Digital Signatures with Memory-Tight Security in the Multi-Challenge Setting

Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu

University of Wuppertal, Wuppertal, Germany
`{denis.diemert, kai.gellert, tibor.jager, lin.lyu}@uni-wuppertal.de`

Abstract. The standard security notion for digital signatures is "single-challenge" (SC) EUF-CMA security, where the adversary outputs a single message-signature pair and "wins" if it is a forgery. Auerbach *et al.* (CRYPTO 2017) introduced *memory-tightness* of reductions and argued that the right security goal in this setting is actually a stronger "multi-challenge" (MC) definition, where an adversary may output many message-signature pairs and "wins" if at least one is a forgery. Currently, no construction from simple standard assumptions is known to achieve full tightness with respect to time, success probability, and memory simultaneously. Previous works showed that memory-tight signatures cannot be achieved via certain natural classes of reductions (Auerbach *et al.*, CRYPTO 2017; Wang *et al.*, EUROCRYPT 2018). These impossibility results may give the impression that the construction of memory-tight signatures is difficult or even impossible.

We show that this impression is false, by giving the first constructions of signature schemes with full tightness in all dimensions in the MC setting. To circumvent the known impossibility results, we first introduce the notion of *canonical reductions* in the SC setting. We prove a general theorem establishing that every signature scheme with a canonical reduction is already memory-tightly secure in the MC setting, provided that it is strongly unforgeable, the adversary receives only one signature per message, and assuming the existence of a tightly-secure pseudorandom function. We then achieve memory-tight *many-signatures-per-message* security in the MC setting by a simple additional generic transformation. This yields the first memory-tightly, strongly EUF-CMA-secure signature schemes in the MC setting. Finally, we show that standard security proofs often already can be viewed as canonical reductions. Concretely, we show this for signatures from lossy identification schemes (Abdalla *et al.*, EUROCRYPT 2012), two variants of RSA Full-Domain Hash (Bellare and Rogaway, EUROCRYPT 1996), and two variants of BLS signatures (Boneh *et al.*, ASIACRYPT 2001).

1 Introduction

Work-factor-tightness. The security of many cryptosystems depends on computational hardness assumptions, where security is proven by a reduction from

Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823.

breaking the cryptosystem with respect to some security definition to breaking the hardness assumption. When such cryptosystems are concretely instantiated, cryptographic parameters such as the size of algebraic groups and moduli must be determined. If this is done in theoretically-sound way, that is, supported by the security guarantees provided by a reduction from breaking the cryptosystem to breaking the underlying assumption, then the *security loss* of the reduction has to be taken into account.

Let $\mathcal{A}$ be an adversary on a given cryptosystem with respect to a given security model, and let $\mathcal{R}$ be a reduction in a security proof that turns $\mathcal{A}$ into an algorithm solving some assumed-to-be-hard computational problem. Let $(t_{\mathcal{A}}, \epsilon_{\mathcal{A}})$ and $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}})$ be the running time and advantage of $\mathcal{A}$ and $\mathcal{R}$, respectively. Then, the security loss is defined as L such that

$$L \cdot \frac{\epsilon_{\mathcal{R}}}{t_{\mathcal{R}}} = \frac{\epsilon_{\mathcal{A}}}{t_{\mathcal{A}}}$$

where $\epsilon_{\mathcal{A}}/t_{\mathcal{A}}$ and $\epsilon_{\mathcal{R}}/t_{\mathcal{R}}$ are the *work factors* of $\mathcal{A}$ and $\mathcal{R}$, respectively.[1] This is the standard approach to measure concrete security, which was established by Bellare and Ristenpart [9, 10].

In the classical asymptotic setting a reduction is considered *efficient* if L is bounded by some polynomial, which may be large. However, if L is large, then a theoretically-sound concrete instantiation must compensate the security loss with larger parameters, at the cost of efficiency of the deployed cryptosystem. Often L depends on deployment parameters (such as the number of users and the number of issued signatures, for instance), which are determined by the application context. These might not be exactly known at the time of initial deployment, or they might unexpectedly encounter significant increase over time. Hence, these parameters must be chosen conservatively, based on a strict upper bounds, which may lead to overly large parameters that come with very significant performance overhead. Therefore it is desirable to have *tight* security proofs, where L is a constant, and thus independent of such deployment parameters. Such schemes can be efficiently instantiated with optimal cryptographic parameters in arbitrary application contexts, independent of the number of users, the number of issued signatures, and other application parameters. If L is a constant, then we usually call $\mathcal{R}$ a *tight* reduction. In this paper, we will refer to this notion as *work-factor-tightness*, in order to distinguish it from the notion of *memory-tightness* discussed below.

Memory-tightness. Auerbach *et al.* [5] explained that in addition to the work factor also the *memory* consumed by a reduction is relevant. This is particularly relevant when security is reduced to so-called *memory-sensitive* computational problems, where the efficiency of known algorithms depends on the amount of

[1] In the asymptotic setting, $\epsilon_{\mathcal{A}}$, $t_{\mathcal{A}}$, $\epsilon_{\mathcal{R}}$, and $t_{\mathcal{R}}$ are functions in a security parameter. In this case L is a function in the security parameter, too. In the concrete security setting the running times, success probabilities, and the security loss are real numbers.

memory that is available. This includes, for instance, known algorithms for the classical discrete logarithm problem modulo a prime number, the integer factorization problem, Learning With Errors (LWE), or Short Integer Solutions (SIS), and many more. Other problems are (currently) not considered memory-sensitive, such as the discrete logarithm problem in elliptic curve groups. However, whether a given computational problem is memory-sensitive or not may change with the discovery of new algorithms and the impact of memory on their performance. See [5] for an in-depth discussion of memory-sensitivity.

In order to address this gap, Auerbach *et al.* [5] introduced the notion of *memory-tightness*, which additionally takes the memory consumed by a reduction into account. In addition to discussing the memory-sensitivity of computational problems, they also consider the memory-tightness of finding multi-collisions for hash functions and of reductions between different security notions of digital signature schemes.

Since its introduction in 2017, the concept of memory tightness has drawn much attention and led to many follow-up works. This includes works on memory lower bounds of reductions by Wang *et al.* [53] (EUROCRYPT 2018), memory tightness of authenticated encryption by Ghoshal, Jaeger, and Tessaro [32] (CRYPTO 2020), memory tightness of hashed ElGamal by Ghoshal and Tessaro [33] (EUROCRYPT 2020), and memory tightness for key encapsulation mechanisms by Bhattacharyya [13] (PKC 2020). Hence, memory tightness is already a well-established concept in cryptography that receives broad interest.

Memory-tightly secure signatures. In the standard *existential unforgeability under chosen-message attacks* (EUF-CMA) security model, the adversary receives a public key pk and then has access to a signing oracle that, on input of any message m from the message space of the signature scheme, computes a signature $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$, stores m in a list $\mathcal{Q}$, and returns σ. The adversary successfully breaks the security of the signature scheme if it outputs a forgery (m^*, σ^*) such that σ^* is a valid signature for m^* with respect to pk, and $m^* \notin \mathcal{Q}$. Auerbach *et al.* call this the *single-challenge* setting, since the adversary has only one attempt to forge a signature. They also introduce a stronger *multi-challenge* security definition, where the adversary may output multiple valid message-signature pairs and it “wins” if at least one of them is a new forgery in the sense that no signature was requested for the corresponding message throughout the security experiment.

Obviously, when considering the random-access memory (RAM) model, both security notions are tightly equivalent when memory consumption is not considered. In one direction, given a multi-challenge adversary, one can simply store all message-signature pairs that the adversary has obtained from its experiment in a list. Whenever the adversary outputs a message-signature pair, it is checked whether it is contained in the list. If not, then it is a valid forgery in the single-challenge setting. The opposite direction is even more trivial. However, note that this reduction is not memory-tight, as it requires memory linear in the number of signing queries. Auerbach *et al.* even showed that it is very difficult to prove that both notions are memory-tightly equivalent, by giving an impossibility result that covers a large class of natural reductions. This result was subsequently revisited and extended by Wang *et al.* [53].

The only known construction of a signature scheme with memory-tight security proof is due to Auerbach *et al.* [5]. They show that the RSA full-domain hash signature scheme can be proven memory-tightly secure under the RSA assumption. This is already a significant result, since it introduces clever tricks to deal with a programmable random oracle in a memory-tight way. However, it is still limited, since the reduction is only memory-tight, but not work-factor-tight. This is because the tightness lower bounds from [7,21,44,45] still apply, such that a linear security loss in the number of signature queries is unavoidable.[2] Furthermore, Auerbach *et al.* only achieve memory-tightness in the weaker single-challenge setting, but not yet in the stronger multi-challenge setting. To the best of our knowledge, there exists currently no signature scheme, which has a security proof that is *fully* tight, that is, simultaneously memory-tight *and* work-factor-tight.

One main difficulty of achieving memory-tightly-secure signatures in the multi-challenge setting is to build a reduction which does not have to store the sequence of random oracle queries made by the adversary. While it seems easy to replace a random oracle with a pseudorandom function, this must be done very carefully, in particular in security proofs that "program" a random oracle, in order to achieve consistency. Here we can partially build upon techniques developed by Auerbach *et al.* [5]. Furthermore, another major difficulty in achieving security in the multi-challenge setting is to build a reduction which does not have to store the history of message-signature pairs obtained by the adversary through signing queries.

Our contributions. We summarize our contributions as follows.

- We present a sequence of transforms that give rise to the first digital signature schemes that simultaneously achieve tightness in all three dimensions: running time, success probability, and memory. The construction is efficient and yields practical signature schemes.
- On a technical level, we show how to circumvent known impossibility result by introducing the notion of "canonical reductions", which can be seen as a new "non-black-box" perspective that applies to many well-known standard reductions in security proofs for signature schemes.
- We show the applicability of this approach by considering the construction of signatures from lossy identification schemes (LID) by Abdalla *et al.* [2,3], which can be viewed as a generalization of the security proof for Katz–Wang signatures [46]. We further demonstrate the versatility of our technique by applying it to well-known signature schemes like RSA-FDH [12] (with the proof following [21] with a loss linear in the number of signing queries). Then, we additionally show that by using the technique by Katz and Wang [46] of

[2] There is also a work-factor-tight security proof for RSA full domain hash based on the Phi Hiding assumption [44,45], but this proof seems not compatible with the memory-tight implementation of the random oracle from [5].

signing the message together with an extra random bit, we can eliminate the linear security loss and achieve both memory and working factor tightness. We also show similar results for Boneh–Lynn–Shacham (BLS) signatures [15]. All of our results directly achieve *strong* unforgeability. For a comparison of our result with previous analyses of these scheme, consider Table 1.

Table 1. Comparison of our result to previous analyses of the considered schemes. All analyses are in the random oracle model. Let λ be the security parameter, let q_H be the number of random oracle queries, let q_S the number of signing queries, let e be the basis of the natural logarithm, let $|\mathbb{G}|$ be the size of the representation of a group element of a cyclic group $\mathbb{G}$ of prime order q, let $|\mathbb{Z}_N|$ denote the size of the representation of an element of $\mathbb{Z}_N$, let N be a RSA modulus, let e be a RSA public exponent, and let $|\mathbb{G}_1|$ (resp. $|\mathbb{G}_2|$) be the size of the representation of a group element of group $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) of some bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. Note that for comparability, we chose to instantiate the LID-based schemes with DDH. Due to collision resistance, the nonce length chosen for our transform from Section 4 is 2λ.

Constr.	**Proof**	**Asm.**	**Sec.**	**Sec. Loss**	**Mem. Loss**	$\boldsymbol{\lvert pk\rvert}$	$\boldsymbol{\lvert\sigma\rvert}$
LID-based	[2,3]	DDH	EUF-CMA	$\mathcal{O}(1)$	$\mathcal{O}(q_\mathsf{H}+q_\mathsf{S})$	$4\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{Z}_q\rvert$
	Ours	DDH	msEUF-CMA	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$4\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{Z}_q\rvert+2\lambda$
RSA-FDH	[20]	RSA	EUF-CMA	$\mathrm{e}\cdot q_\mathsf{S}$	$\mathcal{O}(q_\mathsf{H}+q_\mathsf{S})$	$\lvert N\rvert+\lvert e\rvert$	$\lvert\mathbb{Z}_N\rvert$
	[5]	RSA	EUF-CMA	$\mathrm{e}\cdot q_\mathsf{S}$	$\mathcal{O}(1)$	$\lvert N\rvert+\lvert e\rvert$	$\lvert\mathbb{Z}_N\rvert$
	Ours	RSA	msEUF-CMA	$\mathrm{e}\cdot q_\mathsf{S}$	$\mathcal{O}(1)$	$\lvert N\rvert+\lvert e\rvert$	$\lvert\mathbb{Z}_N\rvert$
RSA-FDH+	[46]	RSA	EUF-CMA	$\mathcal{O}(1)$	$\mathcal{O}(q_\mathsf{H}+q_\mathsf{S})$	$\lvert N\rvert+\lvert e\rvert$	$\lvert\mathbb{Z}_N\rvert$
	Ours	RSA	msEUF-CMA	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\lvert N\rvert+\lvert e\rvert$	$\lvert\mathbb{Z}_N\rvert+2\lambda$
BLS	[15]	(co-)CDH	EUF-CMA	$\mathrm{e}\cdot(q_\mathsf{S}+1)$	$\mathcal{O}(q_\mathsf{H}+q_\mathsf{S})$	$\lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_1\rvert$
	Ours	(co-)CDH	msEUF-CMA	$\mathrm{e}\cdot(q_\mathsf{S}+1)$	$\mathcal{O}(1)$	$\lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_1\rvert$
BLS+	[46]	(co-)CDH	EUF-CMA	$\mathcal{O}(1)$	$\mathcal{O}(q_\mathsf{H}+q_\mathsf{S})$	$\lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_1\rvert$
	Ours	(co-)CDH	msEUF-CMA	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\lvert\mathbb{G}_2\rvert$	$\lvert\mathbb{G}_1\rvert+2\lambda$

Our approach. Our approach can be divided into two steps.

1. At first we show how to generically transform an entire class of signature schemes from the single-challenge setting to the multi-challenge setting. During this step, it is actually useful to consider a weaker "one-signature-per-message" security notion, where an adversary may only request one (instead of many) signature per message via its signing oracle.[3]

[3] Of course, one-signature-per-message security is equivalent to standard security for signature schemes with deterministic signing algorithm, however, we are not aware of any such signature scheme which achieves tight security, not even in the clas-

We require that the security reduction of the underlying scheme follows a canonical pattern that is compatible with our approach to prove memory tightness. Essentially, we require that the reduction can be split into *stateless* "canonical procedures" for simulating signatures, extracting solutions from forgeries, and computing hash values (e.g., if a random oracle is needed).
The main idea is now to "de-randomize" all canonical procedures, meaning that we give all procedures access to the *same* random function but require that they otherwise behave deterministically. Note that the "one-signature-per-message" restriction helps us here, as the procedures can rely on the random function to derive randomness for one signature per message from the message by calling the random function. Giving all procedures access to the same random function, ensures consistency across procedures (e.g., a signature may need to be consistent with the simulation of a random oracle).
We also show that many standard security proofs for signatures indeed can be seen as canonical reductions, so that our generic result applies.
Finally, to generically achieve memory-tightness in the multi-challenge setting, we can replace the "global random function" with a pseudorandom function. This yields a generic transform (with tightness in all dimensions) producing a signature scheme secure in the "one-signature-per-message" and multi-challenge setting.

2. In the second step we apply a simple generic transform (again, with tightness in all dimensions) that lifts any signature scheme from the "one-signature-per-message" to the standard "many-signatures-per-message" setting. To this end, any message is signed alongside a random nonce, which intuitively "expands" the set of valid signatures per message.

Applying both steps sequentially does not influence the tightness of a signature scheme in any dimension.

Related work. In the literature, "tightness" usually refers to what we call work-factor tightness in this paper. That is, running times and success probabilities are considered, but memory is not. There is a large number of research results in this area, with tightly-secure constructions of many different types of cryptosystems, including digital signatures [23, 39, 40, 46, 51], public-key encryption [8, 30, 39], (hierarchical) identity-based encryption [14, 18], authenticated key exchange [6, 19, 34, 48], and symmetric encryption [36, 38, 43], for instance. Tight security is also increasingly considered for real-world cryptosystems, such as [22, 24, 36, 41]. There are also various impossibility results for different types and classes of cryptosystems, such as [21, 27–29, 43–45, 50, 52], for instance.

As already mentioned, the notion of memory-tightness was only relatively recently introduced in [5]. They also introduced the single- and multi-challenge security model, and gave the first (and currently only) memory-tight security proof for a digital scheme in the weaker single-challenge setting, which however

sical sense that does not consider memory tightness. There are several impossibility results, showing that tightness is often difficult to achieve for such signature schemes [7, 21, 45].

is not yet work-factor-tight. They also gave a first impossibility result, showing that a certain class of reductions cannot be used to reduce multi-challenge security to single-challenge security. Wang *et al.* [53] revisited this impossibility result and showed that multi-challenge security is impossible to achieve for a large class of reductions, unless a work-factor tightness is sacrificed. They showed a lower bound on the memory of a large class of black-box reductions from the multi-challenge unforgeability of unique signatures to any computational hardness assumption, another lower bound for restricted reductions from multi-challenge security to single-challenge security for cryptographic primitives with unique keys, and a lower bound for multi-collisions of hash functions with large domain, which extends a similar result from [5]. Bhattacharyya [13] and Ghoshal and Tessaro [33] independently considered the memory-tightness of hashed ElGamal public-key encryption. Ghoshal, Jaeger, and Tessaro [32] considered the memory-tightness of authenticated encryption.

In independent and concurrent work, Ghoshal, Ghosal, Jaeger and Tessaro [31] study how to construct signature schemes with memory-tight security in the multi-challenge setting. More precisely, they show a memory-tight way to upgrade a signature scheme DS with $\mathsf{EUF\text{-}CMA}$ security into another signature scheme RDS with $\mathsf{mEUF\text{-}CMA}$ security assuming the existence of pseudorandom tweakable permutations. Their construction resembles ours in Section 4, which we use to upgrade from the one-signature-per-message setting to the many-signature-per-message setting. Their result complements ours, since they rely on general reductions but not canonical reductions, but achieves only standard (non-strong) $\mathsf{mEUF\text{-}CMA}$ security.

Outline. The remainder of this paper is organized as follows. In Section 2, we define the computational model and the used complexity measures, alongside with standard definitions of cryptographic primitives. In Section 3, we present how to achieve multi-challenge security from any signature scheme secure in the single-challenge setting that follows a canonical reduction. In Section 4, we present our generic transform to lift any signature scheme from "one-signature-per-message" to the standard "many-signatures-per-message" setting. Finally, we show how our transforms can be applied to existing signature schemes, achieving the first fully tight signature schemes in the multi-challenge setting.

2 Preliminaries

For strings a and b, we denote the concatenation of these strings by $a \parallel b$. We denote the operation of assigning a value y to a variable x by $x := y$. If S is a finite set, we denote by $x \xleftarrow{\$} S$ the operation of sampling a value uniformly at random from set S and assigning it to variable x. For any probabilistic algorithm $\mathcal{A}$, we denote $y \leftarrow \mathcal{A}(x; r)$ the process of running $\mathcal{A}$ on input x with random coins r and assign the output to y, and we denote $y \xleftarrow{\$} \mathcal{A}(x)$ as $y \leftarrow \mathcal{A}(x; r)$ for uniformly random r.

2.1 Computational Model and Complexity Measures

In this paper, we adapt the computation model used in [5] and recall the most important aspects in this section.

Algorithms. We assume all algorithms in this paper to be random access machines (RAMs). A RAM has access to memory using words of a fixed size λ and a constant number of registers each holding a single word. If an algorithm $\mathcal{A}$ is probabilistic, then the corresponding RAM is equipped with a special instruction that fills a distinguished register with (independent) random bits. However, we do not allow the RAM to rewind random bits to access previously used random bits. That is, $\mathcal{A}$ needs to store the random bits in this case. To run algorithm $\mathcal{A}$, the RAM is executed, where the input of the algorithm is written in the RAM's memory. To denote this, we overload notation and write $x \xleftarrow{\$} \mathcal{A}(y_1, y_2, \dots)$ to denote that random variable x takes on the value of algorithm $\mathcal{A}$ ran on inputs $y_1, y_2, \dots$ with fresh random coins. Sometimes we also denote this random variable simply by $\mathcal{A}(y_1, y_2, \dots)$. In case $\mathcal{A}$ is deterministic, we write $x \coloneqq \mathcal{A}(y_1, y_2, \dots)$, to denote that $\mathcal{A}$ on inputs $y_1, y_2, \dots$ outputs x.

Oracles. In addition, algorithm $\mathcal{A}$ sometimes has access to (stateful) oracles $(\mathcal{O}_1, \mathcal{O}_2, \dots)$. Each of these oracles also is defined by a RAM. To interact with an oracle $\mathcal{O}_i$, the RAM of algorithm $\mathcal{A}$ has three fixed regions in the memory only used for the oracle state $st_{\mathcal{O}}$, the input to the oracle and the output of the oracle. By default, these regions are empty. To query the oracle $\mathcal{O}_i$, $\mathcal{A}$ writes the query in the region of its memory reserved for the oracle input and executes a special instruction to run the RAM of $\mathcal{O}_i$ on this input together with the oracle state $st_{\mathcal{O}}$. The RAM implementing $\mathcal{O}_i$ uses its own memory and both the output and the updated oracle state $st_{\mathcal{O}}$ in the designated regions in $\mathcal{A}$'s memory. For notation, we denote that an algorithm $\mathcal{A}$ has oracle access to an algorithm *oracle* by $\mathcal{A}^{\mathcal{O}}$.

Security experiment. The security definition and proofs presented in this paper are mostly game-based. A security experiment (or game) can simply be viewed as an algorithm that runs another algorithm as subroutine, e.g., an adversary $\mathcal{A}$, and the subroutine may also be provided with a series of (stateful) oracles. As a security experiment is simply an algorithm it is also implemented by a RAM.

Complexity measures for runtime and memory consumption. We define the complexity measures for runtime and memory according to Auerbach *et al.* [5].

Runtime. Let $\mathcal{A}$ be an algorithm and Exp be a security game. We define $\mathbf{Time}(\mathcal{A})$ to be the runtime of $\mathcal{A}$ as the worst-case number of computation steps over all inputs of length λ and all possible random choices. In addition, we define $\mathbf{LocalTime}(\mathcal{A})$ to be the number of computation steps of $\mathcal{A}$ playing Exp *without* the additional steps induced by the oracle access to Exp. This quantifier allows us to precisely measure how much additional computation steps are necessary per oracle.

Memory consumption. Let $\mathcal{A}$ be an algorithm and Exp be a security game. We define $\mathbf{Mem}(\mathcal{A})$ to be the memory (in λ-width words) of the code of $\mathcal{A}$ plus the worst-case number of registers used at any point during computation, over all inputs of length λ and all possible random choices. Similar to before, we define $\mathbf{LocalMem}(\mathcal{A})$ to be the memory required to execute Exp with algorithm $\mathcal{A}$ *without* the additional memory induced by the oracle access to Exp. This quantifier allows us to precisely measure how much additional memory is necessary per oracle.

2.2 Pseudorandom Functions

We recall the standard indistinguishability definition for pseudorandom functions. This is one of the main tools used to make reductions memory-tight.

Definition 1. *Let $\lambda \in \mathbb{N}$. Let $\mathsf{F}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \mathcal{R}$ be a keyed function, where $\mathcal{R}$ is a finite set. We define the advantage of an adversary $\mathcal{A}$ in breaking the pseudorandomness of F as*

$$\mathsf{Adv}_{\mathsf{F}}^{\mathsf{PRF\text{-}sec}}(\mathcal{A}) := \left| \Pr\left[\mathcal{A}^{\mathsf{F}(k,\cdot)} = 1\right] - \Pr\left[\mathcal{A}^{f(\cdot)} = 1\right] \right|$$

where $k \xleftarrow{\$} \{0,1\}^\lambda$ and $f\colon \{0,1\}^* \to \mathcal{R}$ is a random function.

2.3 Digital Signatures

We recall the standard definition of a *digital signature scheme* by Goldwasser, Micali, and Rivest [35] and its standard security notion.

Definition 2. *A* digital signature scheme *for message space M is a triple of algorithms $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ such that*

1. Gen *is the randomized key generation algorithm generating a public (verification) key pk and a secret (signing) key sk and takes no input.*
2. $\mathsf{Sign}(sk, m)$ *is the randomized signing algorithm outputting a signature σ on input message $m \in M$ and signing key sk.*
3. $\mathsf{Vrfy}(pk, m, \sigma)$ *is the deterministic verification algorithm outputting either* 0 *or* 1.

We say that a digital signature scheme Sig *is* correct *if for any $m \in M$, and $(pk, sk) \xleftarrow{\$} \mathsf{Gen}$, it holds that $\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$.*

One-signature-per-message unforgeability of digital signature. We adapt the one-signature-per-message unforgeability defined by Fersch et al. [25]. First, we consider the "strong" variant of the definition given in [25], i.e., a pair (m, σ) output by the adversary is only considered a valid forgery if σ was not returned to the adversary as answer to an signing query m. In the "standard" variant, the pair is considered valid if for message m never a signature has been queried by the adversary. Second, we implement the fact that the adversary only receives one signature per message different to the original definition. Instead of aborting the whole experiment in case the adversary queries a signature for a message that it already received a signature for, we simply return the same signature to the adversary. Therefore, the adversary still gets only one signature per message, but is allowed to query a message multiple times.

We note that, for deterministic signature schemes, the one-signature-per-message security is equivalent to the many-signatures-per-message security.

Definition 3. *Let* $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a digital signature scheme. Consider the following experiment* $\mathsf{Exp}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A})$ *played between a challenger and an adversary* $\mathcal{A}$*:*

1. *The challenger initializes the set of chosen-message queries* $\mathcal{Q} := \emptyset$*, generates a fresh key pair* $(pk, sk) \xleftarrow{\$} \mathsf{Gen}$ *and forwards* pk *to the adversary as input.*
2. *The adversary may issue queries to the following oracle adaptively:*
   – $\mathsf{Sign}(m)$*: If* $(m, \sigma) \in \mathcal{Q}$*, the challenger returns* σ*. Otherwise, it returns* $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$ *and adds* (m, σ) *to* $\mathcal{Q}$*.*
3. *Finally, the adversary outputs a candidate forgery* (m, σ) *and the challenger outputs* 1 *if* $\mathsf{Vrfy}(pk, m, \sigma) = 1$ *and* $(m, \sigma) \notin \mathcal{Q}$*, and 0 otherwise.*

We denote the advantage of an adversary $\mathcal{A}$ in forging signatures for Sig in the $\mathsf{sEUF\text{-}CMA1}$ security experiment *by*

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}) = 1\right]$$

where $\mathsf{Exp}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A})$ *is as defined above.*

Next, we generalize Definition 3 to the multi-challenge setting. Unforgeability in the multi-challenge setting was proposed by Auerbach et al. [5] and is a generalized version of the standard existential unforgeability against chosen-message attackers notion, in which the adversary has additional access to a "forging oracle" allowing multiple forgery attempts. The adversary wins in this setting if at least one of the forgery attempts is "valid" in the same sense as in the single challenge setting.

Definition 4. *Let* $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a digital signature scheme. Consider the following experiment* $\mathsf{Exp}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A})$ *played between a challenger and an adversary* $\mathcal{A}$*:*

1. *The challenger initializes the set of chosen-message queries* $\mathcal{Q} := \emptyset$ *and the winning flag* $\mathsf{win} := 0$*. Then, it generates a fresh key pair* $(pk, sk) \xleftarrow{\$} \mathsf{Gen}$ *and forwards* pk *to the adversary as input.*
2. *The adversary may issue queries to the following oracles adaptively:*
   – $\mathsf{Sign}(m)$*: If* $(m, \sigma) \in \mathcal{Q}$ *for some* σ*, the challenger returns* σ*. Otherwise, it returns* $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$ *and adds* (m, σ) *to* $\mathcal{Q}$*.*
   – $\mathsf{Forge}(m, \sigma)$*: If* $\mathsf{Vrfy}(pk, m, \sigma) = 1$ *and* $(m, \sigma) \notin \mathcal{Q}$*, then set* $\mathsf{win} := 1$*.*
3. *Finally, the adversary halts and the experiment outputs* win*.*

We denote the advantage of an adversary $\mathcal{A}$ in forging signatures for Sig in the $\mathsf{msEUF\text{-}CMA1}$ security experiment *by*

$$\mathsf{Adv}_{\mathsf{Sig}}^{\mathsf{msEUF\text{-}CMA1}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}_{\mathsf{Sig}}^{\mathsf{msEUF\text{-}CMA1}}(\mathcal{A}) = 1\right]$$

where $\mathsf{Exp}_{\mathsf{Sig}}^{\mathsf{msEUF\text{-}CMA1}}(\mathcal{A})$ *is as defined above.*

Many-signatures-per-message unforgeability. The security notions $\mathsf{sEUF\text{-}CMA1}$ and $\mathsf{msEUF\text{-}CMA1}$ defined above can be generalized to the "many-signatures-per-message" setting by dropping the condition that the respective security experiments return σ if the Sign-oracle is queried with a message m such that $(m, \sigma) \in \mathcal{Q}$, i.e., a message m that was already queried before. Without this condition we obtain the standard strong existential unforgeability under chosen-message attacks ($\mathsf{sEUF\text{-}CMA}$) and its multi-challenge variant (as defined in [5]) $\mathsf{msEUF\text{-}CMA}$.

Adversary behavior. In this work we consider adversaries that are not necessarily well-behaved. That is, an adversary $\mathcal{A}$ may, for instance, submit a forgery (m^*, σ^*) such that σ^* was obtained by a signing query m^*. In principle, any such adversary can be converted to a well-behaved adversary by performing "sanity checks" whenever the adversary submits a forgery. This conversion, however, is not memory-tight as it leads to an increase in memory needed to store the set of chosen-message queries $\mathcal{Q}$.

Considering that there might exist adversaries that are not well-behaved but break the security of a signature scheme (e.g., by producing a forgery without knowing whether it is a fresh one), we prefer a stronger security notion and consider *any* adversary rather than restricting our proofs to a class of well-behaved adversaries. For a more detailed discussion on this topic, we refer the reader to [5, Section 2.3].

3 From the Single-Challenge Setting to the Multi-Challenge Setting

In this section, we will describe a generic construction of a reduction in the multi-challenge setting, based on any "canonical" reduction in the single-challenge setting.

3.1 Non-Interactive Computational Assumptions

The following definition of a non-interactive computational assumptions is based on the corresponding definition by Bader et al. [7], which is originally due to Abe et al. [4]. It captures both "search problems", such as CDH, and "decisional problems", such as DDH. We focus on *non-interactive* computational hardness assumptions, for the following reasons. First, these may be considered the most "interesting" hardness assumption when (memory) tightness is considered. Second, it makes the definitions and proofs significantly cleaner, and therefore makes it easier to understand and verify the core technical ideas and approach.

Definition 5. *A* non-interactive computational assumption *is defined as the tuple* $\Lambda = (\mathsf{InstGen}, \mathsf{V}, \mathsf{U})$*, where*

1. $(\phi, \omega) \xleftarrow{\$} \mathsf{InstGen}(1^\lambda)$*:* $\mathsf{InstGen}$ *is the probabilistic instance generation algorithm that takes as input a security parameter* 1^λ*, and outputs a problem instance* ϕ *and a witness* ω*.*
2. $0/1 := \mathsf{V}(\phi, \omega, \rho)$*:* V *is the deterministic verification algorithm that takes as input a problem instance* ϕ*, a witness* ω *and a candidate solution* ρ*, and outputs* 0 *or* 1*. We say that* ρ *is a correct solution for* ϕ *if* $\mathsf{V}(\phi, \omega, \rho) = 1$*.*
3. $\rho \xleftarrow{\$} \mathsf{U}(\phi)$*:* U *is a probabilistic algorithm that on input* ϕ *outputs a candidate solution* ρ*.*

We define the advantage of an adversary $\mathcal{R}$ *breaking* Λ *as*

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}) := \left| \Pr\left[\mathsf{Exp}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}) = 1\right] - c_\Lambda \right|$$

where the experiment $\mathsf{Exp}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{A})$ *generates* $(\phi, \omega) \xleftarrow{\$} \mathsf{InstGen}(1^\lambda)$*, runs* $\rho \xleftarrow{\$} \mathcal{A}(\phi)$ *then returns* $\mathsf{V}(\phi, \omega, \rho)$ *and the constant* $c_\Lambda := \Pr\left[\mathsf{Exp}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathsf{U}) = 1\right]$ *is called the* trivial advantage of Λ*. We further require that for any* (ϕ_0, ω_0) *output by* $\mathsf{InstGen}(1^\lambda)$*,* $\Pr\left[\mathsf{Exp}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathsf{U}) = 1 \;\middle|\; (\phi, \omega) = (\phi_0, \omega_0)\right] = c_\Lambda$*, which means the trivial advantage does not change for different problem instances.*

Intuitively, U can be seen as the "trivial" solution strategy. For example, if Λ is a decisional problem, such as DDH, U usually would output a uniformly random bit such that $c_\Lambda = \frac{1}{2}$. Then, $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R})$ basically defines the "bit-guessing advantage" against Λ. For a search problem, such as CDH, U would output a special symbol "$\perp$" such that $c_\Lambda = 0$. Then, $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R})$ corresponds to the probability of $\mathcal{R}$ finding a solution ρ for the given problem instance ϕ.

3.2 Canonical Reductions

We introduce the notion of a *canonical reduction*, which essentially defines an abstract pattern of a reduction which is "compatible" with our approach to prove memory-tight security. Many security proofs of signature schemes can be explained as canonical reductions, we will show some concrete examples below. We focus on reductions from $\mathsf{sEUF\text{-}CMA1}$-security to a non-interactive computational assumption Λ (as defined in Section 3.1) in both standard model and random oracle model. For an illustration of a canonical reduction, see Figure 1.

Definition 6. *Let* Sig *be a signature scheme and let* Λ *be a non-interactive computational assumption. Let* $(\mathsf{RGen}, \mathsf{RF}, \mathsf{RSign}, \mathsf{RExtract}, \mathsf{RHash})$ *be the following algorithms that are implemented by a canonical reduction:*

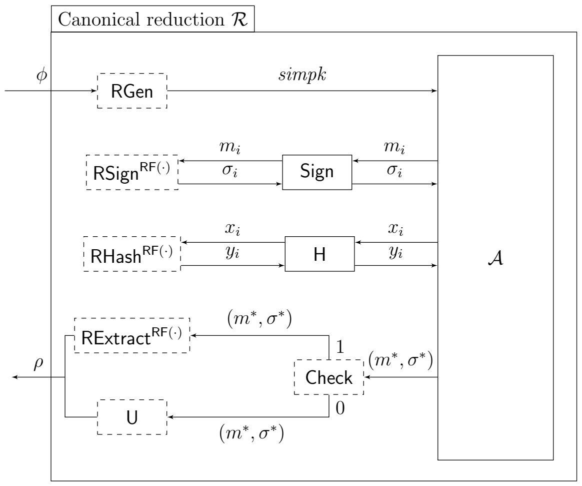

Fig. 1. Canonical reduction $\mathcal{R}$ from sEUF-CMA1-security of a signature scheme Sig to a computational assumption Λ with black-box access to an adversary $\mathcal{A}$. Check is a shorthand defined as $\mathsf{Check}(m^*, \sigma^*) = 1 \iff \mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \land \sigma^* \neq \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$ determining the algorithm to compute the final solution. For a complete formal definition, see Definition 6.

1. $(simpk, simsk) \xleftarrow{\$} \mathsf{RGen}(\phi)$: RGen *is the probabilistic reduction key generation algorithm that takes as input an instance* ϕ *of* Λ*, and outputs a simulated public key simpk and a simulation secret key simsk.*
2. $(r_{\mathsf{RSign}}, r_{\mathsf{RExtract}}, r_{\mathsf{RHash}}) \xleftarrow{\$} \mathsf{RF}(x)$: RF *is a stateful probabilistic algorithm simulating a truly random function with domain* $\{0,1\}^*$ *and range* $\mathsf{Coins}_{\mathsf{RSign}} \times \mathsf{Coins}_{\mathsf{RExtract}} \times \mathsf{Coins}_{\mathsf{RHash}}$ *using a lazily sampled random table, where* $\mathsf{Coins}_{\mathsf{RSign}}$, $\mathsf{Coins}_{\mathsf{RExtract}}$, *and* $\mathsf{Coins}_{\mathsf{RHash}}$ *are sets for random coins of* RSign, RExtract *and* RHash*, respectively.*[4]

Remark 7. Intuitively, RF *has the following purpose. We will below define algorithms* RSign, RExtract, *and* RHash*, which are used by the reduction to simulate signatures, extract from a forgery, and possibly to simulate a random oracle (if in the random oracle model), respectively. We require these algorithms to be* stateless *and* deterministic*, since this will be necessary for our construction of a memory-tight reduction. At the same time, we do not want the algorithms* RSign, RExtract *and,* RHash *to be completely independent of each other. For example, the simulation of a signature by* RSign *may have to be consistent with the random oracle implemented by* RHash*. We*

[4] We note that algorithm RF is part of the canonical reduction. Another option would be providing the canonical reduction with an external random function oracle. We choose the former characterization because it naturally includes the memory consumption of the random table when considering the overall memory consumption of the canonical reduction.

ensure this consistency by giving all oracles access to the same *truly random function simulation algorithm* RF*. The algorithms of the canonical reduction are required to achieve consistency by only having access to* RF*. We will show below that this indeed holds for many standard security proofs for signature schemes.*

3. $\sigma := \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m)$*:* RSign *is the deterministic signature simulation algorithm with access to the algorithm* RF *that takes as input the simulation secret key simsk and a message m, and outputs a simulated signature* σ.[5]
4. $\rho := \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m^*, \sigma^*))$*:* $\mathsf{RExtract}$ *is the deterministic problem solution extraction algorithm with access to the algorithm* RF *that takes as input a forgery* (m^*, σ^*)*, and outputs an extracted solution* ρ.
5. $y := \mathsf{RHash}^{\mathsf{RF}(\cdot)}(simsk, x)$*:* RHash *is the deterministic hash simulation algorithm with access to the algorithm* RF *that takes as input an argument x, and outputs a simulated hash image y.*

We call an algorithm $\mathcal{R}$ *with black-box access to any adversary* $\mathcal{A}$*, write* $\mathcal{R}^{\mathcal{A}}$*, a* (ℓ, δ)-canonical reduction *from* $\mathsf{sEUF\text{-}CMA1}$ *to* Λ *if* $\mathcal{R}$ *satisfies the following properties.*

1. *The reduction* $\mathcal{R}$ *proceeds as follows:*
   (a) *When receiving a problem instance* ϕ*, the reduction* $\mathcal{R}$ *uses* $\mathsf{RGen}(\phi)$ *to simulate a public key simpk of* Sig *and generate the simulation secret key simsk, and starts* $\mathcal{A}$ *on input simpk.*
   (b) *Whenever the adversary* $\mathcal{A}$ *issues a signing query* $\mathsf{Sign}(m)$*, the reduction simulates the signature* σ *with* $\sigma := \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m)$ *and returns* σ *to* $\mathcal{A}$*. Note that* RSign *is deterministic, so even if* $\mathsf{Sign}(m)$ *is queried multiple times, the adversary always gets the same signature in return.*
   (c) *In case the random oracle model (ROM) is considered, the reduction also needs to be able to simulate the random oracle. To this end, the reduction* $\mathcal{R}$ *answers a random oracle query x by running* $y := \mathsf{RHash}^{\mathsf{RF}(\cdot)}(simsk, x)$ *and returns y.*
   (d) *When the adversary* $\mathcal{A}$ *outputs a candidate forgery* (m^*, σ^*)*, the reduction* $\mathcal{R}$ *first tests whether it is a valid forgery by checking*

$$\mathsf{Check}(m^*, \sigma^*) := \mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \wedge \sigma^* \neq \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*).$$

Intuitively, the second check is the main leverage to "recognize" new signatures. If the checks pass, then we know that (m^*, σ^*) *is valid and* not *the signature that* $\mathcal{R}$ *would have simulated. Then* $\mathcal{R}$ *uses* $\mathsf{RExtract}$ *to extract a solution* ρ *to the underlying problem* Λ *with*

$$\rho := \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m^*, \sigma^*)).$$

If the checks fail, $\mathcal{R}$ *runs* $\rho \xleftarrow{\$} \mathsf{U}(\phi)$*. Finally,* $\mathcal{R}$ *outputs* ρ *as the solution to the problem instance* ϕ*.*

[5] Note that the output signature σ is not necessarily a valid signature of Sig with respect to *simpk*.

2. *We require that $\mathcal{R}$ is a "valid" reduction from* sEUF-CMA1*-security to a non-interactive computational assumption Λ. That is, for any adversary $\mathcal{A}$, we have*
$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}^{\mathcal{A}}\right) \geq \frac{1}{\ell}\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}) - \delta.$$

Remark 8. If $\mathcal{R}$ is canonical, q_{S} is the upper bound of the number of Sign queries made by the adversary, q_{H} is the upper bound of the number of random oracle queries and q_{RF} is an upper bound of the number of evaluations of RF, then we obtain that

$$\begin{aligned}\mathbf{LocalTime}\left(\mathcal{R}^{\mathcal{A}}\right) \approx\ & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{Time}(\mathsf{RGen}) + q_{\mathsf{S}} \cdot \mathbf{Time}(\mathsf{RSign}) \\ & + q_{\mathsf{H}} \cdot \mathbf{Time}(\mathsf{RHash}) + \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ & + \max\{\mathbf{Time}(\mathsf{RExtract}), \mathbf{Time}(\mathsf{U})\} + q_{\mathsf{RF}} \cdot \mathbf{Time}(\mathsf{RF}),\end{aligned} \tag{1}$$

and that

$$\begin{aligned}\mathbf{LocalMem}\left(\mathcal{R}^{\mathcal{A}}\right) =\ & \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{RGen}) + \mathbf{Mem}(\mathsf{RSign}) \\ & + \mathbf{Mem}(\mathsf{RHash}) + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + \mathbf{Mem}(\mathsf{RExtract}) \\ & + \mathbf{Mem}(\mathsf{U}) + \mathbf{Mem}(\mathsf{RF}).\end{aligned} \tag{2}$$

Note that by design of the canonical reduction the only common state of the algorithms RGen, RSign, RHash and RExtract is the random table (whose size grows linearly with the number of different queries) in the random function simulation algorithm RF. Otherwise, these algorithms are stateless. This will be the main leverage to achieve memory-tightness, since the random function can be implemented memory-efficiently with a pseudorandom function.

We define an additional property for the canonical reduction when it deals with a non-interactive computational assumption Λ with trivial advantage $c_\Lambda > 0$.

Definition 9. *Let* Sig *be a signature scheme and $\mathcal{R}$ is a (ℓ, δ)-canonical reduction from the* sEUF-CMA1*-security of* Sig *to non-interactive computational assumption Λ with trivial advantage $c_\Lambda > 0$. We say $\mathcal{R}$ has* predictability probability ε_{p} *if for every adversary $\mathcal{A}$ when interacting with $\mathcal{R}$, the probability of $\mathcal{A}$ submitting a* $\mathsf{Forge}(m^*, \sigma^*)$ *query with*

$$\mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q} \wedge \sigma^* = \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$$

is upper bounded by ε_{p} and $\mathcal{Q}$ is the set of message-signature-pairs which are submitted and returned to $\mathcal{A}$ by oracle $\mathsf{Sign}(\cdot)$.

3.3 Multi-Challenge Security for Canonical Reductions

Next, we show how to transform any canonical reduction in the *single*-challenge setting to another reduction in the *multi*-challenge setting. Formally, consider the following theorem.

Theorem 10. *Let* Sig *be a digital signature scheme and let* Λ *be a non-interactive computational assumption with trivial advantage* c_Λ. *Suppose* $\mathcal{R}$ *is a* (ℓ, δ)*-canonical reduction from the* $\mathsf{sEUF\text{-}CMA1}$*-security of* Sig *to* Λ *(that has predictability probability* ε_p *if* $c_\Lambda > 0$*) and* $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \mathsf{Coins}_{\mathsf{RSign}} \times \mathsf{Coins}_{\mathsf{RExtract}} \times \mathsf{Coins}_{\mathsf{RHash}}$ *is a pseudorandom function. Using* $\mathcal{R}$ *and* PRF*, we can build another reduction* $\mathcal{R}'$ *from the* $\mathsf{msEUF\text{-}CMA1}$*-security of* Sig *to* Λ *such that for any* $\mathsf{msEUF\text{-}CMA1}$*-security adversary* $\mathcal{A}'$*, there exists an adversary* $\mathcal{B}$ *so that*

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}'^{\mathcal{A}'}\right) \geq \begin{cases} \dfrac{1}{\ell} \cdot \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}') - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right) - \delta & \text{if } c_\Lambda = 0 \\ \dfrac{1}{\ell} \cdot \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}') - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right) - \delta \ - q_\mathsf{F} \cdot \varepsilon_\mathsf{p} & \text{if } c_\Lambda > 0 \end{cases} \tag{3}$$

Furthermore,

$$\begin{aligned} \mathbf{LocalTime}(\mathcal{R}'^{\mathcal{A}'}) \approx{}& \mathbf{LocalTime}(\mathcal{A}') + \mathbf{Time}(\mathsf{RGen}) + (q_\mathsf{S} + q_\mathsf{F}) \cdot \mathbf{Time}(\mathsf{RSign}) \\ &+ q_\mathsf{H} \cdot \mathbf{Time}(\mathsf{RHash}) + q_\mathsf{F} \cdot \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ &+ \max\{\mathbf{Time}(\mathsf{RExtract}), \mathbf{Time}(\mathsf{U})\} + q_\mathsf{RF} \cdot \mathbf{Time}(\mathsf{PRF}), \\ \mathbf{LocalMem}(\mathcal{R}'^{\mathcal{A}'}) ={}& \mathbf{LocalMem}(\mathcal{A}') + \mathbf{Mem}(\mathsf{RGen}) + \mathbf{Mem}(\mathsf{RSign}) \\ &+ \mathbf{Mem}(\mathsf{RHash}) + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + \mathbf{Mem}(\mathsf{RExtract}) \\ &+ \mathbf{Mem}(\mathsf{U}) + \mathbf{Mem}(\mathsf{PRF}) + 1, \end{aligned} \tag{4}$$

and

$$\begin{aligned} \mathbf{LocalTime}(\mathcal{B}^{\mathcal{A}'}) \approx{}& \mathbf{LocalTime}(\mathcal{A}') + \mathbf{Time}(\mathsf{RGen}) + (q_\mathsf{S} + q_\mathsf{F}) \cdot \mathbf{Time}(\mathsf{RSign}) \\ &+ q_\mathsf{H} \cdot \mathbf{Time}(\mathsf{RHash}) + q_\mathsf{F} \cdot \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ &+ \max\{\mathbf{Time}(\mathsf{RExtract}), \mathbf{Time}(\mathsf{U})\} + \mathbf{Time}(\mathsf{InstGen}) \\ &+ \mathbf{Time}(\mathsf{V}), \\ \mathbf{LocalMem}(\mathcal{B}^{\mathcal{A}'}) ={}& \mathbf{LocalMem}(\mathcal{A}') + \mathbf{Mem}(\mathsf{RGen}) + \mathbf{Mem}(\mathsf{RSign}) \\ &+ \mathbf{Mem}(\mathsf{RHash}) + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + \mathbf{Mem}(\mathsf{RExtract}) \\ &+ \mathbf{Mem}(\mathsf{U}) + \mathbf{Mem}(\mathsf{InstGen}) + \mathbf{Mem}(\mathsf{V}). \end{aligned}$$

where q_F *is the number of* Forge *queries made by* $\mathcal{A}'$*,* q_S *is the number of* Sign *queries made by* $\mathcal{A}'$*,* q_H *is the numbers of queries made to the random oracle*[6]*, and* q_RF *is an upper bound of the number of evaluations of* RF.

Remark 11. For any $\mathsf{sEUF\text{-}CMA1}$ adversary $\mathcal{A}$ and any $\mathsf{msEUF\text{-}CMA1}$ adversary $\mathcal{A}'$, if we define the memory overhead of $\mathcal{R}'$ ($\mathcal{R}$) as

$$\begin{aligned} \Delta(\mathcal{R}') &:= \mathbf{LocalMem}(\mathcal{R}'^{\mathcal{A}'}) - \mathbf{LocalMem}(\mathcal{A}') \\ \Delta(\mathcal{R}) &:= \mathbf{LocalMem}(\mathcal{R}^{\mathcal{A}}) - \mathbf{LocalMem}(\mathcal{A}). \end{aligned}$$

[6] If the reduction is not in the ROM, then $q_\mathsf{H} = 0$ holds.

Then, from Equations (2) and (4), we have that,

$$\Delta(\mathcal{R}') - \Delta(\mathcal{R}) = \mathbf{Mem}(\mathsf{PRF}) + 1 - \mathbf{Mem}(\mathsf{RF}).$$

More intuitively speaking, this means that reduction $\mathcal{R}'$ does not use memory to keep a random function RF whose random table grows linearly with the number of different queries, but instead it uses some small amount of memory to store a PRF key and run the PRF. Furthermore, the algorithms in $\mathcal{R}'$ (RGen, RSign, RHash, $\mathsf{RExtract}$, $\mathsf{Sig.Vrfy}$, PRF and U) are stateless and their memory usage is independent of the number of queries made by adversary. Thus, the memory overhead of $\mathcal{R}'$, i.e., $\Delta(\mathcal{R}')$ will also be independent of the adversary, especially independent of q_{S}.

Remark 12. Equation (3) is equivalent to

$$\mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}') \leq \begin{cases} \ell \cdot \left(\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}'^{\mathcal{A}'}\right) + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right) + \delta\right) & \textit{if } c_\Lambda = 0 \\ \ell \cdot \left(\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}'^{\mathcal{A}'}\right) + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right) + \delta \ + \ q_{\mathsf{F}} \cdot \varepsilon_{\mathsf{p}}\right) & \textit{if } c_\Lambda > 0 \end{cases}$$

It shows that the $\mathsf{msEUF\text{-}CMA1}$ security of Sig builds upon both the security of NICA and the pseudorandomness of PRF. If ℓ is a constant, δ is a negligible value which is independent of the number of queries made by the adversary, ε_{p} is statistically small (when $c_\Lambda > 0$)and PRF is memory-tightly secure, then the $\mathsf{msEUF\text{-}CMA1}$ security of Sig is tight in both working factor and memory. (See Section 5 for more discussions about concrete applications.)

Proof (of Theorem 10). Since $\mathcal{R}$ is a canonical reduction, we know that there are algorithms $(\mathsf{RGen}, \mathsf{RSign}, \mathsf{RExtract}, \mathsf{RHash})$. Using these algorithms and a pseudorandom function, we construct another reduction $\mathcal{R}'$ which transfers any $\mathsf{msEUF\text{-}CMA1}$ adversary $\mathcal{A}'$ to a hard problem solver of Λ.

Construction of $\mathcal{R}'$. The reduction $\mathcal{R}'$ receives as input an instance ϕ of Λ and simulates the experiment $\mathsf{Exp}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}')$ for $\mathcal{A}'$. To this end, it first runs $(simpk, simsk) \xleftarrow{\$} \mathsf{RGen}(\phi)$ to obtain a simulated public key $simpk$ for the signature scheme Sig. Note that this is exactly the same as what $\mathcal{R}$ would do.

In contrast to $\mathcal{R}$, $\mathcal{R}'$ does not simulate a random function with algorithm RF. Instead, it chooses a uniform key $k \xleftarrow{\$} \{0,1\}^\lambda$ for a pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \mathsf{Coins}_{\mathsf{RSign}} \times \mathsf{Coins}_{\mathsf{RExtract}} \times \mathsf{Coins}_{\mathsf{RHash}}$ and uses PRF as a replacement.

$\mathcal{A}'$ then receives as input the simulated public key $simpk$ and gets access to the signing oracle Sign, the random oracle (if ROM is considered) and the "forgery attempt" oracle Forge. To simulate these oracles for $\mathcal{A}'$, the reduction $\mathcal{R}'$ does the following:

Sign-oracle. Upon receiving a signature query $\mathsf{Sign}(m)$ for some message $m \in M$, the reduction $\mathcal{R}'$ runs $\mathcal{R}$'s signature simulation algorithm with oracle access to PRF, i.e., $\sigma := \mathsf{RSign}^{\mathsf{PRF}(k,\cdot)}(simsk, m)$. Then it returns σ to $\mathcal{A}'$. Note that the same signature will be returned if the same message is queried multiple times since RSign is deterministic.

Random oracle. $\mathcal{R}'$ answers a random oracle query x by running RHash with oracle access to PRF, i.e., $y := \mathsf{RHash}^{\mathsf{PRF}(k,\cdot)}(simsk, x)$ and returns y.

Forge-oracle. Upon receiving a forgery attempt (m^*, σ^*), the reduction $\mathcal{R}'$ at first checks whether

$$\mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \quad \text{and} \quad \sigma^* \neq \mathsf{RSign}^{\mathsf{PRF}(k,\cdot)}(simsk, m^*)$$

In case both checks pass, the reduction $\mathcal{R}'$ attempts to extract a solution ρ for the problem instance ϕ from the forgery at hand by running $\rho := \mathsf{RExtract}^{\mathsf{PRF}(k,\cdot)}(simsk, (m^*, \sigma^*))$. Then $\mathcal{R}'$ returns ρ and halts.
In case any of the previous two checks failed, $\mathcal{R}'$ continues to simulate $\mathcal{A}'$. If the adversary $\mathcal{A}'$ fails to output any forgery attempt (m^*, σ^*) that can pass the checks throughout the whole simulation process, $\mathcal{R}'$ runs $\rho \xleftarrow{\$} \mathsf{U}(\phi)$ and outputs ρ.

Note that $\mathcal{R}'$ proceeds exactly as $\mathcal{R}$ but it uses a pseudorandom function instead of a truly random function and it needs to handle at most q_{F} forgery attempts as opposed to just one. Therefore, the running time of $\mathcal{R}'$ is the running time of $\mathcal{R}$ as given in Remark 8, replacing $\mathbf{Time}(\mathsf{RF})$ by $\mathbf{Time}(\mathsf{PRF})$ plus the time required to simulate the additional $q_{\mathsf{F}} - 1$ Forge-queries, namely $(q_{\mathsf{F}} - 1) \cdot (\mathbf{Time}(\mathsf{Vrfy}) + \mathbf{Time}(\mathsf{RSign}))$. This yields the time given in Theorem 10.

Similarly, the memory consumption of $\mathcal{R}'$ is the memory consumed by $\mathcal{R}$ as given in Remark 8, but instead of storing the random table in RF, $\mathcal{R}'$ needs to store the function description of PRF and its corresponding key, which again yields the values given in Theorem 10. In particular, note that the memory consumed by $\mathcal{R}'^{\mathcal{A}'}$ is independent of the number of queries made by $\mathcal{A}'$, as the stateful random table is replaced with the stateless keyed PRF PRF.

We complete the proof of Theorem 10 by analyzing the advantage of $\mathcal{R}'$ as follows.

The advantage of $\mathcal{R}'^{\mathcal{A}'}$. In order to analyse the advantage of $\mathcal{R}'^{\mathcal{A}'}$, we first modify the reduction $\mathcal{R}'$ to get a new reduction $\mathcal{R}_1$. More precisely, $\mathcal{R}_1$ is exactly $\mathcal{R}'$ except that it uses a random function RF instead of a pseudorandom function PRF.

We can easily build an adversary $\mathcal{B}$ and show that

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}'^{\mathcal{A}'}\right) \geq \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}_1^{\mathcal{A}'}\right) - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right). \tag{5}$$

The construction of $\mathcal{B}$ is straightforward. It generates the problem instance together with its witness using $(\phi, \omega) \xleftarrow{\$} \mathsf{InstGen}(1^\lambda)$. Then it simulates the above reductions and interacting with $\mathcal{A}'$ by forwarding all the input to RF/PRF to its own challenger. If the reduction outputs a solution ρ, $\mathcal{B}$ runs the algorithm V and outputs $\mathsf{V}(\phi, \omega, \rho)$. Thus, Equation (5) holds and the running time and memory consumption of $\mathcal{B}$ follows the equations in Theorem 10.

Next we modify $\mathcal{R}_1$ again to get $\mathcal{R}_2$. $\mathcal{R}_2$ is identical to $\mathcal{R}_1$ except that it logs all the chosen message queries with their respective signatures in the set $\mathcal{Q}$ and it replaces the check in Forge-oracle from

$$\mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \wedge \sigma^* \neq \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$$

to the check

$$\mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \wedge \sigma^* \neq \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*) \wedge (m^*, \sigma^*) \notin \mathcal{Q}.$$

Note that the added check $(m^*, \sigma^*) \notin \mathcal{Q}$ is redundant because every (m, σ) pair in $\mathcal{Q}$ has the property that $\sigma = \mathsf{RSign}^{\mathsf{PRF}(\cdot)}(simsk, m)$. Thus, we have that

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}_1^{\mathcal{A}'}\right) = \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}_2^{\mathcal{A}'}\right). \tag{6}$$

Next, we construct a single-challenge $\mathsf{sEUF\text{-}CMA1}$ adversary $\widetilde{\mathcal{A}}$ based on the multi-challenge adversary $\mathcal{A}'$. More precisely, after getting the public key pk, $\widetilde{\mathcal{A}}$ simulates $\mathcal{A}'$ and keep log of the set $\mathcal{Q}$ itself. Whenever $\mathcal{A}'$ submits a $\mathsf{Forge}(m^*, \sigma^*)$ query, $\widetilde{\mathcal{A}}$ checks whether $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q}$. If the check fails, $\widetilde{\mathcal{A}}$ continues the simulation of $\mathcal{A}'$. And $\widetilde{\mathcal{A}}$ outputs the first forgery that can pass this check as its own forgery attempt. Note that $\widetilde{\mathcal{A}}$ can perform these checks efficiently because it knows the public key pk and can log the set $\mathcal{Q}$ itself. Finally, $\widetilde{\mathcal{A}}$ terminates whenever $\mathcal{A}'$ terminates. Hence, $\widetilde{\mathcal{A}}$ is a well-defined single-challenge $\mathsf{sEUF\text{-}CMA1}$ adversary.

We can obtain an important observation on $\widetilde{\mathcal{A}}$: the game that is played between $\mathcal{R}_2$ and the multi-challenge adversary $\mathcal{A}'$ distributes almost identically with the game that is played between the canonical reduction $\mathcal{R}$ and the single-challenge adversary $\widetilde{\mathcal{A}}$. To see this, we only need to analyse how Forge queries are processed because the rest of the game stays the same for $\mathcal{R}_2^{\mathcal{A}'}$ and $\mathcal{R}^{\widetilde{\mathcal{A}}}$. Suppose $\mathcal{A}'$ submits a $\mathsf{Forge}(m^*, \sigma^*)$ query. We do a case distinction over the properties of the forgery.

- $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 0 \vee (m^*, \sigma^*) \in \mathcal{Q}$: In this case, the pair (m^*, σ^*) is not a forgery. Therefore, both $\tilde{\mathcal{A}}$ and $\mathcal{R}_2$ continue to simulate $\mathcal{A}'$.
- $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q} \wedge \sigma^* \neq \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$: Here, $\mathcal{A}'$ has output a valid forgery, and the signature is *not* the signature the reductions $\mathcal{R}$ or $\mathcal{R}_2$ would output in response to a query $\mathsf{Sign}(m^*)$. This means that this actually is a "useful" forgery for us. To that end, $\tilde{\mathcal{A}}$ and $\mathcal{R}_2$, respectively, do the following:
  - $\widetilde{\mathcal{A}}$ submits its final $\mathsf{Forge}(m^*, \sigma^*)$ query to $\mathcal{R}$ and $\mathcal{R}$ will use algorithm $\mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m^*, \sigma^*))$ to extract a solution ρ because (m^*, σ^*) will pass the check made by $\mathcal{R}$, i.e., $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 1$ and σ^* is not equal to $\mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$.
  - The above condition is exactly the condition checked by reduction $\mathcal{R}_2$ before it extracts. Thus, $\mathcal{R}_2$ will use algorithm $\mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m^*, \sigma^*))$ to extract a solution ρ.

- $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q} \wedge \sigma^* = \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$: Now, $\mathcal{A}'$ has output a valid forgery, but the signature σ^* would be the signature reduction $\mathcal{R}$ or $\mathcal{R}_2$ would output in response to query $\mathsf{Sign}(m^*)$. Here, game $\mathcal{R}_2^{\mathcal{A}'}$ differs from $\mathcal{R}^{\widetilde{\mathcal{A}}}$. Namely:
  - $\widetilde{\mathcal{A}}$ submits its final $\mathsf{Forge}(m^*, \sigma^*)$ query to $\mathcal{R}$ and $\mathcal{R}$ will use algorithm U to generate a trivial solution ρ. As there is only one forgery attempt, both $\mathcal{R}$ and $\widetilde{\mathcal{A}}$ will terminate after that.
  - $\mathcal{R}_2$ continues the simulation of $\mathcal{A}'$ as $\mathcal{A}'$ still can query the Forge oracle with a "useful" message-signature pair.

Based on this observation, we establish relations between $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}_2^{\mathcal{A}'})$ and $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}})$ through the following case distinction.

- If $c_\Lambda = 0$: the algorithm U will never output any valid solution ρ. In this case, we have
$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}_2^{\mathcal{A}'}) \geq \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}}). \tag{7}$$
The reason is that reduction $\widetilde{\mathcal{A}}$ will output the first $\mathsf{Forge}(m^*, \sigma^*)$ query of $\mathcal{A}'$ to $\mathcal{R}$ such that $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q}$. In this case, $\mathcal{R}$ only gains non-zero advantage if σ^* does not equal to $\mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$ and will gain no advantage if it does (because U will be run). However, $\mathcal{R}_2$ can do this check itself and can continue the simulation of $\mathcal{A}'$ *until* it finds a useful forgery which does not have to be the first valid forgery.
- If $c_\Lambda > 0$: we bound the difference between $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}_2^{\mathcal{A}'})$ and $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}})$ by proving the following Lemma. The proof idea is that $\mathcal{R}_2^{\mathcal{A}'}$ only differs from $\mathcal{R}^{\widetilde{\mathcal{A}}}$ when the adversary $\mathcal{A}'$ is able to submit (m^*, σ^*) such that σ^* equals $\mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$ and σ^* has never been returned by $\mathcal{R}$. We could bound this difference using the predictability probability of the canonical reduction $\mathcal{R}$.

Lemma 13. *If $c_\Lambda > 0$ and $\mathcal{R}$ has predictability probability ε_{p}, then we have*
$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}_2^{\mathcal{A}'}) \geq \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}}) - q_{\mathsf{F}} \cdot \varepsilon_{\mathsf{p}}. \tag{8}$$

We put the proof of Lemma 13 in Appendix A.

We note that the above case distinction benefit us because we only need the additional predictability probability property for $\mathcal{R}$ when $c_\Lambda > 0$. This makes our canonical reduction definition more general and leaves room for concrete instantiations.

Furthermore, we know that $\widetilde{\mathcal{A}}$ wins the (single-challenge) $\mathsf{sEUF\text{-}CMA1}$ game if and only if $\mathcal{A}'$ wins the (multi-challenge) $\mathsf{msEUF\text{-}CMA1}$ game because of the check $\mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q}$. So, we have that
$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\widetilde{\mathcal{A}}) = \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}').$$

Since $\mathcal{R}$ is canonical, we have that
$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}}) \geq \frac{1}{\ell}\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\widetilde{\mathcal{A}}) - \delta = \frac{1}{\ell}\mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}') - \delta. \tag{9}$$

Combining Equations (5) to (9), we have that

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}\left(\mathcal{R}'^{\mathcal{A}'}\right) \geq \begin{cases} \frac{1}{\ell} \cdot \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}') - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right) - \delta & \text{if } c_\Lambda = 0 \\ \frac{1}{\ell} \cdot \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}') - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}\left(\mathcal{B}^{\mathcal{A}'}\right) - \delta \ - q_{\mathsf{F}} \cdot \varepsilon_{\mathsf{p}} & \text{if } c_\Lambda > 0 \end{cases}$$

and the theorem follows. □

4 From msEUF-CMA1 Security to msEUF-CMA Security

So far we have shown how any signature scheme that can be proven sEUF-CMA1-secure (i.e., single-challenge and one-signature-per-message) via a canonical reduction to some computational problem, can be proven msEUF-CMA1-secure (i.e., *multi-challenge* and one-signature-per-message) in a memory-tight way. In this section, we extend our approach and present a generic transform, which "memory-tightly lifts" *any* signature scheme from msEUF-CMA1 security (i.e., multi-challenge and one-signature-per-message) to the desired msEUF-CMA security (i.e., multi-challenge and *many-signatures-per-message*).

Intuition. The core idea of this transform is to sign a message together with some randomly-chosen nonce n. Intuitively, this nonce "expands" the set of valid signatures for a given message. While this transform is straightforward, we see value to make it explicit.

Transform. Let $\lambda \in \mathbb{N}$ and let $\mathsf{Sig}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ be a signature scheme. We construct a new signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ as follows:

Key generation. Gen behaves exactly like Gen'.

Signing. Sign takes as input the secret key sk and a message m. It samples a nonce $n \xleftarrow{\$} \{0,1\}^\lambda$, computes $\sigma' \xleftarrow{\$} \mathsf{Sign}'(sk, m \parallel n)$, and returns $\sigma = (\sigma', n)$.

Verification. Vrfy takes as input a public key pk, a message m, and a signature $\sigma = (\sigma', n)$. It computes and returns $\mathsf{Sig}'.\mathsf{Vrfy}(pk, m \parallel n, \sigma')$.

Theorem 14. *From each adversary $\mathcal{A}$ breaking the msEUF-CMA-security of the above signature scheme Sig (with q_s signing queries), we can construct an adversary $\mathcal{B}$ such that $\mathsf{Adv}^{\mathsf{msEUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}'}(\mathcal{B}) + \frac{q_s^2}{2^\lambda}$ and*

$$\mathbf{LocalTime}(\mathcal{B}) \approx \mathbf{LocalTime}(\mathcal{A}) \quad \textit{and} \quad \mathbf{LocalMem}(\mathcal{B}) = \mathbf{LocalMem}(\mathcal{A}).$$

The proof of Theorem 14 is straightforward and we provide it in Appendix B
.

5 Applications

In this section, we present how the results of Sections 3 and 4 can be used to yield memory-tight strongly unforgeable signatures in the multi-challenge and many-signatures-per-message setting. In Section 5.1, we present a construction based on lossy identification schemes (similar to the construction by Abdalla et al. [2]) and prove its memory-tight security using our results. Then, in Section 5.2, we show how existing signature schemes such as RSA-FDH [11] benefit from our result and evade the existing impossibility results of [5,53]. In Appendix G, we show similar results for the Boneh, Lynn, and Shacham signature scheme [15,16].

We note that, a pseudorandom function is required when applying our results of Sections 3 and 4. In the standard model, we are aware of several pseudorandom functions that achieve almost tight security based on standard assumptions [42,47,49]. In the random oracle model, such a pseudorandom function exists unconditionally.

5.1 Memory-Tight Signatures from Lossy Identification Schemes

In this section, we present how to construct memory-tight strongly unforgeable signatures in the multi-challenge and many-signatures-per-message setting based on lossy identification schemes. To this end, we first present a formal definition of lossy identification schemes.

Lossy Identification Schemes. We adapt the definition of a *lossy identification scheme* [2,3].

Definition 15. *A* lossy identification scheme LID *is a tuple of algorithms*

$$\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$$

with the following properties.

- $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$ *is the normal key generation algorithm. It takes as input the security parameter and outputs a public verification key* pk *and a secret key* sk.
- $pk \xleftarrow{\$} \mathsf{LID.LossyGen}(1^\lambda)$ *is a lossy key generation algorithm that takes the security parameter and outputs a lossy verification key* pk.
- $\mathsf{LID.Prove}$ *is the prover algorithm that is split into two algorithms:*
  - $(\mathsf{cmt}, \mathsf{st}) \xleftarrow{\$} \mathsf{LID.Prove}_1(sk)$ *is a probabilistic algorithm that takes as input the secret key and returns a commitment* cmt *and a state* st.
  - $\mathsf{resp} \xleftarrow{\$} \mathsf{LID.Prove}_2(sk, \mathsf{cmt}, \mathsf{ch}, \mathsf{st})$ *is a deterministic algorithm*[7] *that takes as input the secret key, a commitment* cmt*, a challenge* ch*, a state* st*, and returns a response* resp.

[7] As far as we know, all the instantiations of lossy identification schemes have a deterministic $\mathsf{LID.Prove}_2$ algorithm. However, if a new instantiation requires randomness, then it can be "forwarded" from $\mathsf{LID.Prove}_1$ in the state variable st. Therefore the requirement that $\mathsf{LID.Prove}_2$ is deterministic is without loss of generality, and only made to simplify our security analysis.

– $\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp}) \in \{0,1\}$ *is a deterministic verification algorithm that takes a public key, and a conversation transcript (i.e., a commitment, a challenge, and a response) as input and outputs a bit, where* 1 *indicates that the proof is "accepted" and* 0 *"rejected".*

We assume that a public key pk implicitly defines two sets, the challenge set CSet *and the response set* RSet.

Definition 16. *Let* $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$ *defined as above. We call* LID lossy *when the following properties hold:*

– Completeness of normal keys. *Let* $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$ *be a key pair and let* $(\mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$ *be an honest transcript (i.e.,* $(\mathsf{cmt}, \mathsf{st}) \xleftarrow{\$} \mathsf{LID.Prove}_1(sk)$*,* $\mathsf{ch} \xleftarrow{\$} \mathsf{CSet}$*, and* $\mathsf{resp} \xleftarrow{\$} \mathsf{LID.Prove}_2(sk, \mathsf{cmt}, \mathsf{ch}, \mathsf{st})$*). We call* LID ρ-complete, *if*

$$\Pr[\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp}) = 1] \geq \rho(\lambda),$$

where ρ is a non-negligible function in λ. We call LID perfectly-complete, *if it is 1-complete.*

– Simulatability of transcripts. *Let* $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$ *be a key pair. We call* LID ε_s-simulatable *if* $\mathsf{LID.Sim}$ *taking public key pk, a challenge* $\mathsf{ch} \in \mathsf{CSet}$ *and a response* $\mathsf{resp} \in \mathsf{RSet}$ *as input, deterministically generates a commitment* cmt *such that* $(\mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$ *is a valid transcript (i.e.,* $\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp}) = 1$*). Furthermore, if* $(\mathsf{ch}, \mathsf{resp})$ *is chosen uniformly random from* $\mathsf{CSet} \times \mathsf{RSet}$*, the distribution of the transcript* $(\mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$ *is statistically indistinguishable (up to an upper bound ε_s) from honestly generated transcripts. If $\varepsilon_s = 0$, we call* LID perfectly simulatable.

– Indistinguishability of keys. *We define the advantage of an adversary $\mathcal{A}$ to break the key-indistinguishability of* LID *as*

$$\mathsf{Adv}_{\mathsf{LID}}^{\mathsf{IND\text{-}KEY}}(\mathcal{A}) := \left|\Pr\left[\mathcal{A}(pk) = 1\right] - \Pr\left[\mathcal{A}(pk') = 1\right]\right|,$$

where $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$ *and* $pk' \xleftarrow{\$} \mathsf{LID.LossyGen}(1^\lambda)$*, is negligible in λ.*

– Lossiness. *Consider the following security experiment* $\mathsf{Exp}_{\mathsf{LID}}^{\mathsf{IMPERSONATE}}(\mathcal{A})$ *described below, played between a challenger and an adversary $\mathcal{A}$:*

1. *The challenger generates a lossy verification key* $pk \xleftarrow{\$} \mathsf{LID.LossyGen}(1^\lambda)$ *and sends it to the adversary $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ may now compute a commitment* cmt *and send it to the challenger. The challenger responds with a random challenge* $\mathsf{ch} \xleftarrow{\$} \mathsf{CSet}$.
3. *Eventually, the adversary $\mathcal{A}$ outputs a response* resp*. The challenger outputs* $\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$.

We call LID *ε_ℓ-lossy if no computationally unrestricted adversary $\mathcal{A}$ wins the above security game with probability*

$$\Pr[\mathsf{Exp}_{\mathsf{LID}}^{\mathsf{IMPERSONATE}}(\mathcal{A}) = 1] \geq \varepsilon_\ell.$$

Definition 17. *A lossy identification scheme*

$$\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$$

is commitment-recoverable *if* $\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$ *first recomputes* $\mathsf{cmt}' = \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$ *and then outputs 1 if and only if* $\mathsf{cmt}' = \mathsf{cmt}$.

This new property captures the predictability probability of algorithm $\mathsf{LID.Sim}$.

Definition 18. *Let* LID *be a lossy ID scheme, pk be any normal public key generated by* $\mathsf{LID.Gen}(1^\lambda)$ *or any lossy public key generated by* $\mathsf{LID.LossyGen}(1^\lambda)$. *The* min-entropy with respect to $\mathsf{LID.Sim}$ *is defined as*

$$\alpha := -\log_2 \left(\max_{pk,\mathsf{cmt}} \Pr[\mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp}) = \mathsf{cmt}] \right)$$

where the probability is taken over $\mathsf{ch} \xleftarrow{\$} \mathsf{CSet}$ *and* $\mathsf{resp} \xleftarrow{\$} \mathsf{RSet}$.

In Appendix C, we explore the connections between this new definition and the definition of the min-entropy for lossy identification scheme in [2,3].

Remark 19. We are aware of five different lossy identification scheme instantiations and they are based on DDH [46], DSDL, Ring-LWE, Subset Sum [2,3] and RSA [37]. As far as we know, all of them are commitment-recoverable. And the schemes based on DDH, DSDL and RSA assumption are perfectly complete and perfectly simulatable.

Memory-Tight Signatures from Lossy Identification Schemes. In the following, we present the construction of the signature scheme based on lossy identification scheme. This construction is slightly different from the construction by Abdalla et al. in [2,3] and can be seen as a variant of the Fiat-Shamir transform [26]. We show that this construction can be proven strongly unforgeable in the single challenge and one-message-per-signature setting (in the sense of $\mathsf{sEUF\text{-}CMA1}$, see Definition 3) in Theorem 20. This result is not yet memory-tight, but work-factor-tight, as the reduction still needs to do book-keeping for a random function, but does not need to store the set of queried messages and there respective signatures in the set $\mathcal{Q}$ anymore. Based this result, we show how to apply Theorems 10 and 14 to yield strong unforgeability in the multi-challenge and many-signatures-per-message setting (in the sense of $\mathsf{msEUF\text{-}CMA}$), which then will be fully tight, i.e., both work-factor- and memory-tight.

Let $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy})$ be a lossy identification scheme and let $\mathsf{H} : \{0,1\}^* \to \mathsf{CSet}$. Consider the following digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Key generation. Algorithm Gen samples a key pair $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$.

Signing. The signing algorithm Sign takes as input sk and a message $m \in \{0,1\}^*$. Then, it computes $(\mathsf{cmt}, \mathsf{st}) \xleftarrow{\$} \mathsf{LID.Prove}_1(sk)$, $\mathsf{ch} := \mathsf{H}(m, \mathsf{cmt})$ and $\mathsf{resp} := \mathsf{LID.Prove}_2(sk, \mathsf{ch}, \mathsf{cmt}, \mathsf{st})$, and outputs the signature $\sigma := (\mathsf{ch}, \mathsf{resp})$.

Verification. The verification algorithm Vrfy takes as input a public key pk, message $m \in \{0,1\}^*$, and a signature $\sigma = (\mathsf{ch}, \mathsf{resp})$. It runs the check $\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$. More precisely, it first recovers

$$\mathsf{cmt} := \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$$

and then computes $\mathsf{ch}' := \mathsf{H}(m, \mathsf{cmt})$ Finally, the reduction outputs 1 if and only if ch equals ch'.

Compared to the signature scheme by Abdalla et al. [2, 3], signature of the above scheme is a pair $(\mathsf{ch}, \mathsf{resp})$ whereas signature in [2, 3] is a pair $(\mathsf{cmt}, \mathsf{resp})$ for a transcript $(\mathsf{cmt}, \mathsf{ch}, \mathsf{resp})$ of the lossy identification scheme. For a concrete instantiation based on DDH assumption, this yields a shorter signature.

Theorem 20. *Let* $\mathsf{H}\colon \{0,1\}^* \to \mathsf{CSet}$ *be modeled as a random oracle and let* LID *be a lossy identification scheme that is commitment-recoverable, perfectly complete,* ε_s*-simulatable,* ε_ℓ*-lossy.*

Then, from each adversary $\mathcal{A}$ *breaking the* $\mathsf{sEUF\text{-}CMA1}$ *security of the above signature scheme, we can construct an adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}KEY}}_{\mathsf{LID}}(\mathcal{B}) + \frac{1}{|\mathsf{CSet}|} + \frac{1}{|\mathsf{RSet}|} + q_{\mathsf{S}} \cdot \varepsilon_s + q_{\mathsf{H}} \cdot \varepsilon_\ell$$

and

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}) \leq{}& \mathbf{LocalTime}(\mathcal{A}) + \mathbf{Time}(\mathsf{LID.LossyGen}) \\ &+ (q_s + q_{\mathsf{H}} + 1) \cdot \mathbf{Time}(\mathsf{RF}) + \mathbf{Time}(\mathsf{Sig.Vrfy}), \\ \mathbf{LocalMem}(\mathcal{B}) ={}& \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{LID.LossyGen}) + \mathbf{Mem}(\mathsf{RF}) \\ &+ \mathbf{Mem}(\mathsf{Sig.Vrfy}),\end{aligned}$$

where q_{S} *is the number of* Sign*-queries issued by* $\mathcal{A}$*,* q_{F} *is the number of* Forge*-queries issued by* $\mathcal{A}$ *and* q_{H} *is the number of hash queries throughout the game.*

The proof of Theorem 20 is similar to the proof by Abdalla et al. in [2, 3]. One technical difference it that, in our proof, we need to memory-tightly switch the winning condition in the $\mathsf{sEUF\text{-}CMA1}$ game into the checks that a canonical reduction would do. For completeness, we provide the full proof in Appendix D
.

Applying Theorem 10. Here, we show how to apply Theorem 10 to lift the security of the LID-based signature scheme to work-factor-tight *and* memory-tight security in the *multi*-challenge and one-per-message setting. To apply the theorem, we show that the adversary $\mathcal{B}$ in Theorem 20 can be "translated" into a canonical reduction $\mathcal{R}_{\mathsf{LID}}$ which satisfies Definition 6.

To this end, we define the canonical reduction $\mathcal{R}_{\mathsf{LID}}$ from $\mathsf{sEUF\text{-}CMA1}$-security to the indistinguishability of keys $\mathsf{IND\text{-}KEY}$ to be the tuple $(\mathsf{RGen}, \mathsf{RF}, \mathsf{RSign}, \mathsf{RExtract}, \mathsf{RHash})$ as follows.

RGen**:** On input $\phi = pk$, RGen return $(pk, \emptyset)$ where $\emptyset$ denotes the empty word in this context.

RF**:** On input any string $x \in \{0,1\}^*$, RF simulates a random function using a lazily sampled random table. In the following, we will omit this table and view RF as a random function. Further, for $(r_{\mathsf{RSign}}, r_{\mathsf{RHash}}) := \mathsf{RF}(x)$, we define the short-hands $r_{\mathsf{RSign}} =: \mathsf{RF}(\texttt{"sim"} \,\|\, x)$ and $r_{\mathsf{RHash}} =: \mathsf{RF}(\texttt{"hash"} \,\|\, x)$.

$\mathsf{RSign}^{\mathsf{RF}(\cdot)}$**:** On input $simsk = \emptyset$ and m, RSign outputs $\sigma = (\mathsf{ch}, \mathsf{resp})$ with $(\mathsf{ch}, \mathsf{resp}) := \mathsf{RF}(\texttt{"sim"} \,\|\, m)$.

$\mathsf{RExtract}^{\mathsf{RF}(\cdot)}$**:** On input $simsk = \emptyset$ and (m^*, σ^*), $\mathsf{RExtract}$ outputs solution $\rho = 1$. Note that by definition $\mathcal{R}_{\mathsf{LID}}$ runs $\mathsf{RExtract}$ only if $\mathsf{Vrfy}(pk, m^*, \sigma^*) = 1$ and $\sigma^* = (\mathsf{ch}^*, \mathsf{resp}^*) \neq \mathsf{RSign}(simsk, m^*) = \mathsf{RF}(\texttt{"sim"} \,\|\, m^*)$, which is exactly the condition introduced in Game 3 of the proof(c.f., Appendix D). Hence, if $\mathsf{RExtract}$ is run the queried forgery is valid.

$\mathsf{RHash}^{\mathsf{RF}(\cdot)}$**:** On input $simsk = \emptyset$ and x, RHash works as follows:

- If x cannot be parsed as $x = m \,\|\, \mathsf{cmt}$, then it returns $\mathsf{RF}(\texttt{"hash"} \,\|\, x)$.
- Otherwise, it parses $m \,\|\, \mathsf{cmt} := x$ and runs $(\mathsf{ch}, \mathsf{resp}) := \mathsf{RF}(\texttt{"sim"} \,\|\, m)$ and then $\mathsf{cmt}' := \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$.
  - If $\mathsf{cmt} = \mathsf{cmt}'$, then it returns ch.
  - Otherwise, it returns $\mathsf{RF}(\texttt{"hash"} \,\|\, x)$.

According to the results of Theorem 20, we have

$$\mathsf{Adv}^{\mathsf{IND\text{-}KEY}}_{\mathsf{LID}}(\mathcal{R}^{\mathcal{A}}_{\mathsf{LID}}) \geq \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}) - \frac{1}{|\mathsf{CSet}|} - \frac{1}{|\mathsf{RSet}|} - q_{\mathsf{S}} \cdot \varepsilon_s - q_{\mathsf{H}} \cdot \varepsilon_\ell$$

where all quantities are defined as in Theorem 20 and $\mathsf{Adv}^{\mathsf{IND\text{-}KEY}}_{\mathsf{LID}}(\mathcal{R}^{\mathcal{A}}_{\mathsf{LID}}) = \mathsf{Adv}^{\mathsf{IND\text{-}KEY}}_{\mathsf{LID}}(\mathcal{B})$. Thus, $\mathcal{R}_{\mathsf{LID}}$ fulfills Definition 6, Property 2 with $\ell = 1$ and $\delta = \frac{1}{|\mathsf{CSet}|} + \frac{1}{|\mathsf{RSet}|} + q_{\mathsf{S}} \cdot \varepsilon_s + q_{\mathsf{H}} \cdot \varepsilon_\ell$.

Note that the indistinguishability of keys assumption $\mathsf{IND\text{-}KEY}$ has trivial advantage $c_{\mathsf{IND\text{-}KEY}} = \frac{1}{2} > 0$. We need to consider the predictability probability of $\mathcal{R}_{\mathsf{LID}}$ and we do it by proving the following theorem.

Theorem 21. *Let* $\mathsf{H}\colon \{0,1\}^* \to \mathsf{CSet}$ *be modeled as a random oracle and let* LID *be a lossy identification scheme that is* ε_s*-simulatable and has min-entropy* α *with respect to algorithm* $\mathsf{LID.Sim}$*. Then reduction* $\mathcal{R}_{\mathsf{LID}}$ *has predictability probability*

$$\varepsilon_{\mathsf{p}} \leq \frac{1}{|\mathsf{CSet}| \times |\mathsf{RSet}|} + q_{\mathsf{H}} \cdot \left(\frac{1}{2^\alpha} + \varepsilon_s\right)$$

where q_{H} *is the number of* $\mathsf{H}(\cdot)$ *queries.*

The proof idea of the above theorem is that the signatures $\mathcal{R}_{\mathsf{LID}}$ simulated are statistically close to real (challenge, response) pairs of LID and the min-entropy of LID makes it difficult to guess. We put the full proof of Theorem 21 in Appendix E.

Applying Theorem 14. It remains to lift the security of the LID-based signature scheme from the one-signature-per-message setting to the many-signatures-per-message-setting. This can easily be done, by applying the transform presented in Section 4. As the reduction presented in Theorem 14 preserves the memory-tightness of the one-per-message scheme Sig', we have that the transformed LID-based signature scheme is memory-tightly strongly unforgeable in the multi-challenge and many-signatures-per-message setting.

5.2 On the Memory-Tightness of RSA-FDH

Auerbach et al. [5] show that RSA-FDH can be proven memory-tightly unforgeable in the single-challenge and many-signatures-per-message setting under the RSA assumption. However, due to the existing tightness lower bounds, they did not achieve work-factor-tightness. In this subsection, we first show that RSA-FDH can be proven memory-tightly unforgeable in the *multi-challenge* setting because the reduction by Auerbach et al. satisfies our definition of a canonical reduction. Furthermore, we additionally show that with one extra random bit in the signature, we are able to achieve both memory and working factor tightness together with strong security.

We briefly recall the RSA assumption in the form of a non-interactive computational assumption.

Definition 22. *Let* GenRSA *be an algorithm that takes as input the security parameter* 1^λ *and returns* $(N = pq, e, d)$*, where* p *and* q *are distinct primes of bit length* $\lambda/2$ *and* e, d *are integers such that* $ed = 1 \bmod \phi(N)$*. The RSA assumption with respect to* GenRSA *is a non-interactive computational assumption* $\Lambda_{\mathsf{RSA}} = (\mathsf{InstGen}_{\mathsf{RSA}}, \mathsf{V}_{\mathsf{RSA}}, \mathsf{U}_{\mathsf{RSA}})$ *where*

1. $\mathsf{InstGen}_{\mathsf{RSA}}(1^\lambda)$ *runs* $(N, e, d) \xleftarrow{\$} \mathsf{GenRSA}(1^\lambda)$*, selects* $x \xleftarrow{\$} \mathbb{Z}_N$*, computes* $y = x^e \bmod N$ *and outputs a problem instance* $\phi = (N, e, y)$ *and a witness* $\omega = x$*.*
2. $\mathsf{V}_{\mathsf{RSA}}(\phi, \omega, \rho)$ *returns* 1 *if and only if* $\rho = \omega$*.*
3. $\mathsf{U}_{\mathsf{RSA}}(\phi)$ *returns a failure symbol* $\perp$*.*

Recall the RSA-FDH signature scheme [11] $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ as follows.

- Gen runs $(N, e, d) \xleftarrow{\$} \mathsf{GenRSA}(1^\lambda)$ and returns $pk = (N, e), sk = (N, d)$.
- $\mathsf{Sign}(sk, m)$ returns $\sigma = H(m)^d \bmod N$ where $H : \{0, 1\}^* \to \mathbb{Z}_n$ is a hash function.
- $\mathsf{Vrfy}(pk, m, \sigma)$ returns 1 if and only if $\sigma^e = H(m) \bmod N$.

The scheme provides existential unforgeability under chosen message attacks, which can be reduced to the RSA assumption in the random oracle model as shown by [12, 20]. However, these proofs are neither work-factor-tight (an inherent loss linear in the number of signature queries) nor memory-tight (implementing the random oracle). Auerbach et al. [5] were able to improve those results by proving RSA-FDH memory-tight in the single-challenge setting, based on the

RSA assumption in the random oracle model. We show how to further improve this result with our techniques.

We proceed as in Section 5.1. That is, we first argue that RSA-FDH is strongly unforgeable under an chosen message attack in the single-challenge and one-signature-per-message setting ($\mathsf{sEUF\text{-}CMA1}$-secure) under the RSA assumption in the random oracle model. From this result, we then construct the canonical reduction to show multi-challenge security. The transform presented in Section 4 then finally gives us many-signatures-per-message security again.

We will omit a full proof of $\mathsf{sEUF\text{-}CMA1}$ security of RSA-FDH but only provide a brief sketch. The proof is very similar to the proof of $\mathsf{EUF\text{-}CMA}$ security presented by Auerbach et al. [5]. Note that RSA-FDH scheme is a unique signature scheme. That is, for every message m there is exactly one valid signature, namely $\sigma = \mathsf{H}(m)^d \mod N$. Thus, whenever $\mathsf{Sign}(m)$ is queried it will always return the same signature σ and the adversary will always see exactly one signature per message. Moreover, given a valid message-signature pair (m^*, σ^*), there exists no second valid signature $\sigma \neq \sigma^*$. Hence,

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\text{RSA-FDH}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\text{RSA-FDH}}(\mathcal{A}). \tag{10}$$

As we need a memory-tight reduction for RSA-FDH up to a truly random function RF, we adapt the result [5, Thm. 5] by Auerbach et al. slightly. Namely, we do not implement the random sampling with a PRF as they are doing, but by a truly random function RF that is maintained with an explicit look-up table. By standard arguments, it is easy to verify that with this adaptation it follows from [5, Thm. 5] and Equation (10) that

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\text{RSA-FDH}}(\mathcal{A}) \leq \exp(1) \cdot q_{\mathsf{S}} \cdot \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{RSA}},\lambda}(\mathcal{B}) \tag{11}$$

where q_{S} denotes the number of signature queried by $\mathcal{A}$ and where $\mathcal{B}$ is identical to $\mathcal{B}_2$ in the proof of [5, Thm. 5] except that $\mathcal{B}$ uses a random function RF with a explicitly stored look-up table instead of a PRF. We have

$$\begin{aligned}
&\mathbf{LocalTime}(\mathcal{B}) \approx \mathbf{LocalTime}(\mathcal{A}) + (q_{\mathsf{H}} + q_{\mathsf{S}}) \cdot \mathbf{Time}(\mathsf{RF}),\\
&\mathbf{LocalMem}(\mathcal{B}) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{RF}) + 3
\end{aligned}$$

where q_{H} is the number of random oracle queries and q_{S} the number of signature queries made by $\mathcal{A}$.

We define the canonical reduction $\mathcal{R}_{\mathsf{RSA}}$ from $\mathsf{sEUF\text{-}CMA1}$-security to the RSA assumption as tuple $(\mathsf{RGen}, \mathsf{RSign}, \mathsf{RExtract}, \mathsf{RHash})$ as follows. In essence, $\mathcal{R}_{\mathsf{RSA}}$ works exactly as $\mathcal{B}$. Let $\mathsf{RF}\colon \{0,1\}^* \to \{0,1\} \times \mathbb{Z}_N$ with $\mathsf{Coins}_{\mathsf{RSign}} = \mathsf{Coins}_{\mathsf{RExtract}} = \emptyset$ and $\{0,1\} \times \mathbb{Z}_N = \mathsf{Coins}_{\mathsf{RHash}}$. Further, for $(b, r) := \mathsf{RF}(x)$, we define the short-hands $b =: \mathsf{RF}_1(x)$ and $r =: \mathsf{RF}_2(x)$. We view RF_1 as an $(1/q_{\mathsf{S}})$-biased random function similar to the biased coin used by Coron [20], i.e., $\Pr[\mathsf{RF}_1(x) = 1] = 1/q_{\mathsf{S}}$, where q_{S} is the number of signature queries issued by the adversary.

RGen**:** Given an RSA instance $\phi = (N, e, y)$, RGen returns $(\mathit{simpk}, \mathit{simsk}) = ((N, e), (N, e, y))$.

$\mathsf{RHash}^{\mathsf{RF}(\cdot)}$**:** Given $simsk = (N, e, y)$ and x, RHash returns $\mathsf{RF}_2(x)^e \cdot y$ if $\mathsf{RF}_1(x) = 1$. Otherwise, it returns $\mathsf{RF}_2(x)^e$.

$\mathsf{RSign}^{\mathsf{RF}(\cdot)}$**:** Given $simsk = (N, e, y)$ and m, RSign outputs a signature $\sigma = \mathsf{RF}_2(m)$ if $\mathsf{RF}_1(m) = 0$. Otherwise, the reduction aborts and terminates by outputting the failure symbol $\perp$.

$\mathsf{RExtract}^{\mathsf{RF}(\cdot)}$**:** Given $simsk = (N, e, y)$ and (m^*, σ^*), $\mathsf{RExtract}$ outputs a solution $\rho = \sigma^* / \mathsf{RF}_2(m)$. Note that by definition $\mathcal{R}_{\mathsf{RSA}}$ runs $\mathsf{RExtract}$ only if $\mathsf{Vrfy}(simpk, m^*, \sigma^*) = 1$ and $\sigma^* \neq \mathsf{RSign}(simsk, m^*)$. The validity of the signature implies that $(\sigma^*)^e = \mathsf{RHash}(simsk, m^*)$ and since we have $\sigma^* \neq \mathsf{RSign}(simsk, m^*)$, we also know that $\mathsf{RF}_1(m^*) = 1$.

Reduction $\mathcal{R}_{\mathsf{RSA}}$ works basically as $\mathcal{B}$, we have due to Equation (11)

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{RSA}},\lambda}(\mathcal{R}^{\mathcal{A}}_{\mathsf{RSA}}) \geq \frac{1}{\exp(1) \cdot q_{\mathsf{S}}} \cdot \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\text{RSA-FDH}}(\mathcal{A}).$$

That is, $\mathcal{R}_{\mathsf{RSA}}$ is a $(\ell, 0)$-canonical reduction for RSA-FDH with value $\ell = 1/(\exp(1) \cdot q_{\mathsf{S}})$. It runs in time

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{R}^{\mathcal{A}}_{\mathsf{RSA}}) \approx\ & \mathbf{LocalTime}(\mathcal{A}) + \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ & + (q_{\mathsf{H}} + q_{\mathsf{S}} + 1) \cdot \mathbf{Time}(\mathsf{RF}),\end{aligned}$$

and requires memory

$$\mathbf{LocalMem}(\mathcal{R}^{\mathcal{A}}_{\mathsf{RSA}}) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{RF}) + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + 3.$$

Note that the RSA assumption has trivial advantage $c_{\mathsf{RSA}} = 0$, so there is no need to consider the predictability probability of $\mathcal{R}_{\mathsf{RSA}}$.

Now, we can use Theorem 10 to lift the security of RSA-FDH to the multi-challenge in a memory-tight way. To this end, we can construct a reduction $\mathcal{R}'_{\mathsf{RSA}}$ from $\mathsf{msEUF\text{-}CMA1}$-security of RSA-FDH to the RSA assumption as presented in the proof Theorem 10. This implies that we can construct an adversary $\mathcal{B}'$ such that

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{RSA}},\lambda}((\mathcal{R}'_{\mathsf{RSA}})^{\mathcal{A}'}) \geq \frac{\mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\text{RSA-FDH}}(\mathcal{A}')}{\exp(1) \cdot q_{\mathsf{S}}} - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}(\mathcal{B}')$$

where $\mathsf{PRF} \colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\} \times \mathbb{Z}_N$ is a keyed function. Moreover, it holds that

$$\begin{aligned}\mathbf{LocalTime}((\mathcal{R}'_{\mathsf{RSA}})^{\mathcal{A}'}) \approx\ & \mathbf{LocalTime}(\mathcal{A}') + \mathbf{Time}(\mathsf{RGen}) \\ & + (q_{\mathsf{S}} + q_{\mathsf{F}} + q_{\mathsf{H}}) \cdot \mathbf{Time}(\mathsf{PRF}) + q_{\mathsf{F}} \cdot \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ \mathbf{LocalMem}((\mathcal{R}'_{\mathsf{RSA}})^{\mathcal{A}'}) =\ & \mathbf{LocalMem}(\mathcal{A}') + 4 + \mathbf{Mem}(\mathsf{Sig.Vrfy}) \\ & + \mathbf{Mem}(\mathsf{PRF}).\end{aligned}$$

Thus, the reduction $\mathcal{R}'_{\mathsf{RSA}}$ is a memory-tight, but not work-factor-tight, reduction from $\mathsf{msEUF\text{-}CMA1}$-security to the RSA assumption.

Note that since RSA-FDH is a unique signature scheme, the one-signature-per-message security automatically implies the many-signatures-per-message security. Thus, we do not need to apply our theorem form Section 4. At first glance, this result seems to contradict the memory lower bound for unique signatures established by Wang *et al.* [53, Theorem 3]. However, this is not the case as our reduction does not meet the criteria for their impossibility result to hold.[8] So we evade their lower bound and achieve memory tightness for RSA-FDH.

On the Overall Tightness of RSA-FDH. In the previous section, we have shown how RSA-FDH can be proven memory-tight in the multi-challenge and many-signatures-per-message setting. As already explained above, due to existing tightness lower bounds, plain RSA-FDH cannot be proven work-fact-tight. However, when considering a slight variant of RSA-FDH, which was proposed by Katz and Wang [46], we can apply our techniques to prove this variant fully tight. In essence, we still consider RSA-FDH, but choose a uniformly random bit b and sign $b \parallel m$ instead of only m. We call this scheme RSA-FDH+ and we can prove the following theorem.

Theorem 23. *For any adversary $\mathcal{A}'$, there exists a reduction $\mathcal{R}'_{\mathsf{RSA+}}$ and adversary $\mathcal{B}'$ such that*

$$\mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\text{RSA-FDH+}}(\mathcal{A}') \leq 2\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{RSA}},\lambda}((\mathcal{R}'_{\mathsf{RSA+}})^{\mathcal{A}'}) + 2\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}(\mathcal{B}').$$

where $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\} \times \mathbb{Z}_N \times \mathbb{Z}_N$ *is a keyed PRF. Moreover, it holds that*

$$\begin{aligned}\mathbf{LocalTime}((\mathcal{R}'_{\mathsf{RSA+}})^{\mathcal{A}'}) &\approx \mathbf{LocalTime}(\mathcal{A}') + \mathbf{Time}(\mathsf{RGen}) \\ &\quad + (q_{\mathsf{S}} + q_{\mathsf{F}} + q_{\mathsf{H}}) \cdot \mathbf{Time}(\mathsf{PRF}) + q_{\mathsf{F}} \cdot \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ \mathbf{LocalMem}((\mathcal{R}'_{\mathsf{RSA+}})^{\mathcal{A}'}) &= \mathbf{LocalMem}(\mathcal{A}') + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + \mathbf{Mem}(\mathsf{PRF}) + 4.\end{aligned}$$

Hence, $\mathcal{R}'_{\mathsf{RSA+}}$ is a fully tight reduction (i.e., work-factor-tight and memory-tight), from msEUF-CMA1-security of RSA-FDH+ to the RSA assumption. Applying the transform of Section 4 and adding an additional nonce that is signed along with the message, we can further lift this result to achieve msEUF-CMA-security under the RSA assumption.

The proof of Theorem 23 follows the Katz-Wang approach. We provide the formal description of scheme RSA-FDH+ and the proof of Theorem 23 in Appendix F .

Acknowledgments. We would also like to thank the anonymous reviewers of

[8] More precisely, Wang *et al.* [53] define two parameters c_r and c_g, where c_r captures the work-factor loss of the reduction and c_g captures the trivial winning probability of the assumption. They require $c_g < 1/2$ and $c_r + c_g > 1/2$ for their lower bound to hold. However, we have $c_g = 0$ for the RSA assumption and $c_r = 1/(\exp(1) \cdot q_{\mathsf{S}})$ for our reduction, implying $c_r + c_g < 1/2$, which does not fall into the realm of Theorem 3 in [53].

ASIACRYPT 2021 for helpful comments and Ashrujit Ghoshal, Riddhi Ghosal, Joseph Jaeger, Stefano Tessaro for spotting a gap in the proof of Theorem 1 in a preliminary version of this paper. We closed this gap by introducing the predictability probability of canonical reduction and with a refined analysis using case distinction.

References

1. Abdalla, M., Ben Hamouda, F., Pointcheval, D.: Tighter reductions for forward-secure signature schemes. In: PKC 2013. LNCS, vol. 7778, pp. 292–311. Springer, Heidelberg (Feb / Mar 2013)
2. Abdalla, M., Fouque, P.A., Lyubashevsky, V., Tibouchi, M.: Tightly-secure signatures from lossy identification schemes. In: EUROCRYPT 2012. LNCS, vol. 7237, pp. 572–590. Springer, Heidelberg (Apr 2012)
3. Abdalla, M., Fouque, P.A., Lyubashevsky, V., Tibouchi, M.: Tightly secure signatures from lossy identification schemes. Journal of Cryptology **29**(3), 597–631 (Jul 2016)
4. Abe, M., Groth, J., Ohkubo, M.: Separating short structure-preserving signatures from non-interactive assumptions. In: ASIACRYPT 2011. LNCS, vol. 7073, pp. 628–646. Springer, Heidelberg (Dec 2011)
5. Auerbach, B., Cash, D., Fersch, M., Kiltz, E.: Memory-tight reductions. In: CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 101–132. Springer, Heidelberg (Aug 2017)
6. Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: TCC 2015, Part I. LNCS, vol. 9014, pp. 629–658. Springer, Heidelberg (Mar 2015)
7. Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 273–304. Springer, Heidelberg (May 2016)
8. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg (May 2000)
9. Bellare, M., Ristenpart, T.: Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In: EUROCRYPT 2009. LNCS, vol. 5479, pp. 407–424. Springer, Heidelberg (Apr 2009)
10. Bellare, M., Ristenpart, T.: Simulation without the artificial abort: Simplified proof and improved concrete security for waters' IBE scheme. Cryptology ePrint Archive, Report 2009/084 (2009), `http://eprint.iacr.org/2009/084`
11. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM CCS 93. pp. 62–73. ACM Press (Nov 1993)
12. Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: EUROCRYPT'96. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (May 1996)
13. Bhattacharyya, R.: Memory-tight reductions for practical key encapsulation mechanisms. In: PKC 2020, Part I. LNCS, vol. 12110, pp. 249–278. Springer, Heidelberg (May 2020)
14. Blazy, O., Kiltz, E., Pan, J.: (Hierarchical) identity-based encryption from affine message authentication. In: CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 408–425. Springer, Heidelberg (Aug 2014)

15. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001)
16. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology **17**(4), 297–319 (Sep 2004)
17. Chaum, D., Evertse, J.H., van de Graaf, J.: An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In: EUROCRYPT'87. LNCS, vol. 304, pp. 127–141. Springer, Heidelberg (Apr 1988)
18. Chen, J., Wee, H.: Fully, (almost) tightly secure IBE and dual system groups. In: CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 435–460. Springer, Heidelberg (Aug 2013)
19. Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness. In: CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 767–797. Springer, Heidelberg (Aug 2019)
20. Coron, J.S.: On the exact security of full domain hash. In: CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (Aug 2000)
21. Coron, J.S.: Optimal security proofs for PSS and other signature schemes. In: EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg (Apr / May 2002)
22. Davis, H., Günther, F.: Tighter proofs for the sigma and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029 (2020), `https://eprint.iacr.org/2020/1029`
23. Diemert, D., Gellert, K., Jager, T., Lyu, L.: More efficient digital signatures with tight multi-user security. In: Public-Key Cryptography – PKC 2021. pp. 1–31. Springer International Publishing, Cham (2021)
24. Diemert, D., Jager, T.: On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Cryptology ePrint Archive, Report 2020/726; to appear in the Journal of Cryptology (2020), `https://eprint.iacr.org/2020/726`
25. Fersch, M., Kiltz, E., Poettering, B.: On the one-per-message unforgeability of (EC)DSA and its variants. In: TCC 2017, Part II. LNCS, vol. 10678, pp. 519–534. Springer, Heidelberg (Nov 2017)
26. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987)
27. Fleischhacker, N., Jager, T., Schröder, D.: On tight security proofs for Schnorr signatures. In: ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 512–531. Springer, Heidelberg (Dec 2014)
28. Fleischhacker, N., Jager, T., Schröder, D.: On tight security proofs for Schnorr signatures. Journal of Cryptology **32**(2), 566–599 (Apr 2019)
29. Garg, S., Bhaskar, R., Lokam, S.V.: Improved bounds on security reductions for discrete log based signatures. In: CRYPTO 2008. LNCS, vol. 5157, pp. 93–107. Springer, Heidelberg (Aug 2008)
30. Gay, R., Hofheinz, D., Kiltz, E., Wee, H.: Tightly CCA-secure encryption without pairings. In: EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 1–27. Springer, Heidelberg (May 2016)
31. Ghoshal, A., Ghosal, R., Jaeger, J., Tessaro, S.: Hiding in plain sight: Memory-tight proofs via randomness programming. Cryptology ePrint Archive, Report 2021/1409 (2021), `https://ia.cr/2021/1409`
32. Ghoshal, A., Jaeger, J., Tessaro, S.: The memory-tightness of authenticated encryption. In: Advances in Cryptology - CRYPTO 2020 - 40th Annual International

Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12170, pp. 127–156. Springer (2020)

33. Ghoshal, A., Tessaro, S.: On the memory-tightness of hashed ElGamal. In: EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 33–62. Springer, Heidelberg (May 2020)
34. Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 95–125. Springer, Heidelberg (Aug 2018)
35. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (Apr 1988)
36. Gueron, S., Lindell, Y.: Better bounds for block cipher modes of operation via nonce-based key derivation. In: ACM CCS 2017. pp. 1019–1036. ACM Press (Oct / Nov 2017)
37. Hasegawa, S., Isobe, S.: Lossy identification schemes from decisional RSA. In: International Symposium on Information Theory and its Applications, ISITA 2014, Melbourne, Australia, October 26-29, 2014. pp. 143–147. IEEE (2014)
38. Hoang, V.T., Tessaro, S.: The multi-user security of double encryption. In: EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 381–411. Springer, Heidelberg (Apr / May 2017)
39. Hofheinz, D., Jager, T.: Tightly secure signatures and public-key encryption. In: CRYPTO 2012. LNCS, vol. 7417, pp. 590–607. Springer, Heidelberg (Aug 2012)
40. Hofheinz, D., Jager, T., Knapp, E.: Waters signatures with optimal security reduction. In: PKC 2012. LNCS, vol. 7293, pp. 66–83. Springer, Heidelberg (May 2012)
41. Jager, T., Kakvi, S.A., May, A.: On the security of the PKCS#1 v1.5 signature scheme. In: ACM CCS 2018. pp. 1195–1208. ACM Press (Oct 2018)
42. Jager, T., Kurek, R., Pan, J.: Simple and more efficient PRFs with tight security from LWE and matrix-DDH. In: ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 490–518. Springer, Heidelberg (Dec 2018)
43. Jager, T., Stam, M., Stanley-Oakes, R., Warinschi, B.: Multi-key authenticated encryption with corruptions: Reductions are lossy. In: TCC 2017, Part I. LNCS, vol. 10677, pp. 409–441. Springer, Heidelberg (Nov 2017)
44. Kakvi, S.A., Kiltz, E.: Optimal security proofs for full domain hash, revisited. In: EUROCRYPT 2012. LNCS, vol. 7237, pp. 537–553. Springer, Heidelberg (Apr 2012)
45. Kakvi, S.A., Kiltz, E.: Optimal security proofs for full domain hash, revisited. Journal of Cryptology **31**(1), 276–306 (Jan 2018)
46. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: ACM CCS 2003. pp. 155–164. ACM Press (Oct 2003)
47. Lewko, A.B., Waters, B.: Efficient pseudorandom functions from the decisional linear assumption and weaker variants. In: ACM CCS 2009. pp. 112–120. ACM Press (Nov 2009)
48. Liu, X., Liu, S., Gu, D., Weng, J.: Two-pass authenticated key exchange with explicit authentication and tight security. In: ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 785–814. Springer, Heidelberg (Dec 2020)
49. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: 38th FOCS. pp. 458–467. IEEE Computer Society Press (Oct 1997)

50. Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (Dec 2005)
51. Schäge, S.: Tight proofs for signature schemes without random oracles. In: EUROCRYPT 2011. LNCS, vol. 6632, pp. 189–206. Springer, Heidelberg (May 2011)
52. Seurin, Y.: On the exact security of Schnorr-type signatures in the random oracle model. In: EUROCRYPT 2012. LNCS, vol. 7237, pp. 554–571. Springer, Heidelberg (Apr 2012)
53. Wang, Y., Matsuda, T., Hanaoka, G., Tanaka, K.: Memory lower bounds of reductions revisited. In: EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 61–90. Springer, Heidelberg (Apr / May 2018)

A Proof of Lemma 13

Proof. In this proof, we focus on the multi-challenge adversary $\mathcal{A}'$ and we call a $\mathsf{Forge}(m^*, \sigma^*)$ query made by $\mathcal{A}'$ a *valid* query if $\mathsf{Sig.Vrfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q}$. Furthermore, we call a valid $\mathsf{Forge}(m^*, \sigma^*)$ query an *equal* query if $\sigma^* = \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$ and we call it a *unequal* query otherwise. Then, we denote E_i as the event that the first $i-1$ valid queries made by $\mathcal{A}'$ are equal queries and the i-th valid query made by $\mathcal{A}'$ is a unequal query. We further denote F as the event that all the valid queries made by $\mathcal{A}'$ are equal queries.

Then, let's first analyse $\mathcal{R}^{\widetilde{\mathcal{A}}}$. The adversary $\widetilde{\mathcal{A}}$ simulates $\mathcal{A}'$ and will forward the first valid $\mathsf{Forge}(m_1^*, \sigma_1^*)$ query made by $\mathcal{A}'$ to $\mathcal{R}$.

– If E_1 happens, $\mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m_1^*, \sigma_1^*))$ is run to extract a solution ρ.
– If E_1 does not happen, U is run to extract a solution.

So we have that $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}})$

$$= \left|\Pr[\mathsf{V}(\phi, \omega, \rho) = 1 \text{ for } \rho \xleftarrow{\$} \mathcal{R}^{\widetilde{\mathcal{A}}}(\phi)] - c_\Lambda\right|$$

$$= \left|\begin{array}{c}\Pr[\mathsf{E}_1]\Pr[\mathsf{V}(\phi, \omega, \rho) = 1 \text{ for } \rho \xleftarrow{\$} \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m_1^*, \sigma_1^*)) \mid \mathsf{E}_1] \\ + \Pr[\neg\mathsf{E}_1]\underbrace{\Pr[\mathsf{V}(\phi, \omega, \rho) = 1 \text{ for } \rho \xleftarrow{\$} \mathsf{U}(\phi) \mid \neg\mathsf{E}_1]}_{=c_\Lambda} - c_\Lambda\end{array}\right|$$

$$= \left|\Pr[\mathsf{E}_1]\Big(\Pr[\mathsf{V}(\phi, \omega, \rho) = 1 \text{ for } \rho \xleftarrow{\$} \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m^*, \sigma^*)) \mid \mathsf{E}_1] - c_\Lambda\Big)\right|$$

Then we analyse $\mathcal{R}_2^{\mathcal{A}'}$. $\mathcal{R}_2$ continues the simulation of $\mathcal{A}'$ until it submits a valid unequal forge query.

– If E_i happens for some i, $\mathsf{RExtract}^{\mathsf{RF}(\cdot)}(simsk, (m_i^*, \sigma_i^*))$ is run to extract a solution ρ.
– If E_i does not happen for any i, then F happens and U is run to extract a solution.

Then, we have that $\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}_2^{\mathcal{A}'})$

$$= \left|\Pr[\mathsf{V}(\phi, \omega, \rho) = 1 \text{ for } \rho \xleftarrow{\$} \mathcal{R}_2^{\mathcal{A}'}(\phi)] - c_\Lambda\right|$$

$$= \left| \sum_{i=1}^{q_{\mathsf{F}}} \Pr[\mathsf{E}_i] \Pr[\mathsf{V}(\phi,\omega,\rho)=1 \text{ for } \rho \xleftarrow{\$} \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(\mathit{simsk},(m_i^*,\sigma_i^*)) \mid \mathsf{E}_i] + \Pr[\mathsf{F}] \underbrace{\Pr[\mathsf{V}(\phi,\omega,\rho)=1 \text{ for } \rho \xleftarrow{\$} \mathsf{U}(\phi) \mid \mathsf{F}]}_{=c_\Lambda} - c_\Lambda \right|$$

$$= \left| \sum_{i=1}^{q_{\mathsf{F}}} \Pr[\mathsf{E}_i] \Big(\Pr[\mathsf{V}(\phi,\omega,\rho)=1 \text{ for } \rho \xleftarrow{\$} \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(\mathit{simsk},(m_i^*,\sigma_i^*)) \mid \mathsf{E}_i] - c_\Lambda \Big) \right|$$

$$\geq \underbrace{\left| \Pr[\mathsf{E}_1] \Big(\Pr[\mathsf{V}(\phi,\omega,\rho)=1 \text{ for } \rho \xleftarrow{\$} \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(\mathit{simsk},(m_1^*,\sigma_1^*)) \mid \mathsf{E}_1] - c_\Lambda \Big) \right|}_{=\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}})}$$

$$- \left| \sum_{i=2}^{q_{\mathsf{F}}} \Pr[\mathsf{E}_i] \Big(\Pr[\mathsf{V}(\phi,\omega,\rho)=1 \text{ for } \rho \xleftarrow{\$} \mathsf{RExtract}^{\mathsf{RF}(\cdot)}(\mathit{simsk},(m_i^*,\sigma_i^*)) \mid \mathsf{E}_i] - c_\Lambda \Big) \right|$$

$$\geq \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}}) - \sum_{i=2}^{q_{\mathsf{F}}} \Pr[\mathsf{E}_i]$$

We have that $\Pr[\mathsf{E}_i] \leq \varepsilon_{\mathsf{p}}$ for any i. The reason is that event E_i implies that $\sigma_i^* = \mathsf{RSign}^{\mathsf{RF}(\cdot)}(\mathit{simsk}, m_i^*)$ (because $\mathsf{Forge}(m_i^*,\sigma_i^*)$ is an equal query) and m_i^* has never been queried to $\mathsf{Sign}(\cdot)$ oracle (because $(m_i^*,\sigma_i^*) \notin \mathcal{Q}$). Thus, we have

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}_2^{\mathcal{A}'}) \geq \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda,\lambda}(\mathcal{R}^{\widetilde{\mathcal{A}}}) - q_{\mathsf{F}} \cdot \varepsilon_{\mathsf{p}}$$

and Lemma 13 follows. □

B Proof of Theorem 14

Proof. We start this proof with the observation that the only difference between msEUF-CMA1 security and msEUF-CMA security is how their signing oracles work. To be more precise, the msEUF-CMA1 signing oracle only provides *one* signature per message (which will be returned repeatedly if the same message is queried multiple times), whereas the msEUF-CMA signing oracle always returns *fresh* signatures, even if the oracle is queried multiple times for the same message.

Construction of $\mathcal{B}$. Upon initialization of the experiment, $\mathcal{B}$ receives a public key pk of Sig' from its msEUF-CMA1 challenger. Additionally, it gets access to a signing oracle Sign' and a forging oracle Forge'. Note that the public key pk is also a public key of Sig, since both signature schemes use the same key generation algorithm. Hence, $\mathcal{B}$ immediately forwards the public key pk to $\mathcal{A}$.

Whenever $\mathcal{A}$ issues a signing query $\mathsf{Sign}(m)$, the adversary $\mathcal{B}$ chooses a fresh nonce $n \xleftarrow{\$} \{0,1\}^\lambda$ and queries $\mathsf{Sign}'(m \parallel n)$ to obtain a signature σ', which is forwarded to $\mathcal{A}$ as $\sigma = (\sigma', n)$. Note that $\mathcal{B}$ does not store the nonce n.

Whenever $\mathcal{A}$ queries a $\mathsf{Forge}(m,\sigma)$, the adversary $\mathcal{B}$ directly relays this query to its forging oracle Forge'. If $\mathcal{A}$ halts, $\mathcal{B}$ will also halt.

Success probability of $\mathcal{B}$. We now analyze the success probability of $\mathcal{B}$. Let us (for now) assume that for any signing query $\mathsf{Sign}(m)$ by $\mathcal{A}$ for a some message m, the nonces n chosen by $\mathcal{B}$ never repeat. In this case, adversary $\mathcal{B}$ never uses the same signing query $\mathsf{Sign}'(m \parallel n)$ twice, directly implying that all signatures $\sigma = (\sigma', n)$ for m will look correctly distributed to $\mathcal{A}$. Since the winning conditions of $\mathsf{msEUF\text{-}CMA}$ and $\mathsf{msEUF\text{-}CMA1}$ are equal, any winning forgery by $\mathcal{A}$ will also be a winning forgery against the $\mathsf{msEUF\text{-}CMA1}$ challenger.

It remains to analyze what happens if the same nonce is chosen twice for some signing query $\mathsf{Sign}(m)$ by $\mathcal{A}$. In this case, $\mathcal{B}$ issues the same signing query $\mathsf{Sign}'(m \parallel n)$ *twice*, and receives (due to the one-signature-per-message nature of the challenger) the same signature σ' twice. The adversary $\mathcal{A}$ would then receive (σ', n) as reply to its signing query. Note that this results in a incorrect distribution of signatures for $\mathcal{A}$ as (σ', n) never differs in the first position.

Formally, let coll be the event that $\mathcal{A}$ receives signatures $\sigma_1 = (\sigma, n)$ and $\sigma_2 = (\sigma', n)$ in response to some (not necessarily distinct) signature queries m and m'. Furthermore, let "win is set for $\mathcal{A}$" and "win is set for $\mathcal{B}$" denote the events that win is set in the $\mathsf{Exp}^{\mathsf{msEUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A})$ and $\mathsf{Exp}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}'}(\mathcal{B})$ security experiments (i.e., an adversary submits a "winning" forgery with respect to its challenger). We have

$$\begin{aligned}\Pr[\mathsf{win} \text{ is set for } \mathcal{A}] &= \Pr[\mathsf{win} \text{ is set for } \mathcal{A} \land \mathsf{coll}] + \Pr[\mathsf{win} \text{ is set for } \mathcal{A} \land \neg\mathsf{coll}] \\ &\leq \Pr[\mathsf{coll}] + \Pr[\mathsf{win} \text{ is set for } \mathcal{A} \mid \neg\mathsf{coll}] \\ &\leq \frac{q_{\mathsf{S}}^2}{2^\lambda} + \Pr[\mathsf{win} \text{ is set for } \mathcal{B}].\end{aligned}$$

Equivalently,

$$\mathsf{Adv}^{\mathsf{msEUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\mathsf{Sig}'}(\mathcal{B}) + \frac{q_{\mathsf{S}}^2}{2^\lambda}.$$

Running time and memory of $\mathcal{B}$. Note that $\mathcal{B}$ only relays the queries of $\mathcal{A}$, except for choosing the nonce during signing queries by $\mathcal{A}$, which adds a mere constant overhead per query in terms of running time. Since $\mathcal{B}$ neither stores the public key nor any of the chosen nonces, it does not consume any additional memory apart from $\mathcal{A}$. Hence, we have

$$\mathbf{LocalTime}(\mathcal{B}) \approx \mathbf{LocalTime}(\mathcal{A}) \quad \text{and} \quad \mathbf{LocalMem}(\mathcal{B}) = \mathbf{LocalMem}(\mathcal{A}).$$

C Min-entropy of $\mathsf{LID.Sim}$

In this section, we explore the connections between Definition 18 and the definition of the min-entropy for lossy identification scheme in [2,3].

First, we observe that, Definition 18 covers the cases of both normal public key and lossy public key. If we do a case distinction between them, we can prove the following lemma.

Lemma 24. *Let* LID *be a lossy identification scheme which has simulator* $\mathsf{LID.Sim}$ *with min-entropy* α*. Let* pk_n *be any normal public key generated by* $\mathsf{LID.Gen}(1^\lambda)$ *and let* pk_l *be any lossy public key generated by* $\mathsf{LID.LossyGen}(1^\lambda)$*. If we define*

$$\alpha_n := -\log_2\left(\max_{pk_n,\mathsf{cmt}} \Pr[\mathsf{LID.Sim}(pk_n,\mathsf{ch},\mathsf{resp}) = \mathsf{cmt}]\right)$$
$$\alpha_l := -\log_2\left(\max_{pk_l,\mathsf{cmt}} \Pr[\mathsf{LID.Sim}(pk_l,\mathsf{ch},\mathsf{resp}) = \mathsf{cmt}]\right)$$

where the probability is taken over $\mathsf{ch} \xleftarrow{\$} \mathsf{CSet}, \mathsf{resp} \xleftarrow{\$} \mathsf{RSet}$*. Then, we have*

$$\alpha = \min\{\alpha_l, \alpha_n\}.$$

Lemma 24 follows from Definition 18.

Then, we first connect α_n with the min-entropy of lossy identification scheme defined in [2,3].

Definition 25 ([2,3]). *Let* LID *be a lossy ID scheme, sk be any secret key generated by* $\mathsf{LID.Gen}(1^\lambda)$ *and* $\mathcal{C}(sk)$ *be the set of all possible commitments outputted by* $\mathsf{LID.Prove}_1(sk)$*. The* min-entropy with respect to LID *is defined as*

$$\gamma := -\log_2\left(\max_{sk,\mathsf{cmt}\in\mathcal{C}(sk)} \Pr[\mathsf{LID.Prove}_1(sk) = \mathsf{cmt}]\right)$$

where the probability is taken over only the random choices made by algorithm $\mathsf{LID.Prove}_1$.

We can prove the following lemma.

Lemma 26. *Let* LID *be a lossy ID scheme with* γ*-bits min-entropy (c.f. Definition 25) and is* ε_s*-simulatable (c.f. Definition 16). Suppose* $\mathsf{LID.Sim}$ *has* α_n*-bits min-entropy with respect to normal keys (as defined in Lemma 24), then we have that*

$$|2^{-\alpha_n} - 2^{-\gamma}| \le \varepsilon_s. \qquad (12)$$

Proof. The "real" commitment outputted by $\mathsf{LID.Prove}_1(sk)$ distributes differently from the simulated commitment outputted by $\mathsf{LID.Sim}(pk_n,\mathsf{ch},\mathsf{resp})$ for uniform ch and resp. However, since LID is ε_s-simulatable, the statistical distance between the two distribution is no larger than ε_s. Thus, we have Equation (12) holds. □

Lemma 26 shows that if γ is large and ε_s is statistically small, then α_n will be large. This holds for all the lossy identification schemes we consider in Remark 19 where $\gamma = \Omega(\lambda)$ and $\varepsilon_s = O(2^{-\lambda})$ (or $\varepsilon_s = 0$).

Next we consider the case of lossy public keys and show α_l is large for many lossy identification schemes we are aware of. More precisely, we show concrete DDH-based LID scheme in Appendix C.1, RSA-based LID scheme in Appendix C.2, ϕ-hiding-based LID scheme in Appendix C.3, and Ring-LWE-based LID scheme in Appendix C.4.

C.1 Lossy Identification Scheme Based on DDH

In this subsection, we recall the DDH-based LID scheme as shown in [17,23] and show that α_l is large for this scheme.

Let $(\mathbb{G}, g, q)$ be a cyclic group with a λ-bit prime order q and generator g and let $h \in \mathbb{G}$. Consider the lossy identification scheme $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$ as follows:

Key generation. The algorithm $\mathsf{LID.Gen}$ chooses a value $x \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random. It sets $pk := (g, h, u, v) = (g, h, g^x, h^x)$ and $sk := x$, and outputs (pk, sk).

Lossy key generation. The algorithm $\mathsf{LID.LossyGen}$ chooses two group elements $u, v \xleftarrow{\$} \mathbb{G}$ uniformly and independently at random. It outputs $pk := (g, h, u, v)$.

Proving. The algorithm $\mathsf{LID.Prove}$ is split up into the following two algorithms:

1. The algorithm $\mathsf{LID.Prove}_1$ takes as input a secret key $sk = x$, chooses a random value $r \xleftarrow{\$} \mathbb{Z}_q$, and computes a commitment $\mathsf{cmt} := (e, f) = (g^r, h^r)$, where g, h are the value of the pk corresponding to sk. It outputs $(\mathsf{cmt}, \mathsf{st})$ with $\mathsf{st} := r$.
2. The algorithm $\mathsf{LID.Prove}_2$ takes as input a secret key $sk = x$, a commitment $\mathsf{cmt} = (e, f)$, a challenge $\mathsf{ch} \in \mathbb{Z}_q$, a state $\mathsf{st} = r$, and outputs a response $\mathsf{resp} := r - \mathsf{ch} \cdot x$.

Verification. The verification algorithm $\mathsf{LID.Vrfy}$ takes as input a public key $pk = (g, h, u, v)$, a commitment $\mathsf{cmt} = (e, f)$, a challenge $\mathsf{ch} \in \mathbb{Z}_q$, and a response $\mathsf{resp} \in \mathbb{Z}_q$. It outputs 1 if and only if $e = g^{\mathsf{resp}} \cdot u^{\mathsf{ch}}$ and $f = h^{\mathsf{resp}} \cdot v^{\mathsf{ch}}$.

Simulation. The simulation algorithm $\mathsf{LID.Sim}$ takes as input a public key $pk = (g, h, u, v)$, a challenge $\mathsf{ch} \in \mathbb{Z}_q$, and a response $\mathsf{resp} \in \mathbb{Z}_q$. It outputs a commitment $\mathsf{cmt} = (e, f) = (g^{\mathsf{resp}} \cdot u^{\mathsf{ch}}, h^{\mathsf{resp}} \cdot v^{\mathsf{ch}})$.

We can prove the following lemma.

Lemma 27. *The DDH-based LID scheme* LID *has* $\alpha_n \geq \log_2 q$ *and* $\alpha_l \geq \log_2 q$.

Proof. $\mathsf{LID.Sim}$ outputs $(e, f) = (g^{\mathsf{resp}} \cdot u^{\mathsf{ch}}, h^{\mathsf{resp}} \cdot v^{\mathsf{ch}})$. As long as g is a generator and resp is chosen uniformly at random, e will be a uniform element in group $\mathbb{G}$. Thus, regardless of the input pk is lossy or not, $\mathsf{LID.Sim}$ has min-entropy at least $\log_2 q = \Theta(\lambda)$. □

C.2 Lossy Identification Scheme Based on RSA

In this subsection, we recall the RSA-based LID scheme as shown in [37] and show that α_l is large for this scheme.

Consider the LID scheme $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$ as follows:

Key generation. The $\mathsf{LID.Gen}$ algorithm generates a RSA modulus $N = PQ$ where $P = 2pp' + 1$ and $Q = 2qq' + 1$ for different primes p, p', q, q' of length

λ. There is a unique factorization of the quadratic residue group modulo N for $\mathbb{QR}_N = \mathbb{G}_1 \times \mathbb{G}_2$ where $|\mathbb{G}_1| = p'q'$ and $|\mathbb{G}_2| = pq$. Select random generator g in $\mathbb{G}_1$ and $a \xleftarrow{\$} \mathbb{Z}_{\phi(N)}$. Compute $y = g^a \bmod N$ and output $pk := (N, g, y), sk := (p, p', q, q', g, a)$

Lossy key generation. The lossy public key is generated exactly the same as the normal public key, except that $\mathsf{LID.LossyGen}$ selects a random generator g in $\mathbb{G}_1$ and a random generator f in $\mathbb{G}_2$. Then select $a \xleftarrow{\$} \mathbb{Z}_{\phi(N)}, b \xleftarrow{\$} \mathbb{Z}_{pq}$ and let $y := g^a f^b \bmod N$.

Proving. The algorithm $\mathsf{LID.Prove}$ is split up into the following two algorithms:

1. The algorithm $\mathsf{LID.Prove}_1$ takes as input a secret key sk, chooses a random value $r \xleftarrow{\$} \mathbb{Z}_{\phi(N)}$, and computes a commitment $\mathsf{cmt} = A := g^r$. Then it outputs cmt together with the state $\mathsf{st} := r$.
2. The algorithm $\mathsf{LID.Prove}_2$ takes as input a secret key sk, a commitment $\mathsf{cmt} = A$, a challenge $\mathsf{ch} = w \in \mathbb{Z}_N$, a state $\mathsf{st} = r$, and outputs a response $\mathsf{resp} = s := aw + r \bmod \phi(N)$.

Verification. $\mathsf{LID.Vrfy}$ outputs 1 iff $Ay^w = g^s \bmod N$ holds.

Simulation. $\mathsf{LID.Sim}$ gets the public key pk, a challenge $\mathsf{ch} = w \in \mathbb{Z}_N$ and a response $\mathsf{resp} = s \in \mathbb{Z}_N$, outputs a commitment $\mathsf{cmt} = A := g^s y^{-w} \bmod N$.

We can prove the following lemma.

Lemma 28. *The ϕ-hiding-based LID scheme* LID *has $\alpha_l \geq 2\lambda$ and $\alpha_n \geq 2\lambda$.*

Proof. Let us first fix the challenge w and consider the distribution of A for uniform s. In this case, since g is a generator of $\mathbb{G}_1$, A distributes uniformly over the set $\mathbb{G}_1/y^w$. Since $|\mathbb{G}_1| = p'q'$ and p', q' are primes of length λ, we have $|\mathbb{G}_1| \geq 2^{2\lambda}$. Then, regardless of the public key is lossy or not, $\mathsf{LID.Sim}$ has min-entropy at least $\log_2(2^{2\lambda}) = 2\lambda$. □

C.3 Lossy Identification Scheme Based on ϕ-hiding

In this subsection, we recall the ϕ-hiding-based LID scheme as shown in [23] and show that α_l is large for this scheme.

Consider the LID scheme $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$ as follows:

Key generation. The algorithm $\mathsf{LID.Gen}$ samples (N, e) where $N = p \cdot q$ is the product of two distinct primes p, q of length $\lambda/2$ and e is random prime of length $\ell_e \leq \lambda/4$ such that e divides $p - 1$. Then it samples $S \xleftarrow{\$} \mathbb{Z}_N^*$ and computes $U = S^e$. It sets $pk = (N, e, U)$ and $sk = (N, e, S)$, where (N, e) are from the common parameters.

Lossy key generation. The lossy key generation algorithm $\mathsf{LID.LossyGen}$ samples U uniformly at random from the e-th non-residues modulo N.[9]

[9] This is indeed efficiently possible as $U \xleftarrow{\$} \mathbb{Z}_N^*$ is a not an e-th residue with probability $1 - 1/e$ and we can efficiently check whether a given U is an e-th residue when the factorization of N is known [1].

Proving. The algorithm $\mathsf{LID.Prove}$ is split up into the following two algorithms:

1. The algorithm $\mathsf{LID.Prove}_1$ takes as input a secret key $sk = (N, e, S)$, chooses a random value $r \xleftarrow{\$} \mathbb{Z}_N^*$, and computes a commitment $\mathsf{cmt} := r^e \bmod N$. It outputs $(\mathsf{cmt}, \mathsf{st})$ with $\mathsf{st} := r$.
2. The algorithm $\mathsf{LID.Prove}_2$ takes as input a secret key $sk = (N, e, S)$, a commitment cmt, a challenge $\mathsf{ch} \in \{0, \dots, 2^{\ell_e} - 1\}$, a state $\mathsf{st} = r$, and outputs a response $\mathsf{resp} := r \cdot S^{\mathsf{ch}} \bmod N$.

Verification. The verification algorithm $\mathsf{LID.Vrfy}$ takes as input a public key $pk = (N, e, U)$, a commitment cmt, a challenge ch, and a response resp. It outputs 1 if and only if $\mathsf{resp} \neq 0 \bmod N$ and $\mathsf{resp}^e = \mathsf{cmt} \cdot U^{\mathsf{ch}}$.

Simulation. The simulation algorithm $\mathsf{LID.Sim}$ takes as input a public key $pk = (N, e, U)$, a challenge ch, and a response resp. It outputs a commitment $\mathsf{cmt} = \mathsf{resp}^e / U^{\mathsf{ch}}$.

We can prove the following lemma.

Lemma 29. *The ϕ-hiding-based LID scheme* LID *has $\alpha_l \geq \frac{3}{4}\lambda$ and $\alpha_n \geq \frac{3}{4}\lambda$.*

Proof. $\mathsf{LID.Sim}$ outputs $\mathsf{cmt} = \mathsf{resp}^e / U^{\mathsf{ch}}$ for uniform $\mathsf{resp} \in \mathbb{Z}_N^*$ and uniform $\mathsf{ch} \in \{0, \dots, 2^{\ell_e} - 1\}$. Let us first fix the challenge ch and define the e-th residues set as

$$\mathcal{S} := \{x^e \bmod N \mid x \in \mathbb{Z}_N^*\}.$$

Then resp^e distributes uniformly over $\mathcal{S}$ and cmt distributes uniformly over the set $\mathcal{T} = \mathcal{S}/U^{\mathsf{ch}}$. Since $\mathcal{S}$ has size at least $\phi(N)/e$, then $|\mathcal{T}| \geq \phi(N)/e$. Since e is a prime of length $\ell_e \leq \lambda/4$, we have that $\log_2 |\mathcal{T}| \geq \log_2(2^{\frac{3}{4}\lambda}) = \frac{3}{4}\lambda$. Thus, regardless of the input pk is lossy or not, $\mathsf{LID.Sim}$ has min-entropy at least $\log_2 |\mathcal{T}| \geq \frac{3}{4}\lambda$. $\square$

C.4 Lossy Identification Scheme Based on Ring-LWE

In this subsection, we recall the Ring-LWE-based LID scheme as shown in [2,3] and show that α_l is large for this scheme.

Consider the LID scheme $\mathsf{LID} = (\mathsf{LID.Gen}, \mathsf{LID.LossyGen}, \mathsf{LID.Prove}, \mathsf{LID.Vrfy}, \mathsf{LID.Sim})$ as follows:

Parameters. Let n be an integer that is a power of 2. Let σ be any positive real. The distribution $D_{\mathbb{Z}^n,\sigma}$ assigns the probability proportional to $e^{-\pi||\mathbf{y}||^2/\sigma^2}$ to every $\mathbf{y} \in \mathbb{Z}^n$ and 0 everywhere else. For any odd prime p which equals 1 mod $2n$, the ring $\mathcal{R} = \mathbb{Z}_p[\mathbf{x}]/(\mathbf{x}^n + 1)$ is represented by polynomials of degree at most $n-1$ with coefficients in range $[-\frac{p-1}{2}, \frac{p-1}{2}]$. As $\mathcal{R}$ is isomorphic to $\mathbb{Z}_p^n$, we use the notation $\mathbf{y} \xleftarrow{\$} D_{\mathcal{R},\sigma}$ to mean that a vector $\mathbf{y}$ is chosen from the distribution $D_{\mathbb{Z}^n,\sigma}$ and then mapped to a polynomial $\mathcal{R}$ in the natural way. Let $\mathcal{R}^\times$ be the group of all the multiplicative invertible elements in $\mathcal{R}$.

Key generation. The algorithm $\mathsf{LID.Gen}$ samples $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} D_{\mathcal{R},\sigma}$ and $\mathbf{a} \xleftarrow{\$} \mathcal{R}^\times$. Then it computes $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$ It sets $pk = (\mathbf{a}, \mathbf{t})$ and $sk = (\mathbf{s}_1, \mathbf{s}_2)$.

Lossy key generation. The lossy key generation algorithm $\mathsf{LID.LossyGen}$ samples $\mathbf{t} \xleftarrow{\$} \mathcal{R}$.

Proving. The algorithm $\mathsf{LID.Prove}$ is split up into the following two algorithms:

1. The algorithm $\mathsf{LID.Prove}_1$ selects $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{M}$ where $\mathcal{M} = \{\mathbf{g} \in \mathcal{R} : ||\mathbf{g}||_\infty \leq n^{3/2}\sigma \log^3 n\}$ and computes $\mathbf{u} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$. Then it sets commitment $\mathsf{cmt} := \mathbf{u}$. It outputs $(\mathsf{cmt}, \mathsf{st})$ with $\mathsf{st} := (\mathbf{y}_1, \mathbf{y}_2)$.
2. The algorithm $\mathsf{LID.Prove}_2$ takes as input a secret key $sk = (\mathbf{s}_1, \mathbf{s}_2)$, a commitment $\mathsf{cmt} = \mathbf{u}$, a challenge $\mathsf{ch} \in \mathcal{C} = \{\mathbf{g} \in \mathcal{R} : ||\mathbf{g}||_\infty \leq \log n\}$, a state $\mathsf{st} = (\mathbf{y}_1, \mathbf{y}_2)$, it computes $\mathbf{z}_i = \mathbf{s}_i\mathbf{c} + \mathbf{y}_i$ for $i \in \{1,2\}$ and outputs a response $\mathsf{resp} := (\mathbf{z}_1, \mathbf{z}_2)$ if for every $i \in \{1,2\}$, $\mathbf{z}_i \in \mathcal{G} = \{\mathbf{g} \in \mathcal{R} : ||\mathbf{g}||_\infty \leq (n-1)\sqrt{n}\sigma \log^3 n\}$. Otherwise output $\mathsf{resp} := \perp$.

Verification. The verification algorithm $\mathsf{LID.Vrfy}$ takes as input a public key $pk = (\mathbf{a}, \mathbf{t})$, a commitment $\mathsf{cmt} = \mathbf{u}$, a challenge $\mathsf{ch} = \mathbf{c}$, and a response $\mathsf{resp} = (\mathbf{z}_1, \mathbf{z}_2)$. It outputs 1 if and only if for every $i \in \{1,2\}$, $\mathbf{z}_i \in \mathcal{G}$ and $\mathbf{u} = \mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}$.

Simulation. The simulation algorithm $\mathsf{LID.Sim}$ takes as input a public key $pk = (\mathbf{a}, \mathbf{t})$, a challenge $\mathsf{ch} = \mathbf{c}$, and a response $\mathsf{resp} = (\mathbf{z}_1, \mathbf{z}_2)$. It outputs a commitment $\mathsf{cmt} := \mathbf{u} = \mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}$.

Lemma 30. *The Ring-LWE-based LID scheme* LID *has* $\alpha_l \geq n$ *and* $\alpha_n \geq n$.

Proof. Regardless of whether the public key is lossy or not, when the $\mathsf{ch} = \mathbf{c}$ and the response $\mathsf{resp} = (\mathbf{z}_1, \mathbf{z}_2)$ are uniformly chosen, the commitment outputs by the simulation algorithm $\mathbf{u} = \mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}$ has entropy at least $\log_2(|\mathcal{G}|)$ because $\mathbf{z}_2$ is chosen uniformly from $\mathcal{G}$. The above lemma follows since $\log_2(|\mathcal{G}|) \geq n$. □

D Proof of Theorem 20

We bound $\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A})$ through a sequence of games. Let X_i denote the event that the experiment outputs 1 in Game i.

Game 0. This is the original security experiment. In this experiment, adversary $\mathcal{A}$ is provided with a sign oracle Sign and a hash oracle H. In the sequel, it will be useful to have an exact specification of the experiment when instantiated with our signature scheme, including all variables and lists used by the experiment to determine whether the adversary has output a valid forgery. Therefore, we specify the experiment as follows.

- The random oracle is a truly random function $\mathsf{H}\colon \{0,1\}^* \to \mathsf{CSet}$.
- The game initializes $\mathcal{Q} := \emptyset$. Then, it generates the key pair by running $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$. Finally, it starts adversary $\mathcal{A}$ on input pk.
- $\mathsf{Sign}(m)$. When the adversary queries the signing oracle with message m, the game first checks whether there is a pair (m, σ) in set $\mathcal{Q}$, and if this is true, it outputs σ to make sure that $\mathcal{A}$ only receives one signature per message. Otherwise, the game computes $(\mathsf{cmt}, \mathsf{st}) \xleftarrow{\$} \mathsf{LID.Prove}_1(sk)$, and sets $\mathsf{ch} := \mathsf{H}(m, \mathsf{cmt})$ by making a hash query. Then, the game computes
$$\mathsf{resp} := \mathsf{LID.Prove}_2(sk, \mathsf{ch}, \mathsf{cmt}, \mathsf{st}),$$
outputs the signature $\sigma := (\mathsf{ch}, \mathsf{resp})$ to $\mathcal{A}$, and stores the pair (m, σ) in set $\mathcal{Q}$.

– When the adversary $\mathcal{A}$ outputs a candidate forgery (m^*, σ^*), the game checks whether $\mathsf{Vrfy}(pk, m^*, \sigma^*) = 1$ and $(m^*, \sigma^*) \notin \mathcal{Q}$. More precisely, for $\sigma^* = (\mathsf{ch}^*, \mathsf{resp}^*)$, the game first recovers $\mathsf{cmt}^* := \mathsf{LID.Sim}(pk, \mathsf{ch}^*, \mathsf{resp}^*)$ and then queries the random oracle to get $\mathsf{ch}' := \mathsf{H}(m^*, \mathsf{cmt}^*)$. Finally, the game outputs 1 if and only if $\mathsf{ch}^* = \mathsf{ch}'$ and $(m^*, \sigma^*) \notin \mathcal{Q}$.

It is clear that

$$\Pr[X_0] = \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathsf{Sig}}(\mathcal{A}).$$

Note that the experiment requires memory which is linear in the number of hash queries and signing queries of $\mathcal{A}$. We will now gradually modify the game, to prepare a memory-tight reduction to the security of the LID scheme.

Game 1. We modify the way the random oracle H is implemented. In Game 0, $\mathsf{H}\colon \{0,1\}^* \to \mathsf{CSet}$ is a random function. In Game 1, we replace this as follows.

Let RF be a random function whose output space depends on a prefix of its input. That is, RF is a random function such that

$$\mathsf{RF}(\texttt{"hash"} \parallel \cdot)\colon \{0,1\}^* \to \mathsf{CSet},$$

and

$$\mathsf{RF}(\texttt{"sim"} \parallel \cdot)\colon \{0,1\}^* \to \mathsf{CSet} \times \mathsf{RSet}.$$

Remark 31. We stress that the introduction of such a function RF whose output space depends on the input is mainly for notational purpose. We could alternatively assume a random function mapping universally to $\{0,1\}^\ell$ for some suitably large ℓ, and then map to appropriate spaces from there, but this would make the notation unnecessarily complex and make the proof more difficult to verify.

The random oracle H is now implemented by using RF as follows.

– If the input x cannot be parsed as $x = m \parallel \mathsf{cmt}$, then the experiment returns $\mathsf{RF}(\texttt{"hash"} \parallel x)$.
– Otherwise, it parses $m \parallel \mathsf{cmt} := x$ and runs $(\mathsf{ch}, \mathsf{resp}) := \mathsf{RF}(\texttt{"sim"} \parallel m)$ and then $\mathsf{cmt}' := \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$.
  - If $\mathsf{cmt} = \mathsf{cmt}'$, then it returns ch.
  - Otherwise, it returns $\mathsf{RF}(\texttt{"hash"} \parallel x)$.

Claim. $\Pr[X_0] = \Pr[X_1]$.

Proof. If x can be parsed as $x = m \| \mathsf{cmt}$, then recall that $\mathsf{LID.Sim}$ is deterministic and therefore for any m there exists a *unique* cmt' such that

$$\mathsf{cmt}' = \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp}) = \mathsf{LID.Sim}(pk, \mathsf{RF}(\texttt{"sim"} \parallel m))$$

Hence, there exists only one unique value cmt such that $\mathsf{cmt} = \mathsf{cmt}'$, in which case the output of H is obtained from pk and $\mathsf{RF}(\texttt{"sim"} \parallel m)$, which is a random function that depends on m. So, the output in this case also implicitly depends on cmt. Furthermore, if $\mathsf{cmt} \neq \mathsf{cmt}'$ or if x cannot be parsed as $x = m \parallel \mathsf{cmt}$, then the output of H is the output of a random function. Hence, the distribution of H is Game 1 is identical to Game 0, which proves the claim. $\square$

Game 2. We now change the way how signatures are computed. We implement a different signing algorithm, which exploits the definition of H from Game 1 in order to be able to compute valid signatures without using the secret key sk.

Whenever the adversary queries $\mathsf{Sign}(m)$ on input of some message m, the signing oracle computes and returns

$$\sigma := (\mathsf{ch}, \mathsf{resp}) \text{ with } (\mathsf{ch}, \mathsf{resp}) := \mathsf{RF}(\texttt{"sim"} \parallel m), \tag{13}$$

and adds (m, σ) to $\mathcal{Q}$.

Note that this implementation of the signing algorithm does not require to check whether a signing query $\mathsf{Sign}(m)$ has already been made before to answer consistently and to ensure that the adversary $\mathcal{A}$ gets only one signature per message. This implementation always outputs the same signature for a queried message m.

Claim. $\Pr[X_1] \leq \Pr[X_2] + q_{\mathsf{S}} \cdot \varepsilon_s$.

Proof. First of all, note that, by the definition of the random oracle introduced in Game 1, the signatures simulated in Game 2 are valid, that is, we have $\mathsf{Vrfy}(pk, m, \sigma) = 1$ for all messages m queried by the adversary and all signatures σ returned by the experiment. To see this, recall that the verification algorithm first runs

$$\mathsf{cmt} := \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$$

to recover the commitment, which is uniquely determined by $(pk, \mathsf{ch}, \mathsf{resp})$. Then, it checks whether $\mathsf{ch} = \mathsf{H}(m, \mathsf{cmt})$ is satisfied, and H is defined such that this indeed holds.

The difference between the games is that in Game 1 signatures are generated by first computing $(\mathsf{cmt}, \mathsf{st}) \xleftarrow{\$} \mathsf{LID.Prove}_1(sk)$, then $\mathsf{ch} := \mathsf{H}(m, \mathsf{cmt})$, where H is a random function, and then $\mathsf{resp} := \mathsf{LID.Prove}_2(sk, \mathsf{ch}, \mathsf{cmt}, \mathsf{st})$. In contrast, in Game 2, we derive $(\mathsf{ch}, \mathsf{resp}) = \mathsf{RF}(\texttt{"sim"} \parallel m)$ using a (truly) random function.

The ε_s-simulatability of the lossy identification scheme guarantees that the two distributions

$$\left\{ (\mathsf{cmt}, \mathsf{ch}, \mathsf{resp}) : \begin{array}{l} (\mathsf{cmt}, \mathsf{st}) \xleftarrow{\$} \mathsf{LID.Prove}_1(sk) \\ \mathsf{ch} \xleftarrow{\$} \mathsf{CSet} \\ \mathsf{resp} \leftarrow \mathsf{LID.Prove}_2(sk, \mathsf{ch}, \mathsf{cmt}, \mathsf{st}) \end{array} \right\}$$

and

$$\left\{ (\mathsf{cmt}, \mathsf{ch}, \mathsf{resp}) : \begin{array}{l} \mathsf{ch} \xleftarrow{\$} \mathsf{CSet} \\ \mathsf{resp} \xleftarrow{\$} \mathsf{RSet} \\ \mathsf{cmt} \leftarrow \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp}) \end{array} \right\}$$

are statistically ε_s-indistinguishable. Since the signing oracle is queried q_{S} times, the statistical distance between Game 1 and Game 2 is at most $q_{\mathsf{S}} \cdot \varepsilon_s$ by the union bound. Thus, the claim follows. $\square$

Game 3. Now, we change the way how the experiment checks the validity of a candidate forgery. Instead of storing the set $\mathcal{Q}$, it proceeds as follows. When the adversary outputs the candidate forgery (m, σ), where $\sigma = (\mathsf{ch}, \mathsf{resp})$, it checks whether $\mathsf{Vrfy}(pk, m, \sigma) = 1$ and $(\mathsf{ch}, \mathsf{resp}) \neq \mathsf{RF}(\texttt{"sim"} \parallel m)$, and only outputs 1 if *both* holds.

Hence, from Game 3 on, the adversary can only win if it outputs a pair (m, σ) such that $(\mathsf{ch}, \mathsf{resp})$ is *not* the random challenge-response pair that the experiment would have computed by running $(\mathsf{ch}, \mathsf{resp}) := \mathsf{RF}(\texttt{"sim"} \parallel m)$ on input m. The purpose of this modification is to filter out signatures that are not "new" forgeries, which enables us to recognize "valid" forgeries without the need of storing all messages-signature pairs that the adversary has obtained from the experiment in the set $\mathcal{Q}$.

Claim. $\Pr[X_2] \leq \Pr[X_3] + \frac{1}{|\mathsf{RSet}|}$.

Proof. An adversary may distinguish Game 3 from Game 2 in two ways. Either it outputs a pair (m, σ) such that 1 is output in Game 2 but not in Game 3, or the other way around.

First of all, note that every message-signature pair $(m, \sigma = (\mathsf{ch}, \mathsf{resp}))$ generated by the experiment in response to a signing query (and stored in set $\mathcal{Q}$ in Game 2) also satisfies $(\mathsf{ch}, \mathsf{resp}) = \mathsf{RF}(\texttt{"sim"} \parallel m)$, due to the modified signing algorithm introduced in Game 2, Equation (13). Hence, Game 3 never outputs 1 for any signature that would not have triggered Game 2 to output 1. Thus, Game 3 is strictly more restrictive than Game 2.

In the opposite direction, note that Game 3 might be too restrictive, by not outputting 1 even for a signature that was *not* a response to a Sign-query, and thus would not have been stored in set $\mathcal{Q}$ in Game 2. Note that this happens only if the adversary outputs $(m, \sigma = (\mathsf{ch}, \mathsf{resp}))$ such that $(\mathsf{ch}, \mathsf{resp}) = \mathsf{RF}(\texttt{"sim"} \parallel m)$, but the adversary has never received this particular signature in response to a Sign-query. In this case, we say that event bad occurs.

To bound the probability of bad, recall that the *uniqueness* of the LID scheme guarantees that for any $(pk, \mathsf{cmt}, \mathsf{ch})$, where pk is honestly generated, there exists at most one resp such that $\mathsf{LID.Vrfy}(pk, \mathsf{cmt}, \mathsf{ch}, \mathsf{resp}) = 1$, which is exactly the value $(\mathsf{ch}, \mathsf{resp}) = \mathsf{RF}(\texttt{"sim"} \parallel m)$ output by the random function on input m.

Hence, bad occurs only if the adversary is able to "predict" the output resp of RF on input $\texttt{"sim"} \parallel m$. There are only two ways for the adversary to learn information about output of the random function $\mathsf{RF}(\texttt{"sim"} \parallel m)$. Namely, the adversary receives information about $(\mathsf{ch}, \mathsf{resp}) = \mathsf{RF}(\texttt{"sim"} \parallel m)$...

1. *... by asking a signature query for m.* However, if such a query is asked, then the adversary already receives back the *unique* signature $(\mathsf{ch}, \mathsf{resp})$ that satisfies Equation (14), and hence this cannot be a *new* forgery.
2. *... by asking random oracle queries to* H. However, note that it is perfectly indistinguishable for the adversary whether the response to a H-query was computed via

$$(\mathsf{ch}, \mathsf{resp}) := \mathsf{RF}(\texttt{"sim"} \parallel m) \qquad (14)$$

or via

$$\mathsf{ch} := \mathsf{RF}(\texttt{"hash"} \,\|\, m \,\|\, \mathsf{cmt}).$$

Hence, it receives no information from H about the value of resp that satisfies Equation (14). This yields that the probability of predicting it is at most $1/|\mathsf{RSet}|$.

This proves the claim. □

Game 4. In this game, we modify the key generation algorithm of the signature scheme. Instead of running $(pk, sk) \xleftarrow{\$} \mathsf{LID.Gen}(1^\lambda)$, we now run $pk \xleftarrow{\$} \mathsf{LID.LossyGen}(1^\lambda)$. Otherwise, the experiment proceeds identical to Game 3. Note that Game 3 is able to simulate valid signatures without requiring the secret key, which gives rise to the following claim.

Claim. There exists a reduction $\mathcal{B}$ such that

$$\begin{aligned}
\Pr[X_3] &\leq \Pr[X_4] + \mathsf{Adv}^{\mathsf{IND\text{-}KEY}}_{\mathsf{LID}}(\mathcal{B}) \\
\mathbf{LocalTime}(\mathcal{B}) &\leq \mathbf{LocalTime}(\mathcal{A}) + \mathbf{Time}(\mathsf{LID.LossyGen}) \\
&\quad + (q_{\mathsf{S}} + q_{\mathsf{H}} + 1) \cdot \mathbf{Time}(\mathsf{RF}) + \mathbf{Time}(\mathsf{Sig.Vrfy}), \\
\mathbf{LocalMem}(\mathcal{B}) &= \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{LID.LossyGen}) + \mathbf{Mem}(\mathsf{RF}) \\
&\quad + \mathbf{Mem}(\mathsf{Sig.Vrfy}).
\end{aligned}$$

Proof. The reduction is again straightforward. Adversary $\mathcal{B}$ receives as input pk, which is either generated by algorithm $\mathsf{LID.Gen}$ or by $\mathsf{LID.LossyGen}$. Then it simulates Game 4 for the adversary $\mathcal{A}$ such that we have

$$\mathsf{Adv}^{\mathsf{IND\text{-}KEY}}_{\mathsf{LID}}(\mathcal{B}) \geq |\Pr[X_3] - \Pr[X_4]|$$

Note that the reduction is *not* memory-tight, $\mathcal{B}$ does not have to store any random oracle queries or any signatures provided to $\mathcal{A}$, due to the changes introduced in previous games, but it has to store the look-up table for the random function RF. This highly depends on the number of queries issued by the adversary. □

Next, we argue that it is statistically unlikely that $\mathcal{A}$ is able to produce a valid forgery in Game 4. The standard argument for lossy identification schemes from [2, 3, 46] is that, due to the lossyness of the key, for any cmt there exists only one ch such that the signature is valid. When $\mathcal{A}$ queries $\mathsf{H}(m, \mathsf{cmt})$, then it has “committed” to one value of cmt, and the probability that H returns the only “matching” value ch is negligible, since H is a random function.

Claim. $\Pr[X_4] \leq \frac{1}{|\mathsf{CSet}|} + q_{\mathsf{H}} \cdot \varepsilon_\ell$, where q_{H} is the number of hash queries in Game 4.

Proof. We focus on analyzing the probability that adversary $\mathcal{A}$ submits a pair (m^*, σ^*) such that $\mathsf{Vrfy}(pk, m^*, \sigma^*) = 1$ in Game 7. More precisely, for $\sigma^* = (\mathsf{ch}^*, \mathsf{resp}^*)$, the game first recovers $\mathsf{cmt}^* = \mathsf{LID.Sim}(pk, \mathsf{ch}^*, \mathsf{resp}^*)$ and then queries the random oracle to get $\mathsf{ch}' = \mathsf{H}(m^*, \mathsf{cmt}^*)$. Finally, the game compares whether $\mathsf{ch}^* = \mathsf{ch}'$. Then, if the hash query $\mathsf{H}(m^*, \mathsf{cmt}^*)$ has never been made by $\mathcal{A}$ or the game, the probability that $\mathsf{ch}^* = \mathsf{ch}'$ is bounded by $\frac{1}{|\mathsf{CSet}|}$. If this hash query has been made before, we bound this probability by the ε_ℓ-lossy property of LID.

To this end, we build a computationally unbounded adversary $\mathcal{B}$ against the lossy property of LID using $\mathcal{A}$. On input a public key pk generated by the $\mathsf{LID.LossyGen}(1^\lambda)$ algorithm, $\mathcal{B}$ forwards pk to $\mathcal{A}$ and select an index $j \xleftarrow{\$} \{1, \cdots, q_\mathsf{H}\}$ where q_H is the number of hash queries in Game 4. $\mathcal{B}$ could perfectly simulate Game 4 for $\mathcal{A}$ and when the j-th hash query $\mathsf{H}(m, \mathsf{cmt})$ is made, $\mathcal{B}$ forwards cmt to its own challenger and the challenger responds with a random challenge $\mathsf{ch} \xleftarrow{\$} \mathsf{CSet}$. $\mathcal{B}$ then returns ch as the response to the hash query. When $\mathcal{A}$ submits a valid forgery, $\mathcal{B}$ outputs resp^* to its own challenger. It is obvious that $\mathcal{B}$ could win if the j-th query is the first time that $\mathsf{H}(m^*, \mathsf{cmt}^*)$ is made. This happens with probability $\frac{1}{q_\mathsf{H}}$. Thus, we have that $\Pr[X_4] \leq \frac{1}{|\mathsf{CSet}|} + q_\mathsf{H} \cdot \varepsilon_\ell$.

Note that this lossy property is actually a statistical property and our analysis here is actually a statistical argument. We do not reduce to any computational problem so the adversary $\mathcal{B}$ could use arbitrary time and memory without breaking the memory tightness security of our signature.

Remark 32. Observe that the above scheme can be rewritten to be memory-tight proof by simply replacing the random function RF before Game 4 by a pseudorandom function to implement this memory-efficiently in the reduction to $\mathsf{IND\text{-}KEY}$-security. This needs to be reverted after Game 4 to make the final statistical argument go through. These changes imply that we need to add two reductions to the PRF security to the sequence of games as described before and thus that the bound on $\mathcal{A}$'s advantage in Theorem 20 would be extended by the advantage terms of the PRF security of the two respective reductions.

E Proof of Theorem 21

Proof. We bound the predictability probability ε_p of $\mathcal{R}_\mathsf{LID}$. Consider an adversary $\mathcal{A}$ interacting with $\mathcal{R}_\mathsf{LID}$ and the event that $\mathcal{A}$ submits a $\mathsf{Forge}(m^*, \sigma^*)$ query with

$$\mathsf{Sig.Vrfy}(simpk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q} \wedge \sigma^* = \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*).$$

This would imply that $\mathcal{A}$ never queries m^* to $\mathsf{Sign}(\cdot)$ oracle, otherwise $\mathcal{A}$ gets $\sigma^* = \mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*)$ as return and $(m^*, \sigma^*) \in \mathcal{Q}$. According to the construction of $\mathcal{R}_\mathsf{LID}$,

$$\mathsf{RSign}^{\mathsf{RF}(\cdot)}(simsk, m^*) = (\mathsf{ch}, \mathsf{resp}) = \mathsf{RF}(\texttt{"sim"} \parallel m^*)$$

and $\mathsf{RF}(\texttt{"sim"} \parallel m^*)$ is not returned to $\mathcal{A}$ through the $\mathsf{Sign}(\cdot)$ oracle because $\mathcal{A}$ never queries m^* to $\mathsf{Sign}(\cdot)$. The only possible reveal of $\mathsf{RF}(\texttt{"sim"} \parallel m^*)$ is in the $\mathsf{H}(\cdot)$ oracle when adversary $\mathcal{A}$ submits a $\mathsf{H}(m^* \parallel \mathsf{cmt}')$ query such that cmt' equals $\mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$ and $\mathcal{A}$ gets back ch as return.

– If $\mathcal{A}$ has never submitted a $\mathsf{H}(m^* \| \mathsf{cmt}')$ query such that $\mathsf{cmt}' = \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$, then $\mathsf{RF}(\texttt{"sim"} \parallel m^*)$ is completely hidden to $\mathcal{A}$ and the probability of correctly guessing it is $\frac{1}{|\mathsf{CSet}| \times |\mathsf{RSet}|}$.
– If $\mathcal{A}$ has ever submitted a $\mathsf{H}(m^* \parallel \mathsf{cmt}')$ query such that $\mathsf{cmt}' = \mathsf{cmt}$ where $\mathsf{cmt} = \mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$, we show that this subcase happens with small probability. This is because the commitment output by $\mathsf{LID.Sim}(pk, \mathsf{ch}, \mathsf{resp})$ has α-bit min-entropy. Thus, the probability of this subcase is bounded by $\frac{q_\mathsf{H}}{2^\alpha}$ where q_H is the number of $\mathsf{H}(\cdot)$ queries made by $\mathcal{A}$. Note that since the min-entropy of $\mathsf{LID.Sim}$ is defined with respect to both normal public key and lossy public key, so we can apply this bound even for the pk that $\mathcal{R}_\mathsf{LID}$ receives (which is normal with probability one half and lossy with probability one half).

So we have

$$\varepsilon_\mathsf{p} \leq \frac{1}{|\mathsf{CSet}| \times |\mathsf{RSet}|} + q_\mathsf{H} \cdot \left(\frac{1}{2^\alpha} + \varepsilon_s\right)$$

and Theorem 21 follows. □

F On the Overall Tightness of RSA-FDH+

The scheme RSA-FDH+ is formally define it as follows:

– Gen runs $(N, e, d) \xleftarrow{\$} \mathsf{GenRSA}(1^\lambda)$ and returns $pk = (N, e), sk = (N, d)$.
– $\mathsf{Sign}(sk, m)$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$ and returns $\sigma = H(b\|m)^d \bmod N$ where H is a hash function with image space $\mathbb{Z}_N$.
– $\mathsf{Vrfy}(pk, m, \sigma)$ returns 1 if and only if either $\sigma^e = H(1 \parallel m) \bmod N$ or $\sigma^e = H(0 \parallel m) \bmod N$.

For detailed proof of work-factor-tightness, we refer to [46, Thm. 2]. Katz and Wang show that RSA-FDH+ is sEUF-CMA-secure provided that the RSA assumption holds. In particular, their definition is a statefull variant of the above scheme. That is, Sign will only output a signature for all messages m for either $b = 0$ or $b = 1$, determined by the first call of Sign on message m.

The reduction is work-factor-tight with a loss of 1/2 and in the random oracle model. We remark, that the reduction by Katz and Wang is almost memory-tight apart from the fact that it requires to maintain the random oracle table. Since Katz and Wang have a stateful variant that only gives out one signature per message by definition of Sign, we trivially get $\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\text{RSA-FDH+}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\text{stRSA-FDH+}}(\mathcal{A})$, where stRSA-FDH+ refers to the stateful variant considered by Katz and Wang. It is fairly straightforward to transfer the proof of [46, Thm. 2] into the following canonical reduction $\mathcal{R}_{\mathsf{RSA+}}$. As before, we define the canonical reduction $\mathcal{R}_{\mathsf{RSA+}}$ from $\mathsf{sEUF\text{-}CMA1}$-security of RSA-FDH+ to the RSA assumption as the tuple $(\mathsf{RGen}, \mathsf{RSign}, \mathsf{RExtract}, \mathsf{RHash})$ as follows. Let $\mathsf{RF}\colon \{0,1\}^* \to \{0,1\} \times \mathbb{Z}_N \times \mathbb{Z}_N$ with $\mathsf{Coins}_\mathsf{RSign} = \mathsf{Coins}_\mathsf{RExtract} = \emptyset$ and $\{0,1\} \times \mathbb{Z}_N \times \mathbb{Z}_N = \mathsf{Coins}_\mathsf{RHash}$.

RGen**:** Given an RSA instance $\phi = (N, e, y)$, RGen returns $(simpk, simsk) = ((N, e), (N, e, y))$.

$\mathsf{RHash}^{\mathsf{RF}(\cdot)}$**:** Given $simsk = (N, e, y)$ and $x = b' \parallel m$, RHash computes $(b, r_b, r_{1-b}) \coloneqq \mathsf{RF}(m)$ and if $b' = b$, it returns r_b^e. Otherwise, it returns $r_{1-b}^e \cdot y$.

$\mathsf{RSign}^{\mathsf{RF}(\cdot)}$**:** Given $simsk = (N, e, y)$ and m, RSign computes $(b, r_b, r_{1-b}) \coloneqq \mathsf{RF}(m)$ and outputs r_b.

$\mathsf{RExtract}^{\mathsf{RF}(\cdot)}$**:** Given $simsk = (N, e, y)$ and (m^*, σ^*), $\mathsf{RExtract}$ computes $(b, r_b, r_{1-b}) \coloneqq \mathsf{RF}(m^*)$ and outputs solution $\rho = \sigma^*/r_{1-b}$. Note that by definition $\mathcal{R}_{\mathsf{RSA}}$ runs $\mathsf{RExtract}$ only if $\mathsf{Vrfy}(simpk, m^*, \sigma^*) = 1$ and $\sigma^* \neq \mathsf{RSign}(simsk, m^*)$. This implies that $(\sigma^*)^e = \mathsf{RHash}(1 - b \parallel m^*)$ for $(b, \cdot) \coloneqq \mathsf{RF}(m^*)$ and thus $\rho = y^d = x$.

As shown by Katz-Wang, we get

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{RSA}},\lambda}(\mathcal{R}_{\mathsf{RSA}+}) \geq \frac{1}{2}\,\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\text{stRSA-FDH+}}(\mathcal{A}) \geq \frac{1}{2}\,\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\text{RSA-FDH+}}(\mathcal{A})$$

and

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{R}_{\mathsf{RSA}+}) &\leq \mathbf{LocalTime}(\mathcal{A}) + (q_{\mathsf{S}} + q_{\mathsf{H}} + 1) \cdot \mathbf{Time}(\mathsf{RF}) \\ &\quad + \mathbf{Time}(\mathsf{Sig.Vrfy}), \\ \mathbf{LocalMem}(\mathcal{R}_{\mathsf{RSA}+}) &= \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{RF}) + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + 3.\end{aligned}$$

With the canonical reduction $\mathcal{R}_{\mathsf{RSA}+}$, we now can apply Theorem 10 to lift the security of RSA-FDH+ to the multi-challenge setting in a memory-tight way. To this end, we construct a reduction $\mathcal{R}'_{\mathsf{RSA}+}$ from $\mathsf{msEUF\text{-}CMA1}$ security of RSA-FDH+ to the RSA assumption as presented in the proof of Theorem 10. This implies that we can construct an adversary $\mathcal{B}'$ such that

$$\mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{RSA}},\lambda}((\mathcal{R}'_{\mathsf{RSA}+})^{\mathcal{A}'}) \geq \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{msEUF\text{-}CMA1}}_{\text{RSA-FDH+}}(\mathcal{A}') - \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}(\mathcal{B}')$$

where $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\} \times \mathbb{Z}_N \times \mathbb{Z}_N$ is a keyed PRF. Moreover, it holds that

$$\begin{aligned}\mathbf{LocalTime}((\mathcal{R}'_{\mathsf{RSA}+})^{\mathcal{A}'}) &\approx \mathbf{LocalTime}(\mathcal{A}') + \mathbf{Time}(\mathsf{RGen}) \\ &\quad + (q_{\mathsf{S}} + q_{\mathsf{F}} + q_{\mathsf{H}}) \cdot \mathbf{Time}(\mathsf{PRF}) + q_{\mathsf{F}} \cdot \mathbf{Time}(\mathsf{Sig.Vrfy}) \\ \mathbf{LocalMem}((\mathcal{R}'_{\mathsf{RSA}+})^{\mathcal{A}'}) &= \mathbf{LocalMem}(\mathcal{A}') + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + \mathbf{Mem}(\mathsf{PRF}) + 4.\end{aligned}$$

Hence, $\mathcal{R}'_{\mathsf{RSA}+}$ is a fully tight reduction (i.e., work-factor-tight and memory-tight), from $\mathsf{msEUF\text{-}CMA1}$-security of RSA-FDH+ to the RSA assumption. Applying the transform of Section 4 and adding an additional nonce that is signed along with the message, we can further lift this result to achieve $\mathsf{msEUF\text{-}CMA}$-security under the RSA assumption.

G On the Memory-Tightness of BLS

Similarly to RSA-FDH discussed in Section 5.2, we can argue memory-tightness for the pairing-based signature scheme proposed by Boneh, Lynn, and Shacham (BLS) [15,16]. In this section, we briefly recall the construction and how our result can be applied to it.

We briefly recall the computational Diffie-Hellman (CDH) problem in the symmetric pairing setting as a non-interactive assumption.

Definition 33. *Let* GGen *be an algorithm that takes as input the security parameter* 1^λ *and returns* $(\mathbb{G}, \mathbb{G}_T, p, g, e)$*, where* $\mathbb{G}$ *and* $\mathbb{G}_T$ *are groups of prime order* $p \geq 2^\lambda$*,* g *is a generator of* $\mathbb{G}$*, and* $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *is a bilinear (type-1) pairing. The* CDH *assumption with respect to* GGen *is a non-interactive computational* $\Lambda_{\mathsf{CDH}} = (\mathsf{InstGen}_{\mathsf{CDH}}, \mathsf{V}_{\mathsf{CDH}}, \mathsf{U}_{\mathsf{CDH}})$ *where*

1. $\mathsf{InstGen}_{\mathsf{CDH}}(1^\lambda)$ *runs* $(\mathbb{G}, \mathbb{G}_T, p, g, e) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$*, chooses* $x, y \xleftarrow{\$} \mathbb{Z}_p$*, and outputs a problem instance* $\phi = (\mathbb{G}, \mathbb{G}_T, p, e, g, X, Y) = (\mathbb{G}, \mathbb{G}_T, p, e, g, g^x, g^y)$ *and a witness* $\omega = (x, y)$*.*
2. $\mathsf{V}_{\mathsf{CDH}}(\phi, \omega, \rho)$ *returns* 1 *if and only if* $\mathsf{dlog}_g(\rho) = xy \mod p$ *with* $(x, y) = \omega$*.*
3. $\mathsf{U}_{\mathsf{CDH}}(\phi)$ *returns a failure symbol* $\perp$*.*

Next, recall the BLS signature scheme. For simplicity, we only present the variant using a symmetric (type 1) pairing. Note that the construction can easily be adapted to asymmetric (type 2 or 3) pairings. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order p and let g be a generator of $\mathbb{G}$. Let $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a type-1 pairing and let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{G}$.

- Gen chooses $x \xleftarrow{\$} \mathbb{Z}_p$, computes g^x and returns $pk := (g, g^x)$ and $sk := x$.
- $\mathsf{Sign}(sk, m)$ returns $\sigma := \mathsf{H}(m)^x \in \mathbb{G}$.
- $\mathsf{Vrfy}(pk, m, \sigma)$ returns 1 if and if $e(H(m), g^x) = e(\sigma, g)$.

The scheme as presented above provides $\mathsf{EUF\text{-}CMA}$ security based on CDH in the random oracle model as shown in [15]. When instantiated with an asymmetric pairing the $\mathsf{EUF\text{-}CMA}$ security is based on the $\mathsf{co\text{-}CDH}$ in the random oracle model. However, the proof by BLS [15] is similar to RSA-FDH discussed in Section 5.2 neither work-factor tight (linear loss in the number of signature queries) nor memory-tight (implementing the random oracle).

To prove memory-tightness for BLS, we can essentially follow the same arguments given for RSA-FDH. To this end, we need to argue that BLS is memory-tightly $\mathsf{sEUF\text{-}CMA1}$-secure in the random oracle model. Similar to RSA-FDH, BLS is a unique signature scheme, i.e., for every m there is exactly one valid signature, namely $\sigma = \mathsf{H}(m)^x$. Hence, we have

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathrm{BLS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{BLS}}(\mathcal{A}).$$

as in Equation (10) for RSA-FDH. It remains to argue that the reduction for $\mathsf{sEUF\text{-}CMA1}$ security is memory-tight up to the usage of a random function RF with an explicitly stored look-up table. Since the reduction $\mathcal{B}$ given in the proof of [16, Thm. 3.2] only stores the three group elements for the CDH challenge, one integer for randomizing the public key and the random oracle table, we can implement the internal randomness of $\mathcal{B}$ using a random function RF and compute the random oracle values on the fly. Given the result of [16, Thm. 3.2] it is easy to verify that

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA1}}_{\mathrm{BLS}}(\mathcal{A}) \leq \exp(1) \cdot (q_{\mathsf{S}} + 1) \cdot \mathsf{Adv}^{\mathsf{NICA}}_{\Lambda_{\mathsf{CDH}},\lambda}(\mathcal{B}').$$

where q_{S} is the number of signatures queried by $\mathcal{A}$ and $\mathcal{B}'$ is as exactly as $\mathcal{B}$ from the proof of [16, Thm. 3.2], but using a random function RF to derive the randomness of for answering the RO queries. We have

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{B}) &\approx \mathbf{LocalTime}(\mathcal{A}) + (q_{\mathsf{H}} + q_{\mathsf{S}}) \cdot \mathbf{Time}(\mathsf{RF}),\\ \mathbf{LocalMem}(\mathcal{B}) &= \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{RF}) + 3.\end{aligned}$$

Next, we can define the canonical reduction $\mathcal{R}_{\mathsf{CDH}}$ from the $\mathsf{sEUF\text{-}CMA1}$ security to the CDH assumption as the tuple $(\mathsf{RGen}, \mathsf{RSign}, \mathsf{RExtract}, \mathsf{RHash})$. To that end, let $\mathsf{RF}\colon \{0,1\}^* \to \{0,1\} \times \mathbb{Z}_p$ with $\mathsf{Coins}_{\mathsf{RSign}} = \mathsf{Coins}_{\mathsf{RExtract}} = \emptyset$ and $\{0,1\} \times \mathbb{Z}_p = \mathsf{Coins}_{\mathsf{RHash}}$. Further, for $(c, b) \coloneqq \mathsf{RF}(x)$, we define the short-hands $c =: \mathsf{RF}_1(x)$ and $b =: \mathsf{RF}_2(x)$. We view RF_1 as an $(1/q_{\mathsf{S}} + 1)$-biased random function similar to the biased coin used by Coron [20], i.e., $\Pr[\mathsf{RF}_1(x) = 0] = 1/(q_{\mathsf{S}} + 1)$, where q_{S} is the number of signature queries issued by the adversary. This is similar to RSA-FDH discussed above.

RGen: Given an CDH instance $(\mathbb{G}, \mathbb{G}_T, p, e, g, X, Y)$, RGen returns $(\mathit{simpk}, \mathit{simsk}) = ((g, u), (g, X, Y, u, r))$ with $u = X \cdot g^r$ and $r \xleftarrow{\$} \mathbb{Z}_p$.

$\mathsf{RHash}^{\mathsf{RF}(\cdot)}$: Given $\mathit{simsk} = (g, X, Y, u, r)$ and x, RHash computes $c = \mathsf{RF}_1(x)$ and $b \coloneqq \mathsf{RF}_2(x)$, and returns $Y^{1-c} \cdot g^b$.

$\mathsf{RSign}^{\mathsf{RF}(\cdot)}$: Given $\mathit{simsk} = (g, X, Y, u, r)$ and m, RSign outputs a signature $\sigma = u^b \cdot g^{rb}$ with $b = \mathsf{RF}_2(m)$ if $\mathsf{RF}_1(m) = 0$. Otherwise, the reduction aborts and terminates by outputting the failure symbol $\perp$.

$\mathsf{RExtract}^{\mathsf{RF}(\cdot)}$: Given $\mathit{simsk} = (g, X, Y, u, r)$ and (m^*, σ^*), $\mathsf{RExtract}$ outputs a solution $\rho = \frac{\sigma^*}{Y^r \cdot u^b \cdot g^{rb}}$. Note that by definition $\mathcal{R}_{\mathsf{CDH}}$ runs $\mathsf{RExtract}$ only if $\mathsf{Vrfy}(\mathit{simpk}, m^*, \sigma^*) = 1$ and $\sigma^* \neq \mathsf{RSign}(\mathit{simsk}, m^*)$. The validity of the signature implies that $(\sigma^*)^e = \mathsf{RHash}(\mathit{simsk}, m^*)$ and since we have $\sigma^* \neq \mathsf{RSign}(\mathit{simsk}, m^*)$, we also know that $\mathsf{RF}_1(m^*) = 0$.

Note that the construction can be easily adapted to an asymmetric pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, by applying the isomorphism $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ to ensure that the values are in the right groups.

Reduction $\mathcal{R}_{\mathsf{CDH}}$ is a $(\ell, 0)$-canonical reduction for BLS with loss $\ell = \exp(1) \cdot (q_{\mathsf{S}} + 1)$, it runs in time

$$\begin{aligned}\mathbf{LocalTime}(\mathcal{R}^{\mathcal{A}}_{\mathsf{CDH}}) \approx \mathbf{LocalTime}(\mathcal{A}) &+ \mathbf{Time}(\mathsf{Sig.Vrfy})\\ &+ (2 \cdot q_{\mathsf{H}} + q_{\mathsf{S}} + 1) \cdot \mathbf{Time}(\mathsf{RF}),\end{aligned}$$

and requires memory

$$\mathbf{LocalMem}(\mathcal{R}_{\mathsf{CDH}}^{\mathcal{A}}) = \mathbf{LocalMem}(\mathcal{A}) + \mathbf{Mem}(\mathsf{RF}) + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + 4.$$

Now, we can use Theorem 10 to lift the security of BLS to the multi-challenge in a memory-tight way. To this end, we can construct a reduction $\mathcal{R}'_{\mathsf{CDH}}$ from $\mathsf{msEUF\text{-}CMA1}$ security of BLS to the CDH assumption as presented in the proof Theorem 10. This supplies that we can construct an adversary $\mathcal{D}$ such that

$$\mathsf{Adv}_{\Lambda_{\mathsf{CDH}},\lambda}^{\mathsf{NICA}}((\mathcal{R}'_{\mathsf{CDH}})^{\mathcal{A}'}) \geq \frac{\mathsf{Adv}_{\mathrm{BLS}}^{\mathsf{msEUF\text{-}CMA1}}(\mathcal{A}')}{\exp(1)\cdot(q_{\mathsf{S}}+1)} - \mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{PRF\text{-}sec}}(\mathcal{D})$$

where $\mathsf{PRF}\colon \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\} \times \mathbb{Z}_p$ is a keyed function. Moreover, it holds that

$$\begin{aligned}\mathbf{LocalTime}((\mathcal{R}'_{\mathsf{CDH}})^{\mathcal{A}'}) &\approx \mathbf{LocalTime}(\mathcal{A}') + \mathbf{Time}(\mathsf{RGen}) \\ &\quad + (q_{\mathsf{S}} + q_{\mathsf{F}} + 2q_{\mathsf{H}})\cdot\mathbf{Time}(\mathsf{PRF}) + q_{\mathsf{F}}\cdot\mathbf{Time}(\mathsf{Sig.Vrfy}) \\ \mathbf{LocalMem}((\mathcal{R}'_{\mathsf{CDH}})^{\mathcal{A}'}) &= \mathbf{LocalMem}(\mathcal{A}') + 5 + \mathbf{Mem}(\mathsf{Sig.Vrfy}) + \mathbf{Mem}(\mathsf{PRF}).\end{aligned}$$

Thus, the reduction $\mathcal{R}'_{\mathsf{CDH}}$ is a memory-tight, but not work-factor-tight, reduction from $\mathsf{msEUF\text{-}CMA1}$-security to the CDH assumption. As BLS is a unique signature scheme as RSA-FDH, one-signature-per-message security implies many-signatures-per-message security.

Similarly to RSA-FDH, we can define a variant BLS+ by applying the technique by Katz and Wang [46] and sign the message with a uniformly chosen bit to achieve work-factor tightness. This extension works exactly as for RSA-FDH. We refer to the discussion of RSA-FDH+ above.