Hiding in Plain Sight: Memory-tight Proofs via Randomness Programming

Ashrujit Ghoshal[1], Riddhi Ghosal[2], Joseph Jaeger[3], and Stefano Tessaro[1]

[1] Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, Washington, US
ashrujit@cs.washington.edu,tessaro@cs.washington.edu
[2] University of California, Los Angeles, US
riddhi@cs.ucla.edu
[3] Georgia Institute of Technology, Atlanta, Georgia, US
josephjaeger@gatech.edu

Abstract. This paper continues the study of *memory-tight reductions* (Auerbach et al, CRYPTO '17). These are reductions that only incur minimal memory costs over those of the original adversary, allowing precise security statements for memory-bounded adversaries (under appropriate assumptions expressed in terms of adversary time and memory usage). Despite its importance, only a few techniques to achieve memory-tightness are known and impossibility results in prior works show that even basic, textbook reductions cannot be made memory-tight.

This paper introduces a new class of memory-tight reductions which leverage random strings in the interaction with the adversary to hide state information, thus shifting the memory costs to the adversary.

We exhibit this technique with several examples. We give memory-tight proofs for digital signatures allowing many forgery attempts when considering randomized message distributions or probabilistic RSA-FDH signatures specifically. We prove security of the authenticated encryption scheme Encrypt-then-PRF with a memory-tight reduction to the underlying encryption scheme. By considering specific schemes or restricted definitions we avoid generic impossibility results of Auerbach et al. (CRYPTO '17) and Ghoshal et al. (CRYPTO '20).

As a further case study, we consider the textbook equivalence of CCA-security for public-key encryption for one or multiple encryption queries. We show two qualitatively different memory-tight versions of this result, depending on the considered notion of CCA security.

Keywords: Provable security, memory-tightness, time-memory trade-offs

1 Introduction

The aim of concrete security proofs is to lower bound, as precisely as possible, the resources needed to break a cryptographic scheme of interest, under

some plausible assumptions. The traditional resource used in provable security is *time complexity* (as well as related metrics, like data complexity). Recent works [1,22,21,17,12,11,7,16,15,9,20,10] have focused on additionally taking the *memory* costs of the adversary into account. This is important, as the amount of available memory can seriously impact the feasibility of an attack.

This paper presents new techniques for *memory-tight reductions*, a notion introduced by Auerbach et al. [1] to relate the assumed time-memory hardness of an underlying computational problem to the security of a scheme. More precisely, the end goal is to prove, via a reduction, that any adversary running in time t and with s bits of memory can achieve at most advantage $\epsilon = \epsilon(t, s)$ in compromising a scheme, by assuming that some underlying computational problem can only be solved with advantage $\delta = \delta(t', s')$ by algorithms running in time t' and with memory s'. A memory-tight reduction guarantees that $s \approx s'$, and usually, we want this to be tight also according to other parameters, i.e., $t \approx t'$ and $\epsilon \approx \delta$.

Memory-tight reductions are of value whenever the underlying problem is (conjectured to) be *memory sensitive*, i.e., the time needed to solve it grows as the amount of memory available to the adversary is reduced. Examples of memory-sensitive problems include classical ones in the public-key setting, such as breaking RSA and factoring, lattice problems and LPN, solving discrete logarithms over finite fields,[4] as well as problems in the secret-key setting, such as finding k-way collisions (for $k > 2$), finding several collisions at once [11], and distinguishing random permutations from random functions [17,12,20].

Developing memory-tight reductions is not always easy, and can be (provably) impossible [1,22,16,15]. This makes it fundamental to develop as many techniques as possible to obtain such reductions. In this paper, we identify a class of examples which admit a new kind of memory-tight reductions. Our approach relies on the availability of random strings exchanged between the adversary and the security game, and which the reduction can leverage to encode state which can be recovered from later queries of the adversary, without the need to store this information locally, and thus saving memory. (In particular, the burden of keeping this information remains on the adversary, which needs to reproduce this random string for this state information to be relevant.) We present these techniques abstractly in the next section, with the help of a motivating example, and then move on to an overview of our specific results.

1.1 Our Techniques - An Overview

As a motivating example, consider the standard UFCMA security notion for signatures. It is defined via a game where the attacker, given the verification key vk, obtains signatures for chosen messages $m_1, m_2, \ldots$, after which it outputs a candidate message-signature pair (m^*, σ^*), and wins if m^* was not signed before, and σ^* is valid for m^*. When ignoring memory, this notion is *tightly* equivalent

[4] However, the discrete logarithm problem in elliptic-curve groups, or any other group in which the best-known attacks are generic, is not memory sensitive, since optimal memory-less attacks are known.

to one (which we refer to as mUFCMA) that allows for an arbitrary number of "forgery attempts" for pairs (m^*, σ^*), and the adversary wins if one of them succeeds in the above sense. This is convenient: we generally target mUFCMA, but only need to deal with proving the simpler UFCMA notion.

The classical reduction transforms any mUFCMA adversary into a roughly equally efficient UFCMA adversary, which wins with the same probability, by (1) simulating forgery queries using the verification key, and (2) outputting the first forgery query (m^*, σ^*) which validates and such that m^* is fresh. This reduction is however *not* memory-tight, as we need to ensure the freshness of m^*, which requires remembering the previously signed messages. ACFK [1] prove that this is in some sense necessary, by showing that a (restricted) class of reductions cannot be memory-tight via a reduction to streaming lower bounds.

Our idea: Efficient tagging. To illustrate our new technique, which we refer to as *efficient tagging*, imagine now that we only use the signature scheme to sign *random* messages $m_1, m_2, \ldots, m_q \leftarrow\!\!\$\ \{0,1\}^\ell$, and consider a corresponding variant of mUFCMA security, which we want to reduce to (plain) UFCMA security. This, intuitively, does not seem to help resolve the above issue, because random messages are hardest to compress.

However, what is important here is that the reduction is responsible for simulating the random messages, and can simulate them in special ways, and *program* them so that they encode state information. For instance, assume that the reduction has access to an *injective* random function $f : [q] \to \{0,1\}^\ell$, with inverse f^{-1}, which can be simulated *succinctly* from a short key as a pseudorandom object. Then, the reduction to UFCMA can set $m_i \leftarrow f(i)$ for the i-th query, and upon simulating a forgery query for (m^*, σ^*), the reduction checks whether $f^{-1}(m^*) \in [q]$ to learn whether m^* is a fresh signing query or not.

Of course, the simulation is not perfect: The original m_i's are not necessarily distinct (this can be handled via the classical "switching lemma"). Also, the reduction could miss a valid forgery if the adversary outputs m_i *before* it is given to the adversary, but this again only occurs with small probability.

Inefficient tagging and non-time-tight reductions. In the above example, we can efficiently check that $f^{-1}(m^*) \in [q]$. However, in some cases we may not – again, consider an example where the messages to be signed are sampled as $m_i \leftarrow h(r_i)$, where h is a hard-to-invert function and r_i is random. Then, we could adapt our proof above by setting $m_i \leftarrow h(f(i))$, but now, to detect a prior signing query, we would have to check whether $m^* = h(f(i))$ for some $i \in [q]$, and this can only be done in linear time. The resulting UFCMA adversary runs in time $t' = t + \Theta(q_F \cdot q)$, where t is the running time of the original adversary and q_F is the number of forgery attempts. For example, if $q \approx q_F \approx t$, the reduction is not time tight, and the adversary runs in time $t' = O(t^2)$.

Are non-time-tight reductions useless? It turns out that such non-time-tight reductions can still be helpful to infer that breaking a scheme requires memory, although this ultimately depends on the concrete security of the problem targeted by the reduction. Say, for example, a reduction for a given scheme transforms a successful adversary running in time t and using memory s into an

adversary running in time t^2 and using memory s breaking discrete logarithms over $\mathbb{F}_p$, for a 4096-bit prime p. It turns out that if we have fewer than 2^{78} bits of memory, no known discrete logarithm algorithm is better than a generic one (i.e. runs in time better than 2^{2048}), which means that our non-time-tight reduction is still sufficient to infer security for any $s < 2^{78}$ as long as $t < 2^{1024}$.

Message encoding. At the highest level, what happens is that the reduction is in control of certain random values which we can exploit to hide state information which can later be uniquely recovered, since triggering a situation where the reduction needs to remember requires the adversary to actually give back to the reduction this value. In the above, this state information is simple, namely whether the query is old or not. But as we will show below, the paradigm can be used to store complex information – we refer to this technique as *message encoding*, and discuss an example below.

A New Viewpoint: $\mathcal{F}$-oracle adversaries. In our technique described above we needed access to a large random injection, which we argue can be simulated pseudorandomly. Prior works have similarly used PRFs to pseudorandomly simulate random oracles [1,6] with low memory. The fact that one needs to decide how to simulate such objects when stating a memory-tight reduction is rather inconvenient: different instantiations seemingly lead to quantitatively different reductions, although this fact does not appear to be a reflection of any particular reality. In this paper, we propose (and advocate for) what we believe to be the "right" viewpoint: Our reductions are stated in terms of $\mathcal{F}$-oracle adversaries where $\mathcal{F}$ is a set of functions and such an adversary expects oracle access to a random $f \in \mathcal{F}$. Then, a memory-tightness theorem is obtained in one of two ways, by either (1) applying a generic lemma stating that f can be instantiated in low memory using an $\mathcal{F}$-pseudorandom function, or (2) assuming that the use of f does not functionally increase the success chances of the adversary because f is independent of the problem instance being solved (this is provably the case for some information theoretic problems). In particular, (1) is more conservative than (2), but it is very likely that (2) is also a viable approach which leads to cleaner result – indeed, we do not expect any of the considered memory-sensitive problems to become easier given access to an oracle from any natural class $\mathcal{F}$ – e.g., Factoring does not become easier given access to a random injection.

1.2 Our Results

We now move to an overview of our results (summarized in Fig. 1) which exemplify different applications of the tagging and message-encoding techniques.

Multi-challenge Security of Digital Signatures. Our first results consider the security of digital signatures in the face of multiple forgery attempts (i.e., challenge queries), generalizing the examples discussed above. We work with a notion we refer to as UFRMA (unforgeability under randomized message attack). This notion is parameterized by a *message distribution* D and when the attacker makes a signing query for m it receives a signature of $m' = \mathsf{D}(m; r)$ for a random r. If m and r can be extracted from m', giving the notion xUFRMA

(or mxUFRMA for many forgery attempts), we can generalize our efficient tagging approach above by having the reduction to UFCMA choose $r = f(m, i)$ where each $f(m, \cdot)$ is a random injection. This setting can capture, e.g., the signatures used in key exchange protocols like TLS 1.3 where the server signs a transcript which includes a random 256-bit nonce. A version of our inefficient tagging example works when only m can be extracted from m' (wUFRMA); we pick $r = f(m, i)$ and in verification of a forgery query perform the linear time check of whether $m^* = \mathsf{D}(m; f(m, i))$ for some $i \in [q]$. This setting captures places where the message to be signed includes a fresh public key or ciphertext. This includes, for example, the use of signatures for signing certificates, in some key exchange protocols, and in signcryption.

We further prove mUFCMA security for particular schemes. First, we can randomize any digital signature scheme DS (obtaining a scheme we call RDS) by signing $m \,\|\, r$ for random r chosen by the signing algorithm and including r as part of the signature. An immediate implication of our mxUFCRA result is a tight reduction from the mUFCMA security of RDS to the UFCMA security of the underlying scheme. One particular instantiation of RDS is Probabilistic Full Domain Hash with RSA ($\mathsf{RSA\text{-}PFDH}$) which was introduced by Coron [8] to provide a variant of Full Domain Hash [4] with an (advantage-) tighter security proof. Using our efficient tagging technique we obtain a fully tight proof of the *strong* mUFCMA security of $\mathsf{RSA\text{-}PFDH}$ from the RSA assumption.

In independent and concurrent work, Diemert, Gellert, Jager, and Lyu [10] studied the mUFCMA security of digital signature schemes. They also considered the RDS construction, proving that if DS can be proven strong UFCMA1 secure[5] with a restricted class of "canonical" memory-tight reductions then there is a memory-tight reduction for the strong mUFCMA security of RDS. This complements our result, showing memory-tight strong mUFCMA security of RDS based on a restricted class of schemes while our result proves memory-tight plain mUFCMA security based on any plain UFCMA scheme. They apply their RDS result to establish tight proofs for the strong mUFCMA security of $\mathsf{RSA\text{-}PFDH}$ (matching our direct proof in Theorem 3)). as well as schemes based on lossy identification schemes and pairings.

Authenticated Encryption Security. Ghoshal, Jaeger, and Tessaro [15] have recently observed that in the context of authenticated encryption (AE), it is difficult to lift confidentiality of the scheme, in terms of INDR security, to full AE security, when additionally assuming ciphertext integrity, if we want to do so in a memory-tight way. This is well motivated, as several works establish tight time-memory trade-offs for INDR security [21,17,12,9,20], which we would like to lift to their AE security. The difficulty in the proof is that the INDR reduction must simulate a decryption oracle which rejects all ciphertexts *except those forwarded from an encryption query*. Recognizing these forwarded ciphertexts seems to require remembering state.

[5] The suffix '1' indicates a variant of UFCMA security in which the adversary can only obtain a single signature per message. The security game always returning the same signature if the adversary repeats signature queries.

Assumption	Scheme	Result	Time	Memory	Advantage	New Technique
1UFCMA	Any	mxUFRMA	✓	✓	✓	Efficient tagging
	RDS	mUFCMA	✓	✓	✓	Efficient tagging
	Any	mwUFRMA	✗	✓	✓	Inefficient tagging
RSA	RSA-PFDH	mSUFCMA	✓	✓	✓	Efficient tagging
(PRF, INDR)	EtP	AE	(✓,✓)	(✗,✓)	(✓,✓)	Efficient tagging
1CCA	Any	mCCA	✗	✓	✗	Inefficient tagging
1$CCA	Any	m$CCA	✓	✓	✗	Message encoding

Fig. 1. Memory-tight reductions we provide. A 1 vs. an m prefix indicates whether one or many challenge queries are allowed. A ✓ vs. an ✗ indicates whether the reduction is tight with respect to that complexity metric. Reductions lacking tightness multiply running time/advantage by $O(q)$ or add $O(q)$ to the memory complexity, where q is the number of queries. An x vs. a w indicates whether the coins underlying the distribution of messages can be extracted from the message. RDS is randomization of any digital signature scheme by padding input messages with randomness. RSA-PFDH is probabilistic full-domain hash with RSA. EtP is the Encrypt-then-PRF AE construction.

Here, we give a different take and show that for specific schemes – in particular, those obtained by adding integrity via a PRF, following the lines of [19,3,18] – a memory-tight reduction *can* be given. Our INDR reduction is applied after arguing that the PRF looks like a random function f and thus forgeries are unlikely to occur. It uses f in a version of our efficient tagging technique to identify whether a ciphertext queried to decryption is fresh.[6]

Chosen ciphertext security: One to many. A classical textbook result for public-key encryption shows that CCA-security against a *single* encryption query (1CCA) implies security against *multiple* queries (mCCA), with a quantitative advantage loss accounting to the number of such queries. ACFK [1] claim, incorrectly, that the associated reduction from 1CCA to mCCA is easy to make memory-tight, but this appears to be an oversight: No such reduction is known, and here we use our techniques to recover a memory-tight version of this result.

Let us consider concretely the "left-or-right" formulation of 1CCA/mCCA-security: The reduction from 1CCA to mCCA, given an adversary $\mathcal{A}$, picks a random $i \leftarrow\!\!\$ [q]$ (where q is the number of encryption queries) and simulates the multi-query challenger to $\mathcal{A}$ by answering its first $i-1$ encryption queries with an encryption of the left message, whereas the last $q-i$ queries are answered by encrypting the right message. Only the answer to the i-th query is answered by the single-query challenger. A problem arises when simulating the decryption queries: Indeed, we need to guarantee that a decryption query for *any* of the challenge ciphertexts $c_1^*, \ldots, c_q^*$ returns an error $\perp$, yet this suggests that we seemingly need to remember the extra challenge ciphertexts c_j^* for $j \neq i$.

[6] Ghoshal et al. [15] in fact described three variants of AE with different conventions for how decryption responds to non-fresh queries. By our results, memory-tight reductions to INDR are possible for two of the three variants.

We will resolve this in two ways. First, we give a new memory-tight reduction using the inefficient tagging method, with the same advantage loss as the original textbook reduction. Our reduction is non-time-tight, so may not be suitable for all situations. The main idea here is that we use the *randomness* used to generate the challenge ciphertext as our tag.

To obtain a reduction which is also tight with respect to time, we resort to the observation that changing to a stronger (but still commonly achieved) definition of CCA-security allows for different memory-tight reductions. We give in particular a memory-tight *and* time-tight reduction (with the usual factor q advantage loss) from the notion of 1$CCA-$\mathtt{m}$ security to the notion of m$CCA-$\mathtt{m}$ security. These are variants of CCA security where (1) encryption queries are with respect to a *single* message, and return either the encryption of the message, or a random, independent ciphertext, and (2) decryption queries on a challenge ciphertext c_i^* returns the associated message.

Our reduction uses the full power of our message encoding approach, simulating random ciphertexts in a careful way which allows for recovering the associated challenge plaintext.

A FEW REMARKS. The above results on CCA security show us that the ability to give a memory-tight reduction is strongly coupled with definitional choices. In particular, different equivalent approaches to modeling the decryption oracle in the memory unbounded regime may not be equivalent in the memory-bounded setting. This means in particular that we need to exercise more care in choosing the right definition. We believe, for example, that the approach taken in m$CCA-$\mathtt{m}$ security is the more "natural" one (as it does not require artificially blocking the output of the decryption oracle, by always returning a message), but there may be contexts where other definitional choices are favored.

Another important lesson learnt from our AE result is that impossibility results, such as those in [1,22,16,15], do not preclude positive results in form of memory-tight reductions, either by leveraging the structure of specific schemes, or by considering restricted security notions.

1.3 Paper Outline

Section 2 introduces notation, our computational model, and basic cryptographic background. Section 3 discusses our convention of using $\mathcal{F}$-oracle adversaries. Section 4 gives our memory-tight reduction for digital signature schemes when many forgery attempts are allowed. In particular, the generic results are in Section 4.2, while the result specific to RSA-PFDH is in Section 4.5. Section 5 proves the security of Encrypt-then-PRF with a memory-tight reduction to the INDR security of the encryption scheme. Section 6 gives our results relating the one- and many-challenge query variants of CCA security. In particular, Section 6.1 gives our result for the traditional "left-vs.-right" notion and Section 6.2 gives our result for the "indistinguishable from random" variant. The full version of this paper [14] contains omitted proofs.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \dots\}$ and $[n] = \{1, \dots, n\}$ for $n \in \mathbb{N}$. If $x \in \{0,1\}^*$ is a string, then $|x|$ denotes its length in bits. If S is a set, then $|S|$ denotes its size. We let $x \,\|\, y \,\|\, \dots$ denote an encoding of the strings $x, y, \dots$ from which the constituent strings can be unambiguously recovered. We identify bitstrings with integers in the standard way.

FUNCTIONS. Let T be a set (called the tweak set) and for each $t \in T$ let D_t and R_t be sets. Then $\mathsf{Fcs}(T, D, R)$ denotes the set of all f such that for each $t \in T$, $f(t, \cdot)$ is a function from D_t to R_t. Similarly, $\mathsf{Inj}(T, D, R)$ denotes the set of all f such that for each $t \in T$, $f(t, \cdot)$ is an injection from D_t to R_t. When D_t or R_t are independent of the choice of t we may omit the subscript.

If $f \in \mathsf{Inj}(T, D, R)$, then its inverse f^{-1} is defined by $f^{-1}(t, f(t, x)) = x$ for all (t, x) and $f^{-1}(t, y) = \bot$ for $y \notin f(t, D_t)$. For such f we let $f^{\pm}$ denote the function defined by $f^{\pm}(+, x) = f(x)$ and $f^{\pm}(-, x) = f^{-1}(x)$. We let $\mathsf{Inj}^{\pm}(T, D, R) = \{f^{\pm} : f \in \mathsf{Inj}(T, D, R)\}$.

2.1 Computational Model

PSEUDOCODE. We regularly use pseudocode inspired by the code-based framework of [5]. We think of algorithms as randomized RAMs when not specified otherwise. If $\mathcal{A}$ is an algorithm, then $y \leftarrow \mathcal{A}^{\mathrm{O}_1, \cdots}(x_1, \dots; r)$ denotes running $\mathcal{A}$ on inputs $x_1, \dots$ with coins r and access to the oracles $\mathrm{O}_1, \dots$ to produce output y. When the coins are implicit we write $\leftarrow\!\!\$$ in place of $\leftarrow$ and omit r.

We let $x \leftarrow\!\!\$ \mathcal{D}$ denote sampling x according to the distribution $\mathcal{D}$. If $\mathcal{D}$ is a set, we overload notation and let $\mathcal{D}$ also denote the uniform distribution over elements of $\mathcal{D}$. The domain of $\mathcal{D}$ is denoted by $[\mathcal{D}]$.

Security notions are defined via games; for an example see Fig. 2. The probability that G outputs true is denoted $\Pr[\mathsf{G}]$. In proofs we sometimes define a sequence of "hybrid" games in one figure, using comments of the form "$//\mathsf{H}_{[i,j)}$." A line of code commented thusly is only included in the hybrids H_k for $i \leqslant k < j$. (We are of course referring only to values of $k \in \mathbb{N}$.) By this convention to identify the differences between H_{k-1} and H_k one looks for comments $\mathsf{H}_{[i,k)}$ (code no longer included in the k-th hybrid) and $\mathsf{H}_{[k,j)}$ (code new to the k-th hybrid).

We let $\bot$ be a special symbol used to indicate rejection. If we do not explicitly include $\bot$ in a set, then $\bot$ is not contained in that set. If $\bot$ is an input to a function or algorithm, then we assume its output is $\bot$. We do not distinguish between $\bot$ and tuples $(\bot, \dots, \bot)$. Algorithms cannot query $\bot$ to their oracles.

COMPLEXITY MEASURES. To measure the complexity of algorithms we follow the conventions of measuring their local complexity, not including the complexity of whatever oracles they interact with. Local complexity was preferred by Auerbach et al. [1] for analyzing memory-limited adversaries so that analysis can be agnostic to minor details of security definitions' implementations. We focus on worst-case runtime $\mathbf{Time}(\mathcal{A})$ and memory complexity $\mathbf{Mem}(\mathcal{A})$ (i.e. how many bits of state it stores for local computation). These exclude the internal

complexity of oracles queried by $\mathcal{A}$, but include the time and memory used to write the query and receive the response. If $\mathcal{A}$ expects access to n oracles then we let $\mathbf{Query}(\mathcal{A}) = (q_1, \ldots, q_n)$ where q_i is an upper bound on the number of queries to its i-th oracle. (Here we index from left to right, so for $\mathcal{A}^{\mathrm{O}_1,\ldots,\mathrm{O}_n}$ the i-th oracle is O_i.) If S is a scheme, then $\mathbf{Time}(\mathsf{S})$ and $\mathbf{Mem}(\mathsf{S})$ are the sums of the corresponding complexities over all of its algorithms. If G is a game, then we define $\mathbf{Time}(\mathsf{G})$ and $\mathbf{Mem}(\mathsf{G})$ to *exclude* the complexity of any adversaries embedded in the game.

2.2 Cryptographic Background

Ideal models. Some schemes we look at may be proven secure in ideal models (e.g. the random oracle or ideal cipher models). To capture this we can think of a scheme S as specifying a set of functions $\mathsf{S.I}$. At the beginning of a security game a function h will be sampled from this set. The adversary and all algorithms of S are given oracle access to h.

Function families. A family of functions F specifies, for each $K \in \mathsf{F.K}$, an efficiently computable function $\mathsf{F}_K \in \mathsf{F.F}$. We refer to $\mathsf{F.F}$ as the function space of F. Pseudorandom (PR) security of F is captured by the game defined in Fig. 2. It measures how F with a random key can be distinguished from a random function in $\mathsf{F.F}$ via oracle access. We define $\mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}) = \Pr[\mathsf{G}^{\mathsf{pr}}_{\mathsf{F},1}(\mathcal{A})] - \Pr[\mathsf{G}^{\mathsf{pr}}_{\mathsf{F},0}(\mathcal{A})]$. The standard notions of (tweakable) pseudorandom functions/injections/permutations or strong injections/permutations are captured by appropriate choices of $\mathsf{F.F}$.

Game $\mathsf{G}^{\mathsf{pr}}_{\mathsf{F},b}(\mathcal{A})$	$\underline{\mathrm{Ev}(x)}$
$\mathrm{h} \leftarrow\!\!\$\ \mathsf{F.I}$	$y_1 \leftarrow \mathsf{F}^{\mathrm{h}}_K(x)$
$K \leftarrow\!\!\$\ \mathsf{F.K}$	$y_0 \leftarrow f(x)$
$f \leftarrow\!\!\$\ \mathsf{F.F}$	Return y_b
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Ev},\mathrm{h}}$	
Return $b' = 1$	

Fig. 2. Security game capturing the pseudorandomness of function family F.

Switching lemma. We make use of the following standard result which bounds how well a random function and a random injection can be distinguished.

Lemma 1 (Switching Lemma). *Fix T, D, and R. Let $N = \min_{t \in T} |R_t|$. Then for any adversary $\mathcal{A}$ with $q = \mathbf{Query}(\mathcal{A})$ we have that*

$$|\Pr[\mathcal{A}^f \Rightarrow 1] - \Pr[\mathcal{A}^g \Rightarrow 1]| \leqslant 0 \cdot q^2/N.$$

The probabilities are measured over the coins of $\mathcal{A}$, the uniform choice of f from $\mathsf{Fcs}(T, D, R)$, and the uniform choice of g from $\mathsf{Inj}(T, D, R)$.

Recent papers [17,11,20] have given improved versions of the switching lemma for adversaries with bounded memory complexity, as long as it does not repeat oracle queries. In our application of the switching lemma the adversary's memory complexity is too large for these bounds to provide any improvement.

OTHER PRIMITIVES. We recall relevant syntax and security definitions for digital signatures, nonce-based encryption, and public key encryption schemes in the sections where we consider them (Sections 4, 5, and 6 respectively).

3 Adversaries With Access to Random Functions

This paper proposes and adopts what we consider to be a better formalism to deal with memory-tight reductions. Namely, all of our reductions will require access to some variety of large random functions which it will query on a small number of inputs (specifically uniformly random functions and invertible random injections). That is, our reduction adversaries can be written in the form shown of the left below, for some set of functions $\mathcal{F}$ and algorithm $\mathcal{A}_2$. (On the right is a pseudorandom version of $\mathcal{A}$ which we will discuss momentarily.)

Adversary $\mathcal{A}^{\mathrm{O}}(in)$	Adversary $\mathcal{A}_{\mathsf{F}}^{\mathrm{O}}(in)$
$f \leftarrow\!\!\$\ \mathcal{F}$	$K \leftarrow\!\!\$\ \mathsf{F.K}$
$out \leftarrow\!\!\$\ \mathcal{A}_2^{\mathrm{O},f}(in)$	$out \leftarrow\!\!\$\ \mathcal{A}_2^{\mathrm{O},\mathsf{F}_K}(in)$
Return out	Return out

We refer to such an $\mathcal{A}$ as an $\mathcal{F}$-oracle adversary. In this section we will generally discuss such adversaries, rather than separately providing the discussion for such adversaries each time we apply them.

The time and memory complexity of any $\mathcal{F}$-oracle adversary must include the complexity of sampling, storing, and evaluating f. This will be significant if $\mathcal{F}$ is large. However, as we will argue, this additional state and time should be assumed to not significantly increase the advantage of $\mathcal{A}$. As such, we will define the *reduced complexity* of $\mathcal{A}$ by

$$\mathbf{Time}^*(\mathcal{A}) = \mathbf{Time}(\mathcal{A}_2) \text{ and } \mathbf{Mem}^*(\mathcal{A}) = \mathbf{Mem}(\mathcal{A}_2).$$

Later we state theorems in terms of reduced complexity.

PSEUDORANDOM REPLACEMENT. The most conservative justification of $\mathcal{F}$-oracle adversaries is to bound how much the oracle can help by replacing it with a pseudorandom version. This was the approach taken by Auerbach et al. [1] when they used pseudorandom functions for purposes such as emulating random oracles and storing the coins required by an adversary with low memory, and has been adopted by follow-up work [22,7,10]. If F is a function family with $\mathsf{F.F} = \mathcal{F}$, then the adversary $\mathcal{A}_{\mathsf{F}}$ we gave above does exactly this. It replaces $\mathcal{A}_2$'s oracle access to f with access to F_K for a random K. The following lemma is straightforward.

Lemma 2. *Let $\mathcal{A}$ be an $\mathcal{F}$-oracle adversary for a game G. Then for any function family F with $\mathsf{F.F} = \mathcal{F}$ we can define a pseudorandomness adversary $\mathcal{A}_{\mathsf{i}}$ such that*

$$\Pr[\mathsf{G}(\mathcal{A})] \leqslant \Pr[\mathsf{G}(\mathcal{A}_{\mathsf{F}})] + \mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}_{\mathsf{i}}), \quad \mathbf{Time}(\mathcal{A}_{\mathsf{i}}) = \mathbf{Time}^*(\mathcal{A}) + \mathbf{Time}(\mathsf{G}(\mathcal{A})),$$
$$\mathbf{Query}(\mathcal{A}_{\mathsf{i}}) = q, \text{ and} \quad \mathbf{Mem}(\mathcal{A}_{\mathsf{i}}) = \mathbf{Mem}^*(\mathcal{A}) + \mathbf{Mem}(\mathsf{G}(\mathcal{A})).$$

Here q is an upper bound on the number of queries $\mathcal{A}_2$ makes to its second oracle.

Note that the complexity of $\mathcal{A}_{\mathsf{F}}$ is given by $\mathbf{Time}(\mathcal{A}_{\mathsf{F}}) = \mathbf{Time}^*(\mathcal{A}) + q \cdot \mathbf{Time}(\mathsf{F})$ and $\mathbf{Mem}(\mathcal{A}_{\mathsf{F}}) = \mathbf{Mem}^*(\mathcal{A}) + \mathbf{Mem}(\mathsf{F})$. Thus the existence of an appropriate pseudorandom F ensures that the memory and time complexity excluded by $\mathbf{Time}^*$ and $\mathbf{Mem}^*$ cannot significantly aid an adversary. In the use of this technique by Auerbach et al. [1] the reduction $\mathcal{A}_{\mathsf{i}}$ was memory-tight. Note this is not strictly necessary as long as we are willing to assume the existence of F with sufficient security as a function of attackers' time and query complexities without regard to memory complexity.

We could have combined Lemma 2 with any of our coming theorems to obtain bounds in terms of **Time** and **Mem**, rather than their reduced version. However we find the use of reduced complexity cleaner as it simplifies our theorems, allowing us to focus on the conceptual core of the proofs without having to repeat the rote step of replacing random objects with pseudorandom ones.

When combining the lemma with a theorem, game G would correspond to the security game played by the reduction adversary. For our theorems, that game will have low time and memory overhead over that of $\mathcal{A}$, so the application of the lemma would be time- and memory-tight. That said, the tightness of this is less important than the tightness of the other components of the theorem we would apply it to. Note that the definition of $\mathcal{A}_{\mathsf{i}}$ is *independent* of the choice of F. Consequently, we can always choose F with a very high security threshold to counteract any looseness in the lemma. In the full version [14], we summarize the $\mathcal{F}$ used in our theorems and how they could be pseudorandomly instantiated.

Assumed independence. As a second observation why the storage of f may not help $\mathcal{A}$, note that f is completely "independent" of the problem $\mathcal{A}$ is trying to solve (as specified by *in* and the behavior of O). In various settings it seems likely that such independent state does not help. For example, it would be very surprising (or even a breakthrough) to show a better factoring or lattice algorithm given access to a random function f from a natural set. Indeed, cryptanalytic work often makes use of random oracles without significant comment (from which other types of random functions can be constructed).

Information theoretic settings. In some information theoretic settings, the "independence" of f from the problem can be made rigorous. Information theoretic results are typically depending only on the query complexity of the attacker or its memory usage, *ignoring code size*. In such settings, we expect bounds of the form $\mathsf{Adv}(\mathcal{A}) \leqslant \epsilon(\mathbf{Mem}(\mathcal{A}), \mathbf{Query}(\mathcal{A}))$ for some function ϵ. Because this bound *does not* depend on the code size of $\mathcal{A}$, if $\mathcal{A}$ is an $\mathcal{F}$-oracle adversary we should be able to prove $\mathsf{Adv}(\mathcal{A}) \leqslant \epsilon(\mathbf{Mem}^*(\mathcal{A}), \mathbf{Query}(\mathcal{A}))$ by a coin-fixing argument in which we fix the random choice of function ahead of time and embed it in the description of the adversary. This is, for example, the case for the recent time-memory tradeoffs shown for distinguishing between a random function and a random injection without repeating queries [17,11,20]. A coin-fixing readily shows that these tradeoffs hold when using $\mathbf{Mem}^*(\mathcal{A})$ in place of $\mathbf{Mem}(\mathcal{A}), \mathbf{Query}(\mathcal{A})$.

4 Multi-challenge Security of Digital Signature Schemes

In the context of memory-tightness, the security of digital signature schemes has been considered in several works [1,22,10]. The standard security notion for signatures asks the attacker, given examples, to come up with a forged signature on a fresh message. A straightforward proof shows (in the standard setting where memory efficiency is not a concern) that the security notion is equivalent whether the attacker is allowed one or many forgery attempts. However, Auerbach et al. [1] proved an impossibility result showing that a (certain form of black-box) reduction cannot be time, memory, and advantage tight. The difficulty faced by the reduction is in distinguishing between when the adversary has produced a novel forgery and when it is simply repeating a signature that it was given.

In this section we show a few ways that security against many forgery attempts (i.e., multiple challenges) can be proven to follow from security against a single forgery (i.e., a single challenge) in a memory-tight manner. Our first results consider a variant definition of digital signature security we introduce (called UFRMA) in which the adversary has only partial control over the messages being signed. Using our new techniques, we show that single challenge UFCMA security implies multi-challenge UFRMA security in a memory-tight manner (for some practically relevant distributions over messages). We also consider the security of the RSA full domain hash digital signature scheme. Auerbach et al. [1] gave a memory-, but not advantage-tight proof of the security of the standard version of this scheme in the single challenge setting. By considering a probabilistic variant of the scheme introduced by Coron [8] we are able to provide a memory-, time-, and advantage-tight proof of the many-forgery SUFCMA security of the variant.

4.1 Syntax and Security

Digital signature syntax. A digital signature scheme DS specifies a key generation algorithm $\mathsf{DS.K}$, a signing algorithm $\mathsf{DS.Sign}$, and a verification algorithm $\mathsf{DS.Ver}$. The syntaxes of these algorithms are shown in Fig. 3. We capture ideals models by providing $\mathsf{DS.Sign}$ and $\mathsf{DS.Ver}$ with oracle access to a function h drawn at random from the set $\mathsf{DS.I}$. When relevant we let $\mathsf{DS.M}$ denote the set of messages it accepts. The verification and signing keys are respectively denoted by vk and sk. The message to be signed is m, the signature produced is σ, and the decision is $d \in \{\mathsf{true}, \mathsf{false}\}$. Correctness requires $\mathsf{DS.Ver}^{\mathrm{h}}(vk, m, \sigma) = \mathsf{true}$ for all $\mathrm{h} \in \mathsf{DS.I}$, all $(vk, sk) \in [\mathsf{DS.K}]$, all $m \in \mathsf{DS.M}$, and all $\sigma \in [\mathsf{DS.Sign}^{\mathrm{h}}(sk, m)]$.

$\underline{\mathsf{DS}\text{ Syntax}}$
$\mathrm{h} \leftarrow\!\!\$\ \mathsf{DS.I}$
$(vk, sk) \leftarrow\!\!\$\ \mathsf{DS.K}$
$\sigma \leftarrow\!\!\$\ \mathsf{DS.Sign}^{\mathrm{h}}(sk, m)$
$d \leftarrow \mathsf{DS.Ver}^{\mathrm{h}}(vk, m, \sigma)$

Fig. 3. Syntax of digital signature scheme.

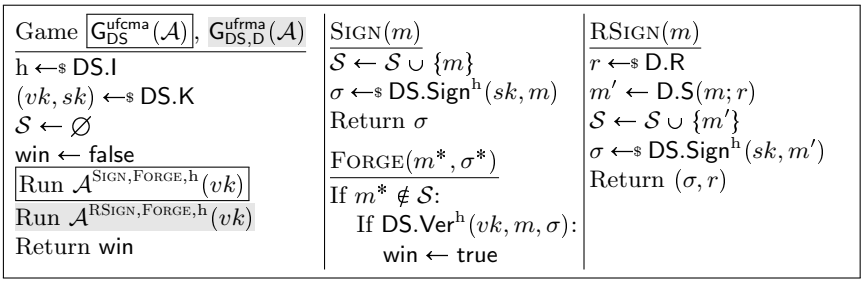

Fig. 4. Security games capturing the unforgeability of a digital signature scheme.

Message distribution syntax. One of the security notions we consider for digital signature schemes will be parameterized by a message distribution via which the adversary is given incomplete control over the messages which are signed. A message distribution D specifies sampling algorithm $\mathsf{D.S}$ which samples an output message m' based on parameters m given as input (written $m' \leftarrow\!\!\$\ \mathsf{D.S}(m)$). The parameters m must be drawn from a set $\mathsf{D.M}$, which we typically leave implicit. When making the randomness of the sampling algorithm explicit we let $\mathsf{D.R}$ be the set from which its randomness is drawn and write $m' \leftarrow \mathsf{D.S}(m; r)$. If there exists an extraction algorithm $\mathsf{D.X}$ such that $\mathsf{D.X}(\mathsf{D.S}(m; r)) = (m, r)$ for all m, r then we say D is *extractable*. If $\mathsf{D.X}(\mathsf{D.S}(m; r)) = m$ for all m, r then we say D is *weakly extractable*. We assume that $\mathsf{D.X}(m') = \perp$ if $m' \neq \mathsf{D.S}(m; r)$ for all m, r. We define the min-entropy of D as

$$\mathsf{D.H}_{\infty} = -\lg \max_{m} \Pr[r \leftarrow\!\!\$\ \mathsf{D.R} : \mathsf{D.S}(m; r) = m'] \,.$$

Unforgeability security. The unforgeability security notions we consider are defined in Fig. 4. The standard notion of UFCMA (unforgeability under chosen message attack) security is captured by $\mathsf{G}^{\mathsf{ufcma}}$ which includes the boxed but not the highlighted code, giving the adversary access to a regular signing oracle Sign. The goal of the adversary is to query Forge with a valid signature σ^* of a message m^* which was not previously included in a signing query (as stored by the set $\mathcal{S}$). We define $\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}) = \Pr[\mathsf{G}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A})]$.

Our new security notion UFRMA (unforgeability under randomized message attack) is captured by the game $\mathsf{G}^{\mathsf{ufrma}}$ which is parameterized by a message distribution D. In this game the adversary is instead given access to the randomized signing oracle RSign where the message to be signed is chosen by D. Note that the coins used by D are returned to the adversary along with the signature. Otherwise this game matches that of UFCMA security. We define $\mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A}) = \Pr[\mathsf{G}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A})]$.

We will relate the advantage of attacks making only a single forgery attempt and those making many such attempts. When wanting to make the distinction explicit we prefix the abbreviation of a security notion with an 'm' or '1'. Strong

UFCMA security, denoted SUFCMA, is captured by modifying $\mathsf{G}^{\mathsf{ufcma}}$ to store the tuple (σ, m) in $\mathcal{S}$ in Sign and checking $(m^*, \sigma^*) \notin \mathcal{S}$ in Forge. We denote this by $\mathsf{G}^{\mathsf{sufcma}}$ and the corresponding advantage by $\mathsf{Adv}^{\mathsf{sufcma}}$. We define SUFRMA, $\mathsf{G}^{\mathsf{sufrma}}$, and $\mathsf{Adv}^{\mathsf{sufrma}}$ analogously. We write xUFRMA when assuming that D is extractable and wUFRMA when assuming it is weakly extractable.

4.2 Multi-Challenge Security for Extractable Message Distributions

The first applications we show for our techniques are generic methods of tightly implying security of a digital signature scheme against multiple forgery attempts (i.e., multi-challenge security). Recall that Auerbach et al. [1] gave a lower bound showing that a black-box reduction proving that single UFCMA security implies many UFCMA cannot be made memory-tight and time-tight. We avoid this in two ways; first by considering mUFRMA, rather than mUFCMA, security and then by considering a particular choice of digital signature scheme.

High-level idea. The primary difficulty of a tight proof that 1UFCMA security implies mUFCMA security is that a successful mUFCMA attacker may have made many Forge queries which verify correctly, one of which is a valid forgery and the rest of which were just forwarded from its Sign oracle. A 1UFCMA reduction must then somehow be able to identify which of the queries is the true forgery so it can forward this to its own Forge oracle.

The technical core of the coming proof for mUFRMA is that our reduction adversary will use the random coins of the message distribution D to signal things to its future self. In particular, when $\mathcal{A}_{\mathsf{r}}$ makes a query RSign(m), the reduction will choose coins for $\mathsf{D.S}$ via $r \leftarrow f(m, i)$ where i is a counter which is incremented with each query and f is a random tweakable function/injection. The coins then act as a sort of authentication tag for m. On a later Forge(m^*, σ^*) query, if $m^* = \mathsf{D.S}(m; r)$ where $r = f(m, i)$ for some $i \in [q_{\text{Sign}}]$ the reduction can safely assume this message was signed by an earlier RSign query.

When D is fully extractable, we can perform the requisite check for Forge by having f be an injection. We extract m and r from m^* and then compute $i \leftarrow f^{-1}(m, r)$. This is the strategy used in Theorem 1. If we assume only that D is weakly extractable, we can extract m if D has a sufficient amount of entropy, and then individually check if $\mathsf{D.S}(m; f(m, i))$ holds for each choice of i. This reduction strategyobtains the same advantage at the cost of an extra runtime being needed to iterate over the possible choices of i in Forge.

Extractable Message Distribution. If the message distribution D is extractable, the following theorem captures that 1UFCMA security tightly implies mUFRMA security. The proof makes use of our efficient tagging technique.

Theorem 1 (1UFCMA $\Rightarrow$ mxUFRMA). *Let* DS *be a digital signature scheme and* D *be an extractable message distribution. Let* $\mathcal{A}_{\mathsf{r}}$ *be an adversary with* $(q_{\text{Sign}}, q_{\text{Forge}}, q_{\text{h}}) = \mathbf{Query}(\mathcal{A}_{\mathsf{r}})$ *and assume* $q_{\mathsf{Sign}} \leqslant 0.5|\mathsf{D.R}|$. *Let* $\mathcal{A}_{\mathsf{u}}$ *be the*

Adversary $\mathcal{A}_{\mathsf{u}}^{\text{Sign},\text{Forge},\mathsf{h}}(vk)$	SimRSign(m)	SimForge(m^*, σ^*)
$i \leftarrow 0$	$i \leftarrow i + 1$	$(m, r) \leftarrow \mathsf{D.X}(m^*)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(\mathsf{DS.M}, [q_{\text{Sign}}], \mathsf{D.R})$	$r \leftarrow f(m, i)$	If $f^{-1}(m, r) \notin [q_{\text{Sign}}]$:
Run $\mathcal{A}_{\mathsf{r}}^{\text{SimRSign},\text{SimForge},\mathsf{h}}(vk)$	$m' \leftarrow \mathsf{D.S}(m; r)$	If $\mathsf{DS.Ver}^{\mathsf{h}}(vk, m^*, \sigma^*)$:
	$\sigma \leftarrow \text{Sign}(m')$	Query Forge(m^*, σ^*)
	Return (σ, r)	Halt execution

Fig. 5. Adversary $\mathcal{A}_{\mathsf{u}}$ used in proof of Theorem 1.

$\mathsf{Inj}^{\pm}(\mathsf{DS.M}, [q_{\text{Sign}}], \mathsf{D.R})$*-oracle adversary shown in Fig. 5. Then,*

$$\mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A}_{\mathsf{r}}) \leqslant \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}}) + (0.5 \cdot q_{\text{Sign}}^2 + 2 \cdot q_{\text{Sign}} \cdot q_{\text{Forge}})/|\mathsf{D.R}|$$
$$\mathbf{Query}(\mathcal{A}_{\mathsf{u}}) = (q_{\text{Sign}}, 1, q_{\mathsf{h}} + q_{\text{Forge}} \cdot \mathbf{Query}(\mathsf{DS}))$$
$$\mathbf{Time}^*(\mathcal{A}_{\mathsf{u}}) = \mathbf{Time}(\mathcal{A}_{\mathsf{r}}) + q_{\text{Sign}} \cdot \mathbf{Time}(\mathsf{D}) + q_{\text{Forge}}(\mathbf{Time}(\mathsf{D}) + \mathbf{Time}(\mathsf{DS}))$$
$$\mathbf{Mem}^*(\mathcal{A}_{\mathsf{u}}) = \mathbf{Mem}(\mathcal{A}_{\mathsf{r}}) + \mathbf{Mem}(\mathsf{D}) + \mathbf{Mem}(\mathsf{DS}) + \lg(q_{\text{Sign}}).$$

This is time-tight because $\mathbf{Time}(\mathcal{A}_{\mathsf{r}}) \in \Omega(q_{\mathsf{Sign}} + q_{\text{Forge}})$ must hold and $\mathbf{Time}(\mathsf{D})$ and $\mathbf{Time}(\mathsf{DS})$ will be small. This is memory-tight because $\mathbf{Mem}(\mathsf{D})$, $\mathbf{Mem}(\mathsf{DS})$, and $\lg(q_{\text{Sign}})$ will be small.

The main idea of $\mathcal{A}_{\mathsf{u}}$ is using the output of an invertible random injection f on the message and a counter as coins instead of sampling them uniformly at random when answering RSign queries. Since D is fully extractable, during a Forge query on m^*, we can extract $(m, r) \leftarrow \mathsf{D.X}(m^*)$ and use the fact that f is invertible to compute $f^{-1}(m, r)$ and check if the index is in $[q_{\text{Sign}}]$. This is used to avoid remembering $\mathcal{S}$. If $m^* \in \mathcal{S}$, and $(m, r) \leftarrow \mathsf{D.X}(m^*)$, then there exists $j \in [q_{\text{Sign}}]$ such that $r = f(m, j)$ – so the check passes. We can argue that if $m^* \notin \mathcal{S}$, our check is unlikely to pass. We give the formal proof of this theorem in Section 4.3. It applies the switching lemma to argue the use of f cannot be distinguished from honestly sampling r with advantage better than $0.5 \cdot q_{\text{Sign}}^2/|\mathsf{D.R}|$ and shows that the probability of falsely making the check pass is bounded by $2q_{\text{Sign}}q_{\text{Forge}}/|\mathsf{D.R}|$.

We would not be able to use the technique in this proof to prove mxSUFRMA from 1SUFCMA in a memory-tight way. In particular, since the coins r of the message distribution are chosen before σ is known, our trick of using r to signal freshness of a forgery query does not work for a message-signature pair.

4.3 Proof of Theorem 1 (1UFCMA⇒mUFRMA)

Proof. We consider a sequence of hybrids H_0 through H_4 defined in Fig. 6. When examining these hybrids recall our conventions regarding "$//\mathsf{H}_{[i,j)}$" comments described in Sec. 2.1. Of these hybrids we will make the following claims, which establish the upper bound on the advantage of $\mathcal{A}_{\mathsf{r}}$ claimed in the proof.

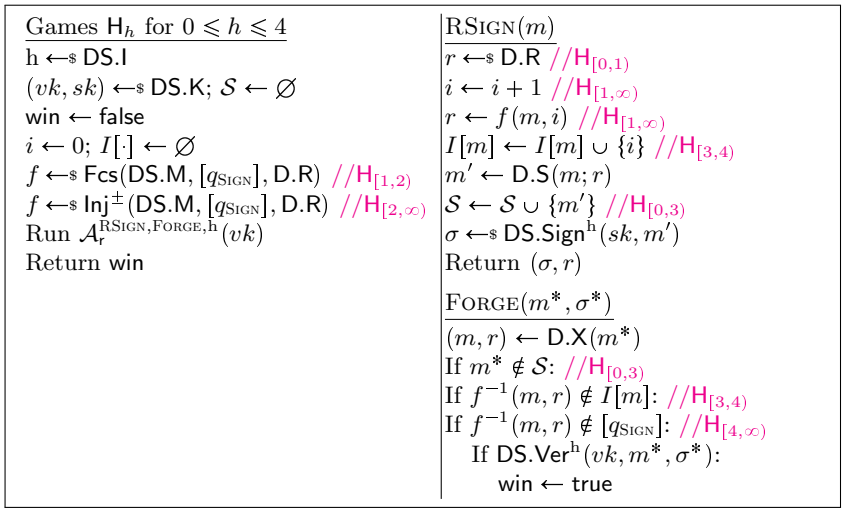

Fig. 6. Hybrid games used in proof of Theorem 1.

1. $\Pr[\mathsf{G}^{\mathsf{ufrma}}_{\mathsf{DS,D}}(\mathcal{A}_{\mathsf{r}})] = \Pr[\mathsf{H}_0] = \Pr[\mathsf{H}_1]$
2. $\Pr[\mathsf{H}_1] \leqslant \Pr[\mathsf{H}_2] + 0.5 \cdot q^2_{\mathrm{SIGN}}/|\mathsf{D.R}|$
3. $\Pr[\mathsf{H}_2] = \Pr[\mathsf{H}_3]$
4. $\Pr[\mathsf{H}_3] \leqslant \Pr[\mathsf{H}_4] + 2q_{\mathrm{SIGN}}q_{\mathrm{FORGE}}/|\mathsf{D.R}|$
5. $\Pr[\mathsf{H}_4] = \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}})$

TRANSITION H_0 TO H_1. The hybrid H_0 is simply a copy of the game $\mathsf{G}^{\mathsf{ufrma}}$. (We also added code to initialize variables i and $I[\cdot]$ that will be used in later hybrids.) Hence $\Pr[\mathsf{G}^{\mathsf{ufrma}}(\mathcal{A}_{\mathsf{r}})] = \Pr[\mathsf{H}_0]$. In hybrid H_1, we replace the random sampling of r for D in RSIGN with the output of a random function f applied to m, using a counter i to provide domain separation between different queries. This method of choosing r is equivalent, so $\Pr[\mathsf{H}_0] = \Pr[\mathsf{H}_1]$.

TRANSITION H_1 TO H_2. In hybrid H_2 we replace the random function with a random injection. This modifies the behavior of the game only in that values of r are guaranteed not to repeat across different signing queries that used the same message. There are at most q_{SIGN} invocations of f, so the switching lemma (Lemma 1) tells us that $\Pr[\mathsf{H}_1] \leqslant \Pr[\mathsf{H}_2] + 0.5 \cdot q^2_{\mathrm{SIGN}}/|\mathsf{D.R}|$.

TRANSITION H_2 TO H_3. In hybrid H_3, we replace the check whether $m^* \notin \mathcal{S}$ in oracle FORGE with a check if $f^{-1}(m, r) \notin I[m]$ where $(m, r) = \mathsf{D.X}(m^*)$. Here $I[\cdot]$ is a new table introduced into the game. In RSIGN, code was added which uses $I[m]$ to store each of the counter values for which $\mathcal{A}_{\mathsf{r}}$ made a signing query for m. Hence $f^{-1}(m, r)$ will be in $I[m]$ iff m^* is in $\mathcal{S}$ and so $\Pr[\mathsf{H}_2] = \Pr[\mathsf{H}_3]$.

TRANSITION H_3 TO H_4. In the final transition to hybrid H_4 we replace the FORGE check $f^{-1}(m, r) \notin I[m]$ with $f^{-1}(m, r) \notin [q_{\mathrm{SIGN}}]$. This *does* change behavior if $\mathcal{A}_{\mathsf{r}}$ ever makes a successful forgery query for $m^* = \mathsf{D.S}(m; f(m, i))$ without its i-th signing query having used the message m. This would require

guessing $f(m,i)$ for some $i \in [q_{\text{SIGN}}] \backslash I[m]$. We can bound the probability of this ever occurring by a union bound over the FORGE queries made by $\mathcal{A}_{\mathsf{r}}$. Consider the set $f(m,[q_{\text{SIGN}}] \backslash I[m]) = \{f(m,i) : i \in [q_{\text{SIGN}}] \backslash I[m]\}$. It has size at most q_{SIGN}. Because f is a random injection it is uniform subset of the set $\mathsf{D.R} \backslash f(m, I[m])$ (which has size at least $|\mathsf{D.R}| - q_{\text{SIGN}}$). Hence the probability of any particular query triggering this different behavior is at most $q_{\text{SIGN}}/(|\mathsf{D.R}| - q_{\text{SIGN}}) \leqslant 2q_{\text{SIGN}}/|\mathsf{D.R}|$. Applying the union bound gives us $\Pr[\mathsf{H}_3] \leqslant \Pr[\mathsf{H}_4] + 2q_{\text{SIGN}} \cdot q_{\text{FORGE}}/|\mathsf{D.R}|$.

REDUCTION TO UFCMA. We complete the proof using adversary $\mathcal{A}_{\mathsf{u}}$ from Fig. 5 which simulates hybrid H_4 and succeeds whenever $\mathcal{A}_{\mathsf{r}}$ would. The adversary $\mathcal{A}_{\mathsf{u}}$ samples f at random from $\mathsf{Inj}(\mathsf{DS.M}, [q_{\text{SIGN}}], \mathsf{D.R})$. When run on input vk, it runs $\mathcal{A}_{\mathsf{r}}$ on the same input. It gives $\mathcal{A}_{\mathsf{r}}$ direct access to h. To simulate a query $\text{RSIGN}(m)$, it computes $m' \leftarrow \mathsf{D.S}(m; f(m,i))$, increments i, and queries $\text{SIGN}(m')$, returning the result to $\mathcal{A}_{\mathsf{r}}$. On a query $\text{FORGE}(m^*, \sigma^*)$, it computes $(m, r) \leftarrow \mathsf{D.X}(m^*)$. If $f^{-1}(r) \notin [q_{\text{SIGN}}]$ and $\mathsf{DS.Ver}(vk, m^*, \sigma^*) = \mathsf{true}$ then it queries its own oracle with (m^*, σ^*) and halts. Otherwise it ignores the query.

If adversary $\mathcal{A}_{\mathsf{u}}$ ever makes a FORGE query, it will succeed. It ensured that (m^*, σ^*) is verified correctly and $f^{-1}(r) \notin [q_{\text{SIGN}}]$ ensures that it is has not previously made a SIGN query for m^*. If $\mathcal{A}_{\mathsf{r}}$ would have succeeded in hybrid H_4, its winning query will cause $\mathcal{A}_{\mathsf{u}}$ to make a FORGE query. Hence, we have $\Pr[\mathsf{H}_4] = \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}})$.

4.4 Applications and Weakly Extractable Variant

We discuss some applications of Theorem 1. This includes scenarios where extractable message distributions are used and proving security of digital signature schemes when their messages are padded with randomness. Additionally, we give a variant of the theorem when the underlying message distribution is only weakly extractable. The resulting reduction is memory- but not time-tight.

EXAMPLE EXTRACTABLE DISTRIBUTIONS. The simplest extractable distribution does not accept parameters as input and simply outputs its randomness as the message. Security with respect to this is the standard notion of security against random message attacks which was originally introduced by Even, Goldreich, and Micali [13].

Extractable distributions arise naturally when the messages being signed include random values. For example, protocols often include random nonces in messages that are signed. In TLS 1.3, for example, when the server is responding to the Client Hello Message it signs a transcript of the conversation up until that point which includes a 256-bit nonce just chosen by the server. We could think of the security for this setting being captured by an extractable distribution D_{tls} that takes as input message parameter m that specifies all of the transcript other than the nonce and sets the nonce to its randomness $r \in \{0,1\}^{256}$.

PADDING SCHEMES WITH RANDOMNESS. Using Theorem 1, we can see that augmenting any digital signature scheme by appending auxiliary randomness will

give us a memory-tight reduction from the mUFCMA security of the augmented scheme to the 1UFCMA security of the original scheme.

Let DS be a digital signature scheme and R be a set. We define $\mathsf{RDS}[\mathsf{DS},\mathsf{R}]$ by having $\mathsf{RDS}[\mathsf{DS},\mathsf{R}].\mathsf{Sign}(sk,m)$ do "$r \leftarrow\!\!\$\ \mathsf{R}$; Return $\mathsf{DS}.\mathsf{Sign}(sk, m \,\|\, r) \,\|\, r$" and having $\mathsf{RDS}[\mathsf{DS},\mathsf{R}].\mathsf{Ver}(vk,m,\sigma')$ do "$\sigma \,\|\, r \leftarrow \sigma'$; Return $\mathsf{DS}.\mathsf{Ver}(vk, m \,\|\, r, \sigma)$." We also define a related message distribution $\mathsf{RD}[\mathsf{R}]$ by $\mathsf{RD}[\mathsf{R}].\mathsf{R} = \mathsf{R}$ and $\mathsf{RD}[\mathsf{R}].\mathsf{S}(m;r) = m \,\|\, r$. Clearly it is extractable.

The following reduces the mUFCMA security of RDS to the mUFRMA security of DS. Theorem 1 can in turn be used to reduce this to the 1UFCMA security of DS. It also relates the mSUFCMA security of RDS to the mSUFRMA security of DS. We note this because if DS has unique signatures, then its mSUFRMA and mUFRMA security are identical and hence UFCMA security of DS implies mSUFCMA security of RDS in a memory-tight way.

Theorem 2. *Let* DS *be a digital signature scheme and* R *be a set. Then for any* $\mathcal{A}_{\mathsf{u}}$ *we can construct* $\mathcal{A}_{\mathsf{r}}$ *such that* $\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{RDS}[\mathsf{DS},\mathsf{R}]}(\mathcal{A}_{\mathsf{u}}) = \mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{RD}[\mathsf{R}]}(\mathcal{A}_{\mathsf{r}})$*. It additionally holds that* $\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{RDS}[\mathsf{DS},\mathsf{R}]}(\mathcal{A}_{\mathsf{u}}) = \mathsf{Adv}^{\mathsf{sufrma}}_{\mathsf{DS},\mathsf{RD}[\mathsf{R}]}(\mathcal{A}_{\mathsf{r}})$*. Adversary* $\mathcal{A}_{\mathsf{r}}$ *has essentially the same complexity as* $\mathcal{A}_{\mathsf{u}}$*.*

Proof (Sketch). The proof of this is straightforward. If $\mathcal{A}_{\mathsf{u}}$ queries $\textsc{Sign}(m)$, then $\mathcal{A}_{\mathsf{r}}$ queries $\textsc{Sign}(m)$ and receives (σ, r) and returns$\sigma \,\|\, r$ to $\mathcal{A}_{\mathsf{u}}$. If $\mathcal{A}_{\mathsf{u}}$ queries $\textsc{Forge}(m^*, \sigma^* \,\|\, r^*)$, then $\mathcal{A}_{\mathsf{r}}$ queries $\textsc{Forge}(m^* \,\|\, r^*, \sigma^*)$. Note that $\mathcal{A}_{\mathsf{r}}$ wins whenever $\mathcal{A}_{\mathsf{u}}$ would. □

In independent and concurrent work, Diemert, Gellert, Jager, and Lyu [10] also considered RDS, proving that if DS can be proven SUFCMA1 secure (in this notion the game records its responses to signature queries and repeats them if the adversary repeats a query) with a restricted class of "canonical" memory-tight reductions, then there is a memory-tight reduction for the mSUFCMA security of RDS. This complements our results as they use a more restrictive assumption to prove mSUFCMA while we use a generic assumption to prove mUFCMA.

In the full version [14], we further show that if D is only *weakly* extractable (but still has high entropy), then we can prove a variant of Theorem 1 with a less efficient reduction. In particular, the running time of the reduction has an additional term of $q_{\textsc{Forge}} \cdot q_{\textsc{Sign}} \cdot \mathbf{Time}(\mathsf{D.S})$. This difference arises because rather than extracting r and computing $j \leftarrow f^{-1}(m,r)$ in $\textsc{Forge}$ we instead need to iterate over the possible values of $f(m,j)$ to check for consistency. Thus the proof for this is an instance of our inefficient tagging technique.

4.5 mSUFCMA Security of RSA-PFDH

The RSA-based Probabilistic Full-Domain Hash (RSA-PFDH) scheme, originally introduced by Coron [8], can be viewed as the result of applying the $\mathsf{RDS}[\cdot]$ transform to RSA-based Full-Domain Hash [4] (RSA-FDH). Auerbach et al. [1] gave a memory-, but not time-tight reduction from the 1UFCMA security of

RSA-FDH to the one-wayness of RSA. Applying Theorems 1 and 2 would give a memory-, but not time-tight reduction for the security of RSA-PFDH.

By a careful combination of our tagging technique with the proof ideas of Coron and of Auerbach et al. we can analyze RSA-PFDH directly. We prove the following result – a time, memory, and advantage tight reduction for the security of RSA-PFDH. The theorem is properly formalized and proven in the full version [14].

Theorem 3 (Informal, mSUFCMA security of RSA-PFDH). *Define* $\mathsf{RSA} := \mathsf{RDS}[\text{RSA-FDH}, \{0,1\}^{\mathsf{rl}}]$. *Let* $\mathcal{A}_{\mathsf{m}}$ *be an adversary with* $(q_{\text{SIGN}}, q_{\text{FORGE}}, q_{\text{h}}) = \mathbf{Query}(\mathcal{A}_{\mathsf{m}})$. *Then we can construct an adversary* $\mathcal{A}_{\mathsf{RSA}}$ *against one-wayness of* rl*-bit RSA such that*

$$\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{RSA}}(\mathcal{A}_{\mathsf{m}}) \leqslant \mathsf{Adv}^{\mathsf{ow\text{-}rsa}}(\mathcal{A}_{\mathsf{RSA}}) + (0.5 \cdot q^2_{\text{SIGN}} + 2 \cdot q_{\text{SIGN}} \cdot q_{\text{FORGE}})/2^{\mathsf{rl}}$$

The running time and memory of $\mathcal{A}_{\mathsf{RSA}}$ *is roughly the same as* $\mathcal{A}_{\mathsf{m}}$.

In concurrent work, Diemert, Gellert, Jager, and Lyu [10] also give a time, memory, and advantage tight reduction for RSA-PFDH via a different proof.

5 AE Security of Encrypt-then-PRF

For nonce-based secret-key encryption schemes, we often want Authenticated Encryption (AE) security which simultaneously asks for confidentiality and ciphertext integrity. The common approach to prove AE security of a nonce-based encryption scheme is to give separate reductions to the indistinguishability of its ciphertexts from truly random ones (INDR security) and its ciphertext integrity. Ghoshal et al. [15] proved an impossibility result showing that a (certain form of black-box) reduction from AE security to INDR security and ciphertext integrity cannot be memory-tight. Making the INDR part memory-tight is of particular interest because of results which establish tight time-memory trade-offs for INDR security [21,17,12,9,20].

In this section we look at a particular scheme which we refer to as Encrypt-then-PRF. Given a nonce-based encryption scheme NE that only has INDR security, one generic way to construct a new encryption scheme NE' which also achieves ciphertext integrity is to use a PRF and let the ciphertext of NE' be the concatenation of the ciphertext of NE and a tag which is the evaluation of the PRF on the ciphertext and the nonce.

We show that in the context of Encrypt-then-PRF, for two of the notions of AE security introduced in [15], we can give a memory-tight reduction to the INDR security of the underlying encryption scheme and a non-memory-tight reduction to the security of the PRF. This shows that we can bypass the generic impossibility result of [15] if we consider specific constructions of nonce-based authenticated encryption schemes. In more detail, the impossibility result of [15] rules out lifting the INDR security of a scheme to full AE security in a memory

Game $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}(\mathcal{A})$	$\underline{\textsc{Enc}_b(n,m)}$	$\underline{\textsc{Dec}^w_b(n,c)}$
$K \leftarrow\!\!\$\ \mathsf{NE.K}$	$c_1 \leftarrow \mathsf{NE.E}(K,n,m)$	If $M[n,c] \neq \bot$:
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b}$	$c_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE.cl}(\lvert m\rvert)}$	Return $M[n,c]$ if $w = \mathtt{m}$
Return $b' = 1$	$M[n,c_b] \leftarrow m$	Return $\diamond$ if $w = \diamond$
Game $\mathsf{G}^{\mathsf{ae}\text{-}w}_{\mathsf{NE},b}(\mathcal{A})$	Return c_b	Return $\bot$ if $w = \bot$
$K \leftarrow\!\!\$\ \mathsf{NE.K}$		$m_1 \leftarrow \mathsf{NE.D}(k,n,c)$
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b,\textsc{Dec}^w_b}$		$m_0 \leftarrow \bot$
Return $b' = 1$		Return m_b

Fig. 7. Games defining INDR and AE-w security of NE for $w \in \{\mathtt{m}, \diamond, \bot\}$.

tight way, when additionally assuming ciphertext integrity *for a generic scheme*. Here, we show that for the *specific* case of Encrypt-then-PRF schemes, lifting the INDR security of the encryption scheme to full AE security of Encrypt-then-PRF is possible in a memory-tight way, assuming security of the PRF.

5.1 Syntax and Security Definitions

Nonce-Based Encryption. A nonce-based (secret-key) encryption scheme NE specifies algorithms $\mathsf{NE.K}$, $\mathsf{NE.E}$, and $\mathsf{NE.D}$. It specifies message space $\mathsf{NE.M}$ and nonce space $\mathsf{NE.N}$. The syntax of the algorithms is shown in Fig. 8. The secret key is denoted by K, the message is m, the nonce is n, and the ciphertext is c. The decryption algorithm may return $m = \bot$ to indicate rejection of the ciphertext. Correctness requires for all $K \in [\mathsf{NE.K}]$, $n \in \mathsf{NE.N}$, and $m \in \mathsf{NE.M}$ that $\mathsf{NE.D}(K, n, \mathsf{NE.E}(K,n,m)) = m$. We assume there is a ciphertext-length function $\mathsf{NE.cl} : \mathbb{N} \to \mathbb{N}$ such that for all $K \in [\mathsf{NE.K}]$, $n \in \mathsf{NE.N}$, and $m \in \mathsf{NE.M}$ we have $|c| = \mathsf{NE.cl}(|m|)$ where $c \leftarrow \mathsf{NE.E}(K,n,m)$. We define $\mathsf{NE.C} = \bigcup_{m \in \mathsf{NE.M}} \{0,1\}^{\mathsf{NE.cl}(|m|)}$. Typically, a nonce-based encryption scheme also takes associated data as input which is authenticated during encryption. This does not meaningfully affect our proof, so we omit it for simplicity.

NE Syntax
$K \leftarrow\!\!\$\ \mathsf{NE.K}$
$c \leftarrow \mathsf{NE.E}(K,n,m)$
$m \leftarrow \mathsf{NE.D}(K,n,c)$

Fig. 8. Syntax of (nonce-based) secret-key encryption scheme.

Encrypt-then-PRF. In this section we consider the Encrypt-then-PRF construction of a nonce-based encryption scheme, due to Rogaway [19]. Namprempre et al. [18] gave a more extensive exploration of the many ways to construct an AEAD encryption scheme via generic composition. Given nonce-based encryption scheme NE and function family F, we define $\mathsf{EtP}[\mathsf{NE},\mathsf{F}]$ by the following algorithms. We refer to the t component of the ciphertext returned by $\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{E}$ as the "tag" below. When including associated data, it would be input to F.

$\underline{\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{K}}$	$\underline{\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{E}(K,n,m)}$	$\underline{\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{D}(K,n,c)}$
$K \leftarrow\!\!\$\ \mathsf{NE.K}$	$(K,K') \leftarrow K$	$(K,K') \leftarrow K;\ (c',t) \leftarrow c$
$K' \leftarrow\!\!\$\ \mathsf{F.K}$	$c' \leftarrow \mathsf{NE.E}(K,n,m)$	If $t = \mathsf{F}_{K'}(n,c')$:
Return (K,K')	$t \leftarrow \mathsf{F}_{K'}(n,c')$	Return $\mathsf{NE.D}(K,n,m)$
	Return (c',t)	Return $\perp$

Our security result will analyze the authenticated security of EtP assuming NE has ciphertexts indistinguishable from random ciphertexts and F is pseudorandom. Let us recall these security notions.

Indistinguishability From Random (INDR) Security. This security notion requires that ciphertexts output by the encryption scheme cannot be distinguished from random strings. Consider the game $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}$ defined in Fig. 7. Here an adversary $\mathcal{A}$ is given access to an encryption oracle $\textsc{Enc}_b$ to which it can query a pair (n,m) and receive an honest encryption of message m with nonce n if $b = 1$ or a random string of the appropriate length if $b = 0$. We restrict attention to "valid" adversaries that never repeat the nonce n across different encryption queries. We define $\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{A}) = \Pr[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{A})] - \Pr[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{A})]$.

Authenticated Encryption (AE) security. AE security simultaneously asks for integrity and confidentiality. Consider the games $\mathsf{G}^{\mathsf{ae}\text{-}w}_{\mathsf{NE},b}$ which defines three variants of authenticated encryption security parameterized by $w \in \{\mathtt{m}, \diamond, \perp\}$ shown in Fig. 7. In this game, the adversary is given access to an encryption oracle and a decryption oracle. Its goal is to distinguish between a "real" and "ideal" world. In the real world ($b = 1$) the oracles use NE to encrypt messages and decrypt ciphertexts. In the ideal world ($b = 0$) encryption returns random messages of the appropriate length and decryption returns $\perp$. For simplicity, we will again restrict attention nonce-respecting adversaries which do not repeat nonces across encryption queries. (Note that there is no restriction placed on nonces used for decryption queries.)

The decryption oracle is parameterized by the value $w \in \{\mathtt{m}, \diamond, \perp\}$ corresponding to three different security notions. In all three, we use a table $M[\cdot,\cdot]$ to detect when the adversary forwards encryption queries on to its decryption oracle. When $w = \mathtt{m}$, the decryption oracle returns $M[n,c]$. When $w = \diamond$, it returns a special symbol $\diamond$. When $w = \perp$, it returns the symbol $\perp$ which is also used by the encryption scheme to represent rejection. For $w \in \{\mathtt{m}, \diamond, \perp\}$ we define the advantage of an adversary $\mathcal{A}$ by $\mathsf{Adv}^{\mathsf{ae}\text{-}w}_{\mathsf{NE}}(\mathcal{A}) = \Pr[\mathsf{G}^{\mathsf{ae}\text{-}w}_{\mathsf{NE},1}(\mathcal{A})] - \Pr[\mathsf{G}^{\mathsf{ae}\text{-}w}_{\mathsf{NE},0}(\mathcal{A})]$.

Discussion of variants. This choice of considering three variants of the definition follows the same choice made by Ghoshal et al. [15]. First off, we note that *if there are no restrictions on the memory of the adversary, all the three definitions are tightly equivalent.* An adversary can simply remember its past encryption queries and answers, and without loss of generality never make a decryption query on the answer of an encryption query. In the memory restricted setting these definitions *no longer appear to be equivalent.* The only known implication is that $w = \diamond$ security tightly implies $w = \perp$ security. Other implications seem to require remembering all encryption queries to properly simulate the decryption oracle. In Sec. 6 we parameterize public-key encryption CCA definitions similarly. This discussion applies to those definitions as well.

Ghoshal et al. argued that $w = \mathtt{m}$ is the "correct" definition. They argue that chosen ciphertext security is intended to capture the power of an adversary that can observe the behavior of a decrypting party. Both the $w = \perp$ and $w = \diamond$ definitions restrict what the adversary learns about this behavior when honestly generated ciphertexts are forwarded, which does not seem to model anything about real use of encryption. The $w = \mathtt{m}$ definition avoids this unnatural restriction.

We provide some technical context for this philosophical argument. In the full version [14] we give memory-tight proofs for the security of encryption schemes constructed with the KEM/DEM paradigm with $w = \mathtt{m}$ and noting this does not seem possible for the other choices of w. In this section and Sec. 6 we prove the AE/CCA-w security of encryption schemes for differing choices of w. We view this as a general exploration of what results are possible with memory-tight proofs. A proof which works for some w, but not others helps build some understanding of how these notions related.

5.2 Security Result

Now we give a proof of the AE-$\diamond$ security of $\mathsf{EtP}[\mathsf{NE}, \mathsf{F}]$. In particular we provide a memory-tight reduction to the INDR security of NE and a non-memory-tight reduction to the security of F. Such a result is useful if a time-memory tradeoff is known for NE and F is sufficiently secure even against high-memory attackers.

Theorem 4 (Security of EtP). *Let* NE *be a nonce-based encryption scheme and* F *be a family of function with* $\mathsf{F.F} = \mathsf{Fcs}(\mathsf{NE.N}, \mathsf{NE.C}, \{0,1\}^\tau)$ *for* $\tau \in \mathbb{N}$. *Let* $\mathcal{A}_a$ *be an* AE-$\diamond$ *adversary with* $(q_{\text{ENC}}, q_{\text{DEC}}) = \mathbf{Query}(\mathcal{A}_a)$. *Define adversaries* $\mathcal{A}_\mathsf{p}$ *and* $\mathcal{A}_r$ *as shown in Fig. 9. Then,*

$$\mathsf{Adv}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}]}(\mathcal{A}_a) \leqslant \mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}_\mathsf{p}) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{A}_r) + 2q_{\text{DEC}}/2^\tau$$

$$\begin{aligned}
&\mathbf{Query}(\mathcal{A}_\mathsf{p}) = q_{\text{ENC}} + q_{\text{DEC}} && \mathbf{Query}(\mathcal{A}_r) = q_{\text{ENC}} \\
&\mathbf{Time}(\mathcal{A}_\mathsf{p}) = \mathbf{Time}(\mathsf{G}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}]}(\mathcal{A}_a)) && \mathbf{Time}^*(\mathcal{A}_r) = \mathbf{Time}(\mathcal{A}_a) \\
&\mathbf{Mem}(\mathcal{A}_\mathsf{p}) = \mathbf{Mem}(\mathsf{G}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}]}(\mathcal{A}_a)) && \mathbf{Mem}^*(\mathcal{A}_r) = \mathbf{Mem}(\mathcal{A}_a).
\end{aligned}$$

Adversary $\mathcal{A}_r$ *is an* $\mathsf{F.F}$*-oracle adversary.*

The standard (not memory-tight) proof of the security of EtP begins identically to our proof; we start in $\mathsf{G}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],1}$ replace the use of F with a truly random function (using $\mathcal{A}_\mathsf{p}$) and then information theoretically argue that the attacker shall be incapable of creating any forgeries. In the standard proof we would transition to a game where the decryption oracle is exactly that of $\textsc{Dec}^\diamond_0$, i.e. it always returns $\perp$ when $M[n, c] = \perp$. Then we reduce to the security of NE to replace the generated ciphertexts with random. However this standard reduction will not be memory-tight because the attacker must store the table $M[\cdot, \cdot]$ to know whether it should return $\diamond$ or $\perp$ when simulating decryption queries.[7]

[7] Note this *would* be memory-tight for AE-$\perp$ security.

Adversary $\mathcal{A}_{\mathsf{p}}^{\mathrm{Ev}}$	$\underline{\textsc{SimEnc}(n,m)}$	$\underline{\textsc{SimDec}(n,c)}$
$K \leftarrow\!\!\$\ \mathsf{NE.K}$	$c' \leftarrow \mathsf{NE.E}(K,n,m)$	If $M[n,c] \neq \bot$: Return $\diamond$
$b' \leftarrow \mathcal{A}_a^{\textsc{SimEnc},\textsc{SimDec}}$	$t \leftarrow \textsc{Ev}(n,c')$	$(t,c') \leftarrow c$
Return b'	$c \leftarrow (t,c')$	If $t = \textsc{Ev}(n,c')$:
	$M[n,c] \leftarrow m$	Return $\mathsf{NE.D}(k,n,c')$
	Return c	Return $\bot$

Adversary $\mathcal{A}_r^{\mathrm{Enc}}$	$\underline{\textsc{SimEnc}(n,m)}$	$\underline{\textsc{SimDec}(n,c)}$
$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathsf{NE.N},\mathsf{NE.M},\{0,1\}^\tau)$	$c' \leftarrow \textsc{Enc}(n,m)$	$(t,c') \leftarrow c$
$b' \leftarrow \mathcal{A}_a^{\textsc{SimEnc},\textsc{SimDec}}$	$t \leftarrow f(n,c')$	If $t = f(n,c')$:
Return b'	$c \leftarrow (t,c')$	Return $\diamond$
	Return c	Return $\bot$

Fig. 9. Adversaries used for proof of Theorem 4.

Instead we first transition to a world where F has been replaced by the random function f and $\textsc{Dec}$ always returns $\diamond$ when given a ciphertext with a correct tag. (Which we can do because either $M[n,c] \neq \bot$ held or the attacker managed to guess a random tag, which is unlikely.) Now we can make our INDR reduction memory-tight. It forwards encryption queries to its encryption oracle and then uses its own function f to create the tag. For decryption queries it checks $f(n,c') = t$, returning $\diamond$ if so and $\bot$ otherwise. Then we can finally conclude by switching to the decryption oracle $\textsc{Dec}_0^{\diamond}$ by arguing that noticing this change requires guessing a random tag. The full proof is given in the full version [14].

It does not seem possible to extend this proof technique to AE-$\mathtt{m}$ security because the tag would be too short to embed values of m we need to remember.

6 Chosen Ciphertext Security of Public Key Encryption

Now we apply our techniques to give memory-tight reductions between single- and multi-challenge notions of chosen-ciphertext security. The standard reduction bounds the advantage of an adversary making $q_{\textsc{Enc}}$ encryption queries by $q_{\textsc{Enc}}$ times the advantage of an adversary making 1 query. The reduction requires memory linear in $q_{\textsc{Enc}}$ and so is not memory-tight.[8] In Section 6.1, we consider the most common "left-or-right" definition of CCA security and introduce three different variants (mirroring the three notions for AE security in Section 5).

[8] Auerbach et al. [1] stated that this reduction is memory-tight for both CPA and CCA security. While the former is correct, the latter depends on the definition of CCA. In personally communication with Auerbach et al. [2], they concurred that their claim was incorrect for their intended definition of CCA security ($w = \diamond$) but pointed out that it does work for an "exclusion" variant, $w = E$, which we discuss in the full version [14].

Game $\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},b}(\mathcal{A})$	$\textsc{Enc}_b(m_0, m_1)$	$\textsc{Dec}^w(c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	// $\lvert m_0\rvert = \lvert m_1\rvert$	If $M[c] \neq \bot$:
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b, \textsc{Dec}}(ek)$	$c_0 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m_0)$	Return $M[c]$ if $w = \mathtt{m}$
Return $b' = 1$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m_1)$	Return $\diamond$ if $w = \diamond$
	$M[c_b] \leftarrow m_1$	Return $\bot$ if $w = \bot$
	Return c_b	$m \leftarrow \mathsf{PKE.D}(dk, c)$
		Return m

Fig. 11. Game defining CCA-w security of PKE for $w \in \{\mathtt{m}, \diamond, \bot\}$.

We give a memory-tight reduction between single- and multi-challenge security for two of the three variants ($\diamond$ and $\bot$), but the reduction is not time-tight. In Section 6.2, we look at the CCA security variant that requires ciphertexts be indistinguishable from random. We give a memory-tight and time-tight reduction between single- and multi-challenge security for all three variants of this notion.

Public key encryption. A public key encryption scheme PKE specifies algorithms PKE.K, PKE.E, and PKE.D. The syntax of these algorithms is shown in Fig. 10. The key generation algorithm PKE.K returns encryption key ek and decryption key dk. The encryption algorithm PKE.E encrypts message m with ek to produce a ciphertext c. We write $\mathsf{PKE.E}(ek, m; r)$ when making random coins $r \in \mathsf{PKE.R}$ explicit. The decryption algorithm decrypts c with dk to produce m. The decryption algorithm may output $m = \bot$ to indicate rejection.

PKE Syntax
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$
$c \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$
$m \leftarrow \mathsf{PKE.D}(dk, c)$

Fig. 10. Syntax of a public key encryption scheme PKE.

Correctness requires that $\mathsf{PKE.D}(dk, c) = m$ for all $(ek, dk) \in [\mathsf{PKE.K}]$, all m, and all $c \in [\mathsf{PKE.E}(ek, m)]$. We define the min-entropy of PKE as

$$\mathsf{PKE.H}_\infty = -\lg \max_{m,ek,c} \Pr[r \leftarrow\!\!\$\ \mathsf{PKE.R} : \mathsf{PKE.E}(ek, m; r) = c] \ .$$

6.1 Left-or-right CCA Security of PKE

Left-or-right CCA security. In this section, we consider the left-or-right definition of CCA-security most commonly used in the literature. For $w \in \{\mathtt{m}, \diamond, \bot\}$ we denote this as CCA-w[9] and the corresponding security game $\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},b}$ is defined in Fig. 11. The adversary gets the encryption key ek and has access to an encryption and a decryption oracle. The encryption oracle takes in messages m_0 and m_1 and encrypts m_b where b is the secret bit. The decryption oracle returns

[9] The discussion in Section 5 about the choice to have three variants of the definitions is applicable here as well.

Adversary $\mathcal{A}_1^{\text{Enc},\text{Dec}}(ek)$	$\text{SimEnc}(m_0, m_1)$	$\text{SimDec}(c)$
$k \leftarrow\!\!\$\ [q_{\text{Enc}}]$	$i \leftarrow i + 1$	If $c = c^*$: Return $\diamond$
$i \leftarrow 0$	For $d \in \{0,1\}$:	$m \leftarrow \text{Dec}(c)$
$D_{(\cdot)} \leftarrow \{0,1\}^{(\cdot)} \times [q_{\text{Enc}}]$	$r_d \leftarrow f(\lvert m_d\rvert, (m_d, i))$	If $m = \bot$: Return $\bot$
$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$	$c_d \leftarrow \mathsf{PKE.E}(ek, m_d; r_d)$	For $j \in [i]$ do:
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\text{SimEnc},\text{SimDec}}(ek)$	If $i < k$: $c \leftarrow c_1$	If $m \in \{m_0^*, m_1^*\}$ and $j = k$:
Return b'	If $i = k$:	Skip to next j
	$c \leftarrow \text{Enc}(m_0, m_1)$	$r \leftarrow f(\lvert m\rvert, (m, j))$
	$c^* \leftarrow c$	If $\mathsf{PKE.E}(ek, m; r) = c$:
	$(m_0^*, m_1^*) \leftarrow (m_0, m_1)$	Return $\diamond$
	If $i > k$: $c \leftarrow c_0$	Return m
	Return c	

Fig. 12. Adversary $\mathcal{A}_1$ for Theorem 5.

the decryption of a ciphertext, unless the ciphertext was previously returned by an encryption query. This is tracked by table M. When $w = \mathtt{m}$, the decryption oracle returns $M[c]$ which is m_1 from the earlier encryption query. When $w = \diamond$, it returns $\diamond$. When $w = \bot$, it returns $\bot$ which is also used by the encryption scheme to represent rejection. The advantage of an adversary $\mathcal{A}$ against the CCA-w security of PKE is defined as $\mathsf{Adv}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE}}(\mathcal{A}) = \Pr[\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},1}(\mathcal{A})] - \Pr[\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},0}(\mathcal{A})]$.

The goal of this section is to relate the advantage of attacks making only a single encryption query and those making many such queries. When wanting to make the distinction explicit we may use the adjectives "many" and "single" or prefix the abbreviation of a security notion with an 'm' or '1'.

1CCA-$\diamond$ IMPLIES mCCA-$\diamond$. The following theorem gives a memory-tight reduction establishing that CCA-$\diamond$ security against adversaries making one encryption query implies security for an arbitrary number of queries. The proof makes use of our inefficient tagging technique. The reduction performs a hybrid over the encryption queries of the original adversary and is thus not advantage-tight.

Theorem 5 (1CCA-$\diamond \Rightarrow$ mCCA-$\diamond$)**.** *Let* PKE *be a public key encryption scheme. Let* $\mathcal{A}_{\mathsf{m}}$ *be an adversary with* $(q_{\text{Enc}}, q_{\text{Dec}}) = \mathbf{Query}(\mathcal{A}_{\mathsf{m}})$*. Define* $D_{(\cdot)}$ *by* $D_n = \{0,1\}^n \times [q_{\text{Enc}}]$*. Let* $\mathcal{A}_1$ *be the* $\mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$*-oracle adversary shown in Fig. 12. Then,*

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) \leqslant q_{\text{Enc}} \cdot \mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_1) + 4 \cdot q_{\text{Enc}} \cdot q_{\text{Dec}} / 2^{\mathsf{PKE.H}_\infty} \\
&\mathbf{Query}(\mathcal{A}_1) = (1, q_{\text{Dec}}) \\
&\mathbf{Time}^*(\mathcal{A}_1) = O(\mathbf{Time}(\mathcal{A}_{\mathsf{m}})) + q_{\text{Enc}}(q_{\text{Dec}} + 1)\mathbf{Time}(\mathsf{PKE}) \\
&\mathbf{Mem}^*(\mathcal{A}_1) = O(\mathbf{Mem}(\mathcal{A}_{\mathsf{m}})) + \mathbf{Mem}(\mathsf{PKE}) + \lg q_{\text{Enc}}.
\end{aligned}$$

The standard (non-memory-tight) reduction against 1CCA security picks an index $k \in [q_{\text{Enc}}]$ where q_{Enc} is the number of encryption queries made by $\mathcal{A}_{\mathsf{m}}$. It runs $\mathcal{A}_{\mathsf{m}}$, simulating encryption queries as follows. For the first $k-1$ encryption

Game $\mathsf{G}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE},b}(\mathcal{A})$	$\textsc{Enc}_b(m)$	$\textsc{Dec}^w(c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	If $M[c] \neq \bot$:
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b, \textsc{Dec}^w}(ek)$	$c_0 \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \lvert m \rvert)$	Return $M[c]$ if $w = \mathtt{m}$
Return $b' = 1$	$M[c_b] \leftarrow m$	Return $\diamond$ if $w = \diamond$
	Return c_b	Return $\bot$ if $w = \bot$
		$m \leftarrow \mathsf{PKE.D}(dk, c)$
		Return m

Fig. 13. Game defining \$CCA-w security of PKE for $w \in \{\mathtt{m}, \diamond, \bot\}$.

queries, it answers with an encryption of m_1, for the k-th encryption query it forwards the query to its own encryption oracle, and the rest of the queries it answers with an encryption of m_0. To answer the decryption queries, the reduction returns $\diamond$ if it was ever queried the ciphertext for a previous encryption query. Otherwise, it forwards the query to its own decryption oracle. Finally, the reduction adversary outputs whatever $\mathcal{A}_{\mathtt{m}}$ outputs. Standard hybrid analysis shows that if the advantage of $\mathcal{A}_{\mathtt{m}}$ is ϵ, then the advantage of the reduction adversary is $\epsilon/q_{\textsc{Enc}}$. Simulating decryption queries required remembering all prior encryption queries and hence the reduction is not memory-tight.

We give an adversary $\mathcal{A}_1$ in Fig. 12 that is very similar to the reduction just described, but avoids remembering prior encryption queries. The main idea is that it picks the coins when encrypting m_0 or m_1 locally as the output of a random function f applied to the message and a counter. This allows $\mathcal{A}_1$ to detect whether a ciphertext c queried to the decryption oracle is one it answered to an earlier encryption query as follows: it first asks for the decryption of c from its own decryption oracle and receives m. Then it iterates over all counter values for which encryption queries have been made so far and checks if c was the encryption of m using the output of f on m and the counter as coins. If any of these checks succeed it returns $\diamond$, otherwise it returns m. If c was the answer of an encryption query $\mathcal{A}_1$ detects it successfully. The probability that $\mathcal{A}_1$ returns $\diamond$ for a decryption query when it should not is small. We give the formal proof of Theorem 5 in the full version [14] where we use a sequence of hybrid games to transition from $\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},b}$ to a hybrid game that is simulated by $\mathcal{A}_1$.

Notice that the additional memory overhead for $\mathcal{A}_1$ is just that required to store a counter, run PKE.E, and store (c^*, m_0^*, m_1^*). However, there is an *increase* in runtime by $q_{\textsc{Enc}} \cdot q_{\textsc{Dec}} \cdot \mathbf{Time}(\mathsf{PKE})$ because of the iteration over the counters to answer decryption queries. As discussed in the introduction, such reductions may be useful when the best attack for the underlying problem with low memory requires significantly more running time than the best attack with high memory.

The same proof strategy would work essentially unchanged for CCA-$\bot$. For CCA-$\mathtt{m}$, the strategy does not suffice. If the adversary queries the decryption oracle on a ciphertext c which was an answer to a previous query for (m_0, m_1) the oracle needs to return m_1 even if c is an encryption of m_0.

6.2 Indistinguishable from Random CCA Security of PKE

We saw in the previous section that we could have a memory-tight reduction from mCCA-$\diamond$ to 1CCA-$\diamond$; however, the reduction is not tight with respect to running time. In this section, we show that for a different formalization of CCA security, we can indeed have a memory-tight and time-tight reduction between many- and single-challenge variants.

Ciphertext and encryption key space. Before describing the indistinguishable from random formalization of CCA security, we need to make some assumptions on PKE. We define the encryption keyspace by $\mathsf{PKE.Ek} = \{ek : (ek, dk) \in \mathsf{PKE.K}\}$. We assume for each $ek \in \mathsf{PKE.Ek}$ and allowed message length $n \in \mathbb{N}$ there is a set $\mathsf{PKE.C}(ek, n)$ such that $\mathsf{PKE.E}(ek, m; r) \in \mathsf{PKE.C}(ek, |m|)$ always holds. Let $\mathsf{PKE.C}^{-1}(ek, c)$ returns n such that $c \in \mathsf{PKE.C}(ek, n)$. Correctness implies that $\mathsf{PKE.C}(ek, n)$ and $\mathsf{PKE.C}(ek, n')$ are disjoint for $n \neq n'$.

Indistinguishable from random ciphertext CCA security. The security notion we will consider in this section is captured by the game $\mathsf{G}^{\$\mathsf{cca}\text{-}w}$ shown in Fig. 13. It requires that ciphertexts output by the encryption scheme cannot be distinguished from ciphertexts chosen at random even given access to a decryption oracle. The adversary gets the encryption key ek and has access to an encryption oracle Enc and a decryption oracle Dec. The adversary needs to distinguish the following real and ideal worlds: in the real world, a query to Enc with a message m returns an encryption of m under ek, while in the ideal world, the same query returns a uniformly random element of $\mathsf{PKE.C}(ek, |m|)$. The decryption oracle $\textsc{Dec}^w$ acts exactly as the corresponding oracle in $\mathsf{G}^{\mathsf{cca}\text{-}w}$.[10] The advantage of an adversary $\mathcal{A}$ against the \$CCA-w security of PKE is defined as $\mathsf{Adv}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE}}(\mathcal{A}) = \Pr[\mathsf{G}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE},1}(\mathcal{A})] - \Pr[\mathsf{G}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE},0}(\mathcal{A})]$.

1\$CCA-$\mathtt{m}$ implies m\$CCA-$\mathtt{m}$. The following theorem captures a memory-tight reduction establishing that 1\$CCA-$\mathtt{m}$ security implies m\$CCA-$\mathtt{m}$ security. The proof makes use of our message encoding technique.

Theorem 6 (1\$CCA-$\mathtt{m}$ $\Rightarrow$ m\$CCA-$\mathtt{m}$). *Let PKE be a public key encryption scheme. Let τ satisfy $|\mathsf{PKE.C}(ek, n)| \geqslant 2^n \cdot 2^\tau$ for all n, ek. Let $\mathcal{A}_\mathsf{m}$ be an adversary with $(q_{\textsc{Enc}}, q_{\textsc{Dec}}, q_\mathrm{h}) = \mathbf{Query}(\mathcal{A}_\mathsf{m})$ and assume $q_{\textsc{Enc}} + q_{\textsc{Dec}} \leqslant 0.5 \cdot 2^\tau$. Let $\mathcal{F} = \mathsf{Inj}^{\pm}(T, D, R)$ where T, D, and R are defined by $T = \mathbb{N} \times \mathsf{PKE.Ek}$, $D_{n,ek} = \{0,1\}^n \times [q_{\textsc{Enc}}]$ and $R_{n,ek} = \mathsf{PKE.C}(ek, n)$. Let $\mathcal{A}_1$ be the $\mathcal{F}$-oracle adversary defined in Fig. 14. Then,*

$$\begin{aligned}
&\mathsf{Adv}^{\$\mathsf{cca}\text{-}m}_{\mathsf{PKE}}(\mathcal{A}_\mathsf{m}) \leqslant q_{\textsc{Enc}} \cdot \mathsf{Adv}^{\$\mathsf{cca}\text{-}m}_{\mathsf{PKE}}(\mathcal{A}_1) + 8 q_{\textsc{Enc}} q_{\textsc{Dec}} / 2^\tau + 5 q_{\textsc{Enc}}^2 / 2^\tau \\
&\mathbf{Query}(\mathcal{A}_1) = (1, q_{\textsc{Dec}}, q_\mathrm{h}) \\
&\mathbf{Time}^*(\mathcal{A}_1) = O(\mathbf{Time}(\mathcal{A}_\mathsf{m})) + q_{\textsc{Enc}} \mathbf{Time}(\mathsf{PKE}) \\
&\mathbf{Mem}^*(\mathcal{A}_1) = O(\mathbf{Mem}(\mathcal{A}_\mathsf{m})) + \mathbf{Mem}(\mathsf{PKE}) \lg q_{\textsc{Enc}}.
\end{aligned}$$

[10] As mentioned, the discussion in Section 5 about the three variants definitions is applicable here as well. In the full version [14] we give an example where we can prove CCA security of a KEM/DEM scheme in the memory restricted setting, but only if we use the $w = \mathtt{m}$ definition.

Adversary $\mathcal{A}_1^{\text{Enc},\text{Dec}}(ek)$	$\text{SimEnc}(m)$	$\text{SimDec}(c)$
$//0 \leqslant h \leqslant 2$	$i \leftarrow i + 1$	If $c = c^*$: Return m^*
$k \leftarrow\!\!\$\ [q_{\text{Enc}}]$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	$n \leftarrow \mathsf{PKE.C}^{-1}(ek, c)$
$i \leftarrow 0$	$c_0 \leftarrow f((\lvert m\rvert, ek), (m, i))$	$(m, j) \leftarrow f^{-1}((n, ek), c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$	If $i < k$: $c \leftarrow c_1$	If $m \neq \perp$ and $k \leqslant j \leqslant i$:
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\text{SimEnc},\text{SimDec}}(ek)$	If $i = k$:	If $(m, j) = (m^*, k)$:
Return b'	$c \leftarrow \text{Enc}(m)$	Skip next line
	$(c^*, m^*) \leftarrow (c, m)$	Return m
	If $i > k$: $c \leftarrow c_0$	$m \leftarrow \text{Dec}(c)$
	Return c	Return m

Fig. 14. Adversary $\mathcal{A}_1$ for Theorem 6.

The standard (non-memory-tight) reduction against 1$CCA security that runs an m$CCA adversary $\mathcal{A}_{\mathsf{m}}$ works in a similar manner as the standard reduction from an 1CCA adversary and an mCCA adversary that we described in Section 6.1. Again here, simulating decryption queries requires remembering all the answers of the encryption queries, and hence the reduction is not memory-tight.

We give an adversary $\mathcal{A}_1$ in Fig. 14 that is very similar to the standard reduction, but avoids remembering all the answers of the encryption queries. The main idea here is picking the ciphertext c_0 as the output of a random injective function f evaluated on the message and a counter, instead of sampling it uniformly at random. This way of picking the c_0 allows $\mathcal{A}_1$ detect whether a ciphertext c queried to the decryption oracle was the answer to an earlier encryption query as follows: it first checks if the inverse of f on the ciphertext is defined (i.e., not $\perp$), it returns the message part of the inverse. Otherwise it asks for the decryption of the ciphertext to its own decryption oracle and returns the answer. Using our assumption on the size of $\mathsf{PKE.C}(ek, n)$, we can argue that except with small probability, $\mathcal{A}_1$ simulates the decryption oracle correctly. We give the formal proof in the full version [14] where we use a sequence of hybrid games to transition from $\mathsf{G}^{\$\text{cca-m}}_{\mathsf{PKE},b}$ to a game that is perfectly simulated by $\mathcal{A}_1$. The additional memory overhead for $\mathcal{A}_1$ is only a counter. Moreover, there is no increase in the running time of $\mathcal{A}_1$ unlike the adversary in Theorem 5.

Extension to $CCA-$\diamond$, $CCA-$\perp$. We can prove the same result for $CCA-$\diamond$, $CCA-$\perp$ but the adversary would not be tight with respect to running time. The adversary in these cases would pick the coins for encrypting m (to compute c_1) like the adversary in Theorem 5. This would require iterating over counters to answer decryption queries and hence lead to looseness with respect to running time. We omit the theorems for these notions because they would not involve any new ideas beyond those presented in Theorems 5 and 6.

Acknowledgements

Ashrujit Ghoshal, Joseph Jaeger, and Stefano Tessaro were partially supported by NSF grants CNS-1930117 (CAREER), CNS-1926324, CNS-2026774, a Sloan Research Fellowship, and a JP Morgan Faculty Award. Joseph Jaeger's work was done while at the University of Washington.

References

1. Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 101–132. Springer, Heidelberg, August 2017. 2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 18, 23
2. Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Personal communication, 2021. 23
3. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008. 6
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 5, 18
5. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. 8
6. Daniel J Bernstein. Extending the salsa20 nonce. In *Workshop record of Symmetric Key Encryption Workshop*, volume 2011. Citeseer, 2011. 4
7. Rishiraj Bhattacharyya. Memory-tight reductions for practical key encapsulation mechanisms. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 249–278. Springer, Heidelberg, May 2020. 2, 10
8. Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 272–287. Springer, Heidelberg, April / May 2002. 5, 12, 18
9. Wei Dai, Stefano Tessaro, and Xihu Zhang. Super-linear time-memory trade-offs for symmetric encryption. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 335–365. Springer, Heidelberg, November 2020. 2, 5, 19
10. Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. Digital signatures with memory-tight security in the multi-challenge setting. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 403–433. Springer, Cham., December 2021. 2, 5, 10, 12, 18, 19
11. Itai Dinur. On the streaming indistinguishability of a random permutation and a random function. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 433–460. Springer, Heidelberg, May 2020. 2, 9, 11

12. Itai Dinur. Tight time-space lower bounds for finding multiple collision pairs and their applications. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 405–434. Springer, Heidelberg, May 2020. 2, 5, 19
13. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, March 1996. 17
14. Ashrujit Ghoshal, Riddhi Ghosal, Joseph Jaeger, and Stefano Tessaro. Hiding in plain sight: Memory-tight proofs via randomness programming. Cryptology ePrint Archive, Report 2021/1409, 2021. https://eprint.iacr.org/2021/1409. 7, 11, 18, 19, 22, 23, 26, 27, 28
15. Ashrujit Ghoshal, Joseph Jaeger, and Stefano Tessaro. The memory-tightness of authenticated encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 127–156. Springer, Heidelberg, August 2020. 2, 5, 6, 7, 19, 21
16. Ashrujit Ghoshal and Stefano Tessaro. On the memory-tightness of hashed ElGamal. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 33–62. Springer, Heidelberg, May 2020. 2, 7
17. Joseph Jaeger and Stefano Tessaro. Tight time-memory trade-offs for symmetric encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 467–497. Springer, Heidelberg, May 2019. 2, 5, 9, 11, 19
18. Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014. 6, 20
19. Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004. 6, 20
20. Ido Shahaf, Or Ordentlich, and Gil Segev. An information-theoretic proof of the streaming switching lemma for symmetric encryption. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 858–863, 2020. 2, 5, 9, 11, 19
21. Stefano Tessaro and Aishwarya Thiruvengadam. Provable time-memory trade-offs: Symmetric cryptography against memory-bounded adversaries. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 3–32. Springer, Heidelberg, November 2018. 2, 5, 19
22. Yuyu Wang, Takahiro Matsuda, Goichiro Hanaoka, and Keisuke Tanaka. Memory lower bounds of reductions revisited. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 61–90. Springer, Heidelberg, April / May 2018. 2, 7, 10, 12