Non-Malleable Codes with Split-State Refresh

Antonio Faonio and Jesper Buus Nielsen

Aarhus University

Abstract. Non-Malleable Codes for the split state model allow to encode a message into two parts such that arbitrary independent tampering on the parts either destroys completely the content or maintains the message untouched. If the code is also leakage resilient it allows limited independent leakage from the two parts. We propose a model where the two parts can be refreshed independently. We give an abstract framework for building codes for this model, instantiate the construction under the external Diffie-Hellman assumption and give applications of such split-state refreshing. An advantage of our new model is that it allows arbitrarily many tamper attacks and arbitrarily large leakage over the life-time of the systems as long as occasionally each part of the code is refreshed. Our model also tolerates that the refreshing occasionally is leaky or tampered with.

Acknowledgement. The authors acknowledge support by European Research Council Starting Grant 279447. The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation.

1 Introduction

Non-malleable codes (NMCs) are a natural relaxation of the notions of error correcting codes and error detecting codes, which tolerates more attacks by relaxing the security guarantees. An error correcting code guarantees that the encoded message is always correctly decoded. The price for this guarantee is that the code can tolerate only limited attacks, e.g., that some small constant fraction of the codeword is tampered with. An error detecting code decodes either the correct message or returns some special symbol $\perp$ signalling an error. They can tolerate more general attacks, e.g., that some larger constant fraction of the codeword is tampered with. A NMC only guarantees that either the encoded message is correctly decoded or the decoder outputs a message which is unrelated to the encoded message. This weak guarantee allows much more general tampering. It is for instance possible to tolerate tampering that modifies the entire codeword.

Despite the weaker security guarantee, NMCs can be used to protect against physical attacks. Consider a physical device D with an embedded secret key K. For instance a signature card which on input m outputs $\sigma = \mathsf{Sign}_K(m)$. Assume the device might fall into the hands of an adversary that can apply a physical attack on the device to tamper with K, producing a different but related key K'. Now, on input m the device outputs $\sigma = \mathsf{Sign}_{K'}(m)$. We would like to ensure that the adversary cannot learn any information about K from seeing $\mathsf{Sign}_{K'}(m)$. Let Encode and Decode denote the encoding and decoding algorithms of a NMC. Consider now a device $\tilde{D}$ on which we store an encoded key $X \leftarrow \mathsf{Encode}(K)$. On input m the device outputs $\sigma = \mathsf{Sign}_{\mathsf{Decode}(X)}(m)$. We call $\tilde{D}$ the strengthened device. In face of a tampering with the key the strengthened device outputs $\sigma = \mathsf{Sign}_{\mathsf{Decode}(X')}(m)$. The value of $\mathsf{Decode}(X')$ will either be K or an unrelated key K'. NMC security guarantees that when K' is an unrelated key, then the adversary could in fact have computed K' itself without any access to K. It follows that the adversary either learns a correct signature $\sigma = \mathsf{Sign}_K(m)$ or a value $\sigma = \mathsf{Sign}_{K'}(m)$ it could have computed itself without access to the device. This ensures that tampering does not result in information leaking from the device.

Formally security is defined as a tampering game between an adversary and a simulator S. The adversary submits to the tampering game a message m and the game computes a random encoding $X \leftarrow \mathsf{Encode}(m)$. The adversary then submits a tampering function T. Now the game either computes $m' = \mathsf{Decode}(T(X))$ and gives m' to the adversary. Or, it computes $m' = \mathsf{S}(T)$ and gives m' to the adversary. The code is called secure if for all adversaries there exists an efficient simulator such that the adversary cannot guess which of the two cases occurred except with negligible advantage. A small but crucial modification of the game is needed. Notice that the adversary might for instance submit T equal to the identity function. In that case $m' = m$ in the first case, so the simulator would be required to compute m too, which is impossible as it is not given m as input and m might be a random value. The game is therefore modified to allow S to give a special output $*$ in which case the game sets $m' = m$ before giving m' to the adversary. Security therefore demonstrates that the adversary when submitting a tampering function T could itself efficiently have computed whether the tampering will have no effect (when $\mathsf{S}(T) = *$) and in case there is an effect, which message $m' = \mathsf{S}(T)$ would be the result of the tampering.

It is clear that we need to put some restriction on the tampering function. If the adversary submits the function $T(X) = \mathsf{Encode}(\mathsf{Decode}(X) + 1)$ the simulator would have to output $m+1$ without knowing m. The most popular way to restrict the tampering functions is to assume the split-state model (STM), which was first used to get leakage-resilient cryptography (see Dziembowski *et al.* [21]). In this model we assume that the encoding X consists of two parts $X = (X^0, X^1)$ stored on two separate storage devices or seperate parts of a chip. The assumption is that the adversary can only tamper independently with the two parts, i.e., in the model it submits tampering functions $T = (T^0, T^1)$ and the result of tampering is $(X'^0, X'^1) = (T^0(X^0), T^1(X^1))$. This is also the model we consider in this paper. In the split state model it is possible to construct codes which tolerates arbitrary tampering, except that the two parts must be tampered independently.

Unfortunately NMC security is not sufficient for device strengthening if the adversary can repeatedly tamper with the device. To see this assume for simplicity that the encoding has the property that if a single bit is flipped in an encoding X, then $\mathsf{Decode}(X') = \perp$. Consider then the tampering function O_i which overwrites the i'th bit in X by 0. Each O_i is allowed in the split-state model. Now, $\mathsf{Decode}(O_i(X)) = \perp$ if and only if the i'th bit of X is 1. Hence by applying $O_1, \ldots, O_{|X|}$ an adversary can learn X and then compute $m = \mathsf{Decode}(X)$. This means that if the code is secure then by definition the simulator can also compute m, which it cannot. Let us call the above attack the fail-or-not attack.

Two different ways to circumvent the fail-or-not attack has been proposed in the literature. In [34] Liu and Lysyanskaya propose that the strengthened device whenever it reconstructed the key $K = \mathsf{Decode}(X)$ resamples a new encoding $X' \leftarrow \mathsf{Encode}(K)$ and overrides X by X' on the storage medium. This way the adversary gets to tamper with each fresh encoding only once and the NMC assumption is sufficient. In [25] Faust *et al.* propose a model where the encoding X remains the same in all tamperings. Instead the authors assume that the strengthened device self destructs when it detects that $\mathsf{Decode}(X) = \perp$. In the fail-or-not attack the adversary is using failure or not to leak information on the encoding X. If the device self destructs on failure this can however only be exploited to leak logarithmic many bits, namely in which round of tampering the self destruction happened. The authors in [25] then use a code which can tolerate limited leakage on the two halves of the encoding and constructs the code such that computing in which round the device would have self-destructed can be done using only limited independent leakage from the two halves, reducing tampering to leakage, an idea we use in our new code too.

Both [34] and [25] consider codes which are additionally leakage resilient in the split state model. In [25] this is needed anyway to protect against tampering and in [34] it is argued to be a natural requirement as we assume the device to be in the hands of an adversary which might learn leakage on the two parts X^0 and X^1 by measuring the device during operation. In both [34] and [25] it is assumed that the circuitry doing the encoding (and refreshing) cannot be tampered with and that it is leakage free, i.e., only the storage devices are subject to tampering and leakage. Below we will partially relax this assumption by allowing occasional leakage and tampering of the refresh procedure.

Our Contributions We propose a new model in line with [34]. In particular we do not assume the device can self destruct and we use refreshing to protect against the fail-or-not attack. We propose two extra requirements on the refreshing which we motivate below. First, we want the refreshing to be split-state, i.e., the refreshing algorithm should be of the form $\mathsf{Refresh}(X) = (\mathsf{Refresh}_0(X^0), \mathsf{Refresh}_1(X^1))$. Second, the code should tolerate multiple tampering attacks in between refreshes.

To motivate the model, imagine the following application for strengthening a device. The parts X^0 and X^1 are placed in separate storages. When the key is needed the device computes $K = \mathsf{Decode}(X)$ and outputs $\mathsf{Sign}_K(m)$. In addition to this, occasionally the device will read up a part X^i and write back $X'^i = \mathsf{Refresh}(X^i)$. The refreshing of the parts might also be done by separate processes sitting in the storage device of the part, as opposed to the circuitry doing the decoding. The practical motivation is as follows. In all existing codes the encoding process is considerably more complex than the decoding process. For instance encoding necessarily needs cryptographic strength randomness, whereas decoding can be deterministic. It could therefore be much harder to create a leakage and tamper free implementation of Encode. Also, refreshing by decoding and re-encoding is unnecessarily risky as (real-world) leakage from this process could be leakage on the decoded key K.

Notice on the other hand that if a partial refreshing $X'^i = \mathsf{Refresh}(X^i)$ is tampered with, then it can simply be considered just another tampering attack on X^i in the split state model. In the same way, if a partial refreshing $X'_i = \mathsf{Refresh}(X_i)$ is leaky, then it can simply be considered just another leakage attack on X_i in the split state model. For this to be true it is important that the refreshing is split state, motivating our first extra requirement. As a consequence, if only occasionally the refreshing succeeds in being tamper and leakage free, all the failed attempts can be recast as tamper and leakage attacks. This means the code remains secure if it can tolerate several tamper and leakage attacks in between refreshes, motivating our second extra requirement. Notice that for this to be true, the security of the code should not depend on the two parts being refreshed at the same time. We can only assume that each part occasionally gets refreshed.

Our model works as follows. The adversary submits to the game a message m and the game samples $(X^0, X^1) \leftarrow \mathsf{Encode}(m)$. The adversary can then repeatedly submit leakage or tamper queries. In a leakage query the adversary submits (i, L) and is given $R = L(X^i)$. In a tampering query the adversary submits (T^0, T^1) and is given $m' = \mathsf{Decode}(T^0(X^0), T^1(X^1))$.[1] The adversary can also make a refresh query by submitting an index j to the game. Then the game refreshes the corresponding part: $E^j \leftarrow \mathsf{Refresh}_j(E^j)$. We give a simulation-based security definition. The simulator is not given m. To simulate a leakage query the simulator is given (j, L) and must return some value R to the adversary. To simulate a tampering query the simulator is given (T^0, T^1) and must return some value m', where $m' = *$ is replaced with $m' = m$ before m' is returned to the adversary. To simulate a refresh query the simulator is given j and has to return nothing. The adversary must not be able to tell whether it is interacting with the real world or the simulator. The only restriction on the adversary is that the length of the leakage and the number of tampering attacks in between refreshes must be limited. For any polynomials $p(\kappa), q(\kappa)$ we construct a code that can tolerate $p(\kappa)$ bits of leakage and $q(\kappa)$ many tampering attacks in between successful refreshes.

Our definition is *not strong* according to the notions of Non-Malleable Codes given in the original paper [20]. In the security experiment of the strong NMCs the adversary receives either the entire tampered codeword (as opposed to receive the decoded message of the tampered codeword) or $*$ in case that the tampering function keeps the codeword unaltered. The goal of the adversary

[1] Notice that tampering does not overwrite the codeword. This is called non-persistent tampering and is stronger than persistent tampering in the split state model as the set of tampering functions is closed under composition—subsequent tamperings can just first reapply all previous tampering functions (cf. Jafargholi and Wichs [32]).

is to distinguish the codewords of two different message given the result of the tampering function. However, such definition cannot be met in presence of a split-state refresh algorithm. In fact the adversary could forward, as tampering function, the refreshing function itself and receives a valid codeword (since it won't be the same codeword). Given the codeword, it can easily distinguish by decoding.

Our techniques borrow ideas from both [34] and [25]. In X^1 we will keep a secret key sk for a public-key encryption scheme. In X^0 we will keep the corresponding public key $pk = \mathsf{PK}(sk)$, an encryption $c = \mathsf{Enc}(pk, m)$ of the encoded message and a simulation-sound NIZK proof of knowledge π of some sk such that $pk = \mathsf{PK}(sk)$ using c as a label. Decoding will check the proof and if it is correct and sk matches pk. If so, it outputs $\mathsf{Dec}(sk', c)$. To tolerate leakage and to allow refreshing we use a leakage resilient encryption scheme which allows to refresh sk and c independently. The public key pk in X^1 will never be refreshed, which is secure as pk might in fact be public. To allow the proof of knowledge to be refreshed we use a non-malleable proof with some controlled malleability. We give a concrete instantiation of this framework based on the Continual Leakage-Resilient scheme of Dodis *et al.* [18] and the Controlled-Malleable NIZK system of Chase *et al.* [10] instantiated with Groth-Sahai proofs [30].

The structure of the encoding scheme is very similar to the one proposed by [34], however there are few substantial differences: 1. We substitute the PKE and the NIZK scheme with cryptographic primitives that allow efficient refresh mechanisms; 2. The NP relationship of the NIZK is different. (In fact, it is inspired by the scheme of [25].)

The main proof technique is to reduce tampering to legal leakage queries on the encryption scheme. In the reduction we are given separate leakage oracles of sk and c. To simulate leakage from X^1, leak from sk. To simulate leakage from X^0, once and for all produce a simulated proof π with label c and simulate each leakage query from X^0 by leaking from (pk, c, π). As for tampering queries, assume that the parts have been tampered into $X'^1 = sk'$ and $X'^0 = (pk', c', \pi')$. First we use leakage to check whether the decoding would fail. Leak from X'^1 the value $pk'' = \mathsf{PK}(sk')$. Then leak from X'^0 a single bit telling whether $pk' = pk''$ and whether π' is a valid proof. This is exactly enough to determine whether the decoding would fail or not. If the decoding would fail, output $\perp$. Otherwise, if the proof π' still has c as label (which implies that $X'^0 = (pk'', c, \pi')$ when the proof is valid), then output $*$ indicating that the decoding would output the original encoded message. If the label of π' is not c, then use the extraction trapdoor of the proof to extract the secret key sk' matching pk''. Then output $\mathsf{Dec}(sk', c')$. This allows to simulate each tampering attack with limited leakage on X^0 and X^1. Therefore the scheme remains secure as long as refreshing happens often enough for the leakage needed to simulate tampering to not grow about the leakage tolerance of the encryption scheme.

In [18], it was shown that Continually Leakage-Resilient Codes with Split-State Refresh are impossible to construct without computational assumptions. The result holds even when the leakage between each updates is 1 bit. It is easy to see that the same result holds for Non-Malleable Codes with Split-State Refresh. (This is because a tampering attack corresponds at least to 1 bit of leakage.)

More Related Work. Non-Malleable Codes were introduced to achieve tamper-proof security of arbitrary cryptographic primitives. Since their introduction many works have constructed NMCs in different models both under cryptographic assumptions or information theoretically (see [19,1,2,12,26,15,32,3,38]).

A related line of work on tamper resilience (see [31,27,33,14]) aims at constructing secure compilers protecting against tampering attacks targeting the computation carried out by a cryptographic device (typically in the form of boolean and arithmetic circuits).

A third line of work on tamper resilience instead aims at constructing ad hoc solutions for different contexts like for example symmetric encryption [6,36,28], public-key encryption [5,39,7,35,24,16,17], hash functions [29] and more [8,38,13].

Roadmap. In the following we will first introduce some known notation and abstract definitions of the properties we need from the primitives in the abstract framework. Then we describe and prove the abstract framework, followed by an instantiation based on External Diffie-Hellman assumption [4,9]. At the end we will present the application to continual-tamper-and-leakage resilient cryptography in more details.

2 Preliminaries

2.1 Notation and Probability Preliminaries

We let $\mathbb{N}$ denote the naturals and $\mathbb{R}$ denote the reals. For $a, b \in \mathbb{R}$, we let $[a, b] = \{x \in \mathbb{R} \ : \ a \leq x \leq b\}$; 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 denote with $x_{(i)}$ 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 randomness r. An algorithm A is *probabilistic polynomial-time* (PPT) if A is allowed to use random choices and the computation of $\mathsf{A}(x; r)$ terminates in at most $poly(|x|)$ steps for any input $x \in \{0, 1\}^*$ and randomness $r \in \{0, 1\}^*$.

Let κ be a security parameter. A function *negl* is called *negligible* in κ (or simply negligible) if it vanishes faster than the inverse of any polynomial in κ. For a relation $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^*$, the language associated with $\mathcal{R}$ is $L_{\mathcal{R}} = \{x : \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$.

For two ensembles $\mathcal{X} = \{X_\kappa\}_{\kappa \in \mathbb{N}}$, $\mathcal{Y} = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$, we write $\mathcal{X} \overset{c}{\approx}_\epsilon \mathcal{Y}$, meaning that every probabilistic polynomial-time distinguisher D has $\epsilon(\kappa)$ advantage in distinguishing $\mathcal{X}$ and $\mathcal{Y}$, i.e., $\frac{1}{2}|\mathbb{P}[D(\mathcal{X}_\kappa) = 1] - \mathbb{P}[D(\mathcal{Y}_\kappa) = 1]| \leq \epsilon(\kappa)$ for all sufficiently large values of κ.

We simply write $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$ when there exists a negligible function ϵ such that $\mathcal{X} \overset{c}{\approx}_\epsilon \mathcal{Y}$. Similarly, we write $\mathcal{X} \approx_\epsilon \mathcal{Y}$ (statistical indistinguishability), meaning that every unbounded distinguisher has $\epsilon(\kappa)$ advantage in distinguishing $\mathcal{X}$ and $\mathcal{Y}$.

Given a string $X = (X^1, X^2) \in (\{0,1\}^*)^2$ and a value $\ell \in \mathbb{N}$ let $\mathcal{O}_\ell(X)$ be *the split-state leakage oracle.* $\mathcal{O}_\ell(X)$ accepts as input tuple of the form (i, f) where the first element i is an index in $\{0,1\}$ and the the second element f is a function defined as a circuit. If the total amount of leakage is below ℓ, $\mathcal{O}_\ell(X)$ outputs $f_1(X^{i_1})$ otherwise it outputs the special symbol $\perp$. More formally, the oracle $\mathcal{O}_\ell(X)$ is a state machine that maintains state variables $\mathcal{O}_\ell(X).l^0$ and $\mathcal{O}_\ell(X).l_1$ and upon input (i, f) where f is an efficiently computable function with co-domain $\{0,1\}^o$ for a value $o \in \mathbb{N}$ outputs $f(X^i)$ if $(l^i + o) \leq \ell$ and then updates the value l^i to $l^i + o$, otherwise it outputs the value $\perp$.

Given two PPT interactive algorithms A and B we write $(y, k) \leftarrow \mathsf{A}(x) \leftrightarrows \mathsf{B}(z)$ to denote the joint execution of the algorithm A with input x and the algorithm B with input z. The string y (resp. z) is the output of A (resp. B) after the interaction. In particular we write $\mathsf{A} \leftrightarrows \mathcal{O}_\ell(X)$ to denote A having oracle access to the leakage oracle with input X. Moreover, we write $\mathsf{A} \leftrightarrows \mathcal{B}, \mathcal{C}$ to denote A interacting in an interleaved fashion both with $\mathcal{B}$ and with $\mathcal{C}$.

2.2 Cryptographic Primitives

NIZK Proof of Knowledge. We first introduce the necessary notation for *label-malleable* NIZK (lM-NIZK for short) argument system. A label-malleable NIZK is intuitively a non-malleable NIZK except that from a proof under a given label one can generate a new proof for the same statement under a different label without using the witness. A lM-NIZK $\mathcal{NIZK} := (\mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{RandProof}, \mathsf{LEval})$ with label space $\mathcal{L}$ is a tuple of PPT algorithms where: (1) The algorithm I upon input the security parameter 1^κ, creates a common reference string (CRS) ω; (2) The prover algorithm P upon input ω, a label $L \in \mathcal{L}$ and a valid instance x together with a witness w produces a proof π. We write $\mathsf{P}^L(\omega, x, w)$; (3) The verifier algorithm V upon input ω, a label L an instance x together with a proof π outputs a verdict in $\{0,1\}$. We write $\mathsf{V}^L(\omega, x, \pi)$; (4) The label-derivation algorithm LEval upon input ω, a transformation ϕ, a label L an instance x and a proof π outputs a new proof π'.

Definition 1 (Adaptive multi-theorem zero-knowledge). *Let $\mathcal{NIZK}$ be a non-interactive argument system for a relation $\mathcal{R}$. We say that $\mathcal{NIZK}$ 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 trapdoor τ_{sim}.
(ii) There exist a PPT simulator S_1 and a negligible function ν such that, for all PPT adversaries A, we have that

$$\Big|\mathbb{P}\left[\mathsf{A}(\omega) \leftrightarrows \mathsf{P}(\omega, \cdot) = 1 \middle|\ \omega \leftarrow \mathsf{I}(1^\kappa)\right] - \mathbb{P}\left[\mathsf{A}(\omega) \leftrightarrows \mathcal{SIM}(\tau_{sim}, \cdot) = 1 \middle|\ (\omega, \tau_{sim}) \leftarrow \mathsf{S}_0(1^\kappa)\right]\Big| \leq \nu(\kappa).$$

The simulation oracle $\mathcal{SIM}(\tau_{sim},\cdot)$ takes as input a tuple (L,x,w) and checks if $(x,w) \in \mathcal{R}$, and, if true, ignores w and outputs a simulated argument $\mathsf{S}_1(\tau_{sim},L,x)$, and otherwise outputs $\bot$.

```
Experiment $\mathbf{Exp}^{\mathcal{T}-\mathsf{lmSE}}_{\mathsf{Ext,S,A}}(\kappa)$:
  $(\omega,\tau_{sim},\tau_{ext}) \leftarrow \mathsf{S}_0(1^\kappa)$;$\mathcal{Q} \leftarrow \emptyset$;
  $(x^*,L^*,\pi^*) \leftarrow \mathsf{A}(\omega) \leftrightarrows \mathcal{SIM}^*(\tau_{sim})$;
  $(w,\phi,L) \leftarrow \mathsf{Ext}_1(\tau_{ext},L^*,x^*,\pi)$;
  Return $(w,\phi,L),\mathcal{Q}$.

Oracle $\mathcal{SIM}^*(\tau_{sim},\mathcal{Q})$:
  Upon message $(L,x) \in \mathcal{L} \times L_{\mathcal{R}}$;
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(L,x)\}$;
  Return $\mathsf{S}_1(\tau_{sim},L,x)$.
```

```
Experiment $\mathbf{Exp}^{\mathcal{T}-\mathsf{lDP}}_{\mathcal{NIZK}}(\kappa)$:
  $\omega \leftarrow \mathsf{I}(1^\kappa)$;$b \leftarrow_{\$} \{0,1\}$;
  $(\phi,L,x,w,\pi),st \leftarrow \mathsf{A}(\omega)$;
  if $(x,w) \notin \mathcal{R} \vee \big(\mathsf{V}^L(\omega,x,\pi)=0\big)$
      then Return 0;
  if $(b=0)$ then $\pi' \leftarrow \mathsf{P}(\omega,\phi(L),x,w)$;
  else $\pi' \leftarrow \mathsf{LEval}(\omega,\phi,(x,L,\pi))$;
  $b' \leftarrow \mathsf{A}(st,\pi')$;
  Return $(b=b')$.
```

Fig. 1: Experiments defining $\mathcal{T}$-ml-SE and label derivation privacy of $\mathcal{NIZK}$.

Given $\mathcal{NIZK}$ that supports the set of labels $\mathcal{L}$, we say that a set $\mathcal{T}$ is a set of label transformations for $\mathcal{NIZK}$ iff for any $\phi \in \mathcal{T}$ the co-domain of ϕ is a subset of $\mathcal{L}$.

Definition 2 ($\mathcal{T}$-Malleable Label Simulation Extractability). *Let $\mathcal{T}$ be a set of label transformations for $\mathcal{NIZK}$. Let $\mathcal{NIZK}$ be a non-interactive argument system for a relation $\mathcal{R}$. We say that $\mathcal{NIZK}$ is $\mathcal{T}$-*malleable label simulation extractable *($\mathcal{T}$-ml-SE) if the following holds:*

(i) There exists an 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 Ext such that, for all PPT adversaries A, the probability, taken over the experiment $\mathbf{Exp}^{\mathcal{T}-\mathsf{lmSE}}_{\mathsf{Ext,S,A}}$ (as defined in Fig. 1), of the conjunction of the following events is negligible in the security parameter κ:

(a) $(L^*,x^*) \notin \mathcal{Q}$ and $\mathsf{V}(\omega,L^*,x^*,\pi^*)=1$;

(b) $(x^*,w) \notin \mathcal{R}$;

(c) Either $\phi \notin \mathcal{T}$ or for any (L,x) either $(L,x) \notin \mathcal{Q}$ or $\phi(L) \neq L^*$.

Moreover, we say that A wins the $\mathcal{T}$-lm SE Experiment when all the above events happen.

Definition 3 (Label Derivation Privacy). *Let $\mathcal{NIZK}$ a lM-NIZK, and let $\mathcal{T}$ be a set of label transformations. We say that $\mathcal{NIZK}$ has* label derivation privacy *if for all PPT A, there exists a negligible function negl such that* $(\mathsf{P}\left[\mathbf{Exp}^{\mathcal{T}-\mathsf{lDP}}_{\mathcal{NIZK}}(\kappa)=1\right]-\frac{1}{2}) \leq negl(\kappa)$ *(where the experiment is defined in Fig. 1).*

Public-Key Encryption. A public-key encryption (PKE) scheme is a tuple of algorithms $E = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows. (1) Algorithm Setup takes as input the security parameter and outputs public parameters $pub \in \{0,1\}^*$. all algorithms are implicitly given pub as input. (2) Algorithm Gen takes as input the security parameter 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 PPT function PK which upon an input $sk \in \mathcal{SK}$ produces a valid public key pk. (3) The randomized algorithm Enc takes as input the public key pk, a message $m \in \mathcal{M}$, and randomness $r \in \mathcal{R}$, 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 $\perp$. Additionally, we also consider two PPT algorithms: 1. Algorithm $\mathsf{UpdateC}$ takes as input a public key pk a ciphertext c and outputs a new ciphertext c. 2. Algorithm $\mathsf{UpdateS}$ takes as input a secret key sk and outputs a new secret key sk'.

Correctness (with Updates). We say that E satisfies *correctness* if for all $pub \leftarrow \mathsf{Setup}(1^\kappa)$ and $(pk, sk) \leftarrow \mathsf{Gen}(pub)$ we have that:

$$\mathbb{P}\left[\mathsf{Dec}(\mathsf{UpdateS}(sk), \mathsf{UpdateC}(pk, \mathsf{Enc}(pk, m))) = m\right] = 1,$$

where the randomness is taken over the internal coin tosses of algorithms Enc, $\mathsf{UpdateS}$ and $\mathsf{UpdateC}$. Additionally, we require that for any $pk, sk \leftarrow \mathsf{Gen}(pub)$: (A) any sk' such that $\mathsf{PK}(sk') = pk$ and any $c \in \mathcal{C}$ we have that $\mathsf{Dec}(sk, c) = \mathsf{Dec}(sk', c)$; (B) any $sk' \leftarrow \mathsf{UpdateS}(sk)$ we have that $\mathsf{PK}(sk) = \mathsf{PK}(sk')$.

CLRS Friendly PKE Security. We now turn to define Continual-Leakage Resilient Storage Friendly public key encryption.

```
Experiment $\mathbf{Exp}^{\mathrm{clrs}}_{E,\mathsf{A}}(\kappa, \ell)$:
$pub \leftarrow \mathsf{Setup}(1^\kappa)$
$(pk, sk) \leftarrow \mathsf{Gen}(pub)$
$b \leftarrow\!\!\$\ \{0,1\}$; $j \leftarrow 1$
$l_0 := 0$; $l_1 := 0$
$(m_0, m_1) \leftarrow \mathsf{A}(pub, pk)$
if $|m_0| \neq |m_1|$ set $m_0 \leftarrow m_1$
$c \leftarrow \mathsf{Enc}(pk, m_b)$
$\mathsf{state}^0 := sk$,$\mathsf{state}^1 := c$;
$st \leftarrow \mathsf{A}(pk) \leftrightarrows \mathsf{Update}, \mathcal{O}_\ell(\mathsf{state})$
$b' \leftarrow \mathsf{A}(pk, c)$
Return $(b' = b)$
```

```
Oracle $\mathsf{Update}(i)$:
$\mathcal{O}_\ell(\mathsf{state}).l^i := 0$
$r' \leftarrow\!\!\$\ \{0,1\}^{p(\kappa)}$
if $(i = 0)$
    $c = \mathsf{state}^0$
    $c' \leftarrow \mathsf{UpdateC}(pk, c)$
    $\mathsf{state}^0 := c'$
if $(i = 1)$
    $sk = \mathsf{state}^1$
    $sk' \leftarrow \mathsf{UpdateS}(sk)$
    $\mathsf{state}^1 := sk'$
```

Fig. 2: Experiment defining CLRS security of E.

Definition 4. *For $\kappa \in \mathbb{N}$, let $\ell = \ell(\kappa)$ be the leakage parameter. We say that $E = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdateC}, \mathsf{UpdateS})$ is ℓ-CLRS Friendly if for all PPT adversaries A there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $\left|\mathbb{P}\left[\mathbf{Exp}^{\mathrm{clrs}}_{E,\mathsf{A}}(\kappa, \ell) = 1\right] - \frac{1}{2}\right| \leq \nu(\kappa)$, (where the experiment is defined in Figure 2).*

We observe that Definition 4 is weaker than the definition of Dodis *et al.* [18], in fact we do not consider leakage from the update process. We introduce an extra property on the $\mathsf{UpdateC}$ algorithm of a CLRS Friendly PKE.

Definition 5. *We say that E is* perfectly ciphertext-update private *if for any $\kappa \in \mathbb{N}$, $pub \leftarrow \mathsf{Setup}(1^\kappa)$, $(pk, sk) \leftarrow \mathsf{Gen}(pub)$ and any $m \in \mathcal{M}$ the distributions $\{\mathsf{Encode}(pk, m)\}$ and $\{\mathsf{UpdateC}(pk, \mathsf{Enc}(pk, m))\}$ are equivalent.*

If E is a CLRS Friendly PKE then the weaker version of the property above where the two distributions are computationally indistinguishable already holds. The construction in Section 4 can be proved secure using the computational ciphertext-update privacy property. However, we prefer to include the perfectly ciphertext-update privacy property because it simplifies the exposition.

3 Definition

In this section we consider three definitions of Non-Malleable Codes with Refresh (NMC-R). The syntax given allows the scheme to depends on a common reference string following [34].

A coding scheme in the CRS model is a tuple $\Sigma = (\mathsf{Init}, \mathsf{Encode}, \mathsf{Decode})$ of PPT algorithms with the following syntax: (1) Init on input 1^κ outputs a common reference string crs. (2) Encode on inputs crs and a message $m \in \mathcal{M}_\kappa$ outputs $X \in \mathcal{C}_\kappa$; (3) Decode is a deterministic algorithm that on inputs crs and a codeword $X \in \mathcal{C}_\kappa$ decodes to $m' \in \mathcal{M}_\kappa$. A coding scheme is correct if for any κ and any $m \in \mathcal{M}_\kappa$ we have $\mathbb{P}_{crs,r_e}[\mathsf{Decode}(crs, \mathsf{Encode}(crs, m; r_e)) = m] = 1$.

We consider coding schemes with an efficient refreshing algorithm. Specifically, for a coding scheme Σ there exists an algorithm Rfrsh that upon inputs crs and a codeword $X \in \mathcal{C}_\kappa$ outputs a codeword $X \in \mathcal{C}_\kappa$. For correctness we require that $\mathbb{P}[\mathsf{Decode}(crs, \mathsf{Rfrsh}(crs, X)) = \mathsf{Decode}(crs, X)] = 1$, where the probability is over the randomness used by the algorithms and the generation of the CRS.

We are interested in coding schemes in the split-state model where the two parts can be refreshed independently and without the need of any interactions. Given a codeword $X := (X^0, X^1)$, we consider the procedure $\mathsf{Rfrsh}(crs, (i, X^i))$ for $i \in \{0, 1\}$ that takes the i-th piece of the codeword and outputs a new piece X'^i. Abusing of notation, given a codeword $X := (X^0, X^1)$ when we write $\mathsf{Rfrsh}(crs, X)$ we implicitly mean the execution of both $\mathsf{Rfrsh}(crs, (i, X^0))$ and $\mathsf{Rfrsh}(crs, (i, X^1))$. Similarly, given a split-state function $T = (T^0, T^1)$ we equivalently write $T(X)$ meaning the application of both $T^0(X^0)$ and $T^1(X^1)$.

We require that for any codeword $X := (X^0, X^1)$ and for any $i \in \{0, 1\}$, let $\bar{X}$ such that $\bar{X}^i \leftarrow \mathsf{Rfrsh}(crs, (i, X^i))$ and $\bar{X}^{i-1} = X^i$ then $\mathbb{P}[\mathsf{Decode}(crs, \bar{X}) = \mathsf{Decode}(crs, X)] = 1$.

NMC with Refresh in the STM. We now give the security definition for Non-Malleable Codes with Refresh. Although the definition would be meaningful for a more general setting, for the sake of concreteness, we specialize it for the split-state model. Let I_m be a function from $\mathcal{M} \cup \{*\}$ to $\mathcal{M}$ which substitutes the symbol $*$ in input with the message m and acts as the identity function otherwise. Let **Tamper** and **SimTamper** be the experiments described in Figure 3.

```
Experiment $\mathbf{Tamper}_{\mathsf{A},\Sigma}(\kappa,\ell,\rho,\tau)$:
Variables $i, t^0, t^1$ set to 0;
$crs \leftarrow \mathsf{Init}(1^\kappa)$;
$(m,z) \leftarrow \mathsf{A}_0(crs)$; $(m_0, st_0) := (m,z)$;
$X_0 := (X_0^0, X_0^1) \leftarrow \mathsf{Encode}(crs, m_0)$;
forall $i < \rho(\kappa)$:
  $(T_{i+1}, st_{i+1}, j) \leftarrow \mathsf{A}_1(m_i, st_i) \leftrightarrows \mathcal{O}^\ell(X_i)$,
  where $T_{i+1}$ is a split-state function;
  $\tilde{X}_{i+1} := T_{i+1}(X_i)$;
  $m_{i+1} := \mathsf{Decode}(crs, \tilde{X}_{i+1})$;
  $X_{i+1} := X_i$;
  For $j' \in \{0,1\}$ if $(t^{j'} > \tau)$ or $(j' = j)$
    $X_{i+1}^{j'} \leftarrow \mathsf{Rfrsh}(crs, (j', X_i^{j'}))$
    Set $\mathcal{O}_\ell(X_i).l^{j'}$ and $t^{j'}$ to 0;
  Increment $t^0, t^1$ and $i$;
Return $\mathsf{A}_2(st_p)$.
```

```
Experiment $\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa,\ell,\rho,\tau)$:
Variable $i$ set to 0;
$(crs, aux) \leftarrow \mathsf{S}_0(1^\kappa)$;
$(m,z) \leftarrow \mathsf{A}(crs)$;$(m_0, st_0) := (m,z)$;
forall $i < \rho(\kappa)$:
  $(T_{i+1}, st_{i+1}, j) \leftarrow \mathsf{A}_1(m_i, st_i) \leftrightarrows \mathsf{S}_2(z)$;
  $\bar{m}_{i+1} \leftarrow \mathsf{S}_1(T_{i+1}, z)$;
  $m_{i+1} := I_{m_0}(\bar{m}_{i+1})$;
  $\mathsf{S}_3(j, z)$
  $i := i + 1$;
Return $\mathsf{A}_2(st_p)$.
```

Fig. 3: Experiments defining the security of NMC with Refresh Σ. Notice that t^j for $j \in \{0,1\}$ counts the number of rounds since the last refresh of X^j. If $t^j > \tau$ or if the adversary triggers it then a refresh of X^j is executed.

Definition 6 (Non-Malleable Codes with Refresh). *For $\kappa \in \mathbb{N}$, let $\ell = \ell(\kappa), \rho = \rho(\kappa), \tau = \tau(\kappa)$ be parameters. We say that the coding scheme Σ is a (ℓ, ρ, τ)-Non-Malleable Code with Refresh (NMC-R) in the split state model if for any adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$ where A_0 and A_2 are a PPT algorithm and A_1 is deterministic polynomial time, there exists a PPT simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3)$ and a negligible function ν such that*

$$\left|\mathbb{P}\left[\mathbf{Tamper}_{\mathsf{A},\Sigma}(\kappa,\ell,\rho,\tau) = 1\right] - \mathbb{P}\left[\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}(\kappa,\ell,\rho,\tau) = 1\right]\right| \leq \nu(\kappa).$$

We give some remarks regarding the definition above. The simulator S is composed of four different parts $\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3$. The algorithm S_0 upon input 1^κ produces a CRS toghether with some trapdoor information *aux*, the CRS produced and the output of Init are computational indistinguishable.

For simplicity, we assume that the state information *aux* is stored in a common read-and-write memory that the simulators $\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3$ have access to. We will sometime referee to S_1 as the *tampering simulator*, to S_2 as the *leakage simulator* and to S_3 as the *refresh simulator*.

The adversary A is composed by a PPT algorithm A_0, a deterministic algorithm A_1 and PPT distinguishing algorithm A_2. The adversary A_0 can sample a message m (as function of the CRS) and some state information z. The latter may encode some side information z about the message m and other information that A_0 wants to pass to A_1. Notice that we can assume without loss of any generality A_1 to be deterministic, in fact, z may also contain random coins. The tampering simulator, the leakage simulator and the refresh simulator take as input the state information z. In addition, in each round, the tampering simulator S_1 receives a split-state tampering function T_{i+1} and it outputs a message $\bar{m}_{i+1}$. First, we notice that, in general, the tampering T_{i+1} can be produced as a function of the initial message m, therefore the simulator (which does not know m) cannot compute the tampering function by its own, even given z. Secondly, the adversary can efficiently produce a tampering function that keeps the same encoded message but modifies the codeword (for example, by submitting the refreshing algorithm Rfrsh as tampering function). The task of the tampering simulator is to detect this (outputting the special symbol $*$), in this case the function I_m forwards to A the initial message m. (We stress that the simulator does not know the message m, so it cannot forward m directly to A but it needs to pass by I_m.)

The tamper experiment takes four parameters: the security parameter κ, the leakage parameter ℓ, the round parameter ρ and the tampering parameter τ. The tampering parameter τ counts how many times the adversary can tamper with the codeword before a refresh of the codeword is needed.

4 Construction

Let $E = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdateC}, \mathsf{UpdateS})$ be a CLRS friendly PKE with ciphertext space $\mathcal{C}_E$. Let $\mathcal{R}$ be the NP relation defined below:

$$\mathcal{R} := \{(pk, sk) \ : \ pk = \mathsf{PK}(sk), sk \in \mathcal{SK}\}\,.$$

Let $\mathcal{T}$ be a set of label transformations defined below:

$$\mathcal{T} := \{\phi \ : \ \exists pk, sk : \forall m, r : \mathsf{Dec}(sk, \phi(\mathsf{Enc}(pk, m; r)) = m, pk = \mathsf{PK}(sk)\}\,.$$

Notice that both $\mathcal{R}$ and $\mathcal{T}$ are implicitly parametrized by the public parameters *pub* of the PKE scheme. Let $\mathcal{U}$ be the following set of label transformations:

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

It is easy to check that $\mathcal{U} \subseteq \mathcal{T}$. In fact, by the correctness of PKE, there exists sk such that $\mathbb{P}\left[\mathsf{Dec}(sk, \mathsf{UpdateC}(pk, \mathsf{Enc}(pk, m)) = m\right] = 1$ and $pk = \mathsf{PK}(sk)$.

Let $\mathcal{NIZK} := (\mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{LEval})$ be a lM-NIZK argument system for the relation $\mathcal{R}$ with label space $\mathcal{C}_E$ and set of transformation $\mathcal{T}$. Let Σ be the following coding scheme with refresh in the CRS model:

- $\underline{\mathsf{Init}(1^\kappa)}$: Sample $\omega \leftarrow \mathsf{I}(1^\kappa)$ and $pub \leftarrow \mathsf{Setup}(1^\kappa)$. Return $crs = (\omega, pub)$.
- $\underline{\mathsf{Encode}(crs, m)}$: Parse $crs = (\omega, pub)$, sample $(sk, pk) \leftarrow \mathsf{Gen}(pub)$, compute $c \leftarrow \mathsf{Enc}(pk, m)$ and $\pi \leftarrow \mathsf{P}^c(\omega, pk, sk)$. Set $X^0 := (pk, c, \pi)$ and $X^1 := sk$ and return $X := (X^0, X^1)$.
- $\underline{\mathsf{Decode}(crs, X)}$: Parse $crs = (\omega, pub)$ and $X = (X^0, X^1)$ where $X^1 = sk$ and $X^0 = (pk, c, \pi)$. Check: (A) $pk = \mathsf{PK}(sk)$ and (B) $\mathsf{V}^c(\omega, pk, \pi) = 1$.
  If both checks (A) and (B) hold then return $\mathsf{Dec}(sk, c)$, otherwise return $\perp$.
- $\underline{\mathsf{Rfrsh}(crs, (j, X^j))}$:
  - $j = 0$, parse $X^0 = (c, pk, \pi)$, $r \leftarrow_{\$} \{0,1\}^\kappa$, compute $c' := \mathsf{UpdateC}(pk, c; r)$ and $\pi' \leftarrow \mathsf{LEval}\,(\omega,\ \mathsf{UpdateC}(pk, \cdot; r),\ (pk, c, \pi))$, return $X^0 := (pk, c', \pi')$.
  - $j = 1$, parse $X^1 = sk$ and compute $sk' \leftarrow \mathsf{UpdateS}(sk)$, return $X^1 := (sk')$.

Theorem 1. *For any polynomial $\tau(\kappa)$, if E is an ℓ'-CLRS-Friendly PKE scheme (Def. 4) with public key space $\mathcal{PK}$ and message space $\mathcal{M}$ and where $\ell'(\kappa) := \ell(\kappa) + \tau(\kappa) \cdot (\max(\kappa + 1, \log(|\mathcal{M}| + 2) + 1, \log|\mathcal{PK}|))$ and if $\mathcal{NIZK}$ is an adaptive multi-theorem zero-knowledge (Def. 1) label-malleable non-interactive argument of knowledge system with malleable label simulation extractability (Def. 2) and label derivation privacy (Def. 3) then the scheme above is a (ℓ, ρ, τ)-Non-Malleable Code with Refresh for any polynomial $\rho(\kappa)$.*

The leakage rate of the encoding scheme depends on the relation between the size of the proofs of the NIZK system and the parameters of the CLRS Friendly PKE. Roughly, setting τ be a constant, assuming the size of the secret key and the size of the ciphertext of the PKE be approximately the same and let $r = |sk|/\ell$ be the leakage ratio of the CLRS-Friendly PKE then the leakage ratio of the coding scheme is strictly less than $r/(2 + 1/poly(\kappa))$. This comes from the extractability property[2] of the NIZK system and the $O(\kappa)$-bits of leakage needed to support tampering attacks.

Proof. The correctness follows immediately from the correctness of the E and $\mathcal{NIZK}$ and $\mathcal{U} \subseteq \mathcal{T}$. The proof of security is divided in two parts. We first define a simulator, then we define a sequence of mental experiments starting with the initial **Tamper** experiment and proceeding toward the **SimTamper** experiment and we prove that the experiments are computationally indistinguishable.

The Simulator. Let $\widetilde{\mathsf{S}}$ be the simulator of the $\mathcal{NIZK}$ as postulated by Def. 1. Given an adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$ consider the following simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3)$ for the **SimTamper** experiment:

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

- Set the variables t^0, t^1, l^0, l^1 to 0.
- Run the $\mathcal{NIZK}$ simulator $(\omega, \tau_{sim}, \tau_{ext}) \leftarrow \widetilde{\mathsf{S}}_0(1^\kappa)$.
- Sample $(pk, sk) \leftarrow \mathsf{Gen}(pub)$, $c \leftarrow \mathsf{Enc}(pk, 0^\kappa)$ and $\pi \leftarrow \tilde{\mathsf{S}}_1(\omega, c, pk)$.

[2] We also are assuming that the NIZK system is not succinct.

- Set the joint state aux to $(\tau_{sim}, \tau_{ext}, X^0, X^1, t^0, t^1)$ where $(X^0, X^1) = ((pk, c, \pi), (sk))$.

Simulator $\mathsf{S}_1(T, z)$:

- Parse $T = (T^0, T^1)$ and aux as $(\tau_{sim}, \tau_{ext}, X^0, X^1, t^0, t^1)$ where $(X^0, X^1) = ((pk, c, \pi), (sk))$.
- Compute $\tilde{X}^i = T^i(X^i)$ for $i \in \{0, 1\}$.
- Check $\tilde{pk} = \mathsf{PK}(\tilde{sk})$ and $\mathsf{V}^{\tilde{c}}(\omega, \tilde{pk}, \tilde{\pi}) = 1$ (check (A) and (B) of Decode), if at least one of the checks fails then return $\perp$.
- Compute $(sk', \phi, c') \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{c},\ \tilde{pk}, \tilde{\pi})$:
  (I) If $\tilde{pk} = \mathsf{PK}(sk')$ then output $\tilde{m} = \mathsf{Dec}(sk', \tilde{c})$;
  (II) If $\phi(c') = \tilde{c}$, $c' = c$ and $T \in \mathcal{T}$ return $*$;
  (III) Else abort.

Simulator $\mathsf{S}_2(z)$:

- Parse aux as $(\tau_{sim}, \tau_{ext}, X^0, X^1, t^0, t^1, l^0, l^1)$ where $(X^0, X^1) = ((pk, c, \pi), (sk))$.
- Upon message (i, L) where $i \in \{0, 1\}$, compute $y \leftarrow L(X^i)$. If $l^i + |y| \leq \lambda$ then update $l^i := l^i + |y|$ and output y else output $\perp$.

Simulator $\mathsf{S}_3(j, z)$:

- Parse aux as $(\tau_{sim}, \tau_{ext}, X^0, X^1, t^0, t^1, l^0, l^1)$ where $(X^0, X^1) = ((pk, c, \pi), (sk))$; If $j \notin \{0, 1\}$ and $t^0, t^1 \leq \tau$ set $X_{i+1} := X_i$; Else:
  - If $j = 0$ or $(t^0 > \tau)$ then compute $c' \leftarrow \mathsf{UpdateC}(pk, c)$ and $\pi' \leftarrow \widetilde{\mathsf{S}}_1(\tau_{sim}, c', pk)$ and reset l^0, t^0 to 0;
  - If $j = 1$ or $(t^1 > \tau)$ then compute $sk' \leftarrow \mathsf{UpdateS}(sk)$ and reset l^1, t^1 to 0.
- Set aux as $(\tau_{sim}, \tau_{ext}, X'^0, X'^1, t^0, t^1, l^0, l^1)$ where $(X'^0, X'^1) = ((pk, c', \pi'), (sk'))$.

The Hybrids. We consider a sequence of mental experiments, starting with the initial **Tamper** experiment which for simplicity we denote by $\mathbf{G}_0$. We summarize the sequence of mental experiments in 4.

Game $\mathbf{G}_0$. This is exactly the game defined by the experiment **Tamper**, where Σ is the coding scheme described above. In particular, the Init algorithm of Σ samples a CRS $\omega \leftarrow \mathsf{I}(1^\kappa)$ for $\mathcal{NIZK}$, a pair $(pk, sk_0) \leftarrow \mathsf{Gen}(pub)$, encrypts $c_0 \leftarrow \mathsf{Enc}(pk, m)$ and computes $\pi_0 \leftarrow \mathsf{P}^c(\omega, pk, sk)$. The Rfrsh^* algorithm if Σ upon input $j = 0$ samples randomness $r_u \leftarrow_\$ \{0, 1\}^\kappa$, defines the transformation $\phi_u(\cdot) := \mathsf{UpdateC}(pk, \cdot\, ; r_u)$ and computes $c_{i+i} := \phi_u(c_i)$ and $\pi_{i+1} := \mathsf{LEval}(\omega, \phi_u,\ (pk, c_i, \pi_i))$.

Game $\mathbf{G}_1$. We change the way the proofs π_{i+1} are refreshed. For each iteration $i \in [\rho(\kappa)]$, the refresh procedure Rfrsh^* upon input $j = 0$ parses X_i^0 as (pk, c_i, π_i), samples randomness $r_u \leftarrow_\$ \{0, 1\}^\kappa$, defines the transformation $\phi_u(\cdot) := \mathsf{UpdateC}(pk, \cdot\, ; r_u)$, computes $c_{i+1} \leftarrow \phi_u(c_i)$ and a *fresh* proof:

$$\pi_{i+1} \leftarrow \mathsf{P}^{c_{i+1}}(\omega, pk, sk_i).$$

Finally, it sets $X_{i+1}^0 := (pk, c_{i+1}, \pi_{i+1})$.

```
$\mathbf{G}_0$, $\mathbf{G}_1$, $\mathbf{G}_2$, $\mathbf{G}_3$, $\mathbf{G}_4$, $\mathbf{G}_5$, $\mathbf{G}_6$, $\mathbf{G}_7$, $\boxed{\mathbf{G}_8}$,
Variables $i, l^0, l^1, t^0, t^1$ set to 0;
$\omega \leftarrow \mathsf{I}(1^\kappa)$; $\omega, \tau_{sim}, \tau_{ext} \leftarrow \widetilde{\mathsf{S}}_0(1^\kappa)$;
$pub \leftarrow E.\mathsf{Setup}(1^\kappa)$;
$crs := (\omega, pub)$;
$(m, z) \leftarrow \mathsf{A}_0(crs)$; $(m_0, st_0) := (m, z)$;
$pk, sk \leftarrow \mathsf{KGen}(pub)$;
$\boxed{c_0 \leftarrow \mathsf{Encode}(pk, m);}$ $c_0 \leftarrow \mathsf{Encode}(pk, 0^\kappa)$;
$\pi_0 \leftarrow \mathsf{P}^c(\omega, pk, sk)$; $\pi_0 \leftarrow \widetilde{\mathsf{S}}_1(\tau_{sim}, c_0, pk)$;
$X_0^0 := (pk, c_0, \pi_0)$;
$X_0^1 := sk_0$; $X_0 := (X_0^0, X_0^1)$;
forall $i < \rho(\kappa)$:
  $(T_{i+1}, st_{i+1}, j) \leftarrow \mathsf{A}_1(m_i, st_i) \leftrightarrows \mathcal{O}^\ell(X_i)$;
  $\tilde{X}_{i+1} := T_{i+1}(X_i)$;
  $(\tilde{pk}, \tilde{c}, \tilde{\pi}), (\tilde{sk}) = \tilde{X}_{i+1}$;
  if $(\mathsf{V}^{\tilde{c}}(\omega, \tilde{pk}, \tilde{\pi}) = 1$ and $\mathsf{PK}(\tilde{sk}) = \tilde{pk})$ then
    $m_{i+1} := \mathsf{Dec}(\tilde{sk}, \tilde{c})$;
    $\boxed{C := \{c_0, \ldots, c_i\};}$ $C := \{c_i\}$;
    $sk', \phi, c' \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{\pi})$;
    if $(\tilde{pk} \neq \mathsf{PK}(sk'))$ and
      $(\phi(c) \neq c'$ or $c' \notin C$ or $\phi \notin \mathcal{T})$;
    then **Abort**
    if $\tilde{pk} = \mathsf{PK}(sk')$
      then $m_{i+1} := I_m(\mathsf{Dec}(sk', \tilde{c}))$;
    if $(\phi(c') = \tilde{c}, c' \in C, \phi \in \mathcal{T})$
      then $m_{i+1} := I_m(*)$;
  else $m_{i+1} := \perp$;
  $X_{i+1} := X_i$;
   For $j' \in \{0,1\}$ if $(t^{j'} > \tau)$ or $(j' = j)$
   $X_{i+1}^{j'} \leftarrow \mathsf{Rfrsh}(crs, (j', X_i^{j'}))$
   Set $\mathcal{O}_\ell(X_i).l^{j'}, t^{j'}$ to 0;
  Increment $t^0, t^1$ and $i$;
Return $\mathsf{A}_2(st_\rho)$.
```

```
Procedure $\mathsf{Rfrsh}(crs, (j, X_i^j))$:
if $j = 0$ then:
  $(pk, c_i, \pi_i) = X_i^0$;
  $r_u \leftarrow\$ \{0,1\}^\kappa$;
  $\phi_u(\cdot) := \mathsf{UpdateC}(pk, \cdot\,; r_u)$;
  $c_{i+1} := \phi_u(c_i)$;
  $\pi_{i+1} \leftarrow \mathsf{LEval}(\omega, \phi_u, X_i^0)$;
  $\boxed{\pi_{i+1} \leftarrow \mathsf{P}^{c_{i+1}}(\omega, pk, sk);}$
  $\pi_{i+1} \leftarrow \widetilde{\mathsf{S}}_1(\tau_{sim}, c_{i+1}, pk)$;
  $X_{i+1}^0 := (pk, c_{i+1}, \pi_{i+1})$;
if $j = 1$ then;
  $sk_i = X_i^1$;
  $sk_{i+1} \leftarrow \mathsf{UpdateS}(sk_i)$;
  $X_{i+1}^1 := (sk_{i+1})$;
```

Fig. 4: Games in the proof of Theorem 1. Game $\mathbf{G}_0$ does not execute any of the colored actions, whereas each colored game executes all actions from the previous game plus the ones of the corresponding color. $\mathbf{G}_6$ executes all actions from the previous game but it does not execute the dash-boxed instructions. Additionally, $\mathbf{G}_8$ does not execute any of the boxed instructions.

Game $\mathbf{G}_2$. We change the way the CRS for the $\mathcal{NIZK}$ and the proofs π_i are computed. Let $\omega, \tau_{sim}, \tau_{ext} \leftarrow \widetilde{\mathsf{S}}_0(1^\kappa)$ and for $i \in [\rho(\kappa)]$ if Rfrsh^* is called at the i-th iteration with input $j = 0$ then the proof π_{i+1} is computed as:

$$\pi_{i+1} \leftarrow \widetilde{\mathsf{S}}_1(\tau_{sim}, c_{i+1}, pk).$$

Also the proof π_0 is computed in the same way.

Game $\mathbf{G}_3$. We extract the witness from the proof $\tilde{\pi}$ and abort if the extraction procedure fails. The game is the same as $\mathbf{G}_2$ but, for each iteration i, let $\tilde{X}_{i+1}$ be the tampered codeword where $\tilde{X}_{i+1} = (\tilde{pk}, \tilde{c}, \tilde{\pi}), (\tilde{sk})$. The game first checks if $\mathsf{V}^{\tilde{c}}(\omega, \tilde{pk}, \tilde{\pi}) = 1$ and if so then it runs:

$$sk', \phi, c' \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{\pi}).$$

Let C be the set of ciphertexts produced by the game until the i-th iteration. Namely $C := \{c_0, \ldots, c_i\}$. If both the conditions: (i) $\tilde{pk} = \mathsf{PK}(sk')$ and (ii) $\phi(c') = \tilde{c}$, $c' \in C$ and $\phi \in \mathcal{T}$ do not hold then the game outputs a special symbol **Abort**.

Game $\mathbf{G}_4$. We change the output of Decode to match point (I) of the simulator S_1. The game is the same as $\mathbf{G}_3$ but, for each iteration $i \in [\rho(\kappa)]$, after the extraction, if the condition $\tilde{pk} = \mathsf{PK}(sk')$ holds, it sets the message $m_{i+1} := I_m(\mathsf{Dec}(sk', \tilde{c}))$.

Game $\mathbf{G}_5$. We change the output of Decode. The game is the same as $\mathbf{G}_4$ but, for each iteration $i \in [\rho(\kappa)]$, after the i-th extraction, if the conditions $\phi(c') = \tilde{c}$, $c' \in C$, where $C = \{c_0, \ldots, c_i\}$ and $\phi \in \mathcal{T}$ hold, it sets the message $m_{i+1} := I_m(*)$ (the original message).

Game $\mathbf{G}_6$. We do not decode explicitly anymore. The game is the same as $\mathbf{G}_5$ but, for each iteration $i \in [\rho(\kappa)]$, in the execution of the decoding algorithm Decode, we do not execute the instruction $m_{i+1} := \mathsf{Dec}(\tilde{sk}, \tilde{c})$.

Game $\mathbf{G}_7$. We change the output of Decode to match point (II) of the simulator S_1. The game is the same as $\mathbf{G}_6$ but, for each iteration $i \in [\rho(\kappa)]$, after the i-th extraction, let the set C be redefined as the singleton containing the ciphertext produced after the last refresh, namely $C := \{c_i\}$, the game checks that the conditions $\phi(c') = \tilde{c}$, $c' \in C$ (instead of $c' \in \{c_0, \ldots, c_i\}$) and $\phi \in \mathcal{T}$ hold then it sets the message $m_{i+1} := I_m(*)$ (the original message).

Game $\mathbf{G}_8$. We replace the ciphertext c with a dummy ciphertext. The game is the same as $\mathbf{G}_7$ but it sets $c \leftarrow \mathsf{Enc}(pk, 0^\kappa)$ (instead of $c \leftarrow \mathsf{Enc}(pk, m)$).

It is easy to check that $\mathbf{G}_8$ is equivalent to the $\mathbf{SimTamper}_{\mathsf{A},\mathsf{S}}$.

Lemma 1. *For all PPT adversaries* A *there exists a negligible function* $\nu_{0,1} : \mathbb{N} \to [0,1]$ *such that* $|\mathbb{P}[\mathbf{G}_0(\kappa) = 1] - \mathbb{P}[\mathbf{G}_1(\kappa) = 1]| \leq \nu_{0,1}(\kappa)$.

Proof. We reduce to label derivation privacy of $\mathcal{NIZK}$ via an hybrid argument. For any $l \in [\rho(\kappa) + 1]$, let $\mathbf{Hyb}_l$ be the hybrid experiment that executes the same code of $\mathbf{G}_1$ until the l-th iteration (the proofs are *new*) and then executes the same code of $\mathbf{G}_1$ (the proofs are *re-labeled*). In particular, for any l, in the hybrid $\mathbf{Hyb}_l$, for any $0 \leq i < l$ the proof π_i is computed as $\mathsf{P}^{c_{i+1}}(\sigma, pk, sk)$ while for $i \geq l$ the proof π_i is computed as $\mathsf{LEval}\,(\omega, \phi_u,\ (pk, c_i, \pi_i))$. Moreover, $\mathbf{Hyb}_{\rho+1}$ is equivalent to $\mathbf{G}_1$ while $\mathbf{Hyb}_1$ is equivalent to $\mathbf{G}_0$.

Suppose there exist a PPT adversary A, an index $l \in [\rho(\kappa)]$ and a polynomial $p(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, the adversary A distinghuishes between $\mathbf{Hyb}_l$ and $\mathbf{Hyb}_{l+1}$ with probability at least $1/p(\kappa)$.

We can construct an adversary B that breaks label derivation privacy. The adversary B with input $\omega \leftarrow \mathsf{I}(1^\kappa)$ runs the code of hybrid $\mathbf{Hyb}_l$ on A until the l-th iteration. At this point B forwards to its own challenger the tuple $(\phi_u, c_{l-1}, pk, sk, \pi_{l-1})$ where $\phi_u(\cdot) := \mathsf{UpdateC}(pk, \cdot\ ; r_u)$ with $r_u \leftarrow\!\!\$\ \{0,1\}^\kappa$, and receives back the proof π'. Notice that $c_l := \phi_u(c_{l-1})$.

If the challenge bit is $b = 0$ then $\pi' \leftarrow \mathsf{P}^{c_l}(\omega, pk, sk)$, and therefore B perfectly simulates $\mathbf{Hyb}_{l+1}$ otherwise if $b = 1$ then $\pi' \leftarrow \mathsf{LEval}(\omega, \phi_u, (pk, c_{l-1}, \pi_{l-1}))$, therefore B perfectly simulates $\mathbf{Hyb}_l$. Therefore B can break label derivation privacy of $\mathcal{NIZK}$ with advantage $1/p(\kappa)$.

Lemma 2. *For all PPT adversaries* A *there exists a negligible function* $\nu_{1,2} : \mathbb{N} \to [0,1]$ *such that* $|\mathbb{P}[\mathbf{G}_1(\kappa) = 1] - \mathbb{P}[\mathbf{G}_2(\kappa) = 1]| \leq \nu_{1,2}(\kappa)$.

Proof. We reduce to adaptive multi-theorem zero-knowledge of $\mathcal{NIZK}$.

Suppose there exist a PPT adversary A and a polynomial p such that, for infinitely many values of $\kappa \in \mathbb{N}$, $|\mathbb{P}[\mathbf{G}_1(\kappa) = 1] - \mathbb{P}[\mathbf{G}_2(\kappa) = 1]| \geq 1/p(\kappa)$. Let B be a PPT adversary for the multi-theorem zero-knowledge game that runs the same code of $\mathbf{G}_2$ but for any i, instead of computing the proof π_i, forwards to its oracle the query (c_i, pk, sk).

The view provided by B to A is equivalent to $\mathbf{G}_2$ if B's oracle is P and equivalent to $\mathbf{G}_3$ if B's oracle is $\mathcal{SIM}(\tau_{sim}, \cdot)$. Therefore B can break multi-theorem zero-knowledge of $\mathcal{NIZK}$ with advantage $1/p(\kappa)$.

Lemma 3. *For all PPT adversaries* A *there exists a negligible function* $\nu_{2,3} : \mathbb{N} \to [0,1]$ *such that* $|\mathbb{P}[\mathbf{G}_2(\kappa) = 1] - \mathbb{P}[\mathbf{G}_3(\kappa) = 1]| \leq \nu_{2,3}(\kappa)$.

Proof. We reduce to the $\mathcal{T}$-Malleable label simulation extractability of $\mathcal{NIZK}$. Let Abort be the event that the game $\mathbf{G}_3$ aborts with message **Abort**. Notice that the two games proceed exactly the same until the event Abort happens. Therefore, we have

$$|\mathbb{P}[\mathbf{G}_2(\kappa) = 1] - \mathbb{P}[\mathbf{G}_3(\kappa) = 1]| \leq \mathbb{P}[\mathsf{Abort}].$$

Suppose there exist a PPT adversary A and a polynomial p such that, for infinitely many values of $\kappa \in \mathbb{N}$, $\mathbb{P}[\mathsf{Abort}] \geq 1/p(\kappa)$, where the probability is over the game $\mathbf{G}_3$ with adversary A.

Let B be a PPT adversary for the malleable label simulation extractability that runs the same code of $\mathbf{G}_3$ but for any i, instead of computing the proof π_i, forwards to its oracle the query (c_i, pk) and, if the event during the i-th iteration the message **Abort** is raised, outputs the value $\tilde{X}^0_{i+1} = (\tilde{pk}, \tilde{c}, \tilde{\pi})$. Notice, that the message **Abort** is raised only if the winning condition of the malleable label simulation extractability experiment are met. Therefore the winning probability of B is the probability of the event Abort in $\mathbf{G}_3$.

Lemma 4. $\mathbb{P}[\mathbf{G}_3(\kappa) = 1] = \mathbb{P}[\mathbf{G}_4(\kappa) = 1]$.

Proof. Notice that the two games proceed the same until $\mathsf{PK}(\tilde{sk}) = \mathsf{PK}(sk')$ but $\mathsf{Dec}(\tilde{sk}, \tilde{c}) \neq \mathsf{Dec}(sk', \tilde{c})$. Let $\mathsf{WrongDec}$ be such event. Then we have

$$|\mathbb{P}[\mathbf{G}_3(\kappa) = 1] - \mathbb{P}[\mathbf{G}_4(\kappa) = 1]| \leq \mathbb{P}[\mathsf{WrongDec}].$$

By the correctness of E we have that the event $\mathsf{WrongDec}$ has probability 0.

Lemma 5. $\mathbb{P}[\mathbf{G}_4(\kappa) = 1] = \mathbb{P}[\mathbf{G}_5(\kappa) = 1]$.

Proof. Notice that two games proceed the same until $\phi(c_i) = \tilde{c}$ and $\phi \in \mathcal{T}$ but $\mathsf{Dec}(sk', \tilde{c}) \neq m$ (the original message). Let $\mathsf{NotSame}$ be such event. Therefore, we have

$$|\mathbb{P}[\mathbf{G}_3(\kappa) = 1] - \mathbb{P}[\mathbf{G}_4(\kappa) = 1]| \leq \mathbb{P}[\mathsf{NotSame}].$$

The definition of the set $\mathcal{T}$ and $\phi \in \mathcal{T}$ together with the fact that c_i is an encryption of m under pk and $\phi(c_i) = \tilde{c}$ imply that $\mathsf{Dec}(sk', \tilde{c}) = m$. In fact $\phi \in \mathcal{T}$ implies that $\mathsf{Dec}(sk, \phi(c))$ decrypts correctly if c is a valid ciphertext under pk and $pk = \mathsf{PK}(sk)$. Therefore, we have that the event $\mathsf{NotSame}$ has probability 0.

Lemma 6. $\mathbb{P}[\mathbf{G}_5(\kappa) = 1] = \mathbb{P}[\mathbf{G}_6(\kappa) = 1]$.

Proof. $\mathbf{G}_6$ does not execute the instruction $m_{i+1} := \mathsf{Dec}(\tilde{sk}, \tilde{c})$, however notice that already in game $\mathbf{G}_5$ either the value m_{i+1} is overwritten or the game outputs **Abort**. So the two game are semantically the same.

Lemma 7. *For all PPT adversaries* A *there exists a negligible function* $\nu_{6,7} : \mathbb{N} \to [0, 1]$ *such that* $|\mathbb{P}[\mathbf{G}_6(\kappa) = 1] - \mathbb{P}[\mathbf{G}_7(\kappa) = 1]| \leq \nu_{6,7}(\kappa)$.

Proof. We reduce to the CLRS security of E via an hybrid argument. For $l \in [\rho(\kappa)]$ let $\mathbf{Hyb}_l$ be an hybrid experiment that executes the code of $\mathbf{G}_6$ until the $(l-1)$-th iteration and, after that, executes the code of $\mathbf{G}_7$. Specifically, for every $i < l$ the hybrid $\mathbf{Hyb}_l$, at the i-th iteration, runs the extractor and checks if the conditions $T'(c) = \tilde{c}$, $c' \in \{c_0, \ldots, c_i\}$ and $T \in \mathcal{T}$ hold, and, if yes, it sets $m_{i+1} := m$. For every $i \geq l$ the hybrid $\mathbf{Hyb}_l$, at the i-th iteration, runs the extractor and checks if the conditions $T'(c) = \tilde{c}$, $c' = c_i$ and $T \in \mathcal{T}$ hold, and, if yes, it sets $m_{i+1} := m$. In particular, $\mathbf{Hyb}_0$ is equivalent to $\mathbf{G}_7$ while $\mathbf{Hyb}_\rho$ is equivalent to $\mathbf{G}_8$.

Given an adversary A and an index $k \in [l-1]$ define the event OldCT_k over the random experiment $\mathbf{Hyb}_l$ to hold if A at the l-th iteration outputs a tampering function T_l such that $T_l(X_l^0) = (\tilde{pk}, \tilde{c}, \tilde{\pi})$ and, let $(\perp, T', c') \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{c}, \tilde{\pi})$, then $c' = c_k$.

Let OldCT be the event $\{\exists k \in [l-1] : \mathsf{OldCT}_k\}$. It is easy to check that

$$\left|\mathbb{P}\left[\mathbf{Hyb}_l = 1\right] - \mathbb{P}\left[\mathbf{Hyb}_{l+1} = 1\right]\right| \leq \mathbb{P}\left[\mathsf{OldCT}\right].$$

In fact, if the event OldCT does not happen in $\mathbf{Hyb}_l$ then the condition $c' = c_l$ holds, therefore the two hybrids behave exactly the same.

Suppose there exist an adversary A and a polynomial $p(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, the adversary A distinguishes between game $\mathbf{G}_6$ and $\mathbf{G}_7$ with probability at least $1/p(\kappa)$. Then $\mathbb{P}\left[\mathsf{OldCT}\right] \geq 1/p(\kappa)$.

We build a PPT adversary B that breaks CLRS Friendly PKE Security of E. Let $\mathcal{H}$ a family of 2-wise independent hash functions with domain $\mathcal{C}_E$ and co-domain $\{0,1\}^\kappa$. We introduce some useful notation in Fig. 5. A formal description of B follows:

Adversary B:

1. Receive pub, pk from the challenger of the CLRS security experiment and get oracle access to $\mathcal{O}_\ell(\mathsf{state})$.
2. Set variables i, t^0, t^1 to 0 (as in the **Tamper** experiment).
3. Run $\omega, \tau_{sim}, \tau_{ext} \leftarrow \widetilde{\mathsf{S}}_0(1^\kappa)$, set $crs := (pub, \omega)$ and send crs to A_0.
4. Let m, z be the output from A_0, let m' be a valid plaintext for E such that the first bit of m' differs from the first bit of m and $|m'| = |m|$. Send the tuple m, m' to the challenger. Set $st_0 := z$.
5. For $i = 0$ to $(l-1)$ execute the following loop:
   (a) Sample $r_i \leftarrow\!\!\$\ \{0,1\}^\kappa$ and run the adversary $\mathsf{A}_1(m_i, st_i)$, upon query (j, L) from A_1, if $j = 1$ forward the same query to the leakage oracle, if $j = 0$ forward the query $(0, \mathbf{L}_{L, r_i, pk})$ to the leakage oracle.
   (b) Eventually, the adversary A_1 outputs (T_{i+1}, st_{i+1}, j).
   (c) Forward the query $(1, \mathsf{PK}(T^1_{i+1}(\cdot)))$ to the leakage oracle and let pk' be the answer. Forward the query $(0, \mathbf{V}_{T^0_{i+1}, r_i, pk, pk'})$ to the leakage oracle and let a be the answer, if $a = 0$ then set $m_{i+1} := \bot$ and continue to the next cycle.
   (d) Otherwise, forward the query $(0, \mathbf{M}_{T_{i+1}, r_i, pk})$ and let m' be the answer, if m' is $\mathbf{Abort}^*$ then abort, otherwise set $m_{i+1} := m'$. Execute the refresh algorithm $\mathsf{Rfrsh}^*(j)$ as defined by $\mathbf{G}_6$. (In particular, use the trapdoor τ_{sim} to sample $\pi_{i+1} \leftarrow \widetilde{\mathsf{S}}_1(\tau_{sim}, c_{i+1}, pk)$.)
6. Sample $H \leftarrow\!\!\$\ \mathcal{H}$ and $r_l \leftarrow\!\!\$\ \{0,1\}^\kappa$ and run the adversary A_1 on input (m_{l-1}, st_{l-1}) and reply to the leakage oracle queries as in step (5a). Eventually, the adversary A_1 outputs (T_l, st_l, j) forward the leakage query $(0, \mathbf{H}_{T_l, r_l, pk, H})$, let h be the answer of the leakage oracle.
7. Set x to be the empty string. For $i := 1$ to η, where $\eta := 2p^2(\kappa) + 2p(\kappa)|c|$, execute the following:
   (a) Sample $r_{l+i} \leftarrow\!\!\$\ \{0,1\}^\kappa$ and run the adversary A_1 on input (m_{l-1}, st_{l-1}) and reply to the leakage oracle queries as in step (5a).
   (b) Eventually the adversary A_1 outputs (T_l, st_l, j), forward the query $(0, \mathbf{H}_{T_l, r, pk, x})$ to the leakage oracle, let a the answer, if $a \neq \bot$ set $x := x \| a$.

- $\mathbf{L}_{L,r,pk}(c)$:

```
Return $L(pk, c, \mathsf{S}_1(\tau_{sim}, c, pk; r))$.
```

- $\mathbf{V}_{T,r,pk,pk'}(c)$:

```
$(\tilde{pk}, \tilde{c}, \tilde{\pi}) := T(pk, c, \mathsf{S}_1(\tau_{sim}, c, pk; r))$;
Output $(\mathsf{V}^{\tilde{c}}(\omega, \tilde{pk}, \tilde{\pi}) = 1 \ \wedge \ \tilde{pk} = pk')$.
```

- $\mathbf{M}_{T,r,pk}(c)$:

```
$(\tilde{pk}, \tilde{c}, \tilde{\pi}) := T(pk, c, \mathsf{S}_1(\tau_{sim}, c, pk; r))$;
$sk', T', c' \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{\pi})$;
if $\tilde{pk} = \mathsf{PK}(sk')$ output $\mathsf{Dec}(sk', \tilde{c})$;
if $(T'(c) = c', c' \in C, T' \in \mathcal{T})$ output $*$;
Output $\mathbf{Abort}^*$.
```

- $\mathbf{H}_{T,r,pk,H}(c)$:

```
$(\tilde{pk}, \tilde{c}, \tilde{\pi}) := T(pk, c, \mathsf{S}_1(\tau_{sim}, c, pk; r))$;
$sk', T', c' \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{\pi})$;
If $(c' \notin \{c, \bot\})$ output $H(c)$.
else output $\bot$.
```

- $\mathbf{C}_{T,r,pk,H,x,h}(c)$:

```
$(\tilde{pk}, \tilde{c}, \tilde{\pi}) := T(pk, c, \mathsf{S}_1(\tau_{sim}, c, pk; r))$;
$sk', T', c' \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{\pi})$;
if $(T'(c) \neq c'$ or $T' \notin \mathcal{T})$ then output $\bot$;
if $h = H(c)$ then
  if $|x| < |c|$ then output $(c_{(|x|+1)})$,
  else output the empty string.
```

Fig. 5: Leakage functions on the ciphertext of E.

(c) Call the oracle $\mathsf{Update}(0)$ and increase the counter i.

8. If $|x| < |c|$ then sample $b' \leftarrow\!\!\$\ \{0,1\}$ and output b'. Otherwise, query the leakage oracle with $(1, (\mathsf{Dec}(\cdot, x))_{(0)})$ and let a be the answer. If $a = m_{(0)}$ output 0 else output 1.

We compute the amount of leakage performed by B. For any executions of the loop in step (5) the adversary B forwards all the leakage queries made by A and, additionally:

- In step (5c) leaks $\log|\mathcal{PK}|$ bits from the secret key and 1 bit from the ciphertext;
- In step (5d) leaks $\log(|\mathcal{M}| + 2)$ bits from the ciphertext (the output is either a message or $*$ or **Abort**);

Notice that τ many of the leakage queries described above are allowed before an invocation of Update is forced. Moreover, in step (6) the adversary B leaks κ bit from the ciphertext and for any executions of the loop in step (7) leaks 1 bit from the ciphertext and then it calls the Update algorithm.

Let ℓ_{A} the maximum amount of leakage between each invocation of the Rfrsh^* algorithm done by A, then the amount of leakage done by B is:

$$\ell' = \ell_{\mathsf{A}} + \tau \cdot (\max(\kappa + 1, \log(|\mathcal{M}| + 2) + 1, \log|\mathcal{PK}|))$$

We compute the winning probability of B. Let $c_0, \ldots, c_{l+\eta}$ be the set of ciphertexts produced (either by Enc, in the case of c_0, or by the $\mathsf{UpdateC}$ procedure otherwise) during the CLRS Security Experiment with B. Consider the following events and random variables:

- Let $\mathsf{Collision}$ be the event $\{\exists i, j \leq [l + \eta] : i \neq j \wedge H(c_i) = H(c_j)\}$;

- Let Hit be the event that $\{\exists k < l : h = H(c_k)\}$, where h is the output of the leakage query $(0, \mathbf{H}_{T_l, r_l, pk, H})$ (see step 6).
- Let Hit_i be the random variable equal to 1 if the condition $(h = H(c))$ in the i-th execution of the leakage query $(0, \mathbf{H}_{T_l, r_{l+i}, pk, x})$ (see step 7) holds, 0 otherwise.
- Let $\mathsf{Complete}$ be the event $|x| = |c|$.

It is easy to check that if $(\neg\mathsf{Collision} \wedge \mathsf{Hit} \wedge \mathsf{Complete})$ holds then, at step (8), there exist a positive index $k < l$ such that $(x = c_k)$ holds. Therefore conditioned on the conjunction of the events the adversary B wins[3] with probability 1.

Claim. $\mathbb{P}[\mathsf{Collision}] \le (\eta + l)^2 \cdot 2^{-\kappa}$.

Proof. Recall that H is 2-wise independent, therefore for any fixed $x, y \in \mathcal{C}_E$ such that $x \neq y$, $\mathbb{P}[H(x) = H(y)] = 2^{-\kappa}$, where the probability is taken over the sampling of H. Moreover, the ciphertexts c_i for $i \in [l + \eta]$ are sampled independently of the choice of H, therefore given two indices i, j where $i \neq j$, by averaging over all the possible assignment of c_i, c_j we have that $\mathbb{P}[H(c_i) = H(c_j)] = 2^{-\kappa}$. By union bound we get the claim.

Claim. $\mathbb{P}[\mathsf{Hit} \mid b = 0] = \mathbb{P}[\mathsf{OldCT}]$.

Proof. In fact, the adversary B (on challenge the ciphertext $\mathsf{Enc}(pk, m)$) follows the code of $\mathbf{Hyb}_l$ until step 6. In particular, B has only oracle access to the ciphertext and the secret key (as prescribed by the CLRS Security experiment), while the hybrid $\mathbf{Hyb}_l$ has full access to them. However, the adversary B can perform the same operations via its own leakage oracle access. Therefore, in the execution of the leakage query $(0, \mathbf{H}_{T_l, r_l, pk, H})$ at step (6), the event $c' = c_k$ where $sk', T', c' \leftarrow \mathsf{Ext}(\tau_{ext}, \tilde{pk})$ holds with the same probability of the event OldCT in the hybrid $\mathbf{Hyb}_l$.

Claim. $\mathbb{P}[\mathsf{Complete}] \le 2^{-2\kappa+1}$.

Proof. Let φ be the variable that denotes all the randomness used (including the challenger randomness) in the CLRS experiment between the challenger and the adversary B just before the execution of the step 7. Let Good the event that $\{\mathbb{P}[\mathsf{Hit}] \ge 1/2p(\kappa)\}$. By a Markov argument the probability $\mathbb{P}[\varphi \in \mathsf{Good}]$ is at least 1/2. We can condition on the event Good.

We analyze the random variables $\{\mathsf{Hit}_i\}_{i \in [\eta]}$. Fixing the choice of the randomness φ, for any i, the random variable Hit_i depends only on r_{l+i} and on the output of $\mathsf{UpdateC}$ at the $(l + i)$-th invocation. Notice that the adversary B at each iteration of step 7 samples a fresh r_{l+i}, moreover by the perfectly ciphertext-update privacy (see Def. 5) of E, for any $j \neq i$ the ciphertext c_i and c_j are independent (in fact, for any k the distribution of c_{k+1} does not depend on

[3] Notice we assume perfect correctness of E.

the value of c_k). Therefore, the random variables $\{\mathsf{Hit}_i\}_{i\in[\eta]}$ for any assignment of φ are independent. Let $Z := \sum_{j\in[\eta]} \mathsf{Hit}_j$, we have that $\mathbb{E}\left[Z \mid \mathsf{Good}\right] \geq \eta/2p(\kappa)$.

$$\begin{aligned}
&\mathbb{P}\left[\neg\mathsf{Complete} \mid \mathsf{Good}\right] = \\
&\quad\mathbb{P}\left[Z < |c| \mid \mathsf{Good}\right] = \mathbb{P}\left[Z < \mathbb{E}\left[Z \mid \mathsf{Good}\right] - (\mathbb{E}\left[Z \mid \mathsf{Good}\right] - |c|) \mid \mathsf{Good}\right] = \\
&\quad\mathbb{P}\left[Z < \mathbb{E}\left[Z \mid \mathsf{Good}\right] - p(\kappa)\cdot\kappa \mid \mathsf{Good}\right] \leq 2^{-2\kappa}
\end{aligned}$$

Where, in the last step of the above disequations, we used the Chernoff bound.

Let $\mathsf{Guess} := (\neg\mathsf{Hit} \vee \neg\mathsf{Complete})$, namely the event that triggers B to guess the challenge bit at random. Obviously, for any $a \in \{0,1\}$, $\mathbb{P}\left[b' = b \mid \mathsf{Guess}, b = a\right] = \frac{1}{2}$. For any $a \in \{0,1\}$ and infinitely many κ:

$$\begin{aligned}
&\mathbb{P}[b' = b] \geq \\
&\quad\geq \tfrac{1}{2}\mathbb{P}\left[\mathsf{Guess}\right] + \mathbb{P}\left[b' = b \wedge \mathsf{Hit} \wedge \mathsf{Complete}\right] \\
&\quad\geq \tfrac{1}{2}\mathbb{P}\left[\mathsf{Guess}\right] + \mathbb{P}\left[\neg\mathsf{Collision} \wedge \mathsf{Hit} \wedge \mathsf{Complete}\right] && (1) \\
&\quad\geq \tfrac{1}{2}\mathbb{P}\left[\mathsf{Guess}\right] + (\mathbb{P}\left[\mathsf{Hit}\right] - \mathbb{P}\left[\neg\mathsf{Complete}\right] - \mathbb{P}\left[\mathsf{Collision}\right]) \\
&\quad\geq \tfrac{1}{2}\mathbb{P}\left[\mathsf{Guess}\right] + \mathbb{P}\left[\mathsf{Hit}\right] - 2^{-2\kappa+1} - (\eta + l)^2 \cdot 2^{-\kappa} \\
&\quad\geq \left(\tfrac{1}{2} - \tfrac{1}{2}\mathbb{P}\left[\mathsf{Hit}\right]\right) + \mathbb{P}\left[\mathsf{Hit}\right] - 2^{-2\kappa+1} - (\eta + l)^2 \cdot 2^{-\kappa} && (2) \\
&\quad\geq \tfrac{1}{2} + \tfrac{1}{4}\cdot(\mathbb{P}\left[\mathsf{OldCT}\right] + \mathbb{P}\left[\mathsf{Hit} \mid b = 1\right]) - ((\eta + l)^2 + 1)\cdot 2^{-\kappa}.
\end{aligned}$$

Where Eq. (1) follows because $\mathbb{P}\left[b' = b \mid \neg\mathsf{Collision} \wedge \mathsf{Hit} \wedge \mathsf{Complete}\right] = 1$ and Eq. (2) follows because $\mathbb{P}\left[\mathsf{Guess}\right] \geq 1 - \mathbb{P}\left[\mathsf{Hit}\right]$.

Lemma 8. *For all PPT adversaries* A *there exists a negligible function* $\nu_{7,8} : \mathbb{N} \to [0,1]$ *such that* $|\mathbb{P}\left[\mathbf{G}_7(\kappa) = 1\right] - \mathbb{P}\left[\mathbf{G}_8(\kappa) = 1\right]| \leq \nu_{7,8}(\kappa)$.

Proof. We reduce to the CLRS Friendly PKE Security of E.

By contradiction, assume that there exists a PPT an adversary and a polynomial $p(\cdot)$ such that for infinitely many values of $\kappa \in \mathbb{N}$, we have that A distinguishes between game $\mathbf{G}_7$ and game $\mathbf{G}_8$ with probability at least $1/p(\kappa)$. We build a PPT adversary B that breaks CLRS Friendly PKE Security of E. The adversary B follows the points (1) to (5) of the adversary defined in Lemma 7 with the following modifications: (i) The adversary B runs internally D; (ii) The messages for the challenge are m and 0^κ; (iii) The cycle in step (5) runs for $i = 0$ to $\rho(\kappa)$; (iv) The adversary B eventually outputs the same output bit as A. Let ℓ_{A} the maximum amount of leakage between each invocation of the Rfrsh^* algorithm done by A, then the amount of leakage done by B is:

$$\ell' = \ell_{\mathsf{A}} + \tau \cdot (\max(\log(|\mathcal{M}| + 2) + 1, \log|\mathcal{PK}|))$$

A formal description of B follows.

Adversary B:

1. Receive pub, pk from the challenger of the CLRS security experiment and get oracle access to $\mathcal{O}_\ell(\mathsf{state})$.

2. Set variables $i, \mathsf{flg}, l^0, l^1, t^0, t^1$ to 0 (as in the **Tamper** experiment).
3. Run $\omega, \tau_{sim}, \tau_{ext} \leftarrow \widetilde{\mathsf{S}}_0(1^\kappa)$, set $crs := (pub, \omega)$ and send crs to A_0.
4. Let m, z be the output from A_0. Send $(m, 0^\kappa)$ to the challenger and set $st_0 := z$.
5. For $i = 0$ to $\rho(\kappa)$ execute the following loop:
   (a) Sample $r_i \leftarrow_{\$} \{0,1\}^\kappa$ and run the adversary $\mathsf{A}_1(m_i, st_i)$, upon query (j, L) from A_1, if $j = 1$ forward the same query to the leakage oracle, if $j = 0$ forward the query $(0, \mathbf{L}_{L, r_i, pk})$ to the leakage oracle.
   (b) Eventually, the adversary A_1 outputs (T_{i+1}, st_{i+1}, j).
   (c) Forward the query $(1, \mathsf{PK}(T^1_{i+1}(\cdot)))$ to the leakage oracle and let pk' be the answer. Forward the query $(0, \mathbf{V}_{T^0_{i+1}, r_i, pk, pk'})$ to the leakage oracle and let a be the answer, if $a = 0$ then set $m_{i+1} := \bot$ and continue to the next cycle.
   (d) Otherwise, forward the query $(0, \mathbf{M}_{T_i, r_i, pk})$ and let m' be the answer, if m' is **Abort*** then abort, otherwise set $m_{i+1} := m'$.
   (e) Execute the refresh algorithm $\mathsf{Rfrsh}^*(j)$ as defined by $\mathbf{G}_6$. (In particular, use the trapdoor τ_{sim} to sample $\pi_{i+1} \leftarrow \widetilde{\mathsf{S}}_1(\tau_{sim}, c_{i+1}, pk)$.)
6. Output $\mathsf{A}_2(st_\rho)$.

The view provided by B to A is equivalent to $\mathbf{G}_7$ if the challenge bit b of the PKE Friendly Security experiment is 0. (This because the encrypted message is m.) Otherwise the view is equivalent to $\mathbf{G}_8$.

Wrapping up all together we have that:

$$\left|\mathbb{P}\left[\mathbf{G}_0 = 1\right] - \mathbb{P}\left[\mathbf{G}_8 = 1\right]\right| \leq \sum_{i \in [7]} \left|\mathbb{P}\left[\mathbf{G}_i = 1\right] - \mathbb{P}\left[\mathbf{G}_{i+1} = 1\right]\right| \leq \sum_{i \in [7]} \nu_{i,i+1} \leq negl(\kappa).$$

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 $\mathcal{G}$ be a PPT pairing generation algorithm that upon input the security parameter 1^κ outputs a tuple $gd = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ 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 e is an efficiently computable non-degenerate pairing function from $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. In what follow, we indicate vectors with bold chars and matrices with capital bold chars, all vectors are row vectors, given a group $\mathbb{G}$, two matrices $\boldsymbol{X} \in \mathbb{G}^{n \times m}, \boldsymbol{Y} \in \mathbb{G}^{m \times t}$ for $n, m, t \geq 1$ and an element $a \in \mathbb{G}$ we denote with $\boldsymbol{X} \cdot \boldsymbol{Y}$ the matrix product of $\boldsymbol{X}$ and $\boldsymbol{Y}$ and with $a \cdot \boldsymbol{X}$ the scalar multiplication of $\boldsymbol{X}$ by a. Given two elements $[a]_g \in \mathbb{G}_1$ and $[b]_h \in \mathbb{G}_2$ we denote with $[a]_g \bullet [b]_h = [a \cdot b]_{e(g,h)}$ the value $e([a]_g, [b]_h)$, the notation is extended 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 rows rank j; given a matrix $\boldsymbol{B}$ we let $\mathsf{Rank}(\boldsymbol{B})$ be the rank of $\boldsymbol{B}$.

Definition 7. *The* k-rank hiding *assumption for a pairing generation algorithm* $pub := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, e) \leftarrow_{\$} \mathcal{G}(1^\kappa)$ *states that for any* $i \in \{1, 2\}$ *and for any* $k \leq j, j' \leq \min(n, m)$ *the tuple* $(g_i, [\boldsymbol{B}]_{g_i})$ *and the tuple* $(g_i, [\boldsymbol{B}']_{g_i})$ *for random* $\boldsymbol{B} \leftarrow_{\$} \mathsf{Rk}_j$ *and* $\boldsymbol{B}' \leftarrow_{\$} \mathsf{Rk}_{j'}$ *are computational indistinghuishable.*

The k-rank hiding assumption was introduced by Naor and Segev in [37] where the authors showed to be implied by the more common k-linear (DLIN) assumption. The assumption gets weaker as k increases. In fact for $k = 1$ this assumption is equivalent to DDH assumption. Unfortunately, it is known that DDH cannot hold in symmetric pairings where $\mathbb{G}_1 = \mathbb{G}_2$. However, it is reasonable to assume that DDH holds in asymmetric pairings. This assumption is often called *external Diffie-Hellman* assumption (SXDH) (see [4,9]).

5.1 The Encryption Scheme.

We consider a a slight variation of the the CLRS Friendly PKE scheme of [18]. Consider the following PKE scheme $E = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdateC}, \mathsf{UpdateS})$ with message space $\mathcal{M} := \{0, 1\}$ and parameters $n, m, d \in \mathbb{N}$.

- $\underline{\mathsf{Setup}(1^\kappa)}$: Sample $gd \leftarrow \mathcal{G}(1^\kappa)$ and vectors $\boldsymbol{p}, \boldsymbol{w} \leftarrow_{\$} \mathbb{Z}_q^m$ such that $\boldsymbol{p} \cdot \boldsymbol{w}^T = 0 \mod q$. Return $pub := (gd, [\boldsymbol{p}]_g, [\boldsymbol{w}]_h)$. (Recall that all algorithms implicitly take pub as input.)
- $\underline{\mathsf{Gen}(pub)}$: Sample $\boldsymbol{t} \leftarrow_{\$} \mathbb{Z}_q^m$, $\boldsymbol{r} \leftarrow_{\$} \mathbb{Z}_q^n$ and compute $sk := [\boldsymbol{r}^T \cdot \boldsymbol{w} + \boldsymbol{1}_n^T \cdot \boldsymbol{t}]_h$, set $\alpha := \boldsymbol{p} \cdot \boldsymbol{t}^T$ and compute $pk := [\alpha]_g$. The latter can be computed given only $[\boldsymbol{p}]_g \in \mathbb{G}_1^m$ and $\boldsymbol{t} \in \mathbb{Z}_q^m$. Return (pk, sk).
- $\underline{\mathsf{Enc}(pk, b)}$: Sample $\boldsymbol{u} \leftarrow_{\$} \mathbb{Z}_q^n$ and compute $c_1 := [\boldsymbol{u}^T \cdot \boldsymbol{p}]_g$ and $c_2 := [\alpha \boldsymbol{u} + b \boldsymbol{1}_n]_g$. Return $C := (c_1, c_2)$.
- $\underline{\mathsf{Dec}(sk, C)}$: Let $f = e(g, h)$, parse $sk = [\boldsymbol{S}]_h \in \mathbb{G}_2^{n \times m}$, let $\boldsymbol{S}_1$ be the first row of $\boldsymbol{S}$ and parse $C = ([\boldsymbol{C}]_g, [\boldsymbol{c}]_g) \in \left(\mathbb{G}_1^{n \times m} \times \mathbb{G}_1^n\right)$. Compute $\boldsymbol{b} := [\boldsymbol{c} - \boldsymbol{C} \cdot \boldsymbol{S}_1^T]_f$ and output 1 if and only if $\boldsymbol{b} = [\boldsymbol{1}_n]_f$. In particular, $[\boldsymbol{b}]_f$ can be computed by first computing $[\boldsymbol{c}]_f := e([\boldsymbol{c}]_g, h)$ and then $[\boldsymbol{C} \cdot \boldsymbol{S}_1^T]_f := \prod_i e(\boldsymbol{C}[i], \boldsymbol{S}[i])$.
- $\underline{\mathsf{UpdateC}(pk, C)}$: Parse $C = ([\boldsymbol{C}]_g, [\boldsymbol{c}]_g) \in \left(\mathbb{G}_1^{n \times m} \times \mathbb{G}_1^n\right)$. Sample $\boldsymbol{B} \leftarrow_{\$} \mathbb{Z}_q^{n \times n}$ such that $\boldsymbol{B} \cdot \boldsymbol{1}_n^T = \boldsymbol{1}_n$ and the rank of $\boldsymbol{B}$ is d. Return $([\boldsymbol{B} \cdot \boldsymbol{C}], [\boldsymbol{B} \cdot \boldsymbol{c}^T])$.
- $\underline{\mathsf{UpdateS}(sk)}$: Parse $sk = [\boldsymbol{S}]_h \in \mathbb{G}_2^{n \times m}$. Sample $\boldsymbol{A} \leftarrow_{\$} \mathbb{Z}_q^{n \times n}$ such that $\boldsymbol{A} \cdot \boldsymbol{1}_n^T = \boldsymbol{1}_n$ and the rank of $\boldsymbol{A}$ is d. Return $[\boldsymbol{A} \cdot \boldsymbol{S}]_h$.

Some remarks are in order. First, the main difference between the scheme above and the PKE of [18] is in the public-keys and in the ciphertexts spaces. Let E_{DLWW} be the scheme proposed by [18]. A public key for E_{DLWW} is the target group element $[\boldsymbol{p} \cdot \boldsymbol{t}^T]_f$, while in the scheme above, a public key belongs to the group $\mathbb{G}_1$. Similarly, a ciphertext for E_{DLWW} is a tuple (c_1, c_2) where $c_2 \in \mathbb{G}_T^n$. The message space of E_{DLWW} is $\mathbb{G}_T$, while the message space of the scheme above is $\{0, 1\}$. This is a big disadvantage, however, thanks to this modification, the public keys, secret keys and ciphertexts belong either to $\mathbb{G}_1$ or $\mathbb{G}_2$. As we will see, this modification is necessary to use lM-NIZK based on Groth-Sahai proof systems [30].

Second, we cannot directly derive from the secret key the public key. However, we can define the function PK' that upon input $sk = [\boldsymbol{S}]_h$ produces $pk' = [\boldsymbol{p}]_g \bullet [\boldsymbol{S}_1]_h$, where $\boldsymbol{S}_1$ is the first row of $\boldsymbol{S}$. Notice that the NMC Σ and the simulator S_1 of Theorem 1 need to check if the public key stored in one side is *valid* for the secret key stored in the other side. We can fix[4] this issue by checking the condition $e(pk, h) = \mathsf{PK}'(sk)$ instead.

Theorem 2. *For any $m \geq 6$,$n \geq 3m - 6$,$d := n - m + 3$ the above scheme is an ℓ-CLRS-friendly encryption scheme under the External Diffie-Hellman Assumption on $\mathcal{G}$ for $\ell = \min\{m/6 - 1, n - 3m + 6\} \cdot \log(q) - \omega(\log \kappa)$.*

The proof of security follows the same line of [18]. The PKE scheme E is perfectly ciphertext-update private. See the full version [23] of the paper for more details.

Unfortunately, the message space of the PKE is $\{0, 1\}$ which limits the number of applications of NMC-R. We propose two different way to overcome this weakness:

- For any $k \in \mathbb{N}$ let $E^{\times k}$ the direct product encryption scheme that given a message in $\{0, 1\}^\kappa$ encrypts it bit by bit. In the full version of the paper we show that if E is a ℓ-CLRS-Friendly secure PKE then $E^{\times k}$ is a ℓ-CLRS-Friendly secure PKE. Unfortunately, the leakage-rate of the encoding scheme gets much worse. In fact, the size of the chipertexts increase κ times but the leakage parameter stays the same.
- We define CLRS-Friendly Key Encapsulation Mechanisms and the Non-Malleable Key-Encoding schemes with Refresh. The latter notion is strictly weaker than Non-Malleable Codes, however, it still allows useful applications in the setting of tamper resilience cryptography. We defer all the details to the full version of the paper.

5.2 The Label-Malleable NIZK.

We can cast a lM-NIZK as a special case of the Controlled-Malleable NIZK (cM-NIZK) argument of knowledge systems [10]. Roughly speaking, cM-NIZK systems allow malleability (from a specific set of allowable transformation) both on the instance and on the NIZK proof. Similarly to lM-NIZK AoK systems, cM-NIZK systems have a form of simulation sound extractability called Controlled-Malleable Simulation Sound Extractability (cM-SSE). Informally, the extractor will either extract a valid witness or will *track back* to a tuple formed by an instance queried to the simulation oracle and the associated simulated proof.

The elegant framework of [10] (full version [11]) builds on the malleability of Groth-Sahai proof systems [30] and provides a set of sufficient conditions to have efficient cM-NIZK systems. Here we translate the conditions to the setting of lM-NIZK systems.

[4] In particular, the reduction in Lemma 7 in steps 5c can leak $(1, \mathsf{PK}'(T_{i+1}(\cdot)))$ and then, in step 5d, we need to modify the function $\mathbf{V}_{T^0_{i+1}, r_i, pk, pk'}$ to check if $e(\tilde{pk}, h) = pk'$ and the function $\mathbf{M}_{T_{i+1}, r_i, pk}$ to check if $e(\tilde{pk}, h) = \mathsf{PK}'(sk')$.

Definition 8. *For a relation $\mathcal{R}$ and a set of transformations $\mathcal{T}$ on the set of labels $\mathcal{L}$, we say $(\mathcal{R}, \mathcal{T})$ is* LM-friendly *if the following five 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 $F_s : L_{\mathcal{R}} \to \mathbb{G}_{i_s}^{d_s}$ for some d_s and i_s, $F_w : W_{\mathcal{R}} \to G_{i_d}^{d_w}$ for some d_w and i_w where $L_{\mathcal{R}} := \{x|\exists w : (x, w) \in \mathcal{R}\}$ and $L_{\mathcal{R}} := \{w|\exists x : (x, w) \in \mathcal{R}\}$ and $F_l : \mathcal{L} \to \mathbb{G}_{i_l}^{d_l}$ for some d_l and $i_l = i_s$.*
2. **Representable transformations:** *any transformation in $\mathcal{T}$ can be represented as a set of group elements; i.e., there is an efficiently computable bijection $F_t : \mathcal{T} \to G_{i_t}^{d_t}$ for some d_t and some i_t.*
3. **Provable statements:** *we can prove the statement $(x, w) \in \mathcal{R}$ (using the above representation for x and w) using pairing product equations; i.e., there is a pairing product statement that is satisfied by $F_s(x)$ and $F_w(w)$ iff $(x, w) \in \mathcal{R}$.*
4. **Provable transformations:** *we can prove the statement "$\phi(L') = L \wedge \phi \in \mathcal{T}$" (using the above representations for labels L, L' and transformation ϕ) using a pairing product equation, i.e. there is a pairing product statement that is satisfied by $F_t(\phi), F_l(L), F_l(L')$ iff $T \in \mathcal{T} \wedge\ \phi(L') = L$.*
5. **Transformable transformations:** *for any $\phi, \phi' \in \mathcal{T}$ there is a valid transformation $t(\phi)$ that takes the statement "$\phi(L') = L \wedge\ \phi \in \mathcal{T}$" (phrased using pairing products as above) for the statement "$(\phi' \circ \phi)(L') = \phi(L) \wedge\ (\phi' \circ \phi) \in \mathcal{T}$" and that preserves*[5] *the label L'.*

The definition above is almost verbatim from [11], the only differences are that the point (1) is extended to support labels and that the original definition has a condition on the malleability of the tuple statement/witness (which trivially holds for lM-NIZK). We adapt a theorem of [11] to the case of Label Malleability:

Theorem 3. *If the DLIN assumption holds then we can construct a lM-NIZK that satisfies derivation privacy for any LM-friendly relation and transformation set $(\mathcal{R}, \mathcal{T})$.*

With this powerful tool in our hand, we are now ready to show that there exists a lM-NIZK for the relation and transformation set $(\mathcal{R}_{pub}, \mathcal{T}_{pub})$ defined above:

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

where $pub = (gd, [\boldsymbol{p}]_g, [\boldsymbol{w}]_h) \leftarrow \mathsf{Setup}(1^\kappa)$. Notice that the set of all the possible updates of a ciphertext,

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

is a subset of $\mathcal{T}_{pub}$. Therefore, we can apply the generic transformation of Sec. 4 given a lM-NIZK for the relation $\mathcal{R}_{pub}$ and the set of transformations $\mathcal{T}_{pub}$ and the CLRS-Friendly PKE defined above. We show that the tuple $(\mathcal{R}_{pub}, \mathcal{T}_{pub})$ is LM-Friendly.

[5] See full version for the formal definition.

Representable statements and labels: Notice that $L_{\mathcal{R}_{pub}} \subseteq \mathbb{G}_1$, while the set of valid label is the set $\mathbb{G}_1^{n\times m} \times \mathbb{G}_1^n$.

Representable transformations: We can describe a transformation $\phi_{\boldsymbol{B}} \in \mathcal{T}_{pub}$ as a matrix of elements $[\boldsymbol{B}]_h \in \mathbb{G}_2^{n\times n}$.

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

Provable transformations: Given a transformation $\phi_{\boldsymbol{B}} \in \mathcal{T}_{pub}$ and labels $c = ([\boldsymbol{C}]_g, [\boldsymbol{c}]_g), c' = ([\boldsymbol{C}']_g, [\boldsymbol{c}']_g)$, the statement "$\phi_{\boldsymbol{B}}(c') = c \wedge \phi_{\boldsymbol{B}} \in \mathcal{T}$" is transformed as the system of pairing product statements:

$$\begin{cases} [\boldsymbol{B}]_h \bullet [\boldsymbol{C}'^T]_g = [\boldsymbol{C}]_g \bullet [1]_h \\ [\boldsymbol{B}]_h \bullet [\boldsymbol{c}'^T]_g = [\boldsymbol{c}]_g \bullet [1]_h \\ [\boldsymbol{B}]_h \bullet [\boldsymbol{1}^T]_g = [\boldsymbol{1}]_f \end{cases} \tag{3}$$

Transformable transformations: Let $\phi_{\boldsymbol{B}}, c, c'$ be as before and let $\phi_{\boldsymbol{B}'} \in \mathcal{T}_{pub}$. We show that we can transform the system in Eq. (3) to be a valid system of pairing product statement for the statement $(\phi_{\boldsymbol{B}'} \circ \phi_{\boldsymbol{B}})(c') = \phi_{\boldsymbol{B}'}(c) \wedge (\phi_{\boldsymbol{B}'} \circ \phi_{\boldsymbol{B}}) \in \mathcal{T}$. Given the system of pairing product equations in Eq. (3) and $\boldsymbol{B}' \in \mathbb{Z}_q^{n\times n}$ we can perform operations at the exponent and derive:

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

The Set $\mathcal{T}_{pub}^{\times k}$ of Transformations for $E^{\times k}$. For any $k \in \mathbb{N}$, let the PKE scheme $E^{\times k}$ be defined as in Sec. 5.1, let $\mathcal{C}_{E^{\times k}} = (\mathcal{C}_E)^k$ be the ciphertexts space of $E^{\times k}$ and let $\mathcal{T}_{pub}^{\times k} = (\mathcal{T}_{pub})^k$. Explicitly, the set of transformations is defined as:

$$\mathcal{T}_{pub}^{\times k} = \left\{ \phi_{\bar{B}} \ : \begin{array}{c} \phi_{\bar{B}}(\bar{c}) = \left([\boldsymbol{B}^i \cdot \boldsymbol{C}^{\boldsymbol{i}T}]_g, [\boldsymbol{B}^i \cdot \boldsymbol{c}^{\boldsymbol{i}T}]_g : i \in [k]\right), \\ \bar{c} = (\boldsymbol{C}^1, \boldsymbol{c}^1), \dots, (\boldsymbol{C}^k, \boldsymbol{c}^k), \\ \bar{B} = \boldsymbol{B}^1, \dots, \boldsymbol{B}^k, \ \forall i \in [k] : \boldsymbol{1} = \boldsymbol{B}^i \cdot \boldsymbol{1}^T \end{array} \right\}$$

For any positive polynomial $k(\kappa)$, the tuple $(\mathcal{R}_{pub}, \mathcal{T}_{pub}^{\times k})$ is LM-Friendly. The result follows straight forward from the framework presented in Sec. B.3 of [11] where it is shown that the for any pair of transformations on statements over pairing product equations we can derive a new transformation for the conjunction of the statements.

6 Applications

Following the same approach of [22,34] we show a compiler that maps any functionality $G(s, \cdot)$ to a Continually-Leakage-and-Tamper Resilient functionality $G'(s', \cdot)$ equipped with refresh procedure Rfrsh. Consider the experiments in Fig. 6.

```
Experiment $\mathbf{IdealFunc}_{\mathsf{S}}^{(G,s)}(\kappa)$:
Variables $\bar{\mathcal{Q}}$ set to $\emptyset$;
$(crs, \mathcal{Q}', st) \leftarrow\$ \mathsf{S}(1^\kappa, G) \leftrightarrows \bar{\mathcal{E}}(G, s)$;
Return $(crs, \bar{\mathcal{Q}} \cup \mathcal{Q}', st)$.

Experiment $\mathbf{TamperFunc}_{\mathsf{A},\Phi}^{(G,s)}(\kappa, \ell, \rho, \tau)$:
Variables $i, k, t^0, t^1, st_0$ set to 0 and $\mathcal{Q} := \emptyset$;
Variables $s'_{j'} := \perp$ for $j' \in [\tau \cdot \rho]$;
$crs \leftarrow\$ \mathsf{Setup}(1^\kappa)$;
$s'_0 \leftarrow\$ \mathsf{MCompile}(crs, s)$;
$G' \leftarrow\$ \mathsf{FCompile}(crs, G)$;
forall $i < \rho(\kappa)$:
    $(st_{i+1}, T, j) \leftarrow \mathsf{A}(crs, st_i) \leftrightarrows \mathcal{O}_\ell(s'_i), \mathcal{E}(G')$;
    $s'_{k+1} := T(s'_0)$;
    If $j \in \{0,1\}$ then $\mathsf{Rfrsh}^*(j)$,
    else $s'_{i+1} := s'_i$;
    For $j' \in \{0,1\}$ if $(t^{j'} > \tau)$
        then $\mathsf{Rfrsh}^*(j')$;
    Increment $k, t^0, t^1$ and $i$;
Return $(crs, \mathcal{Q}, st_\rho)$.
```

```
Oracle $\bar{\mathcal{E}}(G, s)$:
Upon message $x$;
Compute $y \leftarrow G(s, x)$;
$\bar{\mathcal{Q}} := \bar{\mathcal{Q}} \cup \{(x, y)\}$;
Return $y$.

Oracle $\mathcal{E}(G')$:
Upon message $(t, x)$;
If $t \notin [\rho \cdot \tau]$ Return $\perp$
Else $y \leftarrow G'(s'_t, x)$;
    $\mathcal{Q} := \mathcal{Q} \cup \{(x, y)\}$;
    Return $y$.

Procedure $\mathsf{Rfrsh}^*(j)$:
Set $\mathcal{O}_\ell(X_i).l^j, t^j$ to 0;
Increment $t^{1-j}$;
$s'^{1-j}_{i+1} := s'^{1-j}_i$;
$s'^{j}_{i+1} \leftarrow\$ \mathsf{Rfrsh}(crs, (j, s'^{j}_i))$
```

Fig. 6: Experiment defining the security of CLT Resilient Compiler.

Definition 9. *A compiler* $\Phi = (\mathsf{Setup}, \mathsf{FCompile}, \mathsf{MCompile}, \mathsf{Rfrsh})$ *is a Split-State* (ℓ, ρ, τ)*-Continually-Leakage-and-Tamper (for short* (ℓ, ρ, τ)*-CLT) Compiler in the CRS model if for every PPT adversary* A *there exists a simulator* S *such that for every efficient functionality* $G : \{0,1\}^\kappa \times \{0,1\}^i \to \{0,1\}^o$ *for* $\kappa, i, o \in \mathbb{N}$ *and any secret state* $s \in \{0,1\}^\kappa$*, the output of the real experiment* $\mathbf{TamperFunc}_{\mathsf{A},\Phi}^{(G,s)}(\kappa, \ell, \rho, \tau)$ *and the output of the simulated experiment* $\mathbf{IdealFunc}(\kappa)$ *are indistinghuishable.*

Given a NMC-R Σ, consider the following compiler $\Pi = (\mathsf{Setup}, \mathsf{MCompile}, \mathsf{Enc}, \mathsf{FCompile}, \mathsf{Rfrsh})$:

- $\underline{\mathsf{Setup}(1^\kappa)}$: Output $crs \leftarrow\$ \Sigma.\mathsf{Init}(1^\kappa)$;
- $\underline{\mathsf{MCompile}(crs, s)}$: Output $s' \leftarrow\$ \Sigma.\mathsf{Enc}(crs, s)$;
- $\underline{\mathsf{FCompile}(crs, G)}$: Output $G'(s', x) := G(\Sigma.\mathsf{Dec}(crs, s'))$;
- $\underline{\mathsf{Rfrsh}(crs, s')}$: Output $\Sigma.\mathsf{Rfrsh}(crs, s')$.

Theorem 4. *Let* Σ *be a* (ℓ, ρ, τ)*-Non-Malleable Code with Refresh then* Π *as defined above is a Split-State* (ℓ, ρ, τ)*-CTL Compiler in the CRS model.*

References

1. D. Aggarwal, Y. Dodis, and S. Lovett. Non-malleable codes from additive combinatorics. In *STOC*, pages 774–783, 2014.

2. S. Agrawal, D. Gupta, H. K. Maji, O. Pandey, and M. Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In *TCC*, pages 375–397, 2015.
3. M. Ball, D. Dachman-Soled, M. Kulkarni, and T. Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In *EUROCRYPT*, pages 881–908, 2016.
4. L. Ballard, M. Green, B. de Medeiros, and F. Monrose. Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, Report 2005/417, 2005. `http://ia.cr/2005/417`.
5. M. Bellare, D. Cash, and R. Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, pages 486–503, 2011.
6. M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, pages 491–506, 2003.
7. M. Bellare, K. G. Paterson, and S. Thomson. RKA security beyond the linear barrier: IBE, encryption and signatures. In *ASIACRYPT*, pages 331–348, 2012.
8. A. Boldyreva, D. Cash, M. Fischlin, and B. Warinschi. Foundations of non-malleable hash and one-way functions. In *ASIACRYPT*, pages 524–541, 2009.
9. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.
10. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In *EUROCRYPT*, pages 281–300, 2012.
11. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. *IACR Cryptology ePrint Archive*, 2012:12, 2012.
12. E. Chattopadhyay and D. Zuckerman. Non-malleable codes against constant split-state tampering. In *FOCS*, pages 306–315, 2014.
13. Y. Chen, B. Qin, J. Zhang, Y. Deng, and S. S. M. Chow. Non-malleable functions and their applications. In *PKC*, pages 386–416, 2016.
14. D. Dachman-Soled and Y. T. Kalai. Securing circuits and protocols against $1/poly(k)$ tampering rate. In *TCC*, pages 540–565, 2014.
15. D. Dachman-Soled, F. Liu, E. Shi, and H. Zhou. Locally decodable and updatable non-malleable codes and their applications. In *TCC*, pages 427–450, 2015.
16. I. Damgård, S. Faust, P. Mukherjee, and D. Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. In *ASIACRYPT*, pages 140–160, 2013.
17. I. Damgård, S. Faust, P. Mukherjee, and D. Venturi. The chaining lemma and its application. In *ICITS*, pages 181–196, 2015.
18. Y. Dodis, A. B. Lewko, B. Waters, and D. Wichs. Storing secrets on continually leaky devices. In *FOCS*, pages 688–697, 2011.
19. S. Dziembowski, T. Kazana, and M. Obremski. Non-malleable codes from two-source extractors. In *CRYPTO*, pages 239–257, 2013.
20. S. Dziembowski, T. Kazana, and D. Wichs. One-time computable self-erasing functions. In *TCC*, pages 125–143, 2011.
21. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.
22. S. Dziembowski, K. Pietrzak, and D. Wichs. Non-malleable codes. In *Innovations in Computer Science*, pages 434–452, 2010.
23. A. Faonio and J. B. Nielsen. Non-malleable codes with split-state refresh. Cryptology ePrint Archive, Report 2016/1192, 2016. `http://eprint.iacr.org/2016/1192`.
24. A. Faonio and D. Venturi. Efficient public-key cryptography with bounded leakage and tamper resilience. In *ASIACRYPT*, page to appear, 2016.
25. S. Faust, P. Mukherjee, J. B. Nielsen, and D. Venturi. Continuous non-malleable codes. In *TCC*, pages 465–488, 2014.

26. S. Faust, P. Mukherjee, D. Venturi, and D. Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In *EUROCRYPT*, pages 111–128, 2014.
27. S. Faust, K. Pietrzak, and D. Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP*, pages 391–402, 2011.
28. D. Goldenberg and M. Liskov. On related-secret pseudorandomness. In *TCC*, pages 255–272, 2010.
29. V. Goyal, A. O'Neill, and V. Rao. Correlated-input secure hash functions. In *TCC*, pages 182–200, 2011.
30. J. Groth and A. Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.
31. Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private circuits II: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.
32. Z. Jafargholi and D. Wichs. Tamper detection and continuous non-malleable codes. In *TCC, Part I*, pages 451–480, 2015.
33. A. Kiayias and Y. Tselekounis. Tamper resilient circuits: The adversary at the gates. In *ASIACRYPT*, pages 161–180, 2013.
34. F. Liu and A. Lysyanskaya. Tamper and leakage resilience in the split-state model. In *CRYPTO*, pages 517–532, 2012.
35. X. Lu, B. Li, and D. Jia. Related-key security for hybrid encryption. In *Information Security*, pages 19–32, 2014.
36. S. Lucks. Ciphers secure against related-key attacks. In *FSE*, pages 359–370, 2004.
37. M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, volume 2009, pages 18–35, 2009.
38. B. Qin, S. Liu, T. H. Yuen, R. H. Deng, and K. Chen. Continuous non-malleable key derivation and its application to related-key security. In *PKC*, pages 557–578, 2015.
39. H. Wee. Public key encryption against related key attacks. In *PKC*, pages 262–279, 2012.