J Cryptol (2020) 33:2034–2077
https://doi.org/10.1007/s00145-020-09362-z

Continuously Non-malleable Codes in the Split-State Model

Sebastian Faust
Technical University of Darmstadt, Darmstadt, Germany

Pratyay Mukherjee
Visa Research, Palo Alto, USA

Jesper Buus Nielsen
Aarhus University, Aarhus, Denmark

Daniele Venturi
Sapienza University of Rome, Rome, Italy
venturi@di.uniroma1.it

Communicated by Jonathan Katz.

Received 13 March 2015 / Revised 3 July 2020
Online publication 24 July 2020

Abstract. Non-malleable codes (Dziembowski et al., ICS'10 and J. ACM'18) are a natural relaxation of error correcting/detecting codes with useful applications in cryptography. Informally, a code is non-malleable if an adversary trying to tamper with an encoding of a message can only leave it unchanged or modify it to the encoding of an unrelated value. This paper introduces *continuous* non-malleability, a generalization of standard non-malleability where the adversary is allowed to tamper *continuously* with the same encoding. This is in contrast to the standard definition of non-malleable codes, where the adversary can only tamper a *single* time. The only restriction is that after the first invalid codeword is ever generated, a special self-destruct mechanism is triggered and no further tampering is allowed; this restriction can easily be shown to be necessary. We focus on the split-state model, where an encoding consists of two parts and the tampering functions can be arbitrary as long as they act independently on each part. Our main contributions are outlined below.

- We show that continuous non-malleability in the split-state model is impossible without relying on computational assumptions.
- We construct a computationally secure split-state code satisfying continuous non-malleability in the common reference string (CRS) model. Our scheme can be instantiated assuming the existence of collision-resistant hash functions and (doubly enhanced) trapdoor permutations, but we also give concrete instantiations based on standard number-theoretic assumptions.
- We revisit the application of non-malleable codes to protecting arbitrary cryptographic primitives against related-key attacks. Previous applications of non-malleable codes in this setting required *perfect erasures* and the adversary to be restricted in memory. We show that continuously non-malleable codes allow to avoid these restrictions.

Keywords. Non-malleable codes, Tamper resistance, Split-state model.

1. Introduction

Physical attacks targeting cryptographic implementations instead of breaking the black-box security of the underlying algorithm are amongst the most severe threats for cryptographic systems. A particularly important attack on implementations is the so-called *tampering attack*, where 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. [22], 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 [4,11,55,64,70,76,84]) developed new cryptographic techniques to tackle tampering attacks. Non-malleable codes introduced by Dziembowski, Pietrzak and Wichs [55,56] are an important approach to achieve this goal. Intuitively a code is non-malleable w.r.t. a family of tampering functions $\mathcal{F}$ if the message contained in a codeword modified via a function $f \in \mathcal{F}$ 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, we store its encoding and decode it each time the functionality wants to access the key. As long as the adversary can only apply tampering functions from the family $\mathcal{F}$, 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, after which it obtains the decoded output. In this work we introduce *continuously non-malleable codes*, where non-malleability is guaranteed even if the adversary continuously applies functions from the family $\mathcal{F}$ to the *same* codeword. We show that our new security notion is not only a natural extension of the standard definition, but moreover allows to protect against tampering attacks in important settings where earlier constructions fall short to achieve security.

1.1. *Continuous Non-malleability*

A code consists of two polynomial-time algorithms $\Gamma = (\mathsf{Enc}, \mathsf{Dec})$ satisfying the following correctness property: For all messages $m \in \mathcal{M}$, it holds that $\mathsf{Dec}(\mathsf{Enc}(m)) = m$ (with probability one over the randomness of the encoding). To define non-malleability for a function family $\mathcal{F}$, consider the following experiment $\mathbf{Tamper}^{\mathcal{F}}_{\mathsf{A},\Gamma}(\lambda, b)$ with hidden bit $b \in \{0, 1\}$ and featuring a (possibly unbounded) adversary A.

1. The adversary chooses two messages $(m_0, m_1) \in \mathcal{M}^2$.
2. The challenger computes a target codeword $c \leftarrow\!\!\$\ \mathsf{Enc}(m_b)$ using the encoding procedure.

3. The adversary picks a tampering function $f \in \mathcal{F}$, which yields a tampered codeword $\tilde{c} = f(c)$. Hence:
   - If $\tilde{c} = c$, then the attacker receives a special symbol $\diamond$ denoting that tampering did not modify the target codeword;
   - Else, the adversary is given $\mathsf{Dec}(\tilde{c}) \in \mathcal{M} \cup \{\perp\}$, where $\perp$ is a special symbol denoting that the tampered codeword $\tilde{c}$ is invalid.
4. The attacker outputs a guess $b' \in \{0, 1\}$.

A code Γ is said to be statistically (one-time) non-malleable w.r.t. $\mathcal{F}$ if for all attackers $|\mathbb{P}\left[b' = b\right] - 1/2|$ is negligible in the security parameter. This is equivalent to saying that the experiments $\mathbf{Tamper}^{\mathcal{F}}_{\Gamma,\mathsf{A}}(\lambda, 0)$ and $\mathbf{Tamper}^{\mathcal{F}}_{\Gamma,\mathsf{A}}(\lambda, 1)$ are statistically close. Computational non-malleability can be obtained by simply relaxing the above guarantee to computational indistinguishability (for all PPT adversaries).

To define continuously non-malleable codes, we allow the adversary to repeat[1] step 3 from the above game a polynomial number of times, where in each iteration the attacker can adaptively choose a tampering function $f^{(i)} \in \mathcal{F}$. We emphasize that this change of the tampering game allows the adversary to tamper continuously with the target codeword c. As shown by Gennaro et al. [70], such a strong security notion is impossible to achieve without further assumptions. To this end, we rely on the following *self-destruct* mechanism: Whenever in step 3 the experiment detects an invalid codeword and returns $\perp$ for the first time, all future tampering queries will automatically be answered with $\perp$. This is a rather mild assumption as it can, for instance, be implemented using a public, one-time writable, untamperable bit.

From non-malleable codes to tamper resilience As discussed above, the main application of non-malleable codes is to protecting cryptographic schemes against tampering with the secret key [55,56,84]. Consider a reactive functionality F with secret state σ. Using a non-malleable code, earlier work showed how to transform the functionality (F, σ) into a so-called hardened functionality $(\hat{\mathsf{F}}, c)$ that is secure against memory tampering. The transformation works as follows: Initially, c is set to $\mathsf{Enc}(\sigma)$. Then, each time $\hat{\mathsf{F}}$ is executed on input x, the transformed functionality reads the encoding c from the memory, decodes it to obtain $\sigma = \mathsf{Dec}(c)$, and runs the original functionality $\mathsf{F}(\sigma, x)$ obtaining an output y and a new state σ'. Finally, it erases the memory and overwrites c with $\mathsf{Enc}(\sigma')$.

Besides executing evaluation queries, the adversary can issue tampering queries $f^{(i)} \in \mathcal{F}$. The effect of such a query is to overwrite the current codeword c stored in the memory with a tampered codeword $\tilde{c} = f^{(i)}(c)$, so that the functionality $\hat{\mathsf{F}}$ will take $\tilde{c}$ as input when answering the next evaluation query. The first time that $\mathsf{Dec}(\tilde{c}) = \perp$, the functionality $\hat{\mathsf{F}}$ sets the memory to a dummy value (which essentially results in a self-destruct).

The above transformation guarantees *continuous* tamper resilience even if the underlying non-malleable code is secure only against one-time tampering. This security

[1]Our actual definition is stronger, in that as long as the tampered codeword $\tilde{c}$ is valid, and $\tilde{c} \neq c$, the adversary is even given $\tilde{c}$ (and not just the corresponding decoded message). See Sect. 3 for a discussion.

"boost" is achieved by re-encoding the secret state after each execution of $\hat{\mathsf{F}}$. As one-time non-malleability suffices for the above cryptographic application, one may ask why we need continuously non-malleable codes. Besides being a natural generalization of the standard non-malleability notion, our new definition has several important advantages 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 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.

- *It is unnatural* In many cases, secret data, e.g., a cryptographic key, are 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.
- *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.

The second issue may be solved employing locally decodable and updatable non-malleable codes [42–46], which intuitively allow to access/update a portion of the message without reading/modifying the entire codeword. Using our new notion of continuously non-malleable codes we can avoid both issues in one go, and achieve continuous tamper resilience without using *erasures* or relying on inefficient solutions that encode the *entire* state.

Stateless tamper-resilient transformations To achieve tamper resistance from one-time 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. Thanks to continuously non-malleable codes we get continuous security without the need to refresh the encoding after each usage. This is particularly useful when the underlying primitive that we want to protect is stateless itself (e.g., in the case of any standard block-cipher construction that typically keeps the same key).

Using continuously 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 Sect. 5.

1.2. *Our Contribution*

Our main contribution is the first construction of *continuously non-malleable codes* in the split-state model, first introduced in the leakage setting [50,54]. Various recent works study the split-state model for non-malleable codes [3,4,30,53,84] (see more details on related work in Sect. 1.3). In the split-state tampering model, the codeword consists of two halves c_0 and c_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 f_0 and f_1 and replaces the codeword (c_0, c_1) with $(f_0(c_0), f_1(c_1))$. Similar to the earlier work of Liu and Lysyanskaya [84], our construction assumes a public untamperable common reference string (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-time split-state non-malleable codes (without random oracles) was given by Liu and Lysyanskaya [84]. At a high-level their construction encrypts the message m with a leakage-resilient encryption scheme, and generates a non-interactive zero-knowledge (NIZK) proof of knowledge showing that (i) the public/secret key of the PKE are valid, and (ii) the ciphertext is an encryption of m under the public key. Then, c_0 is set to the secret key while c_1 holds the corresponding public key, the ciphertext, and the 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 continuously non-malleable codes allows the adversary to interact in the following game. First, we sample an encoding $(\hat{c}_0, \hat{c}_1)$ of m, and then we repeat the following process a polynomial number of times.

1. The adversary submits two polynomial-time computable functions (f_0, f_1) resulting in a tampered codeword $(\tilde{c}_0, \tilde{c}_1) = (f_0(\hat{c}_0), f_1(\hat{c}_1))$.
2. We consider three different cases:
   - If $(\tilde{c}_0, \tilde{c}_1) = (\hat{c}_0, \hat{c}_1)$, then return $\diamond$.
   - Else, let $\tilde{m}$ be the decoding of $(\tilde{c}_0, \tilde{c}_1)$. If $\tilde{m} \neq \perp$, then return $\tilde{m}$.
   - Else, if $\tilde{m} = \perp$ self-destruct and terminate the experiment.

The main observation that enables the attack against the scheme of [84] is as follows: For a fixed (but adversarially chosen) part c_0 it is easy to come up with two corresponding parts c_1 and c_1' such that both (c_0, c_1) and (c_0, c_1') form a *valid* encoding (i.e., a codeword whose decoding does not yield $\perp$). Suppose further that decoding (c_0, c_1) yields a message m that is different from the message m' obtained by decoding (c_0, c_1'). Then, under continuous tampering, the adversary may permanently replace the original encoding $\hat{c}_0$ with c_0, while depending on whether the i-th bit of $\hat{c}_1$ being 0 or 1 either replace $\hat{c}_1$ by c_1 or c_1'. This allows to recover the entire $\hat{c}_1$ by just n tampering queries (where n is the size of $\hat{c}_1$). Once $\hat{c}_1$ is known to the adversary, it is easy[2] to tamper with $\hat{c}_0$ in a way that depends on the message $\hat{m}$ corresponding to $(\hat{c}_0, \hat{c}_1)$.

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 [3–5,53] 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 applied to the tamper-protection application of cryptographic functionalities.

Uniqueness The attack above exploits that for a fixed known part c_0 it is easy to come up with two valid parts c_1, c_1'. For the encoding of [84] this is indeed easy to achieve. If the secret key c_0 is known, it is easy to come up with two valid parts c_1, c_1': just encrypt two arbitrary messages m, m' such that $m \neq m'$, and generate the corresponding proofs. The above weakness motivates a new property that non-malleable codes shall satisfy in order to achieve continuous non-malleability. We call this property *uniqueness*, which informally guarantees that for any (adversarially chosen) valid encoding (c_0, c_1) it be computationally hard to come up with $c_1' \neq c_1$ such that (c_0, c_1') forms a valid encoding.[3] Clearly, the uniqueness property prevents the above described attack, and hence is a crucial requirement for continuous non-malleability in the split-state model.

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 [50,52] for the split-state model (one may view this as a generalization of the leakage-resilient PKE used in [84]), a collision-resistant hash function, and (similar to [84]) 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 Sect. 4.1 for the details of our construction, and to Sect. 4.2 for an outline of the proof.

Tamper resilience for stateless and stateful primitives We can use our new construction of continuously 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 to protect stateless primitives, then the resulting scheme remains stateless. This solves an open problem posed by Dziembowski, Pietrzak and Wichs

[2]For example, hard-wire $\hat{c}_1$, decode $(\hat{c}_0, \hat{c}_1)$, and overwrite $\hat{c}_0$ with garbage if the first bit of $\hat{m}$ is zero (otherwise leave it unchanged).

[3]Of course, a similar guarantee must hold if we fix the right part c_1 of the encoding.

[55,56]. Notice that while we do not need to keep any secret state, the transformed functionality requires one bit of state to activate the self-destruct mechanism. This bit can be *public*, but must be untamperable, and can, for instance, be implemented through a one-time writable memory. As shown in the work of Gennaro et al. [70], 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 our setting, where data are never erased, an adversary can always reset the functionality to a previous valid state. To avoid this, our transformation uses an untamperable *public* counter[4] that helps us detecting whenever the attacker tries to reset the functionality to a previous state (in which case, a self-destruct is triggered). We notice that such an untamperable counter is necessary, as otherwise there is no way to protect against the above resetting attack.[5]

Bounded leakage resilience As an additional contribution, we show that our code is also secure against bounded leakage attacks. This is similar to the works of [53,84] who also consider bounded leakage resilience of their encoding scheme. Furthermore, as we prove, bounded leakage resilience is also inherited by functionalities that are protected using 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.

1.3. *Related Work*

Constructions of non-malleable codes Besides showing feasibility by a probabilistic argument, Dziembowski et al. [55,56] also built non-malleable codes for bit-wise tampering (later improved in [9,10,32,34,36]) and gave a construction in the split-state model using a random oracle. This result was followed by [33], which proposed non-malleable codes that are secure against block-wise tampering. The first construction of non-malleable codes in the split-state model without random oracles was given by Liu and Lysyanskaya [84], in the computational setting, assuming an untamperable CRS. Several follow-up works focused on constructing split-state non-malleable codes in the information-theoretic setting [1,3,4,7,28,53].

See also [9,10,12–17,24,25,27,31,37,45,57,64,67,77,79–82] for other recent advances on the construction of non-malleable codes. We also notice that the work of Gennaro et al. [70] proposed a generic method that allows to protect arbitrary computation against continuous tampering attacks, without requiring erasures. We refer the reader to [55,56] for a more detailed comparison between non-malleable codes and the solution of [70].

[4]Note that a counter only requires a logarithmic (in the security parameter) number of bits.

[5]This kind of threat is sometimes also known under the name of rewind attacks in the literature [45,46].

Other works on tamper resilience A large body of work shows how to protect specific cryptographic schemes against tampering attacks (see [18,20,21,47,48,78,86] 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.

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 [39,40,66, 69,76,83]. 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.

Subsequent work A preliminary version of this paper appeared as [62]. Subsequent work showed that our impossibility result on information-theoretic continuous non-malleability can be circumvented in weaker tampering models, such as bit-wise tampering [34–36], high-min-entropy and few-fixed points tampering [77], 8-split-state tampering [6], permutations and overwrites [49], and space-bounded tampering [29,61], or by assuming that the number of tampering queries is a priori fixed [26] and that tampering is persistent [8].

Continuously non-malleable codes have also been used to protect Random Access Machines against tampering attacks with the memory and the computation [63], and to obtain domain extension for non-malleable public-key encryption [34,36,49].

A recent work by Dachman-Soled and Kulkarni [41] shows that the strong flavor of continuous non-malleability in the split-state model considered in this paper (sometimes known as *super* non-malleability [64,65]) is impossible to achieve in the plain model (i.e., without assuming an untamperable CRS). On the other hand, Ostrovsky et al. [85] proved that continuous *weak* non-malleability in the split-state model (i.e., the standard flavor of non-malleability in which the attacker only learns the output of the decoding corresponding to each tampered codeword) is possible in the plain model (assuming one-to-one one-way functions). Faonio and Venturi [58,59] further consider continuously non-malleable split-state codes with a special refreshing procedure allowing to update codewords.

Finally, the concept of continuous non-malleability has also been recently studied in the more general setting of non-malleable secret sharing [23,60,71].

2. Preliminaries

2.1. *Notation*

For a string x, we denote its length by $|x|$; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the number of elements in $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow\!\!\$\ \mathcal{X}$. When A is a randomized algorithm, we write $y \leftarrow\!\!\$\ \mathsf{A}(x)$ to denote a run of A on input x (and implicit random coins r) and output y; the value y is a random variable, and $\mathsf{A}(x; r)$ denotes a run of A on input x and randomness r. A randomized algorithm A is *probabilistic polynomial-time* (PPT) if for any input $x, r \in \{0,1\}^*$ the computation of $\mathsf{A}(x;r)$ terminates in a polynomial number of steps (in the size of the input). For a PPT algorithm A, we denote by $\langle \mathsf{A} \rangle$ its description using poly-many bits.

Negligible functions We denote with $\lambda \in \mathbb{N}$ the security parameter. A function p is a polynomial, denoted $p(\lambda) \in \mathtt{poly}(\lambda)$, if $p(\lambda) \in O(\lambda^c)$ for some constant $c > 0$. A function $\varepsilon : \mathbb{N} \rightarrow [0,1]$ is negligible in the security parameter (or simply negligible) if it vanishes faster than the inverse of any polynomial in λ, i.e., $\varepsilon(\lambda) \in O(1/\lambda^c)$ for every constant $c > 0$. We often write $\varepsilon(\lambda) \in \mathtt{negl}(\lambda)$ to denote that $\varepsilon(\lambda)$ is negligible.

Unless stated otherwise, throughout the paper, we implicitly assume that the security parameter is given as input (in unary) to all algorithms.

Random variables For a random variable $\mathbf{X}$, we write $\mathbb{P}[\mathbf{X} = x]$ for the probability that $\mathbf{X}$ takes on a particular value $x \in \mathcal{X}$ (with $\mathcal{X}$ being the set where $\mathbf{X}$ is defined). Given two ensembles $\mathbf{X} = \{\mathbf{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathbf{Y} = \{\mathbf{Y}_\lambda\}_{\lambda \in \mathbb{N}}$, we write $\mathbf{X} \overset{\mathrm{s}}{\approx} \mathbf{Y}$ (resp. $\mathbf{X} \overset{\mathrm{c}}{\approx} \mathbf{Y}$) to denote that $\mathbf{X}$ and $\mathbf{Y}$ are statistically (resp. computationally) close, i.e., for all unbounded (resp. PPT) distinguishers D:

$$\Delta^{\mathsf{D}}(\mathbf{X};\mathbf{Y}) := \left|\mathbb{P}\left[\mathsf{D}(\mathbf{X}_\lambda) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathbf{Y}_\lambda) = 1\right]\right| \in \mathtt{negl}(\lambda).$$

If the above distance is zero, we say that $\mathbf{X}$ and $\mathbf{Y}$ are identically distributed, denoted $\mathbf{X} \equiv \mathbf{Y}$.

We extend the notion of computational indistinguishability to the case of interactive experiments (a.k.a. games) featuring an adversary A. In particular, let $\mathbf{G}_{\mathsf{A}}(\lambda)$ be the random variable corresponding to the output of A at the end of the experiment, where A outputs a decision bit. Given two experiments $\mathbf{G}_{\mathsf{A}}(\lambda, 0)$ and $\mathbf{G}_{\mathsf{A}}(\lambda, 1)$, we write $\{\mathbf{G}_{\mathsf{A}}(\lambda, 0)\}_{\lambda \in \mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{G}_{\mathsf{A}}(\lambda, 1)\}_{\lambda \in \mathbb{N}}$ if for all PPT A it holds that

$$\left|\mathbb{P}[\mathbf{G}_{\mathsf{A}}(\lambda, 0) = 1] - \mathbb{P}[\mathbf{G}_{\mathsf{A}}(\lambda, 1) = 1]\right| \in \mathtt{negl}(\lambda).$$

The above naturally generalizes to statistical distance (in case of unbounded adversaries).

2.2. *Collision-Resistant Hashing*

A family of hash functions $\Pi := (\mathsf{Gen}, \mathsf{Hash})$ is a pair of efficient algorithms specified as follows: (i) The randomized algorithm Gen takes as input the security parameter and outputs a hash-key hk. (ii) The deterministic algorithm Hash takes as input the hash-key hk and a value $x \in \{0,1\}^*$, and outputs a value $y \in \{0,1\}^\lambda$.

Definition 1. (Collision resistance) Let $\Pi = (\mathsf{Gen}, \mathsf{Hash})$ be a family of hash functions. We say that Π is collision resistant if for all PPT adversaries A there exists a negligible function $\nu : \mathbb{N} \rightarrow [0,1]$ such that:

$$\mathbb{P}\left[x \neq x' \wedge \mathsf{Hash}(hk, x) = \mathsf{Hash}(hk, x') : \ (x, x') \leftarrow\!\!\$\ \mathsf{A}(hk), hk \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)\right] \leq \nu(\lambda).$$

2.3. *Non-interactive Zero Knowledge*

Let $\mathcal{R}$ be an NP relation, corresponding to an NP language $\mathcal{L}$. A non-interactive argument system for $\mathcal{R}$ is a tuple of efficient algorithms $\Pi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$ specified as follows: (i) The randomized algorithm CRSGen takes as input the security parameter and outputs a common reference string ω; (ii) The randomized algorithm $\mathsf{Prove}(\omega, \phi, (x, w))$, given $(x, w) \in \mathcal{R}$ and a label $\phi \in \{0, 1\}^*$, outputs an argument π; (iii) The deterministic algorithm $\mathsf{Ver}(\omega, \phi, (x, \pi))$, given an instance x, an argument π, and a label $\phi \in \{0, 1\}^*$, outputs either 0 (for "reject") or 1 (for "accept"). We say that Π is *correct* if for every $\lambda \in \mathbb{N}$, all ω as output by $\mathsf{CRSGen}(1^\lambda)$, any label $\phi \in \{0, 1\}^*$, and any $(x, w) \in \mathcal{R}$, we have that $\mathsf{Ver}(\omega, \phi, (x, \mathsf{Prove}(\omega, \phi, (x, w)))) = 1$ (with probability one over the randomness of the prover algorithm).

We define two properties of a non-interactive argument system. The first property says that honestly computed arguments do not reveal anything beyond the fact that $x \in \mathcal{L}$.

Definition 2. (Adaptive multi-theorem zero-knowledge) A non-interactive argument system Π for a relation $\mathcal{R}$ satisfies adaptive multi-theorem zero-knowledge if there exists a PPT simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$ such that the following holds:

(i) S_0 outputs ω, a simulation trapdoor ζ and an extraction trapdoor ξ.
(ii) For all PPT distinguishers D, we have that

$$\Big|\mathbb{P}\Big[\mathsf{D}^{\mathsf{Prove}(\omega,\cdot,(\cdot,\cdot))}(1^\lambda, \omega) = 1 : \ \omega \leftarrow\$\ \mathsf{CRSGen}(1^\lambda)\Big] \\ - \mathbb{P}\Big[\mathsf{D}^{\mathcal{O}_{\mathsf{sim}}(\zeta,\cdot,(\cdot,\cdot))}(1^\lambda, \omega) = 1 : \ (\omega, \zeta, \xi) \leftarrow\$\ \mathsf{S}_0(1^\lambda)\Big]\Big|$$

is negligible in λ, where the oracle $\mathcal{O}_{\mathsf{sim}}(\zeta, \cdot, (\cdot, \cdot))$ takes as input a tuple $(\phi, (x, w))$ and returns $\mathsf{S}_1(\zeta, \phi, x)$ iff $(x, w) \in \mathcal{R}$ (and otherwise it returns $\perp$).

Groth [73] introduced the concept of simulation extractability, which informally states that knowledge soundness should hold even if the adversary can see simulated arguments for possibly false statements of its choice. For our purpose, it will suffice to consider the weaker notion of true-simulation extractability, as defined by Dodis et al. [51].

Definition 3. (True-simulation extractability) Let Π be a non-interactive argument system for a relation $\mathcal{R}$, that satisfies adaptive multi-theorem zero-knowledge w.r.t. a simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$. We say that Π is *true-simulation extractable* (tSE) if there exists a PPT algorithm K such that for all PPT adversaries A, it holds that

$$\mathbb{P}\left[\begin{array}{c} \mathsf{Ver}(\omega, \phi^*, (x^*, \pi^*)) = 1 \wedge (\phi^*, x^*, \pi^*) \notin \mathcal{Q} \\ \wedge (x^*, w^*) \notin \mathcal{R} \end{array} : \begin{array}{c} (\omega, \zeta, \xi) \leftarrow\$\ \mathsf{S}_0(1^\lambda) \\ (\phi^*, x^*, \pi^*) \leftarrow\$\ \mathsf{A}^{\mathcal{O}_{\mathsf{sim}}(\zeta,\cdot,(\cdot,\cdot))}(1^\lambda, \omega) \\ w^* \leftarrow\$\ \mathsf{K}(\xi, \phi^*, (x^*, \pi^*)) \end{array}\right]$$

is negligible in λ, where the oracle $\mathcal{O}_{\mathsf{sim}}(\zeta, \cdot, (\cdot, \cdot))$ takes as input a tuple $(\phi, (x, w))$ and returns $\mathsf{S}_1(\zeta, \phi, x)$ iff $(x, w) \in \mathcal{R}$ (and otherwise it returns $\perp$), and the list $\mathcal{Q}$ contains all pairs (ϕ, x) queried by A to its oracle, along with the corresponding answer π.

Note that the above definition allows the attacker to win using a pair label/statement (ϕ^*, x^*) for which it already obtained a simulated argument, as long as the value π^* is different from the argument π obtained from the oracle. This flavor is sometimes known as *strong* tSE, and it can be obtained generically from non-strong tSE (i.e., the attacker needs to win using a fresh pair (ϕ^*, x^*)) using any strongly unforgeable one-time signature scheme [51].

2.4. *Leakage-Resilient Storage*

We recall the definition of leakage-resilient storage (LRS) from [50]. A leakage-resilient storage $\Sigma = (\mathsf{LREnc}, \mathsf{LRDec})$ is a pair of algorithms defined as follows: (i) The randomized algorithm LREnc takes as input message $m \in \{0,1\}^k$ and outputs two shares $(s_0, s_1) \in \{0,1\}^{2n}$. (ii) The deterministic algorithm LRDec takes as input shares $(s_0, s_1) \in \{0,1\}^{2n}$ and outputs a value in $\{0,1\}^k$. We say that $(\mathsf{LREnc}, \mathsf{LRDec})$ satisfies correctness if for all $m \in \{0,1\}^k$ it holds that $\mathsf{LRDec}(\mathsf{LREnc}(m)) = m$ (with probability 1 over the randomness of LREnc).

Security of LRS demands that, for any choice of messages $m_0, m_1 \in \{0,1\}^k$, it be hard for an attacker to distinguish bounded, independent, leakage from a target encoding of either m_0 or m_1. Below, we state such a property in the information-theoretic setting, as this flavor is met by known constructions. In what follows, for a leakage parameter $\ell \in \mathbb{N}$, let $\mathcal{O}^{\ell}_{\mathsf{leak}}(s, \cdot)$ be a stateful oracle taking as input functions $g : \{0,1\}^n \to \{0,1\}^*$, and returning $g(s)$ for a maximum of at most ℓ bits.

Definition 4. (Leakage-resilient storage) We call $(\mathsf{LREnc}, \mathsf{LRDec})$ an ℓ-leakage-resilient storage (ℓ-LRS for short) if it holds that

$$\left\{\mathbf{Leak}_{\Sigma,\mathsf{A}}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{Leak}_{\Sigma,\mathsf{A}}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}},$$

where for $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1)$ and $b \in \{0,1\}$ we set:

$$\mathbf{Leak}_{\Sigma,\mathsf{A}}(\lambda, b) := \left\{ b' : \begin{array}{c} (m_0, m_1, \alpha) \leftarrow\!\!\$\ \mathsf{A}_0(1^\lambda); \\ (s_0, s_1) \leftarrow\!\!\$\ \mathsf{LREnc}(m_b); \\ b' \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}^{\ell}_{\mathsf{leak}}(s_0,\cdot), \mathcal{O}^{\ell}_{\mathsf{leak}}(s_1,\cdot)}(1^\lambda, \alpha) \end{array} \right\}.$$

For our construction, we will need a slight variant of the above definition where at the end of the game the attacker is further allowed to obtain one of the two shares in full. Following [2], we refer to this variant as *augmented* LRS.

Definition 5. (Augmented leakage-resilient storage) We call $(\mathsf{LREnc}, \mathsf{LRDec})$ an *augmented* ℓ-LRS if for all $\sigma \in \{0,1\}$ it holds that

$$\left\{\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}}(\lambda, 0, \sigma)\right\}_{\lambda\in\mathbb{N}} \overset{\mathrm{s}}{\approx} \left\{\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}}(\lambda, 1, \sigma)\right\}_{\lambda\in\mathbb{N}},$$

where for $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$ and $b \in \{0, 1\}$ we set:

$$\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}}(\lambda, b, \sigma) := \left\{ b' : \begin{array}{c} (m_0, m_1, \alpha_1) \leftarrow\$ \mathsf{A}_0(1^\lambda); \\ (s_0, s_1) \leftarrow\$ \mathsf{LREnc}(m_b); \\ \alpha_2 \leftarrow\$ \mathsf{A}_1^{\mathcal{O}^{\ell}_{\mathsf{leak}}(s_0,\cdot), \mathcal{O}^{\ell}_{\mathsf{leak}}(s_1,\cdot)}(1^\lambda, \alpha_1); \\ b' \leftarrow\$ \mathsf{A}_2(1^\lambda, \alpha_2, s_\sigma) \end{array} \right\}.$$

The lemma below[6] shows the equivalence between Definitions 5 and 4 up to a loss of a single bit in the leakage parameter.

Theorem 1. *Let $\Sigma = (\mathsf{LREnc}, \mathsf{LRDec})$ be an ℓ-LRS. Then, Σ is an augmented $(\ell-1)$-LRS.*

Proof. We prove the lemma by contradiction. Assume that there exists some adversary $\mathsf{A}^+ = (\mathsf{A}_0^+, \mathsf{A}_1^+, \mathsf{A}_2^+)$ able to distinguish with non-negligible probability between $\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}^+}(\lambda, 0, \sigma)$ and $\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}^+}(\lambda, 1, \sigma)$ for some fixed $\sigma \in \{0, 1\}$ and using $\ell - 1$ bits of leakage. We construct another adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$ which can distinguish between $\mathbf{Leak}_{\Sigma,\mathsf{A}}(\lambda, 0)$ and $\mathbf{Leak}_{\Sigma,\mathsf{A}}(\lambda, 1)$ with the same distinguishing advantage, using ℓ bits of leakage. A description of A follows:

Adversary A :

- A_0 simply runs $\mathsf{A}_0^+(1^\lambda)$ and returns its output (m_0, m_1, α_1).
- A_1 first runs $\mathsf{A}_1^+(1^\lambda, \alpha_1)$ by simply forwarding its leakage queries to its own target left/right leakage oracle.
- Denote by α_2 the output of A_1^+, and let $\hat{g}_\sigma^{\mathsf{A}_2^+,\alpha_2}$ be the leakage function that hard-wires (a description of) A_2^+ and the auxiliary information α_2, and upon input s_σ returns the same as $\mathsf{A}_2^+(1^\lambda, \alpha_2, s_\sigma)$.
- A_1 forward $\hat{g}_\sigma^{\mathsf{A}_2^+,\alpha_2}$ to the leakage oracle $\mathcal{O}^{\ell}_{\mathsf{leak}}(s_\sigma, \cdot)$ obtaining a bit b', and finally outputs b' as its guess.

It is clear that A leaks at most $\ell - 1 + 1 = \ell$ bits. Moreover, A perfectly simulates the view of A^+, and thus, it retains the same advantage. This finishes the proof. □

3. Continuous Non-Malleability

In this section, we formalize the notion of continuous non-malleability against split-state tampering in the common reference string (CRS) model. To that end, we start by describing the syntax of split-state codes in the CRS model.

Formally, a split-state code in the CRS model is a tuple of algorithms $\Gamma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ specified as follows: (i) The randomized algorithm Init takes as input the security parameter and outputs a CRS $\omega \leftarrow\$ \mathsf{Init}(1^\lambda)$. (ii) The randomized encoding algorithm Enc takes as input some message $m \in \{0, 1\}^k$ and the CRS, and outputs a codeword

[6] A similar statement holds for computationally secure LRS.

consisting of two parts $c := (c_0, c_1) \in \{0,1\}^{2n}$ where $c_0, c_1 \in \{0,1\}^n$. (iii) The deterministic algorithm Dec takes as input a codeword $(c_0, c_1) \in \{0,1\}^{2n}$ and the CRS, and outputs either a message $m' \in \{0,1\}^k$ or a special symbol $\perp$.

As usual, we say that Γ is correct if for all $\lambda \in \mathbb{N}$, for all $\omega \in \mathsf{Init}(1^\lambda)$, and for all $m \in \{0,1\}^k$, it holds that $\mathsf{Dec}(\omega, \mathsf{Enc}(\omega, m)) = m$ (with probability 1 over the randomness of Enc).

3.1. *The Definition*

Intuitively, non-malleability captures a setting where an adversary A tampers a *single time* with a target encoding $c := (c_0, c_1)$ of some message m. The tampering attack is arbitrary, as long as it modifies the two parts c_0 and c_1 of the target codeword c independently, i.e., the attacker can choose any tampering function $f := (f_0, f_1)$ where $f_0, f_1 : \{0,1\}^n \to \{0,1\}^n$. This results in a modified codeword $\tilde{c} := (\tilde{c}_0, \tilde{c}_1) := (f_0(c_0), f_1(c_1))$, and different flavors of non-malleability are possible depending on what information the attacker obtains about the decoding of $\tilde{c}$:

- **Weak non-malleability** In this case the attacker obtains the decoded message $\tilde{m} \in \{0,1\}^k \cup \{\perp\}$ corresponding to $\tilde{c}$, unless $\tilde{m} = m$ (in which case the adversary gets a special "same" symbol $\diamond$);
- **Strong non-malleability** In this case the attacker obtains the decoded message $\tilde{m} \in \{0,1\}^k \cup \{\perp\}$ corresponding to $\tilde{c}$, unless $\tilde{c} = c$ (in which case the adversary gets a special "same" symbol $\diamond$);
- **Super non-malleability** In this case the attacker obtains $\tilde{c}$, unless $\tilde{c}$ is invalid (in which case the adversary gets $\perp$), or $\tilde{c} = c$ (in which case the adversary gets a special "same" symbol $\diamond$).

Super non-malleability is the strongest flavor, as it implies that not even the mauled codeword reveals information about the underlying message (as long as it is valid and different from the original codeword). Similarly, one can see that strong non-malleability is strictly stronger than weak non-malleability. For the rest of this paper, whenever we write "non-malleability" (without specifying the flavor weak/strong/super) we implicitly mean "super non-malleability".

Continuous non-malleability generalizes the above setting to the case where the attacker tampers adaptively with the same target codeword c, and for each attempt obtains some information about the decoding of the modified codeword $\tilde{c}$ (as above). The only restriction is that whenever a tampering attempt yields an invalid codeword, the system "self-destructs". Toward defining continuous non-malleability, consider the following oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)$, which is parameterized by a codeword (c_0, c_1) and takes as input functions $f_0, f_1 : \{0,1\}^n \to \{0,1\}^n$.

```
Oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0, f_1))$ :
$(\tilde{c}_0, \tilde{c}_1) = (f_0(c_0), f_1(c_1))$
If $(\tilde{c}_0, \tilde{c}_1) = (c_0, c_1)$ return $\diamond$
If $\mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1)) = \perp$, return $\perp$ and "self-destruct"
Else return $(\tilde{c}_0, \tilde{c}_1)$.
```

By “self-destruct” we mean that once $\mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1))$ outputs $\bot$, the oracle will answer $\bot$ to any further query. We are now ready to define (leakage-resilient) continuous non-malleability.

Definition 6. (Continuous non-malleability) Let $\Gamma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a split-state code in the CRS model. We say that Γ is ℓ-leakage-resilient continuously super-non-malleable (ℓ-CNMLR for short), if it holds that

$$\left\{\mathbf{Tamper}_{\Gamma,\mathsf{A}}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{\mathbf{Tamper}_{\Gamma,\mathsf{A}}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}},$$

where for $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1)$ and $b \in \{0, 1\}$ we set:

$$\mathbf{Tamper}_{\Gamma,\mathsf{A}}(\lambda, b) = \left\{ b' : \begin{array}{c} \omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda) \\ (m_0, m_1, \alpha_1) \leftarrow\!\!\$\ \mathsf{A}_0(1^\lambda, \omega) \\ (c_0, c_1) \leftarrow\!\!\$\ \mathsf{Enc}(\omega, m_b) \\ b' \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}^\ell_{\mathsf{leak}}(c_0,\cdot),\mathcal{O}^\ell_{\mathsf{leak}}(c_1,\cdot),\mathcal{O}_{\mathsf{maul}}((c_0,c_1),\cdot)}(1^\lambda, \alpha_1) \end{array} \right\}.$$

3.2. *Codewords Uniqueness*

As we argue below, constructions that satisfy our new Definition 6 have to meet the following requirement: For any (possibly adversarially chosen) side of an encoding c_0 it is computationally hard to find two corresponding sides c_1 and c_1' such that both (c_0, c_1) and (c_0, c_1') form a valid encoding; moreover, a similar property holds if we fix the right side c_1 of a codeword.

Definition 7. (Codewords uniqueness) Let $\Gamma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a split-state code in the CRS model. We say that Γ satisfies *codewords uniqueness* if for all PPT adversaries A we have:

$$\mathbb{P}\left[\begin{array}{c}\mathsf{Dec}(\omega, (c_0, c_1)) \neq \bot \wedge \mathsf{Dec}(\omega, (c_0, c_1')) \neq \bot \\ \wedge\, c_1 \neq c_1'\end{array} : \begin{array}{c}\omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda) \\ (c_0, c_1, c_1') \leftarrow\!\!\$\ \mathsf{A}(1^\lambda, \omega)\end{array}\right] \in \mathtt{negl}(\lambda)$$

$$\mathbb{P}\left[\begin{array}{c}\mathsf{Dec}(\omega, (c_0, c_1)) \neq \bot \wedge \mathsf{Dec}(\omega, (c_0', c_1)) \neq \bot \\ \wedge\, c_0 \neq c_0'\end{array} : \begin{array}{c}\omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda) \\ (c_0, c_0', c_1) \leftarrow\!\!\$\ \mathsf{A}(1^\lambda, \omega)\end{array}\right] \in \mathtt{negl}(\lambda).$$

The following attack shows that codewords uniqueness is necessary to achieve Definition 6.

Lemma 1. *Let Γ be a 0-leakage-resilient continuously super-non-malleable split-state code. Then, Γ must satisfy codewords uniqueness.*

Proof. For the sake of contradiction, assume that there exists a PPT attacker A' that outputs a triple (c_0, c_1, c_1') violating uniqueness of Γ, i.e., such that (c_0, c_1) and (c_0, c_1') are both valid, with $c_1 \neq c_1'$. We show how to construct a PPT attacker $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1)$ breaking continuous non-malleability of Γ. Attacker A_0, given the CRS, outputs any two messages $m_0, m_1 \in \{0, 1\}^k$ which differ, say, in the first bit; denote by $(\hat{c}_0, \hat{c}_1)$ the target codeword corresponding to m_b in the experiment $\mathbf{Tamper}_{\Gamma,\mathsf{A}}(\lambda, b)$. Attacker

A_1 runs A', and then queries the tampering oracle $\mathcal{O}_{\mathsf{maul}}((\hat{c}_0, \hat{c}_1), \cdot)$ with a sequence of $n \in \mathtt{poly}(\lambda)$ tampering queries, where the i-th query $(f_0^{(i)}, f_1^{(i)})$ is specified as follows:

- Function $f_0^{(i)}$ overwrites $\hat{c}_0$ with c_0.
- Function $f_1^{(i)}$ reads the i-th bit $\hat{c}_1[i]$ of $\hat{c}_1$; in case $c\hat{[i]} = 0$ it overwrites $\hat{c}_1$ with c_1, and else it overwrites $\hat{c}_1$ with c_1'.

Note that, as long as A' breaks codewords uniqueness, $\tilde{c}^{(i)} = (f_0^{(i)}(c_0), f_1^{(i)}(c_1))$ is a valid codeword for all $i \in [n]$. This allows A_1 to fully recover $\hat{c}_1$ after n tampering queries, unless $(\hat{c}_0, \hat{c}_1) \in \{(c_0, c_1), (c_0, c_1')\}$.[7]

Finally, A_1 asks an additional tampering query $(f_0^{(n+1)}, f_1^{(n+1)})$ to $\mathcal{O}_{\mathsf{maul}}((\hat{c}_0, \hat{c}_1), \cdot)$:

- $f_0^{(n+1)}(\hat{c}_0)$ hard-wires $\hat{c}_1$ and computes $m = \mathsf{Dec}(\omega, (\hat{c}_0, \hat{c}_1))$; if the first bit of m is 0, then it behaves like the identity function, and otherwise it overwrites $\hat{c}_0$ with 0^n.
- $f_1^{(n+1)}(\hat{c}_1)$ is the identity function.

The above clearly allows to learn the first bit of the message in the target encoding, and hence contradicts the fact that Γ is continuously non-malleable. $\square$

Attacking existing schemes The attack of Lemma 1 can be used to show that the code of [84] does not satisfy continuous non-malleability as per our definition. Recall that in [84] a message m is encoded as $c_0 = (pk, \gamma := \mathsf{Encrypt}(pk, m), \pi)$ and $c_1 = sk$. Here, (pk, sk) is a valid public/secret key pair and π is an argument of knowledge of (m, sk) such that γ decrypts to m under *sk and* (pk, sk) is well formed. Clearly, for any fixed $c_1 = sk$ it is easy to find two corresponding parts $c_0 \neq c_0'$ such that both (c_0, c_1) and (c_0', c_1) are valid.[8]

Let us mention two important extensions of the above attack, leading to even stronger security breaches.

1. In case the pair of *valid* codewords (c_0, c_1), (c_0, c_1') violating the uniqueness property are such that $\mathsf{Dec}(\omega, (c_0, c_1)) \neq \mathsf{Dec}(\omega, (c_0, c_1'))$, one can show that Lemma 1 even rules out continuous *weak* non-malleability.[9] The latter flavor of uniqueness is sometimes referred to as *message uniqueness* [60,85].
2. In case it is possible to find *both* (c_0, c_1, c_1') and (c_0, c_0', c_1) violating uniqueness, a simple variant of the attack from Lemma 1 allows to recover both halves of the target encoding, which is a total breach of security. However, it is not clear how to do that for the scheme of [84], as once we fix $c_0 = (pk, \gamma, \pi)$, it shall be hard to find two valid secret keys $c_1 = sk \neq sk' = c_1'$ corresponding to pk.

A simple adaptation of the above attack shows that continuous non-malleability in the split-state model is impossible in the information-theoretic setting.

[7]If not, A_1 obtains $\diamond$ as answer to one of its tampering queries, and thus it can safely conclude that $\hat{c}_0 = c_0$ and $\hat{c}_1 \in \{c_1, c_1'\}$. Once $\hat{c}_0$ is known it is trivial to break non-malleability using an additional tampering query.

[8]Namely, just have c_0 (resp. c_1) contain pk, the encryption γ (resp. γ') of any message m (resp. $m' \neq m$) under pk, and the corresponding argument π (resp. π').

[9]Note that this applies in particular to the encoding of [84].

Theorem 2. *There is no split-state code Γ in the CRS model that is* 0*-leakage-resilient continuously super-non-malleable in the presence of a computationally unbounded adversary.*

Proof. By contradiction, assume that there exists an information-theoretically secure continuously non-malleable split-state code Γ with $2n$-bit codewords. By Lemma 1, the code Γ must satisfy codewords uniqueness. In the information-theoretic setting, this means that for all CRSs $\omega \in \mathsf{Init}(1^\lambda)$, and for any codeword $(c_0, c_1) \in \{0,1\}^{2n}$ such that $\mathsf{Dec}(\omega, (c_0, c_1)) \neq \perp$, the following two properties hold: (i) for all $c_1' \in \{0,1\}^n$ such that $c_1' \neq c_1$, we have $\mathsf{Dec}(\omega, (c_0, c_1')) = \perp$; (ii) for all $c_0' \in \{0,1\}^n$ such that $c_0' \neq c_0$, we have $\mathsf{Dec}(\omega, (c_0', c_1)) = \perp$.

Let now $(\hat{c}_0, \hat{c}_1)$ be a target encoding of some secret $m \in \{0,1\}^k$. An unbounded attacker A can define the following tampering query (f_0, f_1):

- f_0, given $\hat{c}_0$ as input, tries all possible $\hat{c}_1 \in \{0,1\}^n$ until one is found such that $\mathsf{Dec}(\omega, (\hat{c}_0, \hat{c}_1)) \neq \perp$. Hence, it runs $m = \mathsf{Dec}(\omega, (\hat{c}_0, \hat{c}_1))$, and it leaves $\hat{c}_0$ in case the first bit of the message is zero whereas it overwrites $\hat{c}_0$ with 0^n otherwise.
- f_1 is the identity function.

Note that by properties (i) and (ii) above, we know that for all $\hat{c}_1' \neq \hat{c}_1$, the decoding algorithm $\mathsf{Dec}(\omega, (\hat{c}_0, \hat{c}_1'))$ outputs $\perp$. Thus, the above tampering query allows to learn the first bit of m with overwhelming probability, which is a clear breach of non-malleability. □

Note that although the attack of Theorem 2 uses a single (inefficient) tampering query, it crucially relies on the assumption that the code Γ be continuously non-malleable, in that it uses the fact that Γ satisfies codewords uniqueness. This is consistent with the fact that all split-state codes that achieve one-time non-malleability in the information-theoretic setting do not satisfy uniqueness.

4. The Code

We describe our split-state code in Sect. 4.1. A detailed outline of the security proof can be found in Sect. 4.2, whereas the formal proof is given in Sect. 4.3–4.5 Finally, in Sect. 4.6, we explain how to instantiate our scheme both from generic and concrete assumptions.

4.1. *Description*

Our construction combines a hash function $(\mathsf{Gen}, \mathsf{Hash})$ (cf. Sect. 2.2), a non-interactive argument system $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$ for proving knowledge of a pre-image of a hash value (cf. Sect. 2.3), and a leakage-resilient storage $\Sigma = (\mathsf{LREnc}, \mathsf{LRDec})$ (cf. Sect. 2.4), as depicted in Fig. 1.

The main idea behind the scheme is as follows: The CRS includes the CRS ω for the argument system and the hash key hk for the hash function. Given a message $m \in \{0,1\}^k$, the encoding procedure first encodes m using LREnc, obtaining shares (s_0, s_1). Hence, Let $\Sigma = (\mathsf{LREnc}, \mathsf{LRDec})$ be a leakage-resilient storage with message space $\{0,1\}^k$ and share space $\{0,1\}^{2n}$. Let $(\mathsf{Gen}, \mathsf{Hash})$ be a hash function with domain $\{0,1\}^n$, and $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$ be a non-interactive argument system for the language

$$\mathcal{L}^{hk}_{\mathsf{hash}} = \{h \in \{0,1\}^\lambda : \exists s \in \{0,1\}^n \text{ s.t. } \mathsf{Hash}(hk, s) = h\},$$

that supports labels in $\{0,1\}^\lambda$. Define the following split-state code $\Gamma^* = (\mathsf{Init}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ in the CRS model, with message space $\{0,1\}^k$.

Initialization algorithm Init^***:** Sample $\omega \leftarrow\!\!\$\ \mathsf{CRSGen}(1^\lambda)$ and $hk \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, and return $\omega^* = (\omega, hk)$.

Encoding algorithm Enc^***:** Upon input $\omega^* = (\omega, hk)$ and a value $m \in \{0,1\}^k$, compute $(s_0, s_1) \leftarrow\!\!\$\ \mathsf{LREnc}(m)$. Hence, let $\pi_0 \leftarrow\!\!\$\ \mathsf{Prove}(\omega, h_1, (s_0, h_0))$ and $\pi_1 \leftarrow\!\!\$\ \mathsf{Prove}(\omega, h_0, (s_1, h_1))$, where $h_0 = \mathsf{Hash}(hk, s_0)$ and $h_1 = \mathsf{Hash}(hk, s_1)$. Output $c^* = (c_0, c_1) = ((s_0, h_1, \pi_0, \pi_1), (s_1, h_0, \pi_0, \pi_1))$.

Decoding algorithm Dec^***:** Upon input $\omega^* = (\omega, hk)$ and some codeword $c^* = ((s_0, h_1, \pi_{0,0}, \pi_{1,0}), (s_1, h_0, \pi_{0,1}, \pi_{1,1}))$, proceed as follows.

(a) `Cross check:` If either (i) $\mathsf{Hash}(hk, s_0) \neq h_0$ or $\mathsf{Hash}(hk, s_1) \neq h_1$, or (ii) $(\pi_{0,0}, \pi_{1,0}) \neq (\pi_{0,1}, \pi_{1,1})$, return $\perp$. Else, let $\pi_0 := \pi_{0,0} = \pi_{1,0}$ and $\pi_1 := \pi_{0,1} = \pi_{1,1}$.

(b) `Local check:` If $\mathsf{Ver}(\omega, h_0, (h_1, \pi_1)) = 0$ or $\mathsf{Ver}(\omega, h_1, (h_0, \pi_0)) = 0$, return $\perp$.

(c) `LRS decoding:` Otherwise, output $\mathsf{LRDec}(s_0, s_1)$.

Fig. 1. Leakage-resilient continuously non-malleable split-state code, in the CRS model.

it hashes both s_0 and s_1, obtaining values h_0 and h_1, and generates a non-interactive argument π_0 (resp. π_1) for the statement $h_0 \in \mathcal{L}^{hk}_{\mathsf{hash}}$ (resp. $h_1 \in \mathcal{L}^{hk}_{\mathsf{hash}}$) using h_1 (resp. h_0) as label. The left part c_0 (resp. right part c_1) of the codeword c^* consists of s_0 (resp. s_1), along with the value h_1 (resp. h_0), and with the arguments π_0, π_1.

The decoding algorithm proceeds in the natural way. Namely, given a codeword $c^* = (c_0, c_1)$ it first checks that the arguments contained in the left and right part are equal, and moreover that the hash values are consistent with the shares; further, it checks that the non-interactive arguments verify correctly w.r.t. the corresponding statement and label. If any of the above checks fails, the algorithm returns $\perp$, and otherwise it outputs the same as $\mathsf{LRDec}(s_0, s_1)$.

Correctness of the code Γ^* follows directly by the correctness properties of the LRS and of the non-interactive argument system. As for security, we establish the following result.

Theorem 3. *Let $\ell \in \mathbb{N}$. Assume that:*

(i) *$(\mathsf{LREnc}, \mathsf{LRDec})$ is an augmented ℓ-leakage-resilient storage;*
(ii) *$(\mathsf{Gen}, \mathsf{Hash})$ is a collision-resistant hash function;*
(iii) *$(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$ is a true-simulation extractable non-interactive zero-knowledge argument system for the language $\mathcal{L}^{hk}_{\mathsf{hash}} = \{h \in \{0,1\}^\lambda : \exists s \in \{0,1\}^n$ s.t. $\mathsf{Hash}(hk, s) = h\}$.*

Then, the split-state code Γ^* described in Fig. 1 is an ℓ^*-leakage-resilient continuously super-non-malleable code in the CRS model, as long as $\ell^* \leq \ell/2 - O(\lambda \log(\lambda))$.

```
Oracle $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0, f_1))$:
For all $\beta \in \{0, 1\}$, do:
    $\Theta_\beta \leftarrow\!\!\$\ \mathsf{SimTamp}(\tilde{c}_\beta, f_\beta)$
If $\Theta_0 \neq \Theta_1$
    Return $\perp$ and self-destruct
Else If $\Theta_0 = \Theta_1 = \perp$
    Return $\perp$ and self-destruct
Else
    Return $\Theta_0$
```

```
Algorithm $\mathsf{SimTamp}(c_\beta, f_\beta)$:
$\tilde{c}_\beta = f_\beta(c_\beta)$
$\tilde{c}_\beta := (\tilde{s}_\beta, \tilde{h}_{1-\beta}, \tilde{\pi}_{0,\beta}, \tilde{\pi}_{1,\beta})$;
$\tilde{h}_{\beta,\beta} = \mathsf{Hash}(hk, \tilde{s}_\beta)$
If $\tilde{c}_\beta = c_\beta$ // Type-A
    $\Theta_\beta = \diamond$
Else If $\mathsf{Ver}(\omega, \tilde{h}_{1-\beta}, (\tilde{h}_{\beta,\beta}, \tilde{\pi}_{\beta,\beta})) = 0$ // Type-B
    $\Theta_\beta = \perp$
Else If $\mathsf{Ver}(\omega, \tilde{h}_{\beta,\beta}, (\tilde{h}_{1-\beta}, \tilde{\pi}_{1-\beta,\beta})) = 0$ // Type-B
    $\Theta_\beta = \perp$
Else If $(\tilde{h}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{1-\beta,\beta}) = (h_{1-\beta}, h_\beta, \pi_{1-\beta})$ // Type-C
    $\Theta_\beta = \perp$
Else // Type-D
    $\tilde{s}_{1-\beta} := \mathsf{K}(\xi, \tilde{h}_{\beta,\beta}, (\tilde{h}_{1-\beta}, \tilde{\pi}_{1-\beta,\beta}))$
    $\tilde{c}_{1-\beta,\beta} := (\tilde{s}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{0,\beta}, \tilde{\pi}_{1,\beta})$
    $\tilde{c}_{\beta,\beta} = \tilde{c}_\beta$
    $\Theta_\beta = (\tilde{c}_{0,\beta}, \tilde{c}_{1,\beta})$
```

Fig. 2. Modified tampering oracle $\mathcal{O}'_{\mathsf{maul}}$ used in the experiment $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$. For simplicity, we assume that the extractor K is deterministic (which is the case for known instantiations, e.g., [51]); a generalization is immediate.

4.2. *Proof Outline*

Our goal is to show that no PPT attacker A^* can distinguish the experiments $\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, 0)$ and $\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, 1)$ (cf. Definition 6). Recall that A^*, after seeing the CRS, can select two messages $m_0, m_1 \in \{0, 1\}^k$, and then adaptively tamper with, and leak from, a target encoding $c^* = (c_0, c_1)$ of either m_0 or m_1, where both the tampering and the leakage act independently on the two parts c_0 and c_1.

In order to prove the theorem, we introduce two hybrid experiments, as outlined below.

$\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$: In the first hybrid, we modify the distribution of the target codeword $c^* = (c_0, c_1)$. In particular, we first use the simulator S_0 of the non-interactive argument system to program the CRS ω, yielding simulation trapdoor ζ and extraction trapdoor ξ, and then we compute the argument π_0 (resp. π_1) by running the simulator S_1 upon input ζ, statement h_0 (resp. h_1) and label h_1 (resp. h_0).

$\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$: In the second hybrid, we modify the way tampering queries are answered. In particular, let (f_0, f_1) be a generic tampering query, and $\tilde{c}^* = (\tilde{c}_0, \tilde{c}_1) = (f_0(c_0), f_1(c_1))$ be the corresponding mauled codeword. Note that $\tilde{c}_0$ can be parsed as $\tilde{c}_0 = (\tilde{s}_0, \tilde{h}_1, \tilde{\pi}_{0,0}, \tilde{\pi}_{1,0})$, and similarly $\tilde{c}_1 = (\tilde{s}_1, \tilde{h}_0, \tilde{\pi}_{0,1}, \tilde{\pi}_{1,1})$. The modified tampering oracle then proceeds as follows, for each $\beta \in \{0, 1\}$.

(a) In case $\tilde{c}_\beta = c_\beta$, define $\Theta_\beta := \diamond$ (cf. Type-A queries in Fig. 2).

(b) In case $\tilde{c}_\beta \neq c_\beta$, but either of the arguments in $\tilde{c}_\beta$ does not verify correctly, define $\Theta_\beta := \perp$ (cf. Type-B queries in Fig. 2).

(c) In case $\tilde{c}_\beta \neq c_\beta$ and both the arguments in $\tilde{c}_\beta$ verify correctly, let $\tilde{h}_{\beta,\beta} := \mathsf{Hash}(hk, \tilde{s}_\beta)$. Check if $(\tilde{h}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{1-\beta,\beta}) = (h_{1-\beta}, h_\beta, \pi_{1-\beta})$; if not (in which case we cannot extract from $\tilde{\pi}_{1-\beta}$), then define $\Theta_\beta := \perp$ (cf. Type-C queries in Fig. 2).

(d) Otherwise, run the knowledge extractor K of the underlying non-interactive argument system upon input extraction trapdoor ξ, statement $\tilde{h}_{1-\beta}$, argument $\tilde{\pi}_{1-\beta,\beta}$, and label $\tilde{h}_{\beta,\beta} := \mathsf{Hash}(hk, \tilde{s}_\beta)$, yielding a share $\tilde{s}_{1-\beta}$. Thus, let $\tilde{c}_{1-\beta} = (\tilde{s}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{0,\beta}, \tilde{\pi}_{1,\beta})$ and define $\Theta_\beta := (\tilde{c}_0, \tilde{c}_1)$ (cf. Type-D queries in Fig. 2).

Finally, if $\Theta_0 = \Theta_1$ the oracle returns this value as answer to the tampering query (f_0, f_1); else, it returns $\perp$ and self-destructs. (Of course, in case $\Theta_0 = \Theta_1 = \perp$, the oracle also returns $\perp$ and self-destructs.)

As a first step, we argue that $\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$ and $\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$ are computationally close. This follows readily from adaptive multi-theorem zero knowledge of the non-interactive argument system, as the only difference between the two experiments is the fact that in the latter the arguments π_0, π_1 are simulated. As a second step, we prove that $\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$ and $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}}(\lambda, b)$ are also computationally indistinguishable. More in details, we show how to bound the probability that the output of the tampering oracle in the two experiments differs in the above described cases (a), (b), (c) and (d):

(a) For Type-A queries, note that when $\tilde{c}_\beta = c_\beta$, we must have $\tilde{c}_{1-\beta} = c_{1-\beta}$ with overwhelming probability, as otherwise, say, $(c_0, c_1, \tilde{c}_1)$ would violate codewords uniqueness, which for our code readily follows from collision resistance of the hash function.

(b) For Type-B queries, the decoding process in the previous hybrid would also return $\perp$, so these queries always yield the same output in the two experiments.

(c) For Type-C queries, we use the facts that (i) the underlying non-interactive argument system is true-simulation extractable, and (ii) the hash function is collision resistant, to show that $\tilde{c}_\beta$ must be of the form $\tilde{c}_\beta = (s_\beta, h_{1-\beta}, \tilde{\pi}_{0,\beta}, \pi_{1,\beta})$ with $\tilde{\pi}_{0,\beta} \neq \pi_{1,\beta}$. As we show, the latter contradicts security of the underlying LRS.

(d) For Type-D queries, note that whenever we run the extractor either the statement $\tilde{h}_{1-\beta}$, or the argument $\tilde{\pi}_{1-\beta,\beta}$, or the label $\tilde{h}_{\beta,\beta}$ are fresh, which ensures the witness must be valid with overwhelming probability by (true-simulation) extractability of the non-interactive argument system.

Next, we show that no PPT attacker A^* can distinguish between experiments $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 0)$ and $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 1)$ with better than negligible probability. To this end, we build a reduction A to the security of the underlying LRS. In order to keep the exposition simple, let us first assume that A^* is not allowed to ask leakage queries. Roughly, the reduction works as follows:

- **Simulate the CRS:** At the beginning, A samples a programmed CRS ω and the hash key hk exactly as defined in $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$, and runs A^* upon $\omega^* := (\omega, hk)$ and fresh randomness r. Upon receiving (m_0, m_1) from A^*, then A forward the same pair of messages to the challenger.

- **Learn the self-destruct index:** Note that in the last hybrid, the tampering oracle answers A^*'s tampering queries by computing both Θ_0 (looking only at c_0) and Θ_1 (looking only at c_1), and then Θ_0 is returned as long as $\Theta_0 = \Theta_1$ (and otherwise a self-destruct is triggered). Since A can leak independently from s_0 and s_1, it can compute all the values Θ_β by running A^* with hard-wired[10] randomness r inside[11] each of its leakage oracles, and then use a pairwise independent hash function to determine using a binary search the first index i^* corresponding to the tampering query where $\Theta_0 \neq \Theta_1$. By pairwise independence, this yields the index of the query i^* in which A^* provokes a self-destruct with overwhelming probability, and by leaking at most $O(\lambda \log \lambda)$ bits from each block.
- **Play the game:** Once the self-destruct index i^* is known, A obtains[12] s_σ and can thus restart A^* outside the leakage oracle, using the same randomness r, and answer to the first $i^* - 1$ tampering queries using $c_\sigma = (s_\sigma, h_{1-\sigma}, \pi_0, \pi_1)$, after which the answer to all remaining tampering queries is set to be $\perp$. This yields a perfect simulation of how tampering queries are handled in the last hybrid, so that A keeps the advantage of A^*.

Finally, let us explain how to remove the simplifying assumption that A^* cannot leak from the target codeword. The difficulty when considering leakage is that we cannot run anymore the entire experiment with A^* inside the leakage oracle, as the answer to A^*'s leakage queries depends on the other half of the target codeword. However, note that in this case we can stop the execution and inform the reduction to leak from the other side whatever information is needed to continue the execution of each copy of A^* inside the leakage oracle.

This allows to obtain the answers to all leakage queries of A^* up to a self-destruct occurs. In order to obtain the answers to the remaining queries, we must re-run A inside the leakage oracle and adjust the simulation consistently with the self-destruct index being i^*. In the worst case, this requires $2\ell^*$ bits of leakage, yielding the final bound of $2\ell^* + O(\lambda \log \lambda)$. At the end, the reduction knows the answer to all leakage queries of A^* with hard-wired randomness r, which allows to play the game with the challenger as explained above.

4.3. *Hybrids*

Let us start by recalling the definition of experiment $\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$ for our code from Fig. 1.

```
$\underline{\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)}$:
$\omega \leftarrow\!\!\$\ \mathsf{CRSGen}(1^\lambda),\ hk \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda),\ \omega^* := (\omega, hk)$
$(m_0, m_1, \alpha_1) \leftarrow\!\!\$\ \mathsf{A}_0^*(1^\lambda, \omega^*)$
$(s_0, s_1) \leftarrow\!\!\$\ \mathsf{LREnc}(m_b)$
$\forall \beta \in \{0,1\}:\ h_\beta = \mathsf{Hash}(hk, s_\beta),\ \pi_\beta \leftarrow\!\!\$\ \mathsf{Prove}(\omega, h_{1-\beta}, (h_\beta, s_\beta))$
```

[10]To be more precise, this also requires to leak the initial hash values, to simulate the non-interactive arguments as done in the last hybrid, and to hard-wire those values in each leakage query.

[11]This makes the reduction non-black-box, as it needs to run A^* inside the leakage oracles.

[12]Recall that the underlying LRS satisfies augmented leakage resilience (cf. Definition 5).

$$c^* := (c_0, c_1) := ((s_0, h_1, \pi_0, \pi_1), (s_1, h_0, \pi_0, \pi_1))$$
$$b' \leftarrow_\$ \mathsf{A}_1^{*\mathcal{O}_{\mathsf{leak}}^{\ell^*}(c_0,\cdot),\mathcal{O}_{\mathsf{leak}}^{\ell^*}(c_1,\cdot),\mathcal{O}_{\mathsf{maul}}((c_0,c_1),\cdot)}(1^\lambda, \alpha_1)$$

Consider the following hybrid experiments.

- $\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$: Let $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$ be the zero-knowledge simulator guaranteed by the non-interactive argument system. This hybrid is identical to the original experiment, except that the instructions $\omega \leftarrow_\$ \mathsf{CRSGen}(1^\lambda)$ and $\pi_\beta \leftarrow_\$ \mathsf{Prove}(\omega, h_{1-\beta}, (h_\beta, s_\beta))$ are replaced by $(\omega, \zeta, \xi) \leftarrow_\$ \mathsf{S}_0(1^\lambda)$ and $\pi_\beta \leftarrow_\$ \mathsf{S}_1(\zeta, h_{1-\beta}, h_\beta)$ respectively.
- $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$: Identical to the previous experiment, except that the oracle $\mathcal{O}_{\mathsf{maul}}$ is replaced by $\mathcal{O}'_{\mathsf{maul}}$ described in Fig. 2.

We first prove, in Sect. 4.4, that

$$\left\{\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}}.$$

Then, in Sect. 4.5, we show that the above hybrids are computationally close, i.e., for all $b \in \{0, 1\}$:

$$\left\{\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\right\}_{\lambda\in\mathbb{N}},$$

thus proving continuous non-malleability:

$$\left\{\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}}.$$

4.4. *The Main Reduction*

Lemma 2. $\{\boldsymbol{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 0)\}_{\lambda\in\mathbb{N}} \approx_c \{\boldsymbol{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. The proof is down to the augmented leakage resilience (cf. Definition 5) of the underlying LRS ($\mathsf{LREnc}, \mathsf{LRDec}$). The reduction relies on a family of weakly universal[13] hash functions Ψ. By contradiction, assume that there exists a PPT adversary $\mathsf{A}^* = (\mathsf{A}_0^*, \mathsf{A}_1^*)$ that can tell apart $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 0)$ and $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, 1)$ with non-negligible probability. Consider the following PPT attacker A against leakage resilience of ($\mathsf{LREnc}, \mathsf{LRDec}$), which relies on the leakage functions defined in Fig. 3 on the following page.

Attacker A for ($\mathsf{LREnc}, \mathsf{LRDec}$) :

1. **(Setup phase.)** Simulate the CRS as follows:
   (a) Run $hk \leftarrow_\$ \mathsf{Gen}(1^\lambda)$ and $(\omega, \zeta, \xi) \leftarrow_\$ \mathsf{S}_0(1^\lambda)$; set $\omega^* = (\omega, hk)$.

[13] A family of hash functions $\Psi = \{\psi_t : \{0,1\}^{2n} \to \{0,1\}^\lambda\}_{t\in\{0,1\}^\lambda}$ is *weakly universal* if for all *distinct* $x_1, x_2 \in \{0,1\}^{2n}$ the following holds: $\mathbb{P}\left[\psi_t(x_1) = \psi_t(x_2) : t \leftarrow_\$ \{0,1\}^\lambda\right] \le 2^{-\lambda}$. Such families exist unconditionally.

```
Function $g_\beta^{\mathsf{hash}}(hk, s_\beta)$:
Return $h_\beta = \mathsf{Hash}(hk, s_\beta)$
```

```
Function $g_\beta^{\mathsf{sd}}(\hat{\alpha}, \Lambda_0, \Lambda_1, \psi_t, q_{\mathsf{med}}, s_\beta)$:
Parse $\hat{\alpha} = (\omega, hk, \xi, h_0, h_1, \pi_0, \pi_1, \langle \mathsf{A}_1^* \rangle, \alpha_1, r_1)$
Parse $\Lambda_0 = (\Lambda_0^{(1)}, \Lambda_0^{(2)}, \ldots), \Lambda_1 = (\Lambda_1^{(1)}, \Lambda_1^{(2)}, \ldots)$
Let $c_\beta := (s_\beta, h_{1-\beta}, \pi_0, \pi_1)$
Run $\mathsf{A}_1^*(1^\lambda, \alpha_1; r_1)$:
  Upon input leakage query $(g_0^{(j)}, g_1^{(j)})$:
    Answer with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$
  Upon input tampering query $(f_0^{(i)}, f_1^{(i)})$:
    $\Theta_\beta^{(i)} := \mathsf{SimTamp}(c_\beta, f_\beta)$
  Upon input guess $b'$
    $y_\beta = \psi_t(\Theta_\beta^{(1)} || \cdots || \Theta_\beta^{(q_{\mathsf{med}})})$
    Output $y_\beta$
```

```
Function $g_\beta^{\mathsf{temp}}(\hat{\alpha}, \Lambda_0, \Lambda_1, s_\beta)$:
Parse $\hat{\alpha} = (\omega, hk, \xi, h_0, h_1, \pi_0, \pi_1, \langle \mathsf{A}_1^* \rangle, \alpha_1, r_1)$
Parse $\Lambda_0 = (\Lambda_0^{(1)}, \Lambda_0^{(2)}, \ldots), \Lambda_1 = (\Lambda_1^{(1)}, \Lambda_1^{(2)}, \ldots)$
Initialize $q_\beta \leftarrow 0$
Let $c_\beta := (s_\beta, h_{1-\beta}, \pi_0, \pi_1)$
Run $\mathsf{A}_1^*(1^\lambda, \alpha_1; r_1)$:
  Upon input leakage query $(g_0^{(j)}, g_1^{(j)})$:
    If $(\Lambda_0^{(j)}, \Lambda_1^{(j)}) \neq (\varepsilon, \varepsilon)$, answer with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$
    Else If $\Lambda_{1-\beta}^{(j)} = \varepsilon$, stop and output $(0, \Lambda_\beta^{(j)} := g_\beta^{(j)}(c_\beta))$
  Upon input tampering query $(f_0^{(i)}, f_1^{(i)})$:
    $q_\beta \leftarrow q_\beta + 1$
    $\Theta_\beta^{(i)} := \mathsf{SimTamp}(c_\beta, f_\beta)$
  Upon input guess $b'$
    Output $(1, q_\beta)$
```

```
Function $g_\beta^{\mathsf{leak}}(\hat{\alpha}, \Lambda_0, \Lambda_1, i^*, s_\beta)$:
Parse $\hat{\alpha} = (\omega, hk, \xi, h_0, h_1, \pi_0, \pi_1, \langle \mathsf{A}_1^* \rangle, \alpha_1, r_1)$
Parse $\Lambda_0 = (\Lambda_0^{(1)}, \Lambda_0^{(2)}, \ldots), \Lambda_1 = (\Lambda_1^{(1)}, \Lambda_1^{(2)}, \ldots)$
Let $c_\beta := (s_\beta, h_{1-\beta}, \pi_0, \pi_1)$
Run $\mathsf{A}_1^*(1^\lambda, \alpha_1; r_1)$:
  Upon input leakage query $(g_0^{(j)}, g_1^{(j)})$:
    If $(\Lambda_0^{(j)}, \Lambda_1^{(j)}) \neq (\varepsilon, \varepsilon)$, answer with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$
    Else If $\Lambda_{1-\beta}^{(j)} = \varepsilon$, stop and output $(0, \Lambda_\beta^{(j)} := g_\beta^{(j)}(c_\beta))$
  Upon input tampering query $(f_0^{(i)}, f_1^{(i)})$:
    $\Theta_\beta^{(i)} := \mathsf{SimTamp}(c_\beta, f_\beta)$
    If $i < i^*$, answer with $\Theta_\beta^{(i)}$
    Else, answer with $\perp$
  Upon input guess $b'$
    Output 1
```

Fig. 3. Leakage functions used in the proof of Lemma 2. For simplicity, we assume that the adversary for continuous non-malleability always asks leakage queries on both parts of the target codeword. See Fig. 2 on page 18 for the definition of algorithm $\mathsf{SimTamp}$.

(b) Sample $r := (r_0, r_1) \leftarrow\!\!\$ \{0,1\}^*$, and run $(m_0, m_1, \alpha_1) \leftarrow\!\!\$ \mathsf{A}_0(\omega^*; r_0)$.
(c) Forward (m_0, m_1) to the challenger, obtaining access to the leakage oracles $\mathcal{O}_{\mathsf{leak}}^{\ell}(s_0, \cdot)$ and $\mathcal{O}_{\mathsf{leak}}^{\ell}(s_1, \cdot)$.
(d) For each $\beta \in \{0,1\}$, query $g_\beta^{\mathsf{hash}}(hk, \cdot)$ to $\mathcal{O}_{\mathsf{leak}}^{\ell}(s_\beta, \cdot)$, obtaining $h_\beta = \mathsf{Hash}(hk, s_\beta)$.
(e) For each $\beta \in \{0,1\}$ generate the argument $\pi_\beta \leftarrow\!\!\$ \mathsf{S}_1(\zeta, h_{1-\beta}, h_\beta)$.
(f) Let $\hat{\alpha} := (\omega, hk, \xi, h_0, h_1, \pi_0, \pi_1, \langle \mathsf{A}_1^* \rangle, \alpha_1, r_1)$, and $\Lambda_0, \Lambda_1 := (\varepsilon, \ldots, \varepsilon)$ be initially empty arrays.

2. **(Obtain the temporary leakages.)** Run the following loop:

(a) Query alternatively $\mathcal{O}_{\mathsf{leak}}^{\ell}(s_0, \cdot)$ and $\mathcal{O}_{\mathsf{leak}}^{\ell}(s_1, \cdot)$ with $g_0^{\mathsf{temp}}(\hat{\alpha}, \Lambda_0, \Lambda_1, \cdot)$ and $g_1^{\mathsf{temp}}(\hat{\alpha}, \Lambda_0, \Lambda_1, \cdot)$.
(b) For any $j \geq 1$, after the j-th query, if at least one of the oracles returns $(0, \Lambda_\beta^{(j)})$ update the j-th entry of Λ_β as in $\Lambda_\beta[j] = \Lambda_\beta^{(j)}$.
(c) If both oracles return $(1, q_0)$ and $(1, q_1)$, respectively, break the loop obtaining the (temporary) leakages Λ_0, Λ_1 and the (temporary) number of tampering queries $q = \min\{q_0, q_1\}$.

3. **(Learn the self-destruct index.)** Set $(q_{\mathsf{min}}, q_{\mathsf{max}}) = (0, q)$. Run the following loop:
   (a) If $q_{\mathsf{min}} = q_{\mathsf{max}}$, break the loop obtaining the self-destruct index $i^* := q_{\mathsf{min}} = q_{\mathsf{max}}$; else, set $q_{\mathsf{med}} = \lfloor \frac{q_{\mathsf{min}}+q_{\mathsf{max}}}{2} \rfloor$.
   (b) Sample $t \leftarrow\!\!\$ \{0, 1\}^\lambda$.
   (c) For each $\beta \in \{0, 1\}$, query $g_\beta^{\mathsf{sd}}(\hat{\alpha}, \Lambda_0, \Lambda_1, \psi_t, q_{\mathsf{med}}, \cdot)$ to $\mathcal{O}_{\mathsf{leak}}^\ell(s_\beta, \cdot)$ obtaining a value $y_\beta \in \{0, 1\}^\lambda$.
   (d) If $y_0 \neq y_1$, update $(q_{\mathsf{min}}, q_{\mathsf{max}}) \leftarrow (q_{\mathsf{min}}, q_{\mathsf{med}})$; else, update $(q_{\mathsf{min}}, q_{\mathsf{max}}) \leftarrow (q_{\mathsf{med}} + 1, q_{\mathsf{max}})$.
4. **(Correct the leakages.)** Set $\Lambda_0, \Lambda_1 \leftarrow (\varepsilon, \ldots, \varepsilon)$, i.e., discard all the temporary leakages. Run the following loop:
   (a) Query alternatively $\mathcal{O}_{\mathsf{leak}}^\ell(s_0, \cdot)$ and $\mathcal{O}_{\mathsf{leak}}^\ell(s_1, \cdot)$ with $g_0^{\mathsf{leak}}(\hat{\alpha}, \Lambda_0, \Lambda_1, i^*, \cdot)$ and $g_1^{\mathsf{leak}}(\hat{\alpha}, \Lambda_0, \Lambda_1, i^*, \cdot)$.
   (b) For any $j \geq 1$, after the j-th query, if at least one of the oracles returns $(0, \Lambda_\beta^{(j)})$ update the j-th entry of Λ_β as in $\Lambda_\beta[j] = \Lambda_\beta^{(j)}$.
   (c) If both oracles return 1, break the loop obtaining the final leakages Λ_0, Λ_1.
5. **(Play the game.)** Run the following loop:
   (a) Forward $\sigma = 0$ to the challenger, obtaining s_0.
   (b) Set $c_0 = (s_0, h_1, \pi_0, \pi_1)$, then start $\mathsf{A}_1^*(1^\lambda, \alpha_1; r_1)$.
   (c) Upon input the j-th leakage query $(g_0^{(j)}, g_1^{(j)})$ from A_1^*, return $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
   (d) Upon input the i-th tampering query $(f_0^{(i)}, f_1^{(i)})$ from A_1^*:
      - If $i < i^*$, answer with $\Theta_0^{(i)} = \mathsf{SimTamp}(c_0, f_0^{(i)})$.
      - If $i \geq i^*$, return $\perp$.
   (e) Upon input a guess b' from A_1^*, forward b' to the challenger and terminate.

For the analysis, we must show that A does not leak too much information and that the reduction is correct (in the sense that the view of A^* is simulated correctly). Note that A makes leakage queries in steps 1d, 2a, 3c, and 4a. The leakage amount in step 1d is equal to λ bits per share (i.e., the size of a hash value). The leakage amount in step 2a is bounded by $\ell^* + O(\log \lambda)$ bits per share (i.e., the maximum leakage asked by A^* plus the indexes q_0, q_1); by a similar argument, the leakage in step 4a consists of at most ℓ^* bits per share. Finally, the leakage in step 3c is bounded by $O(\lambda \log(\lambda))$ (as the loop for the binary search is run at most $O(\log \lambda)$ times, and each time the reduction leaks λ bits per share). Putting it all together, the overall leakage is bounded by

$$\lambda + 2(\ell^* + O(\log \lambda)) + O(\lambda \log(\lambda)) = 2\ell^* + O(\lambda \log(\lambda)) \leq \ell,$$

where the inequality follows by the bound on ℓ^* in the theorem statement.

Next, we argue that A perfectly simulates the view of A^* except with negligible probability. Indeed:

- The distribution of the CRS $\omega^* = (\omega, hk)$ is perfect.
- For each $\beta \in \{0, 1\}$, the distribution of the value c_β assembled inside the leakage oracle $\mathcal{O}^{\ell}_{\mathsf{leak}}(s_\beta, \cdot)$ is identical to that of the target codeword $c^* = (c_0, c_1)$ in $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$, where b is the hidden bit in the security game for $(\mathsf{LREnc}, \mathsf{LRDec})$.
- The simulation of A^*'s leakage queries is perfect. Note that the simulated leakages Λ_0, Λ_1 might be inconsistent after the loop of step 2 terminates. This is because there may exist an index $i \in [q]$ such that $\Theta_0^{(i)} \neq \Theta_1^{(i)}$ (i.e., the answer to the i-th tampering query should be $\perp$, but a different value is passed to A_1^* inside the leakage oracle), which causes a wrong simulation of all leakage queries $j \geq i$ (if any). However, the reduction adjusts the leakages later in step 4, after the index i^* corresponding to the self-destruct query is known.
- Except with negligible probability, the index i^* coincides with the index corresponding to the self-destruct query, i.e. the minimum $i^* \in [q]$ such that $\Theta_0^{(i^*)} \neq \Theta_1^{(i^*)}$ in $\mathbf{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$. The latter follows readily from the weak universality of Ψ, as for each query $g_0^{\mathsf{sd}}(\hat{\alpha}, \Lambda_0, \Lambda_1, \psi_t, q_{\mathsf{med}}, \cdot)$ and $g_1^{\mathsf{sd}}(\hat{\alpha}, \Lambda_0, \Lambda_1, \psi_t, q_{\mathsf{med}}, \cdot)$ the probability that

$$y_0 = \psi_t(\Theta_0^{(1)}||\cdots||\Theta_0^{(q_{\mathsf{med}})}) = \psi_t(\Theta_1^{(1)}||\cdots||\Theta_1^{(q_{\mathsf{med}})}) = y_1,$$

  but $\Theta_0^{(1)}||\cdots||\Theta_0^{(q_{\mathsf{med}})} \neq \Theta_1^{(1)}||\cdots||\Theta_1^{(q_{\mathsf{med}})}$ is at most $2^{-\lambda}$ (over the choice of $t \leftarrow_\$ \{0, 1\}^\lambda$), and thus, by the union bound, the simulation is wrong with probability at most $q \cdot 2^{-\lambda}$.
- In step 5, the reduction obtains c_0, and thus can use the knowledge of the final corrected leakages Λ_0, Λ_1 and of the self-destruct index i^* to perfectly simulate all the queries of A_1^*.

Hence, we have shown that there exists a polynomial $p(\lambda) \in \mathtt{poly}(\lambda)$ and a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\left|\mathbb{P}\left[\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}}(\lambda, 0, 0) = 1\right] - \mathbb{P}\left[\mathbf{Leak}^{+}_{\Sigma,\mathsf{A}}(\lambda, 1, 0) = 1\right]\right| \geq 1/p(\lambda) - \nu(\lambda).$$

This concludes the proof. □

4.5. *Indistinguishability of the Hybrids*

We first establish some useful lemmas, and then analyze each of the two game hops individually.

4.5.1. *Useful Lemmata*

Lemma 3. *The code $\Gamma^* = (\mathsf{Init}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ satisfies codewords uniqueness. Moreover, the latter still holds if we modify $(\mathsf{Init}^*, \mathsf{Enc}^*)$ as defined in $\mathbf{Hyb}^1_{\Gamma^*,A^*}(\lambda, b)$.*

Proof. We show that Definition 7 is satisfied for $\beta = 0$. The proof for $\beta = 1$ is analogous, and therefore omitted.

Assume that there exists a PPT adversary A^* that, given as input $\omega^* = (\omega, hk)$, is able to produce (c_0, c_1, c_1') such that both (c_0, c_1) and (c_0, c_1') are valid, but $c_1 \neq c_1'$. Let $c_0 = (s_0, h_1, \pi_0, \pi_1)$, $c_1 = (s_1, h_0, \pi_0, \pi_1)$, and $c_1' = (s_1', h_0', \pi_0', \pi_1')$. Since s_0 is the same in both codewords, we must have $h_0 = h_0'$ as the hash function is deterministic. Furthermore, since both (c_0, c_1) and (c_0, c_1') are valid, the arguments on the right parts must be equal to the ones on the left part, i.e., $\pi_0' = \pi_0$ and $\pi_1' = \pi_1$. It follows that $c_1' = (s_1', h_0, \pi_0, \pi_1)$, with $s_1' \neq s_1$, and thus (s_1, s_1') are a collision for $\mathsf{Hash}(hk, \cdot)$. The latter contradicts collision resistance of $(\mathsf{Gen}, \mathsf{Hash})$, as it can be seen by the simple reduction that given a target hash key hk embeds it in the CRS $\omega^* = (\omega, hk)$, runs $\mathsf{A}^*(1^\lambda, \omega^*)$ obtaining (c_0, c_1, c_1'), and outputs (s_1, s_1').

The second part of the statement of the lemma follows by the non-interactive zero-knowledge property of the argument system. In fact, the latter implies that the advantage of any PPT adversary A^* in breaking codewords uniqueness must be close (up to a negligible distance) in the first hybrid and in the original experiment. □

Lemma 4. *Whenever A^* outputs a Type-D query in $\mathbf{Hyb}^1_{\Gamma^*, A^*}(\lambda, b)$ the following holds: For all $\beta \in \{0, 1\}$ the codeword $(\tilde{c}_{0,\beta}, \tilde{c}_{1,\beta})$ contained in Θ_β must be valid with overwhelming probability.*

Proof. Fix any $\beta \in \{0, 1\}$, and let (f_0, f_1) be a generic query of Type D. Denote by $\Theta_\beta = (\tilde{c}_{0,\beta}, \tilde{c}_{1,\beta})$ the answer to any tampering query (f_0, f_1) as it would be computed by $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0, f_1))$, i.e., $\tilde{c}_{\beta,\beta} = (\tilde{s}_\beta, \tilde{h}_{1-\beta}, \tilde{\pi}_{0,\beta}, \tilde{\pi}_{1,\beta})$ and $\tilde{c}_{1-\beta,\beta} = (\tilde{s}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{0,\beta}, \tilde{\pi}_{1,\beta})$ with $\tilde{h}_{\beta,\beta} = \mathsf{Hash}(hk, \tilde{s}_\beta)$. By construction, $(\tilde{c}_{0,\beta}, \tilde{c}_{1,\beta})$ is invalid if and only if $\mathsf{Hash}(hk, \tilde{s}_{1-\beta}) \neq \tilde{h}_{1-\beta}$.

Fix $b \in \{0, 1\}$. Let now A^* be a PPT adversary for $\mathbf{Hyb}^1_{\Gamma^*, \mathsf{A}^*}(\lambda, b)$ that with probability at least $1/\mathtt{poly}(\lambda)$ outputs a Type-D tampering query (f_0, f_1) such that $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0, f_1))$ would yield an invalid codeword Θ_β as described above. We construct a PPT attacker A against true-simulation extractability (cf. Definition 3) of the non-interactive argument system. A description of A follows:

<u>Attacker A for $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$:</u>

- Upon receiving the target CRS ω, generate $hk \leftarrow\$ \mathsf{Gen}(1^\lambda)$ and let $\omega^* := (\omega, hk)$.
- Run $(m_0, m_1, \alpha_1) \leftarrow\$ \mathsf{A}_0^*(1^\lambda, \omega^*)$, and let $(s_0, s_1) \leftarrow\$ \mathsf{LREnc}(m_b)$.
- Compute $h_0 = \mathsf{Hash}(hk, s_0)$ and $h_1 = \mathsf{Hash}(hk, s_1)$; forward $(h_1, (s_0, h_0))$ and $(h_0, (s_1, h_1))$ to the challenger, obtaining arguments π_0 and π_1.
- Let $c^* = (c_0, c_1) = ((s_0, h_1, \pi_0, \pi_1), (s_1, h_0, \pi_0, \pi_1))$, and pick $j^* \leftarrow\$ [q]$ where $q \in \mathtt{poly}(\lambda)$ is an upper bound for the number of A^*'s tampering queries.
- Run $\mathsf{A}_1^{*\mathcal{O}^{\ell^*}_{\mathsf{leak}}(c_0, \cdot), \mathcal{O}^{\ell^*}_{\mathsf{leak}}(c_1, \cdot), \mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)}(1^\lambda, \alpha_1)$ by answering all of its leakage and tampering queries as follows:
  - · Upon input a leakage query g_0 (resp. g_1) for $\mathcal{O}^{\ell^*}_{\mathsf{leak}}(c_0, \cdot)$ (resp. $\mathcal{O}^{\ell^*}_{\mathsf{leak}}(c_1, \cdot)$), return $g_0(c_0)$ (resp. $g_1(c_1)$).
  - · Upon input a tampering query (f_0, f_1) for $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (\cdot, \cdot))$, if this is not the j^*-th tampering query answer in the same way as

$\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (\cdot, \cdot))$ would do. Else, run $\tilde{c}_\beta = (\tilde{s}_\beta, \tilde{h}_{1-\beta}, \tilde{\pi}_{0,\beta}, \tilde{\pi}_{1,\beta}) = f_\beta(c_\beta)$, compute $\tilde{h}_{\beta,\beta} = \mathsf{Hash}(hk, s_\beta)$ and forward $(\tilde{h}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{1,\beta})$ to the challenger.

For the analysis, note that the simulation done by A is perfect. In particular, both the distribution of the CRS and the answer to A^*'s leakage and tampering queries is identical to that of $\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$. It follows that, with probability at least $1/q \cdot 1/\mathtt{poly}(\lambda) \in 1/\mathtt{poly}(\lambda)$, the j^*-th tampering query output by attacker A^* is such that all of the following conditions are met: (i) $\mathsf{Ver}(\omega, \tilde{h}_{\beta,\beta}, (\tilde{h}_{1-\beta}, \tilde{\pi}_{1,\beta})) = 1$; (ii) $(\tilde{h}_{1-\beta}, \tilde{h}_{\beta,\beta}, \tilde{\pi}_{1-\beta,\beta})$ is fresh, i.e., it is different from $(h_{1-\beta}, h_\beta, \pi_{1-\beta})$; (iii) $\mathsf{K}(\xi, \tilde{h}_{\beta,\beta}, (\tilde{h}_{1-\beta}, \tilde{\pi}_{1,\beta}))$ outputs a value $\tilde{s}_{1-\beta}$ such that $\mathsf{Hash}(hk, \tilde{s}_{1-\beta}) \neq \tilde{h}_{1-\beta}$ (i.e., $(\tilde{h}_{1-\beta}, \tilde{s}_{1-\beta}) \notin \mathcal{R}^{hk}_{\mathsf{hash}}$). Further, note that A obtains only simulated arguments for true statements. Thus, A breaks true-simulation extractability with non-negligible probability. The lemma follows: □

4.5.2. *First Game Hop*

Lemma 5. $\forall b \in \{0, 1\}$: $\{\boldsymbol{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\boldsymbol{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\}_{\lambda\in\mathbb{N}}$.

Proof. The proof is down to the adaptive multi-theorem zero-knowledge property (cf. Definition 2) of the underlying non-interactive argument system. By contradiction, assume that there exists a PPT attacker $\mathsf{A}^* = (\mathsf{A}^*_0, \mathsf{A}^*_1)$ that, for any fixed $b \in \{0, 1\}$, can distinguish between $\{\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\}_{\lambda\in\mathbb{N}}$ and $\{\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\}_{\lambda\in\mathbb{N}}$ with non-negligible probability. Consider the following PPT distinguisher D attacking the zero-knowledge property.

<u>Distinguisher D for $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$:</u>

- Upon receiving the target CRS ω, generate $hk \leftarrow\$ \mathsf{Gen}(1^\lambda)$ and let $\omega^* := (\omega, hk)$.
- Run $(m_0, m_1, \alpha_1) \leftarrow\$ \mathsf{A}^*_0(1^\lambda, \omega^*)$, and let $(s_0, s_1) \leftarrow\$ \mathsf{LREnc}(m_b)$.
- For each $\beta \in \{0, 1\}$, compute $h_\beta = \mathsf{Hash}(hk, s_\beta)$.
- For each $\beta \in \{0, 1\}$, forward $(h_{1-\beta}, (h_\beta, s_\beta))$ to the challenger, obtaining π_0, π_1.
- Let $c^* = (c_0, c_1) = ((s_0, h_1, \pi_0, \pi_1), (s_1, h_0, \pi_0, \pi_1))$.
- Run $b' \leftarrow\$ \mathsf{A}_1^{*\mathcal{O}^{\ell^*}_{\mathsf{leak}}(c_0,\cdot),\mathcal{O}^{\ell^*}_{\mathsf{leak}}(c_1,\cdot),\mathcal{O}_{\mathsf{maul}}((c_0,c_1),\cdot)}(1^\lambda, \alpha_1)$.
- Output b'.

For the analysis, note that the simulation done by D is perfect. In particular, depending on the game that D is playing, the CRS ω is either generated by running $\mathsf{CRSGen}(1^\lambda)$ or $\mathsf{S}_0(1^\lambda)$; similarly, the arguments π_0, π_1 are either obtained by running $\pi_\beta \leftarrow\$ \mathsf{Prove}(\omega, h_{1-\beta}, (h_\beta, s_\beta))$ or $\pi_\beta \leftarrow\$ \mathsf{S}_1(\zeta, h_{1-\beta}, h_\beta)$. Moreover, the reduction can perfectly emulate the oracles $\mathcal{O}_{\mathsf{maul}}$ and $\mathcal{O}^{\ell^*}_{\mathsf{leak}}$ since it holds a simulated codeword c^* that is either distributed according to $\mathbf{Tamper}_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$ or $\mathbf{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)$. Thus, D has the same distinguishing advantage as that of A^*. The lemma follows: □

4.5.3. *Second Game Hop*

Lemma 6. $\forall b \in \{0, 1\}$: $\{\boldsymbol{Hyb}^1_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\boldsymbol{Hyb}^2_{\Gamma^*,\mathsf{A}^*}(\lambda, b)\}_{\lambda\in\mathbb{N}}$.

Proof. Fix $b \in \{0, 1\}$. We will prove that, with all but a negligible probability, the output of the tampering oracles $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)$ and $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), \cdot)$ are identical up to the first tampering query that is answered with $\perp$ in $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)$. (Afterward, both oracles self-destruct and thus always return $\perp$.) Let $q \in \mathtt{poly}(\lambda)$ be an upper bound on the number of tampering queries in either of the hybrids. Consider the following events, defined over the probability space of both $\mathbf{Hyb}^1_{\Sigma^*,\mathsf{A}}(\lambda, b)$ and $\mathbf{Hyb}^2_{\Sigma^*,\mathsf{A}}(\lambda, b)$:

- **Event** $\mathbf{Bad}_{\mathsf{same}}$: There exists an index $i \in [q]$, corresponding to a tampering query $(f_0^{(i)}, f_1^{(i)})$, such that oracle $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs $\diamond$, whereas oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ would output a value in $\{0, 1\}^{2n} \cup \{\perp\}$.
- **Event** $\mathbf{Bad}_{\perp}$: There exists an index $i \in [q]$, corresponding to a tampering query $(f_0^{(i)}, f_1^{(i)})$, such that oracle $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs $\perp$, whereas oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ would output a value in $\{0, 1\}^{2n} \cup \{\diamond\}$.
- **Event** $\mathbf{Bad}_{\mathsf{cdw}}$: There exists an index $i \in [q]$, corresponding to a tampering query $(f_0^{(i)}, f_1^{(i)})$, such that oracle $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs a codeword $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$, whereas oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ would output either a different codeword or a value in $\{\diamond, \perp\}$.

Denote by $\mathbf{Bad} := \mathbf{Bad}_{\mathsf{same}} \cup \mathbf{Bad}_{\perp} \cup \mathbf{Bad}_{\mathsf{cdw}}$. Since the two hybrids are identical conditioned on **Bad** not happening, by a standard argument, it suffices to prove that **Bad** happens with at most a negligible probability. In what follows, we always write

$$\tilde{c}_0^{(i)} := (\tilde{s}_0^{(i)}, \tilde{h}_1^{(i)}, \tilde{\pi}_{0,0}^{(i)}, \tilde{\pi}_{1,0}^{(i)}) \qquad \tilde{c}_1^{(i)} := (\tilde{s}_1^{(i)}, \tilde{h}_0^{(i)}, \tilde{\pi}_{0,1}^{(i)}, \tilde{\pi}_{1,1}^{(i)})$$

for the tampered codeword $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$ associated with $(f_0^{(i)}, f_1^{(i)})$, and $\tilde{h}_{\beta,\beta}$ for the value $\mathsf{Hash}(hk, \tilde{s}_\beta)$.

Claim 1. $\mathbb{P}[\mathbf{Bad}_{\mathsf{same}}] = 0$.

Proof. Note that $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs $\diamond$ if and only if $(f_0^{(i)}, f_1^{(i)})$ is of Type A, i.e., $\tilde{c}_0^{(i)} = f_0^{(i)}(c_0) = c_0$ and $\tilde{c}_1^{(i)} = f_1^{(i)}(c_1) = c_1$. However, for such a query, oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ would also return $\diamond$, and thus the event $\mathbf{Bad}_{\mathsf{same}}$ never happens. □

Claim 2. $\mathbb{P}[\mathbf{Bad}_{\perp}] \in \mathtt{negl}(\lambda)$.

Proof. Note that $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs $\perp$ if and only if either (i) $\exists \beta \in \{0, 1\} : \Theta_\beta[i] = \perp$, or if (ii) $\perp \neq \Theta_0[i] \neq \Theta_1[i] \neq \perp$. We treat each of these sub-cases separately.

The case $\exists \beta \in \{0, 1\} : \Theta_\beta[i] = \perp$. We observe that $\Theta_\beta[i] = \perp$ if and only if one of the following events happen: (a) $f_\beta^{(i)}$ is of Type B; (b) $f_\beta^{(i)}$ is of Type C; (c) $f_\beta^{(i)}$ is of Type D, and the extractor K returns $\perp$. Below, we analyze each such case.

Type-B queries: Recall that function $f_\beta^{(i)}$ is of Type B if:

$$\mathsf{Ver}(\omega, \tilde{h}_{1-\beta}^{(i)}, (\tilde{h}_{\beta,\beta}^{(i)}, \tilde{\pi}_{\beta,\beta}^{(i)})) = 0 \vee \mathsf{Ver}(\omega, \tilde{h}_{\beta,\beta}^{(i)}, (\tilde{h}_{1-\beta}^{(i)}, \tilde{\pi}_{1-\beta,\beta}^{(i)})) = 0.$$

The above means that the local check on $\tilde{c}_\beta^{(i)}$ (as run by the original decoding algorithm) fails, and thus $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ would also return $\perp$.

Type-C queries: Wlog. assume that $f_0^{(i)}$ is of Type C (the proof in case $f_1^{(i)}$ is of Type C being analogous). Recall that function $f_0^{(i)}$ is of Type C if:

$$(\tilde{h}_1^{(i)}, \tilde{h}_{0,0}^{(i)}, \tilde{\pi}_{1,0}^{(i)}) = (h_1, h_0, \pi_1).$$

Assume now that $(f_0^{(i)}, f_1^{(i)})$ is such that $\mathcal{O}_{\mathsf{maul}}$ returns a value different from $\perp$. We claim this implies that the tampered codeword $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$ must be of the form $((s_0, h_1, \tilde{\pi}_0^{(i)}, \pi_1), (s_1, h_0, \tilde{\pi}_0^{(i)}, \pi_1))$. First, collision resistance[14] of $(\mathsf{Gen}, \mathsf{Hash})$ implies that $\tilde{s}_0^{(i)} = s_0$ and $\tilde{s}_1^{(i)} = s_1$ with all but a negligible probability. Second, $\tilde{\pi}_{1,1}^{(i)} = \pi_1$ and $\tilde{\pi}_{0,0}^{(i)} = \tilde{\pi}_{0,1}^{(i)} = \tilde{\pi}_0^{(i)}$ as otherwise the tampered codeword would not be valid.

Next, we prove that the above contradicts security of the LRS. By contradiction, assume that there exists an adversary A^* that with non-negligible probability outputs a tampering query $(f_0^{(i)}, f_1^{(i)})$ such that $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)}) = ((s_0, h_1, \tilde{\pi}_0^{(i)}, \pi_1), (s_1, h_0, \tilde{\pi}_0^{(i)}, \pi_1))$, with $\tilde{\pi}_0^{(i)} \neq \pi_0$ (otherwise $f_0^{(i)}$ would be of Type A). Since $\tilde{\pi}_0^{(i)}$ is accepting, by true-simulation extractability of the argument system $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Ver})$, it holds that $\mathsf{K}(\xi, h_1, (h_0, \tilde{\pi}_0^{(i)}))$ outputs a value $\tilde{s}_0^{(i)}$ such that $\mathsf{Hash}(hk, \tilde{s}_0^{(i)}) = h_0$ with all but a negligible probability.[15] Moreover, collision resistance of $(\mathsf{Gen}, \mathsf{Hash})$ implies that in this case $\tilde{s}_0^{(i)} = s_0$ with overwhelming probability.[16] Consider now the following attacker A.

Attacker A against $(\mathsf{LREnc}, \mathsf{LRDec})$:

- Run steps 1–4 as described in the reduction A in the proof of Lemma 2. This yields target messages m_0, m_1 (as returned by A_0^*), the final corrected leakages Λ_0, Λ_1, the index i^* corresponding to the self-destruct query, and auxiliary information $\hat{\alpha} = (\omega, hk, \xi, h_0, h_1, \pi_0, \pi_1, \langle \mathsf{A}_1^* \rangle, \alpha_1, r_1)$.
- Forward $\sigma = 1$ to the challenger, obtaining s_1.
- Set $c_1 = (s_1, h_0, \pi_0, \pi_1)$, sample $j^* \leftarrow\!\!\$\ [i^*]$, and then start $\mathsf{A}_1^*(1^\lambda, \alpha_1; r_1)$.
- Upon input the j-th leakage query $(g_0^{(j)}, g_1^{(j)})$ from A_1^*, return $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- Upon input the i-th tampering query $(f_0^{(i)}, f_1^{(i)})$ from A_1^*:
  - · If $i \neq j^* < i^*$, answer with $\Theta_1^{(i)} = \mathsf{SimTamp}(c_1, f_1^{(i)})$.
  - · If $i = j^*$, let $\tilde{c}_1^{(i)} = (\tilde{s}_1^{(i)}, \tilde{h}_0^{(i)}, \tilde{\pi}_{0,1}^{(i)}, \tilde{\pi}_{1,1}^{(i)}) := f_1^{(i)}(s_1, h_0, \pi_0, \pi_1)$. Run $\tilde{s}_0^{(i)} := \mathsf{K}(\xi, h_1, (h_0, \tilde{\pi}_{0,1}^{(i)}))$. If there exists $b' \in \{0, 1\}$ such that

[14] Given an attacker A^* that with non-negligible probability outputs a tampering query $(f_0^{(i)}, f_1^{(i)})$ such that $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$ is valid with $\tilde{h}_0^{(i)} = h_0$ and $\tilde{s}_0^{(i)} \neq s_0$, we can build an attacker A breaking collision resistance of $(\mathsf{Gen}, \mathsf{Hash})$. The reduction is straightforward: Given hk, adversary A simply runs A^* honestly after embedding hk in the CRS, until it outputs the query $(f_0^{(i)}, f_1^{(i)})$, which yields the collision $(s_0, \tilde{s}_0^{(i)})$.

[15] The reduction is analogous to the one in Lemma 4, and therefore omitted.

[16] The reduction is analogous to the one in Footnote 14.

$\mathsf{LRDec}(\tilde{s}_0^{(i)}, \tilde{s}_1^{(i)}) = m_{b'}$, forward b' to the challenger and terminate; else, abort the simulation and terminate.

An argument identical to that used in the proof of Lemma 2 shows that A leaks at most ℓ bits and perfectly simulates the view of A^*. Since with probability $1/\mathtt{poly}(\lambda)$ attacker A^* outputs a tampering query $(f_0^{(i)}, f_1^{(i)})$ such that $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)}) = ((s_0, h_1, \tilde{\pi}_0^{(i)}, \pi_1), (s_1, h_0, \tilde{\pi}_0^{(i)}, \pi_1))$, with $\tilde{\pi}_0^{(i)} \neq \pi_0$, and moreover with probability $1/i^*$ this is the j^*-th tampering query, it follows that $b' = b$ in experiment $\mathbf{Leak}^+_{\Sigma,\mathsf{A}}(\lambda, b)$ with probability at least $1/\mathtt{poly}(\lambda) - \mathtt{negl}(\lambda)$.

Type-D queries: Wlog. assume that $f_0^{(i)}$ is of Type D (the proof in case $f_1^{(i)}$ is of Type D being analogous). Recall that function $f_0^{(i)}$ is of Type D if:

$$(\tilde{h}_1^{(i)}, \tilde{h}_{0,0}^{(i)}, \tilde{\pi}_{1,0}^{(i)}) \neq (h_1, h_0, \pi_1),$$

and moreover the argument $\tilde{\pi}_{1,0}^{(i)}$ is valid w.r.t. statement $\tilde{h}_1^{(i)}$ and label $\tilde{h}_{0,0}^{(i)}$. Lemma 4 implies that in such a situation the extractor K yields a valid witness, so that $\Theta_0[i] \neq \perp$ with overwhelming probability.

The case $\perp \neq \Theta_0[i] \neq \Theta_1[i] \neq \perp$. First note that both $\Theta_0[i]$ and $\Theta_1[i]$ must be different from $\diamond$. This is because if $\Theta_0[i] = \diamond$, it must be the case that $\Theta_1[i] \neq \diamond$, and thus $\tilde{c}_1^{(i)} = f_1^{(i)}(c_1) \neq c_1$ as otherwise $(f_0^{(i)}, f_1^{(i)})$ would be of Type A. Since $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ does not output $\perp$, the above implies that both (c_0, c_1) and $(c_0, \tilde{c}_1^{(i)})$ are valid codewords, which contradicts codewords uniqueness. More formally, given an efficient attacker A^* that provokes the above event, it is straightforward[17] to construct an efficient attacker A that breaks uniqueness, which in turn contradicts Lemma 3.

It remains to analyze the case where both $\Theta_0[i]$ and $\Theta_1[i]$ contain values in $\{0, 1\}^{2n}$, i.e., $\Theta_0[i] = (\tilde{c}_{0,0}^{(i)}, \tilde{c}_{1,0}^{(i)})$ and $\Theta_1[i] = (\tilde{c}_{0,1}^{(i)}, \tilde{c}_{1,1}^{(i)})$ with $(\tilde{c}_{0,0}^{(i)}, \tilde{c}_{1,0}^{(i)}) \neq (\tilde{c}_{0,1}^{(i)}, \tilde{c}_{1,1}^{(i)})$. By definition of $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$, for all $\beta \in \{0, 1\}$, we can write:

$$\tilde{c}_{\beta,\beta}^{(i)} = (\tilde{s}_\beta^{(i)}, \tilde{h}_{1-\beta}^{(i)}, \tilde{\pi}_{0,\beta}^{(i)}, \tilde{\pi}_{1,\beta}^{(i)}) \qquad \tilde{c}_{1-\beta,\beta}^{(i)} := (\tilde{s}_{1-\beta}^{(i)}, \tilde{h}_{\beta,\beta}^{(i)}, \tilde{\pi}_{0,\beta}^{(i)}, \tilde{\pi}_{1,\beta}^{(i)}).$$

However, since $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ does not return $\perp$, it must hold that $\tilde{\pi}_{0,0}^{(i)} = \tilde{\pi}_{0,1}^{(i)} := \tilde{\pi}_0^{(i)}$ and $\tilde{\pi}_{1,0}^{(i)} = \tilde{\pi}_{1,1}^{(i)} := \tilde{\pi}_1^{(i)}$, and furthermore $\tilde{h}_{0,0}^{(i)} = \tilde{h}_0^{(i)} := \mathsf{Hash}(hk, \tilde{s}_0^{(i)})$ and $\tilde{h}_{1,1}^{(i)} = \tilde{h}_1^{(i)} := \mathsf{Hash}(hk, \tilde{s}_1^{(i)})$. But this implies $\tilde{c}_{0,0} = (\tilde{s}_0^{(i)}, \tilde{h}_1^{(i)}, \tilde{\pi}_0^{(i)}, \tilde{\pi}_1^{(i)}) = \tilde{c}_{0,1}$, and similarly $\tilde{c}_{1,0} = (\tilde{s}_1^{(i)}, \tilde{h}_0^{(i)}, \tilde{\pi}_0^{(i)}, \tilde{\pi}_1^{(i)}) = \tilde{c}_{1,1}$, a contradiction. $\square$

Claim 3. $\mathbb{P}[\mathbf{Bad}_{\mathsf{cdw}}] \in \mathtt{negl}(\lambda)$.

[17] Attacker A simply runs A^* on a simulated codeword (c_0, c_1) generated exactly as in the last hybrid, and answers to both leakage and tampering queries honestly until A^* outputs the query $(f_0^{(i)}, f_1^{(i)})$ breaking uniqueness.

Proof. Note that $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$ if and only if $\Theta_0[i] = (\tilde{c}_{0,0}^{(i)}, \tilde{c}_{1,0}^{(i)})$, $\Theta_1[i] = (\tilde{c}_{0,1}^{(i)}, \tilde{c}_{1,1}^{(i)})$, and moreover $(\tilde{c}_{0,0}^{(i)}, \tilde{c}_{1,0}^{(i)}) = (\tilde{c}_{0,1}^{(i)}, \tilde{c}_{1,1}^{(i)}) := (\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$. Let us write $\tilde{c}_0^{(i)} = (\tilde{s}_0^{(i)}, \tilde{h}_1^{(i)}, \tilde{\pi}_0^{(i)}, \tilde{\pi}_1^{(i)})$ and $\tilde{c}_1^{(i)} = (\tilde{s}_1^{(i)}, \tilde{h}_0^{(i)}, \tilde{\pi}_0^{(i)}, \tilde{\pi}_1^{(i)})$. By Lemma 4, we know that $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$ is a valid codeword with overwhelming probability.

First, we claim that $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ cannot output $\diamond$. This is because the latter would imply that $\tilde{c}_0^{(i)} = c_0$ and $\tilde{c}_1^{(i)} = c_1$, which means $(f_0^{(i)}, f_1^{(i)})$ would be of Type A.

Second, let us argue that $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ cannot output $\perp$ either. This is because the latter would imply that either (i) $\exists \beta \in \{0, 1\} : \mathsf{Hash}(hk, \tilde{s}_\beta^{(i)}) \neq \tilde{h}_\beta^{(i)}$, or (ii) $\exists \beta \in \{0, 1\} : \mathsf{Ver}(\omega, \tilde{h}_\beta^{(i)}, (\tilde{h}_{1-\beta}^{(i)}, \tilde{\pi}_{1-\beta}^{(i)})) = 0$. However, both these conditions contradict the fact that $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$ must be valid.

It remains to consider the case where $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$ outputs a codeword $(\hat{c}_0^{(i)}, \hat{c}_1^{(i)}) \neq (\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)})$, where $(\hat{c}_0^{(i)}, \hat{c}_1^{(i)})$ is a valid codeword. Let us generically write $\hat{c}_0^{(i)} = f_0^{(i)}(c_0) = (\hat{s}_0^{(i)}, \hat{h}_1^{(i)}, \hat{\pi}_0^{(i)}, \hat{\pi}_1^{(i)})$ and $\hat{c}_1^{(i)} = f_1^{(i)}(c_1) = (\hat{s}_1^{(i)}, \hat{h}_0^{(i)}, \hat{\pi}_0^{(i)}, \hat{\pi}_1^{(i)})$ for the tampered codeword as computed by $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$. By definition of $\mathcal{O}'_{\mathsf{maul}}((c_0, c_1), (f_0^{(i)}, f_1^{(i)}))$, however, it must be the case that $\tilde{s}_0^{(i)} = \hat{s}_0^{(i)}$ and $\tilde{s}_1^{(i)} = \hat{s}_1^{(i)}$, which in turn implies $\tilde{h}_0^{(i)} = \mathsf{Hash}(hk, \tilde{s}_0^{(i)}) = \hat{h}_0^{(i)}$ and $\tilde{h}_1 = \mathsf{Hash}(hk, \tilde{s}_1) = \hat{h}_1^{(i)}$, and furthermore $(\tilde{\pi}_0^{(i)}, \tilde{\pi}_1^{(i)}) = (\hat{\pi}_0^{(i)}, \hat{\pi}_1^{(i)})$. Hence, $(\tilde{c}_0^{(i)}, \tilde{c}_1^{(i)}) = (\hat{c}_0^{(i)}, \hat{c}_1^{(i)})$, a contradiction. □

The statement of the lemma now follows by the above claims, together with a union bound. □

4.6. *Concrete Instantiation*

Below, we explain how to instantiate the building blocks required for our code, both from generic and concrete assumptions.

- **Leakage-Resilient Storage** As shown in [50,52], this primitive exists unconditionally. Let $\mathbb{F} = \mathbb{GF}(2^k)$ be the Galois field with 2^k elements, and $t \in \mathbb{N}$ be a parameter. Given $m \in \{0, 1\}^k$, algorithm LREnc interprets m as an element of $\mathbb{F}$ and outputs random $s_0, s_1 \in \mathbb{F}^t$ subject to $\langle s_0, s_1 \rangle := \sum_{i=1}^t s_0[i] \cdot s_1[i] = m$. For any $\delta \in [0, 1]$, this yields an ℓ-LRS with $\ell = kt/4 - \log(1/\delta)$, where δ is the statistical distance between the experiments of Definition 4. By Theorem 1, this is also an *augmented* $(kt/4 - \log(1/\delta) - 1)$-LRS.
- **Collision-Resistant Hash Function** Let $\mathbb{G}$ be a cyclic group of prime order q, with generators $g_1, \ldots, g_t$ for some parameter $t \in \mathbb{N}$. Assuming hardness of the Discrete Logarithm problem, we can take $\mathsf{Hash}(hk, s) := \prod_{i=1}^t g_i^{s[i]}$ where $s = (s[1], \ldots, s[t]) \in \mathbb{Z}_q^t$ and $\mathsf{Hash} : \mathbb{Z}_q^t \to \mathbb{G}$.
- **True-Simulation Extractable NIZK** It is well known that simulation extractable NIZKs (i.e., NIZKs that are extractable even in case the attacker observes simulated arguments of possibly false statements) can be obtained for all of NP assuming (doubly enhanced) trapdoor permutations [51,68,87]. More concretely, using the

collision-resistant hash function from the previous bullet point, we need to consider the NP-relation:

$$\mathcal{R}_{\mathsf{hash}}^{(\mathbb{G},g_1,\ldots,g_t,q)} := \left\{(h,s) : h = \prod_{i=1}^{t} g_i^{s[i]}\right\} \subseteq \mathbb{G} \times \mathbb{Z}_q^t.$$

Following [51], an efficient tSE NIZK for the above relation can be obtained under the DLIN assumption by combining classical Groth–Sahai NIZKs [74,75] with Cramer–Shoup encryption [38]. We leave the count of the exact number of group elements as an exercise to the reader.

5. Application to Tamper-Resilient Security

In this section we revisit the classical application of non-malleable codes to protecting arbitrary cryptographic primitives against memory-tampering attacks [55,56,84], highlighting in particular the benefits that our new notion of *continuous* non-malleability brings to the picture.

In what follows, let $\mathsf{F}(\kappa, \cdot)$ be an efficiently computable functionality taking a secret key $\kappa \in \{0,1\}^k$ and value $x \in \{0,1\}^*$ as input, and producing some output $y \in \{0,1\}^*$. We are concerned about attackers changing the original key κ into a related key $\tilde{\kappa}$, and observing the effect of such changes at the output. For concreteness, we will focus on the setting in which the key material is stored into two separate parts of the memory that can be tampered independently, but the results in this section can be easily extended to general tampering (parameterized by a tampering family supported by the non-malleable code). As explained below, we distinguish between stateless and stateful functionalities.

5.1. *Stateless Functionalities*

In the case of stateless functionalities, the key κ is fixed once and for all. The main idea is to transform the original functionality F into some kind of "hardened" functionality $\hat{\mathsf{F}}$ using a non-malleable code Γ to encode the key. Previous transformations aiming at protecting stateless functionalities [55,56,84] required to freshly re-encode the key κ each time the functionality is invoked. Our approach avoids the re-encoding of the key at each invocation, leading to a stateless transformation. This solves an open question from [55,56]. Moreover, we consider a setting where the encoded key is stored into a memory $\mathcal{M} := (\mathcal{M}_0, \mathcal{M}_1)$ which is much larger than the size needed to store the encoding itself (say $|\mathcal{M}_0| = |\mathcal{M}_1| = p(n)$ where $p(n)$ is polynomial in the length n of the encoding). When (perfect) erasures are not possible, this feature naturally allows the adversary to make copies of the initial encoding and tamper continuously with it.

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

Definition 8. (Stateless hardened functionality) Let $\Gamma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a split-state code in the CRS model, with k-bit messages and $2n$-bit codewords. Let $\mathsf{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$ be a *stateless* functionality, and $\delta \in \{0,1\}$ be a public value. We define a *stateless* hardened functionality $\hat{\mathsf{F}}^{\Gamma} := \hat{\mathsf{F}} : \{0,1\}^{2p} \times \{0,1\}^* \to \{0,1\}^*$ with $p = \texttt{poly}(n)$ as a tuple of algorithms $(\mathsf{Setup}, \mathsf{MemCompile}, \mathsf{Exec})$ described as follows:

- $\hat{\omega} \leftarrow\$ \mathsf{Setup}(1^{\lambda})$: Sample $\omega \leftarrow\$ \mathsf{Init}(1^{\lambda})$, initialize $\delta = 0$, and return $\hat{\omega} = (\omega, \delta)$.
- $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow\$ \mathsf{MemCompile}(\hat{\omega}, \kappa)$: Let $(c_0, c_1) \leftarrow\$ \mathsf{Enc}(\omega, \kappa)$. Parse $\hat{\omega} = (\omega, \delta)$. For $\beta \in \{0,1\}$, store c_β in the first n bits of $\mathcal{M}_\beta$; the remaining bits of $\mathcal{M}_\beta$ are set to 0^{p-n}. Define $\mathcal{M} := (\mathcal{M}_0, \mathcal{M}_1)$.
- $y \leftarrow\$ \mathsf{Exec}(\hat{\omega}, \mathsf{F}, \mathcal{M}, x)$: Parse $\hat{\omega} = (\omega, \delta)$. In case $\delta = 1$ output $\perp$; else, let $\tilde{c}_\beta = \mathcal{M}_\beta[1, \ldots, n]$ for $\beta \in \{0,1\}$. Run $\tilde{\kappa} = \mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1))$: If $\tilde{\kappa} = \perp$, then output $\perp$ and set $\delta = 1$. Else, output $y \leftarrow\$ \mathsf{F}(\kappa, x)$. Finally, update the CRS to $\hat{\omega} := (\omega, \delta)$.

Remark 1. (On δ) The public value δ 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 do not want to assume perfect erasures, we use the first approach here.

Note that we assume that δ is untamperable and one-time writable. The latter is necessary, as an adversary tampering with δ could always switch-off the self-destruct feature and apply a variant of the attack from [70] to recover the secret state.

Similarly to [55,56,84], security of $\hat{\mathsf{F}}$ is defined via the comparison of a real and an ideal experiment. The real experiment features an adversary A interacting with $\hat{\mathsf{F}}$; the adversary is allowed to honestly run the functionality on any chosen input, but also to modify the memory and retrieve a bounded amount of information from it. The ideal experiment features a simulator S that is given black-box access to the original functionality F and to the adversary A, but is *not* allowed any tampering or leakage query. The two experiments are formally described below.

Experiment $\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A}}(\lambda, \kappa)$. First $\hat{\omega} \leftarrow\$ \mathsf{Setup}(1^{\lambda})$ and $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow\$ \mathsf{MemCompile}(\hat{\omega}, \kappa)$ are run, and $\hat{\omega}$ is given to A. Then, A can issue the following commands polynomially many times (in any order):

- $\langle \texttt{Leak}, (g_0^{(j)}, g_1^{(j)}) \rangle$: In response to the j-th leakage query compute $\Lambda_0^{(j)} = g_0^{(j)}(\mathcal{M}_0)$ and $\Lambda_1^{(j)} = g_1^{(j)}(\mathcal{M}_1)$, and output $\Lambda^{(j)} := (\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
- $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$: In response to the j-th tampering query compute $\widetilde{\mathcal{M}}_0 = f_0^{(j)}(\mathcal{M}_0)$ and $\widetilde{\mathcal{M}}_1 = f_1^{(j)}(\mathcal{M}_1)$, and replace $\mathcal{M} = (\mathcal{M}_0, \mathcal{M}_1)$ with $\widetilde{\mathcal{M}} = (\widetilde{\mathcal{M}}_0, \widetilde{\mathcal{M}}_1)$.
- $\langle \texttt{Eval}, x^{(j)} \rangle$: In response to the j-th evaluation query run $y^{(j)} \leftarrow\$ \mathsf{Exec}(\hat{\omega}, \mathsf{F}, \mathcal{M}, x^{(j)})$: If $y^{(j)} = \perp$, output $\perp$ and self-destruct; else, output $y^{(j)}$.

At the end of the experiment, A outputs a bit that is an arbitrary function of its view (and this is also the output of the experiment).

Experiment $\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \kappa)$. The simulator sets up the CRS $\hat{\omega} = (\omega, \delta)$ and is given black-box access to the functionality $\mathsf{F}(\kappa, \cdot)$ and the adversary A. At the end of the experiment, A outputs a bit that is an arbitrary function of its view (and this is also the output of the experiment).

Definition 9. (Continuous tamper simulatability) Let Γ be a split-state encoding scheme in the CRS model and consider a stateless functionality F with corresponding hardened functionality $\hat{\mathsf{F}} := \hat{\mathsf{F}}^{\Gamma}$. We say that Γ is ℓ-leakage continuously tamper simulatable in the split-state model (for stateless functionalities), if for all PPT adversaries A leaking at most ℓ bits from each memory part, there exists a PPT simulator S such that for any initial state $\kappa \in \{0,1\}^k$:

$$\left\{\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A}}(\lambda,\kappa)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda,\kappa)\right\}_{\lambda\in\mathbb{N}}.$$

The theorem below says that any continuously non-malleable split-state code is automatically continuously tamper simulatable.

Theorem 4. *Let Γ be any ℓ-leakage-resilient continuously super*[18] *non-malleable split-state code in the CRS model. Then Γ is ℓ-leakage continuously tamper simulatable in the split-state model (for stateless functionalities).*

Proof. Consider the following simulator S. At the outset, S samples $\omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$ and sets $\delta = 0$, yielding $\hat{\omega} = (\omega, \delta)$. Then, it samples a random encoding of zero, namely $(c_0, c_1) \leftarrow\!\!\$\ \mathsf{Enc}(\omega, 0^k)$ and sets $\mathcal{M}_\beta[1,\ldots,n] := c_\beta$ for $\beta \in \{0,1\}$. The remaining bits of $(\mathcal{M}_0, \mathcal{M}_1)$ are set to 0^{p-n}. Next, S runs $\mathsf{A}(1^\lambda, \hat{\omega})$ and answers to its queries alternating between the following two modes (starting with the normal mode):

- **Normal Mode.** Given state $(\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle \texttt{Leak}, (g_0^{(j)}, g_1^{(j)})\rangle$: Upon input the j-th leakage query, compute $\Lambda_\beta^{(j)} = g_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$, and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
  - $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)})\rangle$: Upon input the j-th tampering query, compute $\widetilde{\mathcal{M}}_\beta = f_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$. Let $(\tilde{c}_0, \tilde{c}_1) := (\widetilde{\mathcal{M}}_0[1,\ldots,n], \widetilde{\mathcal{M}}_1[1,\ldots,n])$. If $(\tilde{c}_0, \tilde{c}_1) = (c_0, c_1)$, continue in the current mode. Else, go to the overwritten mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\widetilde{\mathcal{M}}_0, \widetilde{\mathcal{M}}_1)$, and, if $\mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1)) = \perp$, update the CRS to $\hat{\omega} := (\omega, 1)$.
  - $\langle \texttt{Eval}, x^j\rangle$: Upon input the j-th evaluation query, invoke $\mathsf{F}(\kappa, \cdot)$ to get $y^{(j)} \leftarrow\!\!\$\ \mathsf{F}(\kappa, x^{(j)})$ and reply with $y^{(j)}$.
- **Overwritten Mode.** Given state $\mathcal{M} := (\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle \texttt{Leak}, (g_0^{(j)}, g_1^{(j)})\rangle$: Upon input the j-th leakage query, compute $\Lambda_\beta^{(j)} = g_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$, and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
  - $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)})\rangle$: Upon input the j-th tampering query $(f_0^{(j)}, f_1^{(j)})$, compute $\widetilde{\mathcal{M}}_\beta = f_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$. Let $(\tilde{c}_0, \tilde{c}_1) := (\widetilde{\mathcal{M}}_0[1,\ldots,n], \widetilde{\mathcal{M}}_1[1,\ldots,n])$. If $(\tilde{c}_0, \tilde{c}_1) = (c_0, c_1)$, go to the normal mode

[18] We stress that weak non-malleability actually suffices for this application. We use super non-malleability for simplicity (and to be consistent with the rest of the paper).

with state $(\mathcal{M}_0, \mathcal{M}_1) := (\widetilde{\mathcal{M}}_0, \widetilde{\mathcal{M}}_1)$. Else, continue in the current mode, and, if $\mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1)) = \bot$, update the CRS to $\hat{\omega} := (\omega, 1)$.

· $\langle \texttt{Eval}, x^{(j)} \rangle$: Upon input the j-th evaluation query, run $y^{(j)} \leftarrow\$ \mathsf{Exec}(\hat{\omega}, \mathsf{F}, \mathcal{M}, x^{(j)})$ and reply with $y^{(j)}$.

Intuitively, since the code is non-malleable, the adversary can either keep the encoding of the secret key unchanged or overwrite it with the encoding of an unrelated value. These two cases are captured in the above modes: The simulator starts in the normal mode and then, whenever the adversary modifies the initial encoding, it switches to the overwritten mode. However, note that 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 feature is important to maintain simulation.

By contradiction, assume that there exists a PPT attacker A, and some key $\kappa \in \{0,1\}^k$, such that the following holds for the above defined simulator S:

$$\left| \mathbb{P}\left[\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A}}(\lambda, \kappa) = 1 \right] - \mathbb{P}\left[\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \kappa) = 1 \right] \right| \geq 1/\texttt{poly}(\lambda). \tag{1}$$

We build a PPT attacker A' breaking continuous non-malleability of the code Γ. A description of A' follows:

<u>Attacker A' for $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$:</u>

1. Upon receiving the CRS ω, forward $(m_0 := \kappa, m_1 := 0^k)$ to the challenger.
2. Initialize $\tilde{\kappa} := \varepsilon$, and $\texttt{mode}$ to $\texttt{normal}$. Set $\delta = 0$, and pass $\hat{\omega} := (\omega, \delta)$ to A.
3. Upon input a command $\langle \texttt{Tamper}, (f_0^{(j)}, f_1^{(j)}) \rangle$ from A, define the functions $f_{\mathsf{pad}} : \{0,1\}^n \to \{0,1\}^p$ and $f_{\mathsf{cut}} : \{0,1\}^p \to \{0,1\}^n$ as $f_{\mathsf{pad}}(x) = (x || 0^{p-n})$ and $f_{\mathsf{cut}}(x||x') = x$, for any $x \in \{0,1\}^n$ and $x' \in \{0,1\}^{p-n}$. Query the tampering oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)$ with the pair of functions $(\hat{f}_0^{(j)}, \hat{f}_1^{(j)})$, where for each $\beta \in \{0,1\}$ function $\hat{f}_\beta^{(j)}$ is defined as

$$\hat{f}_\beta^{(j)}(\cdot) := f_{\mathsf{cut}} \circ f_\beta^{(j)} \circ f_\beta^{(j-1)} \circ \cdots \circ f_\beta^{(1)} \circ f_{\mathsf{pad}}(\cdot). \tag{2}$$

   Then:

   - If the oracle returns $\bot$, update the CRS to $\hat{\omega} := (\omega, 1)$.
   - If the oracle returns $\diamond$, set $\texttt{mode}$ to $\texttt{normal}$.
   - If the oracle returns a codeword $\tilde{c}^{(j)} := (\tilde{c}_0^{(j)}, \tilde{c}_1^{(j)})$, set $\texttt{mode}$ to $\texttt{overwritten}$ and overwrite $\tilde{\kappa} := \mathsf{Dec}(\omega, (\tilde{c}_0^{(j)}, \tilde{c}_1^{(j)}))$.

4. Upon input a command $\langle \texttt{Leak}, (g_0^{(j)}, g_1^{(j)}) \rangle$ from A, query oracle $\mathcal{O}_{\mathsf{leak}}^{\ell}(c_0, \cdot)$ and $\mathcal{O}_{\mathsf{leak}}^{\ell}(c_1, \cdot)$ with functions $\hat{g}_0^{(j)}$ and $\hat{g}_1^{(j)}$, where for each $\beta \in \{0,1\}$ function $\hat{g}_\beta^{(j)}$ is defined as

$$\hat{g}_\beta^{(j)}(\cdot) := g_\beta^{(j)} \circ f_\beta^{(i)} \circ \cdots \circ f_\beta^{(1)} \circ f_{\mathsf{pad}}(\cdot), \tag{3}$$

where i is the index of the last tampering query. Then, forward the answer $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$ to A.

5. Upon input a command $\langle \texttt{Eval}, x^{(j)} \rangle$ from A, if $\delta = 1$ return $\perp$ to A. Else, proceed as follows:
   - If $\texttt{mode} = \texttt{normal}$, let $y^{(j)} \leftarrow\!\!\$\ \mathsf{F}(\kappa, x^{(j)})$ and return $y^{(j)}$ to A.
   - Else, if $\texttt{mode} = \texttt{overwritten}$, let $y^{(j)} \leftarrow\!\!\$\ \mathsf{F}(\tilde{\kappa}, x^{(j)})$ and return $y^{(j)}$ to A.
6. Upon receiving a guess b' from A, output b' and terminate.

For the analysis, first note that A' runs in polynomial time. Next, we claim that the simulation done by A' is perfect. This is because:

- The CRS $\hat{\omega}$ is distributed identically to the CRS in $\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A},\kappa}(\lambda)$ and $\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S},\kappa}(\lambda)$.
- Depending on the target encoding (c_0, c_1) being either an encoding of κ or an encoding of 0^k, the queries of A are answered exactly as in $\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A},\kappa}(\lambda)$ or in $\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S},\kappa}(\lambda)$. More precisely:
  - Tampering queries are handled using the oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)$. However, note that A' cannot directly forward the tampering functions $(f_0^{(j)}, f_1^{(j)})$ to this oracle. In fact: (i) Each of $f_0^{(j)}, f_1^{(j)}$ maps p bits into p bits, whereas A''s tampering oracle expects functions mapping n bits into n bits; (ii) In both $\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A},\kappa}(\lambda)$ and $\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S},\kappa}(\lambda)$ the tampering functions are applied to the current memory content $(\mathcal{M}_0, \mathcal{M}_1)$, whereas the tampering oracle always uses the target codeword (c_0, c_1).
    However, the reduction can easily handle this mismatch by padding each part of the target codeword with zeroes, recover the current memory, and define the tampered codeword to be the first n bits of $\widetilde{\mathcal{M}}_0 := f_0^{(j)}(\mathcal{M}_0)$ and $\widetilde{\mathcal{M}}_1 := f_1^{(j)}(\mathcal{M}_1)$ (cf. Eq. (2)).
  - Leakage queries are handled using the oracles $\mathcal{O}^{\ell}_{\mathsf{leak}}(c_0, \cdot)$ and $\mathcal{O}^{\ell}_{\mathsf{leak}}(c_1, \cdot)$. By a similar argument as above, the reduction perfectly emulates such queries by adjusting them as in Eq. (3). Furthermore, if A leaks at most ℓ bits from each side of the memory, A' leaks at most ℓ bits from each side of the target codeword.
  - Evaluation queries are perfectly emulated. Indeed, whenever $\texttt{mode}$ equals $\texttt{normal}$, the reduction answers evaluation queries by running F on key κ (this corresponds to the normal mode of simulator S). Similarly, whenever $\texttt{mode}$ equals $\texttt{overwritten}$, the reduction answers evaluation queries by running F on the current tampered key $\tilde{\kappa}$ which results from applying the tampering functions to the initial memory (this corresponds to the overwritten mode of simulator S).

It follows that in case (c_0, c_1) is an encoding of κ, the reduction perfectly emulates the view of A in $\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A},\kappa}(\lambda)$. Similarly, when (c_0, c_1) is an encoding of 0^k, the reduction perfectly emulates the view of A in $\mathbf{Ideal}_{\hat{\mathsf{F}},\mathsf{SA},\kappa}(\lambda)$. Hence, Eq. (1) implies

$$\left|\mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 0) = 1\right] - \mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}}(\lambda, 1) = 1\right]\right| \geq 1/\texttt{poly}(\lambda),$$

a contradiction. □

5.2. *Stateful Functionalities*

In the case of stateful functionalities, the function F has a secret state σ that is updated at each invocation, i.e., $(\sigma', y) \leftarrow\$ \mathsf{F}(\sigma, x)$. As the state gets updated, in this setting we inherently need to re-encode the new state after each execution.

Furthermore, since we do not assume erasures in our model, the following subtlety arises. An adversary can first copy some old (encoded) state to a different part of the memory, and later replace the current (encoded) state by the old one. This essentially corresponds to a reset attack, that cannot be simulated given only black-box access to the original functionality. To overcome this obstacle, our transformation leverages an untamperable *public* counter $\gamma \in \mathbb{N}$ that helps us detecting whenever the functionality is reset to a previous state, in which case a self-destruct is triggered; note that such a counter can be implemented using $\log(\lambda)$ bits (plus the additional bit to implement the self-destruct mechanism).[19]

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

Definition 10. (Stateful hardened functionality) Let $\Gamma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a split-state code in the CRS model, with $(k + \log(\lambda))$-bit messages and $2n$-bit codewords. Let $\mathsf{F} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k \times \{0,1\}^*$ be a *stateful* functionality, $\delta \in \{0,1\}$, $\gamma \in \{0,1\}^\lambda$ be a public values. We define a *stateful* hardened functionality $\hat{\mathsf{F}}^\Gamma := \hat{\mathsf{F}} : \{0,1\}^{2p} \times \{0,1\}^* \to \{0,1\}^{2p} \times \{0,1\}^*$ where $p = \mathtt{poly}(n)$ as a tuple of algorithms $(\mathsf{Setup}, \mathsf{MemCompile}, \mathsf{Exec})$ described as follows:

- $\hat{\omega} \leftarrow\$ \mathsf{Setup}(1^\lambda)$: Sample $\omega \leftarrow\$ \mathsf{Init}(1^\lambda)$, initialize $(\gamma, \delta) = (1, 0)$, and return $\hat{\omega} := (\omega, \gamma, \delta)$.
- $(\mathcal{M}_0, \mathcal{M}_1) \leftarrow\$ \mathsf{MemCompile}(\hat{\omega}, \sigma)$: Parse $\hat{\omega} = (\omega, \gamma, \delta)$. Let $(c_0, c_1) \leftarrow\$ \mathsf{Enc}(\omega, \sigma || 1)$. For $\beta \in \{0,1\}$, store c_β in the first n bits of $\mathcal{M}_\beta$; the remaining bits of $\mathcal{M}_\beta$ are set to 0^{p-n}. Define $\mathcal{M} := (\mathcal{M}_0, \mathcal{M}_1)$.
- $y \leftarrow\$ \mathsf{Exec}(\hat{\omega}, \mathsf{F}, \mathcal{M}, x)$: Parse $\hat{\omega} = (\omega, \gamma, \delta)$. In case $\delta = 1$ output $\perp$; else, let $\tilde{c}_\beta = \mathcal{M}_\beta[1, \ldots, n]$ for $\beta \in \{0,1\}$. Run $\tilde{\sigma} || \tilde{\gamma} = \mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1))$: If either $\tilde{\sigma} = \perp$ or $\tilde{\gamma} \neq \gamma$, then output $\perp$ and set $\delta = 1$. Else, output y such that $(y, \sigma') \leftarrow\$ \mathsf{F}(\tilde{\sigma}, x)$, run $(c'_0, c'_1) \leftarrow\$ \mathsf{Enc}(\omega, \sigma' || \gamma + 1)$, and for each $\beta \in \{0,1\}$ write c'_β in the first n bits of $\mathcal{M}_\beta$. Finally, update the CRS to $\hat{\omega} := (\omega, \gamma + 1, \delta)$.

Remark 2. (On (γ, δ)) As in the case of stateless functionalities, we require δ to be untamperable and one-time writable (cf. Remark 1). Moreover, the counter γ is assumed to be untamperable and poly-time writable. The latter is necessary, as otherwise the attacker can use the extra memory to reset the functionality to a previous valid state, which cannot be simulated with black-box access to the original functionality.

[19] Without assuming an untamperable counter, one can still get continuous tamper simulatability if the simulator is allowed to reset the original functionality. This essentially allows to protect cryptographic primitives with so-called resettable security (see, e.g., [19,72,88]).

Security of a stateful hardened functionality is defined analogously to the stateless case (cf. Definition 9). We show the following result:

Theorem 5. *Let Γ be any ℓ-leakage-resilient continuously super non-malleable split-state code in the CRS model. Then Γ is ℓ-leakage continuously tamper simulatable in the split-state model (for stateful functionalities).*

Proof. Similarly to the proof of Theorem 4, we describe a simulator S running in experiment $\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$ which simulates the view of adversary A in the experiment $\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A}}(\lambda, \sigma)$ for every possible initial state $\sigma \in \{0,1\}^k$. As usual, the simulator S is given black-box access to A (which can issue `Tamper`, `Leak`, and `Eval` commands), and to the reactive functionality $\mathsf{F}(\sigma, \cdot)$ with initial state σ. In order to simplify the simulator, we are going to assume that the attacker A issues commands in rounds, where each round consist exactly of one leakage query, one tampering query, and one execute query (in this order). A generalization is straightforward.

At the outset, S samples $\omega \leftarrow\!\$\ \mathsf{Init}(1^\lambda)$ and sets $(\gamma, \delta) = (1, 0)$, yielding $\hat{\omega} := (\omega, \gamma, \delta)$. Next, S runs $\mathsf{A}(1^\lambda, \hat{\omega})$ and answers to its queries using the following two modes (starting with the normal mode):

- **Normal Mode.** Given state $(\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - Sample $(c_0^{(j)}, c_1^{(j)}) \leftarrow\!\$\ \mathsf{Enc}(\omega, 0^k || 0^{\log \lambda})$, and let $\mathcal{M}_\beta[1, \ldots, n] := c_\beta$ for $\beta \in \{0,1\}$. If $j = 1$ (i.e., during the first round), the remaining bits of $(\mathcal{M}_0, \mathcal{M}_1)$ are set to 0^{p-n}.
  - $\langle$`Leak`$, (g_0^{(j)}, g_1^{(j)})\rangle$: Upon input the j-th leakage query, compute $\Lambda_\beta^{(j)} = g_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$, and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
  - $\langle$`Tamper`$, (f_0^{(j)}, f_1^{(j)})\rangle$: Upon input the j-th tampering query, compute $\widetilde{\mathcal{M}}_\beta = f_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$. Let $(\tilde{c}_0, \tilde{c}_1) := (\widetilde{\mathcal{M}}_0[1, \ldots, n], \widetilde{\mathcal{M}}_1[1, \ldots, n])$. If $(\tilde{c}_0, \tilde{c}_1) = (c_0^{(j)}, c_1^{(j)})$, continue in the current mode. Else, go to the overwritten mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\widetilde{\mathcal{M}}_0, \widetilde{\mathcal{M}}_1)$, and, if $\mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1)) \in \{\bot, \tilde{\sigma}||\tilde{\gamma}\}$ for some $\tilde{\gamma} \neq \gamma$, update the CRS to $\hat{\omega} := (\omega, \gamma, 1)$.
  - $\langle$`Eval`$, x^{(j)}\rangle$: Upon input the j-th evaluation query, invoke $\mathsf{F}(\sigma, \cdot)$ to obtain $y^{(j)}$ s.t. $(\sigma', y^{(j)}) \leftarrow\!\$\ \mathsf{F}(\sigma, x^{(j)})$, where σ is the current state of the functionality and σ' is the new state[20] (which is unknown to S), and reply with $y^{(j)}$. Update the CRS to $\hat{\omega} := (\omega, \gamma + 1, \delta)$.
- **Overwritten Mode.** Given state $\mathcal{M} := (\mathcal{M}_0, \mathcal{M}_1)$, while A continues issuing queries, answer as follows:
  - $\langle$`Leak`$, (g_0^{(j)}, g_1^{(j)})\rangle$: Upon input the j-th leakage query, compute $\Lambda_\beta^{(j)} = g_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$, and reply with $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$.
  - $\langle$`Tamper`$, (f_0^{(j)}, f_1^{(j)})\rangle$: Upon input the j-th tampering query $(f_0^{(j)}, f_1^{(j)})$, compute $\widetilde{\mathcal{M}}_\beta = f_\beta^{(j)}(\mathcal{M}_\beta)$ for $\beta \in \{0,1\}$. Let $(\tilde{c}_0, \tilde{c}_1) := (\widetilde{\mathcal{M}}_0[1, \ldots, n],$

[20] After such a query, the functionality internally re-defines $\sigma := \sigma'$.

$\widetilde{\mathcal{M}}_1[1,\ldots,n]$). Continue in the current mode with state $(\mathcal{M}_0, \mathcal{M}_1) := (\widetilde{\mathcal{M}}_0, \widetilde{\mathcal{M}}_1)$, and, if $\mathsf{Dec}(\omega, (\tilde{c}_0, \tilde{c}_1)) \in \{\bot, \tilde{\sigma}||\tilde{\gamma}\}$ for some $\tilde{\gamma} \neq \gamma$, update the CRS to $\hat{\omega} := (\omega, \gamma, 1)$.

· $\langle \mathtt{Eval}, x^{(j)}\rangle$: Upon input the j-th evaluation query, run $(\sigma', y^{(j)}) \leftarrow\!\!\$\ \mathsf{Exec}(\hat{\omega}, \mathsf{F}, \mathcal{M}, x^{(j)})$ and reply with $y^{(j)}$. Compute $(c_0', c_1') \leftarrow\!\!\$\ \mathsf{Enc}(\omega, \sigma'||\gamma + 1)$, and update the first n bits of $\mathcal{M}_\beta$ to c_β for each $\beta \in \{0, 1\}$, and the CRS to $\hat{\omega} := (\omega, \gamma + 1, \delta)$.

The intuition behind the above simulation strategy is similar to that in the proof of Theorem 4. One important difference is the fact that, after the simulator switches to the overwritten mode, it never goes back to the normal mode. Intuitively, this is because the untamperable counter ensures that the attacker can never copy an old state back to the first part of the memory. A further difficulty in the analysis is that in the case of stateful functionalities the state is re-encoded after each invocation, and thus we cannot directly reduce to the security of the underlying continuously non-malleable code.

Experiment $\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$. In order to overcome the above difficulty, we consider a sequence of mental experiments. Let $q \in \mathtt{poly}(\lambda)$ be an upper bound on the total number of queries (and thus rounds). For all $i \in [q]$, consider the following hybrid experiment where the simulator runs a modified normal mode and the same overwritten mode.

- First $\hat{\omega} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ is run, and $\hat{\omega}$ is given to A.
- For the first i rounds, A's commands are answered exactly as in experiment $\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$.
- Starting from round $i+1$, if the simulator S is already in the overwritten mode, then continue simulating as in the ideal world. Otherwise, proceed with the following modified normal mode:
  · Sample $(c_0^{(j)}, c_1^{(j)}) \leftarrow\!\!\$\ \mathsf{Enc}(\omega, \sigma||\gamma)$, and let $\mathcal{M}_\beta[1,\ldots,n] := c_\beta$ for $\beta \in \{0, 1\}$. If $j = 1$ (i.e., during the first round), the remaining bits of $(\mathcal{M}_0, \mathcal{M}_1)$ are set to 0^{p-n}.
  · $\langle \mathtt{Leak}, (g_0^{(j)}, g_1^{(j)})\rangle$: As in the ideal world.
  · $\langle \mathtt{Tamper}, (f_0^{(j)}, f_1^{(j)})\rangle$: As in the ideal world.
  · $\langle \mathtt{Eval}, x^{(j)}\rangle$: Upon input the j-th evaluation query, run $(\sigma', y^{(j)}) \leftarrow\!\!\$\ \mathsf{F}(\sigma, x^{(j)})$ where σ is the current state of the functionality and σ' is the new state, and reply with $y^{(j)}$. Redefine $\sigma := \sigma'$, and update the CRS to $\hat{\omega} := (\omega, \gamma := \gamma + 1, \delta)$.

Clearly, $\{\mathbf{Hyb}^{(q)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{Ideal}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)\}_{\lambda\in\mathbb{N}}$. Moreover, $\{\mathbf{Hyb}^{(0)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{Real}_{\hat{\mathsf{F}},\mathsf{A}}(\lambda, \sigma)\}_{\lambda\in\mathbb{N}}$. This is because the modified normal mode of the simulator corresponds to answering A's commands exactly as in the real experiment. Hence, the theorem follows by the lemma below.

Lemma 7. $\forall \sigma \in \{0, 1\}^k, \forall i \in [q] :\ \{\boldsymbol{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\boldsymbol{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)\}_{\lambda\in\mathbb{N}}$.

Proof. Fix $i \in [q]$. By contradiction, assume that there exists a PPT attacker A, and some state $\sigma \in \{0,1\}^k$, such that the following holds for the above defined simulator S:

$$\left|\mathbb{P}\left[\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda,\sigma)=1\right]-\mathbb{P}\left[\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda,\sigma)=1\right]\right| \geq 1/\mathtt{poly}(\lambda). \tag{4}$$

We build a PPT attacker A' breaking continuous non-malleability of the code Γ. A description of A' follows:

<u>Attacker A' for $(\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$:</u>

1. Upon receiving the CRS ω, set $(\gamma,\delta)=(1,0)$, and pass $\hat{\omega}:=(\omega,\gamma,\delta)$ to A.
2. For each round $j \leq i-1$, reply to A's commands as described in the ideal experiment. Let $\sigma := \sigma^{(i)}$ and $\hat{\omega}=(\omega,i,\delta)$ be, respectively, the current state[21] of the functionality and the CRS at the end of the $(i-1)$-th round.
3. Forward $(m_0 := \sigma^{(i)}||i, m_1 := 0^k||0^{\log\lambda})$ to the challenger, and initialize $\tilde{\sigma}||\tilde{\gamma} := \varepsilon$ and $\mathtt{mode}$ to $\mathtt{normal}$. In case the simulation of the first $i-1$ tampering queries has already caused the simulator to enter the overwritten mode, output a random b' and terminate; else, continue.
4. For each round $j \geq i$, upon input a command $\langle\mathtt{Tamper}, (f_0^{(j)}, f_1^{(j)})\rangle$ from A, query the tampering oracle $\mathcal{O}_{\mathsf{maul}}((c_0,c_1),\cdot)$ with the pair of functions $(\hat{f}_0^{(j)}, \hat{f}_1^{(j)})$ of Eq. (2). Then:
   - If the oracle returns $\bot$, update the CRS to $\hat{\omega} := (\omega,\gamma,1)$.
   - If the oracle returns $\diamond$, do nothing.
   - If the oracle returns a codeword $\tilde{c}^{(j)} := (\tilde{c}_0^{(j)}, \tilde{c}_1^{(j)})$, set $\mathtt{mode}$ to $\mathtt{overwritten}$ and overwrite $\tilde{\sigma}||\tilde{\gamma} := \mathsf{Dec}(\omega, (\tilde{c}_0^{(j)}, \tilde{c}_1^{(j)}))$. If $\tilde{\gamma} \neq \gamma$, update the CRS to $\hat{\omega} := (\omega,\gamma,1)$.
5. For each round $j \geq i$, upon input a command $\langle\mathtt{Leak}, (g_0^{(j)}, g_1^{(j)})\rangle$ from A, query oracle $\mathcal{O}^{\ell}_{\mathsf{leak}}(c_0,\cdot)$ and $\mathcal{O}^{\ell}_{\mathsf{leak}}(c_1,\cdot)$ with the functions $\hat{g}_0^{(j)}$ and $\hat{g}_1^{(j)}$ of Eq. (3). Then, forward the answer $(\Lambda_0^{(j)}, \Lambda_1^{(j)})$ to A.
6. For each round $j \geq i$, upon input a command $\langle\mathtt{Eval}, x^{(j)}\rangle$ from A, if $\delta = 1$ return $\bot$ to A. Else, proceed as follows:
   - If $\mathtt{mode} = \mathtt{normal}$, let $(\sigma', y^{(j)}) \leftarrow\$ \mathsf{F}(\sigma, x^{(j)})$ and return $y^{(j)}$ to A.
   - If $\mathtt{mode} = \mathtt{overwritten}$, let $(\sigma', y^{(j)}) \leftarrow\$ \mathsf{F}(\tilde{\sigma}, x^{(j)})$ and return $y^{(j)}$ to A.
   - Update the CRS to $\hat{\omega} = (\omega, \gamma+1, \delta)$, and re-define $\sigma := \sigma'$.
7. Upon receiving a guess b' from A, output b' and terminate.

For the analysis, first note that A' runs in polynomial time. Next, we claim that the simulation done by A' is perfect. This is because:

[21] Note that the reduction can compute $\sigma^{(i)}$, as it receives the initial state σ as auxiliary input.

- The CRS $\hat{\omega}$ is distributed identically to the CRS in $\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$ and $\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$.
- The first $i-1$ leakage/tampering/execute queries are answered as in the ideal experiment, which corresponds to the way such queries are answered in both $\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$ and $\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$.
- Let $\mathbf{W}$ be the event that the simulation has already entered the overwritten mode while answering the first $i-1$ tampering queries. Note that conditioning on $\mathbf{W}$, the random variables $\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$ and $\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$ are identical, and thus Eq. (4) implies that $\mathbb{P}[\neg\mathbf{W}] \geq 1/\mathtt{poly}(\lambda)$.
- For each round $j \geq i$, conditioning on $\neg\mathbf{W}$ and depending on the target encoding (c_0, c_1) being either an encoding of $\sigma^{(i)}||i$ or an encoding of $0^k||0^{\log\lambda}$, the queries of A are answered exactly as in $\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$ or in $\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$. More precisely:
  - The remaining leakage and tampering queries are handled using the oracle $\mathcal{O}_{\mathsf{maul}}((c_0, c_1), \cdot)$. As in the proof of Theorem 4, the reduction needs to adjust the leakage/tampering functions taking into account the initial padding of the memory and the previous tampering queries. This can be done using Eq. (2) and Eq. (3) as before.
  - The remaining evaluation queries are perfectly emulated. Indeed, whenever `mode` equals `normal`, the reduction answers evaluation queries by running F on the current state σ (this corresponds to the normal mode of simulator S). Similarly, whenever `mode` equals `overwritten`, the reduction answers evaluation queries by running F on the current tampered state $\tilde{\sigma}$ which results from applying the tampering functions to the initial memory (this corresponds to the overwritten mode of simulator S).

It follows that in case (c_0, c_1) is an encoding of $\sigma^{(i)}||i$ and conditioning on $\neg\mathbf{W}$, the reduction perfectly emulates the view of A in $\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$. Similarly, when (c_0, c_1) is an encoding of $0^k||0^{\log\lambda}$ and conditioning on $\neg\mathbf{W}$, the reduction perfectly emulates the view of A in $\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma)$. Finally, $\mathbb{P}[\neg\mathbf{W}] \geq 1/\mathtt{poly}(\lambda)$, and additionally conditioning on $\mathbf{W}$ we have that $\mathbb{P}\left[b' = b\right] = 1/2$. Hence, we have obtained:

$$\begin{aligned}
&\left|\mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 0) = 1\right] - \mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 1) = 1\right]\right| \\
&= \Big|\mathbb{P}[\mathbf{W}] \cdot \mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 0) = 1|\mathbf{W}\right] + \mathbb{P}[\neg\mathbf{W}] \cdot \mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 0) = 1|\neg\mathbf{W}\right] \\
&\quad - \mathbb{P}[\mathbf{W}] \cdot \mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 1) = 1|\mathbf{W}\right] - \mathbb{P}[\neg\mathbf{W}] \cdot \mathbb{P}\left[\mathbf{Tamper}_{\Gamma,\mathsf{A}'}(\lambda, 1) = 1|\neg\mathbf{W}\right]\Big| \\
&\geq \frac{1}{\mathtt{poly}(\lambda)} \cdot \left|\mathbb{P}\left[\mathbf{Hyb}^{(i-1)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma) = 1\right] - \mathbb{P}\left[\mathbf{Hyb}^{(i)}_{\mathsf{F},\mathsf{A},\mathsf{S}}(\lambda, \sigma) = 1\right]\right| \\
&\geq 1/\mathtt{poly}(\lambda),
\end{aligned}$$

a contradiction. This concludes the proof. □

Acknowledgements

Open access funding provided by Università degli Studi di Roma La Sapienza within the CRUI-CARE Agreement.

References

[1] D. Aggarwal, Affine-evasive sets modulo a prime. *Inf. Process. Lett.* **115**(2), 382–385 (2015).
[2] D. Aggarwal, S. Agrawal, D. Gupta, H. K. Maji, O. Pandey, M. Prabhakaran, Optimal computational split-state non-malleable codes, in *TCC* (2016), pp. 393–417.
[3] D. Aggarwal, Y. Dodis, T. Kazana, M. Obremski, Non-malleable reductions and applications, in *STOC* (2015), pp. 459–468.
[4] D. Aggarwal, Y. Dodis, S. Lovett, Non-malleable codes from additive combinatorics, in *STOC* (2014), pp. 774–783.
[5] D. Aggarwal, Y. Dodis, S. Lovett, Non-malleable codes from additive combinatorics. *SIAM J. Comput.* **47**(2), 524–546 (2018).
[6] D. Aggarwal, N. Döttling, J, B. Nielsen, M. Obremski, E. Purwanto, Continuous non-malleable codes in the 8-split-state model, in *EUROCRYPT* (2019), pp. 531–561.
[7] D. Aggarwal, S. Dziembowski, T. Kazana, M. Obremski, Leakage-resilient non-malleable codes, in *TCC* (2015), pp. 398–426.
[8] D. Aggarwal, T. Kazana, M. Obremski, Inception makes non-malleable codes stronger, in *TCC* (2017), pp. 319–343.
[9] S. Agrawal, D. Gupta, H. K. Maji, O. Pandey, M. Prabhakaran, Explicit non-malleable codes against bit-wise tampering and permutations, in *CRYPTO* (2015), pp. 538–557.
[10] S. Agrawal, D. Gupta, H. K. Maji, O. Pandey, M. Prabhakaran, A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations, in *TCC* (2015), pp. 375–397.
[11] P. Austrin, K.-M. Chung, M. Mahmoody, R. Pass, K. Seth, On the impossibility of cryptography with tamperable randomness, in *CRYPTO* (2014), pp. 462–479.
[12] M. Ball, D. Dachman-Soled, S. Guo, T. Malkin, L.-Y. Tan, Non-malleable codes for small-depth circuits, in *FOCS* (2018), pp. 826–837.
[13] M. Ball, D. Dachman-Soled, M. Kulkarni, H. Lin, T. Malkin, Non-malleable codes against bounded polynomial time tampering, in *EUROCRYPT* (2019), pp. 501–530.
[14] M. Ball, D. Dachman-Soled, M. Kulkarni, T. Malkin, Non-malleable codes for bounded depth, bounded fan-in circuits, in *EUROCRYPT* (2016), pp. 881–908.
[15] M. Ball, D. Dachman-Soled, M. Kulkarni, T. Malkin. Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering, in *EUROCRYPT* (2018), pp. 618–650.
[16] M. Ball, D. Dachman-Soled, M. Kulkarni, T. Malkin, Limits to non-malleability, in *ITCS* (2020), pp. 80:1–80:32.
[17] M. Ball, S. Guo, D. Wichs, Non-malleable codes for decision trees, in *CRYPTO* (2019), pp. 413–434.
[18] M. Bellare, D. Cash, R. Miller, Cryptography secure against related-key attacks and tampering, in *ASIACRYPT* (2011), pp. 486–503.
[19] M. Bellare, M. Fischlin, S. Goldwasser, S. Micali, Identification protocols secure against reset attacks, in *EUROCRYPT* (2001), pp. 495–511.

[20] M. Bellare, T. Kohno, A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications, in *EUROCRYPT* (2003), pp. 491–506.

[21] M. Bellare, K. G. Paterson, S. Thomson, RKA security beyond the linear barrier: IBE, encryption and signatures, in *ASIACRYPT* (2012), pp. 331–348.

[22] D. Boneh, R. A. DeMillo, R. J, Lipton. On the importance of eliminating errors in cryptographic computations, *J. Cryptol.* **14**(2), 101–119 (2001).

[23] G. Brian, A. Faonio, D. Venturi, Continuously non-malleable secret sharing for general access structures, in *TCC* (2019), pp. 211–232.

[24] N. Chandran, V. Goyal, P. Mukherjee, O. Pandey, J. Upadhyay, Block-wise non-malleable codes, in *ICALP* (2016), pp. 31:1–31:14.

[25] N. Chandran, B. Kanukurthi, S. Raghuraman Information-theoretic local non-malleable codes and their applications, in *TCC* (2016), pp. 367–392.

[26] E. Chattopadhyay, V. Goyal, X. Li, Non-malleable extractors and codes, with their many tampered extensions, in *STOC* (2016), pp. 285–298.

[27] E. Chattopadhyay, X. Li, Non-malleable codes and extractors for small-depth circuits, and affine functions, in *STOC* (2017), pp. 1171–1184.

[28] E. Chattopadhyay, D. Zuckerman, Non-malleable codes against constant split-state tampering, in *FOCS* (2014), pp. 306–315.

[29] B. Chen, Y. Chen, K. Hostáková, P. Mukherjee, Continuous space-bounded non-malleable codes from stronger proofs-of-space, in *CRYPTO* (2019), pp. 467–495.

[30] M. Cheraghchi, V. Guruswami, Non-malleable coding against bit-wise and split-state tampering, in *TCC* (2014), pp. 440–464.

[31] M. Cheraghchi, V. Guruswami, Capacity of non-malleable codes. *IEEE Trans. Inf. Theory*, **62**(3), 1097–1118 (2016).

[32] M. Cheraghchi, V. Guruswami, Non-malleable coding against bit-wise and split-state tampering, *J. Cryptol.* **30**(1), 191–241 (2017).

[33] S. G. Choi, A. Kiayias, T. Malkin, BiTR: Built-in tamper resilience, in *ASIACRYPT* (2011), pp. 740–758.

[34] S. Coretti, Y. Dodis, B. Tackmann, D. Venturi. Non-malleable encryption: simpler, shorter, stronger, in *TCC* (2016), pp. 306–335.

[35] S. Coretti, A. Faonio, D. Venturi, Rate-optimizing compilers for continuously non-malleable codes, in *ACNS* (2019), pp. 3–23.

[36] S. Coretti, U. Maurer, B. Tackmann, D. Venturi, From single-bit to multi-bit public-key encryption via non-malleable codes, in *TCC* (2015), pp. 532–560.

[37] R. Cramer, I. Damgård, N. Döttling, I. Giacomelli, C. Xing, Linear-time non-malleable codes in the bit-wise independent tampering model, in *ICITS* (2017), pp. 1–25.

[38] R. Cramer, V. Shoup, A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, in *CRYPTO* (1998), pp. 13–25.

[39] D. Dachman-Soled, Y. T. Kalai, Securing circuits against constant-rate tampering, in *CRYPTO* (2012), pp. 533–551.

[40] D. Dachman-Soled, Y. T. Kalai, Securing circuits and protocols against 1/poly(k) tampering rate, in *TCC* (2014), pp. 540–565.

[41] D. Dachman-Soled, M. Kulkarni, Upper and lower bounds for continuous non-malleable codes, in *PKC* (2019), pp. 519–548.

[42] D. Dachman-Soled, M. Kulkarni, A. Shahverdi, Tight upper and lower bounds for leakage-resilient, locally decodable and updatable non-malleable codes, in *PKC* (2017), pp. 310–332.

[43] D. Dachman-Soled, M. Kulkarni, A. Shahverdi, Local non-malleable codes in the bounded retrieval model, in *PKC* (2018), pp. 281–311.

[44] D. Dachman-Soled, M. Kulkarni, A. Shahverdi, Tight upper and lower bounds for leakage-resilient, locally decodable and updatable non-malleable codes, *Inf. Comput.* p. 268 (2019)

[45] D. Dachman-Soled, F.-H. Liu, E. Shi, H.-S. Zhou, Locally decodable and updatable non-malleable codes and their applications, in *TCC* (2015), pp. 427–450.

[46] D. Dachman-Soled, F.-H. Liu, E. Shi, H.-S. Zhou, Locally decodable and updatable non-malleable codes and their applications. *J. Cryptol.* **33**(1), 319–355 (2020).

[47] I. Damgård, S. Faust, P. Mukherjee, D. Venturi, Bounded tamper resilience: How to go beyond the algebraic barrier, in *ASIACRYPT* (2013), pp. 140–160.

[48] I. Damgård, S. Faust, P. Mukherjee, D. Venturi, The chaining lemma and its application, in *ICITS* (2015), pp. 181–196.

[49] I. Damgård, T. Kazana, M. Obremski, V. Raj, L. Siniscalchi, Continuous NMC secure against permutations and overwrites, with applications to CCA secure commitments, in *TCC* (2018), pp. 225–254.

[50] F. Davì, S. Dziembowski, D. Venturi, Leakage-resilient storage, in *SCN* (2010), pp. 121–137.

[51] Y. Dodis, K. Haralambiev, A. López-Alt, D. Wichs, Efficient public-key cryptography in the presence of key leakage, in *ASIACRYPT* (2010), pp. 613–631.

[52] S. Dziembowski, S. Faust, Leakage-resilient cryptography from the inner-product extractor, in *ASIACRYPT* (2011), pp. 702–721.

[53] S. Dziembowski, T. Kazana, M. Obremski, Non-malleable codes from two-source extractors, in *CRYPTO* (2013), pp. 239–257.

[54] S. Dziembowski, K. Pietrzak, Leakage-resilient cryptography. In *FOCS* (2008), pp. 293–302.

[55] S. Dziembowski, K. Pietrzak, D. Wichs, Non-malleable codes, in *ICS* (2010), pp. 434–452.

[56] S. Dziembowski, K. Pietrzak, D. Wichs, Non-malleable codes. *J. ACM*, **65**(4), 20:1–20:32 (2018).

[57] A. Faonio, J. B. Nielsen, Non-malleable codes with split-state refresh, in *PKC* (2017), pp. 279–309.

[58] A. Faonio, J. B. Nielsen, M. Simkin, D. Venturi, Continuously non-malleable codes with split-state refresh, in *ACNS* (2018), pp. 121–139.

[59] A. Faonio, J. B. Nielsen, M. Simkin, D. Venturi, Continuously non-malleable codes with split-state refresh. *Theor. Comput. Sci.* **759**, 98–132 (2019).

[60] A. Faonio, D. Venturi, Non-malleable secret sharing in the computational setting: adaptive tampering, noisy-leakage resilience, and improved rate, in *CRYPTO* (2019), pp. 448–479.

[61] S. Faust, K. Hostáková, P. Mukherjee, D. Venturi, Non-malleable codes for space-bounded tampering, in *CRYPTO* (2017), pp. 95–126.

[62] S. Faust, P. Mukherjee, J. B. Nielsen, D. Venturi, Continuous non-malleable codes, in *TCC* (2014), pp. 465–488.

[63] S. Faust, P. Mukherjee, J. B. Nielsen, D. Venturi, A tamper and leakage resilient von Neumann architecture, in *PKC* (2015), pp. 579–603.

[64] S. Faust, P. Mukherjee, D. Venturi, D. Wichs, Efficient non-malleable codes and key-derivation for poly-size tampering circuits, in *EUROCRYPT* (2014), pp. 111–128.

[65] S. Faust, P. Mukherjee, D. Venturi, D. Wichs, Efficient non-malleable codes and key derivation for poly-size tampering circuits. *IEEE Trans. Inf. Theory* **62**(12), 7179–7194 (2016).

[66] S. Faust, K. Pietrzak, D. Venturi, Tamper-proof circuits: How to trade leakage for tamper-resilience, in *ICALP* (2011), pp. 391–402.

[67] S. Fehr, P. Karpman, B. Mennink, Short non-malleable codes from related-key secure block ciphers. *IACR Trans. Symmetr. Cryptol.* **2018**(1), 336–352 (2018).

[68] U. Feige, D. Lapidot, A. Shamir, Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract), in *FOCS* (1990), pp. 308–317.

[69] D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, E. Tromer, Circuits resilient to additive attacks with applications to secure computation, in *STOC* (2014), pp. 495–504.

[70] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, T. Rabin, Algorithmic tamper-proof (ATP) security: theoretical foundations for security against hardware tampering, in *TCC* (2004), pp. 258–277.

[71] V. Goyal, A. Kumar, Non-malleable secret sharing, in *STOC* (2018), pp. 685–698.

[72] V. Goyal, A. Sahai, Resettably secure computation, in *EUROCRYPT* (2009), pp. 54–71.

[73] J. Groth, Simulation-sound NIZK proofs for a practical language and constant size group signatures, in *ASIACRYPT* (2006), pp. 444–459.

[74] J. Groth, A. Sahai, Efficient non-interactive proof systems for bilinear groups, in *EUROCRYPT* (2008), pp. 415–432.

[75] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.

[76] Y. Ishai, M. Prabhakaran, A. Sahai, D. Wagner, Private circuits II: Keeping secrets in tamperable circuits, in *EUROCRYPT* (2006), pp. 308–327.

[77] Z. Jafargholi, D. Wichs, Tamper detection and continuous non-malleable codes, in *TCC* (2015), pp. 451–480.

[78] Y. T. Kalai, B. Kanukurthi, A. Sahai, Cryptography with tamperable and leaky memory, in *CRYPTO* (2015), pp. 373–390.

[79] B. Kanukurthi, S. L. B. Obbattu, S. Sekar, Four-state non-malleable codes with explicit constant rate, in *TCC* (2017), pp. 344–375.
[80] B. Kanukurthi, S. L. B. Obbattu, S. Sekar, Four-state non-malleable codes with explicit constant rate. *J. Cryptol.* **33**(3), 1044–1079 (2020).
[81] A. Kiayias, F.-H. Liu, Y. Tselekounis, Practical non-malleable codes from l-more extractable hash functions, in *CCS* (2016), pp. 1317–1328.
[82] A. Kiayias, F.-H. Liu, Y. Tselekounis, Non-malleable codes for partial functions with manipulation detection, in *CRYPTO* (2018), pp. 577–607.
[83] A. Kiayias, Y. Tselekounis, Tamper resilient circuits: The adversary at the gates, in *ASIACRYPT* (2013), pp. 161–180.
[84] F.-H. Liu, A. Lysyanskaya, Tamper and leakage resilience in the split-state model, in *CRYPTO* (2012), pp. 517–532.
[85] R. Ostrovsky, G. Persiano, D. Venturi, I. Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions, in *CRYPTO* (2018), pp. 608–639.
[86] K. Pietrzak, Subspace LWE, in *TCC* (2012), pp. 548–563.
[87] A. D. Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, A. Sahai, Robust non-interactive zero knowledge, in *CRYPTO* (2001), pp. 566–598.
[88] S. Yilek, Resettable public-key encryption: How to encrypt on a virtual machine, in *CT-RSA* (2010), pp. 41–56.