the full version of the paper.

5.3 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.

Theorem 2. *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.*

The proof of this theorem uses some similar techniques as that of Theorem 1 and can be found in the full version of the paper.

6 Functional Encryption

6.1 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 8 (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 9 (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 10 (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:*

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

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).

6.2 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:

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

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

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

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

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

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

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

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

as desired.

Theorem 3. *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.*

The proof of this theorem uses some similar techniques as that of Theorem 1 and can be found in the full version of the paper.

7 Candidate Construction 1

Here, we give a candidate online obfuscation scheme, for NC^1 circuits. This suffices for our applications, provided the underlying building blocks can be computed in NC^1. Note that we might heuristically be able to bootstrap our scheme to all circuits using FHE, but such bootstrapping (e.g. [GGH$^+$13a]) is not known to provably apply to VGB obfuscation. In Section 8, we give a very different construction that directly yields VGB obfuscation.

7.1 Matrix Branching Programs

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

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

Lemma 1 (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 $h = 4^d$.*

7.2 The Basic Framework

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

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

Note that Barrington's theorem implies, for log-depth circuits, that $h = \mathsf{poly}(\lambda)$ and that w can be taken as small as 5. $\mathsf{Convert}$ 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 7.3, where we discuss our particular instantiation of the framework.

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

Construction with Kilian Randomization. We start with the first and simpler scheme, denoted $\mathcal{O}_{\mathrm{Kil}}$, 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. In the full version of the paper, we show two additional schemes that eventually help us carry out the randomization process using only $O(w)$ space. The security of these schemes are equivalent to the security of the construction above, assuming the existence of pseudorandom functions. Therefore, it suffices to analyze the security of the construction above. Next, we will explain how to instantiate $\mathsf{Convert}$ in a way that presumably gives security.

7.3 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], multi-linear 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 to enforce input consistency to prevent mixed-input attacks. 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 $0 \leq \mathsf{w}(x) \leq L$ 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 $\mathsf{w}(x)$, 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 $\mathsf{w}(x)$. 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 $\mathsf{w}(x)$. Notice that $\mathbf{s}'$ has a single 1 at position $(\mathsf{w}(x) \mod L) + 1$. When multiplying $\mathbf{s}'$ by the right bookend $\mathbf{t}$, the result will always be 1, unless $(\mathsf{w}(x) \mod L) + 1 = 1$. The only $\mathsf{w}(x)$ values that satisfy $(\mathsf{w}(x) \mod L) + 1 = 1$ are $\mathsf{w}(x) = 0$ and $\mathsf{w}(x) = L$, 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.[8]

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.

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

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 as BP. The width is now $w + mh$ where $m = \mathsf{poly}(\lambda)$.
- For all $j \in [\ell]$, let S_j be the set of all reads of x_j, i.e. $S_j = \{i | i \in [h], \mathsf{inp}(i) = j\}$. Construct the branching program $\mathsf{BP}_2^{(j)}$ using the BP_2 construction with input length h and $S = S_j$. Overwrite the input function of $\mathsf{BP}_2^{(j)}$ with inp so that it now takes $x \in \{0,1\}^\ell$ as input. Notice that $\mathsf{BP}_2^{(j)}(x) = 0$ if and only if all reads of the j-th bit of x are identical. Sample a fresh random matrix branching program $\mathsf{BP}_*^{(j)}$ with length h, width m, input length ℓ and input function inp. Compute $\mathsf{BP}_3^{(j)} = \mathsf{AND}(\mathsf{BP}_2^{(j)}, \mathsf{BP}_*^{(j)})$. Denote the matrices in $\mathsf{BP}_3^{(j)}$ as $\{\mathbf{M}_{i,b}^{(j)}\}_{i\in[h];b\in\{0,1\}}$, and the bookends as $\mathbf{s}^{(j)}, \mathbf{t}^{(j)}$.
- For all $i \in [h]$, and $b \in \{0,1\}$, construct the matrix $\mathbf{M}'_{i,b}$ by adding all the $\mathbf{M}_{i,b}^{(j)}$'s to the diagonal as $\mathbf{M}'_{i,b} = \mathsf{diag}(\mathbf{M}_{i,b}, \mathbf{M}_{i,b}^{(1)}, \ldots, \mathbf{M}_{i,b}^{(\ell)})$. 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.

8 Candidate Construction 2

Now we present the second candidate construction from digital time-stamping and standard-model obfuscation. The concept of a digital time-stamp was first introduced by Haber and Stornetta [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 using a randomness beacon. A slightly modified definition that uses a stream instead of a randomness beacon will be what we base our candidate construction on.

Definition 11 (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 12 (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 13 (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 $P_{C,k}$:

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

  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 basic 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 input circuit by a constant factor. While no such constructions are currently known, there are also no known impossibilities. 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. Finally, another option is to use the computational time-stamping protocol from [MST04], which shrinks the size of the sketch and the proof, at the cost of relying on computational assumptions. 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

[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.

[BB84] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, 1984.

[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.

[DM04] Stefan Dziembowski and Ueli M. Maurer. On generating the initial key in the bounded-storage model. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 126–137. Springer, Heidelberg, May 2004.

[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.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2008.

[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.

[GRT19] Sumegha Garg, Ran Raz, and Avishay Tal. Time-space lower bounds for two-pass learning. In *34th Computational Complexity Conference (CCC 2019).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[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.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003, 2020. https://eprint.iacr.org/2020/1003.

[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.

[Mli21] Kimberly Mlitz. Data center storage capacity worldwide from 2016 to 2021, by segment, 2021. https://www.statista.com/statistics/638593/worldwide-data-center-storage-capacity-cloud-vs-traditional/.

[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. https://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.

[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.