A preliminary version of this paper is published in the proceedings of the 16th International Conference on Applied Cryptography and Network Security—ACNS 2018. This is the full version.

Continuously Non-Malleable Codes with Split-State Refresh

Antonio Faonio[1], Jesper Buus Nielsen[2], Mark Simkin[2], and Daniele Venturi[3]

[1]*IMDEA Software Institute, Madrid, Spain*
[2]*Aarhus University, Aarhus, Denmark*
[3]*Sapienza University of Rome, Rome, Italy*

June 15, 2018

Abstract

Non-malleable codes for the split-state model allow to encode a message into two parts, such that arbitrary independent tampering on each part, and subsequent decoding of the corresponding modified codeword, yields either the same as the original message, or a completely unrelated value. Continuously non-malleable codes further allow to tolerate an unbounded (polynomial) number of tampering attempts, until a decoding error happens. The drawback is that, after an error happens, the system must self-destruct and stop working, otherwise generic attacks become possible.

In this paper we propose a solution to this limitation, by leveraging a split-state refreshing procedure. Namely, whenever a decoding error happens, the two parts of an encoding can be locally refreshed (i.e., without any interaction), which allows to avoid the self-destruct mechanism in some applications. Additionally, the refreshing procedure can be exploited in order to obtain security against continual leakage attacks. We give an abstract framework for building refreshable continuously non-malleable codes in the common reference string model, and provide a concrete instantiation based on the external Diffie-Hellman assumption.

Finally, we explore applications in which our notion turns out to be essential. The first application is a signature scheme tolerating an arbitrary polynomial number of split-state tampering attempts, without requiring a self-destruct capability, and in a model where refreshing of the memory happens only after an invalid output is produced. This circumvents an impossibility result from a recent work by Fuijisaki and Xagawa (Asiacrypt 2016). The second application is a compiler for tamper-resilient read-only RAM programs. In comparison to other tamper-resilient RAM compilers, ours has several advantages, among which the fact that, in some cases, it does not rely on the self-destruct feature.

Keywords: non-malleable codes; tamper-resilient cryptography; split-state model.

Contents

1 Introduction

Tampering attacks are subtle attacks that undermine the security of cryptographic implementations by exploiting physical phenomena that allow to modify the underlying secrets. Indeed, a long line of works (see, e.g., [12, 10, 42, 53]) has established that black-box interaction with a tampered implementation can potentially expose the entire content of the secret memory. Given this state of affairs, protecting cryptographic schemes against tampering attacks has become an important goal for modern cryptographers.

An elegant solution to the threat of tampering attacks against the memory comes from the notion of non-malleable codes (NMCs), put forward by Dziembowski, Pietrzak, and Wichs [32]. Intuitively, a non-malleable encoding $(\mathsf{Encode}, \mathsf{Decode})$ allows to encode a value M into a codeword $C \leftarrow \mathsf{Encode}(M)$, with the guarantee that a modified codeword $\widetilde{C} = f(C)$ w.r.t. a tampering function $f \in \mathcal{F}$, when decoded, yields either M itself, or a completely unrelated value. An important parameter for characterizing the security guarantee offered by NMCs is the class of modifications $\mathcal{F}$ that are supported by the scheme. Since non-malleability is impossible to obtain for arbitrary (albeit efficient) modifications,[1] research on NMCs has focused on constructing such schemes in somewhat restricted, yet interesting, models. One such model that has been the focus of intensive research (see, e.g., [50, 36, 3, 2]) is the split-state model, where the codeword C consists of two parts (C_0, C_1) that can be modified independently (yet arbitrarily). This setting is also the focus of this paper.

Unfortunately, standard NMCs protect only against a single tampering attack.[2] To overcome this limitation, Faust *et al.* [36] introduced *continuously* non-malleable codes (CNMCs for short), where the attacker can tamper for an unbounded (polynomial) number of times with

[1] As it can be seen by considering the tampering function that first decodes the codeword, flips one bit of the message, and then encodes the result.

[2] When using NMCs to obtain security against memory tampering, one can still obtain security against continuous attacks by enforcing a re-encoding of the secret key after each invocation; however, this comes with several disadvantages [33], among which the fact that the encoding process is considerably more complex than the decoding process.

the codeword, until a decoding error happens which triggers the self-destruction of the device. As argued in [36], the self-destruct capability is necessary, as each decoding error might be used to signal one bit of information about the target codeword.

Another desirable feature of non-malleable codes is their ability to additionally tolerate leakage attacks, by which the adversary can obtain partial information on the codeword while performing a tampering attack. Note that in the split-state model this means that the adversary can leak independently from the two parts C_0 and C_1. All previous constructions of leakage-resilient NMCs either achieve security in the so-called bounded-leakage model [50, 36, 4], where the total amount of leakage (from each part) is upper-bounded by a value ℓ that is a parameter of the scheme, or only satisfy non-continuous non-malleability [33].

1.1 Our Contributions

We introduce a new form of CNMCs (dubbed R-CNMCs) that include a split-state algorithm for refreshing a *valid* codeword. The refresh procedure is invoked either after a decoding error happens, or in order to amplify resilience to leakage, and takes place *directly* on the memory and without the need of a central unit. Our new model has two main attractive features, which we emphasize below.

- It captures security in the so-called continual noisy-leakage model, where between each refresh the adversary can leak an arbitrary (yet independent) amount of information on the two parts C_0, C_1, as long as the leakage does not reveal (information-theoretically) more than ℓ bits of information. Importantly, this restriction is well-known to better capture realistic leakage attacks.
- It avoids the need for the self-destruct capability in some applications. Besides mitigating simple denial-of-service attacks, this feature is useful in situations where a device (storing an encoding of the secret state) is not in the hands of the adversary (e.g., because it has been infected by a malware), as it still allows to (non-interactively) refresh the secret state and continue to safely use the device in the wild.

Our first contribution is an abstract framework for constructing R-CNMCs, which we are able to instantiate under the external Diffie-Hellman assumption. This constitutes the first NMC that achieves at the same time continuous non-malleability and security under continual noisy leakage, in the split-state model (assuming an untamperable common reference string).

Next, we explore applications of R-CNMCs. As second contribution, we show how to construct a split-state[3] signature scheme resilient to continuous tampering and leakage attacks, without relying on the self-destruct capability, and where the memory content is refreshed in case a decoding error is triggered. Interestingly, Fujisaki and Xagawa [40] recently showed that such a notion is impossible to achieve for standard (i.e., non split-state) signature schemes, even if the self-destruct capability is available; hence, our approach can be interpreted as a possible way to circumvent the impossibility result in [40].

Our third contribution consists of two generic compilers for protecting random access machine (RAM) computations against tampering attacks [24, 37]. Here, we build on the important work of Dachman-Soled *et al.* [24], who showed how to compile any RAM to be resilient to continual tampering and leakage attacks, by relying both on an update and a self-destruct mechanism. We refer the reader to Section 6.2 for further details on our RAM compilers. Below, we highlight the main technical ideas behind our code construction.

[3]This means that the signing key is made of two shares that are stored in two separate parts of the memory, and need to be combined upon signing.

1.2 Code Construction

The starting point of our code construction is the recent work of Faonio and Nielsen [33]. The scheme built in [33] follows a template that originates in the work of Liu and Lysyanskaya [50], in which the left side of the encoding stores the secret key *sk* of a public-key encryption (PKE) scheme, whereas the right side of the encoding stores a ciphertext c, encrypting the encoded message M, plus a non-interactive zero-knowledge (NIZK) argument that proves knowledge of the secret key under the label c; the PKE scheme is chosen to be a continual-leakage resilient storage friendly PKE (CLRS friendly PKE for short) scheme (see Dodis *et al.* [29]), whereas the NIZK is chosen to be a malleable NIZK argument of knowledge (see Chase *et al.* [15]). Such a code was shown to admit a split-state refresh procedure, and, at the same time, to achieve *bounded-time* non-malleability.

The NM code of [33] does not satisfy security against continuous attacks. In fact, an attacker can create two valid codewords (C_0, C_1) and (C_0, C_1') such that $\mathsf{Decode}(C_0, C_1) \neq \mathsf{Decode}(C_0, C_1')$. Given this, the adversary can tamper the left side to C_0 and the right side to either C_1 or C_1' according to the bits of the right side of the target encoding. This way, assuming tampering is non-persistent, the adversary can leak all the bits of C_1 without activating the self-destruct mechanism. More in general, for any CNMC it should be hard to find two valid codewords (C_0, C_1) and (C_0, C_1') such that $\mathsf{Decode}(C_0, C_1) \neq \mathsf{Decode}(C_0, C_1')$. This property, called "message uniqueness", was originally defined in [36].[4]

Going back to the described code construction, an attacker can sample a secret key *sk* and create two ciphertexts, c_0 for M and c' for M', where $M \neq M'$, together with the corresponding honestly computed NIZKs, and thus break message uniqueness. We fix this issue by further binding the right and the left side of an encoding. To do so, while still being able to refresh the two parts independently, we keep untouched the structure of the right side of the codeword, but we change the message that it carries. Specifically, the ciphertext c in our code encrypts the message M concatenated with the randomness r for a commitment γ that is stored in the left side of the codeword together with the secret key for the PKE scheme. Observe that "message uniqueness" is now guaranteed by the binding property of the commitment scheme. Our construction additionally includes another NIZK for proving knowledge of the committed value under the label *sk*, in order to further link together the left and the right side of the codeword.

Proof strategy. Although our construction shares similarities with previous work, our proof techniques diverge significantly from the ones in [33, 36]. The main trick of [36] is to show that given one half of the codeword it is possible to fully simulate the view of the adversary in the tampering experiment, until a decoding error happens. To catch when a decoding error happens, [36] carries on two independent simulators in an interleaved fashion; as they prove, a decoding error happens exactly when the outputs of the two simulations diverge. The main obstacle they faced is how to succinctly compute the index where the two simulations diverge, so that non-malleability can be reduced to the security of the inner leakage-resilient storage scheme (see Daví *et al.* [27]) they rely on. To solve this, [36] employs an elegant dichotomous search-by-hash strategy over the partial views produced by the two simulators. At this point the experiment can terminate, and thanks to a specific property of the leakage-resilient storage scheme, the simulator can "extract" the decoded value corresponding to a tampered codeword.

[4]Faust *et al.* also consider "codeword uniqueness", where the fact that $\mathsf{Decode}(C_0, C_1) \neq \mathsf{Decode}(C_0, C_1')$ is not required (as long as $C_1 \neq C_1'$). However, this flavor of uniqueness only allows to rule out so-called continuous *super* non-malleability, where one asks that not only the decoded value, but the entire modified codeword, be independent of the message. It is easy to see that no R-CNMC can satisfy "codeword uniqueness", as for instance C_1' could be obtained as a valid refresh of C_1.

Unfortunately, we cannot generalize the above proof strategy to multiple rounds. Indeed, Faust *et al.* exploit the fact that the leakage-resilient storage scheme remains secure even when the adversary is allowed to obtain one half of the encoding in full. Clearly, after that, the adversary is not allowed to leak further from the other half of the codeword. In our case, we would need to repeat the above trick again and again, in particular after each decoding error (and subsequent refresh of the target encoding) happens; however, once the reduction obtains one half of the codeword it cannot ask leakage queries anymore, so that it is unclear how to complete the proof. We give a solution to this problem by relying on a simple information-theoretic observation.

Let (X_0, X_1) be two random variables, and consider a process that interleaves the computation of a sequence of leakage functions $g^1, g^2, g^3, \ldots$ from X_0 and from X_1. The process continues until, for some index $i \in \mathbb{N}$, we have that $g^i(X_0) \neq g^i(X_1)$. We claim that $\bar{g}^i(X_0) := g^1(X_0), g^2(X_0), \cdots, g^{i-1}(X_0)$ do not reveal more information about X_0 than what X_1 and the index i already reveal. To see this, consider $\widetilde{\mathbb{H}}_\infty(X_0 \mid \bar{g}^i(X_0))$, i.e. the average conditional min-entropy of X_0, which is, roughly speaking, the amount (on average) of uncertainty of X_0 given $\bar{g}^i(X_0)$ as side information. (See Section 2.1 for the formal definition.) Now, since $\bar{g}^i(X_0)$ and $\bar{g}^i(X_1)$ are exactly the same random variables we can derive[5]:

$$\widetilde{\mathbb{H}}_\infty(X_0 \mid \bar{g}^i(X_0)) = \widetilde{\mathbb{H}}_\infty(X_0 \mid \bar{g}^i(X_1)) \geq \widetilde{\mathbb{H}}_\infty(X_0 \mid X_1,\ i).$$

The latter implies that the size of the view of the adversary, although potentially much larger than the leakage bound, might reveal only little information.

One can use the above observation in order to give an alternative proof of security for the scheme in [36], where the reduction to the leakage-resilient storage scheme loses only a factor $O(\kappa)$ in the leakage bound (instead of $O(\kappa \log \kappa)$). Briefly, the idea is to carry on two independent simulators in an interleaved fashion (as in [36]) and, at each invocation, outputting first the hashes[6] of the simulated tampered codeword, and then, if the hashes match, leak the full simulated tampered codeword. This allows to avoid the dichotomous search-by-hash strategy, and thus to limit the total amount of leakage to $O(\kappa)$.

1.3 Limitations of the Refresh Paradigm

The original motivation behind the concept of CNMCs, as discussed in [36], was to make the standard tampering application of NMCs more realistic, in the following sense. Imagine that the device we want to protect, besides storing an encoding of the secret key, also contains other non-encoded data. For example the device could be a laptop, and tampering attacks could be launched by a virus with read and write access to the memory. The virus could exploit the space available on the device to make multiple copies of the original codeword, which thus can be tampered continuously. However, since the underlying NMC is resilient to continuous attacks, the privacy of the secret key is still preserved. Intuitively, this works because the adversary, to gather information about the codeword, needs to trigger a decoding error, however, once such an event happens the device self-destructs and the game is over.

While our notion of R-CNMCs allows for the very same application as considered in [36] (as any R-CNMC is in particular a CNMC), the straight-forward extension of the above scenario to a setting where the self-destruct is not available, but the encoding of the key is refreshed after a decoding error happens, does not work. To make the attack even harder, suppose that all the locations in the memory are encoded using a R-CNMC, and whenever a decoding error happens,

[5] In the last equation, we also use that the output of a function is at most as informative as the input.

[6] By collision resistance of the hash function, if the two hashes match then the simulated tampered codewords are the same for both the simulators.

the full memory is refreshed. An adversary could perform the following tampering attack. In case the i-th bit of the attacked codeword is equal to 1, the i-th location of the memory is corrupted (i.e., it is replaced with an invalid codeword), otherwise it is left untouched. Thus, the adversary reads sequentially all the memory locations. Very soon a decoding error will be triggered and all the codewords refreshed, however the information of the original codeword is still present in the memory (as the untouched codewords will be refreshed to valid codewords, while the corrupted codewords will still trigger a decoding error). This means that eventually the adversary is able to fully retrieve the original codeword (even if it has been refreshed many times), which leads to a clear breach of non-malleability.

1.4 Further Related Work

Several constructions of non-malleable codes in the split-state model appeared in the literature, both for the information-theoretic [32, 31, 3, 20, 4, 2, 14, 5, 48, 49, 17] and computational setting [50, 36, 24, 1, 46, 52, 23]. Non-malleable codes exist also for several other models besides split-state tampering, including bit-wise independent tampering and permutations [20, 6, 7, 22, 21], circuits of polynomial size [32, 19, 38, 39], constant-state tampering [18], block-wise tampering [13], space-bounded algorithms [35, 9], bounded-depth circuits [8, 16], and partial functions [47].

The concept of non-malleable codes with split-state refresh was recently proposed in [33]. The main difference with our work is that in [33] the refresh mechanism essentially allows to only tolerate continual leakage, but the codeword needs to be refreshed at each invocation in order to achieve security against continuous tampering attacks. In contrast, R-CNMCs satisfy both resilience to continual leakage and to continuous tampering without invoking a refresh at each invocation.

Security against tampering attacks against the memory can also be obtained without relying on non-malleable codes. See, e.g., [45, 25, 40, 34, 26] (and the references therein) for some examples.

2 Preliminaries and Building Blocks

2.1 Basic Notation

We let $\mathbb{N}$ denote the naturals and $\mathbb{R}$ denote the reals. For $a, b \in \mathbb{R}$, we define $[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$; for $a \in \mathbb{N}$ we let $[a] = \{0, 1, \ldots, a\}$. If x is a bit-string, we denote its length by $|x|$, and, for any $i \leq |x|$, we let $x[i]$ be the i-th bit of x. If $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the number of elements in $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow \mathcal{X}$. When A is an algorithm, we write $y \leftarrow \mathsf{A}(x)$ to denote a run of A on input x and output y; if A is randomized, then y is a random variable and $\mathsf{A}(x; r)$ denotes a run of A on input x and random coins r. An algorithm A is *probabilistic polynomial-time* (PPT) if A is allowed to make random choices, and the computation of $\mathsf{A}(x; r)$ terminates in at most a polynomial number of steps (in $|x|$), for any input $x \in \{0, 1\}^*$ and randomness $r \in \{0, 1\}^*$.

Indistinguishability and min-entropy. We let $\kappa \in \mathbb{N}$ be a security parameter. A function $\nu : \mathbb{N} \to [0, 1]$ is called *negligible* in κ (or simply negligible) if it vanishes faster than the inverse of any polynomial in κ. For two ensembles $\mathcal{X} = \{X_\kappa\}_{\kappa\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_\kappa\}_{\kappa\in\mathbb{N}}$, we write $\mathcal{X} \approx_c \mathcal{Y}$, meaning that every PPT distinguisher D has negligible advantage in distinguishing $\mathcal{X}$ and $\mathcal{Y}$, i.e., $|\mathbb{P}[\mathsf{D}(X_\kappa) = 1] - \mathbb{P}[\mathsf{D}(Y_\kappa) = 1]| \leq \nu(\kappa)$ where $\nu(\kappa)$ is a negligible function. Similarly, we write $\mathcal{X} \approx_s \mathcal{Y}$ for statistical indistinguishability between $\mathcal{X}$ and $\mathcal{Y}$, meaning that the above indistinguishability holds for all (even computationally unbounded) distinguishers (or, equivalently, that the statistical distance between $\mathcal{X}$ and $\mathcal{Y}$ is negligible).

The min-entropy of a random variable X, defined over a set $\mathcal{X}$, is denoted as $\mathbb{H}_\infty(X) := -\log \max_{x \in \mathcal{X}} \mathbb{P}[X = x]$ and represents the best chance of guessing X by an unbounded adversary. Conditional average min-entropy captures how hard it is to guess X on average, given some side information $Z \in \mathcal{Z}$ (possibly related to X), and it is denoted as $\widetilde{\mathbb{H}}_\infty(X|Z) := -\log \mathbb{E}_{z \in \mathcal{Z}} \max_{x \in \mathcal{X}} \mathbb{P}[X = x | Z = z]$. We rely on the following lemmata [30].

Lemma 1. *Let X and Z be possibly correlated random variables, and let f be any (possibly randomized) function. Then, $\widetilde{\mathbb{H}}_\infty(X|f(Z)) \geq \widetilde{\mathbb{H}}_\infty(X|Z)$.*

Lemma 2. *Let X, Z_1, Z_2 be possibly correlated random variables, where Z_2 takes at most 2^ℓ values. Then, $\widetilde{\mathbb{H}}_\infty(X|Z_1, Z_2) \geq \widetilde{\mathbb{H}}_\infty(X|Z_1) - \ell$.*

Following [28], we define a notion of a function being ℓ-leaky.

Definition 1 (ℓ-leaky function)**.** A (possibly randomized) function $g : \{0,1\}^* \to \{0,1\}^*$ is ℓ-leaky, if for all X distributed over $\{0,1\}^*$ we have that $\widetilde{\mathbb{H}}_\infty(X|g(X)) \geq \mathbb{H}_\infty(X) - \ell$.

As shown in [28, Lemma L.3] the above notion composes nicely, meaning that if two functions are respectively ℓ_1-leaky and ℓ_2-leaky, their concatenation is $(\ell_1 + \ell_2)$-leaky.

Oracle machines. Given a pair of strings $X = (X_0, X_1) \in (\{0,1\}^*)^2$ define the oracle $\mathcal{O}_\infty(X)$ to be the *split-state leakage oracle* that takes as input tuples of the form (β, g), where $\beta \in \{0,1\}$ is an index and g is a function described as a circuit, and outputs $g(X_\beta)$. An adversary A with oracle access to $\mathcal{O}_\infty(X)$ is called *ℓ-valid*, for some $\ell \in \mathbb{N}$, if for all $\beta \in \{0,1\}$ the concatenation of the leakage functions sent by A is an ℓ-leaky function of X_β.

Given two PPT interactive algorithms A and B we write $(y_\mathsf{A}; y_\mathsf{B}) \leftarrow (\mathsf{A}(x_\mathsf{A}) \leftrightarrows \mathsf{B}(x_\mathsf{B}))$ to denote the execution of algorithm A (with input x_A) and algorithm B (with input x_B). The string y_A (resp. y_B) is the output of A (resp. B) at the end of such interaction. In particular, we write $\mathsf{A} \leftrightarrows \mathcal{O}_\infty(X)$ to denote A having oracle access to the leakage oracle with input X. Moreover, we write $\mathsf{A} \leftrightarrows \mathsf{B}, \mathsf{C}$ to denote A interacting in an interleaved fashion both with B and with C.

2.2 Non-Interactive Zero-Knowledge

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP-relation; the language associated with $\mathcal{R}$ is $\mathcal{L}_\mathcal{R} := \{x : \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. We typically assume that given a pair (x, w) it is possible to efficiently verify whether $(x, w) \in \mathcal{R}$ or not. Roughly, a non-interactive argument (NIA) for an NP-relation $\mathcal{R}$ allows to create non-interactive proofs for statements $x \in \mathcal{L}$, when additionally given a valid witness w corresponding to x. More formally, a NIA $\mathcal{NIA} := (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$ for $\mathcal{R}$, with label space Λ, is a tuple of PPT algorithms specified as follows: (1) The (randomized) initialization algorithm CRSGen takes as input the security parameter 1^κ, and creates a common reference string (CRS) $\omega \in \{0,1\}^*$; (2) The (randomized) prover algorithm Prove takes as input the CRS ω, a label $\lambda \in \Lambda$, and a pair (x, w) such that $(x, w) \in \mathcal{R}$, and produces a proof $\pi \leftarrow \mathsf{Prove}^\lambda(\omega, x, w)$; (3) The (deterministic) verifier algorithm Ver takes as input the CRS ω, a label $\lambda \in \Lambda$, and a pair (x, π), and outputs a decision bit $\mathsf{Ver}^\lambda(\omega, x, \pi)$.

Completeness means that for all CRSs ω output by $\mathsf{CRSGen}(1^\kappa)$, for all labels $\lambda \in \Lambda$, and for all pairs $(x, w) \in \mathcal{R}$, we have that $\mathsf{Ver}^\lambda(\omega, x, \mathsf{Prove}^\lambda(\omega, x, w)) = 1$ with all but a negligible probability. Below, we define the security properties we require for NIAs. The first property, known as adaptive multi-theorem zero-knowledge, intuitively says that honestly computed proofs do not reveal anything beyond the validity of the statement.

```
Game $\mathbf{G}^{\text{lm-se}}_{\mathcal{NIA},\mathsf{A},\mathsf{K}}(\kappa, \Phi)$:
$\mathcal{Q}_{\downarrow}, \mathcal{Q} \leftarrow \emptyset$;
$(\omega, \tau_{\text{sim}}, \tau_{\text{ext}}) \leftarrow \mathsf{S}_0(1^\kappa)$;
$(\widetilde{x}, \widetilde{\lambda}, \widetilde{\pi}) \leftarrow \mathsf{A}(\omega) \leftrightarrows \mathcal{O}^*_{\mathsf{sim}}(\cdot)$;
$(\widehat{w}, \phi, \widehat{\lambda}) \leftarrow \mathsf{K}(\tau_{\text{ext}}, \widetilde{\lambda}, \widetilde{x}, \widetilde{\pi})$;
Output 1 iff $((\widetilde{\lambda}, \widetilde{x}), \widetilde{\pi}) \notin \mathcal{Q}$ and $\mathsf{Ver}^{\widetilde{\lambda}}(\omega, \widetilde{x}, \widetilde{\pi}) = 1$,
but either:
    (i) $\widehat{w} \neq \perp$ and $(\widetilde{x}, \widehat{w}) \notin \mathcal{R}$; or
    (ii) $(\widehat{\lambda}, \phi) \neq (\perp, \perp)$ and either $\phi \notin \Phi$,
    or $(\widehat{\lambda}, \widetilde{x}) \notin \mathcal{Q}_{\downarrow}$, or $\phi(\widehat{\lambda}) \neq \widetilde{\lambda}$; or
    (iii) $(\widehat{w}, \phi, \widehat{\lambda}) = (\perp, \perp, \perp)$.

Oracle $\mathcal{O}^*_{\mathsf{sim}}(\lambda, x)$:
$\mathcal{Q}_{\downarrow} \leftarrow \mathcal{Q} \cup \{(\lambda, x)\}$;
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{((\lambda, x), \pi)\}$;
Return $\mathsf{S}_1(\tau_{\text{sim}}, \lambda, x)$.
```

```
Game $\mathbf{G}^{\text{lb-prv}}_{\mathcal{NIA},\mathsf{A}}(\kappa)$:
$\omega \leftarrow \mathsf{CRSGen}(1^\kappa)$; $b \leftarrow \{0,1\}$;
$(\phi, \lambda, x, w, \pi) \leftarrow \mathsf{A}(\omega)$;
If $(b = 0)$
    $\pi' \leftarrow \mathsf{Prove}(\omega, \phi(\lambda), x, w)$;
Else
    $\pi' \leftarrow \mathsf{LEval}(\omega, \phi, (x, \lambda, \pi))$;
$b' \leftarrow \mathsf{A}(\omega, \pi')$;
Output 1 iff:
    (i) $b = b'$;
    (ii) $(x, w) \in \mathcal{R}$;
    (iii) $\mathsf{Ver}^{\lambda}(\omega, x, \pi) = 1$;
    (iv) $\phi \in \Phi$.
```

Figure 1: Games defining Φ-malleable label simulation extractability, and label simulation privacy, of a NIA $\mathcal{NIA}$.

Definition 2 (Adaptive multi-theorem zero-knowledge)**.** Let $\mathcal{NIA}$ be a NIA for a relation $\mathcal{R}$. We say that $\mathcal{NIA}$ satisfies adaptive multi-theorem zero-knowledge if the following holds:

(i) There exists a PPT algorithm S_0 that outputs a CRS ω, and a simulation trapdoor τ_{sim}.

(ii) There exist a PPT simulator S_1 and a negligible function $\nu : \mathbb{N} \to [0,1]$ such that, for all PPT adversaries A, we have

$$\Big| \mathbb{P}\left[\mathsf{A}(\omega) \leftrightarrows \mathcal{O}_{\mathsf{prv}}(\omega, \cdot) = 1 : \ \omega \leftarrow \mathsf{CRSGen}(1^\kappa)\right] - \mathbb{P}\left[\mathsf{A}(\omega) \leftrightarrows \mathcal{O}_{\mathsf{sim}}(\tau_{\text{sim}}, \cdot) = 1 : \ (\omega, \tau_{\text{sim}}) \leftarrow \mathsf{S}_0(1^\kappa)\right] \Big| \leq \nu(\kappa).$$

In the above equation, both oracles $\mathcal{O}_{\mathsf{prv}}$ and $\mathcal{O}_{\mathsf{sim}}$ take as input a tuple (λ, x, w) and check if $(x, w) \in \mathcal{R}$: If this is the case, $\mathcal{O}_{\mathsf{prv}}$ returns $\mathsf{Prove}^{\lambda}(\omega, x, w)$, whereas $\mathcal{O}_{\mathsf{sim}}$ ignores w and outputs a simulated argument $\mathsf{S}_1(\tau_{\text{sim}}, \lambda, x)$; otherwise, both oracles return $\perp$.

Roughly speaking, a NIA satisfies soundness if no efficient malicious adversary can create an accepting proof for a false statement (i.e., for a statement x such that $x \notin \mathcal{L}$). Knowledge soundness is a stronger form of soundness, essentially demanding that we can efficiently extract a witness from any accepting proof created by an efficient adversary. Our construction will be based on a so-called label-malleable NIA, parameterized by a set of label transformations Φ, where for any $\phi \in \Phi$, the co-domain of ϕ is a subset of Λ. For such NIAs, given a proof π under some label $\lambda \in \Lambda$, and the CRS ω, one can efficiently generate new proofs π' for the same statement under a different label $\phi(\lambda)$, for any $\phi \in \Phi$ (without knowing a witness); this is formalized via an additional (randomized) label-derivation algorithm LEval, which takes as input the CRS ω, a transformation $\phi \in \Phi$, a label $\lambda \in \Lambda$, and a pair (x, π), and outputs a new proof π'. The property below intuitively says that a NIA satisfies knowledge soundness, except that labels are malleable w.r.t. Φ.

Definition 3 (Φ-Malleable label simulation extractability)**.** Let $\mathcal{NIA}$ be a NIA for a relation $\mathcal{R}$, with label set Λ, and let Φ be a set of label transformations. We say that $\mathcal{NIA}$ is Φ-*malleable label simulation extractable* (Φ-ml-SE for short) if the following holds.

(i) There exists a PPT algorithm S_0 that outputs a CRS ω, a simulation trapdoor τ_{sim}, and an extraction trapdoor τ_{ext}.

(ii) There exists a PPT algorithm K and a negligible function $\nu : \mathbb{N} \to [0,1]$ such that, for all PPT adversaries A, we have

$$\mathbb{P}\left[\mathbf{G}^{\text{lm-se}}_{\mathcal{NIA},\mathsf{A},\mathsf{K}}(\kappa,\Phi)\right] \leq \nu(\kappa),$$

where the game $\mathbf{G}^{\text{lm-se}}_{\mathcal{NIA},\mathsf{A},\mathsf{K}}(\kappa,\Phi)$ is described in Fig. 1.

The last property, called *label derivation privacy*, says that it is hard to distinguish a fresh proof for some statement x (with witness w) under label λ, from a proof re-randomized using algorithm LEval w.r.t. some function $\phi \in \Phi$; moreover, the latter should hold even if (x, w, λ, ϕ) are chosen adversarially (possibly depending on the CRS).

Definition 4 (Label derivation privacy)**.** Let $\mathcal{NIA}$ be a NIA for a relation $\mathcal{R}$, and let Φ be a set of label transformations. We say that $\mathcal{NIA}$ has *label derivation privacy* if for all PPT A there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\left|\mathbb{P}\left[\mathbf{G}^{\text{lb-prv}}_{\mathcal{NIA},\mathsf{A}}(\kappa) = 1\right] - \frac{1}{2}\right| \leq \nu(\kappa),$$

where game $\mathbf{G}^{\text{lb-prv}}_{\mathcal{NIA},\mathsf{A}}(\kappa)$ is described in Fig. 1.

2.3 Public-Key Encryption

A public-key encryption (PKE) scheme is a tuple of algorithms $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows. (1) The (randomized) algorithm Setup takes as input the security parameter 1^κ and outputs public parameters $\rho \in \{0,1\}^*$; all algorithms are implicitly given ρ as input. (2) The (randomized) algorithm KGen takes as input the public parameters ρ and outputs a public/secret key pair (pk, sk); the set of all secret keys is denoted by $\mathcal{SK}$, and the set of all public keys by $\mathcal{PK}$. Additionally, we require the existence of a (randomized) function PK which, upon input $sk \in \mathcal{SK}$, produces a valid public key $pk \in \mathcal{PK}$. (3) The (randomized) algorithm Enc takes as input the public key pk, a message $m \in \mathcal{M}$, and randomness $r \in \{0,1\}^*$, and outputs a ciphertext $c = \mathsf{Enc}(pk, m; r)$; the set of all ciphertexts is denoted by $\mathcal{C}$. (4) The (deterministic) algorithm Dec takes as input the secret key sk and a ciphertext c, and outputs $m = \mathsf{Dec}(sk, c)$ which is either equal to some message $m \in \mathcal{M}$, or to an error symbol $\perp$. We will require two additional algorithms, the first one to re-randomize a given ciphertext, and the second one for re-randomizing the secret key (without changing the public key). More formally: (5) The (randomized) algorithm $\mathsf{UpdateC}$ takes as input a ciphertext c, and outputs a new ciphertext c'. (6) The (randomized) algorithm $\mathsf{UpdateS}$ takes as input a secret key sk, and outputs a new secret key sk'.

We say that $\mathcal{PKE}$ satisfies *correctness* if for all $\rho \leftarrow \mathsf{Setup}(1^\kappa)$ and for all $(pk, sk) \leftarrow \mathsf{KGen}(\rho)$, we have that $\mathbb{P}[\mathsf{Dec}(\mathsf{UpdateS}(sk), \mathsf{UpdateC}(\mathsf{Enc}(pk, m))) = m] = 1$, where the randomness is taken over the internal coin tosses of algorithms Enc, $\mathsf{UpdateS}$ and $\mathsf{UpdateC}$. Additionally, we require that for all ρ output by Setup, and for all $(pk, sk) \leftarrow \mathsf{KGen}(\rho)$: (i) For any sk' such that $\mathsf{PK}(sk') = pk$, and any $c \in \mathcal{C}$, we have that $\mathsf{Dec}(sk, c) = \mathsf{Dec}(sk', c)$; (ii) For any $sk' \leftarrow \mathsf{UpdateS}(sk)$, we have that $\mathsf{PK}(sk) = \mathsf{PK}(sk')$.

```
Game $\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{A}}(\kappa,\ell)$:
$b \leftarrow \{0,1\}$;
$\rho \leftarrow \mathsf{Setup}(1^\kappa)$;
$(pk, sk) \leftarrow \mathsf{KGen}(\rho)$;
$(m_0, m_1) \leftarrow \mathsf{A}(\rho, pk)$;
If $(|m_0| \neq |m_1|)$
    $m_0 \leftarrow m_1$;
$c \leftarrow \mathsf{Enc}(pk, m_b)$;
$S_0 \leftarrow sk$; $S_1 \leftarrow c$;
$b' \leftarrow (\mathsf{A}(\rho, pk) \leftrightarrows \mathsf{Update}(\cdot), \mathcal{O}_\infty((S_0, S_1), \cdot))$
Return $(b' = b)$.

Oracle $\mathsf{Update}(\beta)$:
If $(\beta = 0)$
    $sk \leftarrow S_0$;
    $sk' \leftarrow \mathsf{UpdateS}(sk)$;
    $S_0 \leftarrow sk'$;
If $(\beta = 1)$
    $c \leftarrow S_1$;
    $c' \leftarrow \mathsf{UpdateC}(c)$;
    $S_1 \leftarrow c'$.
```

Figure 2: Game defining CLRS security of a PKE scheme.

We now turn to define continual leakage-resilient storage (CLRS) friendly PKE. Consider the game $\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{A}}(\kappa,\ell)$ as defined in Fig. 2. The definition below roughly say that a PKE scheme satisfies semantic security against all ℓ-valid efficient adversaries that can observe independent leakage from $S_0 = sk$ and $S_1 = c$ (where c is the challenge ciphertext).

Definition 5 (CLRS friendly PKE)**.** For $\kappa \in \mathbb{N}$, let $\ell = \ell(\kappa)$ be the leakage parameter. We say that $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdateC}, \mathsf{UpdateS})$ is ℓ-noisy CLRS friendly if for all PPT ℓ-valid adversaries A there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\left| \mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{A}}(\kappa,\ell) = 1 \right] - \frac{1}{2} \right| \leq \nu(\kappa),$$

where the game $\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{A}}(\kappa,\ell)$ is described in Fig. 2.

We observe that Definition 5 is slightly different w.r.t. the definition of Dodis *et al.* [29]. In fact, on the one hand, our definition does not consider leakage from the update process, but, on the other hand, we consider noisy leakage and not just bounded leakage as in [29].

Finally, we introduce two extra properties relative to the algorithms $\mathsf{UpdateC}$ and $\mathsf{UpdateS}$ of a PKE scheme. The first property says that the distributions of fresh and updated ciphertexts are the same. The second property formalizes a similar requirement for fresh and updated keys.

Definition 6 (Ciphertext-update privacy)**.** We say that $\mathcal{PKE}$ is *perfectly chipertext-update private* if for all $\kappa \in \mathbb{N}$, all $\rho \leftarrow \mathsf{Setup}(1^\kappa)$, all $(pk, sk) \leftarrow \mathsf{KGen}(\rho)$, and any $m \in \mathcal{M}, c \in \mathcal{C}$ such that $c = \mathsf{Enc}(pk, m; r)$ for some $r \in \{0,1\}^*$, the distributions $\{pk, \mathsf{Enc}(pk, m)\}$ and $\{pk, \mathsf{UpdateC}(c)\}$ are equivalent.

Definition 7 (Secret-key-update privacy)**.** We say that $\mathcal{PKE}$ is *perfectly secret-key-update private* if for all $\kappa \in \mathbb{N}$, all $\rho \leftarrow \mathsf{Setup}(1^\kappa)$, and all $(pk, sk) \leftarrow \mathsf{KGen}(\rho)$, the distributions $\{pk, \mathsf{UpdateS}(sk)\}$ and $\{(pk', sk') \leftarrow \mathsf{KGen}(\rho) : \ pk' = pk\}$ are equivalent.

2.4 Commitment Schemes

A (non-interactive) commitment scheme with message space $\mathcal{M}$ is a tuple of algorithms $\mathcal{COM} = (\mathsf{CRSGen}, \mathsf{Commit})$ defined as follow: (1) The (randomized) algorithm CRSGen takes as input the security parameter 1^κ, and outputs a commitment key ω; (2) The (randomized) algorithm

```
Experiment $\mathbf{G}^{\text{euf-cma}}_{\mathcal{SS},\mathsf{A}}(\kappa)$:
$\mathcal{Q} \leftarrow \emptyset$; $\rho \leftarrow \mathsf{Setup}(1^\kappa)$;
$(vk, sk) \leftarrow \mathsf{Gen}(\rho)$;
$(m^*, \sigma^*) \leftarrow (\mathsf{A}(vk, \rho) \leftrightarrows \mathcal{O}_{\mathsf{sign}}(sk, \cdot))$;
Return 1 iff:
    (i) $m^* \notin \mathcal{Q}$;
    (ii) $\mathsf{Vrfy}(vk, (m^*, \sigma^*) = 1)$.

Oracle $\mathcal{O}_{\mathsf{sign}}(sk, m)$:
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$;
Return $\sigma \leftarrow \mathsf{Sign}(sk, m)$.

Experiment $\mathbf{G}^{\text{hide}}_{\mathcal{COM},\mathsf{A}}(\kappa, b)$:
$\omega \leftarrow \mathsf{CRSGen}(1^\kappa)$;
$(m_0, m_1) \leftarrow \mathsf{A}(\omega)$;
$\gamma \leftarrow \mathsf{Commit}(\omega, m_b)$;
Return $\mathsf{A}(\omega, \gamma)$.
```

Figure 3: Games defining computational hiding of a commitment scheme, and universal unforgeability against chosen-message attacks of a signature scheme.

Commit takes as input the commitment key ω, a message $m \in \mathcal{M}$, and randomness $r \in \{0,1\}^*$, and outputs a commitment γ.

Intuitively a commitment scheme satisfies two properties. The first property, called the *hiding property*, says that the commitment hides the message; the second property, called the *binding property* says that it is hard to open a valid commitment in two different ways. Below, we define these properties directly in the flavor we need for our application.

Definition 8 (Hiding and binding)**.** Let $\mathcal{COM} = (\mathsf{CRSGen}, \mathsf{Commit})$ be a commitment scheme. We say that $\mathcal{COM}$ is:

- Computationally hiding, if for all PPT adversaries A there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\left| \mathbb{P}\left[\mathbf{G}^{\text{hide}}_{\mathcal{COM},\mathsf{A}}(\kappa, 0) = 1\right] - \mathbb{P}\left[\mathbf{G}^{\text{hide}}_{\mathcal{COM},\mathsf{A}}(\kappa, 1) = 1\right] \right| \leq \nu(\kappa),$$

  where the game $\mathbf{G}^{\text{hide}}_{\mathcal{COM},\mathsf{A}}$ is described in Fig. 3.

- Statistically binding if for all even unbounded adversaries A there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that:

$$\mathbb{P}\left[\mathsf{Commit}(\omega, m; r) = \mathsf{Commit}(\omega, m'; r') \wedge m \neq m' : \begin{array}{c} \omega \leftarrow \mathsf{CRSGen}(1^\kappa); \\ (m, r, m', r') \leftarrow \mathsf{A}(\omega) \end{array}\right] \leq \nu(\kappa).$$

2.5 Signature Schemes

A signature scheme is a tuple of algorithms $\mathcal{SS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ defined as follow: (1) The (randomized) algorithm Setup takes as input the security parameter 1^κ and outputs public parameters $\rho \in \{0,1\}^*$; all algorithms are implicitly given ρ as input. (2) The (randomized) algorithm KGen takes as input the public parameters and outputs a verification/signing key pair (vk, sk). (3) The (randomized) algorithm Sign takes as input the secret key sk and a message m, and outputs a signature σ. (4) The (deterministic) algorithm Vrfy takes as input the verification key vk and a pair (m, σ) and outputs a decision bit.

For completeness, we require that for all ρ output by Setup, and for all (vk, sk) output by $\mathsf{KGen}(\rho)$, and for any message m, we have that $\mathsf{Vrfy}(vk, \mathsf{Sign}(sk, m)) = 1$ with probability one over coin tosses of the signing algorithm. As for security, we require that no efficient adversary with oracle access to a signature oracle and given the verification key, can forge a valid signature on a fresh message (i.e., a message not returned by the signature oracle).

```
Experiment $\mathbf{G}^{\text{sem}}_{\mathcal{SKE},\mathsf{A}}(\kappa)$:
$b \leftarrow \{0,1\}$; $K \leftarrow \mathsf{Gen}(1^\kappa)$;
$b' \leftarrow \mathsf{A}^{\mathcal{O}_{\mathsf{lr}}(K,b,\cdot,\cdot)}(1^\kappa)$;
Return 1 iff $b' = b$.
```

```
Oracle $\mathcal{O}_{\mathsf{lr}}(K, b, m_0, m_1)$:
If $|m_0| \neq |m_1|$
    Return $\perp$;
Else
    Return $c \leftarrow \mathsf{Enc}(K, m_b)$.
```

```
Experiment $\mathbf{G}^{\text{auth}}_{\mathcal{SKE},\mathsf{A}}(\kappa)$:
$K \leftarrow \mathsf{KGen}(1^\kappa)$;
$c^* \leftarrow \mathsf{A}^{\mathsf{Enc}(K,\cdot)}(1^\kappa)$;
Return 1 iff:
    (i) $\mathsf{Dec}(K, c^*) \neq \perp$;
    (ii) $c^* \notin \mathcal{Q}$;
```

Figure 4: Games defining semantic security and authenticity of an SKE scheme; the set $\mathcal{Q}$ in the last game contains all ciphertexts returned by oracle $\mathsf{Enc}(K, \cdot)$.

Definition 9 (EUF-CMA security)**.** Let $\mathcal{SS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. We say that $\mathcal{SS}$ is existentially unforgeable against chosen-message attacks (EUF-CMA) if for all PPT adversaries A there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\mathbb{P}\left[\mathbf{G}^{\text{euf-cma}}_{\mathcal{SS},\mathsf{A}}(\kappa) = 1\right] \leq \nu(\kappa),$$

where the game $\mathbf{G}^{\text{euf-cma}}_{\mathcal{SS},\mathsf{A}}(\kappa)$ is described in Fig. 3.

2.6 Authenticated Encryption

A secret key encryption (SKE) scheme consists of three PPT algorithms $\mathcal{SKE} := (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ specified as follows. (1) The (randomized) key generation algorithm KGen takes as input the security parameter 1^κ, and outputs a key K; (2) The (randomized) encryption algorithm Enc takes as input a key K, a message $m \in \{0,1\}^*$, random coins $r \in \{0,1\}^*$, and outputs a ciphertext c; (3) The (deterministic) decryption algorithm Dec takes as input a key K and a ciphertext c, and outputs a message m or a special symbol $\perp$ (indicating an error).

An SKE scheme meets correctness if for all keys K output by $\mathsf{KGen}(1^\kappa)$, and for all messages $m \in \{0,1\}^*$, we have that $\mathsf{Dec}(K, \mathsf{Enc}(K, m)) = m$ with probability one over the random coin tosses of the encryption algorithm. As for security, we define two properties that make SKE a so-called *authenticated encryption* scheme.

Let $\mathcal{O}_{\mathsf{lr}}(K, b)$ be an oracle that, upon input two message $m_0, m_1 \in \{0,1\}^*$, returns $c \leftarrow \mathsf{Enc}(K, m_b)$ if $|m_0| = |m_1|$ (and $\perp$ otherwise). The first property says that it is hard to distinguish the encryption of two messages, even when given access to such an encryption oracle.

Definition 10. (Semantic security) An SKE scheme $\mathcal{SKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is semantically secure if for all PPT adversaries A there is a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\left|\mathbb{P}\left[\mathbf{G}^{\text{sem}}_{\mathcal{SKE},\mathsf{A}}(\kappa) = 1\right] - \frac{1}{2}\right| \leq \nu(\kappa),$$

where the game $\mathbf{G}^{\text{sem}}_{\mathcal{SKE},\mathsf{A}}(\kappa)$ is described in Fig. 4.

The second property says that SKE has an elusive range, meaning that, without knowing the secret key, it is hard to produce a valid ciphertext.

Definition 11. (Authenticity) An SKE scheme $\mathcal{SKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies authenticity if for all PPT adversaries A there is a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\mathbb{P}\left[\mathbf{G}^{\text{auth}}_{\mathcal{SKE},\mathsf{A}}(\kappa) = 1\right] \leq \nu(\kappa),$$

where the game $\mathbf{G}^{\text{auth}}_{\mathcal{SKE},\mathsf{A}}(\kappa)$ is described in Fig. 4.

3 Non-Malleability with Refresh

3.1 Split-State Codes

A coding scheme in the CRS model is a tuple of polynomial-time algorithms $\mathcal{CS} = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ with the following syntax: (1) The (randomized) initialization algorithm Init takes as input the security parameter 1^κ, and outputs a CRS $\omega \in \{0,1\}^*$; (2) The (randomized) encoding algorithm Encode takes as input the CRS ω and a message $M \in \mathcal{M}$, and outputs a codeword $C \in \mathcal{C}$; (3) The (deterministic) decoding algorithm Decode takes as input the CRS ω and a codeword $C \in \mathcal{C}$, and outputs a value $M \in \mathcal{M} \cup \{\perp\}$ (where $\perp$ denotes an invalid codeword). A coding scheme is correct if for all ω output by $\mathsf{Init}(1^\kappa)$, and any $M \in \mathcal{M}$, we have $\mathbb{P}[\mathsf{Decode}(\omega, \mathsf{Encode}(\omega, M)) = M] = 1$, where the probability is taken over the randomness of the encoding algorithm.

We consider coding schemes with an efficient *refreshing algorithm*. Specifically, for a coding scheme $\mathcal{CS}$ we assume there exists a randomized algorithm Rfrsh that, upon input the CRS ω and a codeword $C \in \mathcal{C}$, outputs a codeword $C' \in \mathcal{C}$. For correctness we require that for all ω output by $\mathsf{Init}(1^\kappa)$, we have $\mathbb{P}[\mathsf{Decode}(\omega, \mathsf{Rfrsh}(\omega, C)) = \mathsf{Decode}(\omega, C)] = 1$, where the probability is over the randomness used by the encoding and refreshing algorithms.

Split-state model. In this paper we are interested in coding schemes in the split-state model, where a codeword consists of two parts that can be refreshed independently and without the need of any interaction. More precisely, given a codeword $C := (C_0, C_1)$, the refresh procedure $\mathsf{Rfrsh}(\omega, (\beta, C_\beta))$, for $\beta \in \{0,1\}$, takes as input either the left or the right part of the codeword, and updates it. Sometimes we also write $\mathsf{Rfrsh}(\omega, C)$ as a shorthand for the algorithm that independently executes $\mathsf{Rfrsh}(\omega, (0, C_0))$ and $\mathsf{Rfrsh}(\omega, (1, C_1))$.

Correctness here means that for all ω output by $\mathsf{Init}(1^\kappa)$, for all $C \in \mathcal{C}$, and for any $\beta \in \{0,1\}$, if we let $C' = (C'_0, C'_1)$ be such that $C'_\beta \leftarrow \mathsf{Rfrsh}(\omega, (\beta, C_\beta))$ and $C'_{1-\beta} = C_{1-\beta}$, then $\mathbb{P}[\mathsf{Decode}(\omega, C') = \mathsf{Decode}(\omega, C)] = 1$.

3.2 The Definition

We give the security definition for continuously non-malleable codes with split-state refresh (R-CNMCs for short). Our notion compares two experiments, which we denote by **Tamper** and **SimTamper** (cf. Fig. 5). Intuitively, in the experiment **Tamper** we consider an adversary continuously tampering with, and leaking from, a target encoding $C = (C_0, C_1)$ of a message $M \in \mathcal{M}$ (the message can be chosen adaptively, depending on the CRS). For each tampering attempt (f_0, f_1), the adversary gets to see the output $\widetilde{M}$ of the decoding corresponding to the modified codeword $\widetilde{C} = (f_0(C_0), f_1(C_1))$. Tampering is non-persistent, meaning that each tampering function is applied to the original codeword C, until, eventually, a decoding error happens; after that, the target encoding C is refreshed, and the adversary can start tampering with, and leaking from, the refreshed codeword.

In the experiment **SimTamper**, we consider a simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$, where S_0 outputs a simulated CRS, while S_1's goal is to simulate the view of the adversary in the real experiment; the simulator S_1, in faking a tampering query (f_0, f_1), is allowed to output a special value $\diamond$, signaling that (it believes) the adversary did not change the encoded message, in which case the experiment replaces $\diamond$ with M. We stress that the simulator S is stateful; in particular algorithms $\mathsf{S}_0, \mathsf{S}_1$ implicitly share a state.

For technical reasons, in some of our applications, we will actually need a slightly stronger security property, where after a decoding error happens, and before the original codeword is refreshed, the adversary is allowed to make one extra tampering query (f_0^*, f_1^*). Hence, if the

```
$\mathbf{Tamper}_{\mathcal{CS},\mathsf{A}}(\kappa, \ell, q)$:
$i \leftarrow 0$; $\mathtt{err}, \mathtt{stop} \leftarrow 0$
$\omega \leftarrow \mathsf{Init}(1^\kappa)$
$(M, s_0) \leftarrow \mathsf{A}_0(\omega)$
$C_0 := (C_0^0, C_1^0) \leftarrow \mathsf{Encode}(\omega, M)$
For all $i \in [0, q]$:
  $s_{i+1} \leftarrow (\mathsf{A}_1(s_i) \leftrightarrows \mathcal{O}_\infty(C_i), \mathcal{O}_{\mathsf{tamp}}(C_i))$
  $C_{i+1} \leftarrow \mathsf{Rfrsh}(\omega, C_i)$
  $i \leftarrow i + 1$; $\mathtt{err}, \mathtt{stop} \leftarrow 0$
Return $\mathsf{A}_2(s_q)$.

$\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa, \ell, q)$:
$i \leftarrow 0$
$\omega \leftarrow \mathsf{S}_0(1^\kappa)$
$(M, s_0) \leftarrow \mathsf{A}_0(\omega)$
For all $i \in [0, q]$:
  $s_{i+1} \leftarrow (\mathsf{A}_1(s_i) \leftrightarrows \mathsf{S}_1(\mathtt{Leak}, \cdot), \mathcal{O}_{\mathsf{sim\_tamp}}^{\mathsf{S}_1, M}(\cdot))$
  $i \leftarrow i + 1$
Return $\mathsf{A}_2(s_q)$.

Oracle $\mathcal{O}_{\mathsf{tamp}}(C_i, (f_0, f_1))$:
Upon $(\mathtt{Tamp}, f_0, f_1)$:
   $\widetilde{M} = \mathsf{Decode}(\omega, f_0(C_0^i), f_1(C_1^i))$
   If $(\widetilde{M} = \perp)$
      $\mathtt{err} \leftarrow 1$
   If $((\mathtt{err} = 1) \vee (\mathtt{stop} = 1))$
      $\widetilde{M} \leftarrow \perp$
   Return $\widetilde{M}$
Upon $(\mathtt{Final}, f_0^*, f_1^*)$:
   $\mathtt{stop} \leftarrow 1$
   $\widetilde{C}^* = (f_0^*(C_0^i), f_1^*(C_1^i))$
   $\widetilde{M}^* = \mathsf{Decode}(\omega, \widetilde{C}^*)$
   If $(\widetilde{M}^* \in \{\perp, M\})$
      $\widetilde{C}' \leftarrow \widetilde{M}^*$
   Else, $\widetilde{C}' \leftarrow \mathsf{Rfrsh}(\omega, \widetilde{C}^*)$
   Return $\widetilde{C}'$.

Oracle $\mathcal{O}_{\mathsf{sim\_tamp}}^{\mathsf{S}_1, M}(\cdot)$:
Upon $(\mathtt{Tamp}, f_0, f_1)$:
   $\widetilde{M} \leftarrow \mathsf{S}_1(\mathtt{Tamp}, f_0, f_1)$
   If $(\widetilde{M} = \diamond)$
      $\widetilde{M} \leftarrow M$
   Return $\widetilde{M}$
Upon $(\mathtt{Final}, f_0^*, f_1^*)$:
   $\widetilde{C}' \leftarrow \mathsf{S}_1(\mathtt{Final}, f_0^*, f_1^*)$
   If $((\widetilde{C}' = \diamond) \vee (\mathsf{Decode}(\omega, \widetilde{C}') = M))$
      $\widetilde{C}' \leftarrow M$
   Return $\widetilde{C}'$
```

Figure 5: Experiments defining continuously non-malleable codes with split-state refresh.

corresponding tampered codeword $\widetilde{C}^*$ is valid and is not an encoding of the original message M, the attacker receives a refreshing of $\widetilde{C}^*$; otherwise it obtains either M (in the former case) or $\perp$ (in the latter case). Unfortunately, it is impossible to further generalize our definition to handle the refreshing of invalid codewords.[7]

Definition 12 (Continuous non-malleability with split-state refresh)**.** For $\kappa \in \mathbb{N}$, let $\ell = \ell(\kappa)$ be a parameter. We say that a coding scheme $\mathcal{CS}$ is an ℓ-leakage-resilient and continuously non-malleable code with split-state refresh (R-CNMC for short) if for all adversaries $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$, where A_0 and A_2 are PPT algorithms and A_1 is an ℓ-valid (cf. Section 2.1) deterministic polynomial-time algorithm, there exists a PPT simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$ and a negligible

[7]This can be seen by the following attack. Consider an adversary that computes offline a valid codeword $(\overline{C}_0, \overline{C}_1)$, and then makes two extra tampering queries (at the end of two subsequent rounds, say, i and $i + 1$) such that the first query overwrites (C_0^i, C_1^i) with $(C_0^i, \overline{C}_1)$, and the second query overwrites (C_0^{i+1}, C_1^{i+1}) with $(\overline{C}_0, C_1^{i+1})$; by combining the refreshed codewords obtained as output, the adversary gets a refreshing of the original codeword, which cannot be simulated in the ideal experiment (recall that the refresh algorithm updates the two shares independently).

function $\nu : \mathbb{N} \to [0,1]$ such that, for any polynomial $q(\kappa)$, the following holds:

$$\left|\mathbb{P}\left[\mathbf{Tamper}_{\mathcal{CS},\mathsf{A}}(\kappa,\ell,q)=1\right] - \mathbb{P}\left[\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa,\ell,q)=1\right]\right| \le \nu(\kappa),$$

where the experiments $\mathbf{Tamper}_{\mathcal{CS},\mathsf{A}}(\kappa,\ell,q)$ and $\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa,\ell,q)$ are defined in Fig. 5.

We note that in the **Tamper** security game the adversary does not have "direct" access to a refreshing oracle (namely, an oracle that, upon request, would refresh the codeword). This choice is without loss of any generality, as a call to the refreshing oracle can always be simulated by a tampering query yielding an invalid codeword (which indeed triggers a refreshing in our experiment).

4 Code Construction

Let $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdateC}, \mathsf{UpdateS})$ be a CLRS friendly PKE scheme, with secret-key space $\mathcal{SK}$. Let $\mathcal{COM} = (\mathsf{CRSGen}, \mathsf{Commit})$ be a commitment scheme in the CRS model. Consider the following NP-relations, parameterized by the PKE and the commitment scheme, respectively:

$$\mathcal{R}_0 := \{(pk, sk) : \; pk = \mathsf{PK}(sk), sk \in \mathcal{SK}\},$$
$$\mathcal{R}_1 := \{((\omega,\gamma),(M,r)) : \; \gamma = \mathsf{Commit}(\omega, M; r)\}.$$

Let Φ_0 and Φ_1 be two sets of label transformations defined below:

$$\Phi_0 := \{\phi : \; \exists pk, sk \text{ s.t. } (\forall m, r)\ \mathsf{Dec}(sk, \phi(\mathsf{Enc}(pk, m; r))) = m, pk = \mathsf{PK}(sk)\}$$
$$\Phi_1 := \{\phi : \; (\forall sk)\ \mathsf{PK}(sk) = \mathsf{PK}(\phi(sk))\}.$$

Notice that $\mathcal{R}_0, \mathcal{R}_1, \Phi_0$ and Φ_1 are implicitly parameterized by the public parameters $\rho \in \{0,1\}^*$ of the PKE scheme. Finally, let $\mathcal{U}^0$ and $\mathcal{U}_1$ be the following sets of label transformations:

$$\mathcal{U}^0 := \{\mathsf{UpdateC}(\,\cdot\,; r_u) : \; r_u \in \{0,1\}^*\}$$
$$\mathcal{U}^1 := \{\mathsf{UpdateS}(\,\cdot\,; r_u) : \; r_u \in \{0,1\}^*\}.$$

It is easy to verify that $\mathcal{U}_\beta \subseteq \Phi_\beta$, for $\beta \in \{0,1\}$. In fact, for $\beta = 0$, by the correctness of the PKE scheme, there exists sk such that $\mathbb{P}[\mathsf{Dec}(sk, \mathsf{UpdateC}(\mathsf{Enc}(pk, m))) = m] = 1$ and $pk = \mathsf{PK}(sk)$; similarly, for $\beta = 1$, again by correctness of the PKE scheme, for any $sk' \leftarrow \mathsf{UpdateS}(pk, sk)$ we have that $\mathsf{PK}(sk) = \mathsf{PK}(sk')$.

Scheme description. Let $\mathcal{NIA}_0 = (\mathsf{CRSGen}_0, \mathsf{Prove}_0, \mathsf{Vrfy}_0, \mathsf{LEval}_0)$ and $\mathcal{NIA}_1 = (\mathsf{CRSGen}_1, \mathsf{Prove}_1, \mathsf{Vrfy}_1, \mathsf{LEval}_1)$ be NIAs for the above defined relations $\mathcal{R}_0$ and $\mathcal{R}_1$. Our code $\mathcal{CS} = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ works as follows.

- $\underline{\mathsf{Init}(1^\kappa)}$: For $\beta \in \{0,1\}$, sample $\omega_\beta \leftarrow \mathsf{CRSGen}_\beta(1^\kappa)$, $\omega \leftarrow \mathsf{CRSGen}(1^\kappa)$, and $\rho \leftarrow \mathsf{Setup}(1^\kappa)$. Return $\overline{\omega} = (\omega_0, \omega_1, \omega, \rho)$.
- $\underline{\mathsf{Encode}(\overline{\omega}, M)}$: Parse $\overline{\omega} := (\omega_0, \omega_1, \omega, \rho)$, sample $(pk, sk) \leftarrow \mathsf{KGen}(\rho)$, and $r \leftarrow \{0,1\}^*$. Compute $c \leftarrow \mathsf{Enc}(pk, M||r)$, $\gamma = \mathsf{Commit}(\omega, M; r)$, and $\pi_0 \leftarrow \mathsf{Prove}_0^c(\omega_0, pk, sk)$, and $\pi_1 \leftarrow \mathsf{Prove}_1^{sk}(\omega_1, (\omega, \gamma), (M, r))$. Set $C_0 := (pk, c, \pi_0)$ and $C_1 := (sk, \gamma, \pi_1)$, and return $C := (C_0, C_1)$.
- $\underline{\mathsf{Decode}(\overline{\omega}, C)}$: Parse $\overline{\omega} := (\omega_0, \omega_1, \omega, \rho)$ and $C := (C_0, C_1)$, where $C_1 := (sk, \gamma, \pi_1)$ and $\underline{C_0 = (pk, c, \pi_0)}$. Compute $M||r := \mathsf{Dec}(sk, c)$, and if the following conditions hold return M else return $\perp$:

I. Left check: $\mathsf{Ver}_0^c(\omega_0, pk, \pi_0) = 1$.

II. Right check: $\mathsf{Ver}_1^{sk}(\omega_1, (\omega, \gamma), \pi_1) = 1$.

III. Cross check: $\mathsf{Commit}(\omega, M; r) = \gamma$.

- $\underline{\mathsf{Rfrsh}(\overline{\omega}, (\beta, C_\beta))}$: Parse $\overline{\omega} := (\omega_0, \omega_1, \omega, \rho)$, $C_0 := (pk, c, \pi_0)$, and $C_1 = (sk, \gamma, \pi_1)$. Hence:
  - For $\beta = 0$, pick $r^0_{\text{upd}} \leftarrow \{0,1\}^*$, let $c' := \mathsf{UpdateC}(c; r^0_{\text{upd}})$ and $\pi'_0 \leftarrow \mathsf{LEval}_0(\omega_0, \mathsf{UpdateC}(\cdot; r^0_{\text{upd}}), (pk, c, \pi_0))$, and return $C'_0 := (pk, c', \pi'_0)$.
  - For $\beta = 1$, pick $r^1_{\text{upd}} \leftarrow \{0,1\}^*$, let $sk' := \mathsf{UpdateS}(sk; r^1_{\text{upd}})$, and $\pi'_1 \leftarrow \mathsf{LEval}_1(\omega_1, \mathsf{UpdateS}(\cdot; r^1_{\text{upd}}), ((\gamma, \omega), sk, \pi_1))$, and return $C'_1 := (\gamma, sk', \pi'_1)$.

Theorem 1. *Let $\mathcal{PKE}$ be a PKE scheme with message space $\mathcal{M}_{\text{pke}}$ and public-key space $\mathcal{PK}$, let $\mathcal{COM}$ be a commitment scheme with message space $\mathcal{M}$, and let $\mathcal{NIA}_0$ (resp. $\mathcal{NIA}_1$) be a NIA w.r.t. the relations $\mathcal{R}_0$ (resp. $\mathcal{R}_1$). Define $\mu(\kappa) := \log|\mathcal{M}|$, $\mu_{\text{pke}}(\kappa) := \log|\mathcal{M}_{\text{pke}}|$, and $\delta(\kappa) := \log|\mathcal{PK}|$.*

For any $\ell \in \mathbb{N}$, assuming that $\mathcal{PKE}$ is an $(\ell + 3\mu + 2\kappa + \max\{\delta, \mu_{\text{pke}}\})$-noisy CLRS-friendly PKE scheme (cf. Definition 5), that $\mathcal{COM}$ is a non-interactive statistically binding commitment scheme, and that $\mathcal{NIA}_0$ (resp. $\mathcal{NIA}_1$) satisfies adaptive multi-theorem zero-knowledge (cf. Definition 2), Φ_0-malleable (resp. Φ_1-malleable) label simulation extractability (cf. Definition 3), and label derivation privacy (cf. Definition 4), then the coding scheme $\mathcal{CS}$ described above is an ℓ-leakage-resilient continuously non-malleable code with split-state refresh.

4.1 Proof Intuition

The proof of the above theorem is quite involved. We provide some highlights here. Consider a simulator $(\mathsf{S}_0, \mathsf{S}_1)$, where S_0 simulates a fake CRS $\overline{\omega} = (\omega_0, \omega_1, \omega, \rho)$ by additionally sampling the corresponding zero-knowledge and extraction trapdoors for the NIAs (which are then passed to S_1). At the core of our simulation strategy are two algorithms T^0 and T^1, whose goal is essentially to emulate the outcome of the real tampering experiment, with the important difference that T^0 is only given the left part of a (simulated) codeword C_0 and the left tampering function f_0, whereas T^1 is given (C_1, f_1).

The simulator S_1 then works as follows. Initially, it samples a fresh encoding (C_0, C_1) of 0^μ. More in details, the fresh encoding comes from the (computationally close) distribution where the proofs π_0 and π_1 are simulated proofs. At the beginning of each round, it runs a simulated refresh procedure in which the ciphertext c is updated via $\mathsf{UpdateC}$ (and the simulated proof π_0 is re-computed using fresh randomness), and similarly the secret key sk is updated via $\mathsf{UpdateS}$ (and the simulated proof π_1 is re-computed using fresh randomness). Hence, for each tampering query (f_0, f_1), the simulator S_1 runs $\widetilde{M}_0 := \mathsf{T}^0(C_0, f_0)$, $\widetilde{M}_1 := \mathsf{T}^1(C_1, f_1)$, and it returns $\widetilde{M}_0$ as long as $\bot \neq \widetilde{M}_0 = \widetilde{M}_1 \neq \bot$ (and $\bot$ otherwise). The extra tampering query (f_0^*, f_1^*) is simulated similarly, based on the outcome of the tampering simulators $(\mathsf{T}^0, \mathsf{T}^1)$. We briefly describe the tampering simulators T^0 and T^1:

- Algorithm T^0 lets $f_0(C_0) := (\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0)$. If the proof $\widetilde{\pi}_0$ does not verify, it returns $\bot$. Else, if $(\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0) = (pk, c, \pi_0)$, it returns $\diamond$. Else, it extracts the proof $\widetilde{\pi}_0$, this leads to two possible outcomes:[8]

  (a) The extractor outputs a secret key $\widehat{sk}$ which is used to decrypt $\widetilde{c}$, and the tampering simulator returns the corresponding plaintext $\widetilde{M}$.

[8]The above description is simplified, in that extraction could potentially fail, however, this happens only with negligible probability when the proof verifies correctly.

(b) The extractor outputs a transformation ϕ which maps the label of the simulated proof π_0, namely the encryption of 0^μ, to $\widetilde{c}$. In this case the tampering function f_0 has modified the original ciphertext c to the mauled ciphertext $\widetilde{c}$ which is an encryption of the same message, so the tampering simulator returns $\diamond$.

- Algorithm T^1 lets $f_1(C_1) := (\widetilde{\gamma}, \widetilde{sk}, \widetilde{\pi}_1)$. If the proof $\widetilde{\pi}_1$ does not verify, it returns $\perp$. Else, if $(\widetilde{\gamma}, \widetilde{sk}, \widetilde{\pi}_1) = (\gamma, sk, \pi_1)$, it returns $\diamond$. Else, it extracts the proof $\widetilde{\pi}_1$, again, this leads to two possible outcomes:

  (a) The extractor outputs the committed message $\widetilde{M}$ (along with the randomness of the commitment), so the tampering simulator can simply return $\widetilde{M}$.

  (b) The extractor outputs a transformation ϕ which maps the label of the simulated proof π_1, namely the original secret key sk, to the mauled secret key $\widetilde{sk}$. In this case, the mauled proof $\widetilde{\pi}_1$ must be a valid proof whose instance is the original commitment, and so, once again, the tampering simulator returns $\diamond$.

To show that the above simulator indeed works, we use a hybrid argument where we incrementally change the distribution of the ideal tampering experiment until we reach the distribution of the real tampering experiment. Each step introduces a negligible error, thanks to the security properties of the underlying building blocks. Perhaps, the most interesting step is the one where we switch the ciphertext c from an encryption of zero to an encryption of the real message (to which we always have to append the randomness of the commitment); in order to show that this change is unnoticeable, we rely on the CLRS storage friendly security of the PKE scheme. In particular, this step of the proof is based on the following observations:

- The reduction can perfectly emulate the distribution of the CRS $\overline{\omega}$, and of all the elements $(pk, \pi_0, \gamma, \pi_1)$, except for (c, sk). However, by outputting $(0^\mu || r, M || r)$ as challenge plaintexts—where $r \in \{0,1\}^*$ is the randomness for the commitment—the reduction can obtain independent leakages from C_0 and C_1 with the right distribution.

- Refresh of codewords can also be emulated by exploiting the fact that the reduction is allowed to update the challenge secret key and ciphertext.

- The reduction can answer tampering queries from the adversary by using T^0 and T^1 as leakage functions. The main obstacle is to ensure that T^0 and T^1 are ℓ-leaky, where $\ell \in \mathbb{N}$ is the leakage bound tolerated by the PKE scheme. In particular, between each refresh, the reduction needs to interleave the executions of T^0 and T^1 until their outputs diverge. Let i^* be the number of tampering queries that the simulator performs until triggering a decoding error. The leakage that the reduction needs to perform during this stage (namely, between two consecutive refreshes) is $\mathsf{T}^0(C_0, f_0^0), \mathsf{T}^1(C_1, f_1^0), \ldots, \mathsf{T}^0(C_0, f_0^{i^*}), \mathsf{T}^1(C_1, f_1^{i^*})$ where $(f_0^0, f_1^0), \ldots, (f_0^{i^*}, f_1^{i^*})$ is the list of tampering functions chosen by the adversary. We have:

$$\begin{aligned}&\widetilde{\mathbb{H}}_\infty(C_0 |\ \mathsf{T}^0(C_0, f_0^0), \ldots, \mathsf{T}^0(C_0, f_0^{i^*})) = \\ &\widetilde{\mathbb{H}}_\infty(C_0 |\ \mathsf{T}^1(C_1, f_1^0), \ldots, \mathsf{T}^1(C_1, f_1^{i^*-1}), \mathsf{T}^0(C_0, f_0^{i^*})).\end{aligned}$$

  In fact, the output of $\mathsf{T}^0(C_0, f_0^i)$ and $\mathsf{T}^0(C_0, f_0^i)$ is, by definition, exactly the same whenever $i < i^*$. Moreover, since the output of a function cannot be more informative than its own inputs,

$$\begin{aligned}&\widetilde{\mathbb{H}}_\infty(C_0 |\ \mathsf{T}^1(C_1, f_1^0), \ldots, \mathsf{T}^1(C_1, f_1^{i^*-1}), \mathsf{T}^0(C_0, f_0^{i^*})) \geq \\ &\widetilde{\mathbb{H}}_\infty(C_0 |\ C_1, i^*, \mathsf{T}^0(C_0, f_0^{i^*})).\end{aligned}$$

Lastly, we will use the fact that C_1 reveals only little information about C_0 (a requirement met by known schemes), and further that i^* and $\mathsf{T}^0(C_0, f_0^{i^*})$ can decrease the min-entropy of C_0 of at most their size, which is $O(\kappa)$. The reduction, therefore, defines a valid adversary in the CLRS storage-friendly security experiment of the PKE.

Remark 1 (On the refreshing procedure)**.** *The feature of split-state refresh does not require that a refreshed codeword be indistinguishable from a freshly sampled one. And, indeed, this is not the case in our construction, as the public key pk (resp. the commitment γ) do not change after the refreshing algorithms are executed. However, the latter property is not required for our proof, as the only thing that matters is that the information about the target codeword that an adversary gathers before a refresh takes place will not be useful after the refresh. Put differently, the adversary could potentially leak the entire values pk and γ, but this information would not be useful for breaking the security of our scheme.*

4.2 Description of the Simulator

Let $\mathsf{S}^0 := (\mathsf{S}_0^0, \mathsf{S}_1^0)$ and $\mathsf{S}^1 := (\mathsf{S}_0^1, \mathsf{S}_1^1)$ be the zero-knowledge simulators for the NIAs $\mathcal{NIA}_0$ and $\mathcal{NIA}_1$, respectively. We start by defining the simulator S_0, whose goal is to simulate the CRS generation for the encoding scheme. Briefly, S_0 samples the keys for the PKE scheme and the commitment scheme, and generates an encoding of a dummy message by using simulated proofs in place of real proofs.

Simulator $\mathsf{S}_0(1^\kappa)$:

- For $\beta \in \{0,1\}$ sample $(\omega_\beta, \tau_{\text{sim}}^\beta, \tau_{\text{ext}}^\beta) \leftarrow \mathsf{S}_0^\beta(1^\kappa)$.
- Pick $\omega \leftarrow \mathsf{CRSGen}(1^\kappa)$ and $\rho \leftarrow \mathsf{Setup}(1^\kappa)$.
- Define $\tau^* := (\tau_{\text{sim}}^0, \tau_{\text{ext}}^0, \tau_{\text{sim}}^1, \tau_{\text{ext}}^1)$.
- Sample $r \leftarrow \{0,1\}^*$, let $\gamma := \mathsf{Commit}(\omega, 0^\mu; r)$, and compute $c \leftarrow \mathsf{Enc}(pk, 0^\mu \| r)$, where $\mu \in \mathbb{N}$ is the bit-length associated to the message space $\mathcal{M}$ for the commitment scheme.
- Let $C := ((pk, c, \pi_0), (\gamma, sk, \pi_1)) := (C_0, C_1)$, where $\pi_0 \leftarrow \mathsf{S}_1^0(\tau_{\text{sim}}^0, c, pk)$ and $\pi_1 \leftarrow \mathsf{S}_1^1(\tau_{\text{sim}}^1, sk, (\gamma, \omega))$.
- Output $\overline{\omega} := (\omega_0, \omega_1, \omega, \rho)$ and auxiliary information $\alpha := (\tau^*, (C_0, C_1))$.

We also define a simulated refresh algorithm, which will be used in the simulation in place of the original refresh mechanism.

Algorithm $\mathsf{Rfrsh}^*((\tau_{\text{sim}}^0, \tau_{\text{sim}}^1), (\beta, C_\beta))$:

- For $\beta = 0$, parse $C_0 := (c, pk, \pi_0)$, compute $c' \leftarrow \mathsf{UpdateC}(c)$ and $\pi_0' \leftarrow \mathsf{S}_1^0(\tau_{\text{sim}}^0, c', pk)$, and return $C_0' := (pk, c', \pi_0')$.
- For $\beta = 1$, parse $C_1 := (\gamma, sk, \pi_1)$, compute $sk' \leftarrow \mathsf{UpdateS}(sk)$, and $\pi_1' \leftarrow \mathsf{S}_1^1(\tau_{\text{sim}}^1, sk', (\gamma, \omega))$, and return $C_1' := (\gamma, sk', \pi_1')$.

```
The Tampering Simulators

Algorithm $\mathsf{T}^0(C_0, f_0; r^0_{\text{ext}})$:
  1. Parse $C_0 := (pk, c, \pi_0)$.
  2. Compute $\widetilde{C}_0 = f_0(C_0) := (\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0)$. Hence:
     (a) If $\mathsf{Vrfy}_0^{\widetilde{c}}(\omega_0, \widetilde{pk}, \widetilde{\pi}_0) = 0$, set $\widetilde{M} := \perp$.
     (b) If $(\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0) = (pk, c, \pi_0)$, set $\widetilde{M} = \diamond$.
     (c) Else, run $(\widehat{sk}, \phi_0, \widehat{c}) := \mathsf{K}_0(\tau^0_{\text{ext}}, \widetilde{c}, \widetilde{pk}, \widetilde{\pi}_0; r^0_{\text{ext}})$.
     (d) Output **Abort** in case either of the following happens:
         i. $\widehat{sk} \neq \perp$ and $\widetilde{pk} \neq \mathsf{PK}(\widehat{sk})$; or
         ii. $(\phi_0, \widehat{c}) \neq (\perp, \perp)$ and $(\phi_0(\widehat{c}) \neq \widetilde{c}) \vee (\phi_0 \notin \Phi_0)$; or
         iii. $(\widehat{sk}, \phi_0, \widehat{c}) = (\perp, \perp, \perp)$.
     (e) If $\widehat{sk} \neq \perp$ and $\widetilde{pk} = \mathsf{PK}(\widehat{sk})$, let $\widetilde{M} \| \widetilde{r} := \mathsf{Dec}(\widehat{sk}, \widetilde{c})$.
     (f) If $(\phi_0, \widehat{c}) \neq (\perp, \perp)$ and $(\phi_0(\widehat{c}) = \widetilde{c}) \wedge (\phi_0 \in \Phi_0)$, and $\widehat{c} = c$, let $\widetilde{M} := \diamond$.
  3. Return $\widetilde{M}$.

Algorithm $\mathsf{T}^1(C_1, f_1; r^1_{\text{ext}})$:
  1. Parse $C_1 := (\gamma, sk, \pi_1)$.
  2. Compute $\widetilde{C}_1 = f_1(C_1) := (\widetilde{\gamma}, \widetilde{sk}, \widetilde{\pi}_1)$. Hence:
     (a) If $\mathsf{Vrfy}_1^{\widetilde{sk}}(\omega_1, (\widetilde{\gamma}, \omega), \widetilde{\pi}_1) = 0$, set $\widetilde{M} := \perp$.
     (b) If $(\widetilde{\gamma}, \widetilde{sk}, \widetilde{\pi}_1) = (\gamma, sk, \pi_1)$, set $\widetilde{M} = \diamond$.
     (c) Else, run $(\widehat{M} \| \widehat{r}, \phi_1, \widehat{sk}) := \mathsf{K}_1(\tau^1_{\text{ext}}, \widetilde{sk}, (\widetilde{\gamma}, \omega), \widetilde{\pi}_1; r^1_{\text{ext}})$.
     (d) Output **Abort** in case either of the following happens:
         i. $\widehat{M} \| \widehat{r} \neq \perp$ and $\widetilde{\gamma} \neq \mathsf{Commit}(\omega, \widehat{M}; \widehat{r})$; or
         ii. $(\phi_1, \widehat{sk}) \neq (\perp, \perp)$ and $(\phi_1(\widehat{sk}) \neq \widetilde{sk}) \vee (\phi_1 \notin \Phi_1)$; or
         iii. $(\widehat{M} \| \widehat{r}, \phi_1, \widehat{sk}) = (\perp, \perp, \perp)$.
     (e) If $\widehat{M} \| \widehat{r} \neq \perp$ and $\widetilde{\gamma} = \mathsf{Commit}(\omega, \widehat{M}; \widehat{r})$, let $\widetilde{M} := \widehat{M}$.
     (f) If $(\phi_1, \widehat{sk}) \neq (\perp, \perp)$ and $(\phi_1(\widehat{sk}) = \widetilde{sk}) \wedge (\phi_1 \in \Phi_1)$, and $\widehat{sk} = sk$, let $\widetilde{M} := \diamond$.
  3. Return $\widetilde{M}$.
```

Figure 6: The tampering simulators T^0 and T^1.

The tampering simulators. We now turn to defining the simulator S_1. To facilitate the description, in Fig. 6, we formalize two algorithms that we call the *tampering simulators*. Intuitively, for $\beta \in \{0, 1\}$, algorithm T^β simulates the outcome corresponding to a tampering query (f_0, f_1) from the adversary, by using only C_β (i.e., one piece of the simulated codeword). On a very high level, as long as both T^0 and T^1 return the same message $\widetilde{M}$, the simulator S_1 uses $\widetilde{M}$ to answer the tampering queries from the adversary; otherwise, in case T^0 and T^1 disagree, it emulates a decoding error.

Simulator S_1:

1. Parse $\alpha := ((\tau^0_{\text{sim}}, \tau^0_{\text{ext}}, \tau^1_{\text{sim}}, \tau^1_{\text{ext}}), (C_0, C_1)) := (\tau^*, C)$, and run $\mathsf{Rfrsh}^*((\tau^0_{\text{sim}}, \tau^1_{\text{sim}}), C)$. Set the flag $\texttt{err} \leftarrow 0$.

2. Hence, upon a command from the adversary, behave as follows.

   (a) Upon input a leakage query $(\texttt{Leak}, \beta, g_\beta)$, for $\beta \in \{0,1\}$, return $y := g_\beta(C_\beta)$.

   (b) Upon input a tampering query $(\texttt{Tamp}, (f_0, f_1))$, if $\texttt{err} = 1$ return $\perp$. Otherwise, let $\widetilde{M}_0 \leftarrow \mathsf{T}^0(C_0, f_0)$ and $\widetilde{M}_1 := \mathsf{T}^1(C_1, f_1)$. If either of the tampering simulators returned **Abort**, abort the simulation outputting the special value **Abort**. If either $\widetilde{M}_0 = \perp$, or $\widetilde{M}_1 = \perp$, return $\perp$ and set $\texttt{err} \leftarrow 1$; else, return $\widetilde{M} := \widetilde{M}_0 = \widetilde{M}_1$.

   (c) Upon input the extra tampering query $(\texttt{Final}, (f^*_0, f^*_1))$, let $\widetilde{M}^*_0 := \mathsf{T}^0(C_0, f^*_0; r^0_{\text{ext}})$ and $\widetilde{M}^*_1 := \mathsf{T}^1(C_1, f^*_1; r^1_{\text{ext}})$, for random $r^0_{\text{ext}}, r^1_{\text{ext}} \leftarrow \{0,1\}^*$. If either of the tampering simulators returned **Abort**, abort the simulation outputting the special value **Abort**. If either $\widetilde{M}^*_0 \neq \widetilde{M}^*_1$, or $\widetilde{M}^*_0 = \perp$, or $\widetilde{M}^*_1 = \perp$, return $\perp$. Otherwise, if $\widetilde{M}^*_0 = \widetilde{M}^*_1 = \diamond$, return $\diamond$. Else, proceed as follows:

      i. Let $\widetilde{C}^*_0 := f^*_0(C_0) := (\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0)$ and $\widetilde{C}^*_1 := f^*_1(C_1) := (\widetilde{\gamma}, \widetilde{sk}, \widetilde{\pi}_1)$.

      ii. Extract the values $(\widehat{sk}_0, \phi_0, \widehat{c}_0) := \mathsf{K}_0(\tau^0_{\text{ext}}, \widetilde{c}, \widetilde{pk}, \widetilde{\pi}_0; r^0_{\text{ext}})$, and let $sk' = \mathsf{UpdateS}(\widehat{sk}_0)$.

      iii. Extract the values $((\widehat{M}, \widehat{r}), \phi_1, \widehat{sk}_1) := \mathsf{K}_1(\tau^1_{\text{ext}}, \widetilde{sk}, (\widetilde{\gamma}, \omega), \widetilde{\pi}_1; r^1_{\text{ext}})$, and compute $c' \leftarrow \mathsf{Enc}(\widetilde{pk}, \widehat{M} \| \widehat{r})$.[9]

      iv. Run $\widetilde{\pi}'_0 \leftarrow \mathsf{S}^0_1(\tau^0_{\text{sim}}, c', \widetilde{pk})$ and $\widetilde{\pi}'_1 \leftarrow \mathsf{S}^1_1(\tau^1_{\text{sim}}, sk', (\widetilde{\gamma}, \omega))$.

      v. Output $\widetilde{C}' := ((\widetilde{pk}, c', \widetilde{\pi}'_0), (\widetilde{\gamma}, sk', \widetilde{\pi}'_1))$.

3. Go back to step 1 until the adversary is done with its queries.

4.3 Hybrids

We define a sequence of games, starting with the simulated experiment $\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa, \ell, q)$ for the above defined simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$. The games are described in an incremental manner, meaning that for each game we only highlight the differences with the previous game, implicitly assuming that all other steps are unchanged. Each game is parameterized by the adversary A and, implicitly, by a simulator that is typically derived by the original simulator S.

Game $\mathbf{G}^0_{\mathsf{A}}(\kappa)$. This game is identical to $\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa, \ell, q)$, for the above defined simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$.

Game $\mathbf{G}^1_{\mathsf{A}}(\kappa)$. The simulator S_0 is additionally given M as input, and computes the commitment in the simulated codeword (C_0, C_1) as $\gamma = \mathsf{Commit}(\omega, M; r)$, for $r \leftarrow \{0,1\}^*$.

Game $\mathbf{G}^2_{\mathsf{A}}(\kappa)$. The simulator S_0 computes the ciphertext in the simulated codeword (C_0, C_1) as $c \leftarrow \mathsf{Enc}(pk, M \| r)$.

Game $\mathbf{G}^3_{\mathsf{A}}(\kappa)$. The simulator S_1 keeps track of all the ciphertexts and secret keys contained in each dummy encoding computed in step 1. Namely, for each $i \in [q]$, it lets $\alpha \leftarrow \alpha \| (c_i, sk_i)$, where c_i, sk_i are, respectively, the ciphertext and the secret key as obtained by refreshing the initial encoding C using procedure Rfrsh^*.

[9]Note that the extractors K_0 and K_1 are run, respectively, using the same random coins on which the tampering simulators T^0 and T^1 are run.

Game $\mathbf{G}^4_{\mathsf{A}}(\kappa)$. For each round $i \in [q]$, the tampering simulators T^0 and T^1 are modified as follows (cf. Fig. 6):

$\mathsf{T}^0 = (C_0, f_0)$:

- The check at step 2(d)ii becomes $(\phi_0, \widehat{c}) \neq (\bot, \bot)$ and $(\phi_0(\widehat{c}) \neq \widetilde{c}) \vee (\phi_0 \notin \Phi_0) \vee (\widehat{c} \notin \{c_1, \dots, c_{i-1}\})$.
- The simulator additionally sets $\widetilde{M} = \diamond$ at step 2b if $\widetilde{c} \in \{c_1, \dots, c_{i-1}\}$.

$\mathsf{T}^1 = (C_1, f_1)$:

- The check at step 2(d)ii becomes $(\phi_1, \widehat{sk}) \neq (\bot, \bot)$ and $(\phi_1(\widehat{sk}) \neq \widetilde{sk}) \vee (\phi_1 \notin \Phi_1) \vee (\widehat{sk} \notin \{sk_1, \dots, sk_{i-1}\})$.
- The simulator additionally sets $\widetilde{M} = \diamond$ at step 2b if $\widetilde{sk} \in \{sk_1, \dots, sk_{i-1}\}$.

Game $\mathbf{G}^5_{\mathsf{A}}(\kappa)$. We change the way tampering queries for which the simulator returns $\diamond$ are treated. In particular,

- For each query $(\texttt{Tamp}, (f_0, f_1))$ such that S_1 outputs $\widetilde{M} = \diamond$, we treat this query exactly as in the real experiment (i.e., we apply (f_0, f_1) to the target encoding and decode the resulting codeword).
- For each query $(\texttt{Final}, (f_0^*, f_1^*))$ such that S_1 outputs $\diamond$, we treat this query exactly as in the real experiment (i.e., we apply (f_0^*, f_1^*) to the target encoding and decode the resulting codeword).

Game $\mathbf{G}^6_{\mathsf{A}}(\kappa)$. We change the way tampering queries for which the simulator returns a value which is not in $\{\bot, \diamond\}$ are treated. In particular,

- For each query $(\texttt{Tamp}, (f_0, f_1))$ such that S_1 outputs $\widetilde{M} = \widetilde{M}_0 = \widetilde{M}_1$, we treat this query exactly as in the real experiment (i.e., we apply (f_0, f_1) to the target encoding and decode the resulting codeword).
- For each query $(\texttt{Final}, (f_0^*, f_1^*))$ such that S_1 outputs a codeword $\widetilde{C}'$ that is valid but does not decode to M, we treat this query exactly as in the real experiment (i.e., we apply (f_0^*, f_1^*) to the target encoding and decode the resulting codeword).

Game $\mathbf{G}^7_{\mathsf{A}}(\kappa)$. This game is identical to the previous one, except that the tampering simulators T^0 and T^1 do not perform the check in step 2d, i.e. in game $\mathbf{G}^7_{\mathsf{A}}(\kappa)$ the tampering simulators never output **Abort** (and so does S_1).

Game $\mathbf{G}^8_{\mathsf{A}}(\kappa)$. We change the way tampering queries for which the simulator returns $\bot$ are treated. In particular,

- For each query $(\texttt{Tamp}, (f_0, f_1))$ such that S_1 outputs $\widetilde{M} = \bot$, we treat this query exactly as in the real experiment (i.e., we apply (f_0, f_1) to the target encoding and decode the resulting codeword).
- For each query $(\texttt{Final}, (f_0^*, f_1^*))$ such that S_1 outputs $\bot$, we treat this query exactly as in the real experiment (i.e., we apply (f_0^*, f_1^*) to the target encoding and decode the resulting codeword).

Game $\mathbf{G}^9_{\mathsf{A}}(\kappa)$. We change the way the codeword $\widetilde{C}_0^*$ corresponding to the extra tampering query is computed in each round. Namely, instead of running the extractor K_0 and letting $sk' \leftarrow \mathsf{UpdateS}(\widehat{sk})$ (where $\widehat{sk}$ is the extracted secret key), we directly let $sk' \leftarrow \mathsf{UpdateS}(\widetilde{sk})$.

Game $\mathbf{G}_{\mathsf{A}}^{10}(\kappa)$. We change the way the codeword $\widetilde{C}_1^*$ corresponding to the extra tampering query is computed in each round. Namely, instead of running the extractor K_1 and letting $c' \leftarrow \mathsf{Enc}(pk, \widehat{M}\|\widehat{r})$ (where $\widehat{M}\|\widehat{r}$ is the extracted message/randomness), we directly let $c' \leftarrow \mathsf{UpdateC}(\widetilde{c})$.

Game $\mathbf{G}_{\mathsf{A}}^{11}(\kappa)$. We setup the CRS differently. Namely, we change the distribution of ω_0 as in $\omega_0 \leftarrow \mathsf{CRSGen}_0(1^\kappa)$. Moreover, the simulator S_1 always runs the prover algorithm Prove_0 instead of the zero-knowledge simulator S_1^0. (Observe that this happens in three different steps: (i) When the original codeword (C_0, C_1) is computed; (ii) Whenever S_1 runs algorithm Rfrsh^* at the beginning of each round; (iii) Whenever S_1 needs to answer the extra tampering query.)

Game $\mathbf{G}_{\mathsf{A}}^{12}(\kappa)$. We setup the CRS differently. Namely, we change the distribution of ω_1 as in $\omega_1 \leftarrow \mathsf{CRSGen}_1(1^\kappa)$. Moreover, the simulator S_1 always runs the prover algorithm Prove_1 instead of the zero-knowledge simulator S_1^1. (Once again, this happens in the exact same cases as in the previous game.)

Game $\mathbf{G}_{\mathsf{A}}^{13}(\kappa)$. We change the way a codeword is refreshed. Namely, instead of computing $\pi_0' \leftarrow \mathsf{Prove}_0^{c'}(\omega_0, pk, sk)$ via the prover algorithm, we now run the label evaluation procedure as done by the real refresh algorithm. The same modification affects the way the proof π_0' is computed during each of the extra tampering queries (f_0^*, f_1^*).

Game $\mathbf{G}_{\mathsf{A}}^{14}(\kappa)$. We change the way a codeword is refreshed. Namely, instead of computing $\pi_1' \leftarrow \mathsf{Prove}_1^{sk'}(\omega_1, (\omega, \gamma), (M, r))$ via the prover algorithm, we now run the label evaluation procedure as done by the real refresh algorithm. The same modification affects the way the proof π_1' is computed during each of the extra tampering queries (f_0^*, f_1^*).

Notice that in game $\mathbf{G}_{\mathsf{A}}^{14}(\kappa)$, the "simulator" S_1 computes an encoding of the message M. Moreover, tampering queries are always answered by decoding the modified codeword, and codewords are refreshed exactly as in the real experiment. Thus, $\mathbf{G}_{\mathsf{A}}^{14}(\kappa)$ and $\mathbf{Tamper}_{\mathcal{CS},\mathsf{A}}(\kappa, \ell, q)$ are identically distributed.

4.4 Indistinguishability of the Hybrids

We now prove that the above defined games are computationally indistinguishable. Together with the fact that the last game is identically distributed to the real tampering experiment, this implies Theorem 1.

Lemma 3. *For all PPT adversaries* A *there exists a negligible function* $\nu_{0,1} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{0}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{1}(\kappa) = 1\right]\right| \leq \nu_{0,1}(\kappa)$.

Proof sketch. The proof is down to the hiding property of the commitment scheme (cf. Definition 8). Since the reduction is straightforward, we only highlight the main ideas here and leave the details to the reader.

Assume there exists a PPT adversary A and a polynomial $p_{0,1}(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, we have $\left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{0}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{1}(\kappa) = 1\right]\right| \geq 1/p_{0,1}(\kappa)$. We construct a PPT adversary B (with black-box access to A), that is playing either $\mathbf{G}_{\mathcal{COM},\mathsf{B}}^{\text{hide}}(\kappa, 0)$ or $\mathbf{G}_{\mathcal{COM},\mathsf{B}}^{\text{hide}}(\kappa, 1)$. Roughly, B works as follows:

- Receive ω from the challenger. Compute $\overline{\omega}$ as $\mathsf{S}_0(1^\kappa)$ would do it, except that the CRS for the commitment scheme is set to be equal to the CRS ω from the game.

- Run $\mathsf{A}_0(\overline{\omega})$, obtaining a message M. Forward $(0^\mu, M)$ to the challenger, receiving back a commitment γ.
- Sample the simulated codeword (C_0, C_1) exactly as S_0 would do it, except that the commitment is set to be equal to γ from the game.
- Continue running A and answer its queries exactly as S_1 would do it.
- Return the same output as that of A.

It is immediate to see that if B is playing $\mathbf{G}^{\text{hide}}_{\mathcal{COM},\mathsf{B}}(\kappa, 0)$, then the view of A when running as a sub-routine of B is identical to the view in game $\mathbf{G}^0_{\mathsf{A}}(\kappa)$; similarly, if B is playing $\mathbf{G}^{\text{hide}}_{\mathcal{COM},\mathsf{B}}(\kappa, 1)$, then the view of A when running as a sub-routine of B is identical to the view in game $\mathbf{G}^1_{\mathsf{A}}(\kappa)$. Hence, B retains the same advantage of A, concluding the proof of the lemma. $\square$

Lemma 4. *For all PPT adversaries* A *there exists a negligible function* $\nu_{1,2} : \mathbb{N} \to [0, 1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}^1_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^2_{\mathsf{A}}(\kappa) = 1\right]\right| \le \nu_{1,2}(\kappa)$.

Proof. The proof is down to the fact that $\mathcal{PKE}$ is an ℓ-noisy CLRS friendly PKE scheme. By contradiction, assume there exists a PPT adversary A and a polynomial $p_{1,2}(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, we have $\left|\mathbb{P}\left[\mathbf{G}^1_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^2_{\mathsf{A}}(\kappa) = 1\right]\right| \ge 1/p_{1,2}(\kappa)$. We construct a PPT adversary B (with black-box access to A), that is playing game $\mathbf{G}^{\text{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa, \ell'')$, with $\ell'' := \ell' - 2\kappa$, and ℓ' as in the statement of Theorem 1. A formal description of B follows.

<u>Adversary B:</u>

1. Receive (ρ, pk) from the challenger. For all $\beta \in \{0, 1\}$, sample $(\omega_\beta, \tau^\beta_{\text{sim}}, \tau^\beta_{\text{ext}}) \leftarrow \mathsf{S}^\beta_0(1^\kappa)$, and $\omega \leftarrow \mathsf{CRSGen}(1^\kappa)$. Let $\overline{\omega} = (\omega_0, \omega_1, \omega, \rho)$ and $\tau^* = (\tau^0_{\text{sim}}, \tau^0_{\text{ext}}, \tau^1_{\text{sim}}, \tau^1_{\text{ext}})$.
2. Run $\mathsf{A}_0(\overline{\omega})$, obtaining a message $M \in \mathcal{M}$. Forward $(0^\mu \| r, M \| r)$ to the challenger, for $r \leftarrow \{0, 1\}^*$, and let $\gamma := \mathsf{Commit}(\omega, M; r)$.
3. Repeat the following, until A is done with its queries.
   (a) Query $\mathsf{Update}(0)$ and $\mathsf{Update}(1)$, sample uniform coins $r^0, r^1 \leftarrow \{0, 1\}^*$, and set the flag $\mathtt{err} \leftarrow 0$.
   (b) Upon input $(\mathtt{Leak}, \beta, g_\beta)$, forward (β, g_β) to the target leakage oracle, receiving back a value y; send y to A.
   (c) Upon input $(\mathtt{Tamp}, (f_0, f_1))$, answer this query exactly as S_1 would do, except that the values $\widetilde{M}_0$ and $\widetilde{M}_1$ are obtained, respectively, via queries $(0, g_0)$ and $(1, g_1)$ to the target leakage oracle, where the functions g_0, g_1 are defined as follows:
   $$g_0(\cdot) := \mathsf{T}^0((pk, \cdot, \mathsf{S}^0_1(\tau^0_{\text{sim}}, \cdot, pk; r^0), f_0) \tag{1}$$
   $$g_1(\cdot) := \mathsf{T}^1((\gamma, \cdot, \mathsf{S}^1_1(\tau^1_{\text{sim}}, \cdot, (\gamma, \omega); r^1), f_1). \tag{2}$$
   (Notice that the above might eventually lead to set the flag $\mathtt{err} \leftarrow 1$.)
   (d) Upon input $(\mathtt{Final}, (f^*_0, f^*_1))$, answer this query exactly as S_1 would do, except for the following differences.
      i. The values $\widetilde{M}^*_0, \widetilde{M}^*_1$ are obtained via leakage queries $(0, g_0)$, $(1, g_1)$, where the functions g_0, g_1 are defined as in Eq. (1) and Eq. (2) but with f_0 replaced by f^*_0 and f_1 replaced by f^*_1.

ii. The values $\widetilde{pk}, sk'$ (as part of the simulated codeword $\widetilde{C}'$) are obtained via a leakage query $(0, g_0^*)$, where the function $g_0^*(c)$ hard-wires values $(pk, \tau_{\mathrm{sim}}^0, r^0, \tau_{\mathrm{ext}}^0, r_{\mathrm{ext}}^0)$ and behaves as follows: Compute $\widetilde{C}_0^* := f_0^*(pk, c, \mathsf{S}_1^0(\tau_{\mathrm{sim}}^0, c, pk; r^0)) := (\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0)$, extract the values $(\widehat{sk}_0, \phi_0, \widehat{c}_0) := \mathsf{K}_0(\tau_{\mathrm{ext}}^0, \widetilde{c}, \widetilde{pk}, \widetilde{\pi}_0; r_{\mathrm{ext}}^0)$, and output $(\widetilde{pk}, sk')$, where $sk' \leftarrow \mathsf{UpdateS}(\widehat{sk}_0)$.

iii. The values $\widetilde{\gamma}, c'$ (as part of the simulated codeword $\widetilde{C}'$) are obtained via a leakage query $(1, g_1^*)$, where the function $g_1^*(sk)$ hard-wires values $(\omega, \gamma, \tau_{\mathrm{sim}}^1, r^1, \tau_{\mathrm{ext}}^1, r_{\mathrm{ext}}^1)$ and behaves as follows: Compute $\widetilde{C}_1^* := f_1^*(\gamma, sk, \mathsf{S}_1^1(\tau_{\mathrm{sim}}^1, sk, (\omega, \gamma); r^1)) := (\widetilde{\gamma}, \widetilde{sk}, \widetilde{\pi}_1)$, extract the values $((\widehat{M}, \widehat{r}), \phi_1, \widehat{sk}_1) := \mathsf{K}_1(\tau_{\mathrm{ext}}^1, \widetilde{sk}, (\widetilde{\gamma}, \omega), \widetilde{\pi}_1; r_{\mathrm{ext}}^1)$, and output $(\widetilde{\gamma}, c')$, where $c' \leftarrow \mathsf{Enc}(\widetilde{pk}, \widehat{M} \| \widehat{r})$.

4. Output the same as A does.

Notice that, assuming the leakage bound is not violated, adversary B perfectly simulates the view of A; namely, if the challenge bit is $b = 0$, the view of A is identically distributed to that of game $\mathbf{G}_{\mathsf{A}}^1$, and if the challenge bit is $b = 1$, the view of A is identically distributed to that of game $\mathbf{G}_{\mathsf{A}}^2$. Thus,

$$\begin{aligned}
1/p_{1,2}(\kappa) &\le \left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^1(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^2(\kappa) = 1\right]\right| \\
&= \left|\mathbb{P}\left[\mathbf{G}_{\mathcal{PKE},\mathsf{B}}^{\mathrm{clrs}}(\kappa, \ell'') = 1 |\ b = 1\right] - \mathbb{P}\left[\mathbf{G}_{\mathcal{PKE},\mathsf{B}}^{\mathrm{clrs}}(\kappa, \ell'') = 0 |\ b = 0\right]\right| \\
&= 2\left|\mathbb{P}\left[\mathbf{G}_{\mathcal{PKE},\mathsf{B}}^{\mathrm{clrs}}(\kappa, \ell'') = 1\right] - \frac{1}{2}\right|,
\end{aligned}$$

contradicting the fact that $\mathcal{PKE}$ is a CLRS friendly PKE scheme.

It remains to show that the adversary B does not violate the leakage bound as stated in Theorem 1. To this end, assume that, between each update, the adversary A makes at most $t \in \mathrm{poly}(\kappa)$ tampering queries (not counting the final extra query (f_0^*, f_1^*)), and let $i^* \in [t]$ be the first index for which we have $\widetilde{M}_{0,i^*} := \mathsf{T}^0(C_0, f_0^{i^*}) \neq \mathsf{T}^1(C_1, f_1^{i^*}) := \widetilde{M}_{1,i^*}$. Define

$$Z := (Z_0, Z_1) := ((Y_0, \widetilde{M}_{0,1}, \ldots \widetilde{M}_{0,i^*}, \widetilde{M}_0^*), (Y_1, \widetilde{M}_{1,1}, \ldots, \widetilde{M}_{1,i^*}, \widetilde{M}_1^*))$$

to be the random variable corresponding to the partial view of A in each epoch, up until step 3c included, which consists of the leakage Y_0 done on C_0, the leakage Y_1 done on C_1, and the messages $\widetilde{M}_{0,1}, \ldots \widetilde{M}_{0,i^*}, \widetilde{M}_0^*$ and $\widetilde{M}_{1,1}, \ldots \widetilde{M}_{1,i^*}, \widetilde{M}_1^*$ as defined by the tampering simulators T^0 and T^1. For $\beta \in \{0, 1\}$, let X_β be the random variable corresponding to C_β conditioned on the value pk, and notice that, after each update (performed by the reduction in step 3a), X_0 and X_1 are independent. We can write:

$$\begin{aligned}
\widetilde{\mathbb{H}}_\infty(X_\beta | Z_\beta) &= \widetilde{\mathbb{H}}_\infty(X_\beta |\ Y_\beta, \widetilde{M}_{\beta,1}, \ldots, \widetilde{M}_{\beta,i^*-1}, \widetilde{M}_{\beta,i^*}, \widetilde{M}_\beta^*) && (3) \\
&= \widetilde{\mathbb{H}}_\infty(X_\beta |\ Y_\beta, \widetilde{M}_{1-\beta,1}, \ldots, \widetilde{M}_{1-\beta,i^*-1}, \widetilde{M}_{\beta,i^*}, \widetilde{M}_\beta^*) && (4) \\
&\ge \widetilde{\mathbb{H}}_\infty(X_\beta |\ Y_\beta, X_{1-\beta}, i^*, \widetilde{M}_{\beta,i^*}, \widetilde{M}_\beta^*) && (5) \\
&= \widetilde{\mathbb{H}}_\infty(X_\beta |\ Y_\beta, i^*, \widetilde{M}_{\beta,i^*}, \widetilde{M}_\beta^*) && (6) \\
&\ge \ell - 3\mu, && (7)
\end{aligned}$$

where Eq. (3) uses the definition of the random variable Z_β, Eq. (4) uses the fact that $\widetilde{M}_{0,i} = \widetilde{M}_{1,i}$ for all $i < i^*$, Eq. (5) follows by Lemma 1 and the fact that the messages $\widetilde{M}_{1-\beta,1}, \ldots, \widetilde{M}_{1-\beta,i^*-1}$ can be computed as a deterministic function of $X_{1-\beta}$ and the index i^*, Eq. (6) uses the independence between X_0 and X_1, and Eq. (7) follows by Lemma 2, using the fact that A is ℓ-valid and by the definition of $\mu := \log |\mathcal{M}|$.

It remains to account for the leakage performed by the reduction to answer the final tampering query before the state is updated. In particular:

- In step 3(d)ii, B leaks the public key $\widetilde{pk}$ and the secret key sk'; since sk' is uniformly distributed over the set $\{sk : \mathsf{PK}(sk) = \widetilde{pk}\}$, the function g_0^* is at most $\log|\mathcal{PK}| = \delta$-leaky.
- In step 3(d)iii, B leaks the commitment $\widetilde{\gamma}$ and the ciphertext c'; since c' is uniformly distributed over the set $\{c : \mathsf{Enc}(\widetilde{pk}, \widehat{M}\|\widehat{r})\}$, the function g_1^* is at most $\log|\mathcal{M}_{\text{pke}}| = \mu_{\text{pke}}$-leaky.

Putting the above facts together with the bound from Eq. 7, we obtain that B is ℓ''-admissible for $\ell'' := \ell + 3\mu + \max\{\delta, \mu_{\text{pke}}\} = \ell' - 2\kappa$. This concludes the proof. □

Lemma 5. *For any $\kappa \in \mathbb{N}$, we have $\mathbf{G}^2_{\mathsf{A}}(\kappa) \equiv \mathbf{G}^3_{\mathsf{A}}(\kappa)$.*

Proof. Follows directly by the fact that the change in the game $\mathbf{G}^3_{\mathsf{A}}(\kappa)$ is only syntactical, and in particular the view of the adversary is identical in the two games. □

Lemma 6. *For all PPT adversaries A there exists a negligible function $\nu_{3,4} : \mathbb{N} \to [0,1]$ such that $\left|\mathbb{P}\left[\mathbf{G}^3_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^4_{\mathsf{A}}(\kappa) = 1\right]\right| \leq \nu_{3,4}(\kappa)$.*

Proof. The proof is again down to the security of the CLRS friendly PKE scheme. Consider the following events:

$\mathtt{Old}_0$: The event becomes true whenever, for some $i \in [q]$, either the adversary generates a tampered ciphertext $\widetilde{c} \in \{c_1, \ldots, c_{i-1}\}$, or the tampering simulator T^0 extracts a ciphertext $\widehat{c} \in \{c_1, \ldots, c_{i-1}\}$.

$\mathtt{Old}_1$: The event becomes true whenever, for some $i \in [q]$, either the adversary generates a tampered secret key $\widetilde{sk} \in \{sk_1, \ldots, sk_{i-1}\}$, or the tampering simulator T^1 extracts a secret key $\widehat{sk} \in \{sk_1, \ldots, sk_{i-1}\}$.

Define $\mathtt{Old} = \mathtt{Old}_0 \vee \mathtt{Old}_1$. Clearly, $\mathbf{G}^3_{\mathsf{A}}(\kappa)$ and $\mathbf{G}^4_{\mathsf{A}}(\kappa)$ are identically distributed conditioned on $\mathtt{Old}$ not happening. Hence,

$$\left|\mathbb{P}\left[\mathbf{G}^3_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^4_{\mathsf{A}}(\kappa) = 1\right]\right| \leq \mathbb{P}\left[\mathtt{Old}\right].$$

We now show that $\mathtt{Old}$ only happens with negligible probability. By contradiction, assume there is an efficient adversary A and a polynomial $p_{3,4}$ such that A provokes event $\mathtt{Old}$ with probability at least $1/p_{3,4}(\kappa)$. We construct an adversary B (with black-box access to A), that is playing game $\mathbf{G}^{\text{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa, \ell)$, with the parameter ℓ as in the statement of Theorem 1. The execution of B is almost identical to that of the reduction used to prove Lemma 4; hence, we only outline the main differences between the two reductions below.

- At the beginning B additionally samples $h_0 \leftarrow \mathcal{H}_0$ and $h_1 \leftarrow \mathcal{H}_1$, where $\mathcal{H}_0 := \{h_0 : \mathcal{C} \to \{0,1\}^\kappa\}$ and $\mathcal{H}_1 := \{h_1 : \mathcal{SK} \to \{0,1\}^\kappa\}$ are families of pair-wise independent hash functions.[10]

[10]Recall that $\mathcal{H} := \{h : \mathcal{X} \to \mathcal{Y}\}$ is a family of pair-wise independent hash functions, if the following condition holds: For all distinct $x, x' \in \mathcal{X}$, and any $y, y' \in \mathcal{Y}$,

$$\mathbb{P}_{h \leftarrow \mathcal{H}}\left[h(x) = y \wedge h(x') = y'\right] \leq |\mathcal{Y}|^{-2}.$$

- The challenge messages are chosen to be $(0^\mu \| r, 1^{\mu_{\mathrm{pke}}})$. (The choice of the second message is completely arbitrary, in fact any value in $\mathcal{M}_{\mathrm{pke}}$ would do.)
- Before running the adversary, in each round $i \in [q]$, the reduction makes leakage queries $(0, h_0)$ and $(1, h_1)$, obtaining $y_i^0 = h_0(c_i)$ and $y_i^1 = h_1(sk_i)$, where c_i and sk_i are, respectively, the challenge ciphertext and the secret key after the i-th update.
- The leakage function g_0 of Eq. (1), additionally hard-wires a description of h_0, and all values $y_1^0, \ldots, y_{i-1}^0$, and it checks if either $y_j^0 = h_0(\widetilde{c})$ or $y_j^0 = h_0(\widehat{c})$, for some $j \le i-1$; in case that happens, the leakage function returns $(\mathtt{Old}_0, c^*)$, where $c^* \in \{\widetilde{c}, \widehat{c}\}$ is such that $y_j^0 = h_0(c^*)$, and otherwise it behaves as before.
- The leakage function g_1 of Eq. (2), additionally hard-wires a description of h_1, and all values $y_1^1, \ldots, y_{i-1}^1$, and it checks if either $y_j^1 = h_1(\widetilde{sk})$ or $y_j^1 = h_1(\widehat{sk})$, for some $j \le i-1$; in case that happens, the leakage function returns $(\mathtt{Old}_1, sk^*)$, where $sk^* \in \{\widetilde{sk}, \widehat{sk}\}$ is such that $y_j^1 = h_1(sk^*)$, and otherwise it behaves as before.
- After the adversary is done with tampering queries, the reduction checks whether for some of the leakage queries it received either a pair $(\mathtt{Old}_0, c^*)$ or a pair $(\mathtt{Old}_1, sk^*)$. Thus,
  - In the former case, it asks an additional leakage query $(1, g_{c^*, M\|r, 1^{\mu_{\mathrm{pke}}}})$, where the function $g_{c^*, M\|r, 1^{\mu_{\mathrm{pke}}}}(sk)$ uses the secret key to decrypt c^*, and signals whether the obtained plaintext is equal to $M\|r$ or to $1^{\mu_{\mathrm{pke}}}$.
  - In the latter case, it asks an additional leakage query $(1, g_{sk^*, M\|r, 1^{\mu_{\mathrm{pke}}}})$, where the function $g_{sk^*, M\|r, 1^{\mu_{\mathrm{pke}}}}(c)$ uses the secret key sk^* to decrypt c, and signals whether the obtained plaintext is equal to $M\|r$ or to $1^{\mu_{\mathrm{pke}}}$.

  Otherwise, the reduction returns a random guess $b' \leftarrow \{0, 1\}$.

Assume first that B is an ℓ'-admissible adversary (i.e., it does not violate the leakage bound), where the parameter ℓ' is as in the statement of Theorem 1. Notice that, if $\mathtt{Old}_0$ happens, with probability $1 - 2^{-\kappa}$ due to the use of a pair-wise independent hash function, the extracted ciphertext $\widehat{c}$ obtained by the reduction is equal to one of the ciphertexts c_j obtained by updating the original challenge ciphertext c; hence, in such a case, by correctness of the PKE scheme, adversary B wins with overwhelming probability. Furthermore, a similar argument shows that B wins with overwhelming probability in case $\mathtt{Old}_1$ happens. On the other hand, if none of the events happen, B is successful with probability $1/2$. Putting these observations together with the fact that $\mathtt{Old}$ happens if at least one of $\mathtt{Old}_0$ and $\mathtt{Old}_1$ happen, we obtain

$$\begin{aligned}
&2\left|\mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 1\right] - \frac{1}{2}\right| \\
&= \left|\mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 1 |\ b = 1\right] - \mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 0 |\ b = 0\right]\right| \\
&= \Bigg|\mathbb{P}\left[\mathtt{Old}\right] \cdot \left(\mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 1 | b = 1, \mathtt{Old}\right] - \mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 0 | b = 0, \mathtt{Old}\right]\right) \\
&\qquad + \mathbb{P}\left[\overline{\mathtt{Old}}\right] \cdot \left(\mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 1 | b = 1, \overline{\mathtt{Old}}\right] - \mathbb{P}\left[\mathbf{G}^{\mathrm{clrs}}_{\mathcal{PKE},\mathsf{B}}(\kappa,\ell') = 0 | b = 0, \overline{\mathtt{Old}}\right]\right)\Bigg| \\
&\ge 1/p_{3,4}(\kappa) - 2 \cdot 2^{-\kappa},
\end{aligned}$$

a contradiction.

It remains to show that B is ℓ'-admissible. Assume first that the event $\mathtt{Old}$ does not happen. Then, using an argument identical to that in the proof of the previous lemma, and considering that the reduction here additionally leaks two hash values at the beginning of each round, one can show that the composition of B's leakage functions is $(\ell + 3\mu + 2\kappa + \max\{\delta, \mu_{\text{pke}}\})$-leaky. Thus, in order to conclude the proof, it suffices to show that the additional leakage due to the fact that the event `Old` happens does not reduce the entropy of the state further. While doing this, we distinguish the case where event $\texttt{Old}_0$ and event $\texttt{Old}_1$ happen. In particular:

$\texttt{Old}_0$**:** We use the perfect secret-key-update privacy of the PKE scheme (cf. Definition 7). Let $\mathsf{UpdateS}^i(\cdot) := \mathsf{UpdateS}(\mathsf{UpdateS}^{i-1}(\cdot))$ if $i > 1$, and $\mathsf{UpdateS}^1(\cdot) := \mathsf{UpdateS}(\cdot)$. By Definition 7, for any (p, s) such that $\mathsf{PK}(s) = p$, the distribution $\{p, \mathsf{UpdateS}(s)\}$ is identical to $\{(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa) : \; pk = p\}$; this implies that for any $(p, s) \leftarrow \mathsf{KGen}(1^\kappa)$, if sk is in the support of $\{\mathsf{UpdateS}(s)\}$, then sk is also in the support of $\{sk' : \; (pk', sk') \leftarrow \mathsf{KGen}(1^\kappa), pk' = p\}$. Therefore, by a simple inductive argument, we can show that for any $0 \leq i < i'$ and any $(p, s) \leftarrow \mathsf{KGen}(1^\kappa)$, the distributions below are equivalent:

$$\mathcal{D}^{p,s}_{i,i'} := \left\{ (sk, sk') : \begin{array}{c} sk' \leftarrow \mathsf{UpdateS}^i(s) \\ sk \leftarrow \mathsf{UpdateS}^{i'}(sk') \end{array} \right\}$$
$$\tilde{\mathcal{D}}^{p,s}_{i,i'} := \left\{ (sk, sk') : \begin{array}{c} pk' = p; (pk', sk') \leftarrow \mathsf{KGen}(1^\kappa) \\ sk \leftarrow \mathsf{UpdateS}^{i+i'}(s) \end{array} \right\}.$$

Let now i^* be the first index where $\texttt{Old}_0$ happens, and let P' be an unbounded predictor. Then, for any $i < i^*$, we get:

$$\begin{aligned} \widetilde{\mathbb{H}}_\infty(sk_{i^*} | pk, sk_i) &\geq -\log \max_{\mathsf{P}'} \mathbb{P}\left[\mathsf{P}'(pk, sk_i) = sk_{i^*} : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KGen}(1^\kappa), \\ (sk_{i^*}, sk_i) \leftarrow \mathcal{D}^{pk,sk}_{i,i^*-i} \end{array} \right] \\ &= -\log \max_{\mathsf{P}'} \mathbb{P}\left[\mathsf{P}'(pk, sk_i) = sk_{i^*} : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KGen}(1^\kappa) \\ (sk_{i^*}, sk_i) \leftarrow \tilde{\mathcal{D}}^{pk,sk}_{i,i^*-i} \end{array} \right] \\ &= -\log \max_{\mathsf{P}'} \mathbb{P}\left[\mathsf{P}^{\mathsf{P}'}(pk) = sk_{i^*} : \; (pk, sk) \leftarrow \mathsf{KGen}(1^\kappa) \right] \\ &\geq \widetilde{\mathbb{H}}_\infty(sk_{i^*} | pk), \end{aligned}$$

where P samples sk_i from $\{(pk', sk') \leftarrow \mathsf{KGen}(1^\kappa) : \; pk' = pk\}$, and then runs $\mathsf{P}'(pk, sk_i)$.

$\texttt{Old}_1$**:** We use the perfect ciphertext-update privacy property of the PKE scheme (cf. Definition 6). Let $\mathsf{UpdateC}^i(\cdot) := \mathsf{UpdateC}(\mathsf{UpdateC}^{i-1}(\cdot))$ if $i > 1$, and $\mathsf{UpdateC}^1(\cdot) := \mathsf{UpdateC}(\cdot)$. By Definition 6, for any $m \in \mathcal{M}_{\text{pke}}$ and c, such that $c = \mathsf{Enc}(pk, m; r)$ for some $r \in \{0,1\}^*$, the distribution $\{pk, \mathsf{UpdateC}(c)\}$ is identical to $\{pk, \mathsf{Enc}(pk, m)\}$. This implies that for any $0 \leq i < i'$, and any $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$, the distributions below are equivalent:

$$\mathcal{D}^{pk,m}_{i,i'} := \left\{ (c, c') : \begin{array}{c} \bar{c} \leftarrow \mathsf{Enc}(pk, m); c' \leftarrow \mathsf{UpdateC}^i(\bar{c}) \\ c \leftarrow \mathsf{UpdateC}^{i'}(c') \end{array} \right\}$$
$$\tilde{\mathcal{D}}^{pk,m}_{i,i'} := \left\{ (c, c') : \begin{array}{c} \bar{c} \leftarrow \mathsf{Enc}(pk, m); c' \leftarrow \mathsf{Enc}(pk, m) \\ c \leftarrow \mathsf{UpdateC}^{i+i'}(\bar{c}) \end{array} \right\}.$$

Let now i^* be the first index where $\texttt{Old}_1$ happens, and let P' be an unbounded predictor.

Then, for all $m \in \mathcal{M}_{\text{pke}}$, and any $i < i^*$, we get:

$$\begin{aligned}
\widetilde{\mathbb{H}}_\infty(c_{i^*}|pk, c_i) &\geq -\log \max_{\mathsf{P}'} \mathbb{P}\left[\mathsf{P}'(pk, c_i) = c_{i^*} : \begin{array}{c}(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)\\ (c_{i^*}, c_i) \leftarrow \mathcal{D}^{pk,m}_{i,i^*-i}\end{array}\right]\\
&= -\log \max_{\mathsf{P}'} \mathbb{P}\left[\mathsf{P}'(pk, c_i) = c_{i^*} : \begin{array}{c}(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)\\ (c_{i^*}, c_i) \leftarrow \tilde{\mathcal{D}}^{pk,m}_{i,i^*-i}\end{array}\right]\\
&= -\log \max_{\mathsf{P}'} \mathbb{P}\left[\mathsf{P}^{\mathsf{P}'}(pk) = c_{i^*} : \ (pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)\ \right]\\
&\geq \widetilde{\mathbb{H}}_\infty(c_{i^*}|pk),
\end{aligned}$$

where P samples c_i from $\{\mathsf{Enc}(pk, m)\}$, and then runs $\mathsf{P}'(pk, c_i)$.

□

Lemma 7. *For all $\kappa \in \mathbb{N}$, we have $\mathbf{G}^4_{\mathsf{A}}(\kappa) \equiv \mathbf{G}^5_{\mathsf{A}}(\kappa)$.*

Proof. Recall that the simulator S_1 returns same if and only if both tampering simulators T^0 and T^1 return $\diamond$; furthermore:

- The tampering simulator T^0 returns $\diamond$ at round i if either:
  - $\widetilde{c} \in \{c, c_{i-1}, \ldots, c_1\}$; or
  - $\widehat{c} = c$, and $\phi_0(\widehat{c}) = \widetilde{c}$ (after running the extractor).
- The tampering simulator T^1 returns $\diamond$ at round i if either:
  - $\widetilde{sk} \in \{sk, sk_{i-1}, \ldots, sk_1\}$; or
  - $\widehat{sk} = sk$, and $\phi_1(\widehat{sk}) = \widetilde{sk}$ (after running the extractor).

Let `NotSame` be the event that S_1 outputs $\diamond$ in game $\mathbf{G}^4_{\mathsf{A}}(\kappa)$, but $\mathsf{Decode}(\overline{\omega}, \widetilde{C}) \neq M$ in game $\mathbf{G}^5_{\mathsf{A}}(\kappa)$. Since the two games are identical conditioned on `NotSame` *not* happening, it suffices to bound the probability of event `NotSame`.

By definition of the set Φ_1, we have that, in both cases where T^1 returns $\diamond$, the secret key $\widetilde{sk}$ is a valid secret key for the value pk. Similarly, by definition of the set Φ_0, we have that, in both cases where T_0 returns $\diamond$, the ciphertext $\widetilde{c}$ must meet the condition $\mathsf{Dec}(sk, \widetilde{c}) = \mathsf{Dec}(sk, c)$. Putting these observations together, we obtain that $\mathsf{Dec}(\widetilde{sk}, \widetilde{c}) = \mathsf{Dec}(sk, c) = M\|r$, and thus event $\mathsf{NotSame}$ happens with zero probability (assuming the PKE scheme has perfect correctness). This concludes the proof of the claim. □

Lemma 8. *For all (even unbounded) adversaries A, there exists a negligible function $\nu_{5,6} : \mathbb{N} \to [0, 1]$ such that $\left|\mathbb{P}\left[\mathbf{G}^5_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^6_{\mathsf{A}}(\kappa) = 1\right]\right| \leq \nu_{5,6}(\kappa)$.*

Proof. Game $\mathbf{G}^5_{\mathsf{A}}(\kappa)$ and $\mathbf{G}^6_{\mathsf{A}}(\kappa)$ are identical, unless there exists a tampering query such that both T^0 and T^1 return the same message $\widetilde{M} \notin \{\perp, \diamond\}$, but $\mathsf{Decode}(\overline{\omega}, \widetilde{C}) \neq \widetilde{M}$ in game $\mathbf{G}^6_{\mathsf{A}}(\kappa)$. (This could happen also during the extra tampering query.) This means that the following situation happens in $\mathbf{G}^6_{\mathsf{A}}(\kappa)$:

- $\mathsf{Dec}(\widehat{sk}, \widetilde{c}) = \widetilde{M}\|\widetilde{r}$ and $\widetilde{\gamma} = \mathsf{Commit}(\omega, \widetilde{M}; \widehat{r})$;
- $\mathsf{Dec}(\widetilde{sk}, \widetilde{c}) = \widetilde{M}'\|\widetilde{r}'$ and $\widetilde{\gamma} = \mathsf{Commit}(\omega, \widetilde{M}'; \widetilde{r}')$, for some $\widetilde{M}' \neq \widetilde{M}$.

The above implies that there exist $(\widetilde{M}, \widehat{r}), (\widetilde{M'}, \widehat{r'})$ (with $\widetilde{M} \neq \widetilde{M'}$) such that $\widetilde{\gamma} = \mathsf{Commit}(\omega, \widetilde{M}; \widehat{r}) = \mathsf{Commit}(\omega, \widetilde{M'}; \widehat{r'})$, which in turn contradicts the statistical binding property of the commitment scheme. $\square$

Lemma 9. *For all PPT adversaries* A *there exists a negligible function* $\nu_{6,7} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}^6_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^7_{\mathsf{A}}(\kappa) = 1\right]\right| \leq \nu_{6,7}(\kappa)$.

Proof. The proof is down to the label-malleable simulation extractability of the argument systems $\mathcal{NIA}_0$ and $\mathcal{NIA}_1$. For $\beta \in \{0,1\}$, let $\mathtt{Abort}_\beta$ be the event that S_1 outputs **Abort** in game $\mathbf{G}^6_{\mathsf{A}}(\kappa)$ because T^β returned **Abort** as an answer to one of the tampering queries from the adversary. Define $\mathtt{Abort} = \mathtt{Abort}_0 \vee \mathtt{Abort}_1$, and note that $\mathbf{G}^6_{\mathsf{A}}(\kappa)$ and $\mathbf{G}^7_{\mathsf{A}}(\kappa)$ are identical conditioned on $\mathtt{Abort}$ *not* happening. Thus, it suffices to bound the probability that $\mathtt{Abort}$ happens.

We start by proving that $\mathtt{Abort}_0$ happens only with negligible probability. By contradiction, assume that there exists a PPT adversary A and a polynomial $p_{6,7}(\kappa)$, such that, for infinitely many values of $\kappa \in \mathbb{N}$, adversary A provokes event $\mathtt{Abort}_0$ with probability at least $1/p_{6,7}(\kappa)$. Consider the following adversary B (with black-box access to A), that is playing game $\mathbf{G}^{\text{lm-se}}_{\mathcal{NIA}_0,\mathsf{B},\mathsf{K}_0}(\kappa, \Phi_0)$.

Adversary B:

- Receive ω_0 from the challenger. Run $(\omega_1, \tau^1_{\text{sim}}, \tau^1_{\text{ext}}) \leftarrow \mathsf{S}^1_0(1^\kappa)$, pick $\omega \leftarrow \mathsf{CRSGen}(1^\kappa)$, and $\rho \leftarrow \mathsf{Setup}(1^\kappa)$.
- Run $\mathsf{A}(\overline{\omega})$, where $\overline{\omega} := (\omega_0, \omega_1, \omega, \rho)$, obtaining a message M.
- Compute the initial target encoding (C_0, C_1) as S_0 would do it in game $\mathbf{G}^7_{\mathsf{A}}(\kappa)$, except that the proof π_0 is obtained by forwarding (c, pk) to the oracle $\mathcal{O}^*_{\mathsf{sim}}(\cdot)$.
- Keep running A by answering all of its queries exactly as S_1 would do it in game $\mathbf{G}^7_{\mathsf{A}}(\kappa)$, but with the following differences:
  - Whenever S_1 needs to run $\mathsf{Rfrsh}^*((\tau^0_{\text{sim}}, \tau^1_{\text{sim}}), (C_0, C_1))$, compute $C'_0 := (pk, c', \pi'_0)$ and $C'_1 := (\gamma, sk', \pi'_1)$ exactly as described in algorithm Rfrsh^*, except that the proof π'_0 is obtained by forwarding (c', pk) to the oracle $\mathcal{O}^*_{\mathsf{sim}}(\cdot)$;
  - Whenever S_1 returns **Abort**, upon input a tampering query (f_0, f_1) from the adversary,[11] compute $(\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0) = f_0(pk, c, \pi_0)$, and forward $(\widetilde{\lambda}, \widetilde{x}, \widetilde{\pi})$ to the challenger, where $\widetilde{\lambda} := \widetilde{c}$, $\widetilde{x} := \widetilde{pk}$, and $\widetilde{\pi} := \widetilde{\pi}_0$.

By our assumption, with probability at least $1/p_{6,7}(\kappa)$, the attacker A will provoke event $\mathtt{Abort}_0$. This means that, with the same probability, the tuple $(\widetilde{\lambda}, \widetilde{x}, \widetilde{\pi})$ returned by B satisfies the following:

- The proof $\widetilde{\pi}$ is accepting for the statement $\widetilde{x}$ w.r.t. the label $\widetilde{\lambda}$ (i.e., $\mathsf{Vrfy}^{\widetilde{c}}(\omega_0, \widetilde{pk}, \widetilde{\pi}_0) = 1$).
- The tuple $(\widetilde{\lambda}, \widetilde{x}, \widetilde{\pi})$ is not within the set $\mathcal{Q}$ of simulated proofs and corresponding statements/labels; this is because the tampering simulator extracts the proof only if $(\widetilde{pk}, \widetilde{c}, \widetilde{\pi}_0) \neq (pk, c, \pi_0)$, and further it does not abort if $\widetilde{c} \in \{c_1, \ldots, c_{i-1}\}$ where $c_1, \ldots, c_{i-1}$ are the refreshed ciphertexts computed by the simulator S_1 up to the i-th round.
- Either of the following conditions are met:

[11] Note that the pair (f_0, f_1) might be either part of a $(\mathtt{Tamp}, \cdot)$ command or of a $(\mathtt{Final}, \cdot)$ command.

(i) $\widehat{sk} \neq \bot$ and $(\widetilde{pk}, \widehat{sk}) \notin \mathcal{R}_0$; or

(ii) $(\phi_0, \widehat{c}) \neq (\bot)$ and either $\phi_0 \notin \Phi_0$, or $(\widehat{c}, \widetilde{pk}) \notin \mathcal{Q}_\downarrow$,[12] or $\phi_0(\widehat{c}) \neq \widetilde{c}$; or

(iii) $(\widehat{sk}, \phi_0, \widehat{c}) = (\bot, \bot, \bot)$.

Hence, B breaks label-malleable simulation extractability of $\mathcal{NIA}_0$ with non-negligible probability, and thus $\mathtt{Abort}_0$ only happens with a negligible probability. A similar argument (omitted here) shows that $\mathtt{Abort}_1$ also happens with negligible probability. The proof of the lemma now follows by an application of the union bound. □

Lemma 10. *For all (even unbounded) adversaries* A*, there exists a negligible function* $\nu_{7,8} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}^7_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^8_{\mathsf{A}}(\kappa) = 1\right]\right| \leq \nu_{7,8}(\kappa)$.

Proof. We rely on the statistical binding property of the commitment and the correctness property of the PKE scheme. Assume (f_0, f_1) be any tampering query produced by the adversary, such that $\mathsf{Decode}(\overline{\omega}, (f_0(C_0), f_1(C_1))) = \widetilde{M} \neq \bot$ in game $\mathbf{G}^8_{\mathsf{A}}(\kappa)$, but the simulator S_1 of game $\mathbf{G}^7_{\mathsf{A}}(\kappa)$ instead would return $\bot$. Further, let $\widetilde{M}_0$ and $\widetilde{M}_1$ the values returned by the tampering simulators S_0 and S_1, respectively, upon input the tampering query (f_0, f_1). We distinguish two cases.

Case 1: $\widetilde{M}_0 \neq \widetilde{M}_1$ and $\widetilde{M}_0, \widetilde{M}_1 \neq \bot$. We claim that $\widetilde{M} = \widetilde{M}_0$. In fact, the decoding algorithm decrypts $\widetilde{c}$ using $\widetilde{sk}$, whereas T^0 decrypts $\widetilde{c}$ using $\widehat{sk}$, but $\mathsf{PK}(\widetilde{sk}) = \mathsf{PK}(\widehat{sk})$. Therefore, by the correctness of the PKE scheme, we must have $\widetilde{M} = \widetilde{M}_0$.

On the other hand, the decoding algorithm also checks that $\widetilde{\gamma}$ is a valid commitment to $\widetilde{M}$, and T^1 extracts $\widehat{M} = \widetilde{M}_1$ as a valid opening of $\widetilde{\gamma}$. This contradict the statistical binding property of the commitment scheme.

Case 2: At least one of $\widetilde{M}_0, \widetilde{M}_1$ is equal to $\bot$. Note that the tampering simulator T^0 (resp., T^1) returns $\bot$ only in case the proof $\widetilde{\pi}_0$ (resp., $\widetilde{\pi}_1$) is invalid. However, in such a case also the decoding algorithm outputs $\bot$, and thus we conclude that $\widetilde{M} = \bot$ (a contradiction).

□

Lemma 11. *For all* $\kappa \in \mathbb{N}$*, and for all (even unbounded) adversaries* A*, we have* $\mathbf{G}^8_{\mathsf{A}}(\kappa) \equiv \mathbf{G}^9_{\mathsf{A}}(\kappa)$.

Proof. The only difference between the two games is that, while answering the extra tampering query, the former game computes sk' as a refresh of the extracted secret key $\widehat{sk}_0$, whereas the latter game computes sk' as refresh of the tampered secret key $\widetilde{sk}$. However, note that the simulator S_1 runs the extractor K_0 on the same random coins as used to obtain the messages $\widetilde{M}^*_0$ and $\widetilde{M}^*_1$. Since S_1 never aborts in game $\mathbf{G}^8_{\mathsf{A}}(\kappa)$, and since we are looking at the case $\widetilde{M}^*_0 = \widetilde{M}^*_1 \notin \{\bot, \diamond\}$, we conclude that $\widetilde{sk}$ is a valid secret key corresponding to pk. Hence, the lemma directly follows by the secret-key update privacy property of the PKE scheme. □

Lemma 12. *For all* $\kappa \in \mathbb{N}$*, and for all (even unbounded) adversaries* A*, we have* $\mathbf{G}^9_{\mathsf{A}}(\kappa) \equiv \mathbf{G}^{10}_{\mathsf{A}}(\kappa)$.

[12] The fact that $(\widehat{c}, \widetilde{pk}) \notin \mathcal{Q}_\downarrow$ follows by the fact that T^0 outputs **Abort** in step 2d of game $\mathbf{G}^7_{\mathsf{A}}(\kappa)$ only if $\widehat{c} \notin \{c_1, \ldots, c_{i-1}\}$, where $c_1, \ldots, c_{i-1}$ are the refreshed ciphertexts computed by the simulator S_1 up to the i-th round; furthermore, in case $\widehat{c} = c_i$, the tampering simulator does not abort.

Proof. The only difference between the two games is that, while answering the extra tampering query, the former game computes c' as a fresh encryption of the extracted value $\widehat{M}\|\widehat{r}$, whereas the latter game computes c' as refresh of the tampered ciphertext $\widetilde{c}$. However, note that the simulator S_1 runs the extractor K_1 on the same random coins as used to obtain the messages $\widetilde{M}_0^*$ and $\widetilde{M}_1^*$. Since S_1 never aborts in game $\mathbf{G}_{\mathsf{A}}^9(\kappa)$, and since we are looking at the case $\widetilde{M}_0^* = \widetilde{M}_1^* \not\in \{\bot, \diamond\}$, we conclude that $\widetilde{c}$ is a valid encryption of the value $\widehat{M}\|\widehat{r}$. Hence, the lemma directly follows by the ciphertext update privacy property of the PKE scheme. □

Lemma 13. *For all PPT adversaries* A *there exists a negligible function* $\nu_{10,11} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{10}(\kappa) = 1\right] = \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{11}(\kappa) = 1\right]\right| \leq \nu_{10,11}(\kappa)$.

Proof. The only difference between the two games is that in the former game all proofs for the relation $\mathcal{R}_0$ are computed via the zero-knowledge simulator S_1^0, whereas in the latter game such proofs are computed by running the real prover algorithm Prove_0. Hence, the lemma follows by a straightforward reduction to the adaptive multi-theorem property of $\mathcal{NIA}_0$. □

Lemma 14. *For all PPT adversaries* A *there exists a negligible function* $\nu_{11,12} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{11}(\kappa) = 1\right] = \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{12}(\kappa) = 1\right]\right| \leq \nu_{11,12}(\kappa)$.

Proof. Identical to the proof of the previous lemma, but we now reduce to the (adaptive multi-theorem) zero-knowledge property of $\mathcal{NIA}_1$. □

Lemma 15. *For all PPT adversaries* A *there exists a negligible function* $\nu_{12,13} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{12}(\kappa) = 1\right] = \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{13}(\kappa) = 1\right]\right| \leq \nu_{12,13}(\kappa)$.

Proof. The only difference between the two games is that in the former game, while refreshing a codeword, the proof corresponding to relation $\mathcal{R}_0$ is computed by running the prover algorithm Prove_0, whereas in the latter game such proof is obtained by running the label evaluation procedure LEval_0. Hence, the lemma follows by a straightforward reduction to the label derivation privacy of $\mathcal{NIA}_0$. □

Lemma 16. *For all PPT adversaries* A *there exists a negligible function* $\nu_{13,14} : \mathbb{N} \to [0,1]$ *such that* $\left|\mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{13}(\kappa) = 1\right] = \mathbb{P}\left[\mathbf{G}_{\mathsf{A}}^{14}(\kappa) = 1\right]\right| \leq \nu_{13,14}(\kappa)$.

Proof. Identical to the proof of the previous lemma, but we now reduce to the label derivation privacy property of $\mathcal{NIA}_1$. □

5 Concrete Instantiations

For a group $\mathbb{G}$ of prime order q and a generator g of $\mathbb{G}$, we denote by $[a]_g := g^a \in \mathbb{G}$ the *implicit representation* of an element $a \in \mathbb{Z}_q$. Let G be a PPT pairing generation algorithm that, upon input the security parameter 1^κ, outputs a tuple $\mathtt{params} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, \hat{\mathsf{e}})$ where the first three elements are the description of groups of prime order $q > 2^\kappa$, g (resp. h) is a generator for the group $\mathbb{G}_1$ (resp. $\mathbb{G}_2$), and $\hat{\mathsf{e}} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable non-degenerate pairing function.

Within this section, vectors and matrices are denoted boldface, and vectors are intended as row vectors. Given a group $\mathbb{G}$, two matrices $\mathbf{X} \in \mathbb{G}^{n\times m}$ and $\mathbf{Y} \in \mathbb{G}^{m\times t}$, for some $n, m, t \geq 1$, and an element $a \in \mathbb{G}$, we denote by $\mathbf{X} \cdot \mathbf{Y}$ the matrix product of $\mathbf{X}$ and $\mathbf{Y}$, and by $a \cdot \mathbf{X}$ the scalar multiplication of $\mathbf{X}$ by a. Given two elements $[a]_g \in \mathbb{G}_1$ and $[b]_h \in \mathbb{G}_2$, we denote by $[a]_g \bullet [b]_h = [a \cdot b]_{e(g,h)}$ the value $\hat{\mathsf{e}}([a]_g, [b]_h)$; this notation is also applied to vectors and matrices in the natural way. Given a field $\mathbb{F}$ and natural numbers $n, m, j \in \mathbb{N}$, where $j \leq \min(n, m)$, we define $\mathsf{Rk}_j(\mathbb{F}^{n\times m})$ to be the set of matrices in $\mathbb{F}^{n\times m}$ with row rank j; given a matrix $\mathbf{B}$ we let $\mathsf{Rank}(\mathbf{B})$ be the rank of $\mathbf{B}$.

Definition 13 (k-Rank hiding assumption)**.** Let G be a pairing generation algorithm as above. The *k-rank hiding* assumption holds for G if, for any $k \le j, j' \le \min(n, m)$, we have that $(g, [\mathbf{B}]_g) \approx_c (g, [\mathbf{B}']_g)$ and $(h, [\mathbf{B}]_h) \approx_c (h, [\mathbf{B}']_h)$, where $\mathbf{B} \leftarrow \mathsf{Rk}_j$, $\mathbf{B}' \leftarrow \mathsf{Rk}_{j'}$, and $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, \hat{\mathsf{e}}) \leftarrow \mathsf{G}(1^\kappa)$.

As shown by Naor and Segev in [51], the k-rank hiding assumption follows from the more common k-linear assumption. The assumption gets weaker as k increases; in fact, for $k = 1$, this assumption is equivalent to the DDH assumption. Unfortunately, DDH does not hold in symmetric pairing groups where $\mathbb{G}_1 = \mathbb{G}_2$. However, it is widely believed that DDH still holds in asymmetric pairing groups; this is sometimes known as the *external Diffie-Hellman* (SXDH) assumption [11].

5.1 The PKE Scheme

We consider the CLRS Friendly PKE scheme of [33] (which in turn is based on the one in [29]). Consider the following PKE scheme $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdateC}, \mathsf{UpdateS})$, with message space $\mathcal{M} := \{0,1\}$ and parameters $n, m, d \in \mathbb{N}$.

- $\underline{\mathsf{Setup}(1^\kappa)}$: Sample $\mathtt{params} \leftarrow \mathsf{G}(1^\kappa)$, and vectors $\mathbf{p}, \mathbf{w} \leftarrow \mathbb{Z}_q^m$ such that $\mathbf{p} \cdot \mathbf{w}^{\mathrm{T}} = 0 \mod q$. Return $\rho := (\mathtt{params}, [\mathbf{p}]_g, [\mathbf{w}]_h)$. (Recall that all algorithms implicitly take ρ as input.)

- $\underline{\mathsf{KGen}(\rho)}$: Sample $\mathbf{t} \leftarrow \mathbb{Z}_q^m$, $\mathbf{r} \leftarrow \mathbb{Z}_q^n$ and compute $sk := [\mathbf{r}^{\mathrm{T}} \cdot \mathbf{w} + \mathbf{1}_n^{\mathrm{T}} \cdot \mathbf{t}]_h$, set $\alpha := \mathbf{p} \cdot \mathbf{t}^{\mathrm{T}}$ and compute $pk := [\alpha]_g$. (The latter can be done given only $[\mathbf{p}]_g \in \mathbb{G}_1^m$ and $\mathbf{t} \in \mathbb{Z}_q^m$.) Return (pk, sk).

- $\underline{\mathsf{Enc}(pk, b)}$: Sample $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ and compute $c_1 := [\mathbf{u}^{\mathrm{T}} \cdot \mathbf{p}]_g$ and $c_2 := [\alpha \mathbf{u} + b \mathbf{1}_n]_g$. Return $c := (c_1, c_2)$.

- $\underline{\mathsf{Dec}(sk, C)}$: Let $f = \hat{\mathsf{e}}(g, h)$, parse $sk = [\mathbf{S}]_h \in \mathbb{G}_2^{n \times m}$, let $\mathbf{S}_1$ be the first row of $\mathbf{S}$, and parse $c = ([\mathbf{C}]_g, [\mathbf{c}]_g) \in \mathbb{G}_1^{n \times m} \times \mathbb{G}_1^n$. Compute $\mathbf{b} := [\mathbf{c} - \mathbf{C} \cdot \mathbf{S}_1^{\mathrm{T}}]_f$ and output 1 if and only if $\mathbf{b} = [\mathbf{1}_n]_f$. Note that $[\mathbf{b}]_f$ can be determined by first computing $[\mathbf{c}]_f := \hat{\mathsf{e}}([\mathbf{c}]_g, h)$, and then $[\mathbf{C} \cdot \mathbf{S}_1^{\mathrm{T}}]_f := \prod_i \hat{\mathsf{e}}(\mathbf{C}[i], \mathbf{S}[i])$.

- $\underline{\mathsf{UpdateC}(C)}$: Parse $c = ([\mathbf{C}]_g, [\mathbf{c}]_g) \in \left(\mathbb{G}_1^{n \times m} \times \mathbb{G}_1^n\right)$. Sample $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times n}$ such that $\mathbf{B} \cdot \mathbf{1}_n^{\mathrm{T}} = \mathbf{1}_n$ and the rank of $\mathbf{B}$ is d. Return $([\mathbf{B} \cdot \mathbf{C}], [\mathbf{B} \cdot \mathbf{c}^{\mathrm{T}}])$.

- $\underline{\mathsf{UpdateS}(sk)}$: Parse $sk = [\mathbf{S}]_h \in \mathbb{G}_2^{n \times m}$. Sample $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times n}$ such that $\mathbf{A} \cdot \mathbf{1}_n^{\mathrm{T}} = \mathbf{1}_n$ and the rank of $\mathbf{A}$ is d. Return $[\mathbf{A} \cdot \mathbf{S}]_h$.

Faonio and Nielsen [33] showed that, under the SXDH assumption, the above PKE scheme is CLRS-friendly in the bounded leakage model (i.e., the total amount of leakage is upper bounded by a value $\ell \in \mathbb{N}$). The theorem below states that the same PKE is also secure in the more general noisy leakage model.

Theorem 2. *For any $m \ge 6$, $n \ge 3m - 6$, and $d := n - m + 3$, the above construction gives an ℓ-noisy CLRS-friendly PKE scheme under the SXDH assumption, for $\ell = \min\{m/6 - 1, n - 3m + 6\} \cdot \log(q) - \omega(\log \kappa)$.*

Proof sketch. The proof is heavily based on the elegant proof in [29]. Most of the hybrids are identical. The main difference is that we need a "noisy version" of the so-called subspace hiding lemma (cf. Lemma B7 in [29]). We state this lemma below.

Lemma 17 (Noisy model version of Lemma B.7 of [29]). *Let the integers d, n, s, u be polynomials in the security parameter $\kappa \in \mathbb{N}$. Let $\mathbf{S} \in \mathbb{Z}_q^{d\times s}$ be an arbitrary (fixed and public) matrix, and L an ℓ-leaky function (possibly chosen depending on $\mathbf{S}$). For randomly sampled $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times d}$, $\mathbf{V} \leftarrow \mathbb{Z}_q^{d\times u}$, and $\mathbf{U} \leftarrow \mathbb{Z}_q^{n\times u}$, we have:*

$$(L(\mathbf{A}), \mathbf{AS}, \mathbf{V}, \mathbf{AV}) \approx_s (L(\mathbf{A}), \mathbf{AS}, \mathbf{V}, \mathbf{U}),$$

as long as $(d - s - u)\log(q) - \ell = \omega(\log \kappa)$.

Proof sketch. The proof follows almost verbatim the one in [29]. In particular, it suffices to apply the leftover-hash lemma for leaky sources [30] to each row of $\mathbf{A}$ independently. Take any row $\mathbf{a}_i$ of $\mathbf{A}$, and think of it as a random source whose conditional average min-entropy is

$$\widetilde{\mathbb{H}}_\infty(\mathbf{a}_i|\ \mathbf{AS}, L(\mathbf{A})) \geq \widetilde{\mathbb{H}}_\infty(\mathbf{a}_i|\ L(\mathbf{A})) - s\log(q) \geq d\log(q) - s\log(q) - \ell,$$

where, in the second step, we simply applied the definition of an ℓ-leaky function (notice that $\mathbf{A}$ is uniform). Now, think of $\mathbf{V}$ as the seed of the universal hash function $h_{\mathbf{V}}(\mathbf{a}_i) = \mathbf{a}_i\mathbf{V}$, whose output size is $u\log(q)$ bits. The leftover-hash lemma tells us that the ith row of $\mathbf{AV}$ looks uniform. By using the hybrid argument over all the rows we obtain the lemma. $\square$

$\square$

For any $k \in \mathbb{N}$, let $\mathcal{PKE}^{\times k} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}^{\times k}, \mathsf{Dec}^{\times k})$ where $\mathsf{Enc}^{\times k}(pk, m_1, \dots, m_k) := (\mathsf{Enc}(pk, m_1), \mathsf{Enc}(pk, m_2), \dots, \mathsf{Enc}(pk, m_k))$, and $\mathsf{Dec}^{\times k}$ performs the obvious decryption procedure. The lemma below says that constructing single-bit PKE is sufficient for obtaining k-bit PKE, in the setting of CLRS-friendly security.

Lemma 18. *For any polynomial $k(\kappa)$, if $\mathcal{PKE}$ is an ℓ-noisy CLRS-friendly secure PKE, then $\mathcal{PKE}^{\times k}$ is an ℓ-noisy CLRS-friendly secure PKE.*

5.2 The Commitment Scheme

We use the commitment scheme of [43], that is secure under the 2-LIN (a.k.a. DLIN) assumption. The scheme depends on some global parameters, that for simplicity we consider to be the same as the ones for the PKE scheme from the previous section. Let $\psi : \mathbb{Z}_q^2 \to \mathbb{Z}_q^3$ be such that $\psi(r_1, r_2) = (r_1, r_2, r_1 + r_2)$.

- $\underline{\mathsf{Setup}(1^\kappa)}$: Sample $\mathbf{p} \leftarrow \mathbb{Z}_q^3$, and output $\omega = [\mathbf{x}]_g$.
- $\underline{\mathsf{Commit}(\omega, m)}$: Sample $\mathbf{r} \leftarrow \mathbb{Z}_q^2$, and compute $\gamma = [m \cdot \mathbf{x} + \psi(\mathbf{r})]_g$ where $\omega = [\mathbf{x}]_g$.

5.3 The Label-Malleable NIZK

In [33], the authors show that lm-NIZK are a special kind of Controlled-Malleable NIZK (cm-NIZK) argument of knowledge systems [15], and provide a generic transformation to obtain lm-NIZK. In particular, their paradigm identifies a sufficient set of conditions which we review below.

Definition 14. For a relation $\mathcal{R}$, and a set of transformations Φ on the set of labels Λ, we say $(\mathcal{R}, \Phi)$ is *label-malleable friendly* if the following properties hold.

1. **Representable statements and labels:** Any instance and witness of $\mathcal{R}$ can be represented as a set of group elements; i.e., there are efficiently computable bijections $F_{\mathsf{stm}} : \mathcal{L}_{\mathcal{R}} \to \mathbb{G}_{i_{\mathsf{stm}}}^{d_{\mathsf{stm}}}$ for some d_{stm} and i_{stm}, $F_{\mathsf{wit}} : \mathcal{W}_{\mathcal{R}} \to G_{i_{\mathsf{wit}}}^{d_{\mathsf{wit}}}$ for some d_{wit} and i_{wit}, and where $\mathcal{W}_{\mathcal{R}}$ is the witness space for the relation $\mathcal{R}$, and $F_{\mathsf{lbl}} : \Lambda \to \mathbb{G}_{i_{\mathsf{lbl}}}^{d_{\mathsf{lbl}}}$ for some d_{lbl} and $i_{\mathsf{lbl}} = i_{\mathsf{stm}}$.

2. **Representable transformations:** Any transformation in Φ can be represented as a set of group elements; i.e., there is an efficiently computable bijection $F_{\mathsf{trs}} : \Phi \to G_{i_{\mathsf{trs}}}^{d_{\mathsf{trs}}}$ for some d_{trs} and some i_{trs}.

3. **Provable statements:** We can prove the statement "$(x, w) \in \mathcal{R}$" (using the above representation for x and w) via pairing product equations; i.e., there is a pairing product statement that is satisfied by $F_{\mathsf{stm}}(x)$ and $F_{\mathsf{wit}}(w)$ if and only if $(x, w) \in \mathcal{R}$.

4. **Provable transformations:** We can prove the statement "$\phi(\lambda') = \lambda \wedge \phi \in \Phi$" (using the above representations for labels λ, λ' and transformation ϕ) via a pairing product equation, i.e., there is a pairing product statement that is satisfied by $F_{\mathsf{trs}}(\phi), F_{\mathsf{lbl}}(\lambda), F_{\mathsf{lbl}}(\lambda')$ if and only if $\phi \in \Phi \wedge \ \phi(\lambda') = \lambda$.

5. **Transformable transformations:** For any $\phi, \phi' \in \Phi$, there is an efficient transformation (depending on ϕ, ϕ') that takes the statement "$\phi(\lambda') = \lambda \wedge \ \phi \in \Phi$" (phrased using pairing product equations as above) and transforms it into the statement "$(\phi' \circ \phi)(\lambda') = \phi(\lambda) \wedge \ (\phi' \circ \phi) \in \Phi$" while preserving the label λ'.

Theorem 3 (Theorem 3 of [33])**.** *If the DLIN assumption holds, then we can construct a lM-NIZK with label derivation privacy from any label-malleable friendly relation and transformation set* $(\mathcal{R}, \Phi)$.

Consider now the relation and transformation set $(\mathcal{R}^0_\rho, \Phi^0_\rho)$ defined below:

$$\begin{aligned}\mathcal{R}^0_\rho &= \{([\alpha]_g, [\mathbf{S}]_h) : [\alpha]_g = [\mathbf{p} \cdot \mathbf{S}_1^{\mathrm{T}}]_g\},\\ \Phi^0_\rho &= \left\{\phi_{\mathbf{B}}(\mathbf{C}, \mathbf{c}) := \left([\mathbf{B} \cdot \mathbf{C}^{\mathrm{T}}]_g, [\mathbf{B} \cdot \mathbf{c}^{\mathrm{T}}]_g\right) \ : \ \mathbf{1} = \mathbf{B} \cdot \mathbf{1}^{\mathrm{T}}\right\}.\end{aligned}$$

where $\rho = (\mathtt{params}, [\mathbf{p}]_g, [\mathbf{w}]_h) \leftarrow \mathsf{Setup}(1^\kappa)$ is as in the PKE scheme from Section 5.1. Notice that the set of all the possible updates of a ciphertext, i.e.

$$\left\{\phi : \ \phi(\cdot) = \mathsf{UpdateC}(\rho, \cdot\, ; \mathbf{B}), \mathbf{B} \in \mathbb{Z}_q^{n \times n}, \mathbf{1}_n = \mathbf{B} \cdot \mathbf{1}_n^{\mathrm{T}}, \mathsf{rank}(\mathbf{B}) = d\right\},$$

is a subset of Φ^0_ρ. We show that $(\mathcal{R}^0_\rho, \Phi^0_\rho)$ is label-malleable friendly.

Representable statements and labels: Notice that $\mathcal{L}_{\mathcal{R}^0_\rho} \subseteq \mathbb{G}_1$, while the set of valid labels is equal to $\mathbb{G}_1^{n \times m} \times \mathbb{G}_1^n$.

Representable transformations: We can describe a transformation $\phi_{\mathbf{B}} \in \Phi^0_\rho$ as a matrix of elements $[\mathbf{B}]_h \in \mathbb{G}_2^{n \times n}$.

Provable statements: The relation $\mathcal{R}^0_\rho$ can be represented by the pairing product statement $[\alpha]_g \bullet [1]_h = [\mathbf{p}] \bullet [\mathbf{S}_1^{\mathrm{T}}]_h$.

Provable transformations: Given a transformation $\phi_{\mathbf{B}} \in \Phi^0_\rho$ and labels $c = ([\mathbf{C}]_g, [\mathbf{c}]_g), c' = ([\mathbf{C}']_g, [\mathbf{c}']_g)$, the statement "$\phi_{\mathbf{B}}(c') = c \wedge \phi_{\mathbf{B}} \in \Phi$" can be expressed as a system of pairing product equations:

$$\begin{cases} [\mathbf{B}]_h \bullet [\mathbf{C}'^{\mathrm{T}}]_g = [\mathbf{C}]_g \bullet [1]_h \\ [\mathbf{B}]_h \bullet [\mathbf{c}'^{\mathrm{T}}]_g = [\mathbf{c}]_g \bullet [1]_h \\ [\mathbf{B}]_h \bullet [\mathbf{1}^{\mathrm{T}}]_g = [\mathbf{1}]_f. \end{cases} \tag{8}$$

Transformable transformations: Let $\phi_{\mathbf{B}}, c, c'$ be as before, and let $\phi_{\mathbf{B}'} \in \Phi^0_\rho$. We show that we can transform the system in Eq. (8) into a system of pairing product equations corresponding to the statement $(\phi_{\mathbf{B}'} \circ \phi_{\mathbf{B}})(c') = \phi_{\mathbf{B}'}(c) \wedge (\phi_{\mathbf{B}'} \circ \phi_{\mathbf{B}}) \in \Phi$. In particular, we obtain

$$\begin{cases} [\mathbf{B}' \cdot \mathbf{B}]_h \bullet [\mathbf{C}'^{\mathrm{T}}]_g = [\mathbf{B}' \cdot \mathbf{C}^{\mathrm{T}}]_g \bullet [1]_h \\ [\mathbf{B}' \cdot \mathbf{B}]_h \bullet [\mathbf{c}'^{\mathrm{T}}]_g = [\mathbf{B}' \cdot \mathbf{c}^{\mathrm{T}}]_g \bullet [1]_h \\ [\mathbf{B}' \cdot \mathbf{B}]_h \bullet [\mathbf{1}^{\mathrm{T}}]_g = [\mathbf{1}]_f. \end{cases}$$

For any $k \in \mathbb{N}$, consider the PKE scheme $\mathcal{PKE}^{\times k}$ defined in Section 5.1, let $\mathcal{C}_{\mathcal{PKE}^{\times k}} = (\mathcal{C}_{\mathcal{PKE}})^k$ be the ciphertext space of $\mathcal{PKE}^{\times k}$, and $\Phi_\rho^{0\times k} = (\Phi_\rho^0)^k$. One can easily show that, for any positive polynomial $k(\kappa)$, the tuple $(\mathcal{R}_\rho^0, \Phi_\rho^{0\times k})$ is label-malleable friendly.

Next, consider the relation and transformation set $(\mathcal{R}_\rho^1, \Phi_\rho^1)$ defined below:

$$\begin{aligned} \mathcal{R}_\rho^1 &= \{([\mathbf{x}]_g, [\alpha]_g), (m, \mathbf{r}) : [\alpha]_g = [m \cdot \mathbf{x} + \psi(\mathbf{r})]_g\}, \\ \Phi_\rho^1 &= \left\{\phi_{\mathbf{A}}(\mathbf{S}) := \left([\mathbf{A} \cdot \mathbf{S}^{\mathrm{T}}]_h\right) : \ \mathbf{1} = \mathbf{A} \cdot \mathbf{1}^{\mathrm{T}}\right\}, \end{aligned}$$

where $\rho = (\mathtt{params}, [\mathbf{p}]_g, [\mathbf{w}]_h) \leftarrow \mathsf{Setup}(1^\kappa)$. Notice that the set of all the possible updates of a secret key, i.e.

$$\left\{\phi : \ \phi(\cdot) = \mathsf{UpdateS}(\rho, \cdot\, ; \mathbf{A}), \mathbf{A} \in \mathbb{Z}_q^{n\times n}, \mathbf{1}_n = \mathbf{A} \cdot \mathbf{1}_n^{\mathrm{T}}, \mathsf{rank}(\mathbf{A}) = d\right\},$$

is a subset of Φ_ρ^1. We show that $(\mathcal{R}_\rho^1, \Phi_\rho^1)$ is label-malleable friendly.

Representable statements and labels: Notice that $\mathcal{L}_{\mathcal{R}_\rho^1} \subseteq \mathbb{G}_1$, while the set of valid labels is the equal to $\mathbb{G}_2^{n\times m}$.

Representable transformations: We can describe a transformation $\phi_{\mathbf{A}} \in \Phi_\rho^1$ as a matrix of elements $[\mathbf{A}]_h \in \mathbb{G}_2^{n\times n}$.

Provable statements: The relation $\mathcal{R}_\rho^1$ can, obviously, be represented by the pairing product equation $[\alpha]_g = [m \cdot \mathbf{x} + \psi(\mathbf{r})]_g$.

Provable transformations: Given a transformation $\phi_{\mathbf{A}} \in \Phi_\rho^1$, and labels $sk = [\mathbf{S}]_h$, $sk' = [\mathbf{S}']_h$, the statement "$\phi_{\mathbf{A}}(sk') = sk \wedge \phi_{\mathbf{A}} \in \Phi^1$" can expressed as a system of pairing product equations:

$$\begin{cases} [\mathbf{A}]_g \bullet [\mathbf{S}'^{\mathrm{T}}]_g = [\mathbf{S}]_g \bullet [1]_h \\ [\mathbf{A}]_g \bullet [\mathbf{1}^{\mathrm{T}}]_h = [\mathbf{1}]_f. \end{cases} \tag{9}$$

Transformable transformations: Let $\phi_{\mathbf{A}}, sk, sk'$ be as before, and let $\phi_{\mathbf{A}'} \in \Phi_\rho^1$. We show that we can transform the system in Eq. (9) into a system of pairing product equations for the statement $(\phi_{\mathbf{A}'} \circ \phi_{\mathbf{A}})(sk') = \phi_{\mathbf{A}'}(c) \wedge (\phi_{\mathbf{A}'} \circ \phi_{\mathbf{A}}) \in \Phi$. In particular, we obtain

$$\begin{cases} [\mathbf{A}' \cdot \mathbf{A}]_g \bullet [\mathbf{S}'^{\mathrm{T}}]_h = [\mathbf{A}' \cdot \mathbf{S}^{\mathrm{T}}]_g \bullet [1]_h \\ [\mathbf{A}' \cdot \mathbf{A}]_g \bullet [\mathbf{1}^{\mathrm{T}}]_h = [\mathbf{1}]_f. \end{cases}$$

6 Applications

6.1 Tamper-Resilient Signatures

Consider a signature scheme $\mathcal{SS}$ (cf. Section 2.5). We would like to protect $\mathcal{SS}$ against tampering attacks with the memory, storing the signing key. As observed originally by Gennaro *et al.* [41], however, without further assumptions, this goal is too ambitious. Their attack can be circumvented by either assuming the self-destruct capability, or a key-update mechanism.

Interestingly, Fujisaki and Xagawa [40] observed that, whenever the key-update mechanism is invoked only after an invalid output is generated, the goal of constructing tamper-resilient signature is impossible, even assuming the self-destruct capability. The idea behind the attack is to generate two valid pairs of independent signing/verification keys, and thus to overwrite the original secret key with either of the two sampled signing keys in order to signal one bit of the original key. Note that such an attack never generates invalid signatures, thus rendering both the self-destruct capability and a key-update mechanism useless.

6.1.1 Split-State Signature Schemes

A split-state signature scheme is a tuple of algorithms $\mathcal{SS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Rfrsh})$ specified as follows. (1) The (randomized) algorithm Setup takes as input the security parameter $\kappa \in \mathbb{N}$, and outputs public parameters $\rho \in \{0,1\}^*$; all algorithms are implicitly given ρ as input. (2) The (randomized) algorithm Gen takes as input the public parameters $\rho \in \{0,1\}^*$, and outputs a pair of verification/signing key (vk, sk), such that sk consists of two shares (S_0, S_1). (3) The (randomized) algorithm Sign takes as input two shares S_0, S_1, and a message $m \in \{0,1\}^*$, and outputs a signature σ. (4) The (deterministic) algorithm Vrfy takes as input the verification key vk, and a message/signature pair (m, σ), and outputs a decision bit. (5) The (randomized) algorithm Rfrsh takes as input two shares S_0, S_1, and returns two updated shares S_0', S_1'.

We say that $\mathcal{SS}$ satisfies completeness if for all ρ output by $\mathsf{Setup}(1^\kappa)$, for all $(vk, (S_0, S_1))$ output by $\mathsf{Gen}(\rho)$, and for all messages $m \in \{0,1\}^*$, the following holds:

$$\mathbb{P}\left[\mathsf{Vrfy}(vk, \mathsf{Sign}(sk, \mathsf{Rfrsh}(S_0, S_1), m))\right] \geq 1 - \nu(\kappa),$$

for a negligible function ν.

Informally, a split-state signature scheme is secure against continuous leakage and tampering attacks if no efficient adversary can forge a signature on a "fresh" message, even if it is given access to the following oracles: (i) A leakage oracle that returns entropy-bounded leakage on the shared secret key (computed on the two shares of the key independently); (ii) A tampering oracle that returns signatures on adaptively chosen messages, computed with a related secret key (obtained by modifying the two shares of the key independently). A formal definition (inspired by the work of Halevi and Lin [44] on after-the-fact leakage in PKE) follows below.

Definition 15 (Tamper-resilient signatures)**.** Let $\mathcal{SS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a split-state signature scheme. For any $\ell \in \mathbb{N}$, we say that $\mathcal{SS}$ is ℓ-secure against continuous leakage and tampering attacks, if for all PPT ℓ-valid adversaries A there is a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\mathbb{P}\left[\mathbf{Exp}^{\text{clt-cma}}_{\mathcal{SS},\mathsf{A}}(\kappa, \ell) = 1\right] \leq \nu(\kappa),$$

where the experiment is defined in Fig. 7.

6.1.2 Construction and Analysis

Let $\mathcal{SS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{CS} = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ be a coding scheme. Consider the following construction of a split-state signature scheme $\mathcal{SS}^* = (\mathsf{Setup}^*, \mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*, \mathsf{Rfrsh}^*)$.

- $\mathsf{Setup}^*(1^\kappa)$: Run $\omega \leftarrow \mathsf{Init}(1^\kappa)$ and output $\rho := \omega$.
- $\mathsf{Gen}^*(\rho)$: Parse $\rho := \omega$, run $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and $(C_0, C_1) \leftarrow \mathsf{Encode}(\omega, sk)$; return vk and $(S_0, S_1) := (C_0, C_1)$.

```
Experiment $\mathbf{Exp}^{\text{clt-cma}}_{\mathcal{SS},\mathsf{A}}(\kappa,\ell)$:
$\rho \leftarrow \mathsf{Setup}(1^\kappa)$; $\mathcal{Q} := \emptyset$
$(vk, (S_0, S_1)) \leftarrow \mathsf{Gen}(\rho)$ ; $S := (S_0, S_1)$
$(m^*, \sigma^*) \leftarrow (\mathsf{A}(\rho, vk) \leftrightarrows \mathcal{O}_{\mathsf{sign}}(S), \mathcal{O}_{\mathsf{Rfrsh}}(S), \mathcal{O}_\infty(S))$
Return 1 iff:
    (i) $m^* \not\in \mathcal{Q}$
    (ii) $\mathsf{Vrfy}(vk, (m^*, \sigma^*)) = 1$

Oracle $\mathcal{O}_{\mathsf{sign}}(S_0, S_1, m, f_0, f_1)$:
$(\widetilde{S}_0, \widetilde{S}_1) = (f_0(S_0), f_1(S_1))$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
$\sigma \leftarrow \mathsf{Sign}((\widetilde{S}_0, \widetilde{S}_1), m)$
If $\sigma = \perp$
    Return $\perp$
    $(S'_0, S'_1) \leftarrow \mathsf{Rfrsh}(S_0, S_1)$
    $(S_0, S_1) := (S'_0, S'_1)$
Else
    Return $\sigma$

Oracle $\mathcal{O}_{\mathsf{Rfrsh}}(S_0, S_1)$:
$(S'_0, S'_1) \leftarrow \mathsf{Rfrsh}(S_0, S_1)$
$(S_0, S_1) := (S'_0, S'_1)$
```

Figure 7: Experiment defining unforgeability of a split-state signature scheme against continuous leakage and tampering attacks.

- $\mathsf{Sign}^*((S_0, S_1), m)$: Parse $(S_0, S_1) := (C_0, C_1)$, and let $\widetilde{sk} = \mathsf{Decode}(\omega, (C_0, C_1))$; if the result is $\perp$, set $\sigma := \perp$. Else, let $\sigma \leftarrow \mathsf{Sign}(\widetilde{sk}, m)$. Output σ.
- $\mathsf{Vrfy}^*(vk, (m, \sigma))$: Output $\mathsf{Vrfy}(vk, (m, \sigma))$.
- $\mathsf{Rfrsh}^*(S_0, S_1)$: Parse $(S_0, S_1) := (C_0, C_1)$ and return $(C'_0, C'_1) \leftarrow \mathsf{Rfrsh}(\omega, C_0, C_1)$.

We prove the following theorem.

Theorem 4. *Assume that $\mathcal{SS}$ is a EUF-CMA signature scheme, and that $\mathcal{CS}$ is a refreshable continuously non-malleable and ℓ-leakage-resilient split-state code, then the above defined split-state signature scheme $\mathcal{SS}^*$ is ℓ-secure against continuous leakage and tampering attacks.*

In the proof we define an intermediate experiment where tampering and leakage queries are answered by running the simulator of the underlying R-CNMC. Whenever the simulator returns $\diamond$, we replace the output of the simulator with a signature computed using the original secret key. By the security of the R-CNMC, the above experiment is computationally indistinguishable from the real experiment. Next, we show that no PPT adversary can forge in the intermediate experiment. The proof is down to the unforgeability of the underlying signature scheme Π. The main observation is that tampering queries can now either be simulated using the simulator of the R-CNMC, or using the signing oracle (in case the simulator returns $\diamond$).

Proof sketch. Let $\mathbf{G}^1_{\mathsf{A}}(\kappa) := \mathbf{Exp}^{\text{clt-cma}}_{\mathcal{SS},\mathsf{A}}(\kappa,\ell)$ be the experiment defined in Fig. 7, and denote by $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$ the simulator for the non-malleable code. Consider the hybrid experiment $\mathbf{G}^2_{\mathsf{A}}(\kappa)$ where we make two changes: (i) The public parameters ω are computed using the simulator $\mathsf{S}_0(1^\kappa)$; (ii) The signature oracle $\mathcal{O}_{\mathsf{sign}}(S_0, S_1)$ is modified in such a way that, instead of running Decode as in the original experiment, we first run the simulator S_1 upon input (f_0, f_1) in order to obtain a value $\widetilde{sk} \in \mathcal{SK} \cup \{\diamond\}$, and then we proceed as before unless $\widetilde{sk} = \diamond$, in which case the query is answered with $\sigma \leftarrow \mathsf{Sign}(sk, m)$.

Lemma 19. *For all PPT adversaries* A*, there is a negligible function* $\nu_{1,2} : \mathbb{N} \to [0, 1]$ *such that*

$$\left|\mathbb{P}\left[\mathbf{G}^1_{\mathsf{A}}(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}^2_{\mathsf{A}}(\kappa) = 1\right]\right| \le \nu_{1,2}(\kappa).$$

Proof. Follows by a straightforward reduction to the non-malleability of the underlying code, as the only difference between the two games is that in the former game the tampering oracle always runs Sign on the output of $\mathsf{Decode}(\omega, f_0(C_0), f_1(C_1))$, whereas in the latter game we replaced the output of the decoding algorithm with the output of the simulator of the underlying code. □

Lemma 20. *For all PPT adversaries* A*, there is a negligible function* $\nu_2 : \mathbb{N} \to [0,1]$ *such that*

$$\mathbb{P}\left[\mathbf{G}^2_{\mathsf{A}}(\kappa) = 1\right] \leq \nu_2(\kappa).$$

Proof. Follows by a straightforward reduction to the unforgeability of the underlying signature scheme, since the reduction can simulate signature queries on which the non-malleable code simulator returns $\diamond$ by using the target signing oracle (in the definition of EUF-CMA), whereas all other queries are already answered by running Sign on the output of the simulator of the underlying code. □

□

6.2 Tamper-Resilient RAM

A read-only RAM program $\Lambda = (\Pi, \mathcal{D})$ consists of a next instruction function Π, a state state stored in a non-tamperable but non-persistent register, and some database $\mathcal{D}$. The next instruction function Π takes as input the current state state and input inp, and outputs an instruction I and a new state state'. The initial state is set to $(\mathsf{start}, \star)$.

A RAM compiler is a tuple of algorithms $\Sigma = (\mathsf{Setup}, \mathsf{CompMem}, \mathsf{CompNext})$ specified as follows. (1) Algorithm Setup takes as input the security parameter 1^κ, and outputs an untamperable CRS ω; (2) The memory compiler $\mathsf{CompMem}$ takes as input the CRS ω, and a database $\mathcal{D}$, and outputs a database $\widehat{\mathcal{D}}$ along with an initial internal state state; (3) The next instruction function Π is compiled to $\widehat{\Pi}$ using $\mathsf{CompNext}$ and the CRS. To define security, we compare two experiments (cf. Fig. 8). The real experiment features an adversary A that is allowed, via the interface doNext, to execute RAM programs on chosen inputs *step-by-step*; upon input x, oracle $\mathsf{doNext}(x)$ outputs the result of a single step of the computation, as well as the memory location that is accessed during that step. Additionally, adversary A can also apply tampering attacks that are parameterized by two families of functions $\mathcal{F}_{\mathsf{mem}}$ and $\mathcal{F}_{\mathsf{bus}}$, where each function $f \in \mathcal{F}_{\mathsf{mem}}$ is applied to the compiled memory, whilst each function $f \in \mathcal{F}_{\mathsf{bus}}$ is applied to the data in transit on the bus.

The ideal experiment features a simulator S that is allowed, via the interface $\mathsf{Execute}$, to execute RAM programs on chosen inputs. Upon input x, oracle $\mathsf{Execute}(x)$ outputs the result of the entire computation and the list of all the memory locations that were accessed during that computation. Briefly, a RAM compiler is tamper-resilient if for all possible logics Π, and all efficient adversaries A, there exists a simulator S such that the real and ideal experiment are computationally indistinguishable. A formal definition follows.

Definition 16 (Tamper simulatability)**.** A compiler $\Sigma = (\mathsf{Setup}, \mathsf{CompMem}, \mathsf{CompNext})$ is tamper simulatable w.r.t. $(\mathcal{F}_{\mathsf{bus}}, \mathcal{F}_{\mathsf{mem}})$ if for every next instruction function Π, and for every PPT adversary A, there exists a PPT simulator S and a negligible function $\nu : \mathbb{N} \to [0,1]$ such that, for all PPT distinguisher D and any database $\mathcal{D}$, we have that:

$$\left|\mathbb{P}\left[\mathsf{D}(\mathbf{TamperExec}^{\mathcal{F}_{\mathsf{bus}},\mathcal{F}_{\mathsf{mem}}}_{\mathsf{A},\Sigma,\Lambda}(\kappa)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathbf{IdealExec}_{\mathsf{S},\Lambda}(\kappa)) = 1\right]\right| \leq \mathrm{negl}(\kappa)$$

with $\Lambda := (\Pi, \mathcal{D})$, and where the experiments $\mathbf{TamperExec}^{\mathcal{F}_{\mathsf{bus}},\mathcal{F}_{\mathsf{mem}}}_{\mathsf{A},\Sigma,\Lambda}$ and $\mathbf{IdealExec}_{\mathsf{S},\Lambda}(\kappa)$ are defined in Fig. 8.

```
Experiment $\mathbf{TamperExec}^{\mathcal{F}_{\mathsf{bus}},\mathcal{F}_{\mathsf{mem}}}_{\mathsf{A},\Sigma,\Lambda}(k)$:
$\omega \leftarrow \mathsf{Setup}(1^\kappa)$;
Parse $\Lambda$ as $(\bar{\mathcal{D}}, \bar{\Pi})$; $\mathcal{Q} \leftarrow \emptyset$;
$\mathcal{D} \leftarrow \mathsf{CompMem}(\omega, \bar{\mathcal{D}})$, $\mathcal{D}' \leftarrow \mathcal{D}$;
$\Pi \leftarrow \mathsf{CompNext}(\omega, \bar{\Pi})$;
$b \leftarrow \big(\mathsf{A}(\omega) \leftrightarrows \mathsf{doNext}((\mathcal{D}', \Pi), \cdot), \mathcal{O}_{\mathsf{tamp}}(\cdot)\big)$;
Return $(b, \mathcal{Q})$.

Experiment $\mathbf{IdealExec}_{\mathsf{S},\Lambda}(\kappa)$:
$\mathcal{Q} \leftarrow \emptyset$;
$b \leftarrow \big(\mathsf{S}(1^\kappa) \leftrightarrows \mathsf{Execute}(\Lambda, \cdot), \mathsf{Add}(\cdot)\big)$;
Return $(b, \mathcal{Q})$.

Oracle $\mathsf{Add}(x)$:
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$;
```

```
Oracle $\mathcal{O}_{\mathsf{tamp}}$:
Upon $(\mathtt{TampMem}, f)$:
    If $f \in \mathcal{F}_{\mathsf{mem}}$, then set $\mathcal{D} \leftarrow f(\mathcal{D})$.
Upon $(\mathtt{TampBus}, f)$:
    If $f \in \mathcal{F}_{\mathsf{bus}}$, then set $\mathcal{D}' \leftarrow f(\mathcal{D})$.

Oracle $\mathsf{doNext}((\mathcal{D}, \Pi), x)$:
If $\mathsf{state} = (\mathsf{start}, \star)$
    $\mathsf{inp} \leftarrow x$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$
$(\mathsf{I}, \mathsf{state}') \leftarrow \Pi(\mathsf{state}, \mathsf{inp})$
If $\mathsf{I} = (\mathsf{read}, v)$
    $\mathsf{inp} \leftarrow \mathcal{D}[v]$; $\mathsf{state} := \mathsf{state}'$
If $\mathsf{I} = (\mathsf{stop}, z)$, then $\mathsf{state} \leftarrow (\mathsf{start}, \star)$
Else, $\mathsf{state} := \mathsf{state}'$
Output $\mathsf{I}$.

Oracle $\mathsf{Execute}((\mathcal{D}, \Pi), x)$:
$\mathsf{state} \leftarrow (\mathsf{start}, \star)$, $\mathcal{I} \leftarrow \emptyset$;
repeat $\mathsf{I}' \leftarrow \mathsf{doNext}((\mathcal{D}, \Pi), x)$; $\mathcal{I} \leftarrow \mathcal{I} \| \mathsf{I}'$;
until $\mathsf{I}' = (\mathsf{stop}, v)$;
Output $\mathcal{I}$
```

Figure 8: Experiments defining security of a RAM compiler.

6.2.1 Tampering on the Bus

Roughly, our RAM compiler encodes a randomly generated secret key k of an authenticated encryption scheme using a R-CNMC, creating a codeword (K_0, K_1). Next, it encrypts each data block in the memory under the key k. Let $\mathcal{E}$ be the encrypted blocks. The compiled database consists of two parts $(K_0, \mathcal{E})$ and $(K_1, \mathcal{E})$. A fundamental twist w.r.t. to previous work is that we include the encrypted database in both the parts. This feature allows us to prove security in the split-state model, at the price of requiring the following additional property for the underlying R-CNMC.

Definition 17 (Codeword correlation)**.** A split-state code $\mathcal{CS} = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ has α-correlated codewords if for all ω output by Init, any message $M \in \mathcal{M}$, and $(C_0, C_1) \leftarrow \mathsf{Enc}(\omega, M)$, the following holds for all $\beta \in \{0,1\}$: $\widetilde{\mathbb{H}}_\infty(C_\beta | C_{1-\beta}) \geq \mathbb{H}_\infty(C_\beta) - \alpha$.

It is easy to verify that the code from Section 4 has α-correlated codewords for $\alpha := \max\{\delta(\kappa), \mu(\kappa)\}$ (where $\delta := \log|\mathcal{M}|$ and $\delta := \log|\mathcal{PK}|$).

Let $\mathcal{SKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a secret-key encryption scheme. Let $\mathcal{CS} = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ be a R-CNMC. Our tamper-resilient read-only RAM compiler $\Sigma = (\mathsf{Setup}, \mathsf{CompMem}, \mathsf{CompNext})$ works as follows.

- $\underline{\mathsf{Setup}(1^\kappa)}$: Run $\omega \leftarrow \mathsf{Init}(1^\kappa)$ and output ω.
- $\underline{\mathsf{CompMem}(\omega, \mathcal{D})}$: Run $k \leftarrow \mathsf{KGen}(1^\kappa)$ and $(K_0, K_1) \leftarrow \mathsf{Encode}(\omega, k)$. For each d_i in $\mathcal{D} = (d_1, \ldots, d_n)$ do $c_i \leftarrow \mathsf{Enc}(k, d_i \| i)$ and let $\mathcal{E} = (c_1, \ldots, c_n)$. Output $(\mathcal{D}_0, \mathcal{D}_1)$, where $\mathcal{D}_0 = (K_0, \mathcal{E})$ and $\mathcal{D}_1 = (K_1, \mathcal{E})$), and let the initial internal state be $\mathsf{state} = ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$.
- $\underline{\mathsf{CompNext}(\omega, \Pi)}$: The compiled next instruction function $\widehat{\Pi}$ does the following: Upon input $(\mathsf{state}, \mathsf{inp})$, parse $\mathsf{state} = (\mathsf{state}_{\mathsf{RAM}} \| k_{\mathsf{last}} \| \mathsf{I}_{\mathsf{last}})$. Hence:

```
– If $\mathsf{state}_{\mathsf{RAM}} = (\mathsf{start}, \star)$
    1. $(\mathsf{I}, \mathsf{state}'_{\mathsf{RAM}}) \leftarrow \Pi(\mathsf{state}_{\mathsf{RAM}}, \mathsf{inp})$;
    2. Read from $\mathcal{D}_0$ the value $K_0$ and from $\mathcal{D}_1$ the value $K_1$;
    3. Compute $k \leftarrow \mathsf{Decode}(\omega, (K_0, K_1))$;
    4. Set $\mathsf{state} \leftarrow (\mathsf{state}'_{\mathsf{RAM}} \| k \| \mathsf{I})$;
    5. Output $(\mathsf{I}, \mathsf{state})$.
– If $\mathsf{state}_{\mathsf{RAM}} \neq (\mathsf{start}, \star)$
    1. Parse $\mathsf{I}_{\mathsf{last}}$ as $(\mathtt{read}, v)$;
    2. Read $c_v$ from $\mathcal{D}_0$ and read $c'_v$ from $\mathcal{D}_1$;
    3. Compute $d' \| i' \leftarrow \mathsf{Dec}(k_{\mathsf{last}}, c_v)$;
    4. If $(c_v \neq c'_v)$ or $d' \| i' = \perp$ or $i' \neq v$, then signal $\mathcal{D}_0$ and $\mathcal{D}_1$ to **refresh**, set
       $\mathsf{state} \leftarrow ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$ and output $\perp$.
    5. Else $\mathsf{inp} \leftarrow d'$, compute $(\mathsf{I}, \mathsf{state}'_{\mathsf{RAM}}) \leftarrow \Pi(\mathsf{state}_{\mathsf{RAM}}, \mathsf{inp})$;
    6. If $\mathsf{I} = (\mathsf{stop}, z)$, then set $\mathsf{state} \leftarrow ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$ and output $(\mathsf{I}, \mathsf{state})$;
    7. Else set $\mathsf{state} \leftarrow (\mathsf{state}'_{\mathsf{RAM}} \| k_{\mathsf{last}} \| \mathsf{I})$ and output $(\mathsf{I}, \mathsf{state})$.
```

Theorem 5. *Assume that $\mathcal{SKE}$ is an authenticated encryption scheme with ciphertexts size bounded by $q(\kappa, n) \in \mathrm{poly}(\kappa)$, and that $\mathcal{CS}$ is a $(q + \alpha + O(\log \kappa))$-leakage-resilient and continuously non-malleable code with split-state refresh, that additionally has α-correlated codewords. Then, the above construction defines a tamper-resilient RAM compiler w.r.t. $(\mathcal{F}_{\mathsf{bus}}, \emptyset)$, where $\mathcal{F}_{\mathsf{bus}}$ is the class of split-state tampering functions applied on the bus.*

Proof. Let $\mathcal{C}_E \subset \{0,1\}^{q(\kappa,n)}$ and $\mathcal{C}^0 \times \mathcal{C}^1$ be, respectively, the ciphertext space of $\mathcal{SKE}$ and the codeword space of $\mathcal{COM}$. Without loss of generality, we can parse a function $g \in \mathcal{F}_{\mathsf{bus}}$ as $(g\big|_{\mathsf{key}}, g\big|_{\mathsf{mem}})$ where, in turn, $g\big|_{\mathsf{key}} := (T^0, T^1)$ and $g\big|_{\mathsf{mem}} := (M^0, M^1)$. Here:

- The function T^0 has domain $\mathcal{C}^0 \times \mathcal{C}_E^n$ and co-domain $\{0,1\}^{p(\kappa)}$;
- The function T^1 has domain $\mathcal{C}^1 \times \mathcal{C}_E^n$ and co-domain $\{0,1\}^{p(\kappa)}$;
- The function M^0 has domain $\mathcal{C}^0 \times \mathcal{C}_E^n$ and co-domain $\{0,1\}^{q(\kappa,n)}$.
- The function M^1 has domain $\mathcal{C}^1 \times \mathcal{C}_E^n$ and co-domain $\{0,1\}^{q(\kappa,n)}$.

Informally, $g\big|_{\mathsf{key}}$ acts over the codeword of the NMC and $g\big|_{\mathsf{mem}}$ acts over the n ciphertexts. To prove security of our construction we need to construct a simulator S. Let $L^{i,j}_{\mathsf{cptx}}$ be the leakage function that upon input K_j first computes $\tilde{\mathcal{E}} \leftarrow M^j(K_j, \mathcal{E})$, and then outputs $\tilde{\mathcal{E}}[i]$. Namely, $L^{i,j}_{\mathsf{cptx}}$ is the function that leaks the i-th ciphertext from the tampered database. Let $(\mathsf{S}_0, \mathsf{S}_1)$ denote the simulator for $\mathcal{CS}$.

<u>Simulator S:</u>

1. Generate $\omega \leftarrow \mathsf{S}_0(1^\kappa)$, let $k \leftarrow \mathsf{KGen}(1^\kappa)$ a key for the authenticated encryption, create an encrypted dummy database $\mathcal{D}$, where each entry is an encryption of 0^κ, i.e. $\mathcal{D} = (c_1, \dots, c_n)$ where $c_i = \mathsf{Enc}(k, 0^\kappa \| i)$. Run the simulator S_1. Run the real world adversary A with input ω. Let $\mathtt{flg}_{\mathsf{key}}$ be the flag that indicates the current state of the key. Initially the flag is set to OK. Let $\mathtt{flg}_{\mathsf{start}}$ be the flag that indicates the current state of the execution. Initially the flag is set to False.

2. Upon receiving a query from the adversary A proceed as follows:

   If the query is for $\mathcal{O}_{\mathsf{tamp}}$ and it is of the kind $(\mathtt{TampBus}, g)$ and $g \in \mathcal{F}_{\mathsf{bus}}$ then set g as the *current tampering function.*

   If the query is for doNext with input x, in case $\mathtt{flg}_{\mathsf{start}} = \mathsf{False}$ then set $g' := (T^0((\cdot, \mathcal{D}), T^1(\cdot, \mathcal{D}))$ where (T^0, T^1) is $g\big|_{\mathsf{key}}$ and g is the current tampering query, forward g' to S_1, and do the following:

   (a) If the simulator outputs $\diamond$, we set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{OK}$.
   (b) If the simulator outputs $\perp$, we set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{broken}$.
   (c) The last option for the simulator is to output a valid, but unrelated, key k^*; in this case, we set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{unrelated}$.

   Finally, set the flag $\mathtt{flg}_{\mathsf{start}}$ to True. Afterwards, do the following depending on $\mathtt{flg}_{\mathsf{key}}$:

   (a') If $\mathtt{flg}_{\mathsf{key}} = \mathsf{broken}$, then send the query x to $\mathsf{Add}(\cdot)$ and answer with $\perp$.

   (b') If $\mathtt{flg}_{\mathsf{key}} = \mathsf{unrelated}$ then if $\mathtt{cnt} = 0$ send the query x to $\mathsf{Add}(\cdot)$; compute locally:
   - If $\mathsf{state}_{\mathsf{RAM}} = (\mathsf{start}, \star)$ set $\mathsf{inp} \leftarrow x$ and execute $(\mathsf{I}, \mathsf{state}'_{\mathsf{RAM}}) \leftarrow \Pi(\mathsf{state}_{\mathsf{RAM}}, \mathsf{inp})$, set $\mathsf{state} \leftarrow (\mathsf{state}'_{\mathsf{RAM}} \| k^* \| \mathsf{I})$, increment $\mathtt{cnt}$ and return I;
   - If $\mathsf{I}_{\mathsf{last}} = (\mathtt{read}, v)$ send the leakage oracle queries $(\mathtt{Leak}, (0, L^{v,0}_{\mathsf{cptx}}))$ and $(\mathtt{Leak}, (1, L^{v,1}_{\mathsf{cptx}}))$ to the simulator S_1. Let $\widetilde{c}_v$ and $\widetilde{c}'_v$ be the respective answers. Decrypt $(d' \| i') \leftarrow \mathsf{Dec}(k^*, \widetilde{c}_v)$. If $\widetilde{c}_v \neq \widetilde{c}'_v$ or $(d' \| i') = \perp$ or $i' \neq v$ then output $\perp$. Otherwise, set $\mathsf{inp} \leftarrow d'$ and compute locally $(\mathsf{I}, \mathsf{state}'_{\mathsf{RAM}}) \leftarrow \Pi(\mathsf{state}_{\mathsf{RAM}}, \mathsf{inp})$ set $\mathsf{state} \leftarrow (\mathsf{state}'_{\mathsf{RAM}} \| k^* \| \mathsf{I})$ increment $\mathtt{cnt}$ and return I.
   - If $\mathsf{I}_{\mathsf{last}} = (\mathsf{stop}, v)$ output $\mathcal{I}$, set $\mathsf{state} \leftarrow ((\mathsf{start}, \star), \|0^\kappa\|0^{\log n})$ and reset $\mathtt{cnt}$ to 0 and return I.

   (c') If $\mathtt{flg}_{\mathsf{key}} = \mathsf{OK}$ then if $\mathtt{cnt} = 0$ forward the query x to $\mathsf{Execute}(\Lambda, \cdot)$, let $\mathcal{I}$ the answer, else if $\mathtt{cnt} > 0$ retrieve $\mathcal{I}$. Parse $\mathcal{I} = (\mathsf{I}_1, \ldots, \mathsf{I}_p)$ for $p \in \mathbb{N}$. If $\mathsf{I}_{\mathtt{cnt}}$ is of the form (read, v) then send the leakage oracle queries $(\mathtt{Leak}, (0, L^{v,0}_{\mathsf{cptx}}))$ and $(\mathtt{Leak}, (1, L^{v,1}_{\mathsf{cptx}}))$ to the simulator S_1. Let $\widetilde{c}_v$ and $\widetilde{c}'_v$ be the respective answers. If $\widetilde{c}_v = \mathcal{E}[v] = \widetilde{c}'_v$ then output $\mathsf{I}_{\mathtt{cnt}}$ and increment the counter else output $\perp$.

3. In all of the above cases, whenever return $\perp$ to the adversary, force[13] the simulator S_1 to refresh and reset the flag $\mathtt{flg}_{\mathsf{key}}$ to OK and the flag $\mathtt{flg}_{\mathsf{start}}$ to False.

We prove the indistinguishability in 3 steps. Let H_i be the i-th hybrid adversary, such adversary takes as input 1^κ and the original database $\mathcal{D}$ and interacts with doNext. The latter defines the hybrid experiment $\mathbf{H}_i$, specifically $\mathbf{H}_i(\kappa) := \mathsf{D}\left(\mathsf{H}_i(1^\kappa, \mathcal{D}) \leftrightarrows \mathsf{Execute}(\Lambda, \cdot), \mathsf{Add}(\cdot)\right)$ where $\Lambda = (\Pi, \mathcal{D})$. Let the hybrid adversaries be defined as follows:

Hybrid adversary H_1. It takes as input the database $\mathcal{D}$ and executes the same code of the simulator S, but it honestly encrypts the database $\mathcal{D}$. Namely, $\mathcal{E} = (c_1, \ldots, c_n)$ where $c_i \leftarrow \mathsf{Enc}(k, (d_i \| i))$ for all $1 \leq i \leq n$.

Hybrid adversary H_2. It takes as input the database $\mathcal{D}$ and executes the same code of the hybrid adversary H_1, but in point 2.(c') it does not check that $\widetilde{c}_v = \mathcal{E}[v] = \widetilde{c}'_v$ (where

[13]This can be done by sending a tampering query that overwrites the target encoding with an invalid codeword, thus triggering a decoding error.

$\widetilde{c}_v := L^{v,0}_{\mathsf{cptx}}(K_0)$ and $\widetilde{c}'_v := L^{v,1}_{\mathsf{cptx}}(K_1)$). Instead it decrypts $(d'\|i') = \mathsf{Dec}(k^*, \widetilde{c}_v)$ and if $\widetilde{c}_v \neq \widetilde{c}'_v$ or $(d'\|i') = \perp$ or $i' \neq v$ then outputs $\perp$ and refreshes K_0, K_1. (Namely, it executes the same code the compiler $\mathsf{CompNext}$ would execute.)

Hybrid adversary H_3. It takes as input the database $\mathcal{D}$ and executes the same code of the hybrid adversary H_2, but instead of running the simulator S_1 of the R-CNMC, it samples $(K_0, K_1) \leftarrow \mathsf{Encode}(\omega, k)$, computes the decoding, the leakage function L_{cptx} and the refreshing algorithm directly on K_0, K_1. In particular, whenever it computes a decoding it does the following checks. If the decoded key k^* is $\perp$, then set the flag $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{broken}$, else if k^* is equal to the original key k, then set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{OK}$, and else set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{unrelated}$.

Lemma 21. *For any read-only RAM $\Lambda = (\Pi, \mathcal{D})$ there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $|\mathbb{P}[\mathbf{H}_1(\kappa) = 1] - \mathbb{P}[\mathbf{IdealExec}_{\mathsf{S},\Lambda}(\kappa) = 1]| \leq \nu(\kappa)$.*

Proof sketch. We reduce to the security of the authenticated encryption via a hybrid argument. Let H^i_1 be the hybrid adversary that executes the same code of H_1 but encrypts to $\mathcal{D}$ only the first i locations and the remaining are encrypted to 0. We prove that for any index $i \in [n-1]$ there exists a negligible function ν' such that

$$\left|\mathbb{P}\left[\mathbf{H}^i_1(1^\kappa) = 1\right] - \mathbb{P}\left[\mathbf{H}^{i+1}_1(1^\kappa) = 1\right]\right| \leq \nu'(\kappa),$$

where $\mathbf{H}^i_1(1^\kappa) := \mathsf{D}(\mathsf{H}^i_1(1^\kappa, \mathcal{D}) \leftrightarrows \mathsf{Execute}(\Lambda, \cdot), \mathsf{Add}(\cdot))$.

Let B the adversary that upon input 1^κ and oracle access to $\mathsf{Dec}(k, \cdot)$ executes the same code of the hybrid adversary H^i_1 but queries its own oracle for the encryption of $(\mathcal{D}[i]\|j)$ for $0 \leq j \leq i$. The adversary B outputs $(\mathcal{D}[i]\|i,\ 0)$ as challenge messages and gets c as challenge ciphertext. It is easy to see that B fully simulates H^i_1 if the challenge bit is 0 and H^{i+1}_1 if the challenge bit is 1 which implies a distinguisher against the semantic security of $\mathcal{SKE}$. □

Lemma 22. *For any read-only RAM $\Lambda = (\Pi, \mathcal{D})$ there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $|\mathbb{P}[\mathbf{H}_1(\kappa) = 1] - \mathbb{P}[\mathbf{H}_2(\kappa) = 1]| \leq \nu(\kappa)$.*

Proof sketch. The lemma follows from the authenticity of $\mathcal{SKE}$. In particular, let Forge be the event that $\widetilde{c}_v \neq \mathcal{E}[v] \neq \widetilde{c}'_v$ but $\widetilde{c}_v = \widetilde{c}'_v$ and $(d'\|i') \neq \perp$ and $i' = v$ happens in $\mathbf{H}_2$. The event Forge allows to distinguish between $\mathbf{H}_2$ and $\mathbf{H}_3$, therefore $|\mathbb{P}[\mathbf{H}_2(\kappa) = 1] - \mathbb{P}[\mathbf{H}_3(\kappa) = 1]| \leq \mathbb{P}[\mathsf{Forge}]$. We prove that if $\mathbb{P}[\mathsf{Forge}] \geq 1/p(\kappa)$ for a polynomial p, then we can break authenticity of $\mathcal{SKE}$ with the same probability. Notice that since $\widetilde{c}_v \neq \mathcal{E}[v]$ and $i' = v$ then for any $i \in [n]$ we have that $\widetilde{c}_v \neq \mathcal{E}[v]$; this is because the other ciphertexts encrypt a different index. This means that $\widetilde{c}_v$ is fresh and moreover it decrypts correctly under the key k (since $d'\|i' \neq \perp$). The reduction is straightforward and therefore the details are omitted. □

Lemma 23. *For any read-only RAM $\Lambda = (\Pi, \mathcal{D})$ there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $|\mathbb{P}[\mathbf{H}_2(\kappa) = 1] - \mathbb{P}[\mathbf{H}_3(\kappa) = 1]| \leq \nu(\kappa)$.*

Proof sketch. The lemma follows from the security of the R-CNMC. There are two things to notice. First, the function $g\big|_{\mathsf{key}}$ for any assignment of the encrypted database $c_1, \ldots, c_n$ is a valid split-state tampering function. Second, by continuously tampering with the encrypted split-state database, for each round, the adversary can leak information on K_0, K_1. In particular, both hybrid adversaries compute a sequence of leakage functions on K_0, K_1 from the set $\mathcal{L}$ defined[14] below:

$$\mathcal{L} := \left\{(L^{i,0}_{\mathsf{cptx}}, L^{i,1}_{\mathsf{cptx}}) : i \in \mathbb{N}\right\}.$$

[14]Notice that the leakage functions depends on $g\big|_{\mathsf{mem}}$ which can change during the execution of one round. For simplicity we hide this extra parameter.

We show that, for every round, the sequence of leakage functions applied during that round is below the leakage bound of $\mathcal{CS}$. Fix a round and let $\overline{L} := ((L_1^0, L_1^1), (L_2^0, L_2^1), \ldots, (L_p^0, L_p^1))$ be such a sequence, where $p(\kappa) \in \text{poly}(\kappa)$ is a polynomial.

$$\begin{aligned}
&\widetilde{\mathbb{H}}_\infty(K_j|\overline{L}(K_0, K_1)) \\
&\quad \geq \widetilde{\mathbb{H}}_\infty(K_j|L_1^j(K_j), \ldots, L_p^j(K_j),\ K_{1-j}) \\
&\quad \geq \widetilde{\mathbb{H}}_\infty(K_j|L_1^{1-j}(K_{1-j}), \ldots, L_{p-1}^{1-j}(K_{1-j}), L_p^j(K_j),\ K_{1-j}) \\
&\quad \geq \widetilde{\mathbb{H}}_\infty(K_j|L_1^{1-j}(K_{1-j}), \ldots, L_{p-1}^{1-j}(K_{1-j}), L_p^j(K_j),\ K_{1-j}) \\
&\quad \geq \widetilde{\mathbb{H}}_\infty(K_j|p, L_p^j(K_j),\ K_{1-j}) \\
&\quad \geq \mathbb{H}_\infty(K_j) - O(\log \kappa) - q(\kappa, n) - \alpha(\kappa).
\end{aligned}$$

Where the first equation follows by the fact that, for any i, the value $L_i^{1-j}(K_{1-j})$ is a function of K_{1-j}, the second equation follows by the fact that for any $i < p$, by definition of H_1 and H_2, $L_i^{1-j}(K_{1-j}) = L_i^j(K_j)$ holds, and in the forth equation we applied the α-correlated-codeword property of $\mathcal{CS}$ and the bound on the size of a ciphertext for $\mathcal{SKE}$. □

Lemma 24. $\mathbb{P}\left[\mathbf{H}_3(\kappa) = 1\right] = \mathbb{P}\left[\mathsf{D}(\mathbf{TamperExec}^{\mathcal{F}_{\mathsf{bus}},\emptyset}_{\mathsf{A},\Sigma,\Lambda}(\kappa)) = 1\right]$.

Proof sketch. For each query to doNext with input x, the hybrid experiment checks the flag $\mathtt{flg}_{\mathsf{key}}$. In particular:

- If the flag is set to broken, then it means that $\mathsf{Decode}(\widetilde{K}_0, \widetilde{K}_1)$ is $\perp$ (by inspection), therefore in this case the real experiment would run, add x to the set of queries and output $\perp$; the hybrid $\mathbf{H}_3$ does the same using its interface Add.
- If the flag is set to unrelated, then it means that $\mathsf{Decode}(\widetilde{K}_0, \widetilde{K}_1) = k^* \neq k$ (by inspection). In this case the hybrid experiment simulates perfectly the real experiment, by computing locally as would happen in the real experiment, moreover the hybrid add the query to $\mathcal{Q}$ using its interface Add.
- If the flag is set to OK, then it means that $\mathsf{Decode}(\widetilde{K}_0, \widetilde{K}_1) = k$. In this case the hybrid experiment uses its interface Execute, but before it outputs the transcript it processes it by finding the point where the real experiment would output $\perp$ (if the latter, indeed, outputs $\perp$).

□

□

6.2.2 Tampering on both the Bus and the Memory

The first compiler is not secure against adversaries that can tamper persistently[15] with the memory (in the split-state model) if we do not assume self-destruct. The reason is that only part of the memory is refreshed: the adversary can use the remaining part to backup the old encoding and then rewind the refreshing procedure. (See also the discussion in Section 1.3.) To partially overcome this problem we assume that, once a decoding error is triggered, the system can switch into a *safe mode* where the communication between the CPU and the memory is tamper free. While in safe mode, the system will perform a consistency check. To minimize the

[15]Note that tampering on the bus constitutes a form of non-persistent tampering.

dependency on the assumption we constraint the consistency check to be succinct, meaning that its complexity depends only on the security parameter and not on the size of the RAM program. Finally, if the consistency check passes, the refresh procedure will be executed otherwise a self-destruct is triggered.

The compiler. Let $\mathcal{SKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a secret-key encryption scheme. Let $\mathcal{CS} = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ be a R-CNMC. Our tamper-resilient read-only RAM compiler $\Sigma = (\mathsf{Setup}, \mathsf{CompMem}, \mathsf{CompNext})$ works as follows.

- $\underline{\mathsf{Setup}(1^\kappa)}$: Run $\omega \leftarrow \mathsf{Init}(1^\kappa)$ and output ω.
- $\underline{\mathsf{CompMem}(\omega, \mathcal{D})}$: Run $k \leftarrow \mathsf{KGen}(1^\kappa)$ and $(K_0, K_1) \leftarrow \mathsf{Encode}(\omega, k)$. For each d_i in $\mathcal{D} = (d_1, \ldots, d_n)$ do $c_i \leftarrow \mathsf{Enc}(k, d_i \| i)$ and let $\mathcal{E}, \mathcal{E}_0, \mathcal{E}_1 = (c_1, \ldots, c_n)$. Output $(\mathcal{D}_0, \mathcal{D}_1)$, where $\mathcal{D}_0 = (K_0, \mathcal{E}_0)$ and $\mathcal{D}_1 = (K_1, \mathcal{E}_1))$, and let the initial internal state be $\mathsf{state} = ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$.
- $\underline{\mathsf{CompNext}(\omega, \Pi)}$: The compiled next instruction function $\widehat{\Pi}$ does the following. Upon input $(\mathsf{state}, \mathsf{inp})$, parse $\mathsf{state} = (\mathsf{state}_{RAM} \| k_{\mathsf{last}} \| \mathsf{I}_{\mathsf{last}})$.
  - If $\mathsf{state}_{RAM} = (\mathsf{start}, \star)$
    1. $(\mathsf{I}, \mathsf{state}'_{RAM}) \leftarrow \Pi(\mathsf{state}_{RAM}, \mathsf{inp})$;
    2. Read from $\mathcal{D}_0$ the value K_0 and from $\mathcal{D}_1$ the value K_1;
    3. Compute $k \leftarrow \mathsf{Decode}(\omega, (K_0, K_1))$;
    4. If $k = \perp$ switch to **safe mode** and perform the consistency check:
       (a) Retrieve K_0, K_1 from memory and if $\mathsf{Decode}(\omega, K_0, K_1) = \perp$ then **self-destruct**;
       (b) Sample $i_1, \ldots, i_\zeta \leftarrow [n]$ and for $l \in [\zeta]$ check if $\mathcal{E}_0[i_l] \neq \mathcal{E}_1[i_l]$ then **self-destruct** else continue.

       Switch back to **normal mode**, refresh, and signal $\mathcal{D}_0$ and $\mathcal{D}_1$ to **refresh**. Set $\mathsf{state} \leftarrow ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$ and output $\perp$.
    5. Else set $\mathsf{state} \leftarrow (\mathsf{state}'_{RAM} \| k \| \mathsf{I})$;
    6. Output $(\mathsf{I}, \mathsf{state})$.
  - If $\mathsf{state}_{RAM} \neq (\mathsf{start}, \star)$
    1. Parse $\mathsf{I}_{\mathsf{last}}$ as $(\mathtt{read}, v)$;
    2. Read c_v from $\mathcal{D}_0$ and read c'_v from $\mathcal{D}_1$;
    3. Compute $d' \| i' \leftarrow \mathsf{Dec}(k_{\mathsf{last}}, c_v)$;
    4. If $(c_v \neq c'_v)$ or $d' \| i' = \perp$ or $i' \neq v$, then switch to **safe mode**, perform the consistency check, switch back to **normal mode** and signal $\mathcal{D}_0$ and $\mathcal{D}_1$ to **refresh**, set $\mathsf{state} \leftarrow ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$ and output $\perp$.
    5. Else $\mathsf{inp} \leftarrow d'$, compute $(\mathsf{I}, \mathsf{state}'_{RAM}) \leftarrow \Pi(\mathsf{state}_{RAM}, \mathsf{inp})$;
    6. If $\mathsf{I} = (\mathsf{stop}, z)$, then set $\mathsf{state} \leftarrow ((\mathsf{start}, \star) \| 0^\kappa \| 0^{\log n})$ and output $(\mathsf{I}, \mathsf{state})$;
    7. Else set $\mathsf{state} \leftarrow (\mathsf{state}'_{RAM} \| k_{\mathsf{last}} \| \mathsf{I})$ and output $(\mathsf{I}, \mathsf{state})$.

Theorem 6. *Assume that $\mathcal{SKE}$ is an authenticated encryption scheme with ciphertexts size bounded by $q(\kappa, n) \in \mathrm{poly}(\kappa)$, and that $\mathcal{CS}$ is a $(3q + n + \alpha + O(\log \kappa))$-leakage-resilient and continuously non-malleable code with split-state refresh, that additionally has α-correlated codewords. Then, assuming the system can switch to* safe mode *for* $\mathrm{poly}(\kappa)$ *times and self-destruct afterwards, the above construction defines a tamper-resilient RAM compiler w.r.t.* $(\mathcal{F}_{\mathsf{bus}}, \mathcal{F}_{\mathsf{mem}})$, *where $\mathcal{F}_{\mathsf{bus}}$ and $\mathcal{F}_{\mathsf{mem}}$ are, respectively, the class of split-state tampering functions applied on the bus and on the memory.*

Proof. The proof proceeds similarly to the proof of Theorem 5. Let $L^{i,j,f}_{\mathsf{cptx}}$ be the leakage function that upon input K_j first computes $\tilde{\mathcal{E}} \leftarrow f|^{j}_{\mathsf{mem}}(K_j, \mathcal{E})$, and then outputs $\tilde{\mathcal{E}}[i]$. Let $(\mathsf{S}_0, \mathsf{S}_1)$ denote the simulator for $\mathcal{CS}$.

<u>Simulator S:</u>

1. Generate $\omega \leftarrow \mathsf{S}_0(1^\kappa)$, let $k \leftarrow \mathsf{KGen}(1^\kappa)$ a key for the authenticated encryption, create an encrypted dummy database $\mathcal{E}_0, \mathcal{E}_1$, where each entry is an encryption of 0^κ, i.e. $\mathcal{E}_0, \mathcal{E}_1 = (c_1, \ldots, c_n)$ where $c_i = \mathsf{Enc}(k, 0^\kappa \| i)$. Run the simulator S_1. Run the real world adversary A with input ω. Let $\mathtt{flg}_{\mathsf{key}}$ be the flag that indicates the current state of the key. Initially the flag is set to OK. Let $\mathtt{flg}_{\mathsf{start}}$ be the flag that indicates the current state of the execution. Initially the flag is set to False. Set h to be the identity function.

2. Upon receiving a query from the adversary A do the following:

   If the <u>query is for $\mathcal{O}_{\mathsf{tamp}}$</u> and it is of the kind $(\mathtt{TampMem}, f)$, and $f \in \mathcal{F}_{\mathsf{mem}}$, then set $h \leftarrow h \circ f$ (where $(h \circ f)(x) = f(h(x))$). Call h the *history tampering function.*

   If the <u>query is for $\mathcal{O}_{\mathsf{tamp}}$</u> and it is of the kind $(\mathtt{TampBus}, f)$, and $f \in \mathcal{F}_{\mathsf{bus}}$, then set $g \leftarrow (h \circ f)$. Call g the *current tampering function.*

   If the <u>query is for doNext</u> with input x, in case $\mathtt{flg}_{\mathsf{start}} = \mathsf{False}$ then let g be the current tampering function as defined above, and send the query $(\mathtt{Tamp}, g|_{\mathsf{key}}((\cdot, \mathcal{E}_0), (\cdot, \mathcal{E}_1)))$. (Namely, the restriction of g to the encoding with the second part of the first (resp. second) input set to $\mathcal{E}_0$ (resp. $\mathcal{E}_1$).) Do the following depending on the output of the simulator:

   (a) If the simulator outputs $\diamond$, we set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{OK}$.
   (b) If the simulator outputs $\perp$, we set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{broken}$.
   (c) The last option for the simulator is to output a valid, but unrelated, key k^*. In this case, we set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{unrelated}$.

   Finally, set the flag $\mathtt{flg}_{\mathsf{start}}$ to True, and proceed as follows depending on $\mathtt{flg}_{\mathsf{key}}$:

   (a') If $\mathtt{flg}_{\mathsf{key}} = \mathsf{broken}$, then send the query x to $\mathsf{Add}(\cdot)$ and answer with $\perp$.
   (b') If $\mathtt{flg}_{\mathsf{key}} = \mathsf{unrelated}$ then if $\mathtt{cnt} = 0$ send the query x to $\mathsf{Add}(\cdot)$; compute locally:
   - If $\mathsf{state}_{RAM} = (\mathsf{start}, \star)$ set $\mathsf{inp} \leftarrow x$ and execute $(\mathsf{I}, \mathsf{state}'_{RAM}) \leftarrow \Pi(\mathsf{state}_{RAM}, \mathsf{inp})$, set $\mathsf{state} \leftarrow (\mathsf{state}'_{RAM} \| k^* \| \mathsf{I})$, increment $\mathtt{cnt}$ and return I;
   - If $\mathsf{I}_{\mathsf{last}} = (\mathtt{read}, v)$ send the leakage oracle queries $(\mathtt{Leak}, (0, L^{v,0}_{\mathsf{cptx}}))$ and $(\mathtt{Leak}, (1, L^{v,1}_{\mathsf{cptx}}))$ to the simulator S_1. Let $\widetilde{c}_v$ and $\widetilde{c}'_v$ be the respective answers. Decrypt $(d' \| i') \leftarrow \mathsf{Dec}(k^*, \widetilde{c}_v)$. If $\widetilde{c}_v \neq \widetilde{c}'_v$ or $(d' \| i') = \perp$ or $i' \neq v$ then output $\perp$. Otherwise, set $\mathsf{inp} \leftarrow d'$ and compute locally $(\mathsf{I}, \mathsf{state}'_{RAM}) \leftarrow \Pi(\mathsf{state}_{RAM}, \mathsf{inp})$ set $\mathsf{state} \leftarrow (\mathsf{state}'_{RAM} \| k^* \| \mathsf{I})$ increment $\mathtt{cnt}$ and return I.
   - If $\mathsf{I}_{\mathsf{last}} = (\mathsf{stop}, v)$ output $\mathcal{I}$, set $\mathsf{state} \leftarrow ((\mathsf{start}, \star), \| 0^\kappa \| 0^{\log n})$ and reset $\mathtt{cnt}$ to 0 and return I.

   (c') If $\mathtt{flg}_{\mathsf{key}} = \mathsf{OK}$ then, if $\mathtt{cnt} = 0$, forward the query x to $\mathsf{Execute}(\Lambda, \cdot)$ and let $\mathcal{I}$ be the answer. Else, if $\mathtt{cnt} > 0$, retrieve $\mathcal{I}$. Parse $\mathcal{I} = (\mathsf{I}_1, \ldots, \mathsf{I}_p)$ for $p \in \mathbb{N}$. If $\mathsf{I}_{\mathtt{cnt}}$ is of the form (read, v) then send the leakage oracle queries $(\mathtt{Leak}, (0, L^{v,0,g}_{\mathsf{cptx}}))$ and $(\mathtt{Leak}, (1, L^{v,1,g}_{\mathsf{cptx}}))$ to the simulator S_1. Let $\widetilde{c}_v$ and $\widetilde{c}'_v$ be the respective answers. If $\widetilde{c}_v = \mathcal{E}[v] = \widetilde{c}'_v$ then output $\mathsf{I}_{\mathtt{cnt}}$ and increment the counter, else output $\perp$.

In all of the above cases, whenever the value $\perp$ is returned to the adversary reset the flag $\mathtt{flg}_{\mathsf{key}}$ to OK and the flag $\mathtt{flg}_{\mathsf{start}}$ to False and set $\mathtt{cnt}$ to 0 and run the following *simulated consistency check* :

(a) Let h be the history tampering function, and set $\zeta = q(\kappa, n) \cdot \omega(\log \kappa)$. Send the query $(\mathtt{Final}, h\big|_{\mathsf{key}}((\cdot, \mathcal{E}_0), (\cdot, \mathcal{E}_1))$ to the simulator.

(b) If the simulator outputs $\perp$, then **self-destruct**.

(c) Else (namely, unrelated and same) Sample $i_1, \ldots, i_\zeta \leftarrow [n]$ and for $l \in [\zeta]$ send the leakage query $(\mathtt{Leak}, (0, L_{\mathsf{cptx}}^{i_l,0,h}))$ and $(\mathtt{Leak}, (1, L_{\mathsf{cptx}}^{i_l,1,h}))$ check if the returned values are different then **self-destruct** else continue.

(d) Leak the full containment of the encrypted databases. Specifically for $i \in [n]$ send the leakage query $(\mathtt{Leak}, (0, L_{\mathsf{cptx}}^{i_l,0,h}))$ and $(\mathtt{Leak}, (1, L_{\mathsf{cptx}}^{i_l,1,h}))$. Let $\tilde{c}_i^0$ and $\tilde{c}_i^1$ be the answers. Set $\mathcal{E}_0 \leftarrow ({c^0}_1, \ldots, {c^0}_n)$ and $\mathcal{E}_1 \leftarrow ({c^1}_1, \ldots, {c^1}_n)$.

(e) If the simulator of $\mathcal{CS}$ outputs $\tilde{K}_0, \tilde{K}_1$ then the simulation can continue by executing the code of **TamperExec** on the database $(\tilde{K}_0, \mathcal{E}_0), (\tilde{K}_1, \mathcal{E}_1)$.

We prove the indistinguishability in 3 steps. Let H_i be the i-th hybrid adversary, such adversary takes as input 1^κ and the original database $\mathcal{D}$ and interact with doNext. The latter defines the hybrid experiment $\mathbf{H}_i$, specifically $\mathbf{H}_i(\kappa) := \mathsf{D}\left(\mathsf{H}_i(1^\kappa, \mathcal{D}) \leftrightarrows \mathsf{Execute}(\Lambda, \cdot), \mathsf{Add}(\cdot)\right)$ where $\Lambda = (\Pi, \mathcal{D})$. Let the hybrid adversaries be defined as follow:

Hybrid Adversary H_1. It takes as input the database $\mathcal{D}$ and executes the same code of the simulator S, but it honestly encrypts the database $\mathcal{D}$. Namely, $\mathcal{E} = (c_1, \ldots, c_n)$ where $c_i \leftarrow \mathsf{Enc}(k, (d_i \| i))$ for all $1 \leq i \leq n$.

Hybrid Adversary H_2. It takes as input the database $\mathcal{D}$ and executes the same code of the hybrid adversary H_1, but in point 2.(c') it does not check that $\widetilde{c}_v = \mathcal{E}[v] = \widetilde{c}'_v$ (where $\widetilde{c}_v := L_{\mathsf{cptx}}^{v,0,g}(K_0)$ and $\widetilde{c}'_v := L_{\mathsf{cptx}}^{v,1,g}(K_1)$). Instead it decrypts $(d' \| i') = \mathsf{Dec}(k, \widetilde{c}_v)$, and if $\widetilde{c}_v \neq \widetilde{c}'_v$ or $(d' \| i') = \perp$ or $i' \neq v$ then outputs $\perp$ and refreshes K_0, K_1. (Namely, it executes the same code the compiler $\mathsf{CompNext}$ would execute.)

Hybrid Adversary H_3. It takes as input the database $\mathcal{D}$ and executes the same code of the hybrid adversary H_2, but instead of running the simulator S_1 of the R-CNMC, it samples $(K_0, K_1) \leftarrow \mathsf{Encode}(\omega, k)$, computes the decoding, the leakage function L_{cptx} and the refreshing algorithm directly on K_0, K_1. In particular, whenever it computes a decoding it does the following checks. If the decoded key k^* is $\perp$ then set the flag $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{broken}$, else if k^* is equal to the original key k then set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{OK}$, else set $\mathtt{flg}_{\mathsf{key}} \leftarrow \mathsf{unrelated}$.

Lemma 25. *For any read-only RAM $\Lambda = (\Pi, \mathcal{D})$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $|\mathbb{P}[\mathbf{H}_1(\kappa) = 1] - \mathbb{P}[\mathbf{IdealExec}_{\mathsf{S},\Lambda}(\kappa) = 1]| \leq \nu(\kappa)$.*

The proof of the lemma is identical to that of Lemma 21, and is therefore the proof is omitted.

Lemma 26. *For any read-only RAM $\Lambda = (\Pi, \mathcal{D})$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $|\mathbb{P}[\mathbf{H}_1(\kappa) = 1] - \mathbb{P}[\mathbf{H}_2(\kappa) = 1]| \leq \nu(\kappa)$.*

The proof of the lemma is identical to that of Lemma 22, and is therefore omitted.

Lemma 27. *For any read-only RAM $\Lambda = (\Pi, \mathcal{D})$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $|\mathbb{P}[\mathbf{H}_2(\kappa) = 1] - \mathbb{P}[\mathbf{H}_3(\kappa) = 1]| \leq \nu(\kappa)$.*

Proof sketch. Here, we use the security of the R-CNMC. There are two things to notice. First, the function $g\big|_{\mathsf{key}}$ for any assignment of the encrypted databases $\mathcal{E}_0, \mathcal{E}_1$ is a valid split-state tampering function. Second, by continuously tampering with the encrypted split-state database, for each round, the adversary can leak information on K_0, K_1. In particular, both hybrid adversaries computes a sequence of leakage functions on K_0, K_1 from the set $\mathcal{L}$ defined[16] below:

$$\mathcal{L} := \left\{ (L^{i,0,f}_{\mathsf{cptx}}, L^{i,1,f}_{\mathsf{cptx}}) : i \in \mathbb{N}, f \in \{g, h\} \right\}.$$

We show that, for every round, the sequence of leakage functions applied during that round is below the leakage bound of $\mathcal{CS}$. Fix a round, let $\overline{L} := ((L^0_1, L^1_1), (L^0_2, L^1_2), \ldots, (L^0_p, L^1_p))$ be the sequence of leakage functions sent by the simulator before $\perp$ is triggered ($p = \kappa^c$ is a polynomial). Let $L_{\mathsf{check}}(K_0, K_1)$ be the list of leakage functions that the simulator does during the simulated consistency check. Specifically:

<u>$L_{\mathsf{check}}(K_0, K_1)$</u>:

- Samples $i_1, \ldots, i_\zeta \leftarrow [n]$;
- For $l \in [\zeta]$ and $j \in \{0,1\}$ it computes $C^j_l := L^{i_l,0,h}_{\mathsf{cptx}}(K_k)$, if $C^0_l \neq C^1_l$ the it stops the cycle and outputs the list $\{(C^0_1, C^1_1), \ldots, (C^0_l, C^1_l)\}$.
- Outputs the list $\{(L^{i,0,h}_{\mathsf{cptx}}(K_0), L^{i,0,h}_{\mathsf{cptx}}(K_0)) : i \in [n]\}$.

We bound the average conditional min-entropy of K_j, where $j \in \{0,1\}$, given the view of the simulator.

$$\begin{aligned}
&\widetilde{\mathbb{H}}_\infty(K_j | \overline{L}(K_0, K_1), L_{\mathsf{check}}(K_0, K_1)) \\
&\geq \widetilde{\mathbb{H}}_\infty(K_j | L^j_1(K_j), \ldots, L^j_p(K_j),\ K_{1-j}, L_{\mathsf{check}}(K_0, K_1)) \\
&\geq \widetilde{\mathbb{H}}_\infty(K_j | L^{1-j}_1(K_{1-j}), \ldots, L^{1-j}_{p-1}(K_{1-j}), L^j_p(K_j),\ K_{1-j}, L_{\mathsf{check}}(K_0, K_1)) \\
&\geq \widetilde{\mathbb{H}}_\infty(K_j | L^{1-j}_1(K_{1-j}), \ldots, L^{1-j}_{p-1}(K_{1-j}), L^j_p(K_j),\ K_{1-j}, L_{\mathsf{check}}(K_0, K_1)) \\
&\geq \widetilde{\mathbb{H}}_\infty(K_j | p, L^j_p(K_j),\ K_{1-j}, L_{\mathsf{check}}(K_0, K_1)) \\
&\geq \widetilde{\mathbb{H}}_\infty(K_j | K_{1-j}, L_{\mathsf{check}}(K_0, K_1)) - O(\log \kappa) - q(\kappa, n).
\end{aligned}$$

In the above computation, the first equation follows by the fact that, for any i, the value $L^{1-j}_i(K_{1-j})$ is a function of K_{1-j}, the second equation follows by the fact that for any $i < p$, by definition of H_1 and H_2, $L^{1-j}_i(K_{1-j}) = L^j_i(K_j)$ holds, and in the forth equation we applied the bound on the size of a ciphertext for $\mathcal{SKE}$.

It is left to give a bound on $\widetilde{\mathbb{H}}_\infty(K_j | K_{1-j}, L_{\mathsf{check}}(K_0, K_1))$. Let $(L'^0_1, L'^1_1), , \ldots, (L'^0_{p'}, L'^1_{p'})$ where $p' \leq n$ be the output of L_{check}. Let X be a random variable that counts the number of indexes i' such that $L'^0_{i'} \neq L'^1_{i'}$. By following the same reasoning as before, it can be shown that:

$$\widetilde{\mathbb{H}}_\infty(K_j | K_{1-j}, L_{\mathsf{check}}(K_0, K_1)) \geq \widetilde{\mathbb{H}}_\infty(K_j | K_{1-j}) - \mathbb{E}[X]\, q(k + \log n).$$

Let `Abort` be the indicator random variable for the check $C^0_l \neq C^1_l$ in the computation of L_{check}. Let $B := n/q(\kappa, n)$, since $X \in \mathbb{N}$ we have that

$$\begin{aligned}
\mathbb{E}[X] &= \sum_i i \mathbb{P}[X = i] \\
&= \sum_i i \mathbb{P}[X = i \wedge \texttt{Abort} = 0] + \sum_i i \mathbb{P}[X = i \wedge \texttt{Abort} = 1] \\
&\leq (n \mathbb{P}[X > B \wedge \texttt{Abort} = 0] + B) + 1,
\end{aligned}$$

[16]Notice that the leakage functions depends on $g\big|_{\mathsf{mem}}$ which can change during the execution of one round. For simplicity we hide this extra parameter.

where in the last equation we used that when $\texttt{Abort} = 1$ then $X = 1$, and that $X \leq n$. We show that the probability that $X > B$ and $\texttt{Abort} = 0$ is negligible. Let Z be equal to the sum of ζ independent identically distributed Bernoulli random variables with parameter B/n. It is easy to see that

$$\mathbb{P}[X > B \wedge \texttt{Abort} = 0] \leq \mathbb{P}[\texttt{Abort} = 0 | X > B] \leq \mathbb{P}[Z = 0] = (1 - B/n)^{\zeta}.$$

The latter, by our choice of the parameter $\zeta = q(\kappa, n) \cdot \omega(\log \kappa)$, is negligible. Therefore we have that $\mathbb{E}[X] \leq n/q(\kappa, n) + 2$. Summing up together the amount of leakage performed by the simulator, we get $O(\log \kappa) + 3q(\kappa, n) + n + \alpha$ as required in the statement of the theorem. □

Lemma 28. $\mathbb{P}[\mathbf{H}_3(\kappa) = 1] = \mathbb{P}\left[\mathsf{D}(\mathbf{TamperExec}_{\mathsf{A},\Sigma,\Lambda}^{\mathcal{F}_{\mathsf{bus}},\mathcal{F}_{\mathsf{mem}}}(\kappa)) = 1\right]$.

Proof sketch. For each query to doNext with input x, the hybrid experiment checks the flag $\texttt{flg}_{\mathsf{key}}$. In particular:

- If the flag is set to broken then it means that $\mathsf{Decode}(\widetilde{K}_0, \widetilde{K}_1)$ is $\perp$ (by inspection), therefore in this case the real experiment would run, add x to the set of queries and output $\perp$; the hybrid $\mathbf{H}_3$ does the same using its interface Add.
- If the flag is set to $\mathsf{unrelated}$ then it means that $\mathsf{Decode}(\widetilde{K}_0, \widetilde{K}_1) = k^* \neq k$ (by inspection). In this case the hybrid experiment simulates perfectly the real experiment by computing locally as would happen in the real experiment, moreover the hybrid adds the query to $\mathcal{Q}$ using its interface Add.
- If the flag is set to OK then it means that $\mathsf{Decode}(\widetilde{K}_0, \widetilde{K}_1) = k$. In this case the hybrid experiment uses its interface $\mathsf{Execute}$, but before outputting the transcript it processes it by finding the point where the real experiment would output $\perp$ (if the latter, indeed, outputs $\perp$).

Further, notice that, for each round, in $\mathbf{H}_3$ the tampered database is computed by applying the function $h\big|_{\mathsf{mem}}$ where h is a function of the current codeword and the current tampered database. Also, h is the concatenation of all the persistent tampering functions in the current round. In particular, to build the tampered database for the next round, the hybrid experiment computes it exactly as mentioned above. Therefore, by induction on the number of rounds, the tampered database, as a random variable, is distributed exactly as in the real experiment. □

□

7 Conclusion

We have studied a generalization of the concept of continuously non-malleable codes, for the split-state model, where after a decoding error happens the original codeword gets refreshed instead of triggering a self-destruct. Importantly, the refreshing happens locally on each share of the codeword, without requiring any communication between the two parts. As we have shown, this extra feature allows to avoid self-destruct in some of the applications of non-malleable codes. It also allows naturally to obtain security against continual leakage attacks.

Open problems include whether stronger forms of non-malleability in the split-state model can be achieved, while at the same time allowing for split-state refreshing, and whether refreshable continuously non-malleable codes in the split-state model can be achieved without relying on setup assumptions.

References

[1] Divesh Aggarwal, Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Optimal computational split-state non-malleable codes. In *TCC*, pages 393–417, 2016.

[2] Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. In *STOC*, pages 459–468, 2015.

[3] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *STOC*, pages 774–783, 2014.

[4] Divesh Aggarwal, Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Leakage-resilient non-malleable codes. In *TCC*, pages 398–426, 2015.

[5] Divesh Aggarwal, Tomasz Kazana, and Maciej Obremski. Inception makes non-malleable codes stronger. In *TCC*, pages 319–343, 2017.

[6] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Explicit non-malleable codes against bit-wise tampering and permutations. In *CRYPTO*, pages 538–557, 2015.

[7] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In *TCC*, pages 375–397, 2015.

[8] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In *EUROCRYPT*, pages 881–908, 2016.

[9] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes from average-case hardness: AC0, decision trees, and streaming space-bounded tampering. In *EUROCRYPT*, pages 618–650, 2018.

[10] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[11] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

[12] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, pages 37–51, 1997.

[13] Nishanth Chandran, Vipul Goyal, Pratyay Mukherjee, Omkant Pandey, and Jalaj Upadhyay. Block-wise non-malleable codes. In *ICALP*, pages 31:1–31:14, 2016.

[14] Nishanth Chandran, Bhavana Kanukurthi, and Srinivasan Raghuraman. Information-theoretic local non-malleable codes and their applications. In *TCC*, pages 367–392, 2016.

[15] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *EUROCRYPT*, pages 281–300, 2012.

[16] Eshan Chattopadhyay and Xin Li. Non-malleable codes and extractors for small-depth circuits, and affine functions. In *STOC*, pages 1171–1184, 2017.

[17] Eshan Chattopadhyay and Xin Li. Non-malleable extractors and codes in the interleaved split-state model and more. Electronic Colloquium on Computational Complexity, TR18-070, 2018. https://eccc.weizmann.ac.il/report/2018/070/.

[18] Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *FOCS*, pages 306–315, 2014.

[19] Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. In *Innovations in Theoretical Computer Science*, pages 155–168, 2014.

[20] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In *TCC*, pages 440–464, 2014.

[21] Sandro Coretti, Yevgeniy Dodis, Björn Tackmann, and Daniele Venturi. Non-malleable encryption: Simpler, shorter, stronger. In *TCC*, volume 2015, pages 306–335, 2016.

[22] Sandro Coretti, Ueli Maurer, Björn Tackmann, and Daniele Venturi. From single-bit to multi-bit public-key encryption via non-malleable codes. In *TCC*, pages 532–560, 2015.

[23] Dana Dachman-Soled and Mukul Kulkarni. Upper and lower bounds for continuous non-malleable codes. Cryptology ePrint Archive, Report 2018/517, 2018. https://eprint.iacr.org/2018/517.

[24] Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In *TCC*, pages 427–450, 2015.

[25] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. In *ASIACRYPT*, pages 140–160, 2013.

[26] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. *J. Cryptology*, 30(1):152–190, 2017.

[27] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. In *SCN*, pages 121–137, 2010.

[28] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[29] Yevgeniy Dodis, Allison B. Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. In *FOCS*, pages 688–697, 2011.

[30] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[31] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *CRYPTO*, pages 239–257, 2013.

[32] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *Innovations in Computer Science*, pages 434–452, 2010.

[33] Antonio Faonio and Jesper Buus Nielsen. Non-malleable codes with split-state refresh. In *PKC*, pages 279–309, 2017.

[34] Antonio Faonio and Daniele Venturi. Efficient public-key cryptography with bounded leakage and tamper resilience. In *ASIACRYPT*, pages 877–907, 2016.

[35] Sebastian Faust, Kristina Hostáková, Pratyay Mukherjee, and Daniele Venturi. Non-malleable codes for space-bounded tampering. In *CRYPTO*, pages 95–126, 2017.

[36] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In *TCC*, pages 465–488, 2014.

[37] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. A tamper and leakage resilient von Neumann architecture. In *PKC*, pages 579–603, 2015.

[38] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In *EUROCRYPT*, pages 111–128, 2014.

[39] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key derivation for poly-size tampering circuits. *IEEE Trans. Information Theory*, 62(12):7179–7194, 2016.

[40] Eiichiro Fujisaki and Keita Xagawa. Public-key cryptosystems resilient to continuous tampering and leakage of arbitrary functions. In *ASIACRYPT*, pages 908–938, 2016.

[41] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[42] Sudhakar Govindavajhala and Andrew W. Appel. Using memory errors to attack a virtual machine. In *IEEE Symposium on Security and Privacy*, pages 154–165, 2003.

[43] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, pages 415–432, 2008.

[44] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[45] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.

[46] Aggelos Kiayias, Feng-Hao Liu, and Yiannis Tselekounis. Practical non-malleable codes from l-more extractable hash functions. In *CCS*, pages 1317–1328, 2016.

[47] Aggelos Kiayias, Feng-Hao Liu, and Yiannis Tselekounis. Non-malleable codes for partial functions with manipulation detection. Cryptology ePrint Archive, Report 2018/538, 2018. `https://eprint.iacr.org/2018/538`.

[48] Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *STOC*, pages 1144–1156, 2017.

[49] Xin Li. Non-malleable extractors and non-malleable codes: Partially optimal constructions. Electronic Colloquium on Computational Complexity, TR18-028, 2018. `https://eccc.weizmann.ac.il/report/2018/028/#revision1`.

[50] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In *CRYPTO*, pages 517–532, 2012.

[51] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, volume 2009, pages 18–35, 2009.

[52] Rafail Ostrovsky, Giuseppe Persiano, Daniele Venturi, and Ivan Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions. Cryptology ePrint Archive, Report 2018/542, 2018. https://eprint.iacr.org/2018/542.

[53] Martin Otto. *Fault Attacks and Countermeasures*. PhD thesis, University of Paderborn, Germany, 2006.