Memory Lower Bounds of Reductions Revisited

Yuyu Wang[1,2,3], Takahiro Matsuda[2], Goichiro Hanaoka[2], and Keisuke Tanaka[1]

[1] Tokyo Institute of Technology, Tokyo, Japan
wang.y.ar@m.titech.ac.jp, keisuke@is.titech.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
t-matsuda@aist.go.jp, hanaoka-goichiro@aist.go.jp
[3] IOHK, Tokyo, Japan

Abstract. In Crypto 2017, Auerbach et al. initiated the study on memory-tight reductions and proved two negative results on the memory-tightness of restricted black-box reductions from multi-challenge security to single-challenge security for signatures and an artificial hash function. In this paper, we revisit the results by Auerbach et al. and show that for a large class of reductions treating multi-challenge security, it is impossible to avoid loss of memory-tightness unless we sacrifice the efficiency of their running-time. Specifically, we show three lower bound results. Firstly, we show a memory lower bound of natural black-box reductions from the multi-challenge unforgeability of unique signatures to any computational assumption. Then we show a lower bound of restricted reductions from multi-challenge security to single-challenge security for a wide class of cryptographic primitives with unique keys in the multi-user setting. Finally, we extend the lower bound result shown by Auerbach et al. treating a hash function to one treating any hash function with a large domain.

Keywords. memory, tightness, lower bound, uniqueness, black-box reduction

1 Introduction

1.1 Background

Security proofs for cryptographic primitives are typically supported by the black-box reduction paradigm. A black-box reduction $\mathcal{R}$, which is a probabilistic polynomial-time (PPT) algorithm, allows us to convert an adversary $\mathcal{A}$ against some security game (or we say problem) GM_1 into an algorithm $\mathcal{R}^{\mathcal{A}}$ against another security game GM_2. If breaking GM_2 is believed to be hard, then the existence of $\mathcal{R}$ implies the security of GM_1. The quality of $\mathcal{R}$ depends on its tightness, which measures how close the performances of $\mathcal{A}$ and $\mathcal{R}^{\mathcal{A}}$ are. The tighter a reduction is, the larger class of adversaries can be ruled out. Tightness traditionally takes running-time and success probability into account. However, Auerbach et al. [1] observed that some types of reductions, which are tight in common sense, are memory-loose, meaning that they incur large increase in memory usage when converting adversaries. For example, suppose that $\mathcal{A}$ use t_1

time steps and m_1 memory units, and succeed with probability ϵ_1 in GM_1. Even if $\mathcal{R}^{\mathcal{A}}$ can succeed in GM_2 with probability $\epsilon_2 \approx \epsilon_1$ by using $t_2 \approx t_1$ time steps, it may use $m_2 \gg m_1$ memory units. If the security of GM_2 is memory-sensitive, i.e., it can be broken more quickly with large memory than small memory (when the running-time of $\mathcal{A}$ is reasonably long), then a memory-loose reduction does not rule out as many attacks as expected. Recall the instance about the learning parities with noise (LPN) problem in dimension 1024 and error rate 1/4 in [1]. A memory-loose reduction from some security game to this problem only ensures that adversaries running in time less than 2^{85} cannot succeed in the game. There are many memory-sensitive problems besides the LPN problem, such as factoring, discrete-logarithm in prime fields, learning with errors, approximate shortest vector problem, short integer solution, t-collision-resistance (CR_t) where $t > 2$, etc., as noted in [1]. When proving security of cryptographic primitives based on these problems, memory-tightness should be seriously taken into account.

Memory lower bound of restricted reductions. Auerbach et al. initiated the study on memory-tightness, and provided general techniques helping achieve memory-tight reductions. Surprisingly, as negative results, they showed a memory lower bound of reductions from multi-challenge unforgeability (mUF) to standard unforgeability (UF) for signatures. The former security notion is defined in exactly the same way as the latter except that it gives an adversary many chances to produce a valid forgery rather than one chance. Although it is trivial to reduce mUF security to UF security tightly in both running-time and success probability, Auerbach et al. showed that some class of reductions between these two security notions inherently and significantly increase memory usage, unless they sacrifice the efficiency of the running-time. Specifically, they proved that such a reduction must consume roughly $\Omega(q/(p+1))$ bits of memory, where $2q$ is the number of queries made by an adversary and p is the number of times an adversary is run. The class of black-box reductions they treated is restricted, in the sense that a reduction $\mathcal{R}$ only runs an adversary $\mathcal{A}$ sequentially from beginning to end, and is not allowed to rewind $\mathcal{A}$. Moreover, $\mathcal{R}$ only forwards the public keys and signing queries between its challenger and $\mathcal{A}$, and the forgery made by $\mathcal{R}$ should be amongst the ones generated by $\mathcal{A}$. This result implies that in practice, UF security and mUF security may not really be equivalent. As an open problem left by Auerbach et al., it is not clear whether this result holds when a reduction does not respect the restrictions. Moreover, this result does not rule out the possibility that there exists a memory-tight restricted reduction that directly derives mUF security from some memory-sensitive problem. Therefore, it is desirable to clarify whether there exists a memory lower bound of any natural reduction from mUF security to any common assumption.

Auerbach et al. also showed another similar lower bound of restricted reductions from multi-challenge t-collision-resistance (mCR_t) to standard CR_t security for an artificial hash function that truncates partial bits of its input. Here, both security notions prevent an adversary from finding a t-collision (i.e., outputting t distinct elements having the same hash value), while the mCR_t (respectively, CR_t) game allows an adversary to have many chances (respectively, only one chance) to find a t-collision. Since CR_t security is memory-sensitive, this result indicates that breaking mCR_t security might be much easier than breaking CR_t security in practice. However, since the hash function they considered is specific and not collision-resistant, it is still not clear whether this result holds for collision-resistant hash functions.

Finally, it is desirable to clarify whether there exist memory lower bounds for cryptographic primitives in other settings, which are potentially based on memory-sensitive problems.

1.2 Our Results

We revisit memory-tightness on black-box reductions, and show several lower bound results.

Lower bound for unique signatures. In [6], Coron proved a tightness lower bound of black-box reductions from the security of unique signatures [10,20,19], in which there exists only one valid signature for each pair of public key (not necessarily output by the key generation algorithm) and message, to any non-interactive (computational) assumption. Later, Kakvi and Kiltz [15] and Bader et al. [4] respectively fixed a flaw in the proof and improved the bound. The reductions considered in these works are "natural" reductions, in the sense that they run adversaries only sequentially.

Although the study on the tightness of reductions for unique signatures has a long history, memory-tightness of such reductions has never been taken into account until [1], and it is still unclear, when considering natural reductions or reducing the security of unique signatures to common assumptions, whether memory-tightness is achievable. In our work, we focus on natural reductions for unique signatures from the angle of memory, and prove that loss of memory-tightness is inevitable when reducing their mUF security to computational assumptions. Specifically, we show the existence of a memory lower bound of any natural reduction from the mUF security of unique signatures to any computational assumption (rather than only UF security).[4] Here, a natural black-box reduction can interact with its challenger in any way it wants, and can adaptively rewind an adversary. We do not allow reductions to modify the internal state of an adversary, which is a very natural restriction. Similarly to [1], the bound is roughly $\Omega(q/(p+1))$ bits of memory, where $2q$ is the number of queries made by an adversary and p is the number of times an adversary is rewound. This result indicates that for a unique signature scheme, any natural reduction from its mUF security to a memory-sensitive problem may not rule out as many attacks as expected. Therefore, when using a unique signature scheme based on a memory-sensitive problem in practice, one should make its security parameter larger than indicated by traditional security proofs. As far as we know, this is the

[4] Note that all the memory-sensitive problems discussed in [1] fall under the notion of computational assumptions.

first negative result on memory-tight reductions to any computational assumptions, and also the first one treating memory-tightness of natural reductions.

Moreover, we give our result in a generalized way so that it also captures some other assumptions that do not fall under the definition of computational assumptions. By slightly modifying our proof, we can also show memory lower bounds for the notions of verifiable unpredictable functions (VUFs) and re-randomizable signatures, which are more general primitives and hence capture more instantiations (e.g., [20,19,23,13]).

Lower bound for unique-key primitives in the multi-user setting. Security notions of cryptographic primitives are usually considered in the single-user setting, where an adversary only sees one challenge public key. However, in practice, an attacker may see many public keys and adaptively corrupt secret keys. Hence, considering security of primitives in the multi-user setting [2,3] is necessary. In [4], Bader et al. showed that in this setting, it is impossible to avoid loss of tightness when deriving the security of unique-key primitives, in which there exists only one valid secret key for each public key, from non-interactive assumptions.

In this work, we give the first negative result on memory-tightness in the multi-user setting. Specifically, we show a memory lower bound of restricted black-box reductions from multi-challenge one-wayness in the multi-user setting ($\mathsf{mU\text{-}mOW}$) to standard one-wayness in the multi-user setting ($\mathsf{mU\text{-}OW}$) for unique-key relations. Compared with [1], the reductions we treat are less restricted. We only require them to forward the public keys and corruption queries between the challengers and adversaries, while they can forge secret keys in any way they want (i.e., a forgery is not necessarily amongst the ones output by an adversary). The bound is roughly $\Omega(\max\{q/(p+2), n/(p+2)\})$, where $2q$ is the number of queries, n is the number of users, and p is the number of rewinding procedures. Since unique-key relations are very fundamental primitives, from this result, we can easily derive lower bounds for a large class of primitives with unique keys (including public key encryption (PKE) schemes, signatures, trapdoor commitment schemes (with collision-resistance), etc.), which capture many constructions (e.g., [22,8,5,12,8,7,17,23,14,18]). These results imply that for unique-key primitives in the multi-user setting, the gaps between their multi-challenge security notions and single-challenge security notions might be wider than indicated by conventional security proofs via restricted reductions.

As a by-product result, our result can be extended for primitives with re-randomizable keys [4], where secret keys can be efficiently re-randomized and the distribution of a re-randomized key is uniform.

Lower bound for large-domain hash functions. Finally, we revisit the memory lower bound of restricted reductions from mCR_t security to CR_t security for an artificial hash function shown in [1]. We firstly show a streaming lower bound for all the CR_t secure large-domain hash functions. Specifically, we show that determining whether there exists a t-collision in a data stream consumes large memory. Following from this fact, we extend the result in [1] to a lower bound for all the large-domain hash functions. Here, a hash function is said to have a large domain if its range is negligibly small compared with its domain (e.g., $\mathsf{H} : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^{\lambda}$ where λ is the security parameter). It is a natural property satisfied by most practical hash functions. The bound is roughly $\Omega(\min\{(q-\kappa)/(p+1)\})$ where q is the number of queries, κ is the length of the hash key, and p is the number of rewinding procedures. Since CR_t security (where $t > 2$) is memory-sensitive, this result implies that for any natural hash function, its mCR_t security directly derived from its CR_t security via restricted reductions does not rule out as many attacks as its CR_t security does in practice.

1.3 High-Level Ideas

Like in [1], our lower bound for unique signatures follows from a streaming lower bound result implying that determining the output of a specific function $G(\boldsymbol{y})$ consumes large memory. Here, $\boldsymbol{y}$ is a data stream that does not occupy local memory and can be accessed sequentially. We construct an inefficient adversary $\mathcal{A}_{\boldsymbol{y}}$ (storing $\boldsymbol{y}$) breaking the mUF security of any unique signature scheme iff $G(\boldsymbol{y}) = 1$. Let $\mathcal{R}$ be a black-box reduction from mUF security to a cryptographic game GM. $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$ is likely to succeed in GM when $G(\boldsymbol{y}) = 1$. On the other hand, when $G(\boldsymbol{y}) = 0$, we use the meta-reduction method to show that $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$ will fail. Roughly, we construct a PPT simulator $\mathcal{S}_{\boldsymbol{y}}$ that is indistinguishable from $\mathcal{A}_{\boldsymbol{y}}$ due to uniqueness. If $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$ succeeds in GM, then the PPT algorithm $\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}$ succeeds in GM as well, which gives us the conflict. As a result, we can obtain an algorithm that determines $G(\boldsymbol{y})$ with high probability by simulating the game GM and $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$. Such an algorithm must consume large memory due to the streaming lower bound. Moreover, $\mathcal{A}_{\boldsymbol{y}}$ can be simulated by accessing the stream $\boldsymbol{y}$ with small memory usage. Therefore, $\mathcal{R}$ must use large memory if simulating the challenger in GM does not consume large memory. This is the case in most computational assumptions (including all the memory-sensitive problem noted in [1]), where the challenger saves an answer, which only occupies small memory, when sampling a challenge, and checks whether the final output of an adversary is equal to that answer.

The lower bound of restricted reductions from mU-mOW security to mU-OW security for unique-key primitives is shown in a similar way by constructing an inefficient adversary and its simulator in the mU-mOW game. However, in this case, we face a problem that it consumes large memory to store public keys of users when running the mU-OW and mU-mOW games. This spoils our result since the streaming lower bound does not imply that $\mathcal{R}$ consumes large memory any more. We deal with this problem by running a pseudorandom function (PRF) to simulate random coins used to generate public keys, which is similar to the technique used in [1] for achieving memory-tightness. Whenever a public key is needed, we only have to run the PRF to obtain the corresponding random coin and generate the key again, and hence there is no need to store public keys any more. Here, it might seem that outputs of PRF are not indistinguishable from real random coins since an inefficient adversary is involved in the interaction.

However, we can show that the adversary can be simulated in polynomial-time (PT) due to the uniqueness of secret keys.

Extending the lower bound result for a specific hash function in [1] to all the large-domain hash functions satisfying CR_t security (where t is a constant) involves three steps. Firstly, we prove a theorem saying that for a large-domain hash function satisfying CR_t security, there exist many hash values with more than t pre-images. Intuitively, for a large-domain hash function (using randomly chosen key), if there are few hash values with more than t pre-images, then there should exist some hash value with many pre-images. We prove that the set of all pre-images of such a hash value is so large that t randomly chosen inputs are very likely to fall into this set, which conflicts with CR_t security. Therefore, we conclude that a CR_t secure large-domain hash function should have many hash values with more than t pre-images. Then by exploiting this theorem and the technique used in previous works [16,21,1], we prove the existence of a memory lower bound for determining whether there exists a t-collision in a stream, based on the disjointness problem [16,21]. Following from this result, we achieve a memory lower bound of restricted reductions from mCR_t security to CR_t security for large-domain hash functions.

1.4 Outline of This Paper

In Section 2, we recall some notation and describe the computational model and data stream model. In Section 3, we show a lower bound of black-box reductions from the mUF security of unique signatures to cryptographic games. In Section 4, we show a lower bound of restricted reductions from $\mathsf{mU\text{-}mOW}$ security to $\mathsf{mU\text{-}OW}$ security for unique-key cryptographic primitives. In Section 5, we show a lower bound of restricted reductions from mCR_t security to CR_t security for large-domain hash functions.

2 Preliminaries

In this section, we give several terminologies that are necessary to describe our results, describe the computational model and data stream model, and recall the disjointness problem and a streaming lower bound.

2.1 Notation and Computational Model.

In this paper, all algorithms are RAMs having access to memory and registers that each holds one word. Rewinding random bits used by RAMs is not permitted, so if an algorithm wants to access previously used random bits it must store them. If $\mathcal{A}$ is a deterministic (respectively, probabilistic) algorithm, then $y = \mathcal{A}(x)$ (respectively, $y \leftarrow \mathcal{A}(x)$) means that $\mathcal{A}$ takes as input x and outputs y. By $\mathcal{A}^{\mathcal{O}}$ we mean that $\mathcal{A}$ has access to an oracle $\mathcal{O}$. By $\mathcal{A}_z$ we mean that z is stored in the memory of $\mathcal{A}$. We denote the code and memory consumed by $\mathcal{A}$ (but not its oracle) by $\mathbf{LocalMem}(\mathcal{A})$, where the consumption is measured in bits. *negl* denotes an unspecified negligible function. If $\mathcal{Z}$ is a finite set, then $|\mathcal{Z}|$ denotes the number of (distinct) elements in $\mathcal{Z}$, and $z \leftarrow \mathcal{Z}$ denotes the process of sampling z at uniformly random from $\mathcal{Z}$.

2.2 Data Stream Model

Now we recall stream oracles. To a stream oracle, an algorithm is allowed to make queries to access a large stream of data sequentially, while the local memory consumption remains small. We adopt the notation in [1] to describe stream oracles as follows.

A stream oracle $\mathcal{O}_{\boldsymbol{y}}$ is parameterized by a vector $\boldsymbol{y} = (y_1, \cdots, y_n) \in \mathcal{U}^n$ where $\mathcal{U}$ is some finite set. Whenever receiving a query, $\mathcal{O}_{\boldsymbol{y}}$ runs $i = i+1 \mod n$ (where i is initialized with 0), and returns y_i. Let q be the total number of queries. *The number of passes* is defined as $p = \lceil q/n \rceil$.

2.3 Disjointness Problem and Streaming Lower Bound

Now we recall the disjointness problem, which derives streaming lower bounds.

Theorem 1 ([16,21]). *Let $x_1, x_2 \in \{0,1\}^n$ and $\mathsf{DISJ}(x_1, x_2)$ be defined by*

$$\mathsf{DISJ}(x_1, x_2) = \begin{cases} 1 & \text{if } \exists i : x_1[i] = x_2[i] = 1 \\ 0 & \text{otherwise} \end{cases},$$

where $x_b[j]$ denotes the jth bit of x_b for $j \in \{1, \cdots, n\}$ and $b \in \{0,1\}$. Then any two-party protocol (P_1, P_2), such that $\Pr[\mathsf{DISJ}(x_1, x_2) \leftarrow (P_1(x_1) \leftrightarrows P_2(x_2))] \geq c$ holds for some constant $c > 1/2$ and every $(x_1, x_2) \in \{0,1\}^n$, must have communication $\Omega(n)$ in the worst case. Here, by $\mathsf{DISJ}(x_1, x_2) \leftarrow (P_1(x_1) \leftrightarrows P_2(x_2))$ we mean that the interaction between P_1 and P_2 respectively on input x_1 and x_2 outputs $\mathsf{DISJ}(x_1, x_2)$.

In [1], Auerbach et al. gave a streaming lower bound result, which is a corollary of prior works [16,21] based on the disjointness problem. It shows that determining whether the second half of a stream contains an element not in the first half requires large memory. We now follow [1] to define $G(\boldsymbol{y})$, where $\boldsymbol{y} = \boldsymbol{y}_1 || \boldsymbol{y}_2$ and $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathcal{U}^q$, and recall the streaming lower bound.

$$G(\boldsymbol{y}) = \begin{cases} 1 & \text{if } \exists j \ \forall i : \boldsymbol{y}_2[j] \neq \boldsymbol{y}_1[i] \\ 0 & \text{otherwise} \end{cases}.$$

Theorem 2. *Let $\mathcal{B}$ be a probabilistic algorithm and λ be a (sufficiently large) security parameter. Assuming that there exists some constant $c > 1/2$ such that $\Pr[\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda) = G(\boldsymbol{y})] \geq c$ holds for polynomials $q = q(\lambda)$ and $n = n(\lambda)$, and all $\boldsymbol{y} \in (\{0,1\}^n)^{2q}$ (respectively, $\boldsymbol{y} \in (\{i\}_{i=1}^n)^{2q}$). Then we have* $\mathbf{LocalMem}(\mathcal{B}) = \Omega(\min\{q/p, 2^n/p\})$ *(respectively,* $\mathbf{LocalMem}(\mathcal{B}) = \Omega(\min\{q/p, n/p\})$*), where p is the number of passes $\mathcal{B}$ makes in the worst case.*

The above theorem is slightly different from the one in [1], in the sense that we let $\boldsymbol{y} \in (\{0,1\}^n)^{2q}$ or $\boldsymbol{y} \in (\{i\}_{i=1}^n)^{2q}$ (instead of $\boldsymbol{y} \in \mathcal{U}^{2q}$ for all sufficiently large q and $|\mathcal{U}|$), and require q and n be polynomials in λ. However, the proof for the streaming lower bound in [1, Appendix A] can be directly applied to prove the above theorem. We refer the reader to [1, Appendix A] for details.

3 Lower Bound of Reductions from the mUF Security of Unique Signatures to Cryptographic Games

In this section, we show a memory lower bound of black-box reductions from the mUF security of unique signatures to assumptions captured by cryptographic games. We start by recalling the definition of unique signatures and mUF security, and then show the lower bound.

3.1 Unique Signatures and mUF Security

We now recall the definition of (digital) signatures.

Definition 1 (Digital signature). *A signature scheme consists of PT algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$*. (a)* Gen *is a probabilistic algorithm that takes as input* 1^λ*, and returns a public/secret key pair* (pk, sk)*. (b)* Sign *is a probabilistic algorithm that takes as input a secret key sk and a message* $m \in \{0,1\}^\delta$ *where* $\delta = \delta(\lambda)$ *is some polynomial, and returns a signature* σ*. (c)* Verify *is a deterministic algorithm that takes as input a public key pk, a message m, and a signature* σ*, and returns* 1 *(accept) or* 0 *(reject).*

A signature scheme is required to satisfy correctness, which means that $\mathsf{Verify}_{pk}(m, \sigma) = 1$ *holds for all* $\lambda \in \mathbb{N}$*, all* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*, all* $m \in \{0,1\}^\delta$*, and all* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$*.*

Next we recall the definition of unique signatures, in which there exists only one valid signature for each pair of public key (not necessarily output by $\mathsf{Gen}(1^\lambda)$) and message.

Definition 2 (Unique signature [19]). *A signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be a* unique signature scheme *if for all* $\lambda \in \mathbb{N}$*, all pk (possibly outside the support of* Gen*), and all* $m \in \{0,1\}^\delta$*, there exists no pair* (σ, σ') *that simultaneously satisfies* $\sigma \neq \sigma'$ *and* $\mathsf{Verify}_{pk}(m, \sigma) = \mathsf{Verify}_{pk}(m, \sigma') = 1$*.*

Now we recall mUF security. In the mUF game, an adversary has many chances to produce a valid forgery rather than one chance. Although mUF security can be tightly reduced to UF security straightforwardly in common sense, it is shown in [1] that restricted reductions between these two security notions inherently require increased memory usage.

Definition 3 (mUF [1]). *A signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be* mUF secure *if for any PPT adversary* $\mathcal{A}$*, we have* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mUF}}(\lambda) = \Pr[\mathcal{CH}$ *outputs* $1] \leq negl(\lambda)$ *in the following game.*

1. *The challenger $\mathcal{CH}$ sets $w = 0$ and $\mathcal{Q} = \emptyset$, samples $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, and runs $\mathcal{A}$ on input $(1^\lambda, pk)$. $\mathcal{A}$ may make adaptive signing and verification queries to $\mathcal{CH}$, and $\mathcal{CH}$ responds as follows:*
   - *On receiving a signing query m, $\mathcal{CH}$ computes $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$, adds m to $\mathcal{Q}$, and sends σ to $\mathcal{A}$.*
   - *On receiving a verification query (m^*, σ^*), if $\mathsf{Verify}_{pk}(m^*, \sigma^*) = 1$ and $m^* \notin \mathcal{Q}$, $\mathcal{CH}$ sets $w = 1$.*
2. *At some point, $\mathcal{A}$ makes a stopping query stp to $\mathcal{CH}$, and $\mathcal{CH}$ returns w.*

The definition of UF security is exactly the same as the above one except that $\mathcal{A}$ is allowed to make only one verification query and the advantage of $\mathcal{A}$ is denoted by $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{UF}}(\lambda)$.

3.2 Lower Bound for Unique Signatures

Before giving the main theorem, we recall the definition of cryptographic games.

Definition 4 (Cryptographic game [11]). *A* cryptographic game GM *consists of a (possibly inefficient) random system (called the challenger) $\mathcal{CH}$ and a constant c. On input security parameter 1^λ, $\mathcal{CH}(1^\lambda)$ interacts with some adversary $\mathcal{A}(1^\lambda)$, and outputs a bit b. This interaction is denoted by $b \leftarrow (\mathcal{A}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))$, and the advantage of $\mathcal{A}$ in GM is $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{GM}}(\lambda) = \Pr[1 \leftarrow (\mathcal{A}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] - c$.*

A cryptographic game $\mathsf{GM} = (\mathcal{CH}, c)$ is said to be secure if for any PPT adversary $\mathcal{A}$, we have $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{GM}}(\lambda) \leq negl(\lambda)$.

All commonly used assumptions and most security games in cryptography fall under the framework of cryptographic games. We call a cryptographic game $\mathsf{GM} = (\mathcal{CH}, c)$ a *computational assumption* if $c = 0$.

Black-box reduction. Now we follow [1] to describe black-box reductions. Unlike in [1], we do not fix the random tape of an adversary, and do not give any restriction on the queries made by a reduction.[5]

Let $\mathcal{R}$ be a black-box reduction from GM_1 to GM_2. We write $\mathcal{R}^{\mathcal{A}}$ to mean that $\mathcal{R}$ has oracle access to a (stateful) adversary $\mathcal{A}$ playing game GM_1. Whenever receiving a query from $\mathcal{R}$, $\mathcal{A}$ returns the "next" query to $\mathcal{R}$. $\mathcal{R}$ is not able to modify the current state of $\mathcal{A}$ (i.e., $\mathcal{A}$ runs sequentially), but is allowed to adaptively rewind $\mathcal{A}$ to previous states.

Definition 5 (c-black-box reduction). *Let GM_1 and GM_2 be cryptographic games and $c > 0$ be a constant. An oracle-access PPT machine $\mathcal{R}^{(\cdot)}$ is said to be a c-*black-box reduction *from GM_1 to GM_2, if for any (sufficiently large) security parameter λ and any (possibly inefficient) adversary $\mathcal{A}$, we have $\mathbf{Adv}^{\mathcal{R}^{\mathcal{A}}}_{\mathsf{GM}_2}(\lambda) \geq c \cdot \mathbf{Adv}^{\mathcal{A}}_{\mathsf{GM}_1}(\lambda)$.*

[5] Auerbach et al. requires a reduction to preserve the advantage of an adversary even if the random tape of the adversary is fixed. However, we observe that this restriction is not necessary in their work as well, which we will discuss after giving the proof.

Like many previous works (e.g., [6,15,4,1]), we do not consider reductions that can modify the current state of an adversary. This is a natural restriction, which is respected by most black-box reductions.

We now give a theorem showing a memory lower bound of c_r-black-box reductions from the mUF security of unique signatures to cryptographic games $\mathsf{GM} = (\mathcal{CH}, c_g)$, where $c_g < 1/2$ and $c_r + c_g > 1/2$. When $c_g = 0$, our result captures c_r-black-box reductions where $c_r > 1/2$ to any computational assumption. When $c_r = 1$, it captures 1-black-box reductions to any cryptographic game such that $c_g < 1/2$.[6]

Theorem 3. *Let λ be a (sufficiently large) security parameter, $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a unique signature scheme with message length δ, $\mathsf{GM} = (\mathcal{CH}, c_g)$ be a secure cryptographic game,* **LocalMem**$(\mathcal{CH})$ *be the amount of memory consumed by $\mathcal{CH}$, and $\mathcal{R}$ be a c_r-black box reduction from the* mUF *security of Σ to the security of* GM*. Let $q = q(\lambda)$ be the maximum numbers of signing queries and verification queries made by an adversary in the* mUF *game. If (a) $\mathcal{R}$ rewinds the adversary for at most $p = p(\lambda)$ times and (b) $c_g < 1/2$ and $c_r + c_g > 1/2$, then we have*

$$\begin{aligned}\textbf{LocalMem}(\mathcal{R}) =& \Omega(\min\{q/(p+1), 2^{\delta}/(p+1)\}) - O(\log q)\\ &- \textbf{LocalMem}(\mathcal{CH}) - \textbf{LocalMem}(\mathsf{Verify}).\end{aligned}$$

Roughly, this theorem implies that when the maximum number of signing queries made by an adversary in the mUF game is very large, $\mathcal{R}$ must consume large memory unless it rewinds $\mathcal{A}$ many times, which increases its running-time.

High-level idea. We firstly construct an inefficient adversary $\mathcal{A}_{\boldsymbol{y}}$ where $\boldsymbol{y} = (y_1, \cdots, y_{2q})$. $\mathcal{A}_{\boldsymbol{y}}$ makes signing queries $y_1, \cdots, y_q$, checks the validity of the answers, and then makes verification queries $(y_{q+1}, \sigma_1^*), \cdots, (y_{2q}, \sigma_q^*)$ which are generated by using brute force. Consider the interaction $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda)$. When $G(\boldsymbol{y}) = 1$ (see Section 2.3 for the definition of G), we have $\{y_{q+i}\}_{i=1}^q \nsubseteq \{y_i\}_{i=1}^q$, which means that $\mathcal{A}_{\boldsymbol{y}}$ is a deterministic algorithm breaking mUF security. Since $\mathcal{R}$ is a black-box reduction, $\mathcal{CH}$ is likely to output 1 in this case. When $G(\boldsymbol{y}) = 0$, we have$\{y_{q+i}\}_{i=1}^q \subseteq \{y_i\}_{i=1}^q$, in which case we can construct a PT algorithm $\mathcal{S}_{\boldsymbol{y}}$ running in the same way as $\mathcal{A}_{\boldsymbol{y}}$ does, except that $\mathcal{S}_{\boldsymbol{y}}$ uses the answers of signing queries as its forgeries instead of exploiting brute force. Due to uniqueness, $\mathcal{S}_{\boldsymbol{y}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$. If $\mathcal{CH}$ outputs 1 with probability that is non-negligibly greater than c_g in the interaction with $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$, then we have a PPT algorithm $\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}$ breaking the security of GM, which gives us the conflict. Therefore, $\mathcal{CH}$ is likely to output 0 when $G(\boldsymbol{y}) = 0$.

Then we can construct an algorithm $\mathcal{B}$ with access to a stream $\boldsymbol{y}$ that simulates the interaction $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda)$ and outputs $G(\boldsymbol{y})$ with high probability. According to Theorem 2, the memory consumed by $\mathcal{B}$ is inherently large (to

[6] There are several typical cryptographic games with $0 < c_g < 1/2$, such as recipient-anonymity for IBE schemes [9] and one-wayness for encryption schemes with constantly large message spaces.

some extent). Moreover, although $\mathcal{A}_{\boldsymbol{y}}$ consumes a large amount of memory to store $\boldsymbol{y}$, $\mathcal{B}$ does not have to use large memory when simulating $\mathcal{A}_{\boldsymbol{y}}$ by accessing its stream. As a result, if the memory consumed by $\mathcal{CH}$ is small (which is often the case in computational assumptions), then $\mathcal{R}$ must consume large memory.

Proof (of Theorem 3). Assuming the existence of the reduction $\mathcal{R}$ stated in Theorem 3, we show the existence of a probabilistic algorithm $\mathcal{B}$ such that $\Pr[G(\boldsymbol{y}) \leftarrow \mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda)] \geq c$ for all $\boldsymbol{y} = (y_1, \cdots, y_{2q}) \in (\{0,1\}^\delta)^{2q}$ and some constant $c > 1/2$ by giving hybrid games.

Game 0: In this game, $\mathcal{R}$ has access to an adversary $\mathcal{A}_{\boldsymbol{y}}$ and interacts with $\mathcal{CH}$, where $\mathcal{A}_{\boldsymbol{y}}$ runs as follows.

1. On receiving $(1^\lambda, pk)$, $\mathcal{A}_{\boldsymbol{y}}$ stores $(1^\lambda, pk)$ and makes a signing query y_1.
2. For $i = 1, \cdots, q-1$, on receiving the answer σ_i to the ith signing query, if $\mathsf{Verify}_{pk}(y_i, \sigma_i) \neq 1$, $\mathcal{A}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{A}_{\boldsymbol{y}}$ makes a signing query y_{i+1}.
3. On receiving the answer σ_q to the qth signing query, if $\mathsf{Verify}_{pk}(y_q, \sigma_q) \neq 1$, $\mathcal{A}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{A}_{\boldsymbol{y}}$ exhaustively searches σ_1^* such that $\mathsf{Verify}_{pk}(y_{q+1}, \sigma_1^*) = 1$, and makes a verification query (y_{q+1}, σ_1^*).
4. For $i = 1, \cdots, q-1$, when invoked (with no input) for the ith time, $\mathcal{A}_{\boldsymbol{y}}$ exhaustively searches σ_{i+1}^* such that $\mathsf{Verify}_{pk}(y_{q+i+1}, \sigma_{i+1}^*) = 1$, and makes a verification query $(y_{q+i+1}, \sigma_{i+1}^*)$.
5. When invoked (with no input) for the qth time, $\mathcal{A}_{\boldsymbol{y}}$ makes a stopping query *stp*.

We now show the following lemma.

Lemma 1. $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c$ *for all* $\boldsymbol{y} \in (\{0,1\}^\delta)^{2q}$ *and some constant* $c > 1/2$ *in* **Game 0**.

Proof (of Lemma 1). Firstly, we show the existence of a PT algorithm $\mathcal{S}_{\boldsymbol{y}}$ perfectly simulating $\mathcal{A}_{\boldsymbol{y}}$ on condition that $G(\boldsymbol{y}) = 0$. $\mathcal{S}_{\boldsymbol{y}}$ runs in the same way as $\mathcal{A}_{\boldsymbol{y}}$ except that it uses the answers of the signing queries as its verification queries. Formally, it runs as follows. (Below, the difference from $\mathcal{A}_{\boldsymbol{y}}$ is *emphasized*.)

1. On receiving $(1^\lambda, pk)$, $\mathcal{S}_{\boldsymbol{y}}$ stores $(1^\lambda, pk)$ and makes a signing query y_1.
2. For $i = 1, \cdots, q-1$, on receiving the answer σ_i to the ith signing query, if $\mathsf{Verify}_{pk}(y_i, \sigma_i) \neq 1$, $\mathcal{S}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{S}_{\boldsymbol{y}}$ *stores* (y_i, σ_i) *in its internal list* L *(initialized with* $\emptyset$*)*, and makes a signing query y_{i+1}.
3. On receiving the answer σ_q to the qth signing query, if $\mathsf{Verify}_{pk}(y_q, \sigma_q) \neq 1$, $\mathcal{S}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{S}_{\boldsymbol{y}}$ *stores* (y_q, σ_q) *in* L*, searches a pair* (m, σ) *in* L *such that* $m = y_{q+1}$*, and makes a verification query* (m, σ)*. If the searching procedure fails,* $\mathcal{S}_{\boldsymbol{y}}$ *aborts.*
4. For $i = 1, \cdots, q-1$, when invoked (with no input) for the ith time, $\mathcal{S}_{\boldsymbol{y}}$ *searches a pair* (m, σ) *in* L *such that* $m = y_{q+i+1}$*, and makes a verification query* (m, σ)*. If the searching procedure fails for some* i*,* $\mathcal{S}_{\boldsymbol{y}}$ *aborts.*
5. When invoked (with no input) for the qth time, $\mathcal{S}_{\boldsymbol{y}}$ makes a stopping query *stp*.

When $G(\boldsymbol{y}) = 0$, we have $\{y_{q+i}\}_{i=1}^{q} \subseteq \{y_i\}_{i=1}^{q}$, which means that the searching procedures executed by $\mathcal{S}_{\boldsymbol{y}}$ (in Steps 3 and 4) will not fail. Moreover, due to the uniqueness of Σ, the verification queries made by $\mathcal{S}_{\boldsymbol{y}}$ are exactly the same as those made by $\mathcal{A}_{\boldsymbol{y}}$. Hence, $\mathcal{S}_{\boldsymbol{y}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$ in the view of $\mathcal{R}$.

Due to the security of GM, we have $\mathbf{Adv}_{\mathsf{GM}}^{\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}}(\lambda) = \mathbf{Adv}_{\mathsf{GM}}^{\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}}(\lambda) \leq negl(\lambda)$ when $G(\boldsymbol{y}) = 0$, which implies $\Pr[1 \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda)) \mid G(\boldsymbol{y}) = 0] - c_g \leq negl(\lambda)$, i.e., $\Pr[0 \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda)) \mid G(\boldsymbol{y}) = 0] \geq 1 - c_g - negl(\lambda)$. On the other hand, when $G(\boldsymbol{y}) = 1$, there exists some $1 \leq j \leq q$ such that $y_{q+j} \notin \{y_i\}_{i=1}^{q}$, which implies $\mathbf{Adv}_{\mathsf{mUF}}^{\mathcal{A}_{\boldsymbol{y}}}(\lambda) = 1$. Since $\mathcal{R}$ is a c_r-black-box reduction, we have $\Pr[1 \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda)) \mid G(\boldsymbol{y}) = 1] - c_g \geq c_r$. Since $c_g < 1/2$, $c_r + c_g > 1/2$, and λ *is sufficiently large*, there exists some constant $c > 1/2$ such that $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c$ for all $\boldsymbol{y} \in (\{0,1\}^\delta)^{2q}$, completing the proof of Lemma 1. □

Game 1: This game is exactly the same as **Game 0** except that there exists an algorithm $\mathcal{A}'$ with access to the stream oracle $\mathcal{O}_{\boldsymbol{y}}$ simulating $\mathcal{A}_{\boldsymbol{y}}$ as follows. (Below, the difference from $\mathcal{A}_{\boldsymbol{y}}$ is *emphasized*.)

1. On receiving $(1^\lambda, pk)$, $\mathcal{A}'$ stores $(1^\lambda, pk)$, *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_1, and makes a signing query y_1.
2. For $i = 1, \cdots, q-1$, on receiving the answer σ_i to the ith signing query, if $\mathsf{Verify}_{pk}(y_i, \sigma_i) \neq 1$, $\mathcal{A}'$ aborts. Otherwise, $\mathcal{A}'$ *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_{i+1} and makes a signing query y_{i+1}.
3. On receiving the answer σ_q to the qth signing query, if $\mathsf{Verify}_{pk}(y_q, \sigma_q) \neq 1$, $\mathcal{A}'$ aborts. Otherwise, $\mathcal{A}'$ *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_{q+1}, exhaustively searches σ_1^* such that $\mathsf{Verify}_{pk}(y_{q+1}, \sigma_1^*) = 1$, and makes a verification query (y_{q+1}, σ_1^*).
4. For $i = 1, \cdots, q-1$, when invoked (with no input) for the ith time, $\mathcal{A}'$ *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_{q+i+1}, exhaustively searches σ_{i+1}^* such that $\mathsf{Verify}_{pk}(y_{q+i+1}, \sigma_{i+1}^*) = 1$, and makes a verification query $(y_{q+i+1}, \sigma_{i+1}^*)$.
5. When invoked (with no input) for the qth time, $\mathcal{A}'$ makes a stopping query *stp*.

Whenever $\mathcal{R}$ executes a rewinding procedure, $\mathcal{A}'$ makes another pass on its stream so that it can access the message for the next signing or verification query. Since $\mathcal{A}'^{\mathcal{O}_{\boldsymbol{y}}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$, we immediately obtain the following lemma.

Lemma 2. $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}'^{\mathcal{O}_{\boldsymbol{y}}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c$ *for all* $\boldsymbol{y} \in (\{0,1\}^\delta)^{2q}$ *and some constant* $c > 1/2$ *in* **Game 1**.

Game 2: This game is exactly the same as **Game 1** except that there exists a stream-access algorithm $\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}$ that simulates $\mathcal{CH}$, $\mathcal{R}$, and $\mathcal{A}'^{\mathcal{O}_{\boldsymbol{y}}}$ and returns the output of $\mathcal{CH}$. Since the view of $\mathcal{R}$ does not change at all, we have the following lemma.

Lemma 3. $\Pr[G(\boldsymbol{y}) \leftarrow \mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda)] \geq c$ *for all* $\boldsymbol{y} \in (\{0,1\}^\delta)^{2q}$ *and some constant* $c > 1/2$ *in* **Game 2**.

Since $\mathcal{B}$ makes $p+1$ passes on its stream in total, according to Theorem 2 and Lemma 3, we have

$$\mathbf{LocalMem}(\mathcal{B}) = \Omega(\min\{q/(p+1)\}, 2^{\delta}/(p+1)).$$

Furthermore, the memory used to simulate $\mathcal{CH}$ and $\mathcal{A}'$ is $O(\log q) + \mathbf{LocalMem}(\mathcal{CH}) + \mathbf{LocalMem}(\mathsf{Verify})$, where $O(\log q)$ is the amount of memory used to record q and the index of the next query $\mathcal{A}'$ will make. Therefore, we have

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{B}) = {} & O(\mathbf{LocalMem}(\mathcal{R})) + O(\log q) \\ & + \mathbf{LocalMem}(\mathcal{CH}) + \mathbf{LocalMem}(\mathsf{Verify}).\end{aligned}$$

Combining the above two bounds completes the proof of Theorem 3. $\square$

Remark on security parameter. Theorem 3 holds only when the security parameter λ is sufficiently large, while one may wonder why memory-tightness makes sense when λ is already required to be very large. In fact, λ only has to be large enough to ensure $c_g + \mathbf{Adv}_{\mathsf{GM}}^{\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}}(\lambda) < 1/2$ in the proof of Lemma 1. When c_g is small (e.g., $c_g = 1/4$), it is obvious that $c_g + \mathbf{Adv}_{\mathsf{GM}}^{\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}}(\lambda) < 1/2$ should hold even if λ is small (to some extent) and $\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}$ may consume large memory, due to the security of GM. Therefore, λ is not necessarily very large unless c_g is very close to $1/2$.

Remark on advantage-preserving reductions. In [1], it is required that the black-box reductions are advantage-preserving, which means that they should work well for adversaries with fixed random tapes. However, we observe that this restriction is not necessary. The reason is that we can treat adversaries with fixed random tapes as deterministic ones, for which any black-box reduction should work well. Furthermore, although a deterministic adversary consumes large memory in this case (compared with an adversary with fixed random tape), simulating it with stream does not, hence our result is not spoiled. The same argument is made for our results in other sections.

Remark on reductions to UF security. Auerbach et al. [1] showed a lower bound on the memory usage of restricted reductions from mUF security to UF security. A restricted reduction forwards the public keys generated by $\mathcal{CH}$ to $\mathcal{A}_{\boldsymbol{y}}$, and forwards the signing queries $\boldsymbol{y}$ and one of the forgery made by $\mathcal{A}_{\boldsymbol{y}}$ to the challenger $\mathcal{CH}$ in the UF game. One can see that $\mathcal{CH}$ uses large memory to store $\boldsymbol{y}$ so that it can check whether $\mathcal{R}^{\mathcal{A}}$ succeeds later. Since $\mathbf{LocalMem}(\mathcal{CH})$ is very large in this case, the result in [1] is not directly captured by Theorem 3. However, one can easily modify our proof by letting $\mathcal{CH}$ in **Game 2** access to the stream $\boldsymbol{y}$ instead of storing $\boldsymbol{y}$. By doing this, $\mathbf{LocalMem}(\mathcal{CH})$ can remain small when $\mathcal{R}$ forwards signing queries from $\mathcal{A}$ to $\mathcal{CH}$, and hence, the lower bound in [1] or ones in other similar cases can be derived from our result (when treating unique signatures). We do not take this into account in our formal proof only for simplicity.

Re-randomizable signatures and VUFs. If we give an additional restriction that a reduction does not control the random tape of an adversary, i.e., an adversary uses real random coins (but not ones from the reduction), then by slightly modifying our proof, we can also show a memory lower bound for re-randomizable signatures [23,13], where signatures can be efficiently re-randomized (we refer the reader to [13] for the formal definition). In this case, we only have to let both the inefficient adversary and the simulator re-randomize the forged signatures so that $\mathcal{R}$ cannot distinguish them.

We can also extend our result for the notion of VUFs [20,19], which is exactly the same as the notion of unique signatures except that a proof (which is not necessarily unique) is needed when verifying the validity of a signature. We omit the details since the extension is straightforward.

4 Lower Bound of Restricted Reductions from mU-mOW to mU-OW for Unique-Key Cryptographic Primitives

In this section, we give a memory lower bound of restricted reductions from mU-mOW security to mU-OW security for unique-key one-way primitives. For simplicity, we treat a basic primitive called *unique-key relation* [24] and argue that this result can be easily extended for other unique-key primitives. We start by recalling the definition of unique-key relations and their security in the multi-user setting, and then show the lower bound.

4.1 Unique-Key Relations

We now recall the definition of a unique-key relation. In a unique-key relation, there exists at most one valid secret key for every public key in the support of the key generation algorithm.[7]

Definition 6 (Unique-key relation). *A* unique-key relation *consists of PT algorithms* $(\mathsf{Gen}, \mathsf{Check})$. *(a)* Gen *is a probabilistic algorithm that takes as input* 1^λ*, and returns a public/secret key pair* (pk, sk). *(b)* Check *is a deterministic algorithm that takes as input a public/secret key pair* (pk, sk)*, and returns* 1 *(accept) or* 0 *(reject).*

A unique-key relation is required to satisfy correctness *and* uniqueness*. Correctness is satisfied if* $\mathsf{Check}(pk, sk) = 1$ *holds for all* $\lambda \in \mathbb{N}$ *and all* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*. Uniqueness is satisfied if for all* $\lambda \in \mathbb{N}$ *and all* pk *in the support of* $\mathsf{Gen}(1^\lambda)$*, there exists no pair* (sk, sk') *that simultaneously satisfies* $sk \neq sk'$ *and* $\mathsf{Check}(pk, sk) = \mathsf{Check}(pk, sk') = 1$*.*

Now we give the definitions of the mU-mOW and mU-OW security of unique-key relations [2,3]. In these security games, an adversary sees many public keys and can adaptively corrupt the secret keys. It succeeds if it outputs a valid secret key that is not corrupted.

[7] Unlike the definition of unique signatures, here we do not require uniqueness for public keys outside the support of the key generation algorithm.

Definition 7 ($\mathsf{mU\text{-}mOW}$)**.** *A unique-key relation* $(\mathsf{Gen}, \mathsf{Check})$ *is said to be* $\mathsf{mU\text{-}mOW}$ secure *if for any PPT adversary* $\mathcal{A}$*, we have* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}mOW}}(\lambda) = \Pr[\mathcal{CH}$ *outputs* $1] \leq negl(\lambda)$ *in the following game.*

1. *The challenger* $\mathcal{CH}$ *sets* $w = 0$ *and* $\mathcal{Q} = \emptyset$*, and runs* $\mathcal{A}$ *on input* 1^λ*. Then* $\mathcal{A}$ *may make sampling queries to* $\mathcal{CH}$*, and* $\mathcal{CH}$ *responds as follows.*
   - *On receiving the* i*th sampling query* sp*,* $\mathcal{CH}$ *samples* $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ *and sends* pk_i *to* $\mathcal{A}$*.*
2. *Then* $\mathcal{A}$ *may make adaptive corruption and verification queries to* $\mathcal{CH}$*, and* $\mathcal{CH}$ *responds as follows:*
   - *On receiving a corruption query* i*,* $\mathcal{CH}$ *adds* i *to* $\mathcal{Q}$*, and sends* sk_i *to* $\mathcal{A}$*.*
   - *On receiving a verification query* (i^*, sk^*)*, if* $\mathsf{Check}(pk_{i^*}, sk^*) = 1$ *and* $i^* \notin \mathcal{Q}$*,* $\mathcal{CH}$ *sets* $w = 1$*.*
3. *At some point,* $\mathcal{A}$ *makes a stopping query* stp *to* $\mathcal{CH}$*, and* $\mathcal{CH}$ *returns* w*.*

Definition 8 ($\mathsf{mU\text{-}OW}$)**.** $\mathsf{mU\text{-}OW}$ *security is defined in exactly the same way as* $\mathsf{mU\text{-}mOW}$ *security except that* $\mathcal{A}$ *is allowed to make only one verification query and the advantage of* $\mathcal{A}$ *is denoted by* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}OW}}(\lambda)$*.*

4.2 Lower Bound for Unique-Key Relations

In this section, we define restricted reductions from the $\mathsf{mU\text{-}mOW}$ security to $\mathsf{mU\text{-}OW}$ security of unique-key relations and show a memory lower bound of such reductions.

Restricted black-box reductions from $\mathsf{mU\text{-}mOW}$ to $\mathsf{mU\text{-}OW}$. Let $\mathcal{R}$ be a black-box reduction from $\mathsf{mU\text{-}mOW}$ security to $\mathsf{mU\text{-}OW}$ security. As before, we write $\mathcal{R}^{\mathcal{A}}$ to mean that $\mathcal{R}$ has oracle access to a (stateful) adversary $\mathcal{A}$ playing the $\mathsf{mU\text{-}mOW}$ game. Whenever receiving a query from $\mathcal{R}$, $\mathcal{A}$ returns the "next" query to $\mathcal{R}$. $\mathcal{R}$ is not able to modify the current state of $\mathcal{A}$ (i.e., $\mathcal{A}$ runs sequentially), but is allowed to adaptively rewind $\mathcal{A}$ to previous states.

Definition 9 (c-restricted black-box reduction from $\mathsf{mU\text{-}mOW}$ to $\mathsf{mU\text{-}OW}$). *Let* $c > 0$ *be a constant. An oracle-access PPT machine* $\mathcal{R}^{(\cdot)}$ *is said to be a* c-restricted black-box reduction *from the* $\mathsf{mU\text{-}mOW}$ *security to the* $\mathsf{mU\text{-}OW}$ *security of a unique-key relation, if for any (possibly inefficient) adversary* $\mathcal{A}$*, we have* $\mathbf{Adv}^{\mathcal{R}^{\mathcal{A}}}_{\mathsf{mU\text{-}OW}}(\lambda) \geq c \cdot \mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}mOW}}(\lambda)$*, and* $\mathcal{R}$ *respects the following restriction.*

- *The public keys* $(pk_1, \cdots, pk_n)$ *that* $\mathcal{R}$ *sends to* $\mathcal{A}$ *are the ones generated by the challenger and given to* $\mathcal{R}$ *in the* $\mathsf{mU\text{-}OW}$ *game.*
- *The set of corruption queries* $\{y_1, \cdots, y_q\}$ *made by* $\mathcal{R}$ *is the same as the set of all corruption queries made by* $\mathcal{A}$*.*

Before showing the lower bound, we recall the definition of PRFs which will be exploited in our proof.

Definition 10 (Pseudorandom function (PRF)). $\mathsf{F} : \{0,1\}^{\kappa(\lambda)} \times \{0,1\}^{\delta(\lambda)} \to \{0,1\}^{\rho(\lambda)}$*, where* $\kappa = \kappa(\lambda)$*,* $\delta = \delta(\lambda)$*, and* $\rho = (\lambda)$ *are polynomials, is said to be a* pseudorandom function*, if for any PPT adversary* $\mathcal{A}$*, we have*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathsf{PR}}(\lambda) = |\Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_k}(1^\lambda) \mid k \leftarrow \{0,1\}^\kappa] - \Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda)]| \leq negl(\lambda).$$

Here, $\mathcal{O}_k(i)$ *returns* $\mathsf{F}(k,i)$*.* $\mathcal{O}(i)$ *returns* r *if there exists* (i,r) *in its internal list (initiated with* $\emptyset$*). Otherwise,* $\mathcal{O}(i)$ *returns* $r \leftarrow \{0,1\}^\rho$ *and adds* (i,r) *to its list.*

The main theorem is as follows.

Theorem 4. *Let* λ *be a (sufficiently large) security parameter,* $\Phi = (\mathsf{Gen}, \mathsf{Check})$*, where the internal randomness space of* Gen *is* $\{0,1\}^\rho$*, be a* $\mathsf{mU\text{-}OW}$ *secure unique-key relation,* $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\lambda \to \{0,1\}^\rho$ *be a PRF, and* $\mathcal{R}$ *be a* c_r*-restricted black-box reduction from the* $\mathsf{mU\text{-}mOW}$ *security to the* $\mathsf{mU\text{-}OW}$ *security of* Φ*. Let* $n = n(\lambda)$ *be the maximum number of sampling queries and* $q = q(\lambda)$ *be the maximum numbers of corruption and verification queries made by an adversary in the* $\mathsf{mU\text{-}mOW}$ *game, and* $\mathcal{U} = \{i\}_{i=1}^n$*. If (a)* $\mathcal{R}$ *rewinds the adversary for at most* $p = p(\lambda)$ *times and (b)* $c_r > 1/2$*, then we have*

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{R}) = \Omega(\max\{q/(p+2), n/(p+2)\}) - O(\log q) - O(\log n) - \kappa \\ - \max\{\mathbf{LocalMem}(\mathsf{Gen}), \mathbf{LocalMem}(\mathsf{Check}), \mathbf{LocalMem}(\mathsf{F})\}.\end{aligned}$$

Roughly, this theorem implies that when the maximum number of users and that of corruption queries made by an adversary in the $\mathsf{mU\text{-}mOW}$ game are very large, $\mathcal{R}$ must consume large memory unless it rewinds $\mathcal{A}$ many times, which increases its running-time.

High-level idea of the proof. We firstly construct an inefficient adversary $\mathcal{A}_{\boldsymbol{y}}$ where $\boldsymbol{y} = (y_1, \cdots, y_{2q})$. $\mathcal{A}_{\boldsymbol{y}}$ takes as input and stores public keys $pk_1, \cdots, pk_n$, makes corruption queries $y_1, \cdots, y_q$, checks the validity of the answers, and then makes verification queries $(pk_{y_{q+1}}, sk_1^*), \cdots, (pk_{y_{2q}}, sk_q^*)$ generated by using brute force. When $G(\boldsymbol{y}) = 1$, $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$ is likely to succeed in the $\mathsf{mU\text{-}OW}$ game, since $\mathcal{R}$ is a black-box reduction and $\mathcal{A}_{\boldsymbol{y}}$ is a deterministic algorithm breaking $\mathsf{mU\text{-}mOW}$ security. When $G(\boldsymbol{y}) = 0$, we can construct a PT algorithm $\mathcal{S}_{\boldsymbol{y}}$, which runs in the same way as $\mathcal{A}_{\boldsymbol{y}}$ does except that $\mathcal{S}_{\boldsymbol{y}}$ uses the answers of corruption queries to make verification queries. Due to uniqueness, $\mathcal{S}_{\boldsymbol{y}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$. Since the PPT algorithm $\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}$ is likely to fail in the $\mathsf{mU\text{-}OW}$ game, $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$ is likely to fail as well.

Then, similarly to the proof of Theorem 3, we can construct an algorithm $\mathcal{B}$ with access to a stream $\boldsymbol{y}$ that simulates the $\mathsf{mU\text{-}OW}$ game with $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}$ and outputs $G(\boldsymbol{y})$ with high probability. Therefore, we can show the lower bound on memory consumed by $\mathcal{R}$ since the memory consumed by $\mathcal{B}$ is inherently large, due to Theorem 2. However, one may notice that $\mathcal{B}$ uses a large amount of memory to store $pk_1, \cdots, pk_n$, which spoils our result since $\mathcal{B}$ using large memory does not imply $\mathcal{R}$ using large memory any more. We deal with this problem by using a PRF to simulate random coins used by the challenger, and runs the PRF to output the corresponding random coin used to generate a public key when the key is needed. In this way, $\mathcal{B}$ does not store the public keys anymore. Here, there is a point that $\mathcal{B}$ can simulate $\mathcal{A}_{\boldsymbol{y}}$ efficiently by using secret keys generated by the challenger in the $\mathsf{mU\text{-}OW}$ game, so that the whole interaction $\mathcal{B}$ simulates only runs in polynomial-time and cannot distinguish outputs of the PRF with real random coins.

Proof (of Theorem 4). Assuming the existence of the reduction $\mathcal{R}$ stated in Theorem 4, we show the existence of a probabilistic algorithm $\mathcal{B}$ such that $\Pr[G(\boldsymbol{y}) \leftarrow \mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda)] \geq c$ for all $\boldsymbol{y} = (y_1, \cdots, y_{2q}) \in \mathcal{U}^{2q}$ and some constant $c > 1/2$ by giving hybrid games.

Game 0: In this game, $\mathcal{R}$ has access to an adversary $\mathcal{A}_{\boldsymbol{y}}$ and interacts with the challenger $\mathcal{CH}$ in the $\mathsf{mU\text{-}OW}$ game. $\mathcal{A}_{\boldsymbol{y}}$ runs as follows.

1. On receiving 1^λ, $\mathcal{A}_{\boldsymbol{y}}$ makes a sampling query *sp*.
2. For $i = 1, \cdots, n-1$, on receiving pk_i, $\mathcal{A}_{\boldsymbol{y}}$ stores pk_i and makes a sampling query *sp*.
3. On receiving pk_n, $\mathcal{A}_{\boldsymbol{y}}$ stores pk_n and makes a corruption query y_1.
4. For $i = 1, \cdots, q-1$, on receiving the answer sk'_i to the ith corruption query, if $\mathsf{Check}(pk_{y_i}, sk'_i) \neq 1$, $\mathcal{A}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{A}_{\boldsymbol{y}}$ makes a corruption query y_{i+1}.
5. On receiving the answer sk'_q to the qth corruption query, if $\mathsf{Check}(pk_{y_q}, sk'_q) \neq 1$, $\mathcal{A}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{A}_{\boldsymbol{y}}$ exhaustively searches sk^*_1 such that $\mathsf{Verify}(pk_{y_{q+1}}, sk^*_1) = 1$, and makes a verification query (y_{q+1}, sk^*_1).
6. For $i = 1, \cdots, q-1$, when invoked (with no input) for the ith time, $\mathcal{A}_{\boldsymbol{y}}$ exhaustively searches sk^*_{i+1} such that $\mathsf{Check}(pk_{y_{q+i+1}}, sk^*_{i+1}) = 1$, and makes a verification query (y_{q+i+1}, sk^*_{i+1}).
7. When invoked (with no input) for the qth time, $\mathcal{A}_{\boldsymbol{y}}$ makes a stopping query *stp*.

We now show the following lemma.

Lemma 4. $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c_r$ *for all* $\boldsymbol{y} \in \mathcal{U}^{2q}$ *in* **Game 0**.

Proof (of Lemma 4). Firstly, we show the existence of a PT algorithm $\mathcal{S}_{\boldsymbol{y}}$ perfectly simulating $\mathcal{A}_{\boldsymbol{y}}$ on condition that $G(\boldsymbol{y}) = 0$. $\mathcal{S}_{\boldsymbol{y}}$ runs as follows. (Below, the difference from $\mathcal{A}_{\boldsymbol{y}}$ is *emphasized*.)

1. On receiving 1^λ, $\mathcal{S}_{\boldsymbol{y}}$ makes a sampling query *sp*.
2. For $i = 1, \cdots, n-1$, on receiving pk_i, $\mathcal{S}_{\boldsymbol{y}}$ stores pk_i and makes a sampling query *sp*.
3. On receiving pk_n, $\mathcal{S}_{\boldsymbol{y}}$ stores pk_n and makes a corruption query y_1.
4. For $i = 1, \cdots, q-1$, on receiving the answer sk'_i to the ith corruption query, if $\mathsf{Check}(pk_{y_i}, sk'_i) \neq 1$, $\mathcal{S}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{S}_{\boldsymbol{y}}$ *stores* (y_i, sk'_i) *in its internal list L (initialized with* $\emptyset$*)*, and makes a corruption query y_{i+1}.

5. On receiving the answer sk'_q to the qth corruption query, if $\mathsf{Check}(pk_{y_q}, sk'_q) \neq 1$, $\mathcal{S}_{\boldsymbol{y}}$ aborts. Otherwise, $\mathcal{S}_{\boldsymbol{y}}$ *stores* (y_q, sk'_q) *in* L*, searches a pair* (i^*, sk) *in* L *such that* $i^* = y_{q+1}$*, and makes a verification query* (y_{q+1}, sk)*. If the searching procedure fails,* $\mathcal{S}_{\boldsymbol{y}}$ *aborts.*
6. For $i = 1, \cdots, q-1$, when invoked (with no input) for the ith time, $\mathcal{S}_{\boldsymbol{y}}$ *searches a pair* (i^*, sk) *in* L *such that* $i^* = y_{q+i+1}$*, and makes a verification query* (y_{q+i+1}, sk)*. If the searching procedure fails for some* i*,* $\mathcal{S}_{\boldsymbol{y}}$ *aborts.*
7. When invoked (with no input) for the qth time, $\mathcal{S}_{\boldsymbol{y}}$ makes a stopping query *stp*.

When $G(\boldsymbol{y}) = 0$, we have $\{y_{q+i}\}_{i=1}^{q} \subseteq \{y_i\}_{i=1}^{q}$, which means that the searching procedures executed by $\mathcal{S}_{\boldsymbol{y}}$ (in Steps 5 and 6) will not fail. Moreover, due to the uniqueness of Φ, the verification queries made by $\mathcal{S}_{\boldsymbol{y}}$ are exactly the same as those made by $\mathcal{A}_{\boldsymbol{y}}$. Hence, $\mathcal{S}_{\boldsymbol{y}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$ in the view of $\mathcal{R}$.

Due to the $\mathsf{mU\text{-}OW}$ security of Φ, we have $\mathbf{Adv}_{\mathsf{mU\text{-}OW}}^{\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}}(\lambda) = \mathbf{Adv}_{\mathsf{mU\text{-}OW}}^{\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}}(\lambda) \leq negl(\lambda)$ when $G(\boldsymbol{y}) = 0$, which implies $\Pr[1 \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda)) \mid G(\boldsymbol{y}) = 0] \leq negl(\lambda)$, i.e., $\Pr[0 \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda)) \mid G(\boldsymbol{y}) = 0] \geq 1 - negl(\lambda)$. On the other hand, when $G(\boldsymbol{y}) = 1$, there exists some $1 \leq j \leq q$ such that $y_{q+j} \notin \{y_i\}_{i=1}^{q}$, which implies $\mathbf{Adv}_{\mathsf{mU\text{-}mOW}}^{\mathcal{A}_{\boldsymbol{y}}}(\lambda) = 1$. Since $\mathcal{R}$ is a c_r-restricted black-box reduction, we have $\Pr[1 \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda)) \mid G(\boldsymbol{y}) = 1] \geq c_r$. Since $c_r > 1/2$ and λ *is sufficiently large*, we have $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda))] \geq c_r$, completing the proof of Lemma 4. □

Game 1: This game is exactly the same as **Game 0** except that for each i, $\mathcal{CH}$ generates the ith key pair by computing $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda; \mathsf{F}(k, i))$ where k is randomly chosen from $\{0,1\}^\kappa$ at the beginning of the game.

Lemma 5. $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda))] \geq c$ *for all* $\boldsymbol{y} \in \mathcal{U}^{2q}$ *and some constant* $c > 1/2$ *in* **Game 1**.

Proof (of Lemma 5). Let $\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda))]$ be $c_0^{\boldsymbol{y}}$ (respectively, $c_1^{\boldsymbol{y}}$) in **Game 0** (respectively, **Game 1**). For any $\boldsymbol{y}$, we can construct a PPT adversary $\mathcal{D}$ breaking the pseudorandom property of F with advantage $\mathbf{Adv}_{\mathsf{PR}}^{\mathcal{D}}(\lambda) = |c_0^{\boldsymbol{y}} - c_1^{\boldsymbol{y}}|$ as follows.

$\mathcal{D}$ has access to an oracle $\mathcal{O}_k$ parameterized by $k \leftarrow \{0,1\}^\kappa$ or an oracle $\mathcal{O}$ (see Definition 10 for the descriptions of $\mathcal{O}_k$ and $\mathcal{O}$). $\mathcal{D}$ runs $\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightarrows \mathcal{CH}(1^\lambda)$ in exactly the same way as in **Game 0**, except that $\mathcal{A}_{\boldsymbol{y}}$ receives secret keys generated by $\mathcal{CH}$ from $\mathcal{D}$ to make verification queries, instead of using brute force to recover them. This is possible due to the restriction that all the public keys $\mathcal{R}$ sends to $\mathcal{A}$ are generated by $\mathcal{CH}$. Furthermore, when $\mathcal{CH}$ requires the ith random coin, $\mathcal{D}$ makes a query to its oracle and sends the answer of the query back. If $\mathcal{CH}$ outputs $G(\boldsymbol{y})$, $\mathcal{D}$ outputs 1. Otherwise, $\mathcal{D}$ outputs 0.

When the oracle is $\mathcal{O}$ (respectively, $\mathcal{O}_k$), the view of $\mathcal{CH}$ is exactly the same as its view in **Game 0** (respectively, **Game 1**) due to the unique key property. Therefore, we have $\mathbf{Adv}_{\mathsf{PR}}^{\mathcal{D}}(\lambda) = |c_0^{\boldsymbol{y}} - c_1^{\boldsymbol{y}}|$. Due to the pseudorandom property of F, we have $|c_0^{\boldsymbol{y}} - c_1^{\boldsymbol{y}}| \leq negl(\lambda)$. *Since* λ *is sufficiently large*, combining this bound with Lemma 4 completes the proof of Lemma 5. □

Game 2: This game is exactly the same as **Game 1** except that there exists an algorithm $\mathcal{A}'$ with access to the stream oracle $\mathcal{O}_{\boldsymbol{y}}$ simulating $\mathcal{A}_{\boldsymbol{y}}$ as follows. (Below, the difference from $\mathcal{A}_{\boldsymbol{y}}$ is *emphasized.*)

1. On receiving 1^λ, $\mathcal{A}'$ makes a sampling query sp.
2. For $i = 1, \cdots, n-1$, on receiving pk_i, $\mathcal{A}'$ stores pk_i and makes a sampling query sp.
3. On receiving pk_n, $\mathcal{A}'$ stores pk_n, *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_1, and makes a corruption query y_1.
4. For $i = 1, \cdots, q-1$, on receiving the answer sk'_i to the ith corruption query, if $\mathsf{Check}(pk_{y_i}, sk'_i) \neq 1$, $\mathcal{A}'$ aborts. Otherwise, $\mathcal{A}'$ *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_{i+1}, and makes a corruption query y_{i+1}.
5. On receiving the answer sk'_q to the qth corruption query, if $\mathsf{Check}(pk_{y_q}, sk'_q) \neq 1$, $\mathcal{A}'$ aborts. Otherwise, $\mathcal{A}'$ *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_{q+1}, exhaustively searches sk^*_1 such that $\mathsf{Check}(pk_{y_{q+1}}, sk^*_1) = 1$, and makes a verification query (y_{q+1}, sk^*_1).
6. For $i = 1, \cdots, q-1$, when invoked (with no input), $\mathcal{A}'$ *queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain* y_{q+i+1}, exhaustively searches sk^*_{i+1} such that $\mathsf{Check}(pk_{y_{q+i+1}}, sk^*_{i+1}) = 1$, and makes a verification query (y_{q+i+1}, sk^*_{i+1}).
7. When invoked (without input) for the qth time, $\mathcal{A}'$ makes a stopping query stp.

Whenever $\mathcal{R}$ executes a rewinding procedure, $\mathcal{A}'$ makes another pass on its stream to obtain the index for the next corruption or verification query. Since $\mathcal{A}'^{\mathcal{O}_{\boldsymbol{y}}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$, we have the following lemma.

Lemma 6. *$\Pr[G(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}'^{\mathcal{O}_{\boldsymbol{y}}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c$ for all $\boldsymbol{y} \in \mathcal{U}^{2q}$ and some constant $c > 1/2$ in* **Game 2**.

Game 3: This game is the same as **Game 2** except that there exists a stream access algorithm $\mathcal{A}''^{\mathcal{O}_{\boldsymbol{y}}}$ that runs $k \leftarrow \{0,1\}^\kappa$, stores k, simulates $\mathcal{CH}$, $\mathcal{R}$, and $\mathcal{A}'^{\mathcal{O}_{\boldsymbol{y}}}$, and generates the ith key pair by computing $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda; \mathsf{F}(k, i))$. When $\mathcal{R}$ makes a verification query (i, sk^*), $\mathcal{CH}$ makes another pass on the stream $\boldsymbol{y}$ through $\mathcal{A}''$, and checks whether $i \in \{y_1, \cdots, y_q\}$ and $\mathsf{Check}(pk_i, sk^*) = 1$.[8] If the check works, $\mathcal{CH}$ outputs 1. Otherwise, $\mathcal{CH}$ outputs 0. Then $\mathcal{A}''$ returns the output of $\mathcal{CH}$. Since the view of $\mathcal{CH}$ in this game is identical to its view in **Game 2**, we have the following lemma.

Lemma 7. *$\Pr[G(\boldsymbol{y}) \leftarrow \mathcal{A}''^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda)] \geq c$ for all $\boldsymbol{y} \in \mathcal{U}^{2q}$ and some constant $c > 1/2$ in* **Game 3**.

Game 4: In this game, there exists an algorithm $\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}$ which runs in exactly the same way as $\mathcal{A}''^{\mathcal{O}_{\boldsymbol{y}}}$ except that it does not store $(pk_i)_{i=1}^n$ generated by $\mathcal{CH}$. Instead, whenever $\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}$ needs to see pk_i, $\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}$ computes $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda; \mathsf{F}(k, i))$. Since the view of $\mathcal{CH}$ in **Game 4** is identical to its view in **Game 3**, we have the following lemma.

[8] According to the second restriction in Definition 9, the corruption queries $\mathcal{R}$ has made are $\{y_1, \cdots, y_q\}$.

Lemma 8. $\Pr[G(\boldsymbol{y}) \leftarrow \mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda)] \geq c$ *for all* $\boldsymbol{y} \in \mathcal{U}^{2q}$ *and some constant* $c > 1/2$ *in* **Game 4**.

Since $\mathcal{B}$ makes $p+2$ passes on its stream in total, according to Theorem 2 and Lemma 8, we have

$$\mathbf{LocalMem}(\mathcal{B}) = \Omega(\min\{q/(p+2), n/(p+2)\}).$$

Furthermore, the memory used to simulate $\mathcal{CH}$, $\mathcal{A}''$, and random coins is $O(\log q) + O(\log n) + \max\{\mathbf{LocalMem}(\mathsf{Gen}), \mathbf{LocalMem}(\mathsf{Check}), \mathbf{LocalMem}(\mathsf{F})\} + \kappa$, where $O(\log q) + O(\log n)$ is the amount of memory used to record q, n, and the index of the next query $\mathcal{A}''$ will make. Therefore we have

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{B}) = {} & O(\mathbf{LocalMem}(\mathcal{R})) + O(\log q) + O(\log n) + \kappa \\ & + \max\{\mathbf{LocalMem}(\mathsf{Gen}), \mathbf{LocalMem}(\mathsf{Check}), \mathbf{LocalMem}(\mathsf{F})\}.\end{aligned}$$

Combining the above two bounds completes Theorem 4. □

Remark on security parameter. Similarly to the case of Theorem 3, Theorem 4 holds only when the security parameter λ is sufficiently large, while one may wonder why memory-tightness makes sense when λ is already required to be large. In fact, λ only has to be large enough to ensure $1 - \mathbf{Adv}^{\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}}_{\mathsf{mU\text{-}OW}}(\lambda) - \mathbf{Adv}^{\mathcal{D}}_{\mathsf{PR}}(\lambda) > 1/2$ and $c_r - \mathbf{Adv}^{\mathcal{D}}_{\mathsf{PR}}(\lambda) > 1/2$ in the proofs of Lemmas 4 and 5. When c_r is large, these two inequations should hold even if λ is small (to some extent) and $\mathcal{R}^{\mathcal{S}_{\boldsymbol{y}}}$ and $\mathcal{D}$ may consume large memory, due to $\mathsf{mU\text{-}mOW}$ security and pseudorandomness. Therefore, λ is not necessarily very large unless c_r is very close to $1/2$.

Lower bound for other unique-key and re-randomizable primitives. It is not hard to see that the above result can be easily extended to lower bound results for (one-way secure) PKE schemes, signatures, and many other primitives in the multi-user setting, in which key pairs satisfy unique-key relations. Since unique-key primitives capture many existing natural constructions [22,8,5,12,8,7,17,23,14,18], a very wide class of memory lower bounds in the multi-user setting can be directly derived from our result stated in Theorem 4. For ease of understanding, we take unique-key PKE schemes and unique-key signatures as examples in Appendix A. Concretely, we give the definitions of unique-key PKE schemes and unique-key signatures and their security notions in the multi-user setting. Then we give two corollaries showing that in this setting, reductions from multi-challenge security to single-challenge security for these two types of primitives must consume large memory unless they increase running-time.

Similarly to the case of unique signatures, this result can also be extended for primitives with key re-randomization [4] if reductions do not control random tapes of adversaries.[9]

[9] Similarly to the definition of unique-key relations, we do not require re-randomization for public keys outside the support of the key generation algorithm.

5 Lower Bound of Restricted Reductions from mCR_t to CR_t for Large-Domain Hash Functions

In [1], Auerbach et al. showed a memory lower bound of restricted reductions from mCR_t security to CR_t security for an artificial function which just truncates last λ bits of its input, while such a function does not satisfy CR_t security itself. In this section, we extend this result to a lower bound for all the large-domain hash functions satisfying CR_t security (where t is a constant). To achieve the goal, we prove a streaming lower bound with respect to hash functions.

5.1 Hash Functions

In this section, we define large-domain hash functions, recall mCR_t security and CR_t security, and show a theorem for large-domain hash functions.

Definition 11 (Large-domain hash function). *A hash function $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\delta \to \{0,1\}^\rho$, where $\kappa = \kappa(\lambda)$, $\delta = \delta(\lambda)$, and $\rho = \rho(\lambda)$ are polynomials, is said to have a large domain if $2^{\rho-\delta} \leq negl(\lambda)$.*

Definition 12 (mCR_t [1]). *A hash function $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\delta \to \{0,1\}^\rho$ is said to satisfy mCR_t security (where t is some constant independent of the security parameter λ), if for any PPT adversary $\mathcal{A}$, we have $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mCR}_t}(\lambda) = \Pr[\mathcal{CH}$ outputs $1] \leq negl(\lambda)$ in the following game.*

1. *The challenger $\mathcal{CH}$ sets $\mathcal{Q} = \emptyset$, randomly chooses $k \leftarrow \{0,1\}^\kappa$, and runs $\mathcal{A}$ on input $(1^\lambda, k)$. $\mathcal{A}$ may make adaptive input queries to $\mathcal{CH}$. Every time on receiving a query $m \in \{0,1\}^\delta$ from $\mathcal{A}$, $\mathcal{CH}$ adds m to $\mathcal{Q}$.*
2. *At some point, $\mathcal{A}$ makes a stopping query stp to $\mathcal{CH}$. If there exists $\{m_i^*\}_{i=1}^t \subseteq \mathcal{Q}$ such that $\mathsf{H}_k(m_1^*) = \cdots = \mathsf{H}_k(m_t^*)$ and $|\{m_i^*\}_{i=1}^t| = t$, $\mathcal{CH}$ outputs 1. Otherwise, $\mathcal{CH}$ outputs 0.*

Definition 13 (CR_t [1]). *The definition of CR_t security is defined in exactly the same way as that of mCR_t security, except that that $\mathcal{A}$ is allowed to make at most t input queries and the advantage of $\mathcal{A}$ is denoted by $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{CR}_t}(\lambda)$.*

Next we give a theorem that will be used to prove a streaming lower bound later. This theorem shows that for a large-domain hash function satisfying CR_t security, there exist "many" hash values with more than t pre-images. Intuitively, if this theorem does not hold, then there will be some hash value with many pre-images, so that t randomly chosen inputs are likely to fall into the class of these pre-images, which breaks its CR_t security.

Theorem 5. *Let λ be a (sufficiently large) security parameter, $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\delta \to \{0,1\}^\rho$ be a large-domain hash function satisfying CR_t security, and $n = n(\lambda)$ be any polynomial in λ. For $k \leftarrow \{0,1\}^\kappa$, the probability that there exist more than n elements in $\{0,1\}^\rho$ with more than t pre-images (with respect to H_k) is $1 - negl(\lambda)$.*

Proof (of Theorem 5). Let $n' \leq n$ and E be the event that the number of elements in $\{0,1\}^\rho$ that have more than t pre-images is exactly n'. To prove Theorem 5, we just need to prove $\Pr[E] \leq \mathit{negl}(\lambda)$.

Let $k \leftarrow \{0,1\}^\kappa$, $m \leftarrow \{0,1\}^\delta$, and E_0 be the event that the number of pre-images of $\mathsf{H}_k(m)$ is more than t. Since the number of elements in $\{0,1\}^\delta$, the hash values of which respectively have less than t pre-image, is at most 2^ρ, we have $\Pr[\overline{E_0} \mid E] \leq t \cdot 2^\rho/2^\delta$, i.e., $\Pr[E_0 \mid E] \geq 1 - t \cdot 2^\rho/2^\delta$. Let E_i be event that $\mathsf{H}_k(m)$ is the *ith* lexicographically smallest value in $\{0,1\}^\rho$ with more than t pre-images. Since $\Pr[E_1 \vee \cdots \vee E_{n'} \mid E] = \Pr[E_0 \mid E]$, there must exist some $i^* \in \{1, \cdots, n'\}$ such that

$$\Pr[E_{i^*} \mid E] \geq 1/n' \cdot \Pr[E_0 \mid E] \geq (1/n) \cdot (1 - t \cdot 2^\rho/2^\delta).$$

Now we construct a PPT adversary $\mathcal{A}$ in the CR_t game of H. On receiving $k \leftarrow \{0,1\}^\kappa$, $\mathcal{A}$ randomly chooses $m_1, \cdots, m_t \leftarrow \{0,1\}^\delta$, and uses them as input queries. Let E' be the event that there exist some $i, j \in \{1, \cdots, t\}$ such that $m_i = m_j$. We have $\Pr[E'] \leq 1-(1-(t-1)/2^\delta)^t \leq O(t^2/2^\delta)$. Therefore, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathcal{A}}_{CR_t}(\lambda) =& \Pr[\overline{E'} \wedge \mathsf{H}_k(m_1) = \cdots = \mathsf{H}_k(m_t)] \\
=& \Pr[\mathsf{H}_k(m_1) = \cdots = \mathsf{H}_k(m_t)] - \Pr[E' \wedge \mathsf{H}_k(m_1) = \cdots = \mathsf{H}_k(m_t)] \\
\geq& \Pr[\mathsf{H}_k(m_1) = \cdots = \mathsf{H}_k(m_t) \wedge E] - \Pr[E'] \\
=& \Pr[\mathsf{H}_k(m_1) = \cdots = \mathsf{H}_k(m_t) \mid E] \cdot \Pr[E] - \Pr[E'] \\
\geq& \Pr[E_{i^*}|E]^t \cdot \Pr[E] - O(t^2/2^\delta) \\
\geq& (1/n \cdot (1 - t \cdot 2^\rho/2^\delta))^t \cdot \Pr[E] - O(t^2/2^\delta).
\end{aligned}$$

As a result, the probability that $\mathcal{A}$ breaks CR_t security is larger than $(1/n \cdot (1-t \cdot 2^\rho/2^\delta))^t \cdot \Pr(E) - O(t^2/2^\delta)$, where t is some constant. Since $(1/n \cdot (1-t \cdot 2^\rho/2^\delta))^t \geq 1/n^t - \mathit{negl}(\lambda)$ and $O(t^2/2^\delta) \leq \mathit{negl}(\lambda)$, we have $\Pr[E] \leq \mathit{negl}(\lambda)$, completing the proof of Theorem 5. □

5.2 Streaming Lower Bound for Hash Functions

In this section, we give a theorem, which is another corollary of prior works [16,21] based on the disjointness problem. It it also a variant of a streaming lower bound shown in [1]. It shows the existence of a memory lower bound for determining whether there exists a t-collision, with respect to a CR_t secure large-domain hash function, in a data stream. Before giving the main theorem, we define the function $F_{\mathsf{H},t}(\boldsymbol{y})$ as follows.

Let $\boldsymbol{y} \in (\{0,1\}^\delta)^q$, $F_{\mathsf{H}}(\boldsymbol{y})$ be defined as $F_{\mathsf{H}}(\boldsymbol{y}) = \max_{s \in \{0,1\}^\rho} |\{y_i : \mathsf{H}(y_i) = s\}|$, and $F_{\mathsf{H},t}(\boldsymbol{y})$ be defined as

$$F_{\mathsf{H},t}(\boldsymbol{y}) = \begin{cases} 1 & \text{if } F_{\mathsf{H}}(\boldsymbol{y}) \geq t \\ 0 & \text{otherwise} \end{cases}.$$

Theorem 6. *Let $\mathcal{B}$ be a probabilistic algorithm, λ be a (sufficiently large) security parameter, and $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\delta \to \{0,1\}^\rho$ be a large-domain hash function satisfying CR_t security. Assuming that there exists some constant $c > 1/2$ such that $\Pr[\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda, k) = F_{\mathsf{H}_k,t}(\boldsymbol{y}) \mid k \leftarrow \{0,1\}^\kappa] \geq c$ holds for polynomial $q = q(\lambda)$ and all $\boldsymbol{y} \in (\{0,1\}^\delta)^q$. Then* **LocalMem**$(\mathcal{B}) = \Omega((q-\kappa)/p)$*, where p is the number of passes $\mathcal{B}$ makes in the worst case.*

Proof (of Theorem 6). Let $n = \lfloor q/t \rfloor$. We now construct a two-party protocol (P_1, P_2) by using $\mathcal{B}$ as follows. Taking as input $x_1 \in \{0,1\}^n$, P_1 samples $k \leftarrow \{0,1\}^\kappa$ and sends k to P_2. If there do not exist n elements in $\{0,1\}^\rho$ with more than t pre-images for the hash function $\mathsf{H}_k(\cdot)$, P_1 aborts. Let h_i be the ith lexicographically smallest element in $\{0,1\}^\rho$ with more than t pre-images, m_{ij} be the jth smallest pre-image of h_i, and $\overline{m}$ be an element in $\{0,1\}^\delta$ such that $\mathsf{H}_k(\overline{m}) \notin \{h_i\}_{i=1}^n$. For $i = 1, \cdots, n$, if the ith bit of x_1 is 1, P_1 adds $(m_{ij})_{j=1}^{\lfloor t/2 \rfloor}$ to $\boldsymbol{y}_1$. Taking as input $x_2 \in \{0,1\}^n$, for $i = 1, \cdots, n$, if the ith bit of x_2 is 1, P_2 adds $(m_{ij})_{j=\lceil t/2 \rceil}^{t}$ to $\boldsymbol{y}_2$. Then P_1 and P_2 respectively pad $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ with $\overline{m}$ so that $\boldsymbol{y} = \boldsymbol{y}_1 || \boldsymbol{y}_2$ consists of q elements in total.

Then (P_1, P_2) starts to run $\mathcal{B}(1^\lambda, k)$ in multiple rounds until $\mathcal{B}$ stops and returns $b \in \{0,1\}$. More specifically, in each round, P_1 runs $\mathcal{B}(1^\lambda, k)$, answers queries from $\mathcal{B}$ to the stream $\boldsymbol{y}_1$, and sends the local memory state of $\mathcal{B}$ denoted by s to P_2 after all the elements in $\boldsymbol{y}_1$ having been queried by $\mathcal{B}$. P_2 runs $\mathcal{B}(1^\lambda, k)$ starting from state s, answers queries from $\mathcal{B}$ to the stream $\boldsymbol{y}_2$, and then sends the local memory state of $\mathcal{B}$ back to P_1 after all the elements in $\boldsymbol{y}_2$ having been queried. The final output of (P_1, P_2) is $\mathcal{B}$'s output b.

Since the probability that there exist more than n elements with more than t pre-images is $1 - \mathit{negl}(\lambda)$, we have $\Pr[\mathsf{DISJ}(\boldsymbol{x}_1, \boldsymbol{x}_2) = F_{\mathsf{H}_k,t}(\boldsymbol{y}) \mid k \leftarrow \{0,1\}^\kappa] \geq 1 - \mathit{negl}(\lambda)$ and $\Pr[\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda, k) = F_{\mathsf{H}_k,t}(\boldsymbol{y}) \mid k \leftarrow \{0,1\}^\kappa] \geq c$. As a result, we have $\Pr[\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda, k) = \mathsf{DISJ}(\boldsymbol{x}_1, \boldsymbol{x}_2) \mid k \leftarrow \{0,1\}^\kappa] \geq c - \mathit{negl}(\lambda)$, which implies $\Pr[P_1(\boldsymbol{x}_1) \leftrightarrow P_2(\boldsymbol{x}_2) = \mathsf{DISJ}(\boldsymbol{x}_1, \boldsymbol{x}_2)] \geq c - \mathit{negl}(\lambda)$. Since $c > 1/2$, there must exist some constant $c' > 1/2$ such that $c - \mathit{negl}(\lambda) > c'$ for a *sufficiently large* λ. Therefore, (P_1, P_2) solves the disjointness problem.

Since P_1 and P_2 have communication $\kappa + O(p \cdot \textbf{LocalMem}(\mathcal{B}))$, and Theorem 1 implies that the communication must be $\Omega(n) = \Omega(\lfloor q/t \rfloor) = \Omega(q)$, we have **LocalMem**$(\mathcal{B}) = \Omega((q-\kappa)/p)$, completing the proof. □

5.3 Lower Bound for Large-Domain Hash Functions

In this section, we recall the definition of restricted reductions from mCR_t security to CR_t security and show a memory lower bound of these reductions.

Restricted black-box reductions from mCR_t to CR_t. Let $\mathcal{R}$ be a black-box reduction from mCR_t security to CR_t security. As before, we write $\mathcal{R}^{\mathcal{A}}$ to mean that $\mathcal{R}$ has oracle access to a (stateful) adversary $\mathcal{A}$ playing the mCR_t game. Whenever receiving a query from $\mathcal{R}$, $\mathcal{A}$ returns the "next" query to $\mathcal{R}$. $\mathcal{R}$ is not able to modify the current state of $\mathcal{A}$ (i.e., $\mathcal{A}$ runs sequentially), but is allowed to adaptively rewind $\mathcal{A}$ to previous states.

Definition 14 (c-restricted black-box reduction from mCR_t to CR_t [1]). *Let $c > 0$ be a constant. An oracle-access PPT machine $\mathcal{R}^{(\cdot)}$ is said to be a* c-restricted black-box reduction *from the mCR_t security to the CR_t security of a hash function, if for any (possibly inefficient) adversary $\mathcal{A}$, we have $\mathbf{Adv}_{\mathsf{CR}_t}^{\mathcal{R}^{\mathcal{A}}}(\lambda) \geq c \cdot \mathbf{Adv}_{\mathsf{mCR}_t}^{\mathcal{A}}(\lambda)$, and $\mathcal{R}$ respects the following restrictions.*

- *The key k that $\mathcal{R}$ sends to $\mathcal{A}$ is the one generated by the challenger and given to $\mathcal{R}$ in the CR_t game.*
- *The queries made by $\mathcal{R}$ are amongst the queries made by $\mathcal{A}$.*

Theorem 7. *Let λ be a (sufficiently large) security parameter, $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\delta \to \{0,1\}^\rho$ be a large-domain hash function satisfying CR_t security, and $\mathcal{R}$ be a c_r-restricted black-box reduction from the mCR_t security to the CR_t security of H. Let $q = q(\lambda)$ be the maximum numbers of input queries made by an adversary in the mCR_t game. If (a) $\mathcal{R}$ rewinds the adversary for at most $p = p(\lambda)$ times and (b) $c_r > 1/2$, then we have*

$$\mathbf{LocalMem}(\mathcal{R}) = \Omega(\min\{(q-\kappa)/(p+1)\}) - O(\log q) - \mathbf{LocalMem}(\mathsf{H}).$$

Similarly to before, this theorem implies that when the maximum number of input queries made by an adversary in the mCR_t game is very large, $\mathcal{R}$ must consume large memory unless it rewinds $\mathcal{A}$ many times, which increases its running-time.

Proof (of Theorem 7). Assuming the existence of the reduction $\mathcal{R}$ stated in Theorem 7, we show the existence of a probabilistic algorithm $\mathcal{B}$ such that $\Pr[\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^\lambda, k) = F_{\mathsf{H}_k,t}(\boldsymbol{y}) \mid k \leftarrow \{0,1\}^\kappa] \geq c_r > 1/2$ for all $\boldsymbol{y} = (y_1, \cdots, y_q) \in (\{0,1\}^\delta)^q$.

Game 0: In this game, $\mathcal{R}$ has access to an adversary $\mathcal{A}_{\boldsymbol{y}}$, and interacts with the challenger $\mathcal{CH}$ in the mCR_t game. $\mathcal{A}_{\boldsymbol{y}}$ runs as follows.

- On receiving $(1^\lambda, k)$, $\mathcal{A}_{\boldsymbol{y}}$ makes an input query y_1.
- For $i = 1, \cdots, q-1$, when invoked (with empty input) for the ith time, $\mathcal{A}_{\boldsymbol{y}}$ makes an input query y_{i+1}.
- When invoked (with empty input) for the qth time, $\mathcal{A}_{\boldsymbol{y}}$ makes a stopping query *stp*.

We now show the following lemma.

Lemma 9. $\Pr[F_{\mathsf{H}_k,t}(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c_r$ *for all* $\boldsymbol{y} = (y_1, \cdots, y_q) \in (\{0,1\}^\delta)^q$ *in* **Game 0**.

Proof (of Lemma 9). If $F_{\mathsf{H}_k,t}(\boldsymbol{y}) = 0$, then $\mathcal{CH}$ will output 0. This is due to the restriction that all the input queries made by $\mathcal{R}$ are amongst the elements in $\boldsymbol{y}$. On the other hand, one can see that $\mathbf{Adv}_{\mathsf{mCR}_t}^{\mathcal{A}_{\boldsymbol{y}}}(\lambda) = 1$ when $F_{\mathsf{H}_k,t}(\boldsymbol{y}) = 1$. Since $\mathcal{R}$ is a c_r-restricted black-box reduction, we have $\mathbf{Adv}_{\mathsf{CR}_t}^{\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}}(\lambda) \geq c_r \cdot \mathbf{Adv}_{\mathsf{mCR}_t}^{\mathcal{A}_{\boldsymbol{y}}}(\lambda) = c_r$, i.e., $\Pr[F_{\mathsf{H}_k,t}(\boldsymbol{y}) \leftarrow (\mathcal{R}^{\mathcal{A}_{\boldsymbol{y}}}(1^\lambda) \leftrightharpoons \mathcal{CH}(1^\lambda))] \geq c_r$, completing the proof of Lemma 9. □

Game 1: This game is exactly the same as **Game 0** except that there exists an algorithm $\mathcal{B}$ with access to the stream $\boldsymbol{y}$ that simulates $\mathcal{A}_{\boldsymbol{y}}$, $\mathcal{CH}$, and $\mathcal{R}$. Here, $\mathcal{B}$ takes as input $k \leftarrow \{0,1\}^{\kappa}$ from an external party and uses it as the hash key generated by $\mathcal{CH}$. Moreover, $\mathcal{B}$ does not store $\boldsymbol{y}$ in its local memory but queries $\mathcal{O}_{\boldsymbol{y}}$ to obtain the ith input query y_i for $i = 1, \cdots, q$. Whenever $\mathcal{R}$ executes a rewinding procedure, $\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}$ makes another pass on its stream so that it can access its next input query to $\mathcal{R}$. Since $\mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}$ perfectly simulates $\mathcal{A}_{\boldsymbol{y}}$, we immediately obtain the following lemma.

Lemma 10. $\Pr[F_{\mathsf{H}_k,t}(\boldsymbol{y}) \leftarrow \mathcal{B}^{\mathcal{O}_{\boldsymbol{y}}}(1^{\lambda}, k)] \geq c_r$ *for all* $\boldsymbol{y} \in (\{0,1\}^{\delta})^q$ *in* **Game 1**.

Since $c_r > 1/2$ and $\mathcal{B}$ makes $p+1$ passes on its stream in total, according to Theorem 6 and Lemma 10, we have

$$\mathbf{LocalMem}(\mathcal{B}) = \Omega(\min\{(q-\kappa)/(p+1)\}).$$

Furthermore, the memory used to simulate $\mathcal{CH}$ and $\mathcal{A}_{\boldsymbol{y}}$ is $O(\log q) + \mathbf{LocalMem}(\mathsf{H})$, where $O(\log q)$ is the amount of memory used to record q and the index of the next input query $\mathcal{A}_{\boldsymbol{y}}$ will make. Therefore, we have

$$\mathbf{LocalMem}(\mathcal{B}) = \mathbf{LocalMem}(\mathcal{R}) + O(\log q) + \mathbf{LocalMem}(\mathsf{H}).$$

Combining the two bounds completes Theorem 7. □

Remark on security parameter. Similarly to the case of Theorems 3 and 4, Theorem 7 holds only when the security parameter λ is sufficiently large, while one may wonder why memory-tightness makes sense when λ is already required to be large. Notice that λ is required to be large only when c (in Theorem 6) is very close to 1/2. However, it is not hard to see that when c_r (which is the parameter of the reduction in Theorem 7) is not close to 1/2, c (in Theorem 6) is not necessarily close to 1/2. Hence, λ is not necessarily very large, unless c_r (in Theorem 7) is very close to 1/2.

6 Open Problem

The lower bound results shown in Section 4 and Section 5 only treat reductions which respect restrictions on their queries. It is desirable to clarify whether memory lower bounds of natural black-box reductions exist with respect to those security games. Showing some novel streaming lower bounds based on other problems about parity learning might be a promising way. It is also desirable to know whether there exist memory lower bounds of reductions for the multi-challenge security of other class of cryptographic primitives, and whether it is possible to unify these bounds. Finally, it would be interesting to find memory lower bounds in the random oracle model.

Acknowledgement

A part of this work was supported by Input Output Cryptocurrency Collaborative Research Chair funded by IOHK, Nomura Research Institute, NTT Secure Platform Laboratories, Mitsubishi Electric, I-System, JSPS Fellowship for Young Scientists, JST CREST JPMJCR14D6 and JPMJCR1688, JST OPERA, JSPS KAKENHI JP16H01705, 16J10697, JP17H01695.

References

1. B. Auerbach, D. Cash, M. Fersch, and E. Kiltz. Memory-tight reductions. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 101–132. Springer, 2017.
2. C. Bader. Efficient signatures with tight real world security in the random-oracle model. In *CANS*, volume 8813 of *Lecture Notes in Computer Science*, pages 370–383. Springer, 2014.
3. C. Bader, D. Hofheinz, T. Jager, E. Kiltz, and Y. Li. Tightly-secure authenticated key exchange. In *TCC (1)*, volume 9014 of *Lecture Notes in Computer Science*, pages 629–658. Springer, 2015.
4. C. Bader, T. Jager, Y. Li, and S. Schäge. On the impossibility of tight cryptographic reductions. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 273–304. Springer, 2016.
5. X. Boyen, Q. Mei, and B. Waters. Direct chosen ciphertext security from identity-based techniques. In *ACM CCS 2005*, pages 320–329, 2005.
6. J. Coron. Optimal security proofs for PSS and other signature schemes. In *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 2002.
7. R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000.
8. T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.
9. C. Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.
10. S. Goldwasser and R. Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *CRYPTO 1992*, pages 228–245, 1992.
11. I. Haitner and T. Holenstein. On the (im)possibility of key dependent encryption. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 202–219. Springer, 2009.
12. K. Haralambiev, T. Jager, E. Kiltz, and V. Shoup. Simple and efficient public-key encryption from computational diffie-hellman in the standard model. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.
13. D. Hofheinz, T. Jager, and E. Knapp. Waters signatures with optimal security reduction. In *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 66–83. Springer, 2012.
14. S. Hohenberger and B. Waters. Short and stateless signatures from the RSA assumption. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670. Springer, 2009.

15. S. A. Kakvi and E. Kiltz. Optimal security proofs for full domain hash, revisited. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553. Springer, 2012.
16. B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.
17. D. W. Kravitz. Digital signature algorithm, July 27 1993. US Patent 5,231,668.
18. H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*. The Internet Society, 2000.
19. A. Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612. Springer, 2002.
20. S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.
21. A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.
22. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems (reprint). *Commun. ACM*, 26(1):96–99, 1983.
23. B. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.
24. D. Wichs. Barriers in cryptography with weak, correlated and leaky sources. In *ITCS*, pages 111–126. ACM, 2013.

A Lower Bounds for Unique-Key PKE and Signature Schemes in the Multi-User Setting

In this section, we give the security notions of unique-key signatures and encryption schemes in the multi-user setting. Then we show two memory lower bounds of restricted reductions, which are extensions of the result in Section 4.

A.1 Unique-Key PKE Schemes and Signatures

Unique-key primitives. A cryptographic primitive (which can be PKE scheme, signature scheme, trapdoor commitment scheme (with collision resistance), etc.) with key generation algorithm Gen is called a *unique-key primitive* if there exists some algorithm Check such that $(\mathsf{Gen}, \mathsf{Check})$ forms a unique-key relation (see Definition 7). We now recall the definition of PKE schemes and define unique-key signatures and unique-key PKE schemes as follows.

Definition 15 (Public key encryption (PKE)). *A PKE scheme consists of the PT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. *(a)* Gen *is a probabilistic algorithm that takes as input* 1^λ*, and returns a public/secret key pair* (pk, sk). *(b)* Enc *is a probabilistic algorithm that takes as input a public key* pk *and a message* $m \in \{0,1\}^\delta$*, and returns a ciphertext* ct. *(c)* Dec *is a deterministic algorithm that takes as input a secret key* sk *and a ciphertext* ct*, and returns a message* $m \in \{0,1\}^\delta$ *or* $\perp$.

A PKE scheme is required to satisfy correctness*, which means that* $\mathsf{Dec}_{sk}(ct) = m$ *holds for all* $\lambda \in \mathbb{N}$*, all* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$*, all* $m \in \{0,1\}^\delta$*, and all* $ct \leftarrow \mathsf{Enc}_{pk}(m)$.

Definition 16 (Unique-key signature and PKE). *A signature (respectively, PKE) scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *(respectively,* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*) is said to have the unique-key property if there exists a deterministic PT algorithm* Check *such that* $(\mathsf{Gen}, \mathsf{Check})$ *is a unique-key relation.*

Now we define the security notions for unique-key signatures and PKE schemes. We denote mUF security and UF security in the multi-user setting by mU-mUF and mU-UF. Moreover, we overload the notions mU-mOW and mU-OW (defined for unique-key relations) so that they apply to PKE schemes.

Definition 17 (mU-mUF). *A unique-key signature scheme* $(\mathsf{Gen}, \mathsf{Check}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be* mU-mUF secure *if for any PPT adversary* $\mathcal{A}$*, we have* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}mUF}}(\lambda) = \Pr[\mathcal{CH}$ *outputs* $1] \leq negl(\lambda)$ *in the following game.*

1. *The challenger* $\mathcal{CH}$ *sets* $w = 0$, $\mathcal{Q} = \emptyset$, *and* $\mathcal{Q}_s = \emptyset$, *and runs* $\mathcal{A}$ *on input* 1^λ. *Then* $\mathcal{A}$ *may make sampling queries to* $\mathcal{CH}$, *and* $\mathcal{CH}$ *responds as follows.*
   - *On receiving the ith sampling query sp,* $\mathcal{CH}$ *samples* $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ *and sends* pk_i *to* $\mathcal{A}$.

   Then $\mathcal{A}$ *may make adaptive corruption, signing, and verification queries to* $\mathcal{CH}$, *and* $\mathcal{CH}$ *responds as follows:*
   - *On receiving a corruption query* i, $\mathcal{CH}$ *adds* i *to* $\mathcal{Q}$, *and sends* sk_i *to* $\mathcal{A}$.
   - *On receiving a signing query* (i, m), $\mathcal{CH}$ *computes* $\sigma \leftarrow \mathsf{Sign}_{sk_i}(m)$, *adds* (i, m) *to* $\mathcal{Q}_s$, *and sends* σ *to* $\mathcal{A}$.
   - *On receiving a verification query* (i^*, m^*, σ^*), *if* $\mathsf{Verify}_{pk_{i^*}}(m^*, \sigma^*) = 1$, $i^* \notin \mathcal{Q}$, *and* $(i^*, m^*) \notin \mathcal{Q}_s$, $\mathcal{CH}$ *sets* $w = 1$.
2. *At some point,* $\mathcal{A}$ *makes a stopping query stp to* $\mathcal{CH}$, *and* $\mathcal{CH}$ *returns* w.

Definition 18 (mU-mOW (for PKE)). *A unique-key PKE scheme* $(\mathsf{Gen}, \mathsf{Check}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be* mU-mOW secure *if for any PPT adversary* $\mathcal{A}$*, we have* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}mOW}}(\lambda) = \Pr[\mathcal{CH}$ *outputs* $1] \leq negl(\lambda)$ *in the following game.*

1. *The challenger* $\mathcal{CH}$ *sets* $w = 0$, $\mathcal{Q} = \emptyset$, *and* $\mathcal{Q}_m = \emptyset$, *and runs* $\mathcal{A}$ *on input* 1^λ. *Then* $\mathcal{A}$ *may make sampling queries to* $\mathcal{CH}$, *and* $\mathcal{CH}$ *responds as follows.*
   - *On receiving the ith sampling query sp,* $\mathcal{CH}$ *samples* $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$ *and sends* pk_i *to* $\mathcal{A}$.
2. $\mathcal{A}$ *may make adaptive corruption and challenge queries to* $\mathcal{CH}$, *and* $\mathcal{CH}$ *responds as follows:*
   - *On receiving a corruption query* i, $\mathcal{CH}$ *adds* i *to* $\mathcal{Q}$, *and sends* sk_i *to* $\mathcal{A}$.
   - *On receiving a challenge query* i, $\mathcal{CH}$ *searches* $(i, m) \in \mathcal{Q}_m$. *If the searching procedure fails,* $\mathcal{CH}$ *runs* $m \leftarrow \{0,1\}^\delta$ *and adds* (i, m) *to* $\mathcal{Q}_m$. *Then it returns* $ct \leftarrow \mathsf{Enc}_{pk_i}(m)$ *to* $\mathcal{A}$.
   - *On receiving a verification query* (i^*, m') *from* $\mathcal{A}$. *If* $i^* \notin \mathcal{Q}$ *and* $(i^*, m') \in \mathcal{Q}_m$, $\mathcal{CH}$ *sets* $w = 1$.
3. *At some point,* $\mathcal{A}$ *makes a stopping query stp to* $\mathcal{CH}$, *and* $\mathcal{CH}$ *returns* w.

Definition 19 (mU-UF and mU-OW (for PKE)). *The definitions of* mU-UF *security (respectively,* mU-OW *security for PKE) is defined in exactly the same way as* mU-mUF *security (respectively,* mU-mOW *security for PKE) except that* $\mathcal{A}$ *is allowed to make only one verification query and the advantage of* $\mathcal{A}$ *is denoted by* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}UF}}(\lambda)$ *(respectively,* $\mathbf{Adv}^{\mathcal{A}}_{\mathsf{mU\text{-}OW}}(\lambda)$*).*

A.2 Lower Bounds for Unique-Key PKE Schemes and Signatures

We now show two memory lower bounds for restricted reductions respectively from $\mathsf{mU\text{-}mUF}$ security to $\mathsf{mU\text{-}UF}$ security and $\mathsf{mU\text{-}mOW}$ security to $\mathsf{mU\text{-}OW}$ security. The definition of the latter type of restricted reductions is exactly the same as Definition 9. The definition of the former type is also the same as Definition 9 except that the following restriction is additionally required.

– The set of signing queries made by $\mathcal{R}$ is the same as the set of all signing queries made by $\mathcal{A}$.[10]

These two by-product results can be treated as two examples of memory lower bounds derived from our lower bound result for unique-key relations stated in Theorem 4.

Corollary 1. *Let λ be a (sufficiently large) security parameter, $\Sigma = (\mathsf{Gen}, \mathsf{Check}, \mathsf{Sign}, \mathsf{Verify})$, where the internal randomness space of Gen is $\{0,1\}^\rho$, be a $\mathsf{mU\text{-}UF}$ secure unique-key signature scheme, $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\lambda \to \{0,1\}^\rho$ be a PRF, and $\mathcal{R}$ be a c_r-restricted black-box reduction from the $\mathsf{mU\text{-}mUF}$ security to the $\mathsf{mU\text{-}UF}$ security of Σ. Let $n = n(\lambda)$ be the maximum number of sampling queries and $q = q(\lambda)$ be the maximum numbers of corruption and verification queries made by an adversary in the $\mathsf{mU\text{-}mUF}$ game, and $\mathcal{U} = \{i\}_{i=1}^n$. If (a) $\mathcal{R}$ rewinds the adversary for at most $p = p(\lambda)$ times and (b) $c_r > 1/2$, then we have*

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{R}) = {}& \Omega(\max\{q/(p+2), n/(p+2)\}) - O(\log q) - O(\log n) - \kappa \\ & - \max\{\mathbf{LocalMem}(\mathsf{Gen}), \mathbf{LocalMem}(\mathsf{Check}), \\ & \qquad \mathbf{LocalMem}(\mathsf{Sign}), \mathbf{LocalMem}(\mathsf{Verify}), \mathbf{LocalMem}(\mathsf{F})\}.\end{aligned}$$

Corollary 2. *Let λ be a (sufficiently large) security parameter, $\Pi = (\mathsf{Gen}, \mathsf{Check}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$, where the internal randomness space of Gen is $\{0,1\}^\rho$, be a $\mathsf{mU\text{-}OW}$ secure unique-key PKE scheme, $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\lambda \to \{0,1\}^\rho$ and $\mathsf{F}' : \{0,1\}^\kappa \times \{0,1\}^\lambda \to \mathcal{M}$ be PRFs, and $\mathcal{R}$ be a c_r-restricted black-box reduction from the $\mathsf{mU\text{-}mOW}$ security to the $\mathsf{mU\text{-}OW}$ security of Π. Let $n = n(\lambda)$ be the maximum number of sampling queries and $q = q(\lambda)$ be the maximum numbers of corruption, challenge, and verification queries made by an adversary in the $\mathsf{mU\text{-}mOW}$ game, and $\mathcal{U} = \{i\}_{i=1}^n$. If (a) $\mathcal{R}$ rewinds the adversary for at most $p = p(\lambda)$ times and (b) $c_r > 1/2$, then we have*

$$\begin{aligned}\mathbf{LocalMem}(\mathcal{R}) = {}& \Omega(\max\{q/(p+2), n/(p+2)\}) - O(\log q) - O(\log n) - 2\kappa \\ & - \max\{\mathbf{LocalMem}(\mathsf{Gen}), \mathbf{LocalMem}(\mathsf{Check}), \mathbf{LocalMem}(\mathsf{Enc}), \\ & \qquad \mathbf{LocalMem}(\mathsf{Dec}), \mathbf{LocalMem}(\mathsf{F}), \mathbf{LocalMem}(\mathsf{F}')\}.\end{aligned}$$

[10] This restriction is made due to the fact that if the signing queries are chosen by $\mathcal{R}$, then the challenger may consume large memory to store them, which spoils our result. When considering random message attacks, this restriction can be removed. Also, this restriction is not made for other primitives such as PKE schemes, trapdoor commitment, and chameleon hash function schemes (with collision resistance).

We omit the proofs of the above two corollaries since they are very similar to the proof of Theorem 4. The main difference is that instead of directly giving secret keys to $\mathcal{R}$ as verification queries, the adversary playing the $\mathsf{mU\text{-}mUF}$ or $\mathsf{mU\text{-}mOW}$ game uses the secret keys to forge signatures or decrypt challenge ciphertexts. Moreover, the $\mathsf{mU\text{-}OW}$ challenger uses F' to simulate the random messages chosen for challenge queries so that it does not have to consume large memory to store the list $\mathcal{Q}_m$.