Continuously Non-Malleable Codes against Bounded-Depth Tampering

Gianluca Brian[0000−0002−5352−9763]1 ⋆, Sebastian Faust[0000−0002−8625−4639]2 ⋆⋆, Elena Micheli[2] ⋆⋆, and Daniele Venturi[0000−0003−2379−8564]3 ⋆⋆⋆

[1] Sapienza University of Rome, Rome, Italy.
{brian,venturi}@di.uniroma1.it
[2] Technische Universität Darmstadt, Darmstadt, Germany.
{sebastian.faust,elena.micheli}@tu-darmstadt.de

Abstract. Non-malleable codes (Dziembowski, Pietrzak and Wichs, ICS 2010 & JACM 2018) allow protecting arbitrary cryptographic primitives against related-key attacks (RKAs). Even when using codes that are guaranteed to be non-malleable against a *single* tampering attempt, one obtains RKA security against poly-many tampering attacks at the price of assuming perfect memory erasures. In contrast, *continuously* non-malleable codes (Faust, Mukherjee, Nielsen and Venturi, TCC 2014) do not suffer from this limitation, as the non-malleability guarantee holds against *poly-many* tampering attempts. Unfortunately, there are only a handful of constructions of continuously non-malleable codes, while standard non-malleable codes are known for a large variety of tampering families including, e.g., NC0 and decision-tree tampering, AC0, and recently even bounded polynomial-depth tampering. We change this state of affairs by providing the first constructions of continuously non-malleable codes in the following natural settings:

- Against decision-tree tampering, where, in each tampering attempt, every bit of the tampered codeword can be set arbitrarily after adaptively reading up to d locations within the input codeword. Our scheme is in the plain model, can be instantiated assuming the existence of one-way functions, and tolerates tampering by decision trees of depth $d = O(n^{1/8})$, where n is the length of the codeword. Notably, this class includes NC0.
- Against bounded polynomial-depth tampering, where in each tampering attempt the adversary can select any tampering function that

⋆ Supported by grant SPECTRA from Sapienza University of Rome. This work was partly done while G. Brian was visiting the University of Warsaw, Poland, supported by the Copernicus Award (agreement no. COP/01/2020) from the Foundation for Polish Science and by the Premia na Horyzoncie grant (agreement no. 512681/PnH2/2021) from the Polish Ministry of Education and Science.

⋆⋆ This work has been funded by the German Research Foundation (DFG) CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

⋆⋆⋆ Supported by grant SPECTRA from Sapienza University of Rome.

can be computed by a circuit of bounded polynomial depth (and unbounded polynomial size). Our scheme is in the common reference string model, and can be instantiated assuming the existence of time-lock puzzles and simulation-extractable (succinct) non-interactive zero-knowledge proofs.

1 Introduction

Related-key attacks (RKAs) allow an adversary to break security of a cryptographic primitive by invoking it under one or more keys that satisfy known relations. Such attacks were first introduced as a tool for the cryptanalysis of blockciphers [52,20], but can also be mounted in practice thanks to the ability of attackers to influence secret keys via tampering attacks [22,21,45].

Theoretically, we can model $\mathcal{F}$-RKA security of a given cryptographic primitive as follows: The attacker can choose multiple tampering functions $f_1, f_2, \ldots$ within a family of allowed manipulations $\mathcal{F}$ of the secret key, and later observe the effect of such changes at the output by invoking the primitive on chosen inputs. An elegant solution to the problem of constructing $\mathcal{F}$-RKA-secure cryptoschemes is provided by *non-malleable codes* [40]. Intuitively, an $\mathcal{F}$-non-malleable code allows us to encode a message so that a modified codeword via a function $f \in \mathcal{F}$ either decodes to the same message or to a completely unrelated value. In the application to RKA security, we simply encode the secret key κ and store the corresponding codeword γ in memory. A RKA changes the memory content to $\tilde{\gamma} = f(\gamma)$ for some function $f \in \mathcal{F}$. Hence, at each invocation, we decode the codeword stored in memory and run the corresponding cryptographic primitive using the obtained key. Since the decoded key is either equal to the original or unrelated to it, we obtain $\mathcal{F}$-RKA security.

Unfortunately, there are two important caveats to the above general solution: (i) Since non-malleable codes are only secure against a *single* tampering attempt $f \in \mathcal{F}$, at each invocation we must completely erase the memory and re-encode the key; (ii) In case the modified codeword is invalid, and thus cannot be decoded, we must self-destruct and stop using the underlying primitive. It turns out that limitation (ii) is inherent, in that Gennaro, Lysyanskaya, Malkin, Micali and Rabin [45] established that RKA security is impossible without self-destruct.[3] On the other hand, it would be desirable to remove limitation (i) as perfect erasures of the memory are notoriously hard to implement in practice [26]. Another drawback of limitation (i) is that it makes the cryptoscheme stateful (even if it was stateless to start with) and requires fresh randomness for re-encoding the key.

The stronger notion of *continuously* non-malleable codes [43] allows us to overcome limitation (i): Since such codes guarantee $\mathcal{F}$-non-malleability even

[3] Their attack is simple: The j-th tampering function tries to set the j-th bit of the secret key to 0: If the device returns an invalid output, the next function f_{j+1} additionally sets the j-th bit of the key to 1 and otherwise it sets it to 0.

against poly-many tampering attempts, one immediately obtains $\mathcal{F}$-RKA security without assuming perfect erasures.

1.1 Our Contribution

A nice feature of the above compiler is its generality: In order to achieve $\mathcal{F}$-RKA security all we need to do is to design an $\mathcal{F}$-non-malleable code. In recent years, there has been a tremendous progress in the design of non-malleable codes for several tampering families $\mathcal{F}$ of practical interest, including: bit-wise independent and split-state tampering [40,55,2,6,29,53,54,50,51,4], space-bounded tampering [42], small-locality and small-depth circuits [7,12,10,48], decision-tree and AC0 tampering [13,15], and very recently even bounded polynomial-depth tampering [11,35,14]. In contrast, continuous non-malleability is only known for bit-wise independent tampering [34,32], tampering functions with few fixed points and high entropy [49], constant-state tampering [3], split-state tampering [43,5,56,36] and space-bounded tampering [30], leaving open the following intriguing question:

> *Can we construct* continuously *non-malleable codes against natural non-compartmentalized tampering families, such as decision trees, AC0 or even bounded polynomial-depth circuits?*

We answer the above question in the affirmative:

- In the setting of decision-tree tampering, we construct a code which resists continuous tampering attacks from the family of functions that modify every bit of the tampered codeword arbitrarily after adaptively reading up to d locations from the input codeword. Our scheme is in the plain model, assumes the existence of one-way functions, and tolerates tampering by decision trees of depth $d = O(n^{1/8})$, where n is the length of the codeword. Notably, this class includes NC0.
- In the setting of bounded polynomial-depth tampering, we construct a code that resists continuous tampering attacks, where the adversary can select any tampering function that can be computed by a circuit of bounded polynomial depth (and unbounded polynomial size). Notably, this class includes non-uniform NC. Our scheme is in the common reference string (CRS) model, and assumes the existence of time-lock puzzles and simulation-extractable (succinct) non-interactive zero-knowledge (NIZK) proofs.

We remark that both our constructions rely on computational assumptions. Although we don't know whether they are necessary for decision-tree or bounded-depth continuous tampering, achieving information-theoretic guarantees in the continuous scenario turned out to be challenging for even more well-studied families [32,33,34,49,3,39,30,42,28,5]. We leave this problem open for future work.

1.2 Technical Overview

Due to space constraints, most proofs have been deferred to the full version of this paper [25]. Let us start by reviewing different flavors of *one-time* non-malleability (see Section 3 for formal definitions).

- *Non-malleability w.r.t. message/codeword:* A code is non-malleable *w.r.t. message* (resp. *w.r.t. codeword*) if a tampered codeword either decodes to the original message (resp. is identical to the original codeword) or decodes to a completely unrelated value.
- *Super non-malleability:* A code is *super non-malleable* [43,44] if the tampered codeword itself (when valid) is unrelated to the original message. Note that the distinction between w.r.t. message and w.r.t. codeword also applies here.

Persistent tampering. The above flavors can be naturally extended to the setting of continuous non-malleability. Our first observation is that, in the setting of non-compartmentalized tampering, continuous non-malleability is only achievable in the case of *persistent* tampering, where the j-th tampering function f_j is applied to the output of the previous function f_{j-1}.

The latter can be seen as follows. Consider an adversary that computes offline a valid encoding of two different messages, for simplicity say $\mu_0 = 0^k$ and $\mu_1 = 1^k$. Call γ_0 and γ_1 the corresponding codewords. Next, the attacker prepares a tampering query that hard-wires γ_0, γ_1 and proceeds as follows: It reads the first bit $\gamma[1]$ of the target codeword; if $\gamma[1] = 0$ it overwrites the target codeword with γ_0, while if $\gamma[1] = 1$ it overwrites the target codeword with γ_1. As a result, the adversary learns the first bit of the target codeword. Now, if tampering is non-persistent, the attacker can repeat this procedure to efficiently recover the entire codeword, which clearly violates continuous non-malleability.[4]

In light of the above attack, in what follows, and without loss of generality, when we refer to continuous non-malleability, we implicitly refer to the case of persistent tampering.

Decision-tree tampering. To show our first result, we revisit the recent construction of non-malleable codes against decision-tree tampering by Ball, Guo and Wichs [15]. On a high-level, this construction first encodes the message μ using a *leakage-resilient* non-malleable code in the split-state model, resulting in a codeword $(\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}})$ consisting of a right and a left part. Then, each part γ_i for $i \in \{\mathsf{L}, \mathsf{R}\}$ is encoded independently as follows: we sample a random small set (whose size is that of the underlying codeword) in a much larger array, plant the input in these locations and zero everything else out. Finally, we use a ramp secret sharing with relatively large secrecy threshold to encode a description of the small set (which can be represented by a seed ζ_i). To decode, we can simply

[4] To the best of our knowledge, this observation is new. Previous work in the setting of non-compartmentalized tampering implicitly circumvented the above attack by requiring each tampering function to have high min-entropy and few fixed points, or by assuming that the number of tampering queries is a-priori bounded [49].

extract the seed and output what is in the corresponding locations of the array. This allows us to recover both parts $\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}}$ and thus obtain the initial message.

Ball, Guo and Wichs [15] show how to simulate the decoded message corresponding to one decision-tree tampering query using bounded split-state leakage and one split-state tampering query on the underlying non-malleable code. Although our construction is similar to theirs, proving continuous non-malleability is non-trivial and requires significant new ideas. We discuss some of them below.

First, in the original construction, the positions of the codeword that are not indexed by ζ_i are ignored, since they are not useful for the reconstruction. In our case, however, an attacker could copy the original codeword into the zero bits and overwrite the rest with a valid encoding of an unrelated message, which would allow it to retrieve the original encoding, thus breaking continuous non-malleability. We avoid this by requiring such positions to be 0 for the codeword to be valid. Second, we must ensure that the adversary cannot modify the other parts of the outer codeword without touching the inner codeword: this is because otherwise the adversary could use some tampering queries to save a state inside the codeword, and then use another tampering query to actually tamper with the codeword using more information than he should. We avoid this attack, by relying on computational assumptions. The idea is to sample verification keys $vk_{\mathsf{L}}, vk_{\mathsf{R}}$ for a one-time signature scheme, generate $(\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}})$ as an encoding of the string $\mu||vk_{\mathsf{L}}||vk_{\mathsf{R}}$, and finally append signatures $\sigma_{\mathsf{L}}, \sigma_{\mathsf{R}}$ to the left and right part of the above described final encoding. In Section 3.3, we also show that this trick works generically to compile any super non-malleable code w.r.t. message into a super non-malleable code w.r.t. codeword, so long as the tampering family $\mathcal{F}$ allows us to evaluate the signing algorithm of the signature scheme. Intuitively, our code against decision-tree tampering removes this assumption thanks to the fact that the split-state model allows us to run arbitrary polynomial-time functions (independently on the two parts of the codeword).

In a nutshell, our scheme uses as building blocks a split-state nmc, a signature scheme and a simple procedure transforming states into their sparse versions. The latter takes as input a length-c-string γ, samples a random set $\mathcal{I}$ of c indices in $[n]$ with $n > c$, and outputs the sparse codeword $\gamma^* = (\gamma_1^*, \gamma_2^*)$, where γ_1^* is a RSS encoding of $\mathcal{I}$, and γ_2^* is a length-n-string that has γ in the positions indexed by $\mathcal{I}$, and zeros elsewhere. To extract the original string from the sparse one, it suffices to use the RSS decoding algorithm on the first part, and return the corresponding bits of the second part.
The design of our scheme follows.

Algorithm $\mathsf{Enc}^*(\mu)$**.** Proceed as follows:

1. Sample two pairs of keys $(sk_{\mathsf{L}}, vk_{\mathsf{L}}), (sk_{\mathsf{R}}, vk_{\mathsf{R}})$ for the signature scheme
2. Compute the split-state codeword $(\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}})$ for the message $(\mu||vk_{\mathsf{L}}||vk_{\mathsf{R}})$
3. Compute the sparse strings γ_{L}^* and γ_{R}^* for γ_{L} and γ_{R}.
4. Sign γ_{L}^* and γ_{R}^* with, respectively, sk_{L} and sk_{R}, to get σ_{L} and σ_{R}.
5. The final codeword is $(\sigma_{\mathsf{L}}, \gamma_{\mathsf{L}}^*, \sigma_{\mathsf{R}}, \gamma_{\mathsf{R}}^*)$.

The decoding algorithm extracts γ_{L} and γ_{R} from their sparse versions γ_{L}^* and γ_{R}^* and checks that in the remaining positions there are only zeros, decodes

the split-state codeword $(\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}})$ to get $\mu||vk_{\mathsf{L}}||vk_{\mathsf{R}}$, verifies the signatures and outputs $\perp$ if verification fails, μ otherwise.

Unfortunately, even with the above modifications, it is unclear how to extend the original proof of security to the setting of continuous tampering, even if one assumes the underlying split-state non-malleable code to be continuously non-malleable. The reason is that the reduction needs to leak some bits from the codeword for each tampering query, therefore having a large number of tampering queries would lead to leaking too much information from the split-state codeword. Instead, we exploit the power of *super* non-malleability: Assume the underlying split-state code is super non-malleable w.r.t. codeword.[5] Then, the reduction only needs to know the index q^* of the first tampering query which actually modifies the inner codeword. In case the tampered inner codeword $(\tilde{\gamma}_{\mathsf{L}}, \tilde{\gamma}_{\mathsf{R}})$ is invalid, the experiment stops and we are done. Otherwise, if $(\tilde{\gamma}_{\mathsf{L}}, \tilde{\gamma}_{\mathsf{R}})$ is valid, the reduction obtains it in full. At this point, the reduction is able to simulate the answer to all subsequent tampering queries on its own, as tampering is persistent, which allows us to conclude continuous non-malleability.[6]

It remains to be seen how the reduction can obtain the index q^*. A possible strategy would be to simulate the outcome of all the tampering queries inside the leakage oracle, and then return the index of the first tampering query which actually modifies the codeword; however, each bit of a tampering query can depend on bits of both the left and right part of the inner codeword, while a split-state leakage query is only allowed to see one of these parts. Our strategy is to guess the index q^*, and then check at the end of the experiment if the guess was correct or wrong. Here, we additionally exploit the fact that the underlying split-state super non-malleable code is information-theoretically secure, which essentially allows the reduction to run many instances of the experiment inside the leakage oracle, and check that the adversary does not try to cancel its advantage (due to a wrong simulation). A similar strategy was already used in [56,23,24]. The formal proof appears in Section 4.1.

Bounded polynomial-depth tampering. Our second construction exploits the observation that, for certain tampering families, continuous non-malleability w.r.t. codeword can be reduced to one-time super non-malleability w.r.t. codeword plus logarithmic (in the security parameter) leakage on the codeword. Indeed, this is the case as long as the leakage family allows us to run polynomially-many tampering functions in parallel, and return the index of the first query that actually modifies the codeword (if any). We formalize this observation in Section 3.3 (see Theorem 3). Note that the latter clearly holds true in the setting of bounded polynomial-depth leakage and tampering.[7]

In light of the above, it suffices to construct a one-time super non-malleable code w.r.t. codeword against bounded polynomial-depth tampering. We do so,

[5] We can take, e.g., the non-malleable code of [5] for a concrete instantiation.

[6] As a bonus, we actually prove continuous *super* non-malleability.

[7] The same observation holds true for the setting of AC0 tampering, but not for decision-tree tampering.

by looking at a slightly more general question. Namely, in Section 4.2, we show how to compile a *leakage-resilient* non-malleable code into a super non-malleable code in the CRS model, using simulation-extractable NIZK proofs. The idea is to encode a message μ using the underlying code, and then append to the resulting encoding γ a NIZK proof of knowledge π of the randomness ρ used by the encoder. The decoder outputs $\perp$ if the NIZK proof does not verify correctly.

In the reduction, we can simulate the NIZK proof π and then use a leakage query in order to obtain the tampered proof $\tilde{\pi}$ (so long as the proof $\tilde{\pi}$ is valid), along with the extracted witness $\tilde{\rho}$ corresponding to a tampered codeword $(\tilde{\gamma}, \tilde{\pi}) = f(\gamma, \pi)$ in the experiment defining super non-malleability. Unfortunately, the randomness $\tilde{\rho}$ is too long[8] for being obtained via a leakage query. However, this issue can be resolved by generating ρ using a pseudorandom generator G and letting the corresponding λ-bit seed σ be the witness. This allows the overall leakage to depend only on the security parameter, either assuming simulation-extractable SNARKs [9] (which inherently require non-falsifiable assumptions [47]), or by making the size of the proof depend only on the size of the witness (which can be achieved using fully-homomorphic encryption [46]).

More in detail, our compiler builds on a leakage-resilient one-time non-malleable code $(\mathsf{Enc}, \mathsf{Dec})$, a pseudorandom generator G, and a simulation-extractable proof system. The relation $\mathcal{R}$ for the proof system is satisfied by every couple statement-witness (γ, σ) where $\gamma = \mathsf{Enc}(\mu; \mathsf{G}(\sigma))$ for some message μ. Our encoding (and decoding) algorithm takes as input a CRS ω for the underlying proof system, and is described below.

Algorithm $\mathsf{Enc}^*(\omega, \mu)$**:** Proceed as follows:

1. Generate a random seed σ for the PRG.
2. Use the underlying non-malleable encoding algorithm Enc with randomness $\mathsf{G}(\sigma)$) to compute a codeword γ for μ
3. Generate a proof π for the couple (γ, σ)
4. Output (γ, π).

The decoding algorithm verifies the proof, returns $\perp$ if verification fails, and the message μ underlying γ otherwise.

A subtlety in the above proof sketch is that the leakage family supported by the underlying code must allow simulating the proof π, applying the tampering function f on (γ, π), verifying the tampered proof $\tilde{\pi}$, and extracting the corresponding tampered seed $\tilde{\sigma}$. Similarly, the tampering family supported by the underlying code must allow simulating the proof π and applying the tampering function f on (γ, π). Hence, this compiler does not work for all tampering families. Fortunately, it clearly works for the setting of bounded polynomial-depth tampering.

Our final result is then achieved by adapting a recent construction of Dachman-Soled, Komargodski and Pass [35], who showed how to obtain one-time non-malleability w.r.t. message against bounded polynomial-depth tampering assuming the existence of key-less hash functions and time-lock puzzles (along with

[8] Note that we cannot extract the proof outside the leakage function, as the corresponding statement is the tampered modified codeword $\tilde{\gamma}$ inside the leakage oracle.

other standard assumptions); in the CRS model, we show that their construction can be simplified and proven leakage-resilient one-time non-malleable assuming the existence of time-lock puzzles and simulation-extractable NIZKs. We refer the reader to Section 4.2 and the full version for more details.

Necessity of super non-malleability for the inner split-state code. In our construction against decision-tree tampering, we require the inner split-state encoding to be a super non-malleable code, thus allowing for the simulation of the whole codeword. We argue that this is not an artifact of our proof, but rather a necessity for our construction to achieve security. Indeed, by using a contrived instance of a non-malleable code which is not super non-malleable, and contrived instances of the ramp secret sharing and the signature scheme, we are able to instantiate our scheme so that the adversary becomes able to retrieve the message in full. We consider here a simplified version of our scheme in which we remove the signature scheme, and we point the reader to [25] for the detailed explanation and for how to reintroduce back the signatures.

First of all, we need a split-state non-malleable code which has a good amount of spare bits, initially set to 0, and a secondary mode of operation which uses the spare bits to reconstruct the message instead of the actual relevant bits. Then, we need a malleable RSS encoding which allows to only replace a part of the encoded value leaving everything else intact.

The attack then proceeds as follows: the adversary is now able to tamper with the RSS encoding so that the spare bits of the split-state codeword are in a known location (while keeping the other positions untouched), and he is also able to replace those spare bits with some encoding of either 0 or 1 depending on some bit that the adversary wants to leak, leaving everything else untouched. Finally, the adversary uses multiple queries to leak every bit he left untouched, thus recovering all the bits that are necessary to reconstruct the original message.

Application to RKA security without erasures. It is well known that a continuously $\mathcal{F}$-non-malleable code allows us to obtain a natural notions of $\mathcal{F}$-RKA security for arbitrary cryptographic primitives. This was proven by Faust, Mukherjee, Nielsen, and Venturi [43] for the case of non-persistent tampering. In [25], we show that the same works for the case of persistent tampering.

1.3 Related Work

In recent work, Freitag *et al.* [41] investigate non-malleable time-lock puzzles in the concurrent setting. Their definition generalizes continuous non-persistent non-malleable codes against bounded depth tampering, but requires that the adaptive choice of tampering functions runs in bounded depth too. They provide an impossibility result for the latter, which we extend to all the continuous non-persistent non-malleable codes against global tampering. Given that, they introduce the weaker notion of functional concurrent non-malleable time-lock puzzles, present a construction assuming the existence of (plain) time-lock puzzles in the auxiliary-input random oracle model, and provide interesting applications in coin tossing and electronic auctions.

Dachman-Soled and Kulkarni [36] show that *continuous* super non-malleability in the split-state model inherently requires setup. This impossibility, instead, does not hold for continuous super non-malleability against *persistent* tampering attacks, which can be achieved information-theoretically in the split-state model.

Leakage-Resilient Locally Decodable and Updatable Non-Malleable Codes [38] are a fine-grained tool for protecting RAM machines against leakage and tampering. In literature, there are constructions in the split-state and continuous setting [38], with information theoretic security [27], as well as tight upper and lower bounds [37].

An alternative approach for obtaining generic RKA-security is to rely on non-malleable key derivation [44,57,31]. The difference with non-malleable codes is that in this case one stores a uniformly random string in memory which is used to derive a key for the underlying cryptoscheme at each invocation. Continuously non-malleable key derivation can essentially be achieved only for tampering via polynomials or functions with high entropy.

Another line of research seeks direct constructions of RKA-secure cryptographic primitives, including, e.g., pseudorandom functions [18,16,1] public-key encryption [8,58], identity-based encryption and signatures [19]. RKA security has become a de-facto standard for block-ciphers, and systems are often designed while implicitly relying on the RKA-security of the underlying block-cipher (see, e.g., [17] and references therein).

2 Preliminaries

We start by setting up some basic notation and by recalling the notion of coding schemes. For space reasons, the definition of other standard cryptographic primitives is deferred to the full version [25].

2.1 Notation

We denote by $[n]$ the set $\{1, \ldots, n\}$. For a string $x \in \{0,1\}^*$, we denote its length by $|x|$; if $i \in [|x|]$ and $\mathcal{I} \subseteq [|x|]$, we denote by $x[i]$ the i-th bit of x and by $x[\mathcal{I}]$ the substring of x obtained by only considering the bits indexed by $\mathcal{I}$.

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 ρ) and output y; the value y is a random variable and $\mathsf{A}(x;\rho)$ denotes a run of A on input x and randomness ρ. An algorithm A is *probabilistic polynomial-time* (PPT for short) if A is randomized and for any input $x, \rho \in \{0,1\}^*$, the computation of $\mathsf{A}(x;\rho)$ terminates in a polynomial number of steps (in the size of the input).

Asymptotics. We denote by $\lambda \in \mathbb{N}$ the security parameter. A function p is *polynomial* (in the security parameter), if $p(\lambda) = O(\lambda^c)$ for some constant $c > 0$. A function $\nu : \mathbb{N} \to [0,1]$ is *negligible* (in the security parameter) if it vanishes

faster than the inverse of any polynomial in λ, i.e. $\nu(\lambda) = O(1/p(\lambda))$ for all positive polynomials $p(\lambda)$. Unless stated otherwise, 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 over which $\mathbf{X}$ is defined. The statistical distance between two random variables $\mathbf{X}$ and $\mathbf{Y}$ over $\mathcal{X}$ is defined as $\Delta(\mathbf{X};\mathbf{Y}) := \frac{1}{2}\sum_{x\in\mathcal{X}} |\mathbb{P}[\mathbf{X} = x] - \mathbb{P}[\mathbf{Y} = x]|$. Given two ensembles $\mathbf{X} = \{\mathbf{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathbf{Y} = \{\mathbf{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, we write $\mathbf{X} \equiv \mathbf{Y}$ to denote that $\mathbf{X}_\lambda$ and $\mathbf{Y}_\lambda$ are identically distributed, $\mathbf{X} \overset{\text{s}}{\approx} \mathbf{Y}$ to denote that they are *statistically close*, i.e. $\Delta(\mathbf{X}_\lambda;\mathbf{Y}_\lambda) \leq \nu(\lambda)$ for some negligible function $\nu : \mathbb{N} \to [0,1]$, and $\mathbf{X} \overset{\text{c}}{\approx} \mathbf{Y}$ to denote that they are *computationally indistinguishable*, i.e. for all PPT distinguishers D there is a negligible function $\nu : \mathbb{N} \to [0,1]$ such that:

$$\Delta_{\mathsf{D}}(\mathbf{X}_\lambda;\mathbf{Y}_\lambda) := |\mathbb{P}[\mathsf{D}(X_\lambda) = 1] - \mathbb{P}[\mathsf{D}(Y_\lambda) = 1]| \leq \nu(\lambda).$$

The notion of computational/statistical indistinguishability generalizes immediately to ensembles of interactive experiments $\{\mathbf{G}_{\mathsf{A}}(\lambda)\}_{\lambda\in\mathbb{N}}$ where the adversary A outputs a bit at the end of the interaction.

2.2 Coding Schemes

A (k,n)-code is a pair of algorithms $\Gamma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ specified as follows.

Initialization: The initialization algorithm Init is a randomized algorithm that takes as input the security parameter $\lambda \in \mathbb{N}$ (in unary) and outputs a CRS $\omega \in \{0,1\}^*$.

Encoding: The encoding algorithm Enc is a randomized algorithm that takes as input a CRS $\omega \in \{0,1\}^*$, a message $\mu \in \{0,1\}^k$ and outputs a codeword $\gamma \in \{0,1\}^n$.

Decoding: The decoding algorithm Dec is a deterministic algorithm that takes as input a CRS $\omega \in \{0,1\}^*$, a codeword $\gamma \in \{0,1\}^n$ and outputs either a value in $\{0,1\}^k$ or $\perp$ (denoting an invalid codeword).

We say that Γ satisfies correctness if for all $\omega \in \mathsf{Init}(1^\lambda)$ and all messages $\mu \in \{0,1\}^k$ it holds that $\mathbb{P}[\mathsf{Dec}(\omega, \mathsf{Enc}(\omega,\mu)) = \mu] = 1$, where the probability is over the randomness of the encoding algorithm.

Remark 1 (Coding schemes in the plain model). A code in the plain model can be obtained by restricting Init to output the empty string. In that case, we simply write $\Gamma = (\mathsf{Enc}, \mathsf{Dec})$ and omit the string ω as an input of the encoding and decoding algorithm.

Ramp secret sharing. A ramp secret sharing is a coding scheme satisfying the additional property that any subset of the bits of a codeword with size at most $\lfloor t \cdot n \rfloor$, for some $t \in (0,1)$, reveals nothing about the message.

Definition 1 (Ramp secret sharing). *We say that Γ is a binary (k, n, t)-ramp secret sharing if Γ is a (k, n)-code satisfying the following property: For every $\mu \in \{0,1\}^k$, and for every non-empty subset $\mathcal{I} \subseteq \{0,1\}^n$ of size at most $\lfloor t \cdot n \rfloor$, we have that $\mathsf{Enc}(\mu)|_{\mathcal{I}}$ is identically distributed to the uniform distribution over $\{0,1\}^{|\mathcal{I}|}$.*

As shown by Ball *et al.* [10], any binary linear error correcting code is a binary ramp secret sharing with suitable secrecy. In particular, every binary linear error correcting code correcting at most d errors is a binary ramp secret sharing with secrecy $(d-1)/n$.

Lemma 1 ([10]). *For any message length $k \in \mathbb{N}$ there exist parameters $n \in \mathbb{N}$ and $t \in (0,1)$ such that there is a binary (k, n, t)-ramp secret sharing.*

3 Non-Malleable Codes

In this section, we revisit the definition of non-malleable codes and establish relations among different flavors of non-malleability.

3.1 Non-Malleability

Let Γ be a (k, n)-code, and $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}^n\}$ be a family of functions. Informally, Γ is non-malleable against tampering in $\mathcal{F}$ if decoding a codeword tampered via functions in $\mathcal{F}$ yields either the *original message* or a completely unrelated value. In this paper, we refer to the above flavor of security as *non-malleability w.r.t. message*. Instead, when a tampered codeword (always via functions in $\mathcal{F}$) is either identical to the *original codeword* or decodes to a completely unrelated value, we speak of *non-malleability w.r.t. codeword*.[9]

A stronger (as the name suggests) flavor of non-malleability is the so-called *super non-malleability*, introduced implicitly in [43] (and explicitly in [44]). This property requires that not only the output of the decoding, but *the codeword itself*, is independent of the message, as long as the tampered codeword is valid and either different from the original codeword (yielding super non-malleability w.r.t. codeword) or decoding to something different than the original message (yielding super non-malleability w.r.t. message).

The definition below formalizes continuous (super) non-malleability w.r.t. message/codeword. For readability, it will be useful to introduce the following predicates depending on a code Γ, a CRS ω, two messages μ_0, μ_1, two codewords $\gamma, \tilde{\gamma}$ and a tampering function $f \in \mathcal{F}$:

- $\mathsf{msg}(\omega, \mu_0, \mu_1, \gamma, \tilde{\gamma})$: outputs 1 if and only if $\mathsf{Dec}(\omega, \tilde{\gamma}) \in \{\mu_0, \mu_1\}$;
- $\mathsf{cdw}(\omega, \mu_0, \mu_1, \gamma, \tilde{\gamma})$: outputs 1 if and only if $\tilde{\gamma} = \gamma$;

[9] In the literature, the latter flavor of non-malleability is sometimes known as *strong non-malleability* whereas the former flavor is also known as *weak non-malleability*. However, we find this terminology rather confusing due to the fact that a code can be at the same time weakly non-malleable and super non-malleable (as defined below).

$\mathbf{CNM}^{\mathsf{same},\mathsf{output}}_{\Gamma,\mathsf{A},\mathcal{F},\mathcal{G}}(\lambda, b)$

1: $\omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$
2: $(\mu_0, \mu_1, \alpha_0) \leftarrow\!\!\$\ \mathsf{A}_0(\omega)$
3: $\gamma \leftarrow\!\!\$\ \mathsf{Enc}(\omega, \mu_b)$
4: **return** $\mathsf{A}_1^{\mathcal{O}^{\mathsf{tamper}}(\gamma,\cdot),\mathcal{O}^{\mathsf{leak}}_\ell(\gamma,\cdot)}(\alpha_0)$

Fig. 1. Experiment defining leakage-resilient (super) non-malleable codes, with an adversary A consisting of subroutines $(\mathsf{A}_0, \mathsf{A}_1)$.

- $\mathsf{standard}(\tilde{\mu}, \tilde{\gamma})$: outputs $\tilde{\mu}$;
- $\mathsf{super}(\tilde{\mu}, \tilde{\gamma})$: outputs $\tilde{\mu}$ if $\tilde{\mu} \in \{\diamond, \bot\}$, and $\tilde{\gamma}$ otherwise.

The above algorithms are called inside the tampering oracle $\mathcal{O}^{\mathsf{tamper}}(\gamma, \cdot)$, which initializes[10] $\hat{\gamma} = \gamma$ and self-destruct parameter $\delta = 0$, and behaves as follows:

1. if $\delta = 1$, output $\bot$;
2. compute $\tilde{\gamma} = f(\hat{\gamma})$ and $\tilde{\mu} = \mathsf{Dec}(\omega, \tilde{\gamma})$;
3. if $\mathsf{same}(\omega, \mu_0, \mu_1, \hat{\gamma}, \tilde{\gamma}) = 1$, set $\tilde{\mu} = \diamond$;
4. if $\tilde{\mu} = \bot$, set $\delta = 1$;
5. set $\hat{\gamma} = \tilde{\gamma}$ and return $\mathsf{output}(\tilde{\mu}, \tilde{\gamma})$;

We model leakage resilience by an oracle $\mathcal{O}^{\mathsf{leak}}_\ell(\gamma, \cdot)$ that accepts as input functions $g \in \mathcal{G}$ and returns $g(\gamma)$ (or $\bot$ if $\delta = 1$), for a total of at most ℓ bits.

Definition 2 (Continuously non-malleable codes). *Let Γ be a (k, n)-code, and $\mathcal{F} \subseteq \{f : \{0,1\}^n \to \{0,1\}^n\}$ and $\mathcal{G} \subseteq \{g : \{0,1\}^n \to \{0,1\}^*\}$ be family of functions. For flags $\mathsf{same} \in \{\mathsf{msg}, \mathsf{cdw}\}$ and $\mathsf{output} \in \{\mathsf{standard}, \mathsf{super}\}$ we say that Γ is a $(\mathcal{G}, \ell)$-*leakage-resilient persistent continuously $\mathcal{F}$-non-malleable *code if the following holds for the experiments defined in Fig. 1:*

$$\left\{\mathbf{CNM}^{\mathsf{same},\mathsf{output}}_{\Gamma,\mathsf{A},\mathcal{F},\mathcal{G}}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \left\{\mathbf{CNM}^{\mathsf{same},\mathsf{output}}_{\Gamma,\mathsf{A},\mathcal{F},\mathcal{G}}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}}. \tag{1}$$

In particular:

- *When Eq. (1) holds for* $\mathsf{same} = \mathsf{msg}$ *(resp.* $\mathsf{same} = \mathsf{cdw}$*) we speak of persistent continuous non-malleability* w.r.t. message *(resp.* w.r.t. codeword*);*
- *When Eq. (1) holds for* $\mathsf{output} = \mathsf{super}$*, we refer to persistent continuous super non-malleability w.r.t. message/codeword. When* $\mathsf{output} = \mathsf{standard}$*, we speak of persistent continuous non-malleability w.r.t. message/codeword.*
- *When Eq. (1) holds in the information-theoretic setting with statistical distance at most $\epsilon \in [0, 1]$, we say that Γ is leakage-resilient persistent continuously super non-malleable with* statistical security ϵ.

[10] The oracle additionally takes as input all the values that are required to evaluate the above predicates. We omit them for clarity.

One-time non-malleability. When we restrict the adversary by only allowing one tampering query, we obtain the weaker notion of *one-time non-malleability*. To formalize the latter, it suffices to replace Item 4 with an instruction which sets $\delta = 1$ regardless of the value of $\tilde{\mu}$. We denote the resulting experiment as $\mathbf{1NM}^{\mathsf{same},\mathsf{output}}_{\Gamma,\mathsf{A},\mathcal{F},\mathcal{G}}(\lambda, b)$, and in Definition 2 it only suffices to replace Eq. (1) with

$$\left\{\mathbf{1NM}^{\mathsf{same},\mathsf{output}}_{\Gamma,\mathsf{A},\mathcal{F},\mathcal{G}}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \left\{\mathbf{1NM}^{\mathsf{same},\mathsf{output}}_{\Gamma,\mathsf{A},\mathcal{F},\mathcal{G}}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}} \tag{2}$$

to obtain the new notion.

3.2 Families of Tampering Functions

Below, we define a few tampering families of interest for this paper.

Split-state tampering. Let Γ be a $(k, n_{\mathsf{L}} + n_{\mathsf{R}})$-code. In the split-state model, we think of a codeword $\gamma \in \{0,1\}^n$ as consisting of two parts $\gamma_{\mathsf{L}} \in \{0,1\}^{n_{\mathsf{L}}}, \gamma_{\mathsf{R}} \in \{0,1\}^{n_{\mathsf{R}}}$. Hence, we consider the following families of tampering and leakage functions:

$$\mathcal{F}_{\mathsf{split}}(n_{\mathsf{L}}, n_{\mathsf{R}}) := \{f = (f_{\mathsf{L}}, f_{\mathsf{R}}) : f_{\mathsf{L}} : \{0,1\}^{n_{\mathsf{L}}} \to \{0,1\}^{n_{\mathsf{L}}}, f_{\mathsf{R}}\{0,1\}^{n_{\mathsf{R}}} \to \{0,1\}^{n_{\mathsf{R}}}\}$$
$$\mathcal{G}_{\mathsf{split}}(n_{\mathsf{L}}, n_{\mathsf{R}}) := \left\{g = (g_{\mathsf{L}}, g_{\mathsf{R}}) : g_{\mathsf{L}} : \{0,1\}^{n_{\mathsf{L}}} \to \{0,1\}^{\ell}, g_{\mathsf{R}} : \{0,1\}^{n_{\mathsf{R}}} \to \{0,1\}^{\ell}\right\}.$$

In this case, we simply say that Γ is ℓ-leakage-resilient super non-malleable w.r.t. message/codeword in the split-state model.

Decision trees. Let Γ be a (k, n)-code and $d \in \mathbb{N}$. Consider a binary tree of depth d whose internal nodes are labelled by numbers in $[n]$ and whose leaves contain values in $\{0,1\}$. Given a binary tree as above, we define a decision tree of depth d for $\{0,1\}^n$ as a Boolean function that takes as input a string $\gamma \in \{0,1\}^n$ and is described as follows:

- it starts from the root;
- it reads the label $i \in [n]$ of the node, and observes the i-th bit of the codeword $\gamma_i \in \{0,1\}$: if $\gamma_i = 0$, it descends to the left subtree, while if $\gamma_i = 1$, it moves to the right subtree;
- it outputs the value of the leaf at the end of the path.

We denote with $\mathcal{DT}^d(n)$ the set of all decision trees for $\{0,1\}^n$ with depth at most d. Hence, we consider the tampering family:

$$\mathcal{F}^d_{\mathsf{dtree}}(n) := \left\{f := (f_1, \ldots, f_n) : \forall i \in [n], f_i \in \mathcal{DT}^d(n)\right\},$$

and the leakage family $\mathcal{G}^d_{\mathsf{dtree}}(n) := \mathcal{DT}^d(n)$. In this case, we simply say that Γ is ℓ-leakage-resilient super non-malleable w.r.t. message/codeword against depth-d decision-tree tampering and leakage.

Bounded polynomial-time tampering. Let $S(\lambda), T(\lambda)$ be polynomials in the security parameter. A non-uniform algorithm A is described by a family of algorithms $\{\mathsf{A}_\lambda\}_{\lambda\in\mathbb{N}}$ (i.e., a different algorithm for each choice of the security parameter). Each A_λ corresponds to an algorithm whose input size is $n(\lambda)$, where $n : \mathbb{N} \to \mathbb{N}$. We say that a non-uniform algorithm A is S-size T-time if, for every input of size $n(\lambda)$ for some $\lambda \in \mathbb{N}$, the total work of the algorithm is at most $S(\lambda)$ and its parallel running time is upper bounded by $T(\lambda)$. We denote the family of non-uniform S-size T-time algorithms as $\mathcal{F}^{S,T}_{\mathsf{non\text{-}uni}}(n)$, and let

$$\mathcal{F}^{T}_{\mathsf{non\text{-}uni}}(n) := \bigcup_{S\in poly(\lambda)} \mathcal{F}^{S,T}_{\mathsf{non\text{-}uni}}(n).$$

3.3 Simple Facts

It is not hard to show that (super) non-malleability w.r.t. message is strictly weaker than (super) non-malleability w.r.t. codeword (e.g., consider a contrived code where we append a dummy bit to each codeword which is ignored by the decoding algorithm). It is also easy to see that non-malleability w.r.t. message/codeword is strictly weaker than super non-malleability w.r.t. message/codeword (e.g., consider a contrived code where we encode the message twice and where the decoding algorithm ignores the second copy of the codeword).

Below, we formalize three simple observations. (i) Assuming one-way functions, one can transform any (super) non-malleable code w.r.t. message into one w.r.t. codeword. (ii) For any (super) non-malleable code w.r.t. message/codeword there is a natural tradeoff between security and leakage resilience. (iii) In some cases, one-time super non-malleability w.r.t. codeword, along with leakage resilience, are sufficient to imply continuous non-malleability (in the setting of persistent tampering). All the above statements hold as long as the tampering family $\mathcal{F}$ and the leakage family $\mathcal{G}$ supported by the code are large enough (as detailed below). For simplicity, we stick to the plain model (but similar statements hold true in the CRS model).

Adding super non-malleability w.r.t. codeword. Let $\Gamma = (\mathsf{Enc}, \mathsf{Dec})$ be a code and $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{SigVer})$ be a signature scheme. Consider the following derived code $\Gamma^* = (\mathsf{Enc}^*, \mathsf{Dec}^*)$.

Encoding: The encoding algorithm Enc^* takes as input a message $\mu \in \{0,1\}^k$, samples $(sk, vk) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, computes $\gamma \leftarrow\!\!\$\ \mathsf{Enc}(vk||\mu)$ and $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, \gamma)$, and outputs $\gamma^* = (\gamma, \sigma)$.

Decoding: The decoding algorithm Dec^* takes as input a codeword $\gamma^* = (\gamma, \sigma)$, and computes $\mu^* = vk||\mu = \mathsf{Dec}(\gamma)$. If either $\mu^* = \perp$ or $\mathsf{SigVer}(vk, \gamma, \sigma) = 0$, output $\perp$. Else output μ.

Let $\mathcal{F} \subseteq \{f : \{0,1\}^{n+s} \to \{0,1\}^{n+s}\}$, $\mathcal{G} \subseteq \{g : \{0,1\}^{n+s} \to \{0,1\}^*$ be families of functions. In the theorem below, for any function $f \in \mathcal{F}$, and any $\gamma \in \{0,1\}^n$ and $\sigma \in \{0,1\}^s$, we write $f(\gamma, \sigma)_1$ (resp. $f(\gamma, \sigma)_2$) for the function that outputs the first n bits (resp. the last s bits) of $f(\gamma, \sigma)$.

Theorem 1. *Assume that Σ is a strongly one-time unforgeable signature scheme with $\mathcal{M} = \{0,1\}^n$, $\mathcal{S} = \{0,1\}^s$ and $\mathcal{V} = \{0,1\}^v$, and that Γ is a $(\mathcal{G}(n), \ell + s)$-leakage-resilient persistent continuously $\mathcal{F}(n)$-super-non-malleable $(k+v, n)$-code w.r.t. message. Then, the above defined $(k, n+s)$-code Γ^* is $(\mathcal{G}(n+s), \ell)$-leakage-resilient persistent continuously $\mathcal{F}(n+s)$-super-non-malleable w.r.t. codeword, so long as for all $g \in \mathcal{G}(n+s)$, all $f \in \mathcal{F}(n+s)$, and all $(sk, vk) \in \mathsf{Gen}(1^\lambda)$ and $\rho \in \{0,1\}^*$, it holds that*

$$\mathcal{G}(n) \supseteq \{g(\cdot, \mathsf{Sign}(sk, \cdot; \rho)), f(\cdot, \mathsf{Sign}(sk, \cdot; \rho))_2, \} \tag{3}$$

$$\mathcal{F}(n) \supseteq \{f(\cdot, \mathsf{Sign}(sk, \cdot; \rho))_1, \mathsf{SigVer}(vk, f(\cdot, \mathsf{Sign}(sk, \cdot; \rho)))\}. \tag{4}$$

Intuitively, if the signature scheme is strongly unforgeable, then a tampering attacker cannot maul γ^* while preserving vk. On the other hand, the security of the underlying non-malleable code guarantees that every change to vk makes the mauled message independent.

Remark 2 (On compartmentalized tampering). Note that Theorem 1 does not immediately apply in the split-state setting where $\mathcal{F} = \mathcal{F}_{\mathsf{split}}(n, n)$ and $\mathcal{G} = \mathcal{G}_{\mathsf{split}}(n, n)$, because the conditions of Eq. (3) and Eq. (4) are not satisfied in general. However, we can slightly modify the code Γ^* by signing the left part γ_{L} and the right part γ_{R} of a codeword $\gamma = (\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}}) \in \{0,1\}^{2n}$ separately, yielding signatures σ_{L} and σ_{R}, and letting $\gamma^* = ((\gamma_{\mathsf{L}}, \sigma_{\mathsf{L}}), (\gamma_{\mathsf{R}}, \sigma_{\mathsf{R}}))$ to obtain the above result for the families $\mathcal{G}_{\mathsf{split}}(n+s, n+s)$ and $\mathcal{F}_{\mathsf{split}}(n+s, n+s)$.

Adding leakage resilience. Next, we show how to use complexity leveraging in order to add leakage resilience to any strong-enough non-malleable code. The latter was already shown by Brian *et al.* [23] for the case of split-state tampering, who proved how leakage can be simulated by guessing and later verifying the accuracy of the guess. In particular, the security loss is exponential in the number of bits leaked, as the reduction correctly simulates the leakage only when the guess is exact. We observe that this can be generalized to the case where tampering via $\mathcal{F}$ can reveal whether the answer to a leakage query in $\mathcal{G}$ is correct. We call this property $\mathcal{G}$-friendliness.

Definition 3 (Leakage-friendly tampering). *Let $\mathcal{F} \subseteq \{f : \{0,1\}^n \to \{0,1\}^n\}$ and $\mathcal{G} \subseteq \{g : \{0,1\}^n \to \{0,1\}^\ell\}$ be families of functions. We say that $\mathcal{F}$ is $\mathcal{G}$-leakage friendly if for all $g \in \mathcal{G}$, all $f \in \mathcal{F}$, and all strings $y \in \{0,1\}^\ell$ it holds that $\hat{f} \in \mathcal{F}$ where $\hat{f}$ is the function that upon input $\gamma \in \{0,1\}^n$ outputs $f(\gamma)$ if and only if $y = g(\gamma)$ (and outputs $\perp$ otherwise).*

Theorem 2. *Let $\mathcal{F} \subseteq \{f : \{0,1\}^n \to \{0,1\}^n\}$ and $\mathcal{G} \subseteq \{g : \{0,1\}^n \to \{0,1\}^\ell\}$ be families of functions such that $\mathcal{F}$ is $\mathcal{G}$-leakage friendly. Assume that Γ is persistent continuously $\mathcal{F}$-(super)-non-malleable w.r.t. message/codeword, with statistical security $\epsilon \in (0,1)$. Then, Γ is $\mathcal{G}$-leakage-resilient persistent continuously $\mathcal{F}$-(super)-non-malleable w.r.t. message/codeword, with statistical security $2^\ell \cdot \epsilon$, assuming that all the leakage is done before the first tampering query.*

Remark 3 (On computational security). Theorem 2 also holds in the computational setting, so long as $\ell = O(\log \lambda)$. In fact, it even holds for $\ell = \omega(\log \lambda)$ assuming Γ has sub-exponential security.

Remark 4 (On adaptive leakage). We can extend Theorem 2 to leakage families $\mathcal{G} \subseteq \{g : \{0,1\}^n \to \{0,1\}^*\}$, so long as the notion of leakage friendliness holds for up to q leakage functions. In this case, leakage resilience holds against adversaries making at most[11] q leakage queries.

Achieving persistent continuous super non-malleability. Finally, we establish a connection between one-time super non-malleability and persistent continuous super non-malleability. Intuitively, one can simulate continuous tampering by leaking the index of the first tampering query that modifies the codeword and then obtaining the corresponding mauled codeword via a single tampering query. This connection was first outlined in [49], and later proven formally in [5] in the split-state setting. We generalize this observation to general tampering families.

Theorem 3. *Let Γ be a $(\mathcal{G}(n), \ell+1)$-leakage-resilient $\mathcal{F}(n)$-super-non-malleable (k,n)-code w.r.t. codeword. Assume that for every $q(\lambda) \in poly(\lambda)$, and every tuple of tampering functions $f^{(1)}, \ldots, f^{(q)} \in \mathcal{F}(n)$, the leakage family $\mathcal{G}(n)$ contains the function $\hat{g}(\gamma)$ that computes $(f^{(1)}(\gamma), \ldots, f^{(q)}(\gamma)))$ and returns 1 if and only if $f^{(1)}(\gamma) = \cdots f^{(q-1)}(\gamma) = \gamma$, but $f^{(q)}(\gamma) \neq \gamma$. Then, Γ is also a $(\mathcal{G}(n), \ell)$-leakage-resilient persistent continuously $\mathcal{F}(n)$-super-non-malleable (k,n)-code w.r.t. codeword.*

Remark 5 (On super non-malleability w.r.t. codeword). Theorem 3 holds even starting with a super non-malleable code w.r.t. message, so long as the family $\mathcal{F}$ is closed under composition of poly-many functions (i.e., for all $q(\lambda) \in poly(\lambda)$ and all $f^{(1)}, \ldots, f^{(q)} \in \mathcal{F}$ the function $f^{(q)} \circ f^{(q-1)} \circ \cdots \circ f^{(1)}$ is contained in $\mathcal{F}$).

4 Our Constructions

4.1 Decision-Tree Tampering

Our construction is inspired by [15], with a few modifications that are necessary in order to prove persistent continuous super non-malleability w.r.t. codeword. To facilitate the description, let us introduce the following auxiliary function. For $n, c \in \mathbb{N}$, let $\phi : \{0,1\}^{c \log n} \to \mathcal{P}([n])$ be the function that, upon input a string $\zeta \in \{0,1\}^{c \log n}$ corresponding to c binary representations of distinct numbers in $[n]$, outputs the corresponding set of indices $\mathcal{I} \subseteq [n]$.

Our scheme is made of two layers, where the outer layer takes as input a split-state encoding of the message. Let $n, c, t \in \mathbb{N}$ be such that $t \geq c \log n$. Let $(\mathsf{Enc}_{\mathsf{RSS}}, \mathsf{Dec}_{\mathsf{RSS}})$ be a binary ramp secret sharing with messages in $\{0,1\}^t$. Consider the coding scheme $(\mathsf{Enc}^*_{n,c,t}, \mathsf{Dec}^*_{n,c,t})$ described below.

[11] Note that, e.g., $\mathcal{F}_{\mathsf{split}}$ is $\mathcal{G}_{\mathsf{split}}$-leakage friendly for any $q \in poly(\lambda)$.

Algorithm $\mathsf{Enc}^*_{n,c,t}(\gamma)$**.** Upon input $\gamma \in \{0,1\}^c$:

1. Sample a random string ζ over the set of all strings of length $c \log n$ corresponding to c binary representations of distinct numbers in $[n]$.
2. Let $\mathcal{I} = \phi(\zeta)$ and let $\overline{\mathcal{I}} = [n] \setminus \mathcal{I}$.
3. Let γ^* be such that $\gamma^*[\mathcal{I}] = \gamma$ and $\gamma^*[\overline{\mathcal{I}}] = 0^{n-c}$.
4. Output $(\mathsf{Enc}_{\mathsf{RSS}}(\zeta), \gamma^*)$.

Algorithm $\mathsf{Dec}^*_{n,c,t}(\gamma_{\mathsf{RSS}}, \gamma^*)$**.** Proceed as follows:

1. Decode $\zeta = \mathsf{Dec}_{\mathsf{RSS}}(\gamma_{\mathsf{RSS}})$ and let $\mathcal{I} = \phi(\zeta)$.
2. If there exists $i \in [n] \setminus \mathcal{I}$ such that $\gamma^*[i] = 1$, return $\perp$.
3. Let $\gamma := \gamma^*[\mathcal{I}]$.
4. Output γ (or (γ, ζ) when ζ is explicitly needed).

We observe that the only difference between our version of the $\mathsf{Dec}^*_{n,c,t}$ algorithm and the one in [15] is the check we perform in Item 2. This modification is required in order to obtain super non-malleability because, otherwise, an attacker could copy the original codeword into the 0 bits and then overwrite it with a constant valid codeword, and this would allow for the retrieval of the original encoding, and thus of the underlying message, in full.

We are now ready to define the final encoding scheme $\Gamma^* = (\mathsf{Enc}^*, \mathsf{Dec}^*)$ with security against decision-tree leakage and tampering. Let $m, n_{\mathsf{L}}, n_{\mathsf{R}}, c, t_{\mathsf{L}}, t_{\mathsf{R}}, s, v \in \mathbb{N}$ be parameters. Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{SigVer})$ be a signature scheme with message space $\mathcal{M} = \{0,1\}^*$, signature space $\mathcal{S} = \{0,1\}^s$ and verification keys in $\mathcal{V} = \{0,1\}^v$. Let $\Gamma = (\mathsf{NMEnc}, \mathsf{NMDec})$ be a $(m+2v, 2c)$-code. Let $\mathsf{Enc}_{\mathsf{L}} := \mathsf{Enc}^*_{n_{\mathsf{L}},c,t_{\mathsf{L}}}$, $\mathsf{Enc}_{\mathsf{R}} := \mathsf{Enc}^*_{n_{\mathsf{R}},c,t_{\mathsf{R}}}$, $\mathsf{Dec}_{\mathsf{L}} := \mathsf{Dec}^*_{n_{\mathsf{L}},c,t_{\mathsf{L}}}$, $\mathsf{Dec}_{\mathsf{R}} := \mathsf{Dec}^*_{n_{\mathsf{R}},c,t_{\mathsf{R}}}$.

Algorithm $\mathsf{Enc}^*(\mu)$**.** Upon input $\mu \in \{0,1\}^m$:

1. Sample $(sk_{\mathsf{L}}, vk_{\mathsf{L}}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ and $(sk_{\mathsf{R}}, vk_{\mathsf{R}}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$.
2. Run $(\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}}) \leftarrow\!\!\$\ \mathsf{NMEnc}(\mu || vk_{\mathsf{L}} || vk_{\mathsf{R}})$.
3. Run $\gamma^*_{\mathsf{L}} \leftarrow\!\!\$\ \mathsf{Enc}_{\mathsf{L}}(\gamma_{\mathsf{L}})$ and $\gamma^*_{\mathsf{R}} \leftarrow\!\!\$\ \mathsf{Enc}_{\mathsf{R}}(\gamma_{\mathsf{R}})$.
4. Compute $\sigma_{\mathsf{L}} \leftarrow\!\!\$\ \mathsf{Sign}(sk_{\mathsf{L}}, \gamma^*_{\mathsf{L}})$ and $\sigma_{\mathsf{R}} \leftarrow\!\!\$\ \mathsf{Sign}(sk_{\mathsf{R}}, \gamma^*_{\mathsf{R}})$.
5. Output $\gamma^* := (\sigma_{\mathsf{L}}, \gamma^*_{\mathsf{L}}, \sigma_{\mathsf{R}}, \gamma^*_{\mathsf{R}})$.

Algorithm $\mathsf{Dec}^*(\gamma^*)$**.** Proceed as follows:

1. Parse $\gamma^* = (\sigma_{\mathsf{L}}, \gamma^*_{\mathsf{L}}, \sigma_{\mathsf{R}}, \gamma^*_{\mathsf{R}})$
2. Run $\gamma_{\mathsf{L}} = \mathsf{Dec}_{\mathsf{L}}(\gamma^*_{\mathsf{L}})$ and $\gamma_{\mathsf{R}} = \mathsf{Dec}_{\mathsf{R}}(\gamma^*_{\mathsf{R}})$.
3. Run $\mu || vk_{\mathsf{L}} || vk_{\mathsf{R}} = \mathsf{NMDec}(\gamma_{\mathsf{L}}, \gamma_{\mathsf{R}})$.
4. Check that $\mathsf{SigVer}(vk_{\mathsf{L}}, \gamma^*_{\mathsf{L}}, \sigma_{\mathsf{L}}) = 1$ and $\mathsf{SigVer}(vk_{\mathsf{R}}, \gamma^*_{\mathsf{R}}, \sigma_{\mathsf{R}}) = 1$
5. Output μ, or $\perp$ if the above check fails.

We establish the following theorem.

Theorem 4. *Let Σ, Γ, and Γ^* be as above. Assume that Σ is a strongly one-time unforgeable signature scheme with signature length $s = \beta c$ for some $\beta \in (0,1)$, that Γ is an αc-leakage-resilient super non-malleable $(k + 2v, 2c)$-code w.r.t. codeword in the split-state model for some constant $\alpha < 1$, and that the privacy thresholds $t_{\mathsf{L}}, t_{\mathsf{R}}$ of the ramp secret sharing satisfy $t_{\mathsf{L}} \geq d$ and $t_{\mathsf{R}} \geq (4t_{\mathsf{L}} + c)d$. Then, the code Γ^* described above is a persistent continuously super non-malleable (m,n)-code against depth-d decision-tree tampering for $d = O(c^{1/4})$ and $n = O(c^2)$.*

Remark 6 (On simulating persistent continuous tampering). The definition of persistent continuous tampering states that the adversary A has unlimited access to the tampering oracle, unless A fails to produce a valid codeword, thus receiving $\perp$ in all subsequent tampering queries. However, we observe that this is equivalent to asking that A cannot send any more queries to the tampering oracle after receiving, as a result to a tampering query, a codeword $\tilde{\gamma} \in \{0,1\}^n \cup \{\perp\}$ which is different from $\diamond$. This is because, once obtained a tampered codeword which is either $\perp$ or a valid codeword, the adversary can simulate all the other queries on its own. Notice that this only holds in the case of super non-malleability, since A needs the tampered codeword in order to simulate the subsequent queries.

The remainder of the section is dedicated to the proof of Theorem 4.

Establishing useful notation and procedures. For ease of notation, let $\mathcal{F}^d_{\mathsf{dtree}}(n) := \mathcal{F}^d_{\mathsf{dtree}}, \mathcal{F}_{\mathsf{split}}(c,c) := \mathcal{F}_{\mathsf{split}}, \mathcal{G}_{\mathsf{split}}(c,c) := \mathcal{G}_{\mathsf{split}}$, $\mathbf{G}^*_{\mathsf{A}}(\lambda,b) := \mathbf{CNM}^{\mathsf{cdw},\mathsf{super}}_{\Gamma^*,\mathsf{A},\mathcal{F}^d_{\mathsf{dtree}},\emptyset}(\lambda,b)$ and $\mathbf{G}_{\mathsf{A}}(\lambda,b) := \mathbf{1NM}^{\mathsf{cdw},\mathsf{super}}_{\Gamma,\mathsf{A},\mathcal{F}_{\mathsf{split}}}(\lambda,b)$. For all adversaries A against $\mathbf{G}^*_{\mathsf{A}}(\lambda,b)$, we can assume, without loss of generality, that A performs at most $p = poly(\lambda)$ tampering queries. Finally, let $\mathsf{Enc}^{\mathsf{L}}_{\mathsf{RSS}}$ be the instantiantion of $\mathsf{Enc}_{\mathsf{RSS}}$ used in $\mathsf{Enc}_{\mathsf{L}}$ and let $\mathsf{Enc}^{\mathsf{R}}_{\mathsf{RSS}}$ be the instantiation of $\mathsf{Enc}_{\mathsf{RSS}}$ used in $\mathsf{Enc}_{\mathsf{R}}$.

The proof is by reduction. In order to simplify the exposition, we define a template procedure $\mathsf{ObtainBits}$ which we will invoke several times in the actual proof. Informally, $\mathsf{ObtainBits}$ tries to evaluate the decision trees corresponding to the positions in $\mathcal{I}$ of the codeword tampered via f using the information $\mathcal{L}$ already known to the reduction itself; if some information is missing, it leaks it from the codeword. Since the reduction uses $\mathsf{ObtainBits}$ both inside and outside the leakage oracle, different sub-procedures are needed to leak these bits, depending on when $\mathsf{ObtainBits}$ is invoked. The formal definition follows.

Procedure $\mathsf{ObtainBits}_{\mathsf{Bit},\mathsf{Return}}(f,\mathcal{I},\mathcal{L})$**:**

- **Instantiation:** A sub-procedure Bit taking as input an index $i \in [n]$ and returning a bit $b_i \in \{0,1\}$, and a sub-procedure Return taking as input the set $\mathcal{L}$ and a string $x \in \{0,1\}^*$ and returning some value.
- **Input:** A collection of decision trees f, a set of indices $\mathcal{I} \subseteq [n]$, a set $\mathcal{L} = \{(i,b) : i \in [n], b \in \{0,1\}\}$ such that if $(i_1,b_1),(i_2,b_2) \in \mathcal{L}$ and $i_1 = i_2$ then $b_1 = b_2$. Informally, $\mathcal{I}$ is the set of decision trees the procedure should compute and $\mathcal{L}$ is the prior knowledge of the algorithm invoking the procedure.

1. Let x be an initially empty string.
2. For all $i \in \mathcal{I}$, let initially $\mathsf{T} = f[i]$ and compute T as follows.
   (a) Let r be the label on the root of T.
   (b) If r is a leaf (i.e. $\mathsf{T} = \mathsf{r}$), then append r to x and step to the next index $i \in \mathcal{I}$ (or break the loop if all decision trees have been computed).
   (c) If there exists $(\mathsf{r},b) \in \mathcal{L}$ for a $b \in \{0,1\}$, let $b^* = b$; otherwise, run $b^* \leftarrow \mathsf{Bit}(\mathsf{r})$ and replace $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{r},b^*)\}$.

(d) Replace T with its left subtree if $b^* = 0$ and with its right subtree if $b^* = 1$.
(e) Go to Item 2a.

3. Run $y \leftarrow \mathsf{Return}(\mathcal{L}, x)$ and output y.

As for the sub-procedures, we define the following possibilities for the template argument Bit.

- **Procedure** $\mathsf{Leak}(i)$**:** Use the leakage oracle to leak the bit i from the split-state codeword.
- **Procedure** $\mathsf{Await}(i)$**:** Abort the procedure $\mathsf{ObtainBits}$, returning (await, i).

Finally, we define the following possibilities for the template argument Return.

- **Procedure** $\mathsf{Ready}(\mathcal{L}, x)$**:** Return (ready).
- **Procedure** $\mathsf{Check}(\mathcal{L}, x)$**:** Return 1 if x is the all 0 string and return 0 if x contains at least one 1.
- **Procedure** $\mathsf{Update}(\mathcal{L}, x)$**:** Return the updated set $\mathcal{L}$.

Notice that, when using the sub-procedure Leak, algorithm $\mathsf{ObtainBits}$ presents an undefined behaviour whenever there exists $(i, b) \notin \mathcal{L}$ such that i does not refer to any position on the split-state codeword; however, our reduction only invokes $\mathsf{ObtainBits}$ with sets $\mathcal{L}$ such that the only missing indices are indices which belong to the split-state codeword.

Ruling out signature forgeries. For any adversary A against $\mathbf{G}^*_{\mathsf{A}}(\lambda, b)$, $q \in [p]$, $j \in \{\mathsf{L}, \mathsf{R}\}$, let $\mathbf{W}_j^{(q)}$ be the event in which the first $q-1$ tampering queries from A do not modify the codeword and the q-th tampering query $f^{(q)}$ is such that $f^{(q)}(\gamma^*) = (\tilde{\sigma}_{\mathsf{L}}, \tilde{\gamma}^*_{\mathsf{L}}, \tilde{\sigma}_{\mathsf{R}}, \tilde{\gamma}^*_{\mathsf{R}})$ satisfies (i) $\mathsf{NMDec}(\mathsf{Dec}_{\mathsf{L}}(\tilde{\gamma}^*_{\mathsf{L}}), \mathsf{Dec}_{\mathsf{R}}(\tilde{\gamma}^*_{\mathsf{R}})) = \tilde{\mu}||\tilde{vk}_{\mathsf{L}}||\tilde{vk}_{\mathsf{R}}$ with $\tilde{vk}_j = vk_j$ and (ii) $\mathsf{SigVer}(vk_j, \tilde{\gamma}^*_j, \tilde{\sigma}_j) = 1$ and $(\tilde{\gamma}^*_j, \tilde{\sigma}_j) \neq (\gamma^*_j, \sigma_j)$. Let $\mathbf{W} := \mathbf{W}_{\mathsf{L}} \cup \mathbf{W}_{\mathsf{R}}$, where $\mathbf{W}_{\mathsf{L}} := \bigcup_{q \in [p]} \mathbf{W}_{\mathsf{L}}^{(q)}$ and $\mathbf{W}_{\mathsf{R}} := \bigcup_{q \in [p]} \mathbf{W}_{\mathsf{R}}^{(q)}$. Informally, $\mathbf{W}$ is the event in which the adversary A against $\mathbf{G}^*_{\mathsf{A}}(\lambda, b)$ modifies the message but not the codeword, thus successfully forging a signature.

For $b \in \{0, 1\}$, let $\mathbf{H}^*_{\mathsf{A}}(\lambda, b)$ be the experiment $\mathbf{G}^*_{\mathsf{A}}(\lambda, b)$ in which the challenger aborts whenever $\mathbf{W}$ happens. Clearly, the two experiments $\mathbf{G}^*$ and $\mathbf{H}^*$ are only distinguishable when $\mathbf{W}$ happens, therefore, if we show that $\mathbf{W}$ happens with negligible probability, it follows that $\mathbf{G}^*(\lambda, b)$ and $\mathbf{H}^*(\lambda, b)$ are statistically close.

Lemma 2. *For all PPT adversaries* A *there is a negligible function* $\nu : \mathbb{N} \to [0, 1]$ *such that* $\Pr[\mathbf{W}] \leq \nu(\lambda)$.

Reducing to split-state non-malleability with augmented adversaries. Now we want to perform the reduction to the split-state super non-malleable code. Unfortunately, we cannot convert each decision-tree tampering query to a split-state tampering query because each conversion needs some leakage and we would end up leaking too many bits. However, the reduction only needs to simulate the first tampering query which actually modifies the codeword, because the answer to all previous queries is $\diamond$.

In order to apply this idea, we first define an experiment $\mathbf{H}^{\mathsf{aug}}$ which, informally, is the same of $\mathbf{G}$ except that the adversary is given one-time oracle access to the needed information, i.e. the index of the first tampering query actually modifying the codeword. The only remaining problem is that the reduction $\hat{\mathsf{A}}^{\mathsf{aug}}$ still performs too much leakage to verify whether the tampered codeword is valid or not; this is because the reduction should check that the padding string inside the codeword γ^* does not contain any 1. The solution is to give the reduction $\hat{\mathsf{A}}^{\mathsf{aug}}$ oracle access to this information too, so that now $\hat{\mathsf{A}}^{\mathsf{aug}}$ is able to simulate the experiment to the adversary A without performing too much leakage.

More formally, let A be an adversary telling apart $\mathbf{H}^*(\lambda, 0)$ and $\mathbf{H}^*(\lambda, 1)$ with non-negligible advantage and let $\mathsf{RunInfo}_{\mathsf{A}}(\tau)$ be a function which takes as input the transcript τ of the execution of $\mathsf{A}(1^\lambda; \rho_{\mathsf{A}})$ when the codeword is γ^* and outputs the index $q_\diamond$ of the first tampering query that is not answered $\diamond$ and a bit $b_\perp$ which is 1 if the output of such tampering query is $\perp$ and 0 otherwise. Notice that τ is uniquely determined by the random coins ρ_{A} of A and the decision-tree codeword $\gamma^* = \mathsf{comp}(\gamma)$ which is compiled by some deterministic compilation instructions comp from the split-state codeword γ. Therefore, we can define the oracle $\mathcal{O}^{\mathsf{aug}}_{\mathsf{A}}(\gamma, \mathsf{comp}, \rho_{\mathsf{A}})$ which is initialized with the split-state codeword γ and, upon receiving the query containing the instructions comp and the random coins ρ_{A}, computes τ and outputs $\mathsf{RunInfo}_{\mathsf{A}}(\tau)$. Consider the experiment $\mathbf{H}^{\mathsf{aug}}_{\hat{\mathsf{A}},\mathsf{A}}(\lambda, b)$ which is exactly the same as $\mathbf{G}_{\hat{\mathsf{A}}}$ except that $\hat{\mathsf{A}}$ is given one-time oracle access to $\mathcal{O}^{\mathsf{aug}}_{\mathsf{A}}(\gamma, \cdot, \cdot)$ before the tampering query.

Let $\mathcal{I}^{\mathsf{sgn}}_{\mathsf{L}}, \mathcal{I}^{\mathsf{rss}}_{\mathsf{L}}, \mathcal{I}^{\mathsf{str}}_{\mathsf{L}}, \mathcal{I}^{\mathsf{sgn}}_{\mathsf{R}}, \mathcal{I}^{\mathsf{rss}}_{\mathsf{R}}, \mathcal{I}^{\mathsf{str}}_{\mathsf{R}}$ be a partition of $[n]$ such that, given an encoding γ^*, $\mathcal{I}^{\mathsf{sgn}}_{\mathsf{L}}$ (resp. $\mathcal{I}^{\mathsf{sgn}}_{\mathsf{R}}$) contains the positions of γ^* in which is stored the left (resp. right) signature, $\mathcal{I}^{\mathsf{rss}}_{\mathsf{L}}$ (resp. $\mathcal{I}^{\mathsf{rss}}_{\mathsf{R}}$) contains the positions of γ^* in which is stored the left (resp. right) ramp secret sharing and $\mathcal{I}^{\mathsf{str}}_{\mathsf{L}}$ (resp. $\mathcal{I}^{\mathsf{str}}_{\mathsf{R}}$) contains the positions of γ^* in which is stored the string containing the left (resp. right) part of the codeword and the left (resp. right) zeroes. For $j \in \{\mathsf{L}, \mathsf{R}\}$, let $\mathcal{I}_j = \mathcal{I}^{\mathsf{sgn}}_j \cup \mathcal{I}^{\mathsf{rss}}_j \cup \mathcal{I}^{\mathsf{str}}_j$. We now show a reduction $\hat{\mathsf{A}}^{\mathsf{aug}}$ which is able to tell apart $\mathbf{H}^{\mathsf{aug}}_{\hat{\mathsf{A}}^{\mathsf{aug}},\mathsf{A}}(\lambda, 0)$ and $\mathbf{H}^{\mathsf{aug}}_{\hat{\mathsf{A}}^{\mathsf{aug}},\mathsf{A}}(\lambda, 1)$ with non-negligible advantage.

1. Sample random coins $\rho_{\mathsf{A}}, \rho^{\mathsf{enc}}_{\mathsf{L}}, \rho^{\mathsf{enc}}_{\mathsf{R}}, \rho^{\mathsf{sgn}}_{\mathsf{L}}, \rho^{\mathsf{sgn}}_{\mathsf{R}}$ and random strings $\zeta_{\mathsf{L}}, \zeta_{\mathsf{R}}$.
2. For $j \in \{\mathsf{L}, \mathsf{R}\}$, let $\mathcal{I}^{\mathsf{cdw}}_j = \phi(\zeta_j)$ and $\mathcal{I}^{\mathsf{zero}}_j = \mathcal{I}^{\mathsf{str}}_j \setminus \mathcal{I}^{\mathsf{cdw}}_j$.
3. Compute $\omega_{\mathsf{L}} = \mathsf{Enc}^{\mathsf{L}}_{\mathsf{RSS}}(\zeta_{\mathsf{L}}; \rho^{\mathsf{enc}}_{\mathsf{L}})$ and $\omega_{\mathsf{R}} = \mathsf{Enc}^{\mathsf{R}}_{\mathsf{RSS}}(\zeta_{\mathsf{R}}; \rho^{\mathsf{enc}}_{\mathsf{R}})$.
4. Sample $(sk_{\mathsf{L}}, vk_{\mathsf{L}}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ and $(sk_{\mathsf{R}}, vk_{\mathsf{R}}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$.
5. Run $\mathsf{A}(1^\lambda; \rho_{\mathsf{A}})$, obtaining the challenge messages μ_0, μ_1; then construct the challenge messages $\mu^*_0 := \mu_0 || vk_{\mathsf{L}} || vk_{\mathsf{R}}$ and $\mu^*_1 := \mu_1 || vk_{\mathsf{L}} || vk_{\mathsf{R}}$ and send μ^*_0, μ^*_1 to the challenger.
6. For $j \in \{\mathsf{L}, \mathsf{R}\}$, construct the leakage function g^{sgn}_j which hard-wires the values $\rho^{\mathsf{enc}}_j, \rho^{\mathsf{sgn}}_j, \zeta_j, \omega_j, sk_j$ and, upon input the codeword part γ_j, computes $\gamma^*_j = \mathsf{Enc}_j(\gamma_j; \rho^{\mathsf{enc}}_j, \zeta_j)$ and $\sigma_j = \mathsf{Sign}(sk_j, \gamma^*_j; \rho^{\mathsf{sgn}}_j)$ and outputs σ_j.
7. Send $(g^{\mathsf{sgn}}_{\mathsf{L}}, g^{\mathsf{sgn}}_{\mathsf{R}})$ to the leakage oracle, thus obtaining the signatures $(\sigma_{\mathsf{L}}, \sigma_{\mathsf{R}})$.
8. Let $\mathcal{L}$ be a set which, initially, contains all the pairs (i, b) such that $i \in [n] \setminus \mathcal{I}^{\mathsf{cdw}}_{\mathsf{L}} \cup \mathcal{I}^{\mathsf{cdw}}_{\mathsf{R}}$ and $b = \gamma^*[i]$. Notice that the only bits unknown to $\hat{\mathsf{A}}$ are

the ones belonging to the split-state codeword, namely, the ones in $\mathcal{I}_{\mathsf{L}}^{\mathsf{cdw}}$ and in $\mathcal{I}_{\mathsf{R}}^{\mathsf{cdw}}$; therefore, $\hat{\mathsf{A}}$ is able to construct the set $\mathcal{L}$.

9. Using the information in $\mathcal{L}$, construct the compilation information comp which is used to compile the split-state codeword γ into the decision-tree codeword $\gamma^* = \mathsf{comp}(\gamma)$.
10. Send $(\mathsf{comp}, \rho_{\mathsf{A}})$ to the augmented oracle $\mathcal{O}_{\mathsf{A}}^{\mathsf{aug}}$, thus receiving a pair $(q_\diamond, b_\perp)$.
11. Upon receiving the q-th tampering query $f^{(q)} \in \mathcal{F}_{\mathsf{dtree}}^{d}$ from A, return $\diamond$ if $q < q_\diamond$; otherwise let $f = f^{(q)}$ and do the following.
    (a) *Obtaining the necessary bits:* run the procedure

$$\mathcal{L} \leftarrow \mathsf{ObtainBits}_{\mathsf{Leak},\mathsf{Update}}(f, \mathcal{I}_{\mathsf{L}}^{\mathsf{sgn}} \cup \mathcal{I}_{\mathsf{L}}^{\mathsf{rss}} \cup \mathcal{I}_{\mathsf{R}}^{\mathsf{sgn}} \cup \mathcal{I}_{\mathsf{R}}^{\mathsf{rss}}, \mathcal{L}),$$

    then use the set $\mathcal{L}$ to compute the tampered signatures $\tilde{\sigma}_{\mathsf{L}}, \tilde{\sigma}_{\mathsf{R}}$ and the tampered encodings $\tilde{\omega}_{\mathsf{L}}, \tilde{\omega}_{\mathsf{R}}$.
    (b) *Obtaining the new positions:* for $j \in \{\mathsf{L}, \mathsf{R}\}$, compute $\tilde{\zeta}_j = \mathsf{Dec}_{\mathsf{RSS}}(\tilde{\omega}_j)$ and let $\tilde{\mathcal{I}}_j^{\mathsf{cdw}} = \phi(\tilde{\zeta}_j)$ and $\tilde{\mathcal{I}}_j^{\mathsf{zero}} = \tilde{\mathcal{I}}_j^{\mathsf{str}} \setminus \tilde{\mathcal{I}}_j^{\mathsf{cdw}}$.
    (c) *Leaking the remaining bits for the left part:* construct the leakage function $\hat{g}_{\mathsf{L}}^{\mathcal{L}}$ which hard-wires (a description of) the tampering function f, the sets $\mathcal{I}_{\mathsf{L}}^{\mathsf{cdw}}$ and $\tilde{\mathcal{I}}_{\mathsf{L}}^{\mathsf{cdw}}$ and the set $\mathcal{L}$ and, upon input the left part γ_{L} of the codeword, constructs the set $\mathcal{L}_{\mathsf{L}} = \{(i, \gamma^*[i]) : i \in \mathcal{I}_{\mathsf{L}}^{\mathsf{cdw}}\}$, runs the procedure

$$y \leftarrow \mathsf{ObtainBits}_{\mathsf{Await},\mathsf{Ready}}(f, \tilde{\mathcal{I}}_{\mathsf{L}}^{\mathsf{cdw}}, \mathcal{L} \cup \mathcal{L}_{\mathsf{L}}) \tag{5}$$

    and returns y. Then, send $(\hat{g}_{\mathsf{L}}^{\mathcal{L}}, \epsilon)$ to the leakage oracle, thus obtaining a value y. If $y = (\mathsf{await}, i)$ for some $i \in \mathcal{I}_{\mathsf{R}}^{\mathsf{cdw}}$, leak $\gamma^*[i]$ from the right part of the codeword, update $\mathcal{L} \leftarrow \mathcal{L} \cup \{(i, \gamma^*[i])\}$ and repeat this step.
    (d) *Leaking the remaining bits for the right part:* construct the leakage function $\hat{g}_{\mathsf{R}}^{\mathcal{L}}$ which hard-wires (a description of) the tampering function f, the sets $\mathcal{I}_{\mathsf{R}}^{\mathsf{cdw}}$ and $\tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{cdw}} \cup \tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{zero}}$ and the set $\mathcal{L}$ and, upon input the right part γ_{R} of the codeword, constructs the set $\mathcal{L}_{\mathsf{R}} = \{(i, \gamma^*[i]) : i \in \mathcal{I}_{\mathsf{R}}^{\mathsf{cdw}}\}$, runs the procedure

$$y \leftarrow \mathsf{ObtainBits}_{\mathsf{Await},\mathsf{Ready}}(f, \tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{cdw}} \cup \tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{zero}}, \mathcal{L} \cup \mathcal{L}_{\mathsf{R}})$$

    and returns y. Then, send $(\epsilon, \hat{g}_{\mathsf{R}}^{\mathcal{L}})$ to the leakage oracle, thus obtaining a value y. If $y = (\mathsf{await}, i)$ for some $i \in \mathcal{I}_{\mathsf{L}}^{\mathsf{cdw}}$, leak $\gamma^*[i]$ from the left part of the codeword, update $\mathcal{L} \leftarrow \mathcal{L} \cup \{(i, \gamma^*[i])\}$ and repeat this step.
    (e) *Validating the right part:* construct the leakage function $\hat{h}_{\mathsf{R}}^{\mathsf{chk}}$ which hard-wires (a description of) the tampering function f, the sets $\mathcal{I}_{\mathsf{R}}^{\mathsf{cdw}}$ and $\tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{cdw}} \cup \tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{zero}}$ and the set $\mathcal{L}$ and, upon input the right part γ_{R} of the codeword, constructs the set $\mathcal{L}_{\mathsf{R}} = \{(i, \gamma^*[i]) : i \in \mathcal{I}_{\mathsf{R}}^{\mathsf{cdw}}\}$, runs the procedure

$$b_{\mathsf{valid}} \leftarrow \mathsf{ObtainBits}_{\mathsf{Await},\mathsf{Check}}(f, \tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{cdw}} \cup \tilde{\mathcal{I}}_{\mathsf{R}}^{\mathsf{zero}}, \mathcal{L} \cup \mathcal{L}_{\mathsf{R}})$$

    and returns b_{valid}. Then, send $(\epsilon, \hat{h}_{\mathsf{R}}^{\mathsf{chk}})$ to the leakage oracle, thus obtaining the bit b_{valid}. If $b_{\mathsf{valid}} = 0$, abort the simulation and return a random guess.

(f) *Tampering with the codeword:* for $j \in \{\mathsf{L},\mathsf{R}\}$, construct the tampering function $\hat{f}_j$ which hard-wires the strings $\sigma_{\mathsf{L}}, \sigma_{\mathsf{R}}, \omega_{\mathsf{L}}, \omega_{\mathsf{R}}$, the sets $\mathcal{I}^{\mathsf{cdw}}_{\mathsf{L}}$, $\mathcal{I}^{\mathsf{cdw}}_{\mathsf{R}}$, $\mathcal{L}$ and (a description of) the tampering query f and, upon input the codeword part γ_j, computes the tampered codeword part $\tilde{\gamma}_j$ by using the additional bits given by $\mathcal{L}$ and then returns $\tilde{\gamma}_j$. Send the query $(\hat{f}_{\mathsf{L}}, \hat{f}_{\mathsf{R}})$ to the tampering oracle, thus obtaining a codeword $\tilde{\gamma} \in \{0,1\}^{2c} \cup \{\diamond, \perp\}$. If $\tilde{\gamma} \in \{\diamond, \perp\}$, abort the simulation and return a random guess. Otherwise, let $\tilde{\gamma} = (\tilde{\gamma}_{\mathsf{L}}, \tilde{\gamma}_{\mathsf{R}})$, reconstruct $\tilde{\gamma}^*_{\mathsf{L}}$ (resp. $\tilde{\gamma}^*_{\mathsf{R}}$) using the value $\tilde{\gamma}_{\mathsf{L}}$ and the set $\mathcal{I}^{\mathsf{cdw}}_{\mathsf{L}}$ (resp. the value $\tilde{\gamma}_{\mathsf{R}}$ and the set $\mathcal{I}^{\mathsf{cdw}}_{\mathsf{R}}$) and set $\tilde{\gamma}^* = (\tilde{\sigma}_{\mathsf{L}}, \tilde{\gamma}^*_{\mathsf{L}}, \tilde{\sigma}_{\mathsf{R}}, \tilde{\gamma}^*_{\mathsf{R}})$.

(g) *Checking the signature:* reconstruct the tampered message $\tilde{\mu}||\tilde{vk}_{\mathsf{L}}||\tilde{vk}_{\mathsf{R}}$. Then, check that $\tilde{vk}_{\mathsf{L}} \neq vk_{\mathsf{L}}$ and $\tilde{vk}_{\mathsf{R}} \neq vk_{\mathsf{R}}$, compute $b_{\mathsf{L}} = \mathsf{SigVer}(vk_{\mathsf{L}}, \tilde{\sigma}_{\mathsf{L}}, \tilde{\gamma}^*_{\mathsf{L}})$ and $b_{\mathsf{R}} = \mathsf{SigVer}(vk_{\mathsf{R}}, \tilde{\sigma}_{\mathsf{R}}, \tilde{\gamma}^*_{\mathsf{R}})$, check that $b_{\mathsf{L}} = b_{\mathsf{R}} = 1$ and abort the simulation returning a random guess if one of the previous checks fails.

Finally, set $\tilde{\gamma}^* = \perp$ if $b_\perp = 1$, return $\tilde{\gamma}^*$ to A and output the same distinguishing bit as A.

For the analysis, notice that the reduction $\hat{\mathsf{A}}^{\mathsf{aug}}$ perfectly simulates $\mathbf{H}^*(\lambda, b)$ to A unless the leakage performed exceeds the admissible leakage $\alpha c - 1$; therefore, when the leakage is within the bounds, $\hat{\mathsf{A}}^{\mathsf{aug}}$ has the same advantage of A.

The following lemma allows us to conclude that $\mathbf{G}_{\hat{\mathsf{A}}}(\lambda, 0)$ and $\mathbf{G}_{\hat{\mathsf{A}}}(\lambda, 1)$ are computationally close.

Lemma 3. *If there exists an adversary $\hat{\mathsf{A}}^{\mathsf{aug}}$ which is able to distinguish between $\mathbf{H}^{\mathsf{aug}}_{\hat{\mathsf{A}}^{\mathsf{aug}},\mathsf{A}}(\lambda, 0)$ and $\mathbf{H}^{\mathsf{aug}}_{\hat{\mathsf{A}}^{\mathsf{aug}},\mathsf{A}}(\lambda, 1)$ with non-negligible advantage, then there exists an adversary $\hat{\mathsf{A}}$ which is able to distinguish between $\mathbf{G}_{\hat{\mathsf{A}}}(\lambda, 0)$ and $\mathbf{G}_{\hat{\mathsf{A}}}(\lambda, 1)$ with non-negligible advantage.*

Bounding the leakage. It remains to bound the leakage made by the reduction.

Proposition 1 ([15, Proposition 1]). *Let $n, c, t \in \mathbb{N}$ such that $t \geq c \log n$. Let A be an arbitrary algorithm that reads adaptively at most t bits of $(\mathsf{Enc}_{\mathsf{RSS}}(\zeta), \phi(\zeta))$. Let $\mathbf{Y}$ denote the number of distinct 1's in $\phi(\zeta)$ which are read by A. Then, over the randomness of ζ and $\mathsf{Enc}_{\mathsf{RSS}}$,*

$$\Pr\left[\mathbf{Y} \geq \frac{2tc}{n}\right] \leq \exp\left(-\frac{tc}{3n}\right).$$

Lemma 4. *Suppose $t_{\mathsf{R}} \geq (4t_{\mathsf{L}} + c + 2s)d$. Let $\ell^{\mathsf{bit}}_{\mathsf{R}}$ be the amount of positions b leaked from γ_{R}. Then, for any $\gamma \in \{0,1\}^{2c}$, the event that $\ell^{\mathsf{bit}}_{\mathsf{R}} \geq 2(4t_{\mathsf{R}} + 4t_{\mathsf{L}} + c + 2s)dc/n_{\mathsf{R}}$ happens with probability at most $(4d+1)\exp(-t_{\mathsf{R}}c/3n_{\mathsf{R}})$.*

Lemma 5. *Suppose $t_{\mathsf{L}} \geq d$. Let $\ell^{\mathsf{bit}}_{\mathsf{L}}$ be the amount of positions b leaked from γ_{L}. Then, for any $\gamma \in \{0,1\}^{2c}$, the event that $\ell^{\mathsf{bit}}_{\mathsf{L}} \geq 2(4t_{\mathsf{L}} + 4t_{\mathsf{R}} + n_{\mathsf{R}} + 2s)dc/n_{\mathsf{L}}$ happens with probability at most $(4t_{\mathsf{L}} + 4t_{\mathsf{R}} + n_{\mathsf{R}} + 2s)d/t_{\mathsf{L}} \exp(-t_{\mathsf{L}}c/3n_{\mathsf{L}})$.*

Let

$$\ell^{\mathsf{tamp}} = \left(\ell^{\mathsf{bit}}_{\mathsf{L}} + \ell^{\mathsf{bit}}_{\mathsf{R}}\right)(1 + \log(c))$$

$$= 2dc\left(\frac{4t_{\mathsf{L}} + 4t_{\mathsf{R}} + n_{\mathsf{R}} + 2s}{n_{\mathsf{L}}} + \frac{4t_{\mathsf{R}} + 4t_{\mathsf{L}} + c + 2s}{n_{\mathsf{R}}}\right)(1 + \log(c)).$$

By the above lemmas, the event that the amount of leakage performed by $\hat{\mathsf{A}}$ exceeds $\ell^{\mathsf{tamp}}+2s+2$ (recall that the reduction also leaks $2s$ bits for the signatures and 2 bits for checking the simulation) happens with probability at most

$$(4d+1)\exp(-t_{\mathsf{R}}c/3n_{\mathsf{R}}) + (4t_{\mathsf{L}} + 4t_{\mathsf{R}} + n_{\mathsf{R}} + 2s)d/t_{\mathsf{L}}\exp(-t_{\mathsf{L}}c/3n_{\mathsf{L}}). \quad (6)$$

Lemma 6. *Fix $\alpha \in (0,1)$. Then, there exist constants $\eta_1, \eta_2, \eta_3, \eta_4$ (only dependent on α) such that, if $t_{\mathsf{L}} = \eta_1 c \log n$, $t_{\mathsf{R}} = \eta_2 dc \log n \log c$, $n_{\mathsf{L}} = \eta_3 d^3 c \log n \log^3 c$, $n_{\mathsf{R}} = \eta_4 d^2 c \log n \log^2 c$, then $\ell^{\mathsf{tamp}} \leq \alpha c$ with overwhelming probability.*

By choosing the parameters as in Lemma 6, the length of the final codeword satisfies

$$n = 2s + 4t_{\mathsf{L}} + 4t_{\mathsf{R}} + n_{\mathsf{L}} + n_{\mathsf{R}} = O(d^3 c \log n \log^3 c),$$

which can be rewritten as $n/\log n = O(c^{7/4}\log^3 c)$, thus making $n = O(c^2)$ a good approximation, and the total amount of leakage is $\ell = \ell^{\mathsf{tamp}} + 2\beta c + 2$, which, with a good choice of the parameters $\eta_1, \ldots, \eta_4$ and α, β, can simply be rewritten as $\ell \leq \alpha c$. This concludes the proof of Theorem 4. □

4.2 Bounded Polynomial-Depth Tampering

Our construction for bounded polynomial-depth tampering, works in three steps.

(i) First, we show a compiler for turning any *leakage-resilient* non-malleable code into a leakage-resilient *super* non-malleable code; the compiler is non-black-box, as it relies on NIZK proofs, and thus yields a code in the CRS model (even if the initial code is in the plain model).
(ii) Second, we show how to instantiate the above compiler by simplifying the non-malleable code for bounded polynomial-depth tampering of Dachman-Soled *et al.* [35] (thanks to the fact that we rely on trusted setup).
(iii) Third, we argue that the family of bounded polynomial-depth tampering satisfies the conditions of Theorem 3, so that persistent continuous non-malleability follows by steps (i) and (ii).

Let $\Gamma = (\mathsf{Enc}, \mathsf{Dec})$ be a (k,n)-code with randomness space $\{0,1\}^r$, let $\mathsf{G} : \{0,1\}^s \to \{0,1\}^r$ be a PRG, and let $\Pi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{ProofVer})$ be a non-interactive argument system with proof space $\mathcal{P} = \{0,1\}^m$ for the relation:

$$\mathcal{R} = \left\{(\gamma, \sigma) \in \{0,1\}^n \times \{0,1\}^s : \exists \mu \in \{0,1\}^k \text{ s.t. } \gamma = \mathsf{Enc}(\mu; \mathsf{G}(\sigma))\right\}. \quad (7)$$

Consider the following $(k, n+m)$-code $\Gamma^* = (\mathsf{Init}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ in the CRS model.

Initialization: The initialization algorithm Init^* outputs $\omega \leftarrow\!\!\$\ \mathsf{CRSGen}(1^\lambda)$.
Encoding: The encoding algorithm Enc^* proceeds as follows:

- sample a uniformly random seed $\sigma \leftarrow\!\!\$\ \{0,1\}^s$ and compute $\rho = \mathsf{G}(\sigma)$;
- let $\gamma = \mathsf{Enc}(\mu;\rho)$;
- run $\pi \leftarrow\!\!\$\ \mathsf{Prove}(\omega,\gamma,\sigma)$;
- return $\gamma^* = (\gamma,\pi)$.

Decoding: The decoding algorithm Dec^*, upon input a codeword $\gamma^* = (\gamma,\pi)$, proceeds as follows:

- run $\mathsf{ProofVer}(\omega,\gamma,\pi)$, and output $\perp$ if the verification fails;
- compute $\mu = \mathsf{Dec}(\gamma)$, and output $\perp$ if the decoding fails;
- else, return μ.

Let $\mathcal{F} \subseteq \{f : \{0,1\}^{n+m} \to \{0,1\}^{n+m}\}$ be a family of functions. In the theorem below, for any function $f \in \mathcal{F}$, and any $\gamma \in \{0,1\}^n$ and $\pi \in \{0,1\}^m$, we write $f(\gamma,\pi)_1$ (resp. $f(\gamma,\pi)_2$) for the function that outputs the first n bits (resp. the last m bits) of $f(\gamma,\pi)$.

Theorem 5. *Assume that Π is a one-time simulation extractable non-interactive zero-knowledge argument system for the relation of Eq. (7), with proof space $\mathcal{P} = \{0,1\}^m$, with zero-knowledge simulator $\mathsf{S} = (\mathsf{S}_0,\mathsf{S}_1)$ and with extractor K. Let Γ be a $(\mathcal{G}(n),\ell+s+m)$-leakage-resilient $\mathcal{F}(n)$-non-malleable (k,n)-code w.r.t. message/codeword.*

Then, the above defined $(k,n+m)$-code Γ^* is $(\mathcal{G}(n+m),\ell)$-leakage-resilient $\mathcal{F}(n+m)$-super-non-malleable w.r.t. message/codeword, so long as for every $g \in \mathcal{G}(n+m)$ and every $f \in \mathcal{F}(n+m)$, all $\gamma \in \{0,1\}^n$, all $\pi \in \{0,1\}^m$, and all $(\omega,\zeta,\xi) \in \mathsf{S}_0(1^\lambda)$, it holds that:

$$\mathcal{G} \supseteq \{g(\cdot,\mathsf{S}_1(\zeta,\cdot))_1, \mathsf{ProofVer}(\omega, f(\cdot,\mathsf{S}_1(\zeta,\cdot))), \mathsf{K}(\xi, f(\cdot,\mathsf{S}_1(\zeta,\cdot)))\} \tag{8}$$

$$\mathcal{F} \ni f(\cdot,\mathsf{S}_1(\zeta,\cdot))_1. \tag{9}$$

Instantiating the proof system. Since the underlying code Γ needs to tolerate at least m bits of leakage, where m is the size of a proof under Π, Theorem 5 implicitly requires proofs that are sub-linear in the size of the statement (which is a codeword), but not of the witness (which is a seed for the PRG). In the literature, such proofs are referred as Succinct Non-interactive Arguments of Knowledge (SNARKs). In [9], the authors present a simulation-extractable SNARK whose proofs consist of 4 group elements. The security proof relies on both the generic group model (GGM) and the random oracle model (ROM).

Alternatively, we can use [46], where fully-homomorphic encryption (FHE) and NIZK argument systems are used to achieve succinct-proof NIZK argument systems for all of NP. The succinct proof for an NP relation $\mathcal{R}$ is built as follows:

- The witness x is encrypted with key σ into a ciphertext u of the same length by means of a symmetric-key encryption scheme (namely, one-time pad with a pseudorandom key generated from σ via a PRG G).
- The key generation algorithm of the FHE is called with randomness ρ to get (pk,sk). Next, the FHE scheme is used with keys (pk,sk) and randomness τ to encrypt the symmetric key σ into a ciphertext z. Then, the FHE evaluation algorithm takes as input the ciphertext z, and the NP relation $\mathcal{R}$ over statement y and witness $u \oplus \mathsf{G}(\cdot)$, and returns a ciphertext v.

- The underlying prover provides an argument π for the statement (pk, z, v) and witness (ρ, σ, τ), proving that (pk, sk) are generated according to ρ, that z is an encryption of σ according to pk, τ and that v decrypts to 1.
- The succinct proof is given by (pk, z, u, π).

Since $|u| = |x|$ and (pk, z, π) are polynomial in the security parameter, the proof size is $|x| + poly(\lambda)$. Also note that (pk, z, u, π) is sufficient to verify the proof, as one can obtain v and then call the underlying verification algorithm.

In their work, Gentry *et al.* [46] show that this transformation preserves the soundness and the zero-knowledge property of the underlying NIZK argument system. However, their result also applies to simulation extractability. For a high-level idea, call A an adversary against the simulation-extractability of the succinct-proof scheme. Assume that, given a simulated proof (pk, z, u, π) for a statement y of its choice, A manages to produce an accepting and fresh pair $(\tilde{y}, (\tilde{pk}, \tilde{u}, \tilde{z}, \tilde{\pi}))$. Consider the extractor that takes as input $(\tilde{y}, (\tilde{pk}, \tilde{u}, \tilde{z}, \tilde{\pi}))$ and as trapdoor (pk, sk), and does the following. If $\tilde{pk} \neq pk$, it computes the homomorphic evaluation $\tilde{v}$ of the circuit $\mathcal{R}(\tilde{y}, \tilde{u} \oplus \mathsf{G}(\cdot))$ on ciphertext $\tilde{z}$ with $\tilde{pk}$. Then, it runs the underlying extractor over $((\tilde{pk}, \tilde{z}, \tilde{v}), \tilde{\pi})$ to get $(\tilde{\rho}, \tilde{\sigma}, \tilde{\tau})$. If $\tilde{pk} = pk$, the extractor only needs to decrypt $\tilde{z}$ to get $\tilde{\sigma}$. In both cases, it outputs $\tilde{x} = \tilde{u} \oplus \mathsf{G}(\tilde{\sigma})$.

Instantiating the underlying code. To instantiate the underlying code, we start from the construction of Dachman-Soled *et al.* [35] which is in the plain model and relies on key-less hash functions, time-lock puzzles, as well as other standard assumptions. In the CRS model, their construction can be simplified as follows: The encoding of a message μ consists of a time-lock puzzle ζ computed using μ (with some fixed difficulty parameter) and a simulation-extractable NIZK proof of knowledge π of the message μ inside the puzzle. We refer the reader to [25] for the formal description and the security analysis in the CRS model.

Proving continuous non-malleability. Finally, we invoke Theorem 3 to conclude persistent continuous non-malleability. To do that, we need to check that the leakage family of bounded polynomial-depth circuits contains the function $\hat{g}$ in the statement of the theorem. In our case, it suffices to consider leakage resilience against circuits of depth $\leq T + c$ for a small constant c, and compute the leakage function $\hat{g}$ as follows. Upon input the codeword γ, consider q parallel sub-circuits, where the i-th circuit computes $f^{(i)}(\gamma)$, and outputs $b_i = 1$ if $f^{(i)}(\gamma) = \gamma$, $b_i = 0$ otherwise. The circuit will then output 1 if $b_1 = \cdots = b_{q-1} = 0$ and $b_q = 1$, and 0 otherwise. By inspection, every sub-circuit has depth $\leq T + c$, as it computes a tampering function and a bit-wise comparison (feasible in constant depth). To check if $b_1 = \cdots = b_{q-1} = 0$, it suffices to compute $b = \mathbf{OR}(b_1, \ldots, b_{q-1})$. The leakage function finally outputs $\mathbf{AND}(\mathbf{NOT}(b), b_q)$.

5 Conclusions

We have shown how to achieve continuous non-malleability in two natural settings: (i) decision-tree tampering, and (ii) bounded polynomial-depth tampering.

The first result is in the plain model; the second result requires trusted setup. Both constructions rely on computational assumptions (one-way functions in (i), and time-lock puzzles and simulation-extractable succinct-proof NIZKs in (ii)). Natural open problems include: removing computational assumptions from our construction in (i), and weakening the assumptions from our construction in (ii). We leave these as interesting directions for future research.

Our paper provides the first crucial insights for constructing continuously non-malleable codes against non-compartmentalized tampering. In particular:

- We prove for the first time that security against non-persistent global tampering is impossible in the continuous setting.
- We prove for the first time that, when the target tampering family is powerful enough, continuous non-malleability follows from one-time super non-malleability with log bits of leakage resilience. The latter, in particular, is true for bounded-depth tampering and for AC0 tampering.
- We show a generic transform to reduce one-time *super* non-malleability to one-time non-malleability using NIZK proofs; this transform requires the underlying tampering family to satisfy certain properties, which are met in the setting of bounded polynomial-depth tampering.

We believe the above observations are important, and will turn useful for future constructions of continuously non-malleable codes against other non-compartmentalized tampering families (e.g., AC0 tampering), possibly under weaker assumptions.

References

1. Michel Abdalla, Fabrice Benhamouda, Alain Passelègue, and Kenneth G. Paterson. Related-key security for pseudorandom functions beyond the linear barrier. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 77–94. Springer, Heidelberg, August 2014.
2. Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In David B. Shmoys, editor, *46th ACM STOC*, pages 774–783. ACM Press, May / June 2014.
3. Divesh Aggarwal, Nico Döttling, Jesper Buus Nielsen, Maciej Obremski, and Erick Purwanto. Continuous non-malleable codes in the 8-split-state model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 531–561. Springer, Heidelberg, May 2019.
4. Divesh Aggarwal, Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, Maciej Obremski, and Sruthi Sekar. Rate one-third non-malleable codes. Cryptology ePrint Archive, Report 2021/1042, 2021. https://eprint.iacr.org/2021/1042.
5. Divesh Aggarwal, Tomasz Kazana, and Maciej Obremski. Inception makes non-malleable codes stronger. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 319–343. Springer, Heidelberg, November 2017.
6. Divesh Aggarwal and Maciej Obremski. A constant rate non-malleable code in the split-state model. In *61st FOCS*, pages 1285–1294. IEEE Computer Society Press, November 2020.

7. Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Explicit non-malleable codes against bit-wise tampering and permutations. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 538–557. Springer, Heidelberg, August 2015.
8. Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *ICS 2011*, pages 45–60. Tsinghua University Press, January 2011.
9. Karim Baghery, Zaira Pindado, and Carla Ràfols. Simulation extractable versions of groth's zk-SNARK revisited. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 453–461. Springer, Heidelberg, December 2020.
10. Marshall Ball, Dana Dachman-Soled, Siyao Guo, Tal Malkin, and Li-Yang Tan. Non-malleable codes for small-depth circuits. In Mikkel Thorup, editor, *59th FOCS*, pages 826–837. IEEE Computer Society Press, October 2018.
11. Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, Huijia Lin, and Tal Malkin. Non-malleable codes against bounded polynomial time tampering. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 501–530. Springer, Heidelberg, May 2019.
12. Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes for bounded depth, bounded fan-in circuits. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 881–908. Springer, Heidelberg, May 2016.
13. Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 618–650. Springer, Heidelberg, April / May 2018.
14. Marshall Ball, Dana Dachman-Soled, and Julian Loss. (Nondeterministic) hardness vs. non-malleability. Cryptology ePrint Archive, Report 2022/070, 2022. https://eprint.iacr.org/2022/070.
15. Marshall Ball, Siyao Guo, and Daniel Wichs. Non-malleable codes for decision trees. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 413–434. Springer, Heidelberg, August 2019.
16. Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 666–684. Springer, Heidelberg, August 2010.
17. Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 486–503. Springer, Heidelberg, December 2011.
18. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 491–506. Springer, Heidelberg, May 2003.
19. Mihir Bellare, Kenneth G. Paterson, and Susan Thomson. RKA security beyond the linear barrier: IBE, encryption and signatures. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 331–348. Springer, Heidelberg, December 2012.
20. Eli Biham. New types of cryptanalytic attacks using related keys (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 398–409. Springer, Heidelberg, May 1994.

21. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, Heidelberg, August 1997.
22. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 37–51. Springer, Heidelberg, May 1997.
23. Gianluca Brian, Antonio Faonio, Maciej Obremski, Mark Simkin, and Daniele Venturi. Non-malleable secret sharing against bounded joint-tampering attacks in the plain model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 127–155. Springer, Heidelberg, August 2020.
24. Gianluca Brian, Antonio Faonio, and Daniele Venturi. Continuously non-malleable secret sharing: Joint tampering, plain model and capacity. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 333–364. Springer, Heidelberg, November 2021.
25. Gianluca Brian, Sebastian Faust, Elena Micheli, and Daniele Venturi. Continuously non-malleable codes against bounded-depth tampering. Cryptology ePrint Archive, Paper 2022/1231, 2022. https://eprint.iacr.org/2022/1231.
26. Ran Canetti, Dror Eiger, Shafi Goldwasser, and Dah-Yoh Lim. How to protect yourself without perfect shredding. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 511–523. Springer, Heidelberg, July 2008.
27. Nishanth Chandran, Bhavana Kanukurthi, and Srinivasan Raghuraman. Information-theoretic local non-malleable codes and their applications. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 367–392. Springer, Heidelberg, January 2016.
28. Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 285–298. ACM Press, June 2016.
29. Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *55th FOCS*, pages 306–315. IEEE Computer Society Press, October 2014.
30. Binyi Chen, Yilei Chen, Kristina Hostáková, and Pratyay Mukherjee. Continuous space-bounded non-malleable codes from stronger proofs-of-space. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 467–495. Springer, Heidelberg, August 2019.
31. Yu Chen, Baodong Qin, Jiang Zhang, Yi Deng, and Sherman S. M. Chow. Non-malleable functions and their applications. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 386–416. Springer, Heidelberg, March 2016.
32. Sandro Coretti, Yevgeniy Dodis, Björn Tackmann, and Daniele Venturi. Non-malleable encryption: Simpler, shorter, stronger. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 306–335. Springer, Heidelberg, January 2016.
33. Sandro Coretti, Antonio Faonio, and Daniele Venturi. Rate-optimizing compilers for continuously non-malleable codes. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 3–23. Springer, Heidelberg, June 2019.

34. Sandro Coretti, Ueli Maurer, Björn Tackmann, and Daniele Venturi. From single-bit to multi-bit public-key encryption via non-malleable codes. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 532–560. Springer, Heidelberg, March 2015.
35. Dana Dachman-Soled, Ilan Komargodski, and Rafael Pass. Non-malleable codes for bounded parallel-time tampering. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 535–565, Virtual Event, August 2021. Springer, Heidelberg.
36. Dana Dachman-Soled and Mukul Kulkarni. Upper and lower bounds for continuous non-malleable codes. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 519–548. Springer, Heidelberg, April 2019.
37. Dana Dachman-Soled, Mukul Kulkarni, and Aria Shahverdi. Tight upper and lower bounds for leakage-resilient, locally decodable and updatable non-malleable codes. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 310–332. Springer, Heidelberg, March 2017.
38. Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 427–450. Springer, Heidelberg, March 2015.
39. Ivan Damgård, Tomasz Kazana, Maciej Obremski, Varun Raj, and Luisa Siniscalchi. Continuous NMC secure against permutations and overwrites, with applications to CCA secure commitments. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 225–254. Springer, Heidelberg, November 2018.
40. Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 434–452. Tsinghua University Press, January 2010.
41. Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive, Report 2020/779, 2020. https://eprint.iacr.org/2020/779.
42. Sebastian Faust, Kristina Hostáková, Pratyay Mukherjee, and Daniele Venturi. Non-malleable codes for space-bounded tampering. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 95–126. Springer, Heidelberg, August 2017.
43. Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 465–488. Springer, Heidelberg, February 2014.
44. Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 111–128. Springer, Heidelberg, May 2014.
45. Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 258–277. Springer, Heidelberg, February 2004.
46. Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, October 2015.
47. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

48. Divya Gupta, Hemanta K. Maji, and Mingyuan Wang. Explicit rate-1 non-malleable codes for local tampering. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 435–466. Springer, Heidelberg, August 2019.
49. Zahra Jafargholi and Daniel Wichs. Tamper detection and continuous non-malleable codes. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 451–480. Springer, Heidelberg, March 2015.
50. Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Four-state non-malleable codes with explicit constant rate. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 344–375. Springer, Heidelberg, November 2017.
51. Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Non-malleable randomness encoders and their applications. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 589–617. Springer, Heidelberg, April / May 2018.
52. Lars R. Knudsen. Cryptanalysis of LOKI91. In Jennifer Seberry and Yuliang Zheng, editors, *AUSCRYPT'92*, volume 718 of *LNCS*, pages 196–208. Springer, Heidelberg, December 1993.
53. Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 1144–1156. ACM Press, June 2017.
54. Xin Li. Non-malleable extractors and non-malleable codes: Partially optimal constructions. Cryptology ePrint Archive, Report 2018/353, 2018. https://eprint.iacr.org/2018/353.
55. Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 517–532. Springer, Heidelberg, August 2012.
56. Rafail Ostrovsky, Giuseppe Persiano, Daniele Venturi, and Ivan Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 608–639. Springer, Heidelberg, August 2018.
57. Baodong Qin, Shengli Liu, Tsz Hon Yuen, Robert H. Deng, and Kefei Chen. Continuous non-malleable key derivation and its application to related-key security. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 557–578. Springer, Heidelberg, March / April 2015.
58. Hoeteck Wee. Public key encryption against related key attacks. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 262–279. Springer, Heidelberg, May 2012.