An extended abstract of this paper is published in the proceedings of the 11th IACR Theory of Cryptography Conference — TCC 2014. This is the full version.

Continuous Non-malleable Codes

Sebastian Faust[1], Pratyay Mukherjee[2], Jesper Buus Nielsen[2], and Daniele Venturi[3]

[1]*EPFL, Switzerland*
[2]*Aarhus University, Denmark*
[2]*Sapienza University, Rome*

March 4, 2014

Abstract

Non-malleable codes are a natural relaxation of error correcting/detecting codes that have useful applications in the context of tamper resilient cryptography. Informally, a code is non-malleable if an adversary trying to tamper with an encoding of a given message can only leave it unchanged or modify it to the encoding of a completely unrelated value. This paper introduces an extension of the standard non-malleability security notion – so-called *continuous* non-malleability – where we allow the adversary to tamper *continuously* with an encoding. This is in contrast to the standard notion of non-malleable codes where the adversary only is allowed to tamper a *single* time with an encoding. We show how to construct continuous non-malleable codes in the common split-state model where an encoding consist of two parts and the tampering can be arbitrary but has to be independent with both parts. Our main contributions are outlined below:

1. We propose a new *uniqueness* requirement of split-state codes which states that it is computationally hard to find two codewords $C = (X_0, X_1)$ and $C' = (X_0, X_1')$ such that both codwords are valid, but X_0 is the same in both C and C'. A simple attack shows that uniqueness is necessary to achieve continuous non-malleability in the split-state model. Moreover, we illustrate that non of the existing constructions satisfies our uniqueness property and hence is not secure in the continuous setting.
2. We construct a split-state code satisfying continuous non-malleability. Our scheme is based on the inner product function, collision-resistant hashing and non-interactive zero-knowledge proofs of knowledge and requires an untamperable common reference string.
3. We apply continuous non-malleable codes to protect arbitrary cryptographic primitives against tampering attacks. Previous applications of non-malleable codes in this setting required to *perfectly erase* the entire memory after each execution and and required the adversary to be restricted in memory. We show that continuous non-malleable codes avoid these restrictions.

Keywords: non-malleable codes, split-state, tamper resilience

Contents

1 Introduction

Physical attacks that target cryptographic implementations instead of breaking the black-box security of the underlying algorithm are amongst the most severe threats for cryptographic systems. A particular important attack on implementations is the so-called tampering attack. In a tampering attack the adversary changes the secret key to some related value and observes the effect of such changes at the output. Traditional black-box security notions do not incorporate adversaries that change the secret key to some related value; even worse, as shown in the celebrated work of Boneh et al. [6] already minor changes to the key suffice for complete security breaches. Unfortunately, tampering attacks are also rather easy to carry out: a virus corrupting a machine can gain partial control over the state, or an adversary that penetrates the cryptographic implementation with physical equipment may induce faults into keys stored in memory.

In recent years, a growing body of work (see [20, 21, 17, 23, 1, 2, 18] and many more) develop new cryptographic techniques that protect against tampering attacks. Non-malleable codes introduced by Dziembowski, Pietrzak and Wichs [17] are an important approach to achieve this goal. Intuitively a code is non-malleable w.r.t. a set of tampering functions $\mathcal{T}$ if the message contained in a codeword modified via a function in $\mathcal{T}$ is either the original message, or a completely unrelated value. Non-malleable codes can be used to protect any cryptographic functionality against tampering with the memory. Instead of storing the key in memory, we store its encoding and decode it each time the functionality wants to accesses the key. As long as the adversary can only apply tampering functions from the set $\mathcal{T}$, the non-malleability property guarantees that the (possibly tampered) decoded value is not related to the original key.

The standard notion of non-malleability considers a one-shot game: the adversary is allowed to tamper a single time with the codeword and obtains the decoded output. In this work we introduce so-called *continuous non-malleable codes*, where non-malleability is guaranteed even if the adversary continuously applies functions from the set $\mathcal{T}$ to the codeword. We show that our new security notion is not only a natural extension of the standard one-shot notion, but moreover allows to protect against tampering attacks in important settings where earlier constructions fall short to achieve security.

Continuous non-malleable codes. A non-malleable code consists of two algorithms $\mathsf{Code} = (\mathrm{Encode}, \mathrm{Decode})$ that satisfy the correctness property $\mathrm{Decode}(\mathrm{Encode}(x)) = x$, for all $x \in \mathcal{X}$. To define non-malleability for a function class $\mathcal{T}$, consider the random variable $\mathsf{Tamper}_{\mathsf{T},x}$ defined for every function $\mathsf{T} \in \mathcal{T}$ and any message $x \in \mathcal{X}$ in the game below:

1. Compute an encoding $X \leftarrow \mathrm{Encode}(x)$ using the encoding procedure.

2. Apply the tampering function $\mathsf{T} \in \mathcal{T}$ to obtain the tampered codeword $X' = \mathsf{T}(X)$.

3. If $X' = X$ then return the special symbol $\mathsf{same}^\star$; otherwise, return $\mathrm{Decode}(X')$. Notice that $\mathrm{Decode}(X')$ may return the special symbol $\perp$ in case the tampered codeword X' was invalid.

A coding scheme Code is said to be (one-shot) non-malleable with respect to functions in $\mathcal{T}$ and message space $\mathcal{X}$, if for every $\mathsf{T} \in \mathcal{T}$ and any two messages $x, y \in \mathcal{X}$ the distributions $\mathsf{Tamper}_{\mathsf{T},x}$ and $\mathsf{Tamper}_{\mathsf{T},y}$ are indistinguishable.

To define continuous non-malleable codes, we do not fix a single tampering function T a-priori.[1] Instead, we let the adversary repeat step 2 and step 3 from the above game a polynomial number of times, where in each iteration the adversary can adaptively choose a tampering function $\mathsf{T}_i \in \mathcal{T}$. We emphasize that this change of the tampering game allows the adversary to tamper continuously with the initial encoding X. As shown by Gennaro *et al.* [20] such a strong security notion is impossible to achieve without further assumptions. To this end, we rely on a self-destruct mechanism as used in earlier works on non-malleable codes. More precisely, when in step 3 the game detects an invalid codeword and returns $\perp$ for the first time, then it self-destructs. This is a rather mild assumption as it can, for instance, be implemented using a single public untamperable bit.

From non-malleable codes to tamper resilience. As discussed above one main application of non-malleable codes is to protect cryptographic schemes against tampering with the secret key [17, 23]. Consider a reactive functionality $\mathcal{G}$ with secret state st that can be executed on input m, e.g., $\mathcal{G}$ may be the AES with key st encrypting messages m. Using a non-malleable code earlier work showed how to transform the functionality $(\mathcal{G}, st)$ into a functionality $(\mathcal{G}^{\mathsf{Code}}, X)$ that is secure against tampering with X. The transformation compiling $(\mathcal{G}, st)$ into $(\mathcal{G}^{\mathsf{Code}}, X)$ works as follows. Initially, X is set to $X \leftarrow \mathrm{Encode}(st)$. Each time $\mathcal{G}^{\mathsf{Code}}$ is executed on input m, the transformed functionality reads the encoding X from memory, decodes it to obtain $st = \mathrm{Decode}(X)$ and runs the original functionality $\mathcal{G}(st, m)$. Finally, it erases the memory and stores the new state $X \leftarrow \mathrm{Encode}(st)$. Additionally to executing evaluation queries the adversary can issue tampering queries $\mathsf{T}_i \in \mathcal{T}$. A tampering query replaces the current secret state X with a tampered state $X' = \mathsf{T}_i(X)$, and the functionality $\mathcal{G}^{\mathsf{Code}}$ continues its computation using X' as the secret state. Notice that in case of $\mathrm{Decode}(X') = \perp$ the functionality $\mathcal{G}^{\mathsf{Code}}$ sets the memory to a dummy value—resulting essentially in a self-destruct.

The above transformation guarantees continuous tamper resilience even if the underlying non-malleable code is secure only against one-shot tampering. This security "boost" is achieved by re-encoding the secret state/key after each execution of the primitive $\mathcal{G}^{\mathsf{Code}}$. As one-shot non-malleability suffices in the above cryptographic application, one may ask why we need continuous non-malleable codes. Besides being a natural extension of the standard non-malleability notion, our new notion has several important applications that we discuss in the next two paragraphs.

Tamper resilience without erasures. The transformation described above necessarily requires that after each execution the entire content of the memory is erased. While such perfect erasures may be feasible in some settings, they are rather problematic in the presence of tampering. To illustrate this issue consider a setting where besides the encoding of a key, the memory also contains other non-encoded data. In the tampering setting, we cannot restrict the erasure to just the part that stores the encoding of the key as a tampering adversary may copy the encoding to some different part of the memory. A simple solution to this problem is to erase the entire memory, but such an approach is not possible in most cases: for instance, think of the memory as being the hard-disk of your computer that besides the encoding of a key stores other important files that you don't want to be erased. Notice that this situation is quite different from the leakage setting, where we also require perfect erasures to achieve

[1] Our actual definition is slightly stronger than what is presented next (cf. Section 3).

continuous leakage resilience. In the leakage setting, however, the adversary cannot mess around with the state of the memory by, e.g., copying an encoding of a secret key to some free space, which makes erasures significantly easier to implement.

One option to prevent the adversary from keeping permanent copies is to encode the entire state of the memory. Such an approach has, however, the following drawbacks.

1. *It is unnatural:* In many cases secret data, e.g., a cryptographic key, is stored together with non-confidential data. Each time we want to read some small part of the memory, e.g., the key, we need to decode and re-encode the entire state—including also the non-confidential data.

2. *It is inefficient:* Decoding and re-encoding the entire state of the memory for each access introduces additional overhead and would result in highly inefficient solutions. This gets even worse as most current constructions of non-malleable codes are rather inefficient.

3. *It does not work in general:* Consider a setting where we want to compute with non-malleable codes in a tamper resilient way (similar in spirit to tamper resilient circuits). Clearly, in this setting the memory will store many independent encodings of different secrets that cannot be erased. Continuous non-malleable codes are hence a first natural step towards non-malleable computation.

Using our new notion of continuous non-malleable codes we can avoid the above issues and achieve continuous tamper resilience without using *erasures* and without relying on inefficient solutions that encode the *entire* state.

Stateless tamper resilient transformations. To achieve tamper resilience from one-shot non-malleability we necessarily need to re-encode the state using fresh randomness. This not only reduces the efficiency of the proposed construction, but moreover makes the transformation stateful. Using continuous non-malleable codes we get continuous tamper resilience for free, eliminating the need to refresh the encoding after each usage. This is in particular useful when the underlying primitive that we want to protect is stateless itself. Think, for instance, of any standard block-cipher construction that typically keeps the same key. Using continuous non-malleable codes the tamper resilient implementation of such stateless primitives does not need to keep any secret state. We discuss the protection of stateless primitives in further detail in Section 5.

1.1 Our Contribution

In this work, we propose the first construction of *continuous non-malleable codes* in the split-state model first introduced in the leakage setting [16, 13]. Various recent works study the split-state model for non-malleable codes [23, 15, 1] (see more details on related work in Section 1.2). In the split-state tampering model, the codeword consists of two halves X_0 and X_1 that are stored on two different parts of the memory. The adversary is assumed to tamper with both parts independently, but otherwise can apply any efficiently computable tampering function. That is, the adversary picks two polynomial-time computable functions T_0 and T_1 and replaces the state (X_0, X_1) with the tampered state $(\mathsf{T}_0(X_0), \mathsf{T}_1(X_1))$. Similar to the earlier work of Liu and Lysyanskaya [23] our construction assumes a public untamperable CRS. Notice that this is a rather mild assumption as the CRS can be hard-wired into the functionality and is independent of any secret data.

Continuous non-malleability of existing constructions. The first construction of (one-shot) split-state non-malleable codes in the standard model was given by Liu and Lysyanskaya [23]. At a high-level the construction encrypts the input x with a leakage resilient encryption scheme and generates a non-interactive zero-knowledge proof of knowledge showing that (a) the public/secret key of the PKE are valid, and (b) the ciphertext is an encryption of x under the public key. Then, X_0 is set to the secret key while X_1 holds the corresponding public key, the ciphertext and the above described NIZK proof.

Unfortunately, it is rather easy to break the non-malleable code of Liu and Lysyanskaya in the continuous setting. Recall that our security notion of continuous non-malleable codes allows the adversary to interact in the following game. First, we sample a codeword $(X_0, X_1) \leftarrow \text{Encode}(x)$ and then repeat the following process a polynomial number of times:

1. The adversary submits two polynomial-time computable functions $(\mathsf{T}_0, \mathsf{T}_1)$ resulting in a tampered state $(X_0', X_1') = (\mathsf{T}_0(X_0), \mathsf{T}_1(X_1))$.
2. We consider three different cases: (1) if $(X_0', X_1') = (X_0, X_1)$ then return $\mathsf{same}^\star$; (2) otherwise compute $x' = \text{Decode}(X_0', X_1')$ and return x' if $x' \neq \bot$; (3) if $x' = \bot$ self-destruct and terminate the experiment.

The main observation that enables the attack against the scheme of [23] is as follows. For a fixed (but adversarially chosen) part X_0' it is easy to come-up with two corresponding parts X_1' and X_1'' such that both (X_0', X_1') and (X_0', X_1'') form a valid codeword that *does not* lead to a self-destruct. Suppose further that $\text{Decode}(X_0', X_1') \neq \text{Decode}(X_0', X_1'')$, then under continuous tampering the adversary may permanently replace the original encoding X_0 with X_0', while depending on whether the i-th bit of X_1 is 0 or 1 either replace X_1 by X_1' or X_1''. This allows to recover the entire X_1 by just $|X_1|$ tampering attacks. Once X_1 is known to the adversary it is easy to tamper with (X_0, X_1) in a way that depends on $\text{Decode}(X_0, X_1)$.

Somewhat surprisingly, our attack can be generalized to break *any* non-malleable code that is secure in the information theoretic setting. Hence, also the recent breakthrough results on information theoretic non-malleability [15, 1] fail to provide security under continuous attacks. Moreover, we emphasize that our attack does not only work for the code itself, but (in most cases) can be also applied to the tamper-protection application of cryptographic functionalities.

Uniqueness. The attack above exploits that for a fixed known part X_0' it is easy to come-up with two valid parts X_1', X_1''. For the encoding of [23] this is indeed easy to achieve. If the secret key X_0' is known it is easy to come-up with two valid parts X_1', X_1'': just encrypt two arbitrary messages $x_0 \neq x_1$ and generate the corresponding proofs. The above weakness motivates a new property that non-malleable codes shall satisfy in order to achieve security against continuous non-malleability. We call this property *uniqueness*, which informally guarantees that for any (adversarially chosen) valid encoding (X_0', X_1') it is computationally hard to come up with $X_b'' \neq X_b'$ such that (X_b', X_{1-b}'') forms a valid encoding. Clearly the uniqueness property prevents the above described attack, and hence is a crucial requirement for continuous non-malleability.

A new construction. In light of the above discussion, we need to build a non-malleable code that achieves our uniqueness property. Our construction uses as building blocks a leakage resilient storage (LRS) scheme [13, 14] for the split-state model (one may view this as a generalization of the leakage resilient PKE used in [23]), a collision-resistant hash function and (similar to [23]) an extractable NIZK. At a high-level we use the LRS to encode the secret message, hash the resulting shares using the hash function and generate a NIZK proof of knowledge that indeed the resulting hash values are correctly computed from the shares. While it is easy to show that collision resistance of the hash function guarantees the uniqueness property, a careful analysis is required to prove continuous non-malleability. We refer the reader to Section 4 for the details of our construction and to Section 4.1 for an outline of the proof.

Tamper resilience for stateless and stateful primitives. We can use our new construction of continuous non-malleable codes to protect arbitrary computation against continuous tampering attacks. In contrast to earlier works our construction does not need to re-encode the secret state after each usage, which besides being more efficient avoids the use of erasures. As discussed above, erasures are problematic in the tampering setting as one would essentially need to encode the entire state (possibly including large non-confidential data).

Additionally, our transformation does not need to keep any secret state. Hence, if our transformation is used for stateless primitives, then the resulting scheme remains stateless. This solves an open problem of Dziembowski, Pietrzak and Wichs [17]. Notice that while we do not need to keep any secret state, the transformed functionality requires one single bit to switch to self-destruction mode. This bit can be *public* but must be untamperable, and can for instance be implemented through one-time writable memory. As shown in the work of Gennaro et al. [20] continuous tamper resilience is impossible to achieve without such a mechanism for self-destruction.

Of course, our construction can also be used for stateful primitives, in which case our functionality will re-encode the new state during execution. Note that in this setting, as data is never erased, an adversary can always reset the functionality to a previous valid state. To avoid this, our transformation uses an untamperable *public* counter[2] that helps us to detect whenever the functionality is reset to a previous state, leading to a self-destruct. We notice that such an untamperable counter is necessary, as otherwise there is no way to protect against the above resetting attack.

Adding leakage. As a last contribution, we show that our code is also secure against bounded leakage attacks. This is similar to the works of [23, 15] who also consider bounded leakage resilience of their encoding scheme. We then show that bounded leakage resilience is also inherited by functionalities that are protected by our transformation. Notice that without perfect erasures bounded leakage resilience is the best we can achieve, as there is no hope for security if an encoding that is produced at some point in time is gradually revealed to the adversary.

1.2 Related Work

Constructions of non-malleable codes. Besides showing feasibility by a probabilistic argument, [17] also built non-malleable codes for bit-wise tampering and gave a construction in the split-state model using a random oracle. This result was followed by [9] which proposed non-malleable codes that are secure against block-wise tampering. The first construction of non-malleable codes in the split-state model was given by Liu and Lysyanskaya [23] assuming an untamperable CRS. Very recently two beautiful works showed how to build non-malleable codes in the split-state model without relying on a CRS [15, 1] even when the adversary has unlimited computing power. Dziembowski *et al.* [15] show how to encode a single bit using the inner product function. Agrawal *et al.* [1] developed a construction that goes beyond single-bit encoding but induces a huge overhead.

See also [8, 7, 18] for other recent advances on the construction of non-malleable codes. We also notice that the work of Genarro *et al.* [20] proposed a generic method that allows to protect arbitrary computation against continuous tampering attacks, without requiring erasures. We refer the reader to [17] for a more detailed comparison between non-malleable codes and the solution of [20].

Other works on tamper resilience. A large body of work shows how to protect specific cryptographic schemes against tampering attacks (see [4, 3, 22, 5, 24, 12] and many more). While these works consider a strong tampering model (e.g., they do not require the split-state assumption), they only offer security for specific schemes. In contrast non-malleable codes are generally applicable and can provide tamper resilience of any cryptographic scheme.

[2]Note that a counter uses very small (logarithmic in the security parameter) number of bits.

In all the above works, including ours, it is assumed that the circuitry that computes the cryptographic algorithm using the potentially tampered key runs correctly, and is not subject to tampering attacks. An important line of works analyze to what extent we can guarantee security when even the circuitry is prone to tampering attacks [21, 19, 11]. These works typically consider a restricted class of tampering attacks (e.g., individual bit tampering) and assume that large parts of the circuit (and memory) remain untampered.

2 Preliminaries

2.1 Notation

We let $\mathbb{N}$ be the set of naturals. For $n \in \mathbb{N}$, we write $[n] := \{1, \ldots, n\}$. Given a set $\mathcal{S}$, we write $s \leftarrow \mathcal{S}$ to denote that element s is sampled uniformly from $\mathcal{S}$. If S is an algorithm, $y \leftarrow \mathsf{S}(x)$ denotes an execution of S with input x and output y; if S is randomized, then y is a random variable.

Throughout the paper we denote the security parameter by $k \in \mathbb{N}$. A function $\delta(k)$ is called *negligible* in k (or simply negligible) if it vanishes faster than the inverse of any polynomial in k, i.e., $\delta(k) = k^{-\omega(1)}$. A machine S is called *probabilistic polynomial time* (PPT) if for any input $x \in \{0,1\}^*$ the computation of $\mathsf{S}(x)$ terminates in at most $poly(|x|)$ steps and S is probabilistic (i.e., it uses randomness as part of its logic).

Oracle $\mathcal{O}^\ell(s)$ is parametrized by a value s and takes as input functions L and outputs $\mathsf{L}(s)$, returning a total of at most ℓ bits.

2.2 Robust Non-Interactive Zero Knowledge

Given an $\mathbf{NP}$-relation, let $\mathcal{L} = \{x : \exists w \text{ such that } \mathcal{R}(x,w) = 1\}$ be the corresponding language. A robust non-interactive zero knowledge (NIZK) proof system for $\mathcal{L}$, is a tuple of algorithms $(\mathrm{G}_{\mathtt{NIZK}}, \mathrm{Prove}, \mathrm{Verify}, \mathrm{Sim} = (\mathrm{Sim}_1, \mathrm{Sim}_2), \mathrm{Xtr})$ such that the following properties hold [25].

Completeness. For all $x \in \mathcal{L}$ of length k and all w such that $\mathcal{R}(x,w) = 1$, for all $\Omega \leftarrow \mathrm{G}_{\mathtt{NIZK}}(1^k)$ we have that $\mathrm{Verify}(\Omega, x, \mathrm{Prove}(\Omega, w, x)) = \mathtt{accept}$

Multi-theorem zero knowledge. For all PPT adversaries A, we have $\mathsf{Real}(k) \approx \mathsf{Simu(k)}$, where $\mathsf{Real}(k)$ and $\mathsf{Simu(k)}$ are distributions defined via the following experiment:

$$\mathsf{Real}(k) = \left\{\Omega \leftarrow \mathrm{G}_{\mathtt{NIZK}}(1^k);\, out \leftarrow \mathsf{A}^{\mathrm{Prove}(\Omega,\cdot,\cdot)}(\Omega); \text{Output: } out.\right\}$$
$$\mathsf{Simu}(k) = \left\{(\Omega, tk) \leftarrow \mathrm{Sim}_1(1^k);\, out \leftarrow \mathsf{A}^{\mathrm{Sim}_2(\Omega,\cdot,tk)}(\Omega); \text{Output: } out.\right\}.$$

Extractability. There exists a PPT algorithm Xtr such that, for all PPT adversaries A, we have

$$\mathbb{P}\left[\begin{array}{c}(\Omega, tk, ek) \leftarrow \mathrm{Sim}_1(1^k); (x,\pi) \leftarrow \mathsf{A}^{\mathrm{Sim}_2(\Omega,\cdot,tk)}(\Omega); w \leftarrow \mathrm{Xtr}(\Omega,(x,\pi),ek);\\ \mathcal{R}(x,w) \neq 1 \wedge (x,\pi) \notin \mathcal{Q} \wedge \mathrm{Verify}(\Omega, x, \pi) = \mathtt{accept}\end{array}\right] \leq negl(k),$$

where the list $\mathcal{Q}$ contains the successful pairs (x_i, π_i) that A has queried to Sim_2.

Similarly to [23], we assume that different statements have different proofs, i.e., if $\mathrm{Verify}(\Omega, x, \pi) = \mathtt{accept}$ we have that $\mathrm{Verify}(\Omega, x', \pi) = \mathtt{reject}$ for all $x' \neq x$. This property can be achieved by appending the statement to its proof.

We also require that the proof system supports labels, so that the Prove, Verify, Sim and Xtr algorithms now also take a public label λ as input, and the completeness, zero knowledge and extractability properties are updated accordingly. (This can be easily achieved by appending the label λ to the statement x.) More precisely, we write $\mathrm{Prove}^\lambda(\Omega, w, x)$ and $\mathrm{Verify}^\lambda(\Omega, x, \pi)$ for the prover and the verifier, and $\mathrm{Sim}_2^\lambda(\Omega, x, tk)$ and $\mathrm{Xtr}^\lambda(\Omega, (x,\pi), ek)$ for the simulator and the extractor.

2.3 Leakage Resilient Storage

We recall the definition of leakage resilient storage from [13, 14]. A leakage resilient storage scheme $(\mathrm{LRS}, \mathrm{LRS}^{-1})$ is a pair of algorithms defined as follows. (1) Algorithm LRS takes as input a secret x and outputs an encoding (s_0, s_1) of x. (2) Algorithm LRS^{-1} takes as input shares (s_0, s_1) and outputs a message x'. Since the LRS that we use in this paper is secure against computationally unbounded adversaries, we state the definition below in the information theoretic setting. It is easy to extend it to also consider computationally bounded adversaries.

Definition 1 (LRS)**.** We call $(\mathrm{LRS}, \mathrm{LRS}^{-1})$ an ℓ-leakage resilient storage scheme (ℓ-LRS) if for all $\theta \in \{0,1\}$, all secrets x, y and all adversaries A it holds that

$$\left\{\mathsf{Leakage}_{\mathsf{A},x,\theta}(k)\right\}_{k\in\mathbb{N}} \approx_s \left\{\mathsf{Leakage}_{\mathsf{A},y,\theta}(k)\right\}_{k\in\mathbb{N}},$$

where

$$\mathsf{Leakage}_{\mathsf{A},x,\theta}(k) = \left\{ \ (s_0, s_1) \leftarrow \mathrm{LRS}(x);\ out_{\mathsf{A}} \leftarrow \mathsf{A}^{\mathcal{O}^\ell(s_0,\cdot),\mathcal{O}^\ell(s_1,\cdot)};\ \text{Output: } (s_\theta, out_{\mathsf{A}}). \ \right\}.$$

We remark that Definition 1 is stronger than the standard definition of LRS, in that the adversary is allowed to see one of the two shares after he is done with leakage queries. A careful analysis of the proof, however, shows that the LRS scheme of [14, Lemma 22] satisfies the above generalized notion since the inner product function is a strong randomness extractor [10].

3 Continuous Non-Malleability

We start by formally defining an encoding scheme in the common reference string (CRS) model.

Definition 2 (Split-state Encoding Scheme in the CRS Model)**.** A split-state encoding scheme in the common reference string (CRS) model is a tuple of algorithms $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ specified below.

- Init takes as input the security parameter and outputs a CRS $\Omega \leftarrow \mathrm{Init}(1^k)$.
- Encode takes as input some message $x \in \{0,1\}^k$ and the CRS Ω and outputs a codeword consisting of two parts (X_0, X_1) such that $X_0, X_1 \in \{0,1\}^n$.
- Decode takes as input a codeword $(X_0, X_1) \in \{0,1\}^{2n}$ and the CRS and outputs either a message $x' \in \{0,1\}^k$ or a special symbol $\perp$.

Consider the following oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1))$, which is parametrized by an encoding (X_0, X_1) and takes as input functions $\mathsf{T}_0, \mathsf{T}_1 : \{0,1\}^n \to \{0,1\}^n$.

$\underline{\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), (\mathsf{T}_0, \mathsf{T}_1))}$:
$(X'_0, X'_1) = (\mathsf{T}_0(X_0), \mathsf{T}_1(X_1))$
If $(X'_0, X'_1) = (X_0, X_1)$ return $\mathsf{same}^\star$
If $\mathrm{Decode}(\Omega, (X'_0, X'_1)) = \perp$, return $\perp$ and "self-destruct"
Else return (X'_0, X'_1).

By "self-destruct" we mean that once $\mathrm{Decode}(\Omega, (X'_0, X'_1))$ outputs $\perp$, the oracle will answer $\perp$ to any further query.

Definition 3 (Continuous Non-Malleability)**.** Let $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ be a split-state encoding scheme in the CRS model. We say that Code is q-continuously non-malleable ℓ-leakage resilient ((ℓ, q)-CNMLR for short), if for all messages $x, y \in \{0,1\}^k$ and all PPT adversaries A it holds that

$$\left\{\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}(k)\right\}_{k\in\mathbb{N}} \approx_c \left\{\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},y}(k)\right\}_{k\in\mathbb{N}}$$

where

$$\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}(k) = \left\{ \begin{array}{c} \Omega \leftarrow \mathrm{Init}(1^k); (X_0, X_1) \leftarrow \mathrm{Encode}(\Omega, x); \\ out_{\mathsf{A}} \leftarrow \mathsf{A}^{\mathcal{O}^\ell(X_0), \mathcal{O}^\ell(X_1), \mathcal{O}_{\mathsf{cnm}}((X_0,X_1))}; \text{Output: } out_{\mathsf{A}} \end{array} \right\}$$

and A asks a total of q queries to $\mathcal{O}_{\mathsf{cnm}}$.

Without loss of generality we assume that the variable out_A consists of all the bits leaked from X_0 and X_1 (in a vector $\mathbf{\Lambda}$) and all the outcomes from oracle $\mathcal{O}_{\mathsf{cnm}}(X_0, X_1)$ (in a vector $\mathbf{\Theta}$); we write this as $out_A = (\mathbf{\Lambda}, \mathbf{\Theta})$ where $|\mathbf{\Lambda}| \le 2\ell$ and $\mathbf{\Theta}$ has exactly q elements.

Intuitively, the above definition captures a setting where a fully adaptive adversary A tries to break non-malleability by tampering several times with a target encoding, obtaining each time some leakage from the decoding process. The only restriction is that whenever a tampering attempt decodes to $\perp$, the system "self-destructs".[3] Note that whenever the adversary mauls (X_0, X_1) to a valid encoding (X'_0, X'_1), oracle $\mathcal{O}_{\mathsf{cnm}}$ returns (X'_0, X'_1). This is different from [17, 23], where the experiment returns the output of the decoded message, i.e. $\mathrm{Decode}(\Omega, (X'_0, X'_1))$. The recent work of Faust et al. [18] consider a similar extension where also the codeword is returned instead of the decoded message and call it super strong non-malleability. Also, we remark that Definition 3 implies strong non-malleability (as defined in [17, 23]) if we restrict A to ask a single query (i.e., $q = 1$) to oracle $\mathcal{O}_{\mathsf{cnm}}$.[4] We choose the formulation above because it is stronger and at the same time achieved by our code!

3.1 Uniqueness

As we argue below, constructions that satisfy our new Definition 3 have to meet the following *uniqueness* requirement. Informally this means that for any (possibly adversarially chosen) side of an encoding X'_b it is computationally hard to find two corresponding sides X'_{1-b} and X''_{1-b} such that both (X'_b, X'_{1-b}) and (X'_b, X''_{1-b}) form a valid encoding.

Definition 4 (Uniqueness)**.** Let $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ be a split-state encoding in the CRS model. We say that Code satisfies *uniqueness* if for all PPT adversaries A and for all $b \in \{0,1\}$ we have:

$$\mathbb{P}\left[\begin{array}{c} \Omega \leftarrow \mathrm{Init}(1^k); (X'_b, X'_{1-b}, X''_{1-b}) \leftarrow \mathsf{A}(1^k, \Omega); X'_{1-b} \neq X''_{1-b}; \\ \mathrm{Decode}(\Omega, (X'_b, X'_{1-b})) \neq \perp; \mathrm{Decode}(\Omega, (X'_b, X''_{1-b})) \neq \perp \end{array} \right] \le negl(k).$$

The following attack shows that the uniqueness property is necessary to achieve Definition 3.

Lemma 1. *Let* Code *be* $(0, poly(k))$*-CNMLR. Then* Code *must satisfy uniqueness.*

Proof. For the sake of contradiction, assume that we can efficiently find a triple (X'_0, X'_1, X''_1) such that (X'_0, X'_1) and (X'_0, X''_1) are both valid and $X'_1 \neq X''_1$. For a target encoding (Y_0, Y_1), we describe an efficient algorithm recovering Y_1 with overwhelming probability, by asking $n = poly(k)$ queries to $\mathcal{O}_{\mathsf{cnm}}((Y_0, Y_1), \cdot)$.

[3] As described in [20] it is easy to see that without such a restriction non-malleability can indeed be broken, since A can simply recover the entire (X_0, X_1) after polynomially many queries.

[4] It is easy to see that encoding from [23] satisfies the stronger variant of strong non-malleability.

For all $i \in [n]$ repeat the following:
Prepare the i-th tampering function as follows:
- $\mathsf{T}_0^{(i)}(Y_0)$: Replace Y_0 by X_0';
- $\mathsf{T}_1^{(i)}(Y_1)$: If $Y_1[i] = 0$ replace Y_1 by X_1'; otherwise replace it by X_1''.
Submit $(\mathsf{T}_0^{(i)}, \mathsf{T}_1^{(i)})$ to $\mathcal{O}_{\mathsf{cnm}}((Y_0, Y_1), \cdot)$ and obtain (Y_0', Y_1').
If $(Y_0', Y_1') = (X_0', X_1')$, set $Z[i] \leftarrow 0$.
Otherwise, if $(Y_0', Y_1') = (X_0', X_1'')$, set $Z[i] \leftarrow 1$.
Output Z as the guess for Y_1.

The above algorithm clearly succeeds with overwhelming probability, whenever $X_1' \neq Y_1 \neq X_1''$.[5]

Once Y_1 is known, we ask one additional query $(\mathsf{T}_0^{(n+1)}, \mathsf{T}_1^{(n+1)})$ to $\mathcal{O}_{\mathsf{cnm}}((Y_0, Y_1), \cdot)$, as follows:

- $\mathsf{T}_0^{(n+1)}(Y_0)$ hard-wires Y_1 and computes $y \leftarrow \mathrm{Decode}(\Omega, (Y_0, Y_1))$; if the first bit of y is 0 then T_0 behaves like the identity function, otherwise it overwrites Y_0 with 0^n.
- $\mathsf{T}_1^{(n+1)}(Y_0)$ is the identity function.

The above clearly allows to learn one bit of the message in the target encoding and hence contradicts the fact that Code is $(0, poly(k))$-CNMLR. □

Attacking existing schemes. The above procedure can be applied to show that the encoding of [23] does not satisfy our notion. Recall that in [23] a message x is encoded as $X_0 = (pk, c := \mathrm{Enc}(pk, x), \pi)$ and $X_1 = sk$. Here, (pk, sk) is a valid key pair and π is a proof of knowledge of a pair (x, sk) such that c decrypts to x under sk *and* (pk, sk) forms a valid key-pair. Clearly, for some $X_1' = sk'$ it is easy to find two valid corresponding parts $X_0' \neq X_0''$ which violates uniqueness.

We mention two important extensions of the attack from Lemma 1, leading to even stronger security breaches:

1. In case the valid pair of encodings (X_0', X_1'), (X_0', X_1'') which violates the uniqueness property are such that $\mathrm{Decode}(\Omega, (X_0', X_1')) \neq \mathrm{Decode}(\Omega, (X_0', X_1''))$, one can show that Lemma 1 still holds in the weaker version of the Definition 3 in which the experiment does not output tampered encodings but only the corresponding decoded message. Note that this applies in particular to the encoding of [23].

2. In case it is possible to find *both* (X_0', X_1', X_1'') and (X_0', X_0'', X_1') violating uniqueness, a simple variant of the attack allows us to recover both halves of the target encoding which is a total breach of security! However, it is not clear for the scheme of [23] how to find two valid corresponding parts X_1', X_1'', because given pk' it shall of course be computationally hard to find two corresponding valid secret keys sk', sk''.

The above attack can be easily extended to the information theoretic setting to break the constructions of the non-malleable codes (in split-state) recently introduced in [15] and in [1]. In fact, in the following lemma we show that there does *not* exist any information theoretic secure CNMLR code.

Lemma 2. *It is impossible to construct information theoretically secure* $(0, poly(k))$*-CNMLR codes.*

Proof. We prove the lemma by contradiction. Assume that there exists an information theoretically secure $(0, poly(k))$-CNMLR code with $2n$ bits codewords. By Lemma 1, the code must satisfy the

[5] In case $(X_0', X_1') = (Y_0, Y_1)$ or $(X_0', X_1'') = (Y_0, Y_1)$, then the entire encoding can be recovered even with more ease. In this case, whenever the oracle returns $\mathsf{same}^\star$ we know $Y_0 = X_0'$ and $Y_1 \in \{X_1', X_1''\}$. In the next step we replace the encoding with (X_0', X_1'); if the oracle returns $\mathsf{same}^\star$ again, then we conclude that $Y_1 = X_1'$, otherwise we conclude $Y_1 = X_1''$.

uniqueness property. In the information theoretic setting this means that, for all codewords $(X_0, X_1) \in \{0,1\}^{2n}$ such that $\mathrm{Decode}(\Omega, (X_0, X_1)) \neq \perp$, the following holds: (i) for all $X_1' \in \{0,1\}^n$ such that $X_1' \neq X_1$, we have $\mathrm{Decode}(\Omega, (X_0, X_1')) = \perp$; (ii) for all $X_0' \in \{0,1\}^n$, such that $X_0' \neq X_0$, we have $\mathrm{Decode}(\Omega, (X_0', X_1)) = \perp$.

Given a target encoding (X_0, X_1) of some secret x, an unbounded A can define the following tampering function T_b (for $b \in \{0,1\}$): Given X_b as input, try all possible $X_{1-b} \in \{0,1\}^n$ until $\mathrm{Decode}(\Omega, (X_0, X_1)) \neq \perp$. By property (i)-(ii) above, we conclude that for all $X_{1-b}' \neq X_{1-b}$, the decoding algorithm $\mathrm{Decode}(\Omega, (X_b, X_{1-b}'))$ outputs $\perp$ with overwhelming probability. Thus, T_b can recover $x = \mathrm{Decode}(\Omega, (X_b, X_{1-b}))$ and if the first bit of the decoded value is 0 leave the target encoding unchanged, otherwise $(\mathsf{T}_0, \mathsf{T}_1)$ modifies the encoding with an invalid codeword. The above clearly allows to learn one bit of the message in the target encoding, and hence contradicts the fact that the code is $(0, \mathit{poly}(k))$-CNMLR. □

Note that the attack of Lemma 2 requires the tampering function to be unbounded. In case when the tampering functions are computationally bounded and only the adversary is computationally unbounded we do not know how to make the above attack work.

4 The Code

Consider the following split-state encoding scheme in the CRS model $(\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$, based on an LRS scheme $(\mathrm{LRS}, \mathrm{LRS}^{-1})$, on a family of collision resistant hash functions $\mathcal{H} = \{H_t : \{0,1\}^{\mathit{poly}(k)} \rightarrow \{0,1\}^k\}_{t \in \{0,1\}^k}$ and on a robust non-interactive zero knowledge proof system $(\mathrm{G}_{\mathtt{NIZK}}, \mathrm{Prove}, \mathrm{Verify})$ which supports labels, for language $\mathcal{L}_{\mathcal{H},t} = \{h : \exists s \text{ such that } h = H_t(s)\}$.

$\mathrm{Init}(1^k)$. Sample $t \leftarrow \{0,1\}^k$ and run $\Omega \leftarrow \mathrm{G}_{\mathtt{NIZK}}(1^k)$.

$\mathrm{Encode}(\Omega, x)$. Let $(s_0, s_1) \leftarrow \mathrm{LRS}(x)$. Compute $h_0 = H_t(s_0)$, $h_1 = H_t(s_1)$ and $\pi_0 \leftarrow \mathrm{Prove}^{\lambda_1}(\Omega, s_0, h_0)$, $\pi_1 \leftarrow \mathrm{Prove}^{\lambda_0}(\Omega, s_1, h_1)$, where the labels are defined as $\lambda_0 = h_0$, $\lambda_1 = h_1$. (Note that the pre-image of h_b is s_b and the proof π_b is computed for statement h_b using label h_{1-b}.) Output $(X_0, X_1) = ((s_0, h_1, \pi_1, \pi_0), (s_1, h_0, \pi_0, \pi_1))$.

$\mathrm{Decode}(\Omega, (X_0, X_1))$. The decoding parses X_b as $(s_b, h_{1-b}, \pi_{1-b}, \pi_b)$, computes $\lambda_b = H_t(s_b)$ and then proceeds as follows:

(a) *Local check.* If $\mathrm{Verify}^{\lambda_1}(\Omega, h_0, \pi_0)$ or $\mathrm{Verify}^{\lambda_0}(\Omega, h_1, \pi_1)$ output `reject` in any of the two sides X_0, X_1, return $x' = \perp$.

(b) *Cross check.* If (i) $h_0 \neq H_t(s_0)$ or $h_1 \neq H_t(s_1)$, *or* (ii) the proofs (π_0, π_1) in X_0 are different from the ones in X_1, then return $x' = \perp$.

(c) *Decoding.* Otherwise, return $x' = \mathrm{LRS}^{-1}(s_0, s_1)$.

We start by showing that the above code satisfies the uniqueness property (cf. Definition 4).

Lemma 3. *Let* $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ *be as above. Then, if* $\mathcal{H}$ *is a family of collision resistant hash functions* Code *satisfies uniqueness.*

Proof. We show that Definition 4 is satisfied for $b = 0$. The proof for $b = 1$ is identical and is therefore omitted.

Assume that there exists a PPT adversary A that, given as input $\Omega \leftarrow \mathrm{Init}(1^k)$, is able to produce (X_0', X_1', X_1'') such that both (X_0', X_1') and (X_0', X_1'') are valid, but $X_1' \neq X_1''$. Let $X_0' = (s_0', h_1', \pi_1', \pi_0')$, $X_1' = (s_1', h_0', \pi_0', \pi_1')$ and $X_1'' = (s_1'', h_0'', \pi_0'', \pi_1'')$.

Since s'_0 is the same in both encodings, we must have $h'_0 = h''_0$ as the hash function is deterministic. Furthermore, since both (X'_0, X'_1) and (X'_0, X''_1) are valid, the proofs must verify successfully and therefore we must have $\pi'_0 = \pi''_0$ and $\pi'_1 = \pi''_1$. It follows that $X''_1 = (s''_1, h'_0, \pi'_0, \pi'_1)$, such that $s''_1 \neq s'_1$. Clearly (s'_1, s''_1) is a collision for h'_1, a contradiction. □

Theorem 1. *Let* $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ *be as above. Assume that* $(\mathrm{LRS}, \mathrm{LRS}^{-1})$ *is an* ℓ'*-LRS,* $\mathcal{H} = \{H_t : \{0,1\}^{poly(k)} \to \{0,1\}^k\}_{t \in \{0,1\}^k}$ *is a family of collision resistant hash functions and* $(\mathrm{G}_{\mathtt{NIZK}}, \mathrm{Prove}, \mathrm{Verify})$ *is a robust NIZK proof system for language* $\mathcal{L}_{\mathcal{H},t}$*. Then* Code *is* (ℓ, q)*-CNMLR, for any* $q = poly(k)$ *and* $\ell' \geq \max\{2\ell + (k+1)\lceil \log(q) \rceil, 2k+1\}$.

4.1 Outline of the Proof

In order to build some intuition, let us first explain why a few natural attacks do not work. Clearly, the uniqueness property (cf. Lemma 3) rules out the attack of Lemma 1. As a first attempt, the adversary could try to modify the proof π_0 to a different proof π'_0, by using the fact that X_0 contains the corresponding witness s_0 and the correct label h_1. However, to ensure the validity of (X'_0, X'_1), this would require to place π'_0 in X'_1, which should be hard without knowing a witness (by the robustness of the proof system). Alternatively, one could try to maul the two halves (s_0, s_1) of the LRS scheme, into a pair (s'_0, s'_1) encoding a related message.[6] This requires, for instance, to change the proof π_0 into π'_0 and place π'_0 in X'_1, which again should be hard without knowing a witness and the correct label.

Let us now try to give a high-level overview of the proof. Given a polynomial time distinguisher D that violates continuous non-malleability of Code, we build another polynomial time distinguisher D' which breaks leakage resilience of $(\mathrm{LRS}, \mathrm{LRS}^{-1})$. Distinguisher D', which can access oracles $\mathcal{O}^{\ell'}(s_0)$ and $\mathcal{O}^{\ell'}(s_1)$, has to distinguish whether (s_0, s_1) is the encoding of message x or message y and will do so with the help of D's advantage in distinguishing $\mathsf{Tamper}_x^{\mathrm{cnmlr}}$ from $\mathsf{Tamper}_y^{\mathrm{cnmlr}}$. The main difficulty in the reduction is how D' can simulate the answers from the tampering oracle $\mathcal{O}_{\mathsf{cnm}}$ (cf. Definition 3), without knowing the target encoding (X_0, X_1). This is the main point where our techniques diverge significantly from [23] (as in [23] the reduction "knows" a complete half of the encoding). In our case, in fact, D' can only access the two halves X_0 and X_1 "inside" the oracles $\mathcal{O}^{\ell'}(s_0)$ and $\mathcal{O}^{\ell'}(s_1)$.[7] However, it is not clear how this helps answering tampering queries, as the latter requires access to *both* X_0 and X_1 for decoding the tampered message, whereas the reduction can only access X_0 and X_1 *separately*.

For ease of description, in what follows we simply assume that D' can access directly $\mathcal{O}^{\ell'}(X_0)$ and $\mathcal{O}^{\ell'}(X_1)$. Furthermore, let us assume that D can only issue tampering queries (we discuss how to additionally handle leakage briefly at the end of the outline). Like any standard reduction, D' samples some randomness r and fixes the random tape of D to r. Our novel strategy is to construct a polynomial time algorithm $\mathsf{F}(r)$ that, given access to $\mathcal{O}^{\ell'}(X_0)$, $\mathcal{O}^{\ell'}(X_1)$, outputs the smallest index j^* which indicates the round where $\mathsf{D}(r)$ provokes a self-destruct in $\mathsf{Tamper}_*^{\mathrm{cnmlr}}$. Before explaining how the actual algorithm works, let us explain how D' can complete the reduction using such a self-destruct finder F. At the beginning, it runs $\mathsf{F}(r)$ in order to leak the index j^*. At this point D' is done with leakage queries and asks to get X_0 (i.e., it chooses $\theta = 0$ in Definition 1).[8] Given X_0, distinguisher D' runs $\mathsf{D}(r)$ (with the *same* random coins r used for F). Hence, for all $1 \leq j < j^*$, upon input the j-th tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$, distinguisher D' lets $X'_0 = \mathsf{T}_0^{(j)}(X_0) = (s'_0, h'_1, \pi'_1, \pi'_0)$ and answers as follows:

1. In case $X'_0 = X_0$, output $\mathsf{same}^\star$ (cf. type A queries in Figure 1).

[6] When the LRS is implemented using the inner product extractor this is indeed possible, as argued in [15].

[7] Looking ahead, this can be achieved by first leaking the hash values h_0, h_1 of s_0, s_1, simulating the proofs π_0, π_1, and then hard-wiring these values into all leakage queries.

[8] Recall that this is a simplification, as by choosing $\theta = 0$ the distinguisher will obtain s_0. See also footnote 7.

2. In case $X_0' \neq X_0$ and either of the proofs in X_0' does not verify correctly, output $\perp$ (cf. type B queries in Figure 1).

3. In case $X_0' \neq X_0$ and both the proofs in X_0' verify correctly, check if $\pi_1' = \pi_1$; if 'yes' (in which case there is no hope to extract from π_1') then output $\perp$ (cf. type C queries in Figure 1).

4. Otherwise, attempt to extract s_1' from π_1', define $X_1' = (s_1', h_0', \pi_0', \pi_1')$ and output (X_0', X_1') (cf. type D queries in Figure 1).

Note that from round j^* on, all queries can be answered with $\perp$, and this is a correct simulation as $\mathsf{D}(r)$ provokes a self-destruct at round j^* in the real experiment.

In the proof of Theorem 1, we show that the above strategy is sound. with overwhelming probability over the choice of r the output produced by the above simulation is equal to the output that $\mathsf{D}(r)$ would have seen in the real experiment *until a self-destruct occurs* (cf. Lemma 4).[9]

Let us give some intuition why the above simulation is indeed sound. For type A queries, note that when $X_0' = X_0$ we must have $X_1' = X_1$ with overwhelming probability, as otherwise (X_0, X_1, X_1') would violate uniqueness. In case of type B queries, the decoding process in the real experiment would output $\perp$, so D' does a perfect simulation. The case of type C queries is a bit more delicate. In this case we use the facts that (i) in the NIZK proof system we use, different statements must have different proofs and (ii) the hash function is collision resistant, to show that X_0' must be of the form $X_0' = (s_0, h_1, \pi_1, \pi_0')$ and $\pi_0' \neq \pi_1$. A careful analysis shows that the latter contradicts leakage resilience of the underlying LRS scheme. Finally, for type D queries, note that whenever D' extracts a witness from a valid proof $\pi_1' \neq \pi_1$, the witness must be valid with overwhelming probability (as the NIZK is simulation extractable).

Next, let us explain how to construct the algorithm F. Roughly, $\mathsf{F}(r)$ runs $\mathsf{D}(r)$ "inside" the oracles $\mathcal{O}^{\ell'}(X_0)$, $\mathcal{O}^{\ell'}(X_1)$ as part of the leakage functions, and simulates the answers for $\mathsf{D}(r)$'s tampering queries using only one side of the target encoding, in the exact same way as outlined in (1)-(4) above. Let $\mathbf{\Theta}_b$, for $b \in \{0, 1\}$, denote the output simulated by F inside $\mathcal{O}^{\ell'}(X_b)$. To locate the self-destruct index j^*, we rely on the following property (cf. Lemma 5): the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ contain identical values until coordinate $j^* - 1$, but $\mathbf{\Theta}_0[j^*] \neq \mathbf{\Theta}_1[j^*]$ with overwhelming probability (over the choice of r). This implies that j^* can be computed as the first coordinate where $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are different. Hence, F can obtain the self-destruct index by using its adaptive access to oracles $\mathcal{O}^{\ell'}(X_0)$, $\mathcal{O}^{\ell'}(X_1)$ and apply a standard binary search algorithm to $\mathbf{\Theta}_0$, $\mathbf{\Theta}_1$. Note that the latter requires at most a logarithmic number of bits of adaptive leakage.

One technical problem is that F, in order to run $\mathsf{D}(r)$ inside of, say $\mathcal{O}^{\ell'}(X_0)$, and compute $\mathbf{\Theta}_0$, has also to answer leakage queries from $\mathsf{D}(r)$. Clearly, all leakages from X_0 can be easily computed, however it is not clear how to simulate leakages from X_1 (as we cannot access $\mathcal{O}^{\ell'}(X_1)$ inside $\mathcal{O}^{\ell'}(X_0)$). Fortunately, the latter issue can be avoided by letting F query $\mathcal{O}^{\ell'}(X_0)$ and $\mathcal{O}^{\ell'}(X_1)$ alternately, and aborting the execution of $\mathsf{D}(r)$ whenever it is not possible to answer a leakage query. It is not hard to show that after at most ℓ steps all leakages will be known, and F can run $\mathsf{D}(r)$ inside $\mathcal{O}^{\ell'}(X_0)$ without having access to $\mathcal{O}^{\ell'}(X_1)$. (All this comes at the price of some loss in the leakage bound, but, as we show in the proof, not too much.)

4.2 Proof of Theorem 1

Assume there exists an adversary A, a pair of messages (x, y), some non-negligible ε and a distinguisher D such that

$$\left|\mathbb{P}\left[\mathsf{D}(\Omega, \mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}) = 1\right] - \mathbb{P}\left[\mathsf{D}(\Omega, \mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},y}) = 1\right]\right| > \varepsilon.$$

[9]It is crucial that both the real and simulated experiments are run with the same r.

We will build a distinguisher D' against the underlying LRS scheme. Distinguisher D' will use D, A, (x, y) to construct an adversary A' and some value $\theta \in \{0, 1\}$ such that

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathsf{Leakage}_{\mathsf{A}',x,\theta}) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathsf{Leakage}_{\mathsf{A}',y,\theta}) = 1\right]\right| > \varepsilon'$$

for some non-negligible ε' (a contradiction).

Without loss of generality, we assume that D outputs its guess after a self-destruct takes place. In fact, in case A does not provoke a self-destruct, we can always modify A such that it asks an additional query to $\mathcal{O}_{\mathsf{cnm}}$ generating a self-destruct before outputting its guess; clearly this does not decrease D's advantage. We will construct D' in several steps. As a first step, we describe a mental experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$ (for a value $b \in \{0, 1\}$) where a simulator S (depending on A) simulates the view of experiment $\mathsf{Tamper}_{\mathsf{A}}$ given only access to X_b and a leakage oracle for X_{1-b} (here (X_0, X_1) is the target encoding). We will then explain how to mimic S by using A and access to $\mathcal{O}^{\ell}(X_0)$, $\mathcal{O}^{\ell}(X_1)$. Finally we show that S can be used by D' to locate the index j^* where a self-destruct happens in $\mathsf{Tamper}_{\mathsf{A}}$. The value of j^* will then be used in the reduction to the security of the leakage-resilient storage scheme. (See also Section 4.1 for a more detailed outline of the proof.)

Note that we can assume without loss of generality that D distinguishes the two distributions of experiment $\mathsf{Tamper}_{\mathsf{A}}$ even in case we replace real proofs with simulated proofs. Consider indeed a modified experiment $\mathsf{Tamper}'_{\mathsf{A}}$ which is identical to $\mathsf{Tamper}_{\mathsf{A}}$, but where the $\mathrm{G}_{\mathtt{NIZK}}$ algorithm is replaced by $(\Omega, tk, ek) \leftarrow \mathrm{Sim}_1(1^k)$ and a proof π is computed by running $\mathrm{Sim}_2^{\lambda}(\Omega, \cdot, tk)$. If D distinguishes $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},x}$ and $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A},y}$ but does not not distinguish $\mathsf{Tamper}'_{\mathsf{A},x}$ and $\mathsf{Tamper}'_{\mathsf{A},y}$, then one can use D, A, x and y to break the non-interactive zero knowledge property of the proof system with non-negligible advantage. By a standard argument, this implies $|\mathbb{P}\left[\mathsf{D}(\Omega, \mathsf{Tamper}'_{\mathsf{A},x}) = 1\right] - \mathbb{P}\left[\mathsf{D}(\Omega, \mathsf{Tamper}'_{\mathsf{A},y}) = 1\right]| > \varepsilon/2$.

A mental experiment. Let $q = poly(k)$ be the number of queries A asks to $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$; here (X_0, X_1) is the target encoding of some value x (to be specified later). We start by considering a mental experiment where a simulator S (running A) is given one complete side X_b of the encoding (for some $b \in \{0, 1\}$), and has oracle access to $\mathcal{O}^{\ell}(X_{1-b})$. The simulator outputs vectors $(\mathbf{\Lambda}_b, \mathbf{\Theta}_b)$ and is described in Figure 1.

For a message x and a bit $b \in \{0, 1\}$, consider the following experiment featuring the above PPT simulator S:

$$\mathsf{Tamper}''_{\mathsf{S},x}(k, b) = \left\{ \begin{array}{c} \Omega \leftarrow \mathrm{Init}(1^k); (X_0, X_1) \leftarrow \mathrm{Encode}'(\Omega, x); \\ \mathsf{out}_{\mathsf{S},b} \leftarrow \mathsf{S}^{\mathcal{O}^{\ell}(X_{1-b})}(X_b); \text{Output: } \mathsf{out}_{\mathsf{S},b} \end{array} \right\}.$$

Algorithm Encode' is identical to Encode, but the NIZKs are replaced with simulated NIZKs (as in experiment $\mathsf{Tamper}'_{\mathsf{A}}$). We will compare an execution of experiment $\mathsf{Tamper}'_{\mathsf{A}}$ with an execution of experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$, where both A and S run with the same random tape r and attack the same target encoding (X_0, X_1) of x. Denote with

$$\mathsf{view}_{\mathsf{A}} = ((\mathbf{\Lambda}[0], \mathbf{\Theta}[0]), \ldots, (\mathbf{\Lambda}[q], \mathbf{\Theta}[q]))$$

the view of adversary $\mathsf{A}(r)$ (i.e., A with random tape r) tampering with encoding (X_0, X_1) in $\mathsf{Tamper}'_{\mathsf{A}}$. We write $\mathsf{view}^{(j)}_{\mathsf{A}}$ for the view until round j.

Similarly, let[10]

$$\mathsf{view}_{\mathsf{S},b} = ((\mathbf{\Lambda}_b[0], \mathbf{\Theta}_b[0]), \ldots, (\mathbf{\Lambda}_b[q], \mathbf{\Theta}_b[q]))$$

be the view of produced by $\mathsf{S}(X_b; r)$ (i.e. S running A with random tape r) in $\mathsf{Tamper}''_{\mathsf{S}}(b)$. The first lemma says that, with overwhelming probability over the choice of r and the coin tosses to sample the encoding, the vectors $\mathsf{view}_{\mathsf{A}}$ and $\mathsf{view}_{\mathsf{S}}$ are identical *before a self-destruct happens*.

[10]Note that A can leak at most ℓ bits, so some of the values $\mathbf{\Lambda}[j]$ and $\mathbf{\Lambda}_b[j]$ are empty.

Simulator $\mathsf{S}^{\mathcal{O}^\ell(X_{1-b})}(X_b; r)$

For a value $b \in \{0,1\}$ and a target encoding (X_0, X_1), the algorithm has access to X_b and $\mathcal{O}^\ell(X_{1-b})$, and is defined as follows:

- Define (initially empty) vectors $\mathbf{\Theta}_b$, $\mathbf{\Lambda}_b$ and initialize $\mathsf{A}(r)$.
- Repeat the following for all $j = 1, \ldots, q$:
  1. Receive $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$ from $\mathsf{A}(r)$ (if any). Compute $\Lambda_b^{(j)} = \mathsf{L}_b(X_b)$ and forward L_{1-b} to $\mathcal{O}^\ell(X_{1-b})$ receiving back a value $\Lambda_{1-b}^{(j)}$. Set $\mathbf{\Lambda}_b[j] := (\Lambda_0^{(j)}, \Lambda_1^{(j)})$ and give $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$ to $\mathsf{A}(r)$.
  2. Receive $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ from $\mathsf{A}(r)$ and let $X_b' = \mathsf{T}_b^{(j)}(X_b) = (s_b', h_{1-b}', \pi_{1-b}', \pi_b')$.
  3. Set the value $\mathbf{\Theta}_b[j]$ as follows:
     (a) (Type A query.) If $X_b = X_b'$ define $\mathbf{\Theta}_b[j] := \mathsf{same}^\star$ and return $\mathsf{same}^\star$ to $\mathsf{A}(r)$.
     (b) (Type B query.) Else if $X_b \neq X_b'$ but the local check on X_b' fails (i.e., if $\mathrm{Verify}^{h_1'}(\Omega, h_0', \pi_0')$ or $\mathrm{Verify}^{h_0'}(\Omega, h_1', \pi_1')$ output `reject`), define $\mathbf{\Theta}_b[j] := \perp_b$ (where $\perp_0, \perp_1$ are special symbols such that $\perp_0 \neq \perp_1$) and return $\perp$ to $\mathsf{A}(r)$.
     (c) (Type C query.) Else if $\pi_{1-b}' = \pi_{1-b}$ define $\mathbf{\Theta}_b[j] := \perp_b$ and return $\perp$ to $\mathsf{A}(r)$.
     (d) (Type D query.) Else run $s_{1-b}' \leftarrow \mathrm{Xtr}^{h_b'}(\Omega, (h_{1-b}', \pi_{1-b}'), ek)$ and define $\mathbf{\Theta}_b[j] := (X_0', X_1')$ where $X_b' = (s_b', h_{1-b}', \pi_{1-b}', \pi_b')$, $X_{1-b}' = (s_{1-b}', h_b', \pi_b', \pi_{1-b}')$; return (X_0', X_1') to $\mathsf{A}(r)$.
- Output $out_{\mathsf{S},b} = (\mathbf{\Lambda}_b, \mathbf{\Theta}_b)$.

Figure 1: The simulator S of experiment $\mathsf{Tamper}_{\mathsf{S}}''(b)$

Lemma 4. *For all x and all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that A generates a self-destruct in $\mathsf{Tamper}_{\mathsf{A},x}'$. Denote with $\mathsf{view}_{\mathsf{A}}$ (resp. $\mathsf{view}_{\mathsf{S},b}$) the view of $\mathsf{A}(r)$ (resp. $\mathsf{S}(r)$) running in experiment $\mathsf{Tamper}_{\mathsf{A},x}'$ (resp. $\mathsf{Tamper}_{\mathsf{S},x}''(b)$) with target encoding (X_0, X_1). Then, for all $b \in \{0,1\}$, we have that $\mathsf{view}_{\mathsf{A}}^{(j^*-1)} = \mathsf{view}_{\mathsf{S},b}^{(j^*-1)}$ with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding.*

The second lemma says that the values contained in the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ produced by $\mathsf{S}(r)$ in a run of $\mathsf{Tamper}_{\mathsf{S}}''(b)$ with encoding (X_0, X_1) are identical until the coordinate corresponding to the self-destruct index (with overwhelming probability).

Lemma 5. *For all x and for all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that $\mathbf{\Theta}[j^*] = \perp$ in $\mathsf{Tamper}_{\mathsf{A},x}'(k)$. Denote with $\mathsf{view}_{\mathsf{S},b} = (\mathbf{\Lambda}_b, \mathbf{\Theta}_b)$ the view of $\mathsf{S}(r)$ running in experiment $\mathsf{Tamper}_{\mathsf{S},x}''(b)$ with target encoding (X_0, X_1). Then, the following holds with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding:*

(i) $\mathbf{\Theta}_0[j] = \mathbf{\Theta}_1[j]$, $\forall 1 \leq j \leq j^* - 1$.

(ii) $\mathbf{\Theta}_0[j^*] \neq \mathbf{\Theta}_1[j^*]$.

Locating the self-destruct point. Consider now the experiment $\mathsf{Tamper}_{\mathsf{A}}'$ and let (X_0, X_1) be the encoding that is computed at the beginning of the experiment. We define an algorithm $\mathsf{F}^{\mathcal{O}^\ell(X_0),\mathcal{O}^\ell(X_1)}(r)$ which is given access to oracles $\mathcal{O}^\ell(X_0), \mathcal{O}^\ell(X_1)$ and finds the index j^* where $\mathsf{A}(r)$ provokes a self-destruct in $\mathsf{Tamper}_{\mathsf{A}}'$. The main idea will be to compute the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ inside the leakage oracles (by running the simulator S of Figure 1), and then leak the first position where the vectors $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are different. (By Lemma 5, this is the right index with overwhelming probability.) Note that the latter can be done with logarithmic amount of adaptive leakage, e.g. by using a standard binary search algorithm, *once $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are defined.*

Algorithm $\mathsf{F}^{\mathcal{O}^\ell(X_0),\mathcal{O}^\ell(X_1)}(r)$

For a target encoding (X_0, X_1), the algorithm has access to oracles $\mathcal{O}^\ell(X_0)$, $\mathcal{O}^\ell(X_1)$ and is defined as follows:

1. **Setup:** Let $\mathcal{H}$ be a set of collision resistant hash functions. Set $j^* := 0$ and $\mathbf{\Lambda}_0, \mathbf{\Lambda}_1 = \emptyset$.
2. **Prepare the leakage to obtain $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$:** Define the following leakage query $\mathsf{L}_b(X_b, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$, with hard-wired values $(\mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$:
   (a) Attempt to run $\mathsf{S}^{\mathcal{O}^\ell(X_{1-b})}(X_b; r)$.
   (b) Answer all leakage queries on X_{1-b} by using the next value in $\mathbf{\Lambda}_{1-b}$.
   (c) In case no such value is found, terminate and output "`not done`" together with any value S leaked on X_b which is not already contained in $\mathbf{\Lambda}_b$. Update the vector $\mathbf{\Lambda}_b$ accordingly.
   (d) In case all leakage queries on X_{1-b} can be answered (i.e. S was run successfully), output "`done`".
3. **Loop to obtain $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$:** Query alternately $\mathcal{O}^\ell(X_0)$ with $\mathsf{L}_0(X_0, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$ and $\mathcal{O}^\ell(X_1)$ with $\mathsf{L}_1(X_1, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$ until they both output "`done`".
4. **Prepare the leakage to obtain j^*:** Sample a hash function $H_t \leftarrow \mathcal{H}$. Define the following leakage query $\mathsf{L}_b(X_b, i_{\max}, i_{\min})$ depending on two (hard-wired) indexes $i_{\max}, i_{\min} \in [q]$:
   (a) Run $\mathsf{S}^{\mathcal{O}^\ell(X_{1-b})}(X_b; r)$. (Note that this can be done without having access to $\mathcal{O}^\ell(X_{1-b})$, as all the leakages on X_{1-b} are already contained in $\mathbf{\Lambda}_{1-b}$). Let $\mathbf{\Theta}_b$ be the vector defined by S.
   (b) Return $H_t(\mathbf{\Theta}_b[i_{\min}], \ldots, \mathbf{\Theta}_b[i_{\max}])$.
5. **Loop for the binary search:** Set $i_{\max} = 1, i_{\min} = q$ and repeat the following until $i_{\max} = i_{\min}$:
   (a) Let $i_{\mathrm{mid}} := \lfloor \frac{i_{\min}+i_{\max}}{2} \rfloor$.
   (b) Forward $\mathsf{L}_0(X_0, i_{\mathrm{mid}}, i_{\min})$ to $\mathcal{O}^\ell(X_0)$ and $\mathsf{L}_1(X_1, i_{\mathrm{mid}}, i_{\min})$ to $\mathcal{O}^\ell(X_1)$; denote with Λ_0^*, Λ_1^* the answers from the oracles.
   (c) If $\Lambda_0^* \neq \Lambda_1^*$ set $i_{\max} := i_{\mathrm{mid}} - 1$, otherwise set $i_{\min} = i_{\mathrm{mid}} + 1$.
6. **Output:** Define $j^* := i_{\max} = i_{\min}$; return $(j^*, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$.

Figure 2: The algorithm F locating the self-destruct index j^*

A technical difficulty is that we cannot run the simulator (say) on X_0 because S needs to access $\mathcal{O}^\ell(X_1)$, but we do not have access to such oracle inside the leakage function. To solve this problem, F will attempt to run S alternately on X_0, X_1 and stop every time it encounters a leakage query it cannot answer. (Looking ahead this comes at the price of some loss in the leakage bound but, as we show below, the loss is no more than 2ℓ bits.) After a finite number of steps all the leakages will be known and S can be run until the end on both sides. At this point, F can perform the binary search (again using adaptive leakage queries to $\mathcal{O}^\ell(X_0)$, $\mathcal{O}^\ell(X_1)$) in order to learn j^*. A formal description of F is given in Figure 2. We prove the following property for F.

Lemma 6. *For all x and for all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that $\mathbf{\Theta}[j^*] = \perp$ in $\mathsf{Tamper}'_{\mathsf{A},x}(k)$. Then, algorithm F outputs j^* with overwhelming probability. Moreover F runs in polynomial time and requires a total of $4\ell + 2k\lceil\log(q)\rceil$ bits of leakage.*

Proof. The fact that F outputs the correct j^* follows from Lemma 4 and Lemma 5. Indeed, assuming that F terminates, we have that the views

$$\mathsf{view}_{\mathsf{S},b}^{(j^*-1)} = (\mathbf{\Lambda}_b, \mathbf{\Theta}_b[1], \ldots, \mathbf{\Theta}_b[j^*-1]) \qquad \text{and} \qquad \mathsf{view}_{\mathsf{A}}^{(j^*-1)} = (\mathbf{\Lambda}, \mathbf{\Theta}[1], \ldots, \mathbf{\Theta}[j^*-1])$$

(cf. step 4a in the description of F), are identical with overwhelming probability (by Lemma 4). This implies that j^* can be computed as the first entry where $\mathbf{\Theta}_0$ and $\mathbf{\Theta}_1$ are different (by Lemma 5), which is exactly what the algorithm does (cf. step 5a—5c in the description of F) exploiting the fact that the hash function is collision resistant.

It remains to show that F runs in polynomial time. This follows by inspection of the description of F in Figure 2 as:

1. The loop of step 3 ends after at most 2ℓ steps (since S cannot ask more than ℓ adaptive leakage queries on each side).
2. The loop of step 5 is a standard binary search algorithm which requires a logarithmic number of steps.

Notice that F requires at most 4ℓ bits of leakage (2ℓ bits from the oracles $\mathcal{O}^{\ell}(X_0)$ and $\mathcal{O}^{\ell}(X_1)$ and at most 2ℓ more bits for each of the times S returns "not done") plus $2k\lceil\log(q)\rceil$ bits for the binary search as the range of the hash function is k bits.

□

Description of the reduction. We construct D' and A' that for infinitely many k break the security of the LRS with non-negligible advantage. To this end, they are given access to D and A that win in game $\mathsf{Tamper}'_{\mathsf{A}}$ with non-negligible advantage (recall that in $\mathsf{Tamper}'_{\mathsf{A}}$ we have replaced the NIZKs with simulated NIZKs). First, A' samples randomness r and runs the sub-routine F of Figure 2 with this randomness. By Lemma 6 the algorithm F requires at most $4\ell + 2k\lceil\log(q)\rceil$ bits of leakage to learn the position j^* of the query where $\mathsf{A}(r)$ provokes the self-destruct. Then, A' runs A as subroutine using the same random coins r that were used to identify the self-destruct position. We emphasize that using the same r is crucial for the reduction to work. A formal description of the reduction follows:

1. **Simulate initial encoding:** Sample uniformly at random a key $t \leftarrow \{0,1\}^k$ for the hash function and run $(\Omega, tk, ek) \leftarrow \mathrm{Sim}_1(1^k)$ to generate a CRS and the corresponding simulation trapdoor tk. With access to the target leakage oracle $(\mathcal{O}^{\ell'}(s_0), \mathcal{O}^{\ell'}(s_1))$ obtain $h_0 = H_t(s_0)$ and $h_1 = H_t(s_1)$; set the labels $\lambda_0 = h_0$ and $\lambda_1 = h_1$. Notice that given (h_b, λ_{1-b}) it is easy to simulate π_b, i.e., we can compute $\pi_b \leftarrow \mathrm{Sim}_2^{\lambda_{1-b}}(\Omega, h_b, tk)$.
2. **Learn the self-destruct point:** Sample random coins $r \leftarrow \{0,1\}^*$ and run $\mathsf{F}^{\mathcal{O}^{\ell}(X_0), \mathcal{O}^{\ell}(X_1)}(r)$. Note that each leakage query to $\mathcal{O}^{\ell}(X_b)$ can be translated to a leakage query to $\mathcal{O}^{\ell'}(s_b)$ by hard-wiring $(h_{1-b}, \pi_{1-b}, \pi_b)$. Let $(j^*, \mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$ be the output of F.
3. **Simulate tampering queries:** At this point A' is done with the leakage queries and picks $\theta = 0$, asking to reveal s_0. Hence, A' runs $\mathsf{A}(r)$ simulating the answer for its tampering queries as follows:
   (a) For all $j = 1, \ldots, j^*$ given tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ forward to A the value $\mathbf{\Theta}_0$ as computed in step 3a—3d of the description of $\mathsf{S}^{\mathcal{O}^{\ell}(X_1)}(X_0; r)$ (cf. Figure 1). Notice that to run S no further leakage query is needed, as the answers to $\mathsf{A}(r)$'s leakage queries are already contained in $(\mathbf{\Lambda}_0, \mathbf{\Lambda}_1)$.
   (b) For all $j > j^*$ given tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ forward to A the value $\perp$.
4. **Guess:** Output whatever D does.

For the analysis, note that by Lemma 6 the reduction runs in polynomial time and requires at most $4\ell + (2k+1)\lceil\log(q)\rceil$ bits of leakage. Moreover, by Lemma 4, the simulation of the answers to A's tampering queries is identical to the distribution A would have seen in the real experiment (with overwhelming probability). Since we are assuming that D, A have a non-negligible advantage, we have that D', A' have also a non-negligible advantage, contradicting the assumption that the LRS scheme is secure. This finishes the proof.

4.3 Proof of Lemma 4

Before proving the lemma, we establish some notation that we will use also in the proof of Lemma 5. Denote with $X_0 = (s_0, h_1, \pi_1, \pi_0)$, $X_1 = (s_1, h_0, \pi_0, \pi_1)$ the target encoding that is sampled at the beginning of the two experiments $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}$. Given a tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ we write $X'_0 = \mathsf{T}_0^{(j)}(X_0)$, $X'_1 = \mathsf{T}_1^{(j)}(X_1)$ for the tampered codeword in experiment $\mathsf{Tamper}'_{\mathsf{A}}$. For some value $b \in \{0, 1\}$, we write $\mathsf{S}(b)$ as a short hand of $\mathsf{S}^{\mathcal{O}^\ell(X_{1-b})}(X_b)$. Let $X'_{b,b} = \mathsf{T}_b^{(j)}(X_b)$ be the tampered half computed by the simulator $\mathsf{S}(b)$ in experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$. Recall that in case $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type D, then according to the description of S (c.f. step 3d in Figure 1), then the simulator "extracts" the other half of the codeword which we denote by $X'_{b,1-b}$.

On the probability space. In each experiment there are two sources of randomness: (i) the randomness of the encoding process to generate the target encoding (X_0, X_1) and (ii) the randomness r of the adversary. We have to show that with overwhelming probability over the choice of this randomness the simulator $\mathsf{S}(r)$ produces exactly the same view as $\mathsf{A}(r)$ would have seen in the real experiment. In the proof we will consider some "bad" event over the above randomness space and prove that such event happens with negligible probability to obtain the statement. In what follows all probabilities are taken over the above randomness space.

We emphasize that we cannot directly assume that $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is the same in $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}$, as the tampering functions are chosen adaptively by A depending on the view in the two experiments. However, since in both the experiments A is run with the same random tape r, we have that $(\mathsf{T}_0^{(1)}, \mathsf{T}_1^{(1)})$ is the same in $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}$ as the choice of the first tampering query depends only on the distribution of the CRS Ω (which is the same in the two worlds).

Below, we state that whenever the simulator S extracts a value $X'_{b,1-b}$, then the resulting encoding is valid with overwhelming probability.

Claim 1. *Whenever $\mathsf{S}(b)$ sets $\mathbf{\Theta}_b[j] = (X'_{b,b}, X'_{b,1-b})$ in response to a tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ for some $1 \le j \le q$, then $\mathbf{\Theta}_b[j]$ is a valid encoding with overwhelming probability.*

Proof. We make the proof for $b = 0$; the proof for $b = 1$ is similar. Following the description of $\mathsf{S}(0)$, we observe that the experiment $\mathsf{Tamper}''_{\mathsf{S}}(0)$ sets $\mathbf{\Theta}_0[j] = (X'_{0,0}, X'_{0,1})$ only in the step 3d (i.e., when the j-th tampering query is of type D). Let $X'_{0,0} = (s'_0, h'_1, \pi'_1, \pi'_0)$ and $X'_{0,1} = (s'_1, h'_0, \pi'_0, \pi'_1)$, where $s'_1 \leftarrow \mathrm{Xtr}^{h'_0}(\Omega, (h'_1, \pi'_1), ek)$.[11] Now, for the sake of contradiction, assume that $(X'_{0,0}, X'_{0,1})$ is invalid. By definition of the decoding algorithm (c.f. Section 4), we observe that the only possibility is that the extracted value s'_1 does not match the statement h'_1, i.e. $h'_1 \neq H_t(s'_1)$. By simulation extractability of the NIZK, this can only happen with negligible probability. Therefore we conclude that $(X'_{0,0}, X'_{0,1})$ is a valid codeword with overwhelming probability, as desired. □

We now turn to the proof of Lemma 4.

Lemma 4. *For all x and all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that A generates a self-destruct in $\mathsf{Tamper}'_{\mathsf{A},x}$. Denote with $\mathsf{view}_{\mathsf{A}}$ (resp. $\mathsf{view}_{\mathsf{S},b}$) the view of $\mathsf{A}(r)$ (resp. $\mathsf{S}(r)$) running in experiment $\mathsf{Tamper}'_{\mathsf{A},x}$ (resp. $\mathsf{Tamper}''_{\mathsf{S},x}(b)$) with target encoding (X_0, X_1). Then, for all $b \in \{0, 1\}$, we have that $\mathsf{view}_{\mathsf{A}}^{(j^*-1)} = \mathsf{view}_{\mathsf{S},b}^{(j^*-1)}$ with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding.*

Proof. We make the proof for $b = 0$; the proof for $b = 1$ is similar. Without loss of generality we will always assume that in the j-th round the adversary $\mathsf{A}(r)$ asks a leakage query and a tampering

[11] Looking at the description of the simulator, the hashes (h'_0, h'_1) and the proofs (π'_0, π'_1) are the same in $X'_{0,0}$ and in $X'_{0,1}$.

query. The corresponding output of the experiments in this round is denoted by $\mathsf{out}_{\mathsf{A}}^{(j)} = (\mathbf{\Lambda}[j], \mathbf{\Theta}[j])$ in $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{out}_{\mathsf{S},0}^{(j)} = (\mathbf{\Lambda}_0[j], \mathbf{\Theta}_0[j])$ in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Denote the distribution of the outputs of the two experiments $\mathsf{Tamper}''_{\mathsf{S}}(0)$ and $\mathsf{Tamper}'_{\mathsf{A}}$ until round j as

$$\mathsf{view}_{\mathsf{S},0}^{(j)} = (\mathbf{\Lambda}_0[1], \mathbf{\Theta}_0[1], \ldots, \mathbf{\Lambda}_0[j], \mathbf{\Theta}_0[j]) \qquad \text{and} \qquad \mathsf{view}_{\mathsf{A}}^{(j)} = (\mathbf{\Lambda}[1], \mathbf{\Theta}[1], \ldots, \mathbf{\Lambda}[j], \mathbf{\Theta}[j]).$$

We compute the following:

$$\begin{aligned}
&\mathbb{P}\left[\mathsf{view}_{\mathsf{A}}^{(j^*-1)} \neq \mathsf{view}_{\mathsf{S},0}^{(j^*-1)}\right] \\
&\leq \mathbb{P}\left[\exists j \in [j^*-1] : (\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)}) \wedge (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \sum_{j=1}^{j^*-1} \mathbb{P}\left[(\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)}) \wedge (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \sum_{j=1}^{j^*-1} \mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \cdot \mathbb{P}\left[(\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \sum_{j=1}^{j^*-1} \mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \\
&\leq \mathit{negl}(k)
\end{aligned}$$

The last inequality follows from Lemma 7 (which we prove below), and the fact the sum is taken only for polynomially many values since j^* is polynomially bounded. □

Lemma 7. *For all $j \in [j^*-1]$, we have that*

$$\mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right] \leq \mathit{negl}(k).$$

Proof. Fix any $j \in [j^*-1]$. Let $\textsc{Good}^{(j-1)}$ denote the event that $(\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})$. Note that, since we are comparing the variables $\mathsf{out}_{\mathsf{A}}^{(j)}$ and $\mathsf{out}_{\mathsf{S},0}^{(j)}$ conditioned on the event $\textsc{Good}^{(j-1)}$, in both the experiments $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$ the leakage query $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$ and the tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ issued by adversary $\mathsf{A}(st; r)$ (where st is either $\mathsf{view}_{\mathsf{A}}^{(j-1)}$ or $\mathsf{view}_{\mathsf{S},0}^{(j-1)}$) in the j-th round are the same.

Consider the following events in the experiment $\mathsf{Tamper}''_{\mathsf{S}}(0)$.

- $\textsc{Bad}_1^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type A *and* we have $\mathsf{T}_1^{(j)}(X_1) \neq X_1$.
- $\textsc{Bad}_2^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type B.
- $\textsc{Bad}_3^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type C.
- $\textsc{Bad}_4^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type D, *and* either of the following two facts happens:
  - The encoding $(X'_{0,0}, X'_{0,1})$ is not valid; *or*
  - The encoding $(X'_{0,0}, X'_{0,1})$ is valid, but we have $\mathsf{T}_1^{(j)}(X_1) \neq X'_{0,1}$ (recall that $X'_{0,1}$ is the "extracted half").

Define the union of all those four events by $\textsc{Bad}^{(j)} = \bigvee_{i=1}^{4} \textsc{Bad}_i^{(j)}$.

The next four claims show that the probability of each event $\text{BAD}_i^{(j)}$ is negligible, conditioned on the fact that the views until round $j-1$ are equal in the real and the simulated experiment.

Claim 2. $\mathbb{P}\left[\text{BAD}_1^{(j)}|\text{GOOD}^{(j-1)}\right] \leq negl(k)$.

Proof. Recall the definition of the event $\text{BAD}_1^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type A *and* we have $\mathsf{T}_1^{(j)}(X_1) \neq X_1$ in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. By definition of type A query (cf. Figure 1), we have that $X'_{0,0} = \mathsf{T}_0^{(j)}(X_0) = X_0$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$ would output $\mathsf{same}^\star$ in this round. Now, since we are conditioning on the the event $\text{GOOD}^{(j-1)}$, we get the same tampering query in this round which implies $X'_0 = X'_{0,0} = X_0$ and $X'_1 = \mathsf{T}_1^{(j)}(X_1) \neq X_1$. As (X_0, X_1) and (X_0, X'_1) are both valid, (X_0, X_1, X'_1) violates uniqueness (c.f. Lemma 3) of our encoding scheme. □

Claim 3. $\mathbb{P}\left[\text{BAD}_2^{(j)}|\text{GOOD}^{(j-1)}\right] \leq negl(k)$.

Proof. Recall the definition of the event $\text{BAD}_2^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type B in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Let $X'_{0,0} = (s'_0, h'_1, \pi'_1, \pi'_0)$ be the value computed in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. From the description of S we observe that in this case $X'_{0,0} \neq X_0$ and at least one of the proofs π'_0, π'_1 does not verify correctly.

Now, conditioning on $\text{GOOD}^{(j-1)}$, we get $X'_0 = X'_{0,0}$ in $\mathsf{Tamper}'_{\mathsf{A}}$. In this case the experiment would output $\perp$, resulting in a self-destruct, which contradicts our assumption that $j < j^*$. □

Claim 4. $\mathbb{P}\left[\text{BAD}_3^{(j)}|\text{GOOD}^{(j-1)}\right] \leq negl(k)$.

Proof. Recall the definition of the event $\text{BAD}_3^{(j)}$: The event becomes true when $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type C, i.e. $X'_{0,0} = (s'_0, h'_1, \pi'_1, \pi'_0) \neq X_0$ and $\pi'_1 = \pi_1$ in $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Again conditioning on $\text{GOOD}^{(j-1)}$, we get that $X'_0 = (s'_0, h'_1, \pi_1, \pi'_0)$ in $\mathsf{Tamper}'_{\mathsf{A}}$. In this experiment consider the other half of the tampered codeword, i.e. $X'_1 = (s'_1, h'_0, \pi'_0, \pi_1)$. Note that the proofs match, as otherwise we would get $\perp$ in $\mathsf{Tamper}'_{\mathsf{A}}$ which contradicts the fact that $j < j^*$.

Note that we consider a NIZK proof system with the following two properties: (i) different statements have different proofs; (ii) it supports public labels which are appended with the statements (c.f. Section 2.2). Using these properties we get that $\pi'_1 = \pi_1$ implies $h'_1 = h_1$ and $h'_0 = h_0$. Here we use the fact that π_1 is a proof of statement h_1 with label h_0. In the next step, by collision resistance of the hash function we get $s'_1 = s_1$ and $s'_0 = s_0$, with overwhelming probability.

So far, we have shown that $X'_0 = (s_0, h_1, \pi_1, \pi'_0)$ and $X'_1 = (s_1, h_0, \pi'_0, \pi_1)$ in $\mathsf{Tamper}'_{\mathsf{A}}$ (with overwhleming probability). Since we must have $X'_0 = X'_{0,0} \neq X_0$, we get $\pi'_0 \neq \pi_0$ and hence the proof π'_0 can be extracted. We claim that the above contradicts leakage resilience of the underlying LRS scheme. To see this, we construct an adversary A' against $(\text{LRS}, \text{LRS}^{-1})$ attacking a target encoding (s_0, s_1) given a pair $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ which provokes the event $\text{BAD}_3^{(j)}$. Adversary A' works as follows:

- First, A' queries its own leakage oracles $\mathcal{O}^\ell(s_0), \mathcal{O}^\ell(s_1)$ to get the hash values $H_t(s_0) = h_0$ and $H_t(s_1) = h_1$. Then it runs $(\Omega, tk, ek) \leftarrow \text{Sim}_1(1^k)$ and computes the simulated proofs $\pi_0 \leftarrow \text{Sim}_2^{h_1}(\Omega, h_0, tk)$ and $\pi_1 \leftarrow \text{Sim}_2^{h_0}(\Omega, h_1, tk)$.
- Next, A' asks one additional leakage query to the oracle $\mathcal{O}^\ell(s_0)$ with hard-wired values h_1, π_0, π_1, and ek, and hard-wired function $\mathsf{T}_1^{(j)}$ as follows: Compute $X'_1 = \mathsf{T}_1^{(j)}(X_1) = (s_1, h_0, \pi'_0, \pi_1)$; extract $s_0 \leftarrow \text{Xtr}^{h_1}(\Omega, (h_0, \pi'_0), ek)$ and output the first bit of the decoded value $(\text{LRS}^{-1}(s_0, s_1))$.

Note that the above attack requires $2k + 1 \leq \ell'$ bits of leakage and clearly allows to break $(\text{LRS}, \text{LRS}^{-1})$ with overwhelming probability. This concludes the proof of claim. □

Claim 5. $\mathbb{P}\left[\text{BAD}_4^{(j)}|\text{GOOD}^{(j-1)}\right] \leq negl(k)$.

Proof. Recall the definition of the event $\textsc{Bad}_4^{(j)}$: The event becomes true when in $\mathsf{Tamper}''_{\mathsf{S}}(0)$, $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type D *and* either of the following two facts happens:

- The encoding $(X'_{0,0}, X'_{0,1})$ is not valid; *or*
- The encoding $(X'_{0,0}, X'_{0,1})$ is valid, but we have $\mathsf{T}_1^{(j)}(X_1) \neq X'_{0,1}$ which is actually the extracted half.

Note that by Claim 1, the encoding $(X'_{0,0}, X'_{0,1})$ must be valid with overwhelming probability. So, by a union bound, we can consider only the second case.

Conditioning on $\textsc{Good}^{(j-1)}$ implies that in $\mathsf{Tamper}'_{\mathsf{A}}$, $X'_0 = \mathsf{T}_0^{(j)}(X_0) = X'_{0,0}$ and $X'_1 = \mathsf{T}_1^{(j)}(X_1) \neq X'_{0,1}$. Moreover, (X'_0, X'_1) is a valid encoding pair as we are assuming $j < j^*$. Clearly this violates uniqueness, as in this case $X'_1 \neq X'_{0,1}$, concluding the proof.

□

To conclude the proof of Lemma 7 we observe the following:

$$\begin{aligned}
&\mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid (\mathsf{view}_{\mathsf{A}}^{(j-1)} = \mathsf{view}_{\mathsf{S},0}^{(j-1)})\right]\\
&= \mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \textsc{Good}^{(j-1)}\right]\\
&\leq \mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}\right] + \mathbb{P}\left[\textsc{Bad}^{(j)} | \textsc{Good}^{(j-1)}\right] && (1)\\
&\leq \mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}\right] + \sum_{i=1}^{4} \mathbb{P}\left[\textsc{Bad}_i^{(j)} | \textsc{Good}^{(j-1)}\right] && (2)\\
&\leq \mathit{negl}(k). && (3)
\end{aligned}$$

Eq. (1) follows from Bayes Theorem, Eq. (2) follows from the union bound and Eq. (3) from Claim 2—5 and Claim 6 (see below).

Claim 6. $\mathbb{P}\left[\mathsf{out}_{\mathsf{A}}^{(j)} \neq \mathsf{out}_{\mathsf{S},0}^{(j)} \mid \overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}\right] \leq \mathit{negl}(k)$.

Proof. Here we condition on both the events $\textsc{Good}^{(j-1)}$ and $\overline{\textsc{Bad}^{(j)}}$. Conditioning on $\textsc{Good}^{(j-1)}$ implies that $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$ and $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ are the same in both $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$. Given that, we need to analyze the following leakage and tampering components: $\mathsf{out}_{\mathsf{A}}^{(j)} = (\mathbf{\Lambda}[j], \mathbf{\Theta}[j])$ and $\mathsf{out}_{\mathsf{S},0}^{(j)} = (\mathbf{\Lambda}_0[j], \mathbf{\Theta}_0[j])$. It is easy to observe that, S simulates the leakage correctly as it can compute $\mathsf{L}_0^{(j)}(X_0)$ directly and forward $\mathsf{L}_1^{(j)}$ to $\mathcal{O}^{\ell}(X_1)$. This shows that $\mathbf{\Lambda}[j] = \mathbf{\Lambda}_0[j]$.

Now consider the following cases based on the type of the tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ in experiment $\mathsf{Tamper}''_{\mathsf{S}}(0)$:

- If $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type A, then $\mathbf{\Theta}_0[j] = \mathsf{same}^{\star}$. Since $\overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}$ implies $\overline{\textsc{Bad}_1^{(j)}} \wedge \textsc{Good}^{(j-1)}$, we get $\mathbf{\Theta}[j] = \mathsf{same}^{\star}$ in the experiment $\mathsf{Tamper}'_{\mathsf{A}}$.
- Since $\overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}$ implies $\overline{\textsc{Bad}_2^{(j)}} \wedge \overline{\textsc{Bad}_3^{(j)}} \wedge \textsc{Good}^{(j-1)}$, the tampering query $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is neither of type B nor of type C.
- If $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ is of type D, since $\overline{\textsc{Bad}^{(j)}} \wedge \textsc{Good}^{(j-1)}$ implies $\overline{\textsc{Bad}_4^{(j)}} \wedge \textsc{Good}^{(j-1)}$ we get $(X'_0, X'_1) = \mathbf{\Theta}[j] = \mathbf{\Theta}_0[j] = (X'_{0,0}, X'_{0,1})$ in both $\mathsf{Tamper}'_{\mathsf{A}}$ and $\mathsf{Tamper}''_{\mathsf{S}}(0)$.

We conclude that in every case we get $\mathbf{\Theta}[j] = \mathbf{\Theta}_0[j]$ conditioned on $\overline{\text{BAD}^{(j)}}$ and $\text{GOOD}^{(j-1)}$ which proves the claim. □

□

4.4 Proof of Lemma 5

Lemma 5. *For all x and for all PPT adversaries A, let $j^* \in [q]$ be the smallest index such that $\mathbf{\Theta}[j^*] = \perp$ in $\mathsf{Tamper}'_{\mathsf{A},x}(k)$. Denote with $\mathsf{view}_{\mathsf{S},b} = (\mathbf{\Lambda}_b, \mathbf{\Theta}_b)$ the view of $\mathsf{S}(r)$ running in experiment $\mathsf{Tamper}''_{\mathsf{S},x}(b)$ with target encoding (X_0, X_1). Then, the following holds with overwhelming probability over the choice of the randomness r and the coin tosses to sample the encoding:*

(i) $\mathbf{\Theta}_0[j] = \mathbf{\Theta}_1[j], \forall 1 \leq j \leq j^* - 1$.

(ii) $\mathbf{\Theta}_0[j^*] \neq \mathbf{\Theta}_1[j^*]$.

Proof. We compute the probability for which property (i) does not hold and show that this probability is negligible.

$$
\begin{aligned}
&\mathbb{P}\left[\exists j \in \{1, \dots, j^* - 1\} \;:\; \mathbf{\Theta}_0[j] \neq \mathbf{\Theta}_1[j]\right] \\
&\quad \leq \sum_{j=1}^{j^*-1} \mathbb{P}\left[\mathbf{\Theta}_0[j] \neq \mathbf{\Theta}_1[j]\right] \qquad (4)\\
&\quad \leq \sum_{j=1}^{j^*-1} \Big(\mathbb{P}\left[\mathbf{\Theta}_0[j] \neq \mathbf{\Theta}_1[j] \mid \mathbf{\Theta}_0[j] = \mathbf{\Theta}[j]\right] \cdot \mathbb{P}\left[\mathbf{\Theta}_0[j] = \mathbf{\Theta}[j]\right] \\
&\qquad\qquad + \mathbb{P}\left[\mathbf{\Theta}_0[j] \neq \mathbf{\Theta}_1[j] \mid \mathbf{\Theta}_0[j] \neq \mathbf{\Theta}[j]\right] \cdot \mathbb{P}\left[\mathbf{\Theta}_0[j] \neq \mathbf{\Theta}[j]\right] \Big) \\
&\quad \leq \sum_{j=1}^{j^*-1} \Big(\mathbb{P}\left[\mathbf{\Theta}_0[j] \neq \mathbf{\Theta}_1[j] \mid \mathbf{\Theta}_0[j] = \mathbf{\Theta}[j]\right] + \mathbb{P}\left[\mathbf{\Theta}_0[j] \neq \mathbf{\Theta}[j]\right] \Big) \\
&\quad \leq \sum_{j=1}^{j^*-1} \sum_{b=1}^{2} \mathbb{P}\left[(\mathbf{\Theta}_b[j] \neq \mathbf{\Theta}[j]\right] \qquad (5)\\
&\quad \leq negl(k) \qquad (6)
\end{aligned}
$$

Eq. (4) follow by a union bound. Eq. (6) follows from Lemma 4 and the fact that j^* is polynomially bounded. This concludes the proof of part (i).

Now we prove part (ii) of the lemma. Define the following event BAD', over the probability space of experiment $\mathsf{Tamper}''_{\mathsf{S}}(b)$: The event becomes true when in round j^*, we have $\mathbf{\Theta}_0[j^*] = \mathbf{\Theta}_1[j^*]$. In order to satisfy this condition, the only possible values $\mathbf{\Theta}_b[j^*]$ (for $b \in \{0, 1\}$) can take are as follows:

1. $\mathbf{\Theta}_0[j^*] = \mathbf{\Theta}_1[j^*] = \mathsf{same}^\star$. By Lemma 4, we get that before round j^* the view of $\mathsf{A}(r)$ in $\mathsf{Tamper}'_{\mathsf{A}}$ is equal to the view of $\mathsf{A}(r)$ in $\mathsf{Tamper}''_{\mathsf{S}}(b)$ (with overwhelming probability). This implies that in round j^* the tampering query $(\mathsf{T}_0^{(j^*)}, \mathsf{T}_1^{(j^*)})$ is the same in both experiments. This gives $\mathbf{\Theta}[j^*] = \mathsf{same}^\star$, which contradicts the fact that j^* is the self-destruct round in $\mathsf{Tamper}'_{\mathsf{A}}$.

2. $\mathbf{\Theta}_0[j^*] = \mathbf{\Theta}_1[j^*] = (X'_0, X'_1)$. Note that (X'_0, X'_1) is valid with overwhelming probability (by Claim 1). Now due to the same reason as above Lemma 4 implies that $\mathbf{\Theta}[j^*] = (X'_0, X'_1)$ in $\mathsf{Tamper}'_{\mathsf{A}}$ with overwhelming probability, which is a contradiction to the fact that j^* is the self-destruct round.

Therefore using a union bound we can argue that $\mathbb{P}[\text{BAD}'] \leq negl(k)$ which concludes the proof of part (ii).

□

5 Application to Tamper Resilient Security

In this section we apply our notion of CNMLR codes to protect arbitrary functionalities against split-state tampering and leakage attacks.

5.1 Stateless Functionalities

We start by looking at the case of *stateless* functionalities $\mathcal{G}(st, \cdot)$, which take as input a secret state $st \in \{0,1\}^k$ and a value $x \in \{0,1\}^u$ to produce some output $y \in \{0,1\}^v$. The function $\mathcal{G}$ is public and can be randomized.

The main idea is to transform the original functionality $\mathcal{G}(st, \cdot)$ into some "hardened" functionality $\mathcal{G}^{\mathsf{Code}}$ via a CNMLR code Code. Previous transformations aiming to protect stateless functionalities [17, 23] required to freshly re-encode the state st each time the functionality is invoked. Our approach avoids the re-encoding of the state at each invocation, leading to a stateless transformation. This solves an open question from [17]. Moreover we consider a setting where the encoded state is stored in a memory $(\mathcal{M}_0, \mathcal{M}_1)$ which is much larger than the size needed to store the encoding itself (say $|\mathcal{M}_0| = |\mathcal{M}_1| = s$ where s is polynomial in the length of the encoding). When (perfect) erasures are not possible, this feature allows the adversary to make copies of the initial encoding and tamper continuously with it, and was not considered in previous models.

Let us formally define what it means to harden a stateless functionality.

Definition 5 (Stateless hardened functionality)**.** Let $\mathsf{Code} = (\text{Init}, \text{Encode}, \text{Decode})$ be a split-state encoding scheme in the CRS model, with k bits messages and $2n$ bits codewords. Let $\mathcal{G} : \{0,1\}^k \times \{0,1\}^u \to \{0,1\}^v$ be a *stateless* functionality with secret state $st \in \{0,1\}^k$, and let $\varphi \in \{0,1\}$ be a public value initially set to zero. We define a *stateless* hardened functionality $\mathcal{G}^{\mathsf{Code}} : \{0,1\}^{2s} \times \{0,1\}^u \to \{0,1\}^v$ with a modified state $st' \in \{0,1\}^{2s}$ and $s = poly(n)$. The hardened functionality $\mathcal{G}^{\mathsf{Code}}$ is a triple of algorithms $(\text{Init}, \text{Setup}, \text{Execute})$ described as follows:

- $\Omega \leftarrow \text{Init}(1^k)$: Run the initialization procedure of the coding scheme to sample $\Omega \leftarrow \text{Init}(1^k)$.
- $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow \text{Setup}(\Omega, st)$: Let $(X_0, X_1) \leftarrow \text{Encode}(\Omega, st)$. For $b \in \{0,1\}$, store X_b in the first n bits of $\mathcal{M}_b$, i.e. $\mathcal{M}_b[1 \dots n] \leftarrow X_b$. (The remaining bits of $\mathcal{M}_b$ are set to 0^{s-n}.) Define $st' := (\mathcal{M}_0, \mathcal{M}_1)$.
- $y \leftarrow \text{Execute}(x)$: Read the public value φ. In case $\varphi = 1$ output $\perp$. Otherwise, let $X_b = \mathcal{M}_b[1 \dots n]$ for $b \in \{0,1\}$. Run $st \leftarrow \text{Decode}(\Omega, (X_0, X_1))$; if $st = \perp$, then output $\perp$ and set $\varphi = 1$. Otherwise output $y \leftarrow \mathcal{G}(st, x)$.

Remark (On φ)**.** The public value φ is just a way how to implement the "self-destruct" feature. An alternative approach would be to let the hardened functionality simply output a dummy value and overwrite $(\mathcal{M}_0, \mathcal{M}_1)$ with the all-zero string. As we insist on the hardened functionality being stateless, we use the first approach here.

Note that we assume that φ is untamperable. It is easy to see that this is necessary, as an adversary tampering with φ could always switch-off the self-destruct feature and apply a variant of the attack from [20] to recover the secret state.

Similarly to [17, 23], security of $\mathcal{G}^{\mathsf{Code}}$ is defined via the comparison of a real and an ideal experiment. The real experiment features an adversary A interacting with $\mathcal{G}^{\mathsf{Code}}$; the adversary is allowed to honestly run the functionality on any chosen input, but also to modify the secret state and retrieve a bounded amount of information from it. The ideal experiment features a simulator S; the simulator is given black-box access to the original functionality $\mathcal{G}$ and to the adversary A, but is *not* allowed any tampering or leakage query. The two experiments are formally described below.

Experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$**.** First $\Omega \leftarrow \mathrm{Init}(1^k)$ and $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathrm{Setup}(\Omega, st)$ are run and Ω is given to A. Then A can issue the following commands polynomially many times (in any order):

- $\langle \mathtt{Leak}, (\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})\rangle$: In response to the j-th leakage query, compute $\Lambda_0^{(j)} \leftarrow \mathsf{L}_0^{(j)}(\mathcal{M}_0)$ and $\Lambda_1^{(j)} \leftarrow \mathsf{L}_1^{(j)}(\mathcal{M}_1)$ and output $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- $\langle \mathtt{Tamper}, (\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})\rangle$: In response to the j-th tampering query, compute $\mathcal{M}_0' \leftarrow \mathsf{T}_0^{(j)}(\mathcal{M}_0)$ and $\mathcal{M}_1' \leftarrow \mathsf{T}_1^{(j)}(\mathcal{M}_1)$ and replace $(\mathcal{M}_0, \mathcal{M}_1)$ with $(\mathcal{M}_0', \mathcal{M}_1')$.
- $\langle \mathtt{Eval}, x_j\rangle$: In response to the j-th evaluation query, run $y_j \leftarrow \mathrm{Execute}(x_j)$. In case $y_j = \perp$ output $\perp$ and self-destruct; otherwise output y_j.

The output of the experiment is defined as

$$\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k) = (\Omega; ((x_1, y_1), (x_2, y_2), \dots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots)).$$

Experiment $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$**.** The simulator sets up the CRS Ω and is given black-box access to the functionality $\mathcal{G}(st, \cdot)$ and the adversary A. The output of the experiment is defined as

$$\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k) = (\Omega; ((x_1, y_1), (x_2, y_2), \dots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots)),$$

where $((x_j, y_j), ((\Lambda_0^{(j)}, \Lambda_1^{(j)})))$ are the input/output/leakage tuples simulated by S.

Definition 6 (Polyspace leak/tamper simulatability)**.** Let Code be a split-state encoding scheme in the CRS model and consider a stateless functionality $\mathcal{G}$ with corresponding hardened functionality $\mathcal{G}^{\mathsf{Code}}$. We say that Code is polyspace (ℓ, q)-leak/tamper simulatable for $\mathcal{G}$, if the following conditions are satisfied:

1. Each memory part $\mathcal{M}_b$ (for $b \in \{0, 1\}$) has size $s = poly(n)$.
2. The adversary asks at most q tampering queries and leaks a total of at most ℓ bits from each memory part.
3. For all PPT adversaries A there exists a PPT simulator S such that for any initial state st,

$$\left\{\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)\right\}_{k\in\mathbb{N}} \approx_c \left\{\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)\right\}_{k\in\mathbb{N}}.$$

We show the following result.

Theorem 2. *Let $\mathcal{G}$ be a stateless functionality and* $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ *be any (ℓ, q)-CNMLR split-state encoding scheme in the CRS model. Then* Code *is polyspace (ℓ, q)-leak/tamper simulatable for $\mathcal{G}$.*

Proof. We discuss the overall proof approach first. We start with describing a simulator S running in experiment $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$ which attempts to simulate the view of adversary A running in the experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$; the simulator is given black-box access to A (which can issue `Tamper`, `Leak`, and `Eval` queries) and to the functionality $\mathcal{G}(st,\cdot)$ for some state st. To argue that our simulator is "good" we show that if there exists a PPT distinguisher D and a PPT adversary A such that for some state st, D distinguishes the experiments $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$ and $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$ with non-negligible probability, then we can build another distinguisher D' and an adversary A' such that D' can distinguish $\mathsf{Tamper}_{\mathsf{A}',0^k}^{\mathrm{cnmlr}}$ and $\mathsf{Tamper}_{\mathsf{A}',st}^{\mathrm{cnmlr}}$ with non-negligible probability. In the last step essentially we reduce the CNMLR property of Code to the polyspace leak/tamper simulatability of the code itself.

The simulator starts by sampling the common reference string $\Omega \leftarrow \mathrm{Init}(1^k)$ and the public value $\varphi = 0$. Then it samples a random encoding of 0^k, namely $(Z_0, Z_1) \leftarrow \mathrm{Encode}(\Omega, 0^k)$ and sets $\mathcal{M}_b[1\dots,n] \leftarrow Z_b$ for $b \in \{0,1\}$. The remaining bits of $(\mathcal{M}_0,\mathcal{M}_1)$ are set to 0^{s-n}. Hence, S alternates between the following two modes (starting with the normal mode in the first round):

- *Normal Mode.* Given state $(\mathcal{M}_0,\mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle \mathtt{Eval}, x_j\rangle$: Upon input the j-th evaluation query invoke $\mathcal{G}(st,\cdot)$ to get $y_j \leftarrow \mathcal{G}(st,x_j)$ and reply with y_j.
  - $\langle \mathtt{Tamper}, (\mathsf{T}_0^{(j)},\mathsf{T}_1^{(j)})\rangle$: Upon input the j-th tampering query, compute $\mathcal{M}'_b \leftarrow \mathsf{T}_b^{(j)}(\mathcal{M}_b)$ for $b\in\{0,1\}$. In case $(\mathcal{M}'_0[1\dots n],\mathcal{M}'_1[1\dots n]) = (Z_0,Z_1)$ then continue in the current mode. Otherwise go to the overwritten mode defined below with state $(\mathcal{M}'_0,\mathcal{M}'_1)$.
  - $\langle \mathtt{Leak}, (\mathsf{L}_0^{(j)},\mathsf{L}_1^{(j)})\rangle$: Upon input the j-th leakage query, compute $\Lambda_b^{(j)} = \mathsf{L}_b^{(j)}(Z_b)$ for $b\in\{0,1\}$ and reply with $(\Lambda_0^{(j)},\Lambda_1^{(j)})$.
- *Overwritten Mode.* Given state $(\mathcal{M}'_0,\mathcal{M}'_1)$, while A continues issuing queries, answer as follows:
  - Let $\tau = (\mathcal{M}'_0,\mathcal{M}'_1)$. Simulate the hardened functionality $\mathcal{G}^{\mathsf{Code}}(\tau,\cdot)$ and answer all `Eval` and `Leak` queries as the real experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(\tau,\cdot)}(k)$ would do.
  - Upon input the j-th tampering query $(\mathsf{T}_0^{(j)},\mathsf{T}_1^{(j)})$, compute $\mathcal{M}''_b \leftarrow \mathsf{T}_b^{(j)}(\mathcal{M}'_b)$ for $b\in\{0,1\}$. In case $(\mathcal{M}''_0[1\dots n],\mathcal{M}''_1[1\dots n]) = (Z_0,Z_1)$ then go to the normal mode with state $(\mathcal{M}_0,\mathcal{M}_1) := (\mathcal{M}''_0,\mathcal{M}''_1)$. Otherwise continue in the current mode.
- When A halts and outputs $\mathsf{view}_{\mathsf{A}} = (\Omega; ((x_1,y_1),(x_2,y_2),\dots); ((\Lambda_0^{(1)},\Lambda_1^{(1)}),(\Lambda_0^{(2)},\Lambda_1^{(2)}),\cdots))$, set $\mathsf{view}_{\mathsf{S}} = \mathsf{view}_{\mathsf{A}}$ and output $\mathsf{view}_{\mathsf{S}}$ as output of $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$.

Intuitively, since the coding scheme is non-malleable, the adversary can either keep the encoding unchanged or overwrite it with the encoding of some unrelated message. These two cases are captured in the above modes: The simulator starts in the normal mode and then, whenever the adversary mauls the initial encoding, it switches to the overwritten mode. However, the adversary can use the extra space to keep a copy of the original encoding and place it back at some later point in time. When this happens, the simulator switches back to the normal mode; this switching is important to maintain simulation.

To finish the proof, we have to argue that the output of experiment $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$ is computationally indistinguishable from the output of experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$. This is done in the lemma below.

Lemma 8. *Let S be defined as above. Then for all PPT adversaries A and all $st \in \{0,1\}^k$, the following holds:*

$$\left\{\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)\right\}_{k\in\mathbb{N}} \approx_c \left\{\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)\right\}_{k\in\mathbb{N}}.$$

Proof. By contradiction, assume that there exists a PPT distinguisher D, a PPT adversary A and some state $st \in \{0,1\}^k$ such that:

$$\left|\mathbb{P}\left[\mathsf{D}(\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)) = 1\right]\right| \geq \epsilon, \tag{7}$$

where $\epsilon(k)$ is some non-negligible function of the security parameter k.

We build a PPT distinguisher D' and a PPT adversary A' telling apart the experiments $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A}',0^k}(k)$ and $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A}',st}(k)$; this contradicts our assumption that Code is CNMLR. The distinguisher D' is given the CRS $\Omega \leftarrow \mathrm{Init}(1^k)$ and can access $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ (for at most q times) and $\mathcal{O}^{\ell}(X_0)$, $\mathcal{O}^{\ell}(X_1)$; here (X_0, X_1) is either an encoding of 0^k or an encoding of st. The distinguisher D' keeps a flag SAME (initially set to TRUE) and a flag STOP (initially set to FALSE). After simulating the public values, D' mimics the enviroment for D as follows:

- $\langle \mathtt{Tamper}, (\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})\rangle$: Upon input tampering functions $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$, the distinguisher D' uses the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ to answer them.[12] However, it can not simply forward the queries because of the following two reasons:

  - The tampering functions $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ maps from s bits to s bits, whereas the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ expects tampering functions mapping from n bits to n bits.
  - In both the real and the ideal experiments the tampering functions are applied to the current state (which may be different from the initial state), whereas in experiment $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A}',*}$ the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ always applies $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ to the target encoding (X_0, X_1).

  To take into account the above differences, D' modifies $(\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})$ as follows. Define the functions $\mathsf{T}_{\mathsf{in}} : \{0,1\}^n \rightarrow \{0,1\}^s$ and $\mathsf{T}_{\mathsf{out}} : \{0,1\}^s \rightarrow \{0,1\}^n$ as $\mathsf{T}_{\mathsf{in}}(x) = (x||0^{s-n})$ and $\mathsf{T}_{\mathsf{out}}(x||x') = x$, for any $x \in \{0,1\}^n$ and $x' \in \{0,1\}^{s-n}$. The distinguisher D' queries $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ with the function pair $(\tilde{\mathsf{T}}_0^{(j)}, \tilde{\mathsf{T}}_1^{(j)})$ where each $\tilde{\mathsf{T}}_b^{(j)}$ is defined as $\tilde{\mathsf{T}}_b^{(j)} := \mathsf{T}_{\mathsf{out}} \circ \mathsf{T}_b^{(j)} \circ \mathsf{T}_b^{(j-1)} \circ \ldots \circ \mathsf{T}_b^{(1)} \circ \mathsf{T}_{\mathsf{in}}$ for $b \in \{0,1\}$.

  In case the oracle returns $\perp$, then D' sets STOP to TRUE. In case the oracle returns $\mathsf{same}^{\star}$, then D' sets SAME to TRUE. Otherwise, in case the oracle returns an encoding (X_0', X_1'), then D' sets SAME to FALSE.

- $\langle \mathtt{Leak}, (\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})\rangle$: Upon input leakage functions $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$, the distinguisher D' defines $(\tilde{\mathsf{L}}_0^{(j)}, \tilde{\mathsf{L}}_1^{(j)})$ (in a similar way as above), forwards those functions to $\mathcal{O}^{\ell}(X_0)$, $\mathcal{O}^{\ell}(X_1)$ and sends the answer from the oracles back to D.

- $\langle \mathtt{Eval}, x_j\rangle$: Upon input an evaluation query for value x_j, the distinguisher D' checks first that STOP equals FALSE. If this is not the case, then D' returns $\perp$ to D. Otherwise, D' checks that SAME equals TRUE. If this is the case, it runs $y_j \leftarrow \mathcal{G}(st, x_j)$ and gives y_j to D. Else (if SAME equals FALSE), it computes $y_j \leftarrow \mathcal{G}(st', x_j)$, where st' is the output of $\mathrm{Decode}(\Omega, (X_0', X_1'))$, and gives y_j to D.

Finally, D' outputs whatever D outputs.

For the analysis, first note that D' runs in polynomial time. Furthermore, D' asks exactly q queries to $\mathcal{O}_{\mathsf{cnm}}$ and leaks at most ℓ bits from the target encoding (X_0, X_1). It is also easy to see that in case (X_0, X_1) is an encoding of $st \in \{0,1\}^k$, then D' perfectly simulates the view of adversary D in the experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$. On the other hand, in case (X_0, X_1) is an encoding of 0^k, we claim that

[12] Formally D' has to access $\mathcal{O}_{\mathsf{cnm}}(\cdot)$ via A'. For simplicity we assume that D' can access the oracle directly. In fact, A' just acts as an interface between the experiment $\mathsf{Tamper}^{\mathrm{cnmlr}}_{\mathsf{A}',*}$ and D'.

D' perfectly simulates the view of D in the experiment $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$. This is because: (i) Whenever SAME equals TRUE, then D' answers evaluation queries by running $\mathcal{G}$ on state st and tampering/leakage queries using a pre-sampled encoding of 0^k (this corresponds to the normal mode of S); (ii) Whenever SAME equals FALSE, then D' answers evaluation queries by running $\mathcal{G}$ on the current tampered state st' which results from applying the tampering functions to a pre-sampled encoding of 0^k (this corresponds to the overwritten mode of S).

Combining the above argument with Eq. (7) we obtain

$$\left|\mathbb{P}\left[\mathsf{D}(\mathsf{Tamper}_{\mathsf{A}',0^k}^{\mathrm{cnmlr}}(k)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathsf{Tamper}_{\mathsf{A}',st}^{\mathrm{cnmlr}}(k)) = 1\right]\right| \geq \epsilon,$$

which is a contradiction to the fact that Code is (ℓ, q)-CNMLR. □

□

5.2 Stateful Functionalities

Finally, we consider the case of primitives that update their state at each execution, i.e. functionalities of the type $(st_{\mathsf{new}}, y) \leftarrow \mathcal{G}(st, x)$ (a.k.a. *stateful* functionalities). Note that in this case the hardened functionality re-encodes the new state at each execution.

Note that, since we do *not* assume erasure in our model, an adversary can always 'reset' the functionality to a previous valid state as follows: It could just copy the previous state to some part of the large memory and replace the current encoding by that. To avoid this, our transformation uses an untamperable *public* counter (along with the untamperable self-destruct bit) that helps us to detect whether the functionality is reset to a previous state, leading to a self-destruct. However such a counter can be implemented, for instance using $\log(k)$ bits. We notice that such a counter is necessary to protect against the above resetting attack. However, we stress that if we do not assume such a counter this "resetting" is the only harm the adversary can make in our model.

Below, we define what it means to harden a stateful functionality.

Definition 7 (Stateful hardened functionality)**.** Let $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ be a split-state encoding scheme in the CRS model, with $2k$ bits messages and $2n$ bits codewords. Let $\mathcal{G} : \{0,1\}^k \times \{0,1\}^u \to \{0,1\}^k \times \{0,1\}^v$ be a *stateful* functionality with secret state $st \in \{0,1\}^k$, $\varphi \in \{0,1\}$ be a public value and let $\langle\gamma\rangle$ be a public $\log(k)$-bit counter both initially set to zero. We define a *stateful* hardened functionality $\mathcal{G}^{\mathsf{Code}} : \{0,1\}^{2s} \times \{0,1\}^u \to \{0,1\}^{2s} \times \{0,1\}^v$ with a modified state $st' \in \{0,1\}^{2s}$ and $s = poly(n)$. The hardened functionality $\mathcal{G}^{\mathsf{Code}}$ is a triple of algorithms $(\mathrm{Init}, \mathrm{Setup}, \mathrm{Execute})$ described as follows:

- $\Omega \leftarrow \mathrm{Init}(1^k)$: Run the initialization procedure of the coding scheme to sample $\Omega \leftarrow \mathrm{Init}(1^k)$.
- $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathrm{Setup}(\Omega, st)$: Let $(X_0, X_1) \leftarrow \mathrm{Encode}(\Omega, st||\langle 1\rangle)$ and increment $\langle\gamma\rangle \leftarrow \langle\gamma\rangle + 1$. For $b \in \{0,1\}$, store X_b in the first n bits of $\mathcal{M}_b$, i.e. $\mathcal{M}_b[1 \dots n] \leftarrow X_b$.[13] (The remaining bits of $\mathcal{M}_b$ are set to 0^{s-n}.) Define $st' := (\mathcal{M}_0, \mathcal{M}_1)$.
- $y \leftarrow \mathrm{Execute}(x)$: Read the public bit φ. In case $\varphi = 1$ output $\perp$. Otherwise recover $X_b = \mathcal{M}_b[1 \dots n]$ for $b \in \{0,1\}$ and run $(st''||\langle\gamma'\rangle) \leftarrow \mathrm{Decode}(\Omega, (X_0, X_1))$. Read the public counter $\langle\gamma\rangle$. If $\langle\gamma\rangle \neq \langle\gamma'\rangle$ or $st'' = \perp$, set $\varphi = 1$. Else run $(st_{\mathsf{new}}, y) \leftarrow \mathcal{G}(st'', x)$ and output y. Finally, write $\mathrm{Encode}(\Omega, st_{\mathsf{new}}||\langle\gamma + 1\rangle)$ in $(\mathcal{M}_0[1, \dots, n], \mathcal{M}_1[1, \dots, n])$ and increment $\langle\gamma\rangle \leftarrow \langle\gamma\rangle + 1$.

Remark (On $\langle\gamma\rangle$)**.** Note that the counter is incremented after each evaluation query, and the current value is encoded together with the new state. We require $\langle\gamma\rangle$ to be untamperable. This assumption

[13] Without erasure this can be easily implemented by a stack.

is necessary, as otherwise an adversary could always use the extra space to keep a copy of a previous valid state and place it back at some later point in time. The above attack allows essentially to reset the functionality to a previous state, and cannot be simulated with black-box access to the original functionality.

In the case of stateful primitives, the hardened functionality has to re-encode the new state at each execution. Still, as the memory is large, the adversary can use the extra space to tamper continuously with a target encoding of some valid state. Security of a stateful hardened functionality is defined analogously to the stateless case (cf. Definition 6). We show the following result:

Theorem 3. *Let $\mathcal{G}$ be a stateful functionality and* $\mathsf{Code} = (\mathrm{Init}, \mathrm{Encode}, \mathrm{Decode})$ *be any* (ℓ, q)*-CNMLR encoding scheme in the split-state CRS model. Then* Code *is polyspace* (ℓ, q)*-leak/tamper simulatable for* $\mathcal{G}$.

Proof. Similarly to the proof of Theorem 2, we describe a simulator S running in experiment $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$ which simulates the view of adversary A in the experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$. The simulator is given black-box access to A (which can issue `Tamper`, `Leak`, and `Eval` queries) and to the reactive functionality $\mathcal{G}(st,\cdot)$ with initial state st. Simulator S sets $(\mathcal{M}_0, \mathcal{M}_1)$ to $(0^s, 0^s)$ and keeps a $\log(k)$-bit counter $\langle\gamma\rangle$ (initially set to zero). Then, it samples the common reference string $\Omega \leftarrow \mathrm{Init}(1^k)$, simulates the public values, and proceeds as follows:

- *Normal Mode.* Given state $(\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle \mathtt{Eval}, x_j\rangle$: Upon input the j-th evaluation query, forward the input x_j to $\mathcal{G}(st_{\mathsf{curr}},\cdot)$ and send the reply y_j back to A. (Here st_{curr} is the current state, as the functionally updates the state at each invocation.) Increment the counter $\langle\gamma\rangle$.
  - $\langle \mathtt{Tamper}, (\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)})\rangle$: Upon input the j-th tampering query sample a random encoding of 0^{2k}, namely $(Z_0^{(j)}, Z_1^{(j)}) \leftarrow \mathrm{Encode}(\Omega, 0^{2k})$, and set $\mathcal{M}_b[1\ldots,n] \leftarrow Z_b^{(j)}$ for $b \in \{0,1\}$. Hence, compute $\mathcal{M}'_b \leftarrow \mathsf{T}_b^{(j)}(\mathcal{M}_b)$. In case $(\mathcal{M}'_0[1\ldots n], \mathcal{M}'_1[1\ldots n]) = (Z_0^{(j)}, Z_1^{(j)})$ then continue in the current mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\mathcal{M}'_0, \mathcal{M}'_1)$. Otherwise go to the overwritten mode defined below with the new state $(\mathcal{M}'_0, \mathcal{M}'_1)$.
  - $\langle \mathtt{Leak}, (\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})\rangle$: Upon input the j-th leakage query, compute $\Lambda_b^{(j)} = \mathsf{L}_b^{(j)}(Z_b^{(j)})$ for $b \in \{0,1\}$ and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- *Overwritten Mode.* Given state $(\mathcal{M}'_0, \mathcal{M}'_1)$ and current counter $\langle\gamma\rangle$, while A continues issuing queries, answer as follows:
  - Let $\tau = (\mathcal{M}'_0, \mathcal{M}'_1)$. Simulate the hardened functionality $\mathcal{G}^{\mathsf{Code}}(\tau,\cdot)$ (with counter set to $\langle\gamma\rangle$ and self-destruct bit $\varphi = 0$) and answer all `Eval`, `Leak` and `Tamper` queries as the real experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(\tau,\cdot)}(k)$ would do.
- When A halts and outputs $\mathsf{view}_{\mathsf{A}} = (\Omega; ((x_1, y_1), (x_2, y_2), \ldots); ((\Lambda_0^{(1)}, \Lambda_1^{(1)}), (\Lambda_0^{(2)}, \Lambda_1^{(2)}), \cdots))$, set $\mathsf{view}_{\mathsf{S}} = \mathsf{view}_{\mathsf{A}}$ and output $\mathsf{view}_{\mathsf{S}}$ as output of $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$.

We show the following lemma about S's simulation, which directly implies the theorem.

Lemma 9. *Let* S *be defined as above. Then for all PPT adversaries* A *and all* $st \in \{0,1\}^k$*, the following holds:*

$$\left\{\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)\right\}_{k\in\mathbb{N}} \approx_c \left\{\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)\right\}_{k\in\mathbb{N}}.$$

Proof. By contradiction, assume that there exists a PPT distinguisher D, a PPT adversary A and some state $st \in \{0,1\}^k$ such that:

$$\left|\mathbb{P}\left[\mathsf{D}(\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)) = 1\right]\right| \geq \epsilon, \tag{8}$$

where $\epsilon(k)$ is some non-negligible function of the security parameter k.

Similar to [23], without loss of generality we define a round as a sequence of queries which contains at most one execute query, one tampering query and one leakage query. For all $i = 1, \ldots, q$, consider the following hybrid experiments where in each hybrid the simulator does a modified normal mode and the same overwritten mode:

Experiment $\mathsf{HYB}_{\mathsf{S},st}^{(i)}(k)$**.** The simulator S starts with generating the public values and proceeds as follows.

- In the first i round it does exactly the same as in the experiment $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$.
- Starting from the $i+1$-th round if the simulator is already in the overwritten mode, then continue simulating in the overwritten mode. Otherwise, let st_{curr} be the current state of the functionality. Proceed with the following modified normal mode:
  - $\langle \mathtt{Eval}, x_j \rangle$: Run $(st_{\mathsf{new}}, y_j) \leftarrow \mathcal{G}(st_{\mathsf{cur}}, x_j)$, set $st_{\mathsf{cur}} := st_{\mathsf{new}}$ and give y_j to A. Increment the counter $\langle \gamma \rangle$.
  - $\langle \mathtt{Tamper}, (\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)}) \rangle$: Sample a random encoding of st_{cur}, namely $(Z_0^{(j)}, Z_1^{(j)}) \leftarrow \mathrm{Encode}(\Omega, st_{\mathsf{cur}}||\langle \gamma \rangle)$, and set $\mathcal{M}_b[1\ldots, n] \leftarrow Z_b^{(j)}$ for $b \in \{0,1\}$. Hence, compute $\mathcal{M}'_b \leftarrow \mathsf{T}_b^{(j)}(\mathcal{M}_b)$. In case $(\mathcal{M}'_0[1\ldots n], \mathcal{M}'_1[1\ldots n]) = (Z_0^{(j)}, Z_1^{(j)})$ then continue in the current mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\mathcal{M}'_0, \mathcal{M}'_1)$. Otherwise go to the overwritten mode with state $(\mathcal{M}'_0, \mathcal{M}'_1)$.
  - $\langle \mathtt{Leak}, (\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)}) \rangle$: Compute $\Lambda_b^{(j)} = \mathsf{L}_b^{(j)}(Z_b^{(j)})$ for $b \in \{0,1\}$ and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.

Note that $\mathsf{HYB}_{\mathsf{S},st}^{(0)}(k)$ is distributed exactly as experiment $\mathsf{REAL}_{\mathsf{A}}^{\mathcal{G}^{\mathsf{Code}}(st',\cdot)}(k)$, and $\mathsf{HYB}_{\mathsf{S},st}^{(q)}(k)$ is distributed exactly as $\mathsf{IDEAL}_{\mathsf{S}}^{\mathcal{G}(st,\cdot)}(k)$. Hence, by a standard argument, Eq (8) implies

$$\left|\mathbb{P}\left[\mathsf{D}(\mathsf{HYB}_{\mathsf{S},st}^{(i)}(k)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathsf{HYB}_{\mathsf{S},st}^{(i+1)}(k)) = 1\right]\right| \geq \epsilon/q, \tag{9}$$

a non-negligible quantity. We will exploit the advantage of D in distinguishing between $\mathsf{HYB}_{\mathsf{S},st}^{(i)}(k)$ and $\mathsf{HYB}_{\mathsf{S},st}^{(i+1)}(k)$, to build another distinguisher D′ and an adversary A′ breaking CNMLR property of Code. The distinguisher works similar to the proof of [23, Lemma 22] with some technical differences, that we emphasize below.

Given the common reference string Ω, the distinguisher D′ initializes a flag STOP to FALSE and runs the simulator S of hybrid $\mathsf{HYB}_{\mathsf{S},st}^{(i)}$ until round i to obtain the current state st_{cur} and the current memory content $(\mathcal{M}'_0, \mathcal{M}'_1)$. (Note that D′ can do this as it is given st as advice.) Hence, D′ outputs the message pair $(0^{2k}, st_{\mathsf{cur}}||\langle i \rangle)$ and is given access to oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ and $\mathcal{O}^{\ell}(X_0)$, $\mathcal{O}^{\ell}(X_1)$, where (X_0, X_1) is either an encoding of 0^{2k} or an encoding of $st_{\mathsf{cur}}||\langle i \rangle$. For all rounds $i+1 \leq j \leq q$, the distinguisher handles A's queries as follows:

- $\langle \mathtt{Tamper}, (\mathsf{T}_0^{(j)}, \mathsf{T}_1^{(j)}) \rangle$: The distinguisher will access the oracle $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$. Similarly to the proof of Theorem 2, D′ cannot simply forward the pair of functions to the oracle, so it modifies it as follows. Define the functions $\mathsf{T}_{\mathsf{in}} : \{0,1\}^n \to \{0,1\}^s$ and $\mathsf{T}_{\mathsf{out}} : \{0,1\}^s \to \{0,1\}^n$ as $\mathsf{T}_{\mathsf{in}}(x) = (x||\mathcal{M}'_b[n+1,\ldots,s])$ and $\mathsf{T}_{\mathsf{out}}(x||x') = x$, for any $x \in \{0,1\}^n$ and

$x' \in \{0,1\}^{s-n}$; here $(\mathcal{M}'_0, \mathcal{M}'_1)$ is the above state obtained at the end of round i. The distinguisher D' queries $\mathcal{O}_{\mathsf{cnm}}((X_0, X_1), \cdot)$ with the function pair $(\tilde{\mathsf{T}}_0^{(j)}, \tilde{\mathsf{T}}_1^{(j)})$ where each $\tilde{\mathsf{T}}_b^{(j)}$ is defined as $\tilde{\mathsf{T}}_b^{(j)} := \mathsf{T}_{\mathsf{out}} \circ \mathsf{T}_b^{(j)} \circ \mathsf{T}_b^{(j-1)} \circ \ldots \circ \mathsf{T}_b^{(i+1)} \circ \mathsf{T}_{\mathsf{in}}$ for $b \in \{0,1\}$.

In case the oracle returns $\perp$, then D' sets STOP to TRUE. In case the oracle returns $\mathsf{same}^\star$, then D' does nothing. Otherwise, in case the oracle returns an encoding (X'_0, X'_1), then D' checks that the last k bits of $\mathrm{Decode}(\Omega, (X'_0, X'_1))$ equal $\langle j \rangle$; if that is not the case, it sets STOP to TRUE.

- $\langle \mathtt{Leak}, (\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)}) \rangle$: Upon input leakage functions $(\mathsf{L}_0^{(j)}, \mathsf{L}_1^{(j)})$, the distinguisher D' defines $(\tilde{\mathsf{L}}_0^{(j)}, \tilde{\mathsf{L}}_1^{(j)})$ (in a similar way as above), forwards the functions to $\mathcal{O}^\ell(X_0)$, $\mathcal{O}^\ell(X_1)$ and sends the answer from the oracles back to D.

- $\langle \mathtt{Eval}, x_j \rangle$: Upon input an evaluation query for value x_j, the distinguisher D' checks first that STOP equals FALSE. If this is not the case, then D' returns $\perp$ to D. Otherwise, it runs $(st_{\mathsf{new}}, y_j) \leftarrow \mathcal{G}(st_{\mathsf{cur}}, x_j)$, set $s_{\mathsf{cur}} := s_{\mathsf{new}}$ and give y_j to D.

Define the following event FAIL: the event becomes true in case the simulation has entered already the overwritten mode at round $i+1$, or the simulation is in normal mode but D never issues `Tamper` or `Leak` commands after round i. Now, if FAIL happens, D' gives-up and outputs a random guess. Otherwise it outputs whatever D does.

It is easy to see that $\mathsf{HYB}_{\mathsf{S},st}^{(i)}(k)$ and $\mathsf{HYB}_{\mathsf{S},st}^{(i+1)}(k)$ are statistically close in case the simulation enters the overwritten mode before round i. It follows that there is a non-negligible chance that the simulation is in normal mode at round i and moreover D will either issue a `Leak` or a `Tamper` command in some later round. The above argument implies that: (i) D' correctly simulates either the distribution of $\mathsf{HYB}_{\mathsf{S},st}^{(i)}(k)$ or $\mathsf{HYB}_{\mathsf{S},st}^{(i+1)}(k)$ (depending on the value in the target encoding (X_0, X_1)), conditioning on $\overline{\text{FAIL}}$; (ii) $\mathbb{P}\left[\overline{\text{FAIL}}\right] \geq \alpha$, where α is non-negligible; (iii) D' has advantage $1/2$ conditioning on FAIL. Let $st^* = st_{\mathsf{cur}} || \langle i \rangle$. We have obtained

$$
\begin{aligned}
&\left| \mathbb{P}\left[\mathsf{D}'(\mathsf{Tamper}_{\mathsf{A}',0^{2k}}^{\mathrm{cnmlr}}(k)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathsf{Tamper}_{\mathsf{A}',st^*}^{\mathrm{cnmlr}}(k)) = 1\right] \right| \\
&= \Bigg| \mathbb{P}[\text{FAIL}]\, \mathbb{P}\left[\mathsf{D}'(\mathsf{Tamper}_{\mathsf{A},0^{2k}}^{\mathrm{cnmlr}}(k)) = 1 | \text{FAIL}\right] + \mathbb{P}\left[\overline{\text{FAIL}}\right] \mathbb{P}\left[\mathsf{D}'(\mathsf{Tamper}_{\mathsf{A}',0^{2k}}^{\mathrm{cnmlr}}(k)) = 1 | \overline{\text{FAIL}}\right] \\
&\qquad - \mathbb{P}[\text{FAIL}]\, \mathbb{P}\left[\mathsf{D}'(\mathsf{Tamper}_{\mathsf{A}',st^*}^{\mathrm{cnmlr}}(k)) = 1 | \text{FAIL}\right] - \mathbb{P}\left[\overline{\text{FAIL}}\right] \mathbb{P}\left[\mathsf{D}'(\mathsf{Tamper}_{\mathsf{A}',st^*}^{\mathrm{cnmlr}}(k)) = 1 | \overline{\text{FAIL}}\right] \Bigg| \\
&= \left| \mathbb{P}\left[\mathsf{D}(\mathsf{HYB}_{\mathsf{S},st}^{(i)}(k)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathsf{HYB}_{\mathsf{S},st}^{(i+1)}(k)) = 1\right] \right| \cdot \mathbb{P}\left[\overline{\text{FAIL}}\right] \\
&\geq \epsilon / q \cdot \alpha,
\end{aligned}
$$

a non-negligible quantity. This concludes the proof. □

□

Acknowledgments

Pratyay acknowledges support from a European Research Commission Starting Grant (no. 279447) and the CTIC and CFEM research center. Part of this work was done while this author was at the University of Warsaw and was supported by the WELCOME/2010-4/2 grant founded within the framework of the EU Innovative Economy Operational Programme.

Most of the work was done while Daniele was at Aarhus University, supported by the Danish Council for Independent Research via DFF Starting Grant 10-081612.

References

[1] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:81, 2013.

[2] Per Austrin, Kai-Min Chung, Mohammad Mahmoody, Rafael Pass, and Karn Seth. On the (im)possibility of tamper-resilient cryptography: Using fourier analysis in computer viruses. *IACR Cryptology ePrint Archive*, 2013:194, 2013.

[3] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, pages 486–503, 2011.

[4] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, pages 491–506, 2003.

[5] Mihir Bellare, Kenneth G. Paterson, and Susan Thomson. RKA security beyond the linear barrier: IBE, encryption and signatures. In *ASIACRYPT*, pages 331–348, 2012.

[6] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[7] Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:118, 2013.

[8] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. *IACR Cryptology ePrint Archive*, 2013:565, 2013.

[9] Seung Geol Choi, Aggelos Kiayias, and Tal Malkin. BiTR: Built-in tamper resilience. In *ASIACRYPT*, pages 740–758, 2011.

[10] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.

[11] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *CRYPTO*, pages 533–551, 2012.

[12] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. In *ASIACRYPT (2)*, pages 140–160, 2013.

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

[14] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In *ASIACRYPT*, pages 702–721, 2011.

[15] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *CRYPTO (2)*, pages 239–257, 2013.

[16] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[17] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[18] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. *IACR Cryptology ePrint Archive*, 2013:702, 2013.

[19] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP (1)*, pages 391–402, 2011.

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

[21] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

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

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

[24] Krzysztof Pietrzak. Subspace lwe. In *TCC*, pages 548–563, 2012.

[25] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.