# Rate-Optimizing Compilers for Continuously Non-Malleable Codes

Sandro Coretti[1], Antonio Faonio[2], and Daniele Venturi[3]

[1]*New York University, USA*
[2]*IMDEA Software Institute, Spain*
[3]*Sapienza University of Rome, Italy*

January 29, 2019

**Abstract**

We study the *rate* of so-called *continuously* non-malleable codes, which allow to encode a message in such a way that (possibly adaptive) continuous tampering attacks on the codeword yield a decoded value that is unrelated to the original message. Our results are as follows:

- For the case of bit-wise independent tampering, we establish the existence of rate-one continuously non-malleable codes with information-theoretic security, in the plain model.
- For the case of split-state tampering, we establish the existence of rate-one continuously non-malleable codes with computational security, in the (non-programmable) random oracle model. We further exhibit a rate-1/2 code and a rate-one code in the common reference string model, but the latter only withstands *non-adaptive* tampering. It is well known that computational security is inherent for achieving continuous non-malleability in the split-state model (even in the presence of non-adaptive tampering).

Continuously non-malleable codes are useful for protecting *arbitrary* cryptographic primitives against related-key attacks, as well as for constructing non-malleable public-key encryption schemes. Our results directly improve the efficiency of these applications.

# Contents



# 1 Introduction

## 1.1 Background

The beautiful concept of non-malleable codes [28] has recently emerged at the intersection between cryptography and information theory. Given a function family $\mathcal{F}$, such codes allow to encode a $k$-bit value $s$ into an $n$-bit codeword $c$, such that, for each $f \in \mathcal{F}$, it is unlikely that $f(c)$ encodes a value $\tilde{s}$ that is related to $s$. On the theoretical side, being a weaker guarantee than error correction/detection, non-malleability is achievable for very rich families $\mathcal{F}$; on the practical side, non-malleable codes have interesting applications to cryptography.

**Continuous non-malleability.** In the original definition of non-malleable codes, the property of non-malleability is guaranteed as long as a *single*, possibly adversarial, function $f \in \mathcal{F}$ is applied to a target codeword. All bets are off, instead, if an adversary can tamper *multiple* times with the *same* codeword. While "one-time" non-malleability is already sufficient in some cases, it comes with some shortcomings, among which, for instance, the fact that in applications, after a decoding takes place, we always need to re-encode the message using fresh randomness; the latter might be problematic, as such a re-encoding procedure needs to take place in a tamper-proof environment.

Motivated by these limitations, Faust *et al.* [33] introduced a natural extension of non-malleable codes where the adversary is allowed to tamper a target codeword by specifying polynomially-many functions $f_j \in \mathcal{F}$; in case the functions can be chosen adaptively, depending

on the outcome of previous queries, we speak of adaptive tampering, and otherwise we say that tampering is non-adaptive. As argued in [33], such *continuously* non-malleable codes allow to overcome several limitations of one-time non-malleable codes, and further yield new applications where continuous non-malleability is essential [20, 21, 33, 34].

**Bit-wise and split-state tampering.** Since non-malleable codes do not involve secret keys, it is impossible to achieve (even one-time) non-malleability against all efficient families of functions $\mathcal{F}$. (In fact, whenever the encoding and decoding algorithms belong to $\mathcal{F}$, it is always possible to decode the target codeword, obtain the message, and encode a related value.) For this reason, research on non-malleable codes has focused on obtaining (continuous) non-malleability for limited, yet interesting, particular families. Two prominent examples, which are also the focus of this work, are described below:

- **Bit-wise independent tampering:** Here, each function $f \in \mathcal{F}^n_{\mathsf{bit}}$ is specified as a tuple $f := (f_1, \ldots, f_n)$, where each $f_i$ is an arbitrary map determining whether the $i$-th bit of the codeword should be kept, flipped, set to zero, or set to one. Continuously non-malleable codes for bit-wise independent tampering, with information-theoretic security, exist in the plain model [21] (i.e., without assuming a trusted setup).
- **Split-state tampering:** Here, each function $f \in \mathcal{F}^{n_0,n_1}_{\mathsf{split}}$ is specified as a pair $f := (f_0, f_1)$, where $n = n_0 + n_1$, and $f_0$ and $f_1$ are arbitrary functions to be applied, respectively, to the first $n_0$ bits and to the last $n_1$ bits of the codeword. Continuously non-malleable codes for split-state tampering, with computational security, were constructed in the common reference string (CRS) model [30, 33] (i.e., assuming a trusted setup), and very recently in the plain model [49] (assuming injective one-way functions).

It is well known that continuous non-malleability is impossible in the split-state model with information-theoretic security, even for non-adaptive tampering [33]. Furthermore, non-adaptive continuous non-malleability for both the above families requires a special "self-destruct" capability that instructs the decoding algorithm to always output the symbol $\perp$ (meaning "decoding error") after the first invalid codeword is decoded, otherwise generic attacks are possible [33, 36].

An important parameter of non-malleable codes is their *rate*, defined as the asymptotic ratio between the length of the message to the length of its encoding, as the message length goes to infinity. The optimal rate is one, whereas a code has rate zero if the length of the codeword is super-linear in the length of the message. Non-malleable codes with optimal rate for bit-wise independent tampering [8] (with information-theoretic security) and split-state tampering [1] (with computational security), were recently constructed. To the best of our knowledge, however, the achievable rate for *continuously* non-malleable codes for the same families is poorly understood.

## 1.2 Our Contributions

In this paper, we make significant progress towards characterizing the achievable rate for continuously non-malleable codes in the bit-wise independent and split-state tampering model.

**Split-state tampering.** In §3, we give three constructions of continuously non-malleable codes in the split-state model, with a natural trade-off in terms of efficiency, security, and assumptions. In particular, we show:

**Theorem 1** (Informal)**.** *There exists a continuously non-malleable code in the split-state model in the following settings:*

*(i)* *With rate* 1 *and with security against* non-adaptive *tampering in the common reference string model, assuming collision-resistant hash functions and non-interactive zero-knowledge proofs.*

*(ii)* *With rate* 1/2 *and with computational security against* adaptive *tampering in the common reference string model, assuming collision-resistant hash functions and non-interactive zero-knowledge proofs.*

*(iii)* *With rate* 1 *and with computational security against* adaptive *tampering in the non-programmable random oracle model.*

Recall that computational security is inherent for continuous non-adaptive non-malleability in the split-state model, even in the random oracle model.

**Bit-wise independent tampering.** In §4, we show a similar result for the case of bit-wise independent tampering, unconditionally:

**Theorem 2** (Informal)**.** *There exists a rate-one continuously non-malleable code against bit-wise independent tampering, achieving information-theoretic security against adaptive tampering in the plain model.*

From a technical perspective, the above theorems are proved by exhibiting so-called rate compilers. A rate compiler is a black-box transformation from a rate-zero non-malleable code $\Sigma$ for some family $\mathcal{F}$ into a non-malleable code $\Sigma'$ for the same family and with improved rate. In fact, we show that the rate compilers constructed in [1,8] already work, with some tweaks, in the continuous case. We stress, however, that while the constructions we analyze are similar to previous work, our security proofs differ significantly from the non-continuous case, and require several new ideas. We refer the reader directly to §3 and §4 for an overview of the main technical challenges we had to overcome.

### 1.3 Related Work

Several constructions of non-malleable codes for bit-wise [4, 7, 8, 20, 21, 28] and split-state [1–3, 5, 6, 14, 18, 22, 27–30, 33, 45, 45, 46, 49] tampering appear in the literature; out of those, only a few achieve continuous non-malleability [6, 20, 21, 30, 33, 42, 49].[1] Non-malleable codes also exist for a plethora of alternative models, including bit-wise tampering composed with permutations [7, 8, 18], circuits of polynomial size [17, 28, 35], constant-state tampering [4, 16, 43], block-wise tampering [13], functions with few fixed points and high entropy [42], space-bounded algorithms [10, 32], and bounded-depth circuits [9, 15].

The capacity (i.e., the best achievable rate) of information-theoretic non-malleable coding was first studied by Cheraghchi and Guruswami [17], who established that $1-\alpha$ is the maximum rate for function families which are only allowed to tamper the first $\alpha n$ bits of the codeword. This translates into a lower bound of 1/2 for the case of split-state tampering, and we also know that computational assumptions, in particular one-way functions, are necessary to go beyond the 1/2 barrier [1].

Non-malleable codes find applications to cryptography, in particular for protecting arbitrary cryptographic primitives against related-key attacks [28]. In this context, continuous non-malleability is a plus [33, 34]. Additional applications include constructions of non-malleable commitments [38], interactive proof systems [37], and domain extenders for public-key non-malleable encryption [20, 21, 47] and commitments [7].

[1] Strictly speaking, [6] only achieves continuous non-malleability for the weaker case of persistent tampering (where each tampering function is applied to the output of the previous tampering function).

# 2 Preliminaries

## 2.1 Notation

For a string $x$, we denote respectively its length by $|x|$ and the $i$-th bit by $x_i$; 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 $\mathsf{A}$ is an algorithm, we write $y \leftarrow\!\!\$\ \mathsf{A}(x)$ to denote a run of $\mathsf{A}$ on input $x$ and output $y$; if $\mathsf{A}$ is randomized, then $y$ is a random variable and $\mathsf{A}(x;r)$ denotes a run of $\mathsf{A}$ on input $x$ and randomness $r$. An algorithm $\mathsf{A}$ is *probabilistic polynomial-time* (PPT) if $\mathsf{A}$ is randomized and for any input $x, r \in \{0,1\}^*$ the computation of $\mathsf{A}(x;r)$ terminates in a polynomial number of steps (in the size of the input). Given two strings $x, y \in \{0,1\}^n$, we define the Hamming distance $\Delta(x,y) := \sum_{i\in[n]}(x_i + y_i \mod 2)$, where the sum is over the integers.

**Negligible functions.** We denote with $\lambda \in \mathbb{N}$ the security parameter. A function $\nu : \mathbb{N} \to [0,1]$ is negligible in the security parameter (or simply negligible) if it vanishes faster than the inverse of any polynomial in $\lambda$. We sometimes write $\nu(\lambda) \in \mathtt{negl}(\lambda)$ to denote that $\nu(\lambda)$ is negligible.

**Random variables.** For a random variable $\mathbf{X}$, we write $\mathbb{P}[\mathbf{X} = x]$ for the probability that $\mathbf{X}$ takes on a particular value $x \in \mathcal{X}$ (with $\mathcal{X}$ being the set where $\mathbf{X}$ is defined). The statistical distance between two random variables $\mathbf{X}$ and $\mathbf{X}'$ defined over the same set $\mathcal{X}$ is defined as $\mathbb{SD}(\mathbf{X};\mathbf{X}') = \frac{1}{2}\sum_{x\in\mathcal{X}} |\mathbb{P}[\mathbf{X} = x] - \mathbb{P}[\mathbf{X}' = x]|$.

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 they are identically distributed, $\mathbf{X} \approx_s \mathbf{Y}$ to denote that they are statistically close, i.e. $\mathbb{SD}(\mathbf{X}_\lambda;\mathbf{X}'_\lambda) \in \mathtt{negl}(\lambda)$, and $\mathbf{X} \approx_c \mathbf{Y}$ to denote that they are computationally indistinguishable, i.e., for all PPT distinguishers $\mathsf{D}$:

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

## 2.2 Non-Malleable Codes

We start by recalling the standard notion of a coding scheme in the common reference string (CRS) model.[2]

**Definition 1** (Coding scheme)**.** Let $k(\lambda) = k \in \mathbb{N}$ and $n(\lambda) = n \in \mathbb{N}$ be functions of the security parameter $\lambda \in \mathbb{N}$. A $(k,n)$-code is a tuple of algorithms $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ specified as follows: (1) The randomized algorithm $\mathsf{Init}$ takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a CRS $\omega \in \{0,1\}^{p(\lambda)}$, where $p(\lambda) \in \mathtt{poly}(\lambda)$; (2) The randomized algorithm $\mathsf{Enc}$ takes as input a value $s \in \{0,1\}^k$, and outputs a codeword $c \in \{0,1\}^n$; (3) The deterministic decoding algorithm $\mathsf{Dec}$ takes as input a codeword $c \in \{0,1\}^n$, and outputs a value $s \in \{0,1\}^k \cup \{\perp\}$ (where $\perp$ denotes an invalid codeword).

We say that $\Sigma$ satisfies correctness if for all $\omega \in \{0,1\}^{p(\lambda)}$ output by $\mathsf{Init}(1^\lambda)$, and for all values $s \in \{0,1\}^k$ the following holds: $\mathbb{P}[\mathsf{Dec}(\omega, \mathsf{Enc}(\omega, s)) = s] = 1$.

An important parameter of a coding scheme is its *rate*, i.e. the asymptotic ratio of the length of a message to the length of its encoding (in bits), as the message length increases to infinity. More formally, $\rho(\Sigma) := \inf_{\lambda\in\mathbb{N}} \lim_{k\to\infty} \frac{k(\lambda)}{n(\lambda)}$. The best rate possible is 1; if the length of the encoding is super-linear in the length of the message, the rate is 0.

[2]Such codes are sometimes also called non-explicit. Explicit codes are obtained by enforcing algorithm $\mathsf{Init}$ to output the empty string.

$\underline{\mathbf{Real}_{\Sigma,\mathsf{A},\mathcal{F}}(\lambda)}$:
$\omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$
$(s,\alpha_0) \leftarrow\!\!\$\ \mathsf{A}_0(\omega)$
$c \leftarrow\!\!\$\ \mathsf{Enc}(\omega, s)$
$\alpha_1 \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}_{\mathsf{maul}}(\omega,c,\cdot)}(\alpha_0)$
Return $\alpha_1$

$\underline{\text{Oracle } \mathcal{O}_{\mathsf{maul}}(\omega, c, \cdot)}$:
Upon $f \in \mathcal{F}$:
$\tilde{c} = f(c)$
$\tilde{s} = \mathsf{Dec}(\omega, \tilde{c})$
If $\tilde{s} = \perp$, self-destruct
Return $\tilde{s}$

$\underline{\mathbf{Simu}_{\mathsf{S},\mathsf{A},\mathcal{F}}(\lambda)}$:
$(\omega,\sigma) \leftarrow\!\!\$\ \mathsf{S}_0(1^\lambda)$
$(s,\alpha_0) \leftarrow\!\!\$\ \mathsf{A}_0(\omega)$
$\alpha_1 \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}_{\mathsf{sim}}(\mathsf{S}_1,\sigma,s,\cdot)}(\alpha_0)$
Return $\alpha_1$

$\underline{\text{Oracle } \mathcal{O}_{\mathsf{sim}}(\mathsf{S}_1, \sigma, s, \cdot)}$:
Upon $f \in \mathcal{F}$:
$\tilde{s} \leftarrow\!\!\$\ \mathsf{S}_1(\sigma, f)$
If $(\tilde{s} = \diamond)$, then $\tilde{s} \leftarrow s$
If $\tilde{s} = \perp$, self-destruct
Return $\tilde{s}$

**Figure 1:** Experiments defining continuously non-malleable codes. The self-destruct command causes the tamper oracles $\mathcal{O}_{\mathsf{maul}}$ and $\mathcal{O}_{\mathsf{sim}}$ to return $\perp$ on all subsequent queries.

**Non-malleability.** Let $\mathcal{F}$ be a family of functions $\mathcal{F} := \{f : \{0,1\}^n \to \{0,1\}^n\}$. The notion of $\mathcal{F}$-non-malleability [28] captures the intuition that any modification of a given target encoding via functions $f \in \mathcal{F}$ yields a codeword that either decodes to the same message as the original codeword, or to a completely unrelated value.

The definition below formalizes the above intuition in a more general setting where non-malleability is required to hold against (fully adaptive) adversaries that can maul the original encoding several times. This is often referred to as *continuous* non-malleability [33]. Roughly speaking, security is defined by comparing two experiments (cf. Fig. 1). In the "real experiment", the adversary tampers continuously with a target encoding of a chosen message (possibly dependent on the CRS);[3] for each tampering attempt, represented by a function $f \in \mathcal{F}$, the adversary learns the outcome corresponding to the decoding of the modified codeword. In the "simulated experiment", the view of the adversary is faked by a simulator which is completely oblivious of the message being encoded; importantly, the simulator is allowed to return a special symbol $\diamond$ meaning that (it believes) the tampering function yields a modified codeword which decodes to the original message. Both experiments self-destruct upon the first occurrence of $\perp$, i.e., they answer all subsequent queries by $\perp$.

**Definition 2** (Continuous non-malleability)**.** Let $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a $(k,n)$-code in the CRS model. We say that $\Sigma$ is *continuously* $\mathcal{F}$-non-malleable if for all PPT adversaries $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1)$ there exists a simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$ such that

$$\{\mathbf{Real}_{\Sigma,\mathsf{A},\mathcal{F}}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_{\mathrm{c}} \{\mathbf{Simu}_{\mathsf{S},\mathsf{A},\mathcal{F}}(\lambda)\}_{\lambda\in\mathbb{N}},$$

where the experiments $\mathbf{Real}_{\Sigma,\mathsf{A},\mathcal{F}}(\lambda)$ and $\mathbf{Simu}_{\mathsf{S},\mathsf{A},\mathcal{F}}(\lambda)$ are defined in Fig. 1.

**Remark 1** (Non-adaptive tampering)**.** *We model non-adaptive tampering by allowing the adversary* $\mathsf{A}_1$ *to submit a* single *query* $(f_j)_{j\in[q]}$ *to the oracle* $\mathcal{O}_{\mathsf{maul}}$*, for some polynomial* $q(\lambda) \in \mathtt{poly}(\lambda)$*. Upon input such a query, the oracle computes* $\tilde{c}_j = f_j(c)$*, and returns* $\tilde{s}_j = \mathsf{Dec}(\omega, \tilde{c}_j)$ *for all* $j \in [q]$ *(up to self-destruct). In this case, we say that* $\Sigma$ *is* non-adaptively *continuously* $\mathcal{F}$*-non-malleable.*

[3] Importantly, each tampering function is applied to the original coding; this setting is sometimes known as *non-persistent* tampering.

| $\mathbf{G}^{\mathsf{ind}}_{\Pi,\mathsf{A}}(\lambda, b)$: | $\mathbf{G}^{\mathsf{auth}}_{\Pi,\mathsf{A}}(\lambda)$: |
|---|---|
| $\kappa \leftarrow\!\!\$\ \{0,1\}^d$ | $\kappa \leftarrow\!\!\$\ \{0,1\}^d$ |
| $(\mu_0, \mu_1, \alpha) \leftarrow\!\!\$\ \mathsf{A}_0(1^\lambda)$ | $(\mu, \alpha) \leftarrow\!\!\$\ \mathsf{A}_0(1^\lambda)$ |
| $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, \mu_b)$ | $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, \mu)$ |
| Return $\mathsf{A}_1(\gamma, \alpha)$ | $\gamma' \leftarrow\!\!\$\ \mathsf{A}_1(\gamma, \alpha)$ |
| | Return 1 iff: |
| | (i) $\gamma' \neq \gamma$; and |
| | (ii) $\mathsf{ADec}(\kappa, \gamma') \neq \perp$. |

**Figure 2:** Experiments defining security of SKE.

**Tampering families.** We are particularly interested in the following tampering families.

- **Split-state tampering:** This is the family of functions $\mathcal{F}^{n_0,n_1}_{\mathsf{split}} := \{(f_0, f_1) : f_0 : \{0,1\}^{n_0} \to \{0,1\}^{n_0}, f_1 : \{0,1\}^{n_1} \to \{0,1\}^{n_1}\}$, for some fixed $n_0(\lambda) = n_0 \in \mathbb{N}$ and $n_1(\lambda) = n_1 \in \mathbb{N}$ such that $n_0 + n_1 = n$. Given an input codeword $c = (c_0, c_1)$, tampering with a function $(f_0, f_1) \in \mathcal{F}^{n_0,n_1}_{\mathsf{split}}$ results in a modified codeword $\tilde{c} = (f_0(c_0), f_1(c_1))$, where $c_0$ (resp., $c_1$) consists of the first $n_0$ (resp., the last $n_1$) bits of $c$.
- **Bit-wise independent tampering:** This is the family of functions $\mathcal{F}^n_{\mathsf{bit}} := \{(f_1, \ldots, f_n) : \forall i \in [n], f_i : \{0,1\} \to \{0,1\}\}$. Given an input codeword $c = (c_1, \ldots, c_n)$, tampering with a function $f \in \mathcal{F}^n_{\mathsf{bit}}$ results in a modified codeword $\tilde{c} = (f_1(c_1), \ldots, f_n(c_n))$, where each $f_i$ is any of the following functions: (i) $f_i(x) = x$ (`keep`); (ii) $f_i(x) = 1 \oplus x$ (`flip`); (iii) $f_i(x) = 0$ (`zero`); (iv) $f_i(x) = 1$ (`one`).

## 2.3 Authenticated Encryption

A secret-key encryption (SKE) scheme is a tuple of algorithms $\Pi := (\mathsf{KGen}, \mathsf{AEnc}, \mathsf{ADec})$ specified as follows: (1) The randomized algorithm $\mathsf{KGen}$ takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a uniform key $\kappa \leftarrow\!\!\$\ \{0,1\}^d$; (2) The randomized algorithm $\mathsf{AEnc}$ takes as input a key $\kappa \in \{0,1\}^d$, a message $\mu \in \{0,1\}^k$, and outputs a ciphertext $\gamma \in \{0,1\}^m$; (3) The deterministic algorithm $\mathsf{ADec}$ takes as input a key $\kappa \in \{0,1\}^d$, a ciphertext $\gamma \in \{0,1\}^m$, and outputs a value $\mu \in \{0,1\}^k \cup \{\perp\}$ (where $\perp$ denotes an invalid ciphertext). The values $d(\lambda), k(\lambda), m(\lambda)$ are all polynomials in the security parameter $\lambda \in \mathbb{N}$, and sometimes we call $\Pi$ a $(d, k, m)$-SKE scheme.

We say that $\Pi$ meets correctness if for all $\kappa \in \{0,1\}^d$, and all messages $\mu \in \{0,1\}^k$, we have that $\mathbb{P}[\mathsf{ADec}(\kappa, \mathsf{AEnc}(\kappa, \mu)) = \mu] = 1$ (over the randomness of $\mathsf{AEnc}$). As for security, an authenticated SKE scheme should satisfy two properties (see below for formal definitions). The first property, usually known as *semantic security*, says that it is hard to distinguish the encryptions of any two (adversarially chosen) messages. The second property, usually called *authenticity*, says that, without knowing the secret key, it is hard to produce a valid ciphertext (i.e., a ciphertext that does not decrypt to $\perp$).

**Definition 3** (Security of SKE)**.** We say that $\Pi = (\mathsf{KGen}, \mathsf{AEnc}, \mathsf{ADec})$ is a secure authenticated SKE scheme if the following holds for the games of Fig. 2:

- For all PPT adversaries $\mathsf{A}$, we have $\mathbb{P}\left[\mathbf{G}^{\mathsf{auth}}_{\Pi,\mathsf{A}}(\lambda) = 1\right] \in \mathtt{negl}(\lambda)$;
- $\left\{\mathbf{G}^{\mathsf{ind}}_{\Pi,\mathsf{A}}(\lambda, 0)\right\}_{\lambda\in\mathbb{N}} \approx_{\mathrm{c}} \left\{\mathbf{G}^{\mathsf{ind}}_{\Pi,\mathsf{A}}(\lambda, 1)\right\}_{\lambda\in\mathbb{N}}$.

Note that since both authenticity and semantic security are one-time properties, in principle, information-theoretic constructions with such properties are possible when $d \leq k$. However, we

are interested in constructions where $k > d$, for which the existence of one-way functions is a necessary assumption.

### 2.4 Concentration Bound

**Lemma 1** ( [19]). *Let $\delta \in (0, \frac{1}{2})$ be a constant, $m, n \in \mathbb{N}$ and $m \in [\delta n, (1-\delta)n]$. Let $\mathbf{X}$ be a random variable distributed uniformly over all $n$-bit strings with exactly $m$ 1s. Then, for every $\mathcal{T} \subseteq [n]$ of size $t$, and any $\varepsilon > 0$,*

$$\mathbb{P}\left[\sum_{i \in \mathcal{T}} \mathbf{X}_i \neq t\left(\tfrac{m}{n} \pm \varepsilon\right)\right] \;\leq\; 2e^{-\varepsilon^2 t/3}\,.$$

### 2.5 Error-Correcting Sharing Schemes

Intuitively, an error-correcting sharing scheme is an error-correcting code satisfying some form of privacy.

**Definition 4** (Error-correcting sharing scheme)**.** A $(k, n, T, D)$ *error correcting sharing scheme (ECSS)* is a triple of algorithms $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{ECorr})$, where $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ is probabilistic, $\mathsf{Dec} : \{0,1\}^n \to \{0,1\}^k$, and $\mathsf{ECorr} : \{0,1\}^n \to \{0,1\}^n \cup \{\perp\}$, with the following properties:

- **Correctness:** For all $s \in \{0,1\}^k$, $\mathsf{Dec}(\mathsf{Enc}(s)) = 1$ with probability 1 (over the randomness of $\mathsf{Enc}$).
- **Privacy:** For all $s \in \{0,1\}^k$, any subset of up to $T$ bits of $\mathsf{Enc}(s)$ are distributed uniformly and independently (over the randomness of $\mathsf{Enc}$).
- **Distance:** Any two codewords in the range of $\mathsf{Enc}$ have Hamming distance at least $D$.
- **Error correction:** For any codeword $c$ in the range of $\mathsf{Enc}$ and any $\tilde{c} \in \{0,1\}^n$, $\mathsf{ECorr}(\tilde{c}) = c$ if their Hamming distance is less than $D/2$, and $\mathsf{ECorr}(\tilde{c}) = \perp$ otherwise.

## 3 Split-State Tampering

In this section, we study several rate-optimizing compilers for continuously non-malleable codes in the split-state setting. As a starting point, in §3.1, we prove that, under certain assumptions on the initial rate-zero code, the compiler of Aggarwal *et al.* [1] actually achieves continuous security against *non-adaptive* tampering. Unfortunately, as we show via an explicit adaptive attack, the limitation of non-adaptive security is inherent for this particular construction.

Motivated by this limitation, we propose two variants of the rate compiler from [1] that guarantee continuous security in the presence of adaptive tampering attacks. The first variant, which is described in §3.2, achieves rate 1/2. The second variant, which is described in §3.3, achieves rate one in the (non-programmable) random oracle model.

### 3.1 Rate-One Compiler (Non-Adaptive Tampering)

Let $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a rate-zero $(d, n)$-code, and $\Pi = (\mathsf{KGen}, \mathsf{AEnc}, \mathsf{ADec})$ be a $(d, k, m)$-SKE scheme. Consider the following construction of a $(k, n')$-code $\Sigma' = (\mathsf{Init}', \mathsf{Enc}', \mathsf{Dec}')$, where $n' := m + n$.

$\mathsf{Init}'(1^\lambda)$**:** Upon input $\lambda \in \mathbb{N}$, return the same as $\mathsf{Init}(1^\lambda)$.

$\mathsf{Enc}'(\omega, s)$**:** Upon input $\omega$ and a value $s \in \{0,1\}^k$, sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, compute $c \leftarrow\!\!\$\ \mathsf{Enc}(\omega, \kappa)$ and $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, s)$; return $c' := c||\gamma$.

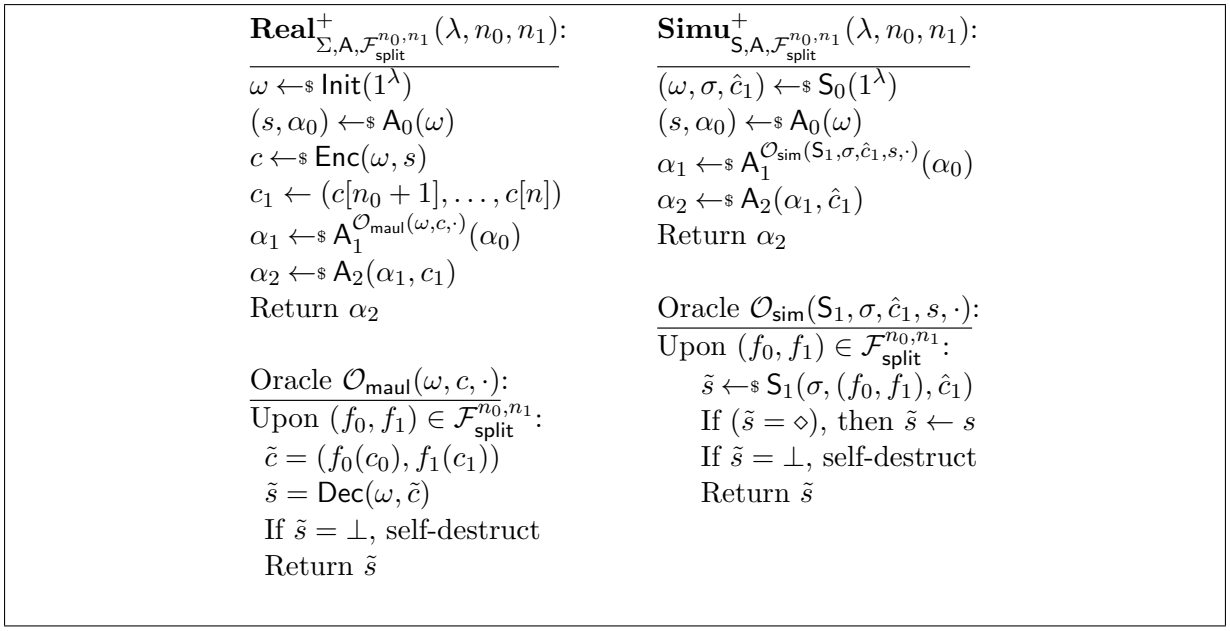


**Figure 3:** Experiments defining *augmented* continuously non-malleable codes.

$\mathsf{Dec}'(\omega,c')$**:** Parse $c' := c||\gamma$, and let $\tilde{\kappa}=\mathsf{Dec}(\omega,c)$. If $\tilde{\kappa}=\perp$, return $\perp$ and self-destruct; else let $\tilde{s}=\mathsf{ADec}(\tilde{\kappa},\gamma)$. If $\tilde{s}=\perp$, return $\perp$ and self-destruct; else return $\tilde{\mu}$.

Roughly speaking, the compiler uses the underlying (rate-zero) code to encode a uniform key for the authenticated encryption scheme; such a key is then used to encrypt the message, and the resulting ciphertext is appended to the encoding of the key. The decoding algorithm, naturally decodes the encoding of the key, and hence uses the resulting key to decrypt the ciphertext.

### 3.1.1 Augmented Continuous Non-Malleability

Assume that $\Sigma$ is non-malleable in the split-state setting, where the encoding $c$ is split in two halves $c_0$ and $c_1$ (consisting of $n_0$ and $n_1$ bits, respectively) that can be modified arbitrarily (yet independently). Intuitively, we would like to show that $\Sigma'$ is continuously non-malleable against the class of split-state functions that modifies $c'_0 := c_0$ and $c'_1 := (c_1,\gamma)$ independently.

The difficulty, originally observed in [1], is that, although $(c_0,c_1)$ is a non-malleable encoding of $\kappa$ (as long as $c_0$ and $c_1$ are mauled independently), the adversary could attempt to (independently) modify $c'_1$ and $c'_0$ yielding shares $\tilde{c}'_1 := (\tilde{c}_1,\tilde{\gamma})$ and $\tilde{c}'_0$ such that $(\tilde{c}_0,\tilde{c}_1)$ decodes to a key $\tilde{\kappa}$ which is unrelated to $\tilde{\kappa}$, yet decrypting $\tilde{\gamma}$ with $\tilde{\kappa}$ results in a message $\tilde{s}$ that is related to $s$.

A similar difficulty, of course, appears in the continuous setting. In order to overcome this obstacle, inspired by the approach taken in [1], we define a notion of *augmented* continuous non-malleability. Such a notion is a stronger form of continuous non-malleability where, in the "real experiment" after $\mathsf{A}$ is done with tampering queries, it is additionally given one share of the *original* encoding (say, $c_1$). In turn, the "ideal experiment" features a sort of "canonical" simulator $\mathsf{S}$ that at the beginning of the simulation computes an encoding $\hat{c} := (\hat{c}_0,\hat{c}_1)$ of, say, the all-zero string; hence, the dummy encoding $\hat{c}$ is used to answer tampering queries from $\mathsf{A}$, and, after the adversary is done with tampering queries, the simulator returns $\hat{c}_1$ to $\mathsf{A}$. The formal definition appears below.

**Definition 5** (Augmented continuous non-malleability)**.** Let $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a $(k, n)$-code in the CRS model, and let $n_0(\lambda) = n_0 \in \mathbb{N}$ and $n_1(\lambda) = n_1 \in \mathbb{N}$ be such that $n = n_0 + n_1$. We say that $\Sigma$ is *augmented* continuously $\mathcal{F}_{\mathsf{split}}^{n_0,n_1}$-non-malleable if for all PPT adversaries $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$ there exists a simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$ such that

$$\left\{\mathbf{Real}^{+}_{\Sigma,\mathsf{A},\mathcal{F}_{\mathsf{split}}^{n_0,n_1}}(\lambda, n_0, n_1)\right\}_{\lambda\in\mathbb{N}} \approx_{\mathrm{c}} \left\{\mathbf{Simu}^{+}_{\mathsf{S},\mathsf{A},\mathcal{F}_{\mathsf{split}}^{n_0,n_1}}(\lambda, n_0, n_1)\right\}_{\lambda\in\mathbb{N}},$$

where the experiments $\mathbf{Real}^{+}_{\Sigma,\mathsf{A},\mathcal{F}_{\mathsf{split}}^{n_0,n_1}}$ and $\mathbf{Simu}^{+}_{\mathsf{S},\mathsf{A},\mathcal{F}_{\mathsf{split}}^{n_0,n_1}}$ are defined in Fig. 3.

### 3.1.2 Security Analysis

We establish the following result.

**Theorem 3.** *Assume that $\Sigma$ is an* augmented *continuously $\mathcal{F}_{\mathsf{split}}^{n_0,n_1}$-non-malleable $(d, n)$-code, and that $\Pi$ is a* secure authenticated *$(d, k, m)$-SKE scheme. Then $\Sigma'$ as defined in §3.1 is a* non-adaptively *continuously $\mathcal{F}_{\mathsf{split}}^{n_0,n_1+m}$-non-malleable $(k, m + n)$-code.*

**Remark 2.** *Similarly to [1], the analysis actually shows that the code $\Sigma'$ also preserves* augmented *continuous non-malleability (and not just continuous non-malleability). However, since our goal is to construct continuously non-malleable codes (in the standard sense), we do not give the proof for the augmented case.*

*We also stress that it suffices to start from an augmented code $\Sigma'$ that is* non-adaptively *continuously non-malleable. However, we rely on the stronger assumption of full adaptivity in order to simplify the exposition, and because, looking ahead, our instantiation from §5.1 achieves this property.*

**Proof intuition.** We sketch the main ideas behind the security proof. We need to describe a simulator $\mathsf{S}'$ that can emulate arbitrary non-adaptive split-state tampering with a target encoding $c' := (c_0, (c_1, \gamma))$ of a message $s$, without knowing $s$. Roughly, $\mathsf{S}'$ does the following.

- At the beginning, run the simulator $\mathsf{S}_0^+$ of the underlying augmented non-malleable code, obtaining a fake CRS $\omega$ and a simulated right share $\hat{c}_1$.
- Sample a key $\kappa$ for the authenticated encryption scheme, and define $\gamma$ as an encryption of $0^k$ under the sampled key.
- Upon receiving a sequence of non-adaptive tampering queries $(f'_{0,j}, f'_{1,j})_{j\in[q]}$ behave as follows for each $j \in [q]$:
  - Invoke the simulator $\mathsf{S}_1^+$ of the underlying augmented non-malleable code upon $(f'_{0,j}, f'_{1,j}, \hat{c}_1)$, obtaining a simulated decoded key $\tilde{\kappa}_j \in \{\diamond, \perp\} \cup \{0,1\}^d$.
  - Compute the mauled ciphertext $\tilde{\gamma}_j$ by applying $f'_{1,j}$ on $(\hat{c}_1, \gamma)$.
- For each key $\tilde{\kappa}_j$:
  - If $\tilde{\kappa}_j = \perp$ set $\tilde{s}_j := \perp$.
  - Else if $\tilde{\kappa}_j = \diamond$, set $\tilde{s}_j := \perp$ in case $\tilde{\gamma}_j$ is different from the original ciphertext $\gamma$, and otherwise set $\tilde{s} := \diamond$.
  - Else set $\tilde{s}_j$ as the decryption of $\tilde{\gamma}_j$ under $\tilde{\kappa}_j$.
  - Simulate a self-destruct by taking the minimum index $j^*$ such that either $\tilde{\kappa}_{j^*} = \perp$ or $\tilde{s}_{j^*} = \perp$, and overwrite all values $\tilde{s}_{j^*+1}, \ldots, \tilde{s}_q$ with $\perp$.
- Return $\tilde{s}_1, \ldots, \tilde{s}_q$.

In order to prove that the above simulation is indeed correct, we define a sequence of hybrid experiments starting with the real experiment (where the adversary $\mathsf{A}'$ tampers non-adaptively with a target encoding computed using $\Sigma'$) and ending with the ideal experiment (where the above simulator is used to answer $\mathsf{A}'$'s tampering queries). In the first hybrid, we change the way a non-adaptive tampering query $(f'_{0,j}, f'_{1,j})_{j\in[q]}$ is answered. In particular, given each $(f'_{0,j}, f'_{1,j})$, we run the augmented simulator $\mathsf{S}_1^+$ upon $(f_{0,j}, f_{1,j})$, where $f_{0,j}$ is identical to $f'_{0,j}$, whereas $f_{1,j}$ is obtained by hard-wiring the ciphertext $\gamma$ (encrypting the real message $s$) into $f'_{1,j}$. This allows us to get a mauled key $\tilde{\kappa}_j$ that is then used to decrypt the ciphertext $\tilde{\gamma}_j$ defined by applying the function $f'_{1,j}$ on $(\hat{c}_1, \gamma)$, where $\hat{c}_1$ is the right share of an encoding produced at the beginning of the experiment by running the augmented simulator $\mathsf{S}_0^+$.

The most interesting part of the proof is to show that the real experiment and the above hybrid are computationally indistinguishable; here, the augmented non-malleability of the underlying code $\Sigma$ plays a crucial role. For the purpose of this proof sketch, we only focus on this particular step of the proof, and refer the reader to the full proof for the analysis of the other hybrids. The main challenge is to reduce the attacker $\mathsf{A}'$ against $\Sigma'$ to an attacker $\mathsf{A}$ against $\Sigma$. In fact, the attacker $\mathsf{A}'$ expects to attack a target encoding of the form $(c_0, (c_1, \gamma))$, whereas the attacker $\mathsf{A}$ can only tamper with $(c_0, c_1)$. This issue is resolved by having $\mathsf{A}$ encrypt the value $s$ chosen by $\mathsf{A}'$ under a uniformly random key $\kappa$ for the authenticated encryption, and by mapping each pair of tampering functions $(f'_{0,j}, f'_{1,j})$ into a pair $(f_{0,j}, f_{1,j})$ such that $f_{0,j} := f'_{0,j}$ and $f_{1,j}(\cdot) := f'_{1,j}(\cdot, \gamma)$ (i.e., the ciphertext $\gamma$ is hard-wired into the right tampering function).

The above trick allows the reduction to obtain a mauled key $\tilde{\kappa}_j \in \{\diamond, \perp\} \cup \{0,1\}^d$ that is either distributed as in the real experiment (where decoding takes place) or as in the hybrid experiment (where the augmented simulator $\mathsf{S}_1^+$ is used). Unfortunately, this information alone is not sufficient to complete the simulation; in fact, the reduction would need to use the key $\tilde{\kappa}_j$ to decrypt the mauled ciphertext $\tilde{\gamma}_j$ which is obtained by applying the function $f'_{1,j}$ upon input the ciphertext $\gamma$ and either the real share $c_1$ (in the real experiment) or the simulated share $\hat{c}_1$ (in the hybrid experiment). Now, if $\mathsf{A}'$ were fully adaptive, the reduction would get to know the right share of the encoding only after the last tampering query, which makes it difficult to complete the reduction. Here is where we rely on the fact that tampering is non adaptive, as in this case $\mathsf{A}'$ specifies all functions $(f'_{0,j}, f'_{1,j})_{j\in[q]}$ in one go, which in turn allows $\mathsf{A}$ to specify $(f_{0,j}, f_{1,j})_{j\in[q]}$ as defined above, obtain all values $(\tilde{\kappa}_j)_{j\in[q]}$ together with the right share (i.e., either $c_1$ or $\hat{c}_1$), compute the ciphertexts $(\tilde{\gamma}_j)_{j\in[q]}$, and finally complete the simulation.

*Proof of Theorem 3.* For simplicity, let us write $\mathcal{F} := \mathcal{F}_{\mathsf{split}}^{n_0,n_1}$ and $\mathcal{F}' := \mathcal{F}_{\mathsf{split}}^{n_0,n_1+m}$. We must describe a simulator $\mathsf{S}' = (\mathsf{S}'_0, \mathsf{S}'_1)$ such that for all adversaries $\mathsf{A}' = (\mathsf{A}'_0, \mathsf{A}'_1)$ we have:

$$\left\{\mathbf{Real}_{\Sigma',\mathsf{A}',\mathcal{F}'}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_{\mathrm{c}} \left\{\mathbf{Simu}_{\mathsf{S}',\mathsf{A}',\mathcal{F}'}(\lambda)\right\}_{\lambda\in\mathbb{N}},$$

where the real/ideal experiments are defined in Fig. 1.

The proof proceeds by introducing a sequence of hybrid experiments, formally described in Fig. 4, that ease the description of the simulator. The experiments are informally described below.

**Hybrid $\mathbf{H}_0(\lambda)$:** The first hybrid experiment is identical to $\mathbf{Real}_{\Sigma',\mathsf{A}',\mathcal{F}}(\lambda)$, where $\Sigma'$ is the transformed coding scheme described in §3.1.

**Hybrid $\mathbf{H}_1(\lambda)$:** We change the way tampering queries are handled. In particular, at the beginning of the experiment we sample the CRS by running the algorithm $\mathsf{S}_0^+$ of the underlying code $\Sigma$. Furthermore, the answer to each tampering query is now computed by first obtaining the tampered key $\tilde{\kappa}$ via algorithm $\mathsf{S}_1^+$, and then decrypting the mauled ciphertext $\tilde{\gamma}$. Here is where we rely on the fact that the simulator $\mathsf{S}^+$ is able to fake the distribution of the right share of an encoding, which allows to simulate the tampered ciphertext.

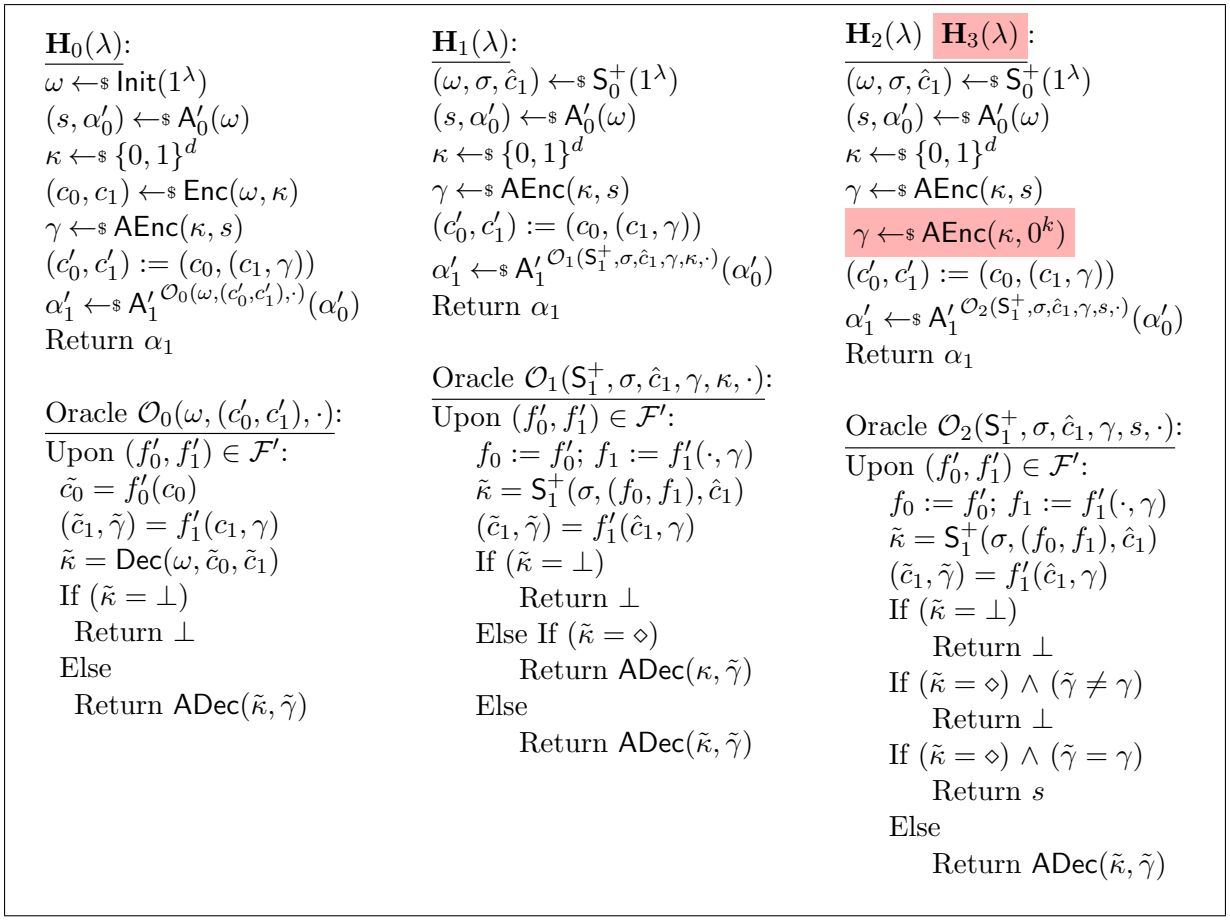


**Figure 4:** Hybrid experiments in the proof of Theorem 3.

**Hybrid $\mathbf{H}_2(\lambda)$:** We change the way tampering queries are handled. Namely, whenever the simulator $\mathsf{S}^+$ returns $\diamond$, we force the output of the tampering query to be either $\perp$ (in case $\tilde{\gamma} \neq \gamma$) or the original message $s$ (in case $\tilde{\gamma} = \gamma$).

**Hybrid $\mathbf{H}_3(\lambda)$:** We change the distribution of the target encoding. In particular, we compute the ciphertext $\gamma$ by encrypting the all-zero string $0^k$.

We emphasize that the above experiments are further parameterized by the adversary $\mathsf{A}'$ and the function family $\mathcal{F}$, but we omit writing those for simplifying notation.

**Hybrids' indistinguishability.** Next, we proceed to show indistinguishability of the above defined hybrids.

**Lemma 2.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{0,1}(\lambda) \in \mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_0(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda)) = 1\right]\right| \leq \nu_{0,1}(\lambda).$$

*Proof.* The proof is down to the augmented continuous non-malleability of the underlying code $\Sigma$. By contradiction, assume that there exists a PPT adversary $\mathsf{A}'$, a polynomial $p_{0,1}(\lambda) \in \mathtt{poly}(\lambda)$, and a PPT distinguisher $\mathsf{D}'$ such that for infinitely many values of $\lambda \in \mathbb{N}$:

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_0(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda)) = 1\right]\right| \geq 1/p_{0,1}(\lambda).$$

We construct a PPT adversary $\mathsf{A}$ and a PPT distinguisher $\mathsf{D}$ such that for infinitely many values of $\lambda \in \mathbb{N}$:

$$\left|\mathbb{P}\left[\mathsf{D}(\mathbf{Real}^{+}_{\Sigma,\mathsf{A},\mathcal{F}}(\lambda, n_0, n_1)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathbf{Simu}^{+}_{\mathsf{S}^{+},\mathsf{A},\mathcal{F}}(\lambda, n_0, n_1)) = 1\right]\right| \geq 1/p_{0,1}(\lambda),$$

where the experiments $\mathbf{Real}^{+}_{\Sigma,\mathsf{A},\mathcal{F}}$ and $\mathbf{Simu}^{+}_{\mathsf{S}^{+},\mathsf{A},\mathcal{F}}$ are depicted in Fig. 3, and with $\mathsf{S}^{+}$ being the (augmented) simulator guaranteed by Definition 5. Adversary $\mathsf{A}$ proceeds as follows:

- Upon receiving $\omega$, run $\mathsf{A}'_0(\omega)$ obtaining a pair $(s, \alpha'_0)$.
- Sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$ and compute $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, s)$.
- Run $\mathsf{A}_1(\alpha'_0)$, obtaining a sequence of tampering functions $f'_1, \ldots, f'_q$, where $q(\lambda) \in \mathtt{poly}(\lambda)$).
- For each $j \in [q]$, define $f_{0,j} = f'_{0,j}$ and $f_{1,j} = f'_{1,j}(\cdot, \gamma)$ and query $(f_{0,j}, f_{1,j})$ to the target tampering oracle, obtaining a response $\tilde{\kappa}_j$.
- Upon receiving the share $c_1$, output $\alpha_2 := (\alpha'_0, \kappa, \gamma, (\tilde{\kappa}_j)_{j\in[q]}, (f_{0,j}, f_{1,j})_{j\in[q]}, c_1)$.

Given the final state $\alpha_2$ as computed by $\mathsf{A}$, distinguisher $\mathsf{D}$ proceeds as follows:

- Parse $\alpha_2 := (\alpha'_0, \kappa, \gamma, (\tilde{\kappa}_j)_{j\in[q]}, (f_{0,j}, f_{1,j})_{j\in[q]}, c_1)$.
- Let $j^* \in [q]$ be the smallest index such that $\tilde{\kappa}_{j^*} = \perp$; set $\tilde{s}_{j^*}, \ldots, \tilde{s}_q := \perp$.
- For each $j \leq j^* - 1$:
  - If $\tilde{\kappa}_j = \diamond$, set $\tilde{s}_j := \mathsf{ADec}(\kappa, \tilde{\gamma}_j)$ where $(\tilde{c}_j, \tilde{\gamma}_j) := f_{1,j}(c_1, \gamma)$.
  - Else, set $\tilde{s}_j := \mathsf{ADec}(\tilde{\kappa}_j, \tilde{\gamma}_j)$ where $(\tilde{c}_j, \tilde{\gamma}_j) := f_{1,j}(c_1, \gamma)$.
  - In case $\tilde{s}_j = \perp$, define $\tilde{s}_{j+1} := \cdot := \tilde{s}_{j^*-1} := \perp$ and stop.
- Run $\mathsf{A}_1(\alpha'_0, \tilde{s}_1, \ldots, \tilde{s}_q)$, obtaining state information $\alpha'_1$, and return the same as $\mathsf{D}'(\alpha'_1)$.

We observe that depending on $\mathsf{A}$'s target oracle being either $\mathcal{O}_{\mathsf{maul}}$ or $\mathcal{O}_{\mathsf{sim}}$, the simulation of the answers corresponding to $\mathsf{A}'$'s tampering queries is identical to either that of $\mathbf{H}_0(\lambda)$ or to that of $\mathbf{H}_1(\lambda)$. Hence, $(\mathsf{A}, \mathsf{D})$ retain the same advantage as that of $(\mathsf{A}', \mathsf{D}')$. The lemma follows. □

**Lemma 3.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{1,2}(\lambda) \in \mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda)) = 1\right]\right| \leq \nu_{1,2}(\lambda).$$

*Proof.* Consider the following event $W$, defined over the probability space of $\mathbf{H}_1(\lambda)$: The event becomes true if during a run of the experiment adversary $\mathsf{A}'_1$ returns a sequence of tampering functions $(f'_{0,j}, f'_{1,j})_{j\in[q]}$ such that for some index $j^*$ we have $\tilde{\kappa}_{j^*} = \diamond$, $\tilde{\gamma}_{j^*} \neq \gamma$, where $(\tilde{c}_{1,j^*}, \tilde{\gamma}_{j^*}) = f'_{1,j^*}(c, \gamma)$, and $\mathsf{ADec}(\kappa, \tilde{\gamma}_{j^*}) \neq \perp$. Notice that conditioning on event $W$ not happening, the answer to $\mathsf{A}'_1$'s tampering queries is identical in $\mathbf{H}_1(\lambda)$ and $\mathbf{H}_2(\lambda)$, and thus it suffices to upper bound the probability of event $W$ happening.

Suppose that there exists a PPT adversary $\mathsf{A}' = (\mathsf{A}'_0, \mathsf{A}'_1)$ and a polynomial $p_{1,2}(\lambda) \in \mathtt{poly}(\lambda)$, such that for infinitely many values of $\lambda \in \mathbb{N}$, attacker $\mathsf{A}$ provokes event $W$ with probability at least $1/p_{1,2}(\lambda)$. We build a PPT adversary $\mathsf{A}$ that violates the authenticity property of the SKE scheme. A description of $\mathsf{A}$ follows:

- Sample $(\omega, \sigma, \hat{c}_0) \leftarrow\!\!\$\ \mathsf{S}^{+}_0(1^\lambda)$, and run $\mathsf{A}'_0(\omega)$ obtaining a pair $(s, \alpha'_0)$ which is given to the challenger.
- Run $\mathsf{A}'_1(\alpha'_0)$, obtaining a sequence of tampering functions $(f'_{0,j}, f'_{1,j})_{j\in[q]}$; for each $(f'_{0,j}, f'_{1,j})$, let $\tilde{\kappa}_j \leftarrow\!\!\$\ \mathsf{S}^{+}_1(\sigma, (f_{0,j}, f_{1,j}), \hat{c}_1)$ where $f_{0,j} := f'_{0,j}$ and $f_{1,j} := f'_{1,j}(\cdot, \gamma)$.
- Upon receiving the challenge ciphertext $\gamma$, pick a random $\hat{j} \leftarrow\!\!\$\ [q]$, compute $(\tilde{c}_1, \tilde{\gamma}) = f'_{1,\hat{j}}(\hat{c}_1, \gamma)$, and output $\tilde{\gamma}$.

The view of $(\mathsf{A}'_0, \mathsf{A}'_1)$ when run as a sub-routine of $\mathsf{A}$ is identical to that in experiment $\mathbf{H}_1(\lambda)$. It follows that $\mathsf{A}'_1$ will thus provoke event $W$ with probability at least $1/p_{1,2}(\lambda)$. Observe that, whenever $W$ happens, and as long as $\hat{j} = j^*$, adversary $\mathsf{A}$ wins the unforgeability game, because the ciphertext $\tilde{\gamma}$ is different from the challenge ciphertext $\gamma$, but $\mathsf{ADec}(\kappa, \tilde{\gamma}) \neq \perp$. Hence, the reduction succeeds with probability at least $1/q \cdot 1/p_{1,2}(\lambda)$, which is non-negligible. □

**Lemma 4.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{2,3}(\lambda) \in \mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_3(\lambda)) = 1\right]\right| \leq \nu_{2,3}(\lambda).$$

*Proof.* The proof is down to the semantic security of the authenticated encryption scheme. By contradiction, assume that there exists a PPT adversary $\mathsf{A}'$, a polynomial $p_{2,3}(\lambda)$, and a PPT distinguisher $\mathsf{D}'$ such that for infinitely many values of $\lambda \in \mathbb{N}$:

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_3(\lambda)) = 1\right]\right| \geq 1/p_{2,3}(\lambda).$$

We construct a PPT adversary $\mathsf{A}$, and a PPT distinguisher $\mathsf{D}$, such that for infinitely many values of $\lambda \in \mathbb{N}$:

$$\left|\mathbb{P}\left[\mathsf{D}(\mathbf{G}^{\mathsf{ind}}_{\Pi,\mathsf{A}}(\lambda, 0)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathbf{G}^{\mathsf{ind}}_{\Pi,\mathsf{A}}(\lambda, 1)) = 1\right]\right| \geq 1/p_{2,3}(\lambda).$$

A description of $\mathsf{A}$.

- Sample $(\omega, \sigma, \hat{c}_0) \leftarrow\!\!\$\ \mathsf{S}^+_0(1^\lambda)$, and run $\mathsf{A}'_0(\omega)$ obtaining a pair $(s, \alpha'_0)$.
- Forward $(\mu_0 := s, \mu_1 := 0^k)$ to the challenger, obtaining a challenge ciphertext $\gamma$.
- Run $\mathsf{A}'_1(\alpha'_0)$, obtaining a sequence of tampering functions $(f'_{0,j}, f'_{1,j})_{j\in[q]}$; for each $(f'_{0,j}, f'_{1,j})$, let $\tilde{\kappa}_j \leftarrow\!\!\$\ \mathsf{S}^+_1(\sigma, (f_{0,j}, f_{1,j}), \hat{c}_1)$ where $f_{0,j} := f'_{0,j}$ and $f_{1,j} := f'_{1,j}(\cdot, \gamma)$.
- Let $j^* \in [q]$ be the smallest index such that $\tilde{\kappa}_{j^*} = \perp$; set $\tilde{s}_{j^*}, \ldots, \tilde{s}_q := \perp$.
- For each $j \leq j^* - 1$ let $(\tilde{c}_j, \tilde{\gamma}_j) := f'_{1,j}(c_1, \gamma)$ and:
  - If $\tilde{\kappa}_j = \diamond$ and $\tilde{\gamma}_j \neq \gamma$, set $\tilde{s}_j := \perp$.
  - Else if $\tilde{\kappa}_j = \diamond$ and $\tilde{\gamma}_j = \gamma$, set $\tilde{s}_j := s$.
  - Else set $\tilde{s}_j := \mathsf{ADec}(\tilde{\kappa}_j, \tilde{\gamma}_j)$.
  - In case $\tilde{s}_j = \perp$, define $\tilde{s}_{j+1} := \cdot := \tilde{s}_{j^*-1} := \perp$ and stop.

Finally, $\mathsf{D}$ runs $\mathsf{A}_1(\alpha'_0, \tilde{s}_1, \ldots, \tilde{s}_q)$, obtaining state information $\alpha'_1$, and returns the same as $\mathsf{D}'(\alpha'_1)$.

We observe that depending on the challenge ciphertext being either an encryption of $s$ or an encryption of $0^k$, $\mathsf{A}$'simulation of $\mathsf{A}'$'s tampering queries is identical to either that of $\mathbf{H}_2(\lambda)$ or to that of $\mathbf{H}_3(\lambda)$. Hence, $\mathsf{D}$ retains the same advantage as that of $\mathsf{D}'$. The lemma follows. □

**Simulator's description.** We are now ready to describe the simulator $\mathsf{S}' = (\mathsf{S}'_0, \mathsf{S}'_1)$.

$\underline{\mathsf{S}'_0(1^\lambda)}$:

- Run $(\omega, \sigma, \hat{c}_0) \leftarrow\!\!\$\ \mathsf{S}_0^+(1^\lambda)$.
- Output $\sigma' := (\sigma, \hat{c}_0)$.

$\underline{\mathsf{S}'_1(\sigma', (f'_{0,j}, f'_{1,j})_{j\in[q]})}$:

- Parse $\sigma' := (\sigma, \hat{c}_0)$.
- Sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, and let $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, 0^k)$.
- For each $(f'_{0,j}, f'_{1,j})$, let $\tilde{\kappa}_j \leftarrow\!\!\$\ \mathsf{S}_1^+(\sigma, (f_{0,j}, f_{1,j}), \hat{c}_1)$ where $f_{0,j} := f'_{0,j}$ and $f_{1,j} := f'_{1,j}(\cdot, \gamma)$.
- Let $j^* \in [q]$ be the smallest index such that $\tilde{\kappa}_{j^*} = \bot$; set $\tilde{s}_{j^*}, \ldots, \tilde{s}_q := \bot$.
- For each $j \leq j^* - 1$:
  - If $\tilde{\kappa}_j = \diamond$ and $\tilde{\gamma}_j \neq \gamma$, set $\tilde{s}_j := \bot$.
  - Else if $\tilde{\kappa}_j = \diamond$ and $\tilde{\gamma}_j = \gamma$, set $\tilde{s}_j := \diamond$.
  - Else set $\tilde{s}_j := \mathsf{ADec}(\tilde{\kappa}_j, \tilde{\gamma}_j)$.
  - In case $\tilde{s}_j = \bot$, define $\tilde{s}_{j+1} := \cdots := \tilde{s}_{j^*-1} := \bot$ and stop.
- Return $\tilde{s}_1, \ldots, \tilde{s}_q$.

The theorem then follows by combining the above lemmas, and by observing that $\mathbf{H}_3(\lambda)$ is identically distributed to $\mathbf{Simu}_{\mathsf{S}',\mathsf{A}',\mathcal{F}'}(\lambda)$. □

### 3.1.3 An Adaptive Attack

We describe an *adaptive* attack against the above code construction. The attack makes $\lceil n'/k \rceil$ non-adaptive tampering queries followed by a final adaptive tampering query. Let $(c_0, c_1\|\gamma)$ be the target encoding of some message $s \in \{0,1\}^k$. Since the code has rate one, we can assume wlog. that the right part of the codeword $c_1\|\gamma$ can be parsed as two strings $\delta_1, \delta_2$ of size $k$. Let $\tilde{\kappa} \in \{0,1\}^\lambda$ be a key for the authenticated encryption scheme, and let $(\tilde{c}_0, \tilde{c}_1)$ be an encoding of $\tilde{\kappa}$ under the underlying rate-zero non-malleable code. The attack proceeds as described below:

1. Let $(f_{0,1}, f_{1,1})$ be such that $f_{0,1}(c_0)$ outputs $\tilde{c}_0$, whereas $f_{1,1}(c_1\|\gamma)$ outputs $\tilde{c}_1\|\mathsf{AEnc}(\tilde{\kappa}, \delta_1)$;
2. Let $(f_{0,2}, f_{1,2})$ be such that $f_{0,2}(c_0)$ outputs $\tilde{c}_0$, whereas $f_{1,2}(c_1\|\gamma)$ outputs $\tilde{c}_1\|\mathsf{AEnc}(\tilde{\kappa}, \delta_2)$.

Now, the attacker queries the tampering oracle with $(f_{0,1}, f_{1,1})$ and $(f_{0,2}, f_{1,2})$, which yields $\delta_1\|\delta_2 = c_1\|\gamma$. The next tampering query, which is adaptive, hard-codes the value $c_1\|\gamma$ into its description:

3. Let $(f_{0,3}, f_{1,3})$ be such that:
   - $f_{0,3}(c_0)$ computes $\mathsf{Dec}(c_0, c_1\|\gamma)$ and outputs $0^{n_0}$ if the first bit of the decoded message is 0, and else it outputs $\tilde{c}_0$;
   - $f_{1,3}(c_1\|\gamma)$ simply outputs $\tilde{c}_1\|\mathsf{AEnc}(\tilde{\kappa}, 0^k)$.

In case the first bit of the target message $s$ is zero, the third query triggers a self-destruct, else it causes the output to be $0^k$. This is a clear breach of non-malleability, and thus completes the attack description.

The above attack shows that the assumption of augmented non-malleability is not an artifact of our proof, as the adversary might eventually retrieve one half of the codeword via non-adaptive tampering. The latter also explains the reason why the above rate compiler fails to achieve security against adaptive attacks: the attacker can use the ciphertext $\gamma$ as a side channel, in order to leak information about the target codeword without incurring the risk of self-destruct. In the next subsections we show how to disable such a side channel by storing the ciphertext $\gamma$ on both sides of the target codeword.

### 3.2 Rate-1/2 Compiler (Adaptive Tampering)

In §B of the appendix, we explain how to slightly modify the compiler from §3.1 in order to get adaptive security, at the price or reducing the rate of the compiled code to 1/2. The main difference is that the authenticated ciphertext $\gamma$ is stored in both halves of the target codeword, i.e. a codeword is now a tuple $(c_0||\gamma_0, c_1||\gamma_1)$ where $\gamma_0 = \gamma_1 := \gamma$, and the decoding algorithm additionally checks that, indeed, the two ciphertexts $\gamma_0, \gamma_1$ are the same.

Intuitively, an adaptive adversary cannot store useful information about the inner encoding $c_1$ in the part of the codeword that stores $\gamma_1$. The idea is that in such a case, the same information must be guessed on the other side and overwritten in $\tilde{\gamma}_1$, as otherwise the decoding algorithm would output $\perp$ with consequent self-destruct; but then the adversary could have guessed this information directly, even without the need of a tampering oracle.

Note that the adversary might still be able to learn some partial information about the inner encoding, however, we show that this is not a problem as long as the underlying rate-0 continuously non-malleable code satisfies the additional property of being leakage resilient [5, 33, 46]. (Augmented non-malleability is not required here.)

### 3.3 Rate-One Compiler (Adaptive Tampering)

We give yet another twist of the rate-optimizing compiler from §3.1, in order to achieve optimal rate in the (non-programmable) random oracle model. The main idea is to store the ciphertext $\gamma$ on one share of the codeword, say the right share, as before, and to add the hash of $\gamma$ on the left share. Specifically, a codeword is now a tuple $(c_0\|h, c_1\|\gamma)$ where $h = H(\gamma)$, and the decoding additionally checks that indeed the value $h$ is equal to $H(\gamma)$. The intuition is that having $H(\gamma)$ in one share is equivalent to having $\gamma$ itself, as in the random oracle model the value $H(\gamma)$ can be seen as a "handle" for the value $\gamma$.

**Non-malleability in the random oracle model.** We start by explaining what it means to construct a continuously non-malleable code in the (non-programmable) random oracle model. First, the construction itself might make use of the random oracle, so that a code is now a tuple $\Sigma = (\mathsf{Init}^H, \mathsf{Enc}^H, \mathsf{Dec}^H)$ where all algorithms can additionally make random-oracle queries (as in the code sketched above). Second, the adversary $\mathsf{A}$ is allowed to make random-oracle queries, and to specify split-state tampering functions of the form $f := (f_0, f_1)$, such that $f_0$ and $f_1$ can additionally query the random oracle.

When defining non-malleability in the random oracle model, we also assume that the simulator can query the random oracle. We restrict to simulators that simply observe the random oracle queries made by the tampering functions, but do not *program* them, i.e. the so-called non-programmable random oracle model.

**Proof intuition.** We now give an informal argument for the security of the above construction. We do so by showing a reduction to the continuous non-malleability of the code from §3.2; in order to simplify the exposition, we sketch the analysis in the programmable random oracle model, where the reduction/simulator is further allowed to program the random oracle. In §C of the appendix, we give a (slightly more complicated) direct proof that does not require to program the random oracle.

Let $\mathsf{A}$ be an adaptive adversary against the security of the rate-one code; we build an adversary $\mathsf{B}$ against the security of the rate-1/2 code. Adversary $\mathsf{B}$ simply emulates $\mathsf{A}$, keeping a list $\mathcal{Q}_{H,\mathsf{A}}$ of all the random-oracle queries made by $\mathsf{A}$. Upon input a split-state tampering query $(f_0, f_1)$ from $\mathsf{A}$, adversary $\mathsf{B}$ specifies its own tampering function $(f'_0, f'_1)$ as follows:

Tampering function $f_0'(c_0\|\gamma_0)$:

- Compute $h = H(\gamma)$, then execute $f_0(c_0\|h)$.
- Keep a list $\mathcal{Q}_{H,f}$ of all the queries made by $f_0$ to the random oracle.
- Eventually, $f_0$ outputs $(\tilde{c}_0\|\tilde{h})$, try to find a value $\tilde{\gamma} \in \mathcal{Q}_{H,\mathsf{A}} \cup \mathcal{Q}_{H,f}$ such that $H(\tilde{\gamma}) = \tilde{h}$; if such value is found output $(\tilde{c}_0\|\tilde{\gamma})$ else output $\perp$.

Tampering function $f_1'(c_1\|\gamma_1)$:

- Run $f_1(c_1\|\gamma_1)$.

One can show that $\mathsf{B}$ simulates almost perfectly the tampering experiment with $\mathsf{A}$. In fact, the only bad event is when the hash of $\tilde{\gamma}$ as computed by $f_1$ is equal to $\tilde{h}$, but $\tilde{\gamma}$ has never been queried to $H$. However, if the adversary $\mathsf{A}$ or the tampering function $f_0$ do not query the random oracle with $\tilde{\gamma}$, then the bad event happens only with probability $2^{-\lambda}$.

In the above description, we did not specify how the reduction treats random-oracle queries asked by the tampering functions $f_0$ and $f_1$. The latter can be done by replacing the random oracle $H$ with the evaluation of a pseudorandom function $F$ (with random key $\kappa'$ sampled by the reduction) which we can hard-code in the description of $(f_0', f_1')$. This allows to simulate random-oracle queries consistently, but requires to program the random oracle.

# 4 Bit-Wise Tampering

The compiler from §3 automatically implies a rate-compiler for continuously non-malleable codes tolerating bit-wise independent tampering (as $\mathcal{F}^n_{\mathsf{bit}} \subset \mathcal{F}^{n_0,n_1,n}_{\mathsf{split}}$) in the computational setting. However, since continuously non-malleable codes for bit-wise tampering also exist unconditionally [21], it might be possible to obtain such codes with optimal rate in the information-theoretic setting. This section shows that this is indeed possible, by extending the analysis of the compiler from Agrawal *et al.* [8] to the continuous case.

## 4.1 Description of the Compiler

The compiler combines a low-rate continuously non-malleable code (CNMC) $\Sigma'$ against $\mathcal{F}^n_{\mathsf{bit}}$ with an error-correcting secret-sharing scheme (ECSS) $\Pi$ with high rate (cf. §2.5). The main idea of the compiler is to carefully introduce random errors into an encoding of a message $s$ under $\Pi$ and record these errors in a tag $\tau$, which is encoded with $\Sigma'$.

Specifically, let $\Pi = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{ECorr})$ be a $(k, n, T, D)$-ECSS and $\Sigma' = (\mathsf{Init}', \mathsf{Enc}', \mathsf{Dec}')$ be a continuously $\mathcal{F}^{n'}_{\mathsf{bit}}$-non-malleable $(k', n')$-code. Let $E \leq n$ be a parameter to be set later. Consider the following construction[4] of a $(k, n'')$-code $\Sigma'' = (\mathsf{Init}'', \mathsf{Enc}'', \mathsf{Dec}'')$, where $n'' := n + n'$.

$\mathsf{Init}''(1^\lambda)$**:** Upon input $\lambda \in \mathbb{N}$, return $\mathsf{Init}'(1^\lambda)$.

$\mathsf{Enc}''(\omega, s)$**:** Upon input $\omega$ and a message $s \in \{0,1\}^k$:

(a) Choose a set $\mathcal{I} = \{i_1, \ldots, i_E\} \subseteq [n]$ of cardinality $E$ and a string $\xi = (\xi_{i_1}, \ldots, \xi_{i_E}) \in \{0,1\}^E$ uniformly at random and let $\tau = (\mathcal{I}, \xi)$.[5]

[4] While we describe the compiler in the CRS model, our instantiation in §5.2 does not require any trusted setup.

[5] Note that the bits of $\xi$ are indexed by the elements of $\mathcal{I}$.

(b) Compute $a \leftarrow\!\!\$\ \mathsf{Enc}(s)$ and, for $i \in [n]$, let

$$c_i^{(1)} \;=\; \begin{cases} \xi_i & \text{if } i \in \mathcal{I}, \\ a_i & \text{otherwise.} \end{cases}$$

(c) Compute $c^{(2)} \leftarrow\!\!\$\ \mathsf{Enc}'(\omega, \tau)$ and return $c = (c^{(1)}, c^{(2)})$.

$\mathsf{Dec}''(\omega, \tilde{c})$**:** Upon input $\omega$ and $\tilde{c} = (\tilde{c}^{(1)}, \tilde{c}^{(2)})$,

(a) Compute $\tau^* = \mathsf{Dec}'(\omega, \tilde{c}^{(2)})$. If $\tau^* = \bot$, return $\bot$.
(b) Let $a^* = \mathsf{ECorr}(\tilde{c}^{(1)})$. If $a^* = \bot$, return $\bot$.
(c) Let $\tau^* = (\mathcal{I}^*, \xi^*)$ with $\mathcal{I}^* = \{i_1, \ldots, i_E\}$ and $\xi^* = (\xi^*_{i_1}, \ldots, \xi^*_{i_E})$. Define $c^* = (c_1^*, \ldots, c_n^*)$ as

$$c_i^* \;=\; \begin{cases} \xi_i^* & \text{if } i \in \mathcal{I}, \\ a_i^* & \text{otherwise.} \end{cases}$$

If $c^* \neq \tilde{c}^{(1)}$, output $\bot$.
(d) Return $\mathsf{Dec}(a^*)$.

### 4.2 Security Analysis

We establish the following result.

**Theorem 4.** *Let $\Pi$ be a $(k, n, T, D)$-ECSS with rate $\rho = k/n$ and $T = \omega(\log n)$, and let $\Sigma'$ be a continuously $\mathcal{F}^{n'}_{\mathsf{bit}}$-non-malleable code with rate $\rho'$. Then, for any $E$ satisfying*

$$\frac{n \cdot \omega(\log n)}{D} \;=\; E \;<\; \frac{D}{4},$$

*$\Sigma''$ is is a continuously $\mathcal{F}^{n+n'}_{\mathsf{bit}}$-non-malleable code with rate $\rho'' = \frac{k}{\rho^{-1}k + 2\rho'^{-1}E}$.*

**Proof intuition.** Before coming to the formal proof, we discuss some intuition.We start with the real security experiment for code $\Sigma''$ and consider a series of hybrid experiments $\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3$ such that a simulation strategy for the ideal experiment is immediately apparent in $\mathbf{H}_3$.

The first hybrid $\mathbf{H}_1$ changes the way the tampered tag $\tau^*$ is computed when $\mathcal{O}_{\mathsf{maul}}$ answers a tamper query $f$: Instead of computing it from a tampered encoding $f^{(2)}(c^{(2)})$, the simulator $\mathsf{S}'_1$ for the underlying non-malleable code $\Sigma'$ is invoked to determine the outcome of applying $f$. The indistinguishability of the real experiment and $\mathbf{H}_1$ follows directly from the security of $\Sigma'$.

Once the switch to $\mathbf{H}_1$ has been made, the right part $f^{(2)}$ of a tamper function $f = (f^{(1)}, f^{(2)})$ can have one of three effects on the tag $\tau^*$, which lead to the definition of the second hybrid $\mathbf{H}_2$:

1. $\tau^* = \bot$, in which case the outcome of tampering with $f$ is $\bot$ as well.
2. $\tau^*$ is equal to the original tag $\tau$. Thus, if the attacker changes too many bits of the left-hand side encoding $c^{(1)}$, the result will almost surely be $\bot$ since the changes are likely to be inconsistent with the parts of $c^{(1)}$ recorded in the tag and are independent of it. Correspondingly, $\mathbf{H}_2$ is defined to *always* answer such tamper queries by $\bot$. If there are only few changes on the left-hand side, $\mathbf{H}_2$ proceeds as $\mathbf{H}_1$.
3. $\tau^*$ is independent of the original tag. Thus, if the attacker overrides too few bits of $c^{(1)}$, the random errors in $c^{(1)}$ are highly unlikely to match the corresponding bits in $\tau^*$ or not to be detected by the error correction. Correspondingly, $\mathbf{H}_2$ is defined to *always* answers such tamper queries by $\bot$. If there are many overrides on the left-hand side, $\mathbf{H}_2$ proceeds as $\mathbf{H}_1$.

To show that hybrids $\mathbf{H}_1$ and $\mathbf{H}_2$ are indistinguishable, one first argues, drawing on an idea from [20], that for every adaptive strategy, there is an equally good non-adaptive one.[6] The advantage of non-adaptive attackers is bounded by using a simple concentration bound to argue that it is highly unlikely that the query types described above are not caught by comparing the left-hand side to the tag or by performing error correction.

Returning to the case distinction above, it remains to consider the two cases where $\mathbf{H}_1$ was not changed:

2. Suppose $\tau^*$ is equal to the original tag $\tau$ and the tamper function changes only a few bits on the left-hand side. In such a case, it can be shown that the result of the tampering is either the original message $s$ or $\perp$. The key observation here is that in order to determine which is the case, one needs merely to find out whether the tamper function "guesses" the bits of $c^{(1)}$ it overrides correctly.
3. Suppose $\tau^*$ is independent of the original tag and the tamper function overrides most of the bits on the left-hand side. In this case, it can be argued that the outcome of the tampering is either $\perp$ or a *unique* message, stemming from a unique encoding $\tilde{a}$. To see which is the case, one need only determine if the positions that are not overridden by the tampering function match $\tilde{a}$.

This process can be abstracted as a guessing game for a randomly generated encoding $a$ of $s$, where the game ends in a self-destruct as soon as an incorrect guess is made. The self-destruct property allows to argue that the guessing game for $a$ generated as an encoding for $s$ is indistinguishable from the guessing game for, say, the all-zero message (by privacy of the ECSS). Correspondingly, hybrid $\mathbf{H}_3$ is defined to work as $\mathbf{H}_2$, except that it works on an encoding of the all-zero message. The indistinguishability of the hybrids follows directly from the indistinguishability of the guessing games. Since hybrid $\mathbf{H}_3$ is independent of the originally encoded message, it is straight-forward to design a simulation strategy.

*Proof of Theorem 4.* For ease of description, consider the following two algorithms, which capture parts of the encoding and decoding processes, respectively:

$\mathsf{Enc}^*(a, \tau)$**:** Upon input $a$ and $\tau = (\mathcal{I}, \xi)$ with $\mathcal{I} = \{i_1, \ldots, i_E\}$ and $\xi = (\xi_{i_1}, \xi_{i_2}, \ldots, \xi_{i_E})$, output $c$ defined by

$$c_i \;=\; \begin{cases} \xi_i & \text{if } i \in \mathcal{I}, \\ a_i & \text{otherwise.} \end{cases}$$

$\mathsf{ECorr}^*(c, \tau^*)$**.** Upon input $a$ and $\tau^*$:

1. If $\tau^* = \perp$, return $\perp$.
2. Compute $a^* = \mathsf{ECorr}(c)$. If $a^* = \perp$, return $\perp$.
3. Let $c^* = \mathsf{Enc}^*(a^*, \tau^*)$. If $c^* \neq c$, return $\perp$.
4. Return $a^*$.

The proof proceeds via three hybrid experiments $\mathbf{H}_1(\lambda)$, $\mathbf{H}_2(\lambda)$, and $\mathbf{H}_3(\lambda)$, such that $\mathbf{H}_3(\lambda)$ immediately implies a suitable simulation strategy. For brevity, set $\mathbf{Real} := \{\mathbf{Real}_{\Sigma'',\mathsf{A},\mathcal{F}_{\mathsf{bit}}}(\lambda)\}_{\lambda\in\mathbb{N}}$ and $\mathbf{H}_j := \{\mathbf{H}_j(\lambda)\}_{\lambda\in\mathbb{N}}$ for $j \in [3]$. The following parameter conditions need to be met for $E$ and additional parameters $\alpha$ and $\beta$ (introduced merely for readability) for the proof to succeed:

$$\alpha + E < D/2, \qquad \beta + 2E < D, \qquad 3E < D.$$

By setting $\alpha := \beta := D/4$ and using that, by assumption, $E < D/4$, all three conditions are satisfied.

[6] Recall that a non-adaptive attacker submits all tamper queries at once.

**Hybrid $\mathbf{H}_1(\lambda)$.** The first hybrid experiment proceeds as the real experiment $\mathbf{Real}(\lambda)$, except that tag-related tampering is run through the simulator $\mathsf{S}' = (\mathsf{S}'_0, \mathsf{S}'_1)$ for the underlying CNMC $\Sigma'$. Specifically, the following changes are made to $\mathbf{Real}(\lambda)$:

- The common reference string is computed as $(\omega, \sigma) \leftarrow \mathsf{S}'_0(1^\lambda)$.
- Queries $f$ to $\mathcal{O}_{\mathsf{maul}}(\omega, c, \cdot)$ are answered as follows:
  1. Let $f^{(1)} = (f_1, \dots f_n)$ and $f^{(2)} = (f_{n+1}, \dots f_{n+n'})$.
  2. Compute $\tilde{c}^{(1)} = f^{(1)}(c^{(1)})$, and run the simulator to get $\tilde{c}^{(2)} \leftarrow \mathsf{S}'_1(\sigma, f^{(2)})$.
  3. If $\tilde{c}^{(2)} = \diamond$, set $\tau^* := \tau$ else $\tau^* \leftarrow \mathsf{Dec}'(\omega, \tilde{c}^{(2)})$.
  4. Compute $a^* \leftarrow \mathsf{ECorr}^*(\tilde{c}^{(1)}, \tau^*)$. If $a^* = \bot$, return $\bot$.
  5. Return $\mathsf{Dec}(a^*)$.

The following lemma follows immediately from the security of CNMC $\Sigma'$.

**Lemma 5.** $\mathbf{Real} \approx_s \mathbf{H}_1$.

**Hybrid $\mathbf{H}_2(\lambda)$.** The second hybrid experiment $\mathbf{H}_2(\lambda)$ differs from $\mathbf{H}_1(\lambda)$ in that it follows the intuition laid out in the above proof intuition. Let $\mathcal{L} = [n]$ be the indices corresponding to $c^{(1)}$ and define

$$V(f) \;:=\; \{i \in \mathcal{L} \mid f_i \neq \texttt{keep}\} \quad \text{as well as} \quad J(f) \;:=\; \{i \in \mathcal{L} \mid f_i \in \{\texttt{keep}, \texttt{flip}\}\}\,.$$

$\mathbf{H}_2(\lambda)$ answers tamper queries $f$ to $\mathcal{O}_{\mathsf{maul}}(\omega, c, \cdot)$ as $\mathbf{H}_1(\lambda)$, except for Step 5:

5. Depending on the value of $\tilde{c}^{(2)}$, proceed as follows:
   (a) If $\tilde{c}^{(2)} = \bot$, return $\bot$.
   (b) If $\tilde{c}^{(2)} = \diamond$:
   - If $|V(f)| > \alpha$, return $\bot$.
   - Else if $\tilde{c}_i^{(1)} \neq c_i^{(1)}$ for some $i$, return $\bot$.
   - Else, return $s$.

   (c) If $\tilde{c}^{(2)} = \tau^*$:
   - If $|J(f)| > \beta$, return $\bot$.
   - Else, compute $a^* \leftarrow \mathsf{ECorr}^*(\tilde{c}^{(1)}, \tau^*)$ and return $\mathsf{Dec}(a^*)$.

**Lemma 6.** $\mathbf{H}_1 \approx_s \mathbf{H}_2$.

The proof of Lemma 6 makes use of a lemma for experiments with self-destruct, which was introduced in [20]. To understand said lemma, consider the notion of a *g-oracle with self-destruct (g-OSD)*, which is an oracle $\mathcal{O}$ that:

- Answers queries $x$ from a set $\mathcal{X}$ by values $y = g(x, r)$ from a set $\mathcal{Y}$, where $g : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ is an arbitrary function, and $r \in \mathcal{R}$ is the internal randomness of the oracle;
- "Self-destructs" after the first occurrence of $\bot \in \mathcal{Y}$, i.e., answers all subsequent queries by $\bot$.

Let $\mathcal{D} \subseteq \mathcal{X}$, called *dangerous* queries. The $\mathcal{D}$-*bending* $\mathcal{O}'$ of $\mathcal{O}'$ is the $g'$-OSD with

$$g'(x, r) \;:=\; \begin{cases} \bot & \text{if } x \in \mathcal{D}, \\ g(x, r) & \text{otherwise.} \end{cases}$$

The following lemma states that adaptivity does not help in distinguishing an OSD from the $\mathcal{D}$-bending of it, provided that every *non-bent* query $x \notin \mathcal{D}$ can only be answered by a unique value $y_x$ or $\bot$.

**Lemma 7** ( [20]). *Let $\mathcal{O}$ be a g-OSD and $\mathcal{O}'$ be its $\mathcal{D}$-bending for some $\mathcal{D} \subseteq \mathcal{X}$. If for every $x \notin \mathcal{D}$ there exists $y_x \in \mathcal{Y}$ such that $\{g(x,r) \mid r \in \mathcal{R}\} = \{y_x, \perp\}$, then, for any attacker $\mathsf{A}$ interacting with $\mathcal{O}$ or $\mathcal{O}'$,*

$$\left|\mathbb{P}\left[\mathsf{A}^{\mathcal{O}} = 1\right] - \mathbb{P}\left[\mathsf{A}^{\mathcal{O}'} = 1\right]\right| \;\leq\; \max_{x \in \mathcal{D}} \mathbb{P}\left[g(x, \mathbf{R}) \neq \perp\right],$$

*where the probability is over the choice of $\mathbf{R}$.*

*Proof of Lemma 6.* The lemma is proved even conditioned on *all* randomness in either hybrid *except* for the coins $r$ used to generate $\tau = (\mathcal{I}, \xi)$. In order to apply Lemma 7, it must first be established that $\mathbf{H}_2(\lambda)$ is the $\mathcal{D}$-bending of $\mathbf{H}_1(\lambda)$ for some set $\mathcal{D}$. This can be seen via the following choices:

- Let $\mathcal{X} := \mathcal{F}_{\mathsf{bit}}^n$, $\mathcal{R}$ be the randomness space for generating $\tau$, and $\mathcal{Y} := \{0,1\}^k \cup \{\perp\}$.
- Let $g : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ correspond to the oracle $\mathcal{O}_{\mathsf{maul}}(\omega, c, \cdot)$. Observe that (i) $g$ depends on the above conditioning, and (ii) the output of $\mathcal{O}_{\mathsf{maul}}$ can be determined from $f$ and $r \in \mathcal{R}$ alone.
- Let $\mathcal{D} = \mathcal{D}_\diamond \cup \mathcal{D}_{\mathsf{fix}}$ be the set of dangerous queries, where[7]
  - $f \in \mathcal{D}_\diamond$ if $\mathsf{S}_1'(f) = \diamond$ and $|V(f)| > \alpha$, or
  - $f \in \mathcal{D}_{\mathsf{fix}}$ if $\mathsf{S}_1'(f) = \tau^*$ and $|J(f)| > \beta$.

Consider a tamper query $f \notin \mathcal{D}$ and the following three cases:

- $\mathsf{S}_1'(f) = \perp$: in this case, $g(f, \cdot) = \perp$.
- $\mathsf{S}_1'(f) = \diamond$: by the above definition of $\mathcal{D}_\diamond$, this implies that $|V(f)| \leq \alpha$, i.e., $f$ changes at most $\alpha$ positions of the left encoding. Since during the encoding procedure, at most $E$ errors have been added to $a$, and because $\alpha + E < D/2$ the error correction during the decoding recovers the original $a$. Hence, tampering with $f$ can lead either to $\perp$ or $y_f := a$.
- $\mathsf{S}_1'(f) = \tau^*$: by the above definition of $\mathcal{D}_{\mathsf{fix}}$, this implies that $|J(f)| \leq \beta$, i.e., $f$ fixes at least $n - \beta$ positions of the left encoding. Since during the decoding procedure, at most $E$ errors are tolerated, and because $\beta + 2E < D$, there exists only a single codeword $\tilde{a}$ that error correction can possibly find after $f$ has been applied. Hence, tampering with $f$ can lead either to $\perp$ or $y_f := \mathsf{Dec}(\tilde{a})$.

This shows that the preconditions of Lemma 7 are satisfied. In order to finish the proof, one must determine $\max_{f \in \mathcal{D}} \mathbb{P}\left[g(f, \mathbf{R}) \neq \perp\right]$. Let $\varepsilon > 0$ and consider first $f \in \mathcal{D}_\diamond$, assuming $f$ does not contain any flips. According to Lemma 1, (cf. §2.4),except with probability $2e^{-\varepsilon^2 \alpha/3}$,

$$|V(f) \cap \mathcal{I}| \;\geq\; \alpha\left(\frac{E}{n} - \varepsilon\right).$$

Moreover, the probability that all of the overrides $f_i$ (i.e., $f_i \in \{\mathtt{zero}, \mathtt{one}\}$) for $i \in V(f) \cap \mathcal{I}$ match the corresponding $\xi_i$ in $\xi$ is at most $2^{-\alpha(E/n - \varepsilon)}$.

Similarly, consider $f \in \mathcal{D}_{\mathsf{fix}}$ and let $\tau^* = (\mathcal{I}^*, \xi^*)$ be the tag output by the simulator. Lemma 1 implies that except with probability $2e^{-\varepsilon^2 \beta/3}$,

$$E \;\geq\; |J(f) \cap \mathcal{I}| \;\geq\; \beta\left(\frac{E}{n} - \varepsilon\right).$$

Since everything but $\tau$ is fixed, the tampered left-hand-side encodings resulting from the various choices of $\tau$ differ in at most $|J(f) \cap \mathcal{I}| \leq E$ positions. Since the error correction tolerates at

[7] Note that, given the above conditioning, membership in $\mathcal{D}_\diamond$ and $\mathcal{D}_{\mathsf{fix}}$ can be decided without knowing $r$.

most $E$ errors, using that $3E < D$, there exists only a single codeword $a^*$ that can appear while decoding the tampered codeword. The probability that the bits in $J(f) \cap \mathcal{I}$ match the corresponding bits of $a^*$ or $\xi^*$ is at most $2^{-\beta(E/n-\varepsilon)}$. The above shows that for any $\varepsilon > 0$,

$$\mathbb{SD}\left(\mathbf{H}_1(\lambda), \mathbf{H}_2(\lambda)\right) \;\leq\; 2e^{-\varepsilon^2\alpha/3} + 2^{-\alpha(E/n-\varepsilon)} + 2e^{-\varepsilon^2\beta/3} + 2^{-\beta(E/n-\varepsilon)}\,.$$

By choosing $\varepsilon := E/(2n)$ and inserting the choices for $\alpha = \beta = D/4$, this bound simplifies to

$$4e^{-E^2D/(48n^2)} + 2 \cdot 2^{-ED/(8n)}\,,$$

which is negligible by assumption. □

**Hybrid $\mathbf{H}_3(\lambda)$.** The third hybrid experiment $\mathbf{H}_3(\lambda)$ is defined to work exactly as $\mathbf{H}_2(\lambda)$, except that instead of encoding $s$ as specified by the attacker, $\mathbf{H}_3(\lambda)$ simply encodes the all-zero message.

**Lemma 8.** $\mathbf{H}_2 \approx_s \mathbf{H}_3$.

Towards showing that $\mathbf{H}_2$ and $\mathbf{H}_3$ are indistinguishable, fix some message $s$ and consider the following two guessing oracles, both of which self-destruct, i.e., after returning $\perp$ for the first time, all subsequent queries are answered by $\perp$.

- $\mathcal{G}_0$: It internally computes $a \leftarrow \mathsf{Enc}(s)$ and answers guessing queries $(i, b)$ by 1 if $a_i = b$ and by $\perp$, otherwise.
- $\mathcal{G}_1$: It works as $\mathcal{G}_0$ except that $a \in \{0,1\}^n$ is chosen uniformly at random.

The privacy of the ECSS immediately implies the following lemma.

**Lemma 9.** *For any attacker* $\mathsf{A}$ *interacting with* $\mathcal{G}_0$ *or* $\mathcal{G}_1$*:*

$$\left|\mathbb{P}\left[\mathsf{A}^{\mathcal{G}_0} = 1\right] - \mathbb{P}\left[\mathsf{A}^{\mathcal{G}_1} = 1\right]\right| \leq 2^{-T}.$$

*Proof of Lemma 8.* The lemma is proved by observing that there exists a wrapper $\mathsf{W}$ such that $\mathsf{W}^{\mathcal{G}_0}$ and $\mathsf{W}^{\mathcal{G}_1}$ behave as $\mathbf{H}_2(\lambda)$ and $\mathbf{H}_3(\lambda)$, respectively. In particular, $\mathsf{W}$ internally generates $\tau = (\mathcal{I}, \xi)$ and answers queries $f$ to $\mathcal{O}_{\mathsf{maul}}(\omega, c, \cdot)$ as follows:

1. Let $f^{(1)} = (f_1, \ldots f_n)$ and $f^{(2)} = (f_{n+1}, \ldots f_{n+n'})$.
2. Compute $\tilde{c}^{(1)} = f^{(1)}(c^{(1)})$, and run the simulator to get $\tilde{c}^{(2)} \leftarrow \mathsf{S}'_1(\sigma, f^{(2)})$.
3. Depending on the value of $\tilde{c}^{(2)}$, do the following:
   (a) If $\tilde{c}^{(2)} = \perp$, return $\perp$.
   (b) If $\tilde{c}^{(2)} = \diamond$:
   - Return $\perp$ if
     - $|V(f)| > \alpha$,
     - there is even a single $f_i = \texttt{flip}$, or
     - there is an $i \in V(f)$ indexed by $\mathcal{I}$ such that $f_i$ fixes the $i^{\text{th}}$ position to $b$, but $\xi_i \neq b$.
   - Otherwise, for each $i \in V(f) \setminus \mathcal{I}$, query $(i, b)$ to the oracle, where $b$ is the value $f_i$ fixes the $i^{\text{th}}$ position to. If any of the answers from the oracle is $\perp$, return $\perp$. Otherwise, return $s$.

   (c) If $\tilde{c}^{(2)} = \tau^* = (\mathcal{I}^*, \xi^*)$:
   - Return $\perp$
     - if $|J(f)| > \alpha$, or

– there is an $i \in \mathcal{L} \setminus J(f)$ indexed by $\mathcal{I}^*$ such that $f_i$ fixes the $i^{\text{th}}$ position to $b$, but $\xi_i^* \neq b$.

- If there exists no codeword $y_f = \tilde{a}$ as in the proof of Lemma 6, output $\perp$.
- Otherwise, let $\tilde{a}$ be said codeword. For each $i \in J(f)$, query $(i, \tilde{a}_i \oplus b)$ to the oracle, where $b = 0$ for $f_i = \texttt{keep}$ and $b = 1$ for $f_i = \texttt{flip}$. If any of the answers from the oracle is $\perp$, return $\perp$. Otherwise, return $\mathsf{Dec}(\tilde{a})$.

Since, by Lemma 9, the distance between $\mathcal{G}_0$ and $\mathcal{G}_1$ is at most $2^{-T}$, so is the distance between $\mathsf{W}^{\mathcal{G}_0}$ and $\mathsf{W}^{\mathcal{G}_1}$. The lemma follows. □

The theorem now follows by combining the above lemmas. □

# 5 Instantiating the Compilers

## 5.1 Split-State Model

### 5.1.1 Rate-One Code (Non-Adaptive Tampering)

In order to instantiate the compiler from §3.1, we need to exhibit an augmented continuously non-malleable code in the split-state model. Below, we give a short description of such a code, highlighting the main technical challenges. We assume the reader is familiar with the concept of zero-knowledge proofs; see §A of the appendix for formal definitions and proofs.

**The code.** The encoding scheme is a variation of the code from [33]. Given a $k$-bit string $s$, its encoding has the form $(c_0, c_1) = ((c'_0, h_1, \pi_1), (c'_1, h_0, \pi_0))$, where $h_0$ (resp. $h_1$) is a collision-resistant hashing of $c'_0$ (resp. $c'_1$), $\pi_0$ (resp. $\pi_1$) is a NIZK proof of knowledge of a pre-image of the hash value $h_0$ (resp. $h_1$), and $(c'_0, c'_1)$ is a leakage-resilient encoding [23] of the input.[8] The decoding algorithm first checks the validity of the proofs locally on the left and right share, and then it makes sure that $h_0$ (resp. $h_1$) is indeed the hash of $c'_0$ (resp. $c'_1$); if any of the checks fails, it returns $\perp$, and else it decodes $(c'_0, c'_1)$ using the decoding procedure of the leakage-resilient code.

The security proof differs significantly from that of [33]. In particular, we exploit the following additional properties of the leakage-resilient code: (1) It should tolerate so-called noisy leakage [24,31,48], meaning that the parameter $\ell$ is an upper bound on the average min-entropy gap induced by the leakage (and not its bit-length). (2) Indistinguishability should hold even if the distinguisher is given one of the two shares of the target codeword, at the end of the experiment; this property is the one that allows to show *augmented* non-malleability. (3) For all messages, the distributions corresponding to the two shares $c'_0, c'_1$ of an encoding are almost independent. Properties (2) was already used in [33], whereas properties (1) and (3) are easily seen to be met by known constructions (cf. §A.1.2 for details).

**Simulator.** The (augmented) code simulator roughly works as follows. It starts by sampling a dummy encoding $(c'_0, c'_1)$ of the message $0^k$ under the leakage-resilient code, and hence it computes the hash values $h_0, h_1$ and simulates the zero-knowledge proofs $\pi_0, \pi_1$; this defines a simulated codeword $(c_0, c_1) = ((c'_0, h_1, \pi_1), (c'_1, h_0, \pi_0))$. Thus, given a tampering query $(f_0, f_1)$, we design a special simulation strategy that outputs a candidate decoded message acting only either on $(f_0, (c'_0, h_1, \pi_1))$ or on $(f_1, (c'_1, h_0, \pi_0))$. Let $\tilde{s}_0$ and $\tilde{s}_1$ be such candidate messages.

[8] Such an encoding roughly guarantees that $\ell$ bits of independent leakage from $c'_0$ and $c'_1$ do not reveal anything on the encoded message.

Finally, as long as $\tilde{s}_0 = \tilde{s}_1$ the simulator outputs $\tilde{s}_0$, and otherwise it outputs $\perp$ and self-destructs.

Intuitively, we want to make a reduction to the security of the leakage-resilient code in order to switch the dummy encoding of $0^k$ with an encoding of the real message. In such a reduction, the values $\tilde{s}_0$ and $\tilde{s}_1$ are obtained via leakage queries, and thus the main challenge is to argue that such leakage is allowed. Take for instance the left share. The main observation is that, as long as $\tilde{s}_0 = \tilde{s}_1$, then the leakage on $c'_0$ reveals no additional information beyond what is revealed by $c'_1$ and the hash of $c'_0$. In fact, since $\tilde{s}_0 = \tilde{s}_1$, the leakage performed on $c'_0$ could have been also performed on $c'_1$ (as the leaked values are the same!), and furthermore, by property (3) above and by the fact that the hash is short, those values do not reduce the min-entropy of $c'_0$ by too much. On the other hand, if $\tilde{s}_0 \neq \tilde{s}_1$, the amount of leakage can be naively[9] bounded by $2k$, but notice that this happens only once, since the simulator self-destructs after the first $\perp$ is obtained.

**Further optimizations.** Along the way, we were also able to improve the parameters w.r.t. the original proof given by [33]. In particular, the leakage parameter we require from the underlying leakage-resilient code is $\ell' \in O(\lambda)$ instead that $\ell' \in \Omega(\lambda \log \lambda)$ in the original proof. This improvement also yields better efficiency in terms of computational complexity for the zero-knowledge proof system (e.g., when using the Groth-Sahai proof system [40, 41]).

**Putting it together.** Summarizing the above discussion, assuming collision-resistant hash functions and non-interactive zero-knowledge proofs, we have obtained a *rate-optimal* continuously non-malleable code with computational security against *non-adaptive* split-state tampering in the common reference string model, as stated in item (i) of Theorem 1.

### 5.1.2 Rate-1/2 Code (Adaptive Tampering)

In order to instantiate the compiler from §3.2, we need a leakage-resilient continuously non-malleable code in the split-state model. Luckily, as we explain in §B of the appendix, the above construction inherits leakage resilience from the underlying leakage-resilient code.

Hence, assuming collision-resistant hash functions and non-interactive zero-knowledge proofs, we have obtained a rate-1/2 continuously non-malleable code with computational security against *adaptive* split-state tampering in the common reference string model, as stated in item (ii) of Theorem 1.

### 5.1.3 Rate-One Code (Adaptive Tampering)

Finally, as we prove in §C of the appendix, we can instantiate the compiler from §3.3 under the same assumptions of the previous code, i.e. all we need is a leakage-resilient continuously non-malleable code in the split-state model. Here, we can further simplify the above construction by relying on the random oracle heuristic, and consider codewords of the form $(c_0, c_1) = ((c'_0, h_1), (c'_1, h_0))$, where $h_0$ (resp. $h_1$) is computed by hashing $c'_0$ (resp. $c'_1$) via a random oracle. One can prove that this construction achieves (computational) continuous non-malleability in the split-state model.

Hence, we have obtained a *rate-optimal* continuously non-malleable code with computational security against *adaptive* split-state tampering in the (non-programmable) random oracle model, as stated in item (iii) of Theorem 1.

[9] The leakage parameter can be improved to $O(\lambda)$ by leaking a hash of the message.

### 5.2 Bit-Wise Independent Model

The ECSS for the $\mathcal{F}^n_{\mathsf{bit}}$-compiler can be instantiated using share packing, as shown in [8]. This results in a $(k, n, T, D)$-ECSS with $T = D = \tilde{\Theta}(n^{3/4})$ and $n = (1 + o(1))k$, which in turn allows to choose, e.g., $E = n^{1/4+\gamma}$ for any $\gamma > 0$.

The low-rate CNMC $\Sigma'$ can be instantiated, e.g., by the codes of [18, 21]. Note that such codes are in the plain model (i.e., algorithm $\mathsf{Init}'$ returns the empty string), and thus Theorem 4 yields a rate-optimal continuously non-malleable code with information-theoretic security against adaptive bit-wise independent tampering, and without trusted setup, as stated in Theorem 2.

## 6 Conclusions

We have provided several constructions of rate-optimizing compilers for continuously non-malleable codes in the bit-wise independent and split-state tampering models. While in the former case our compiler is optimal both in terms of rate and assumptions (in fact, the result is unconditional), in the latter case we only get rate-optimal codes for the case of non-adaptive tampering and assuming trusted setup, and in the random oracle model. Thus, the main problem left open by our work is whether rate-one continuously non-malleable codes for the split-state model, with adaptive security and without random oracles, actually exist (with or without trusted setup).

A first step towards solving this problem would be to instantiate the random oracle using extractable hash functions [11, 44], or non-malleable hash functions [12]. The difficulty with using the former is that the complexity of the extractor for the hash function depends on that of the adversary, which creates an exponential blow-up when using the naïve strategy of running the extractor to answer each tampering query. The difficulty with using the latter is that non-malleable hash functions in the plain model can only support a limited class of malleability attacks which do not seem sufficient in our analysis.

A middle-ground solution would be to consider a slightly weaker tampering model than fully-fledged split-state tampering, as defined by Dachman-Soled *et al.* [22] (in the context of locally decodable and updatable non-malleable codes). Roughly speaking, we could consider "split-state" tampering functions $(f_0, f_1)$ such that $f_1$ can be further parsed as $f_1 = (f_1^0, f_1^1)$, where $f_1^0$ takes as input the full codeword $c_1 \| \gamma$ but outputs only a tampered value $\tilde{c}_1$, whereas $f_1^1$ takes as input $\gamma$ and outputs a tampered value $\tilde{\gamma}$. In this simpler case, one can show that the rate-compiler of §3.1 actually already achieves adaptive security. Intuitively, this is because the function $f_1$ cannot be exploited to leak information about the value $c_1$ (as $\tilde{\gamma}$ cannot depend on $c_1$).

## References


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

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

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

[4] Divesh Aggarwal, Nico Dottling, Jesper Buus Nielsen, Maciej Obremski, and Erick Purwanto. Continuous non-malleable codes in the 8-split-state model. Cryptology ePrint Archive, Report 2017/357, 2017. https://eprint.iacr.org/2017/357.

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

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

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

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

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

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

[11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science*, pages 326–349, 2012.

[12] Alexandra Boldyreva, David Cash, Marc Fischlin, and Bogdan Warinschi. Foundations of non-malleable hash and one-way functions. In *ASIACRYPT*, pages 524–541, 2009.

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

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

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

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

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

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

[19] V. Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.

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

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

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

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

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

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

[26] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In *ASIACRYPT*, 2011.

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

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

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

[30] Antonio Faonio, Jesper Buus Nielsen, Mark Simkin, and Daniele Venturi. Continuously non-malleable codes with split-state refresh. In *ACNS*, pages 121–139, 2018.

[31] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Fully leakage-resilient signatures revisited: Graceful degradation, noisy leakage, and construction in the bounded-retrieval model. *Theor. Comput. Sci.*, 660:23–56, 2017.

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

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

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

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

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

[37] Vipul Goyal, Aayush Jain, and Dakshita Khurana. Non-malleable multi-prover interactive proofs and witness signatures. Cryptology ePrint Archive, Report 2015/1095, 2015. https://eprint.iacr.org/2015/1095.

[38] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In *ACM STOC*, pages 1128–1141, 2016.

[39] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[40] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.

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

[42] Zahra Jafargholi and Daniel Wichs. Tamper detection and continuous non-malleable codes. In *TCC*, pages 451–480, 2015.

[43] Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Four-state non-malleable codes with explicit constant rate. In *TCC*, pages 344–375, 2017.

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

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

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

[47] Takahiro Matsuda and Goichiro Hanaoka. An asymptotically optimal method for converting bit encryption to multi-bit encryption. In *ASIACRYPT*, pages 415–442, 2015.

[48] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012.

[49] Rafail Ostrovsky, Giuseppe Persiano, Daniele Venturi, and Ivan Visconti. Continuously non-malleable codes in the split-state model from minimal assumptions. In *CRYPTO*, pages 608–639, 2018.

# A Obtaining Augmented Continuous Non-Malleability

## A.1 Ingredients

Here, we introduce the cryptographic building blocks necessary to describe and analyze the coding scheme from [33].

### A.1.1 Conditional Average Min-Entropy

We recall the notions of min-entropy and of conditional average min-entropy. The min-entropy of a random variable $\mathbf{X}$, defined over a set $\mathcal{X}$, is denoted as $\mathbb{H}_\infty(\mathbf{X}) := -\log \max_{x\in\mathcal{X}} \mathbb{P}[\mathbf{X} = x]$ and represents the best chance of guessing $\mathbf{X}$ by an unbounded adversary. Conditional average min-entropy captures how hard it is to guess $\mathbf{X}$ on average, given some side information $\mathbf{Z} \in \mathcal{Z}$ (possibly related to $\mathbf{X}$), and it is denoted as $\widetilde{\mathbb{H}}_\infty(\mathbf{X}|\mathbf{Z}) := -\log \mathbb{E}_{z\in\mathcal{Z}} \max_{x\in\mathcal{X}} \mathbb{P}[\mathbf{X} = x|\mathbf{Z} = z]$. We rely on the following standard lemmata [25].

**Lemma 10.** *Let* $\mathbf{X}$ *and* $\mathbf{Z}$ *be possibly correlated random variables, and let* $g$ *be any (possibly randomized function). Then,* $\widetilde{\mathbb{H}}_\infty(\mathbf{X}|g(\mathbf{Z})) \geq \widetilde{\mathbb{H}}_\infty(\mathbf{X}|\mathbf{Z})$.

**Lemma 11.** *Let* $\mathbf{X}, \mathbf{Z}_1, \mathbf{Z}_2$ *be possibly correlated random variables, where* $\mathbf{Z}_2$ *takes at most* $2^\ell$ *values. Then,* $\widetilde{\mathbb{H}}_\infty(\mathbf{X}|\mathbf{Z}_1, \mathbf{Z}_2) \geq \widetilde{\mathbb{H}}_\infty(\mathbf{X}|\mathbf{Z}_1) - \ell$.

**Lemma 12.** *Let* $\mathbf{X}, \mathbf{Z}$ *be possibly correlated random variables. If* $\widetilde{\mathbb{H}}_\infty(\mathbf{X}|\mathbf{Z}) \geq \mathbb{H}_\infty(\mathbf{X}) - \ell$, *then for any* $1 \geq \delta > 0$ *we have that* $\mathbb{P}_z[\mathbb{H}_\infty(\mathbf{X}|\mathbf{Z} = z) \leq \mathbb{H}_\infty(\mathbf{X}) - \ell + \log \delta] \leq \delta$.

### A.1.2 Noisy-Leakage Resilient Codes

Let $\Sigma = (\mathsf{Enc}, \mathsf{Dec})$ be a $(k, n)$-code. Intuitively, $\Sigma$ is leakage resilient in the split-state model if independent leakage from two shares of a codeword $c := (c_0, c_1)$ does not reveal any information on the encoded message. This notion was introduced by Daví *et al.* [23] for the case where the leakage is arbitrary but of length at most $\ell$ bits. Below, we define a generalization where the bound $\ell \in \mathbb{N}$ is not an upper bound on the length of the leakage, but rather an upper bound on how much information the leakage reveals on the codeword information theoretically.

**Definition 6** (Admissible adversaries)**.** Let $\Sigma$ be a $(k, n)$-code, and $n_0, n_1 \in \mathbb{N}$ be such that $n_0 + n_1 = n$. Consider the experiment in Fig. 5, with a possibly unbounded adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$ and hidden bit $b \in \{0, 1\}$. We say that $\mathsf{A}$ is $\ell$-admissible, if during an execution of the experiment algorithm $\mathsf{A}_1$ outputs a sequence of $q \in \mathbb{N}$ leakage functions $(f_{0,j}, f_{1,j})_{j\in[q]}$ such that for all $\beta \in \{0, 1\}$:

$$\widetilde{\mathbb{H}}_\infty\left(\mathbf{C}_\beta | f_{0,1}(\mathbf{C}_\beta), \cdots, f_{\beta,q}(\mathbf{C}_\beta)\right) \geq \mathbb{H}_\infty(\mathbf{C}_\beta) - \ell,$$

where $(\mathbf{C}_0, \mathbf{C}_1)$ is the random variable corresponding to $\mathsf{Enc}(s_b)$.

**Definition 7** (Noisy-leakage-resilient codes)**.** Let $\Sigma = (\mathsf{Enc}, \mathsf{Dec})$ be a $(k, n)$-code, and $n_0, n_1 \in \mathbb{N}$ be such that $n = n_0 + n_1$. We say that $\Sigma$ is an $\ell$-noisy-leakage resilient code in the split-state model if for all unbounded $\ell$-admissible adversaries $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$ we have that

$$\left\{\mathbf{ExpLeak}_{\Sigma,\mathsf{A}}(\lambda, n_0, n_1, 0)\right\}_{\lambda\in\mathbb{N}} \approx_s \left\{\mathbf{ExpLeak}_{\Sigma,\mathsf{A}}(\lambda, n_0, n_1, 1)\right\}_{\lambda\in\mathbb{N}},$$

where the experiment $\mathbf{ExpLeak}_{\Sigma,\mathsf{A}}(\lambda, n_0, n_1, b)$ is defined in Fig. 3.

Let $\mathbb{F}$ be a finite field, and $t(\lambda) \in \mathbb{N}$ be a parameter. Dziembowski and Faust [26] considered the following encoding scheme $\Sigma = (\mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{Enc}(s)$ samples $c_0, c_1 \leftarrow\!\!\$\ \mathbb{F}^n$ subject to $\langle c_0, c_1\rangle = s$.
- $\mathsf{Dec}(c_0, c_1)$ outputs $\langle c_0, c_1\rangle$.

$\mathbf{ExpLeak}_{\Sigma,\mathsf{A}}(\lambda, n_0, n_1, b)$:
$(s_0, s_1, \alpha_0) \leftarrow\!\!\$\ \mathsf{A}_0(1^\lambda)$
$c := (c_0, c_1) \leftarrow\!\!\$\ \mathsf{Enc}(s_b)$
$\alpha_1 \leftarrow\!\!\$\ \mathsf{A}_1^{\mathcal{O}_{\mathsf{leak}}(c,\cdot)}(\alpha_0)$
$\alpha_2 \leftarrow\!\!\$\ \mathsf{A}_2(\alpha_1, c_1)$
Return $\alpha_2$

Oracle $\mathcal{O}_{\mathsf{leak}}(c, \cdot)$:
Upon $(f_0, f_1) \in \mathcal{F}_{\mathsf{split}}^{n_0,n_1}$:
  Parse $c = (c_0, c_1)$
  Return $(f_0(c_0), f_1(c_1))$

**Figure 5:** Experiment defining security of leakage-resilient codes.

**Theorem 5.** *Let $\mathbb{F}$ be a finite field with size $|\mathbb{F}| \in \Omega(t)$ for a parameter $t \in \mathbb{N}$. Then, for any $\varepsilon(\lambda) \in \mathbb{R}^+$, the above encoding scheme $\Sigma$ is a $(\frac{1}{4}t \log |\mathbb{F}| - \log \varepsilon^{-1})$-noisy-leakage-resilient code, where the parameter $\varepsilon$ is the statistical distance between the two experiments in Def. 7.*

The original proof in [26] shows that the above code $\Sigma$ is in fact leakage resilient in the bounded-leakage model, where $\ell$ is an upper bound on the total amount of leakage. However, the proof of Dziembowski and Faust uses the fact that the length of the leakage is at most $\ell$ bits only to argue that, for any $\delta, \ell > 0$, the following condition is satisfied:

$$\mathbb{P}_z[\mathbb{H}_\infty(\mathbf{C}_\beta|\mathbf{Leak} = z) \leq \mathbb{H}_\infty(c_\beta) - \ell - \log(1/\delta)] \leq \delta,$$

where **Leak** is the random variable corresponding to the leakage performed by the adversary. (See [26, Lemma 20].) By using Lemma 12, we can reach exactly the same conclusion only assuming that $\widetilde{\mathbb{H}}_\infty(\mathbf{C}_\beta|\mathbf{Leak}) \leq \mathbb{H}_\infty(\mathbf{C}_\beta) - \ell$. The rest of the proof is unchanged.

Another observation is that the original definition of leakage-resilient codes does not allow the adversary to obtain one of the two shares of the original encoding at the end of the experiment. However, Faust *et al.* [35] proved that any $\ell$-leakage-resilient code according to the definition of Dziembowski and Faust is also an $(\ell - 1)$-leakage-resilient code according to our definition, and the latter is true even for the case of noisy leakage.
The following lemma will be useful in the sequel.

**Lemma 13.** *Let $\Sigma = (\mathsf{Enc}, \mathsf{Dec})$ be the above defined leakage-resilient code. For any $s \in \mathbb{F}$, define $(\mathbf{C}_0, \mathbf{C}_1)$ to be the random variable corresponding to $\mathsf{Enc}(s)$. Then, for all $\beta \in \{0, 1\}$,*

$$\widetilde{\mathbb{H}}_\infty(\mathbf{C}_\beta|\mathbf{C}_{1-\beta}) \geq (t - 1) \log |\mathbb{F}|.$$

*Proof.* Follows readily by the fact that $\mathbf{C}_0, \mathbf{C}_1$ are sampled uniformly at random conditioned on $\langle \mathbf{C}_0, \mathbf{C}_1 \rangle = s$. In particular, this can be done by sampling $c_0^1, \ldots, c_0^t, c_1^1, \ldots, c_1^{t-1}$ uniformly at random from $\mathbb{F}$, and finally computing $c_1^t$ in such a way that $s = \sum_{i=1}^t c_0^i \cdot c_1^i$. □

## A.2 Non-Interactive Zero-Knowledge Proofs

Let $\mathcal{R}$ be a relation, corresponding to an NP language $\mathcal{L}$. A non-interactive zero-knowledge (NIZK) proof system for $\mathcal{R}$ is a tuple of efficient algorithms $\Pi = (\mathsf{I}, \mathsf{P}, \mathsf{V})$ specified as follows. (i) The randomized algorithm $\mathsf{I}$ takes as input the security parameter and outputs a common reference string $\omega$; (ii) The randomized algorithm $\mathsf{P}(\omega, \phi, (x, w))$, given $(x, w) \in \mathcal{R}$ and a label $\phi \in \{0, 1\}^*$, outputs a proof $\pi$; (iii) The deterministic algorithm $\mathsf{V}^\phi(\omega, \phi, (x, \pi))$, given an instance $x$, a proof $\pi$, and a label $\phi \in \{0, 1\}^*$, outputs either 0 (for "reject") or 1 (for "accept"). We say that a NIZK for relation $\mathcal{R}$ is *correct* if for every $\omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$, any label $\phi \in \{0, 1\}^*$, and any $(x, w) \in \mathcal{R}$, we have that $\mathsf{V}(\omega, \phi, (x, \mathsf{P}(\omega, \phi, (x, w)))) = 1$.

We define two properties of a NIZK proof system. The first property says that honest proofs do not reveal anything beyond the fact that $x \in \mathcal{L}$.

**Definition 8** (Adaptive multi-theorem zero-knowledge)**.** A NIZK with labels $\Pi$ for a relation $\mathcal{R}$ satisfies adaptive multi-theorem zero-knowledge if there exists a PPT simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$ such that the following holds:

(i) Simulator $\mathsf{S}_0$ outputs $\omega$, a simulation trapdoor $\tau_{\mathsf{sim}}$ and an extraction trapdoor $\tau_{\mathsf{ext}}$.
(ii) For all PPT distinguishers $\mathsf{D}$, we have that

$$\Big| \mathbb{P}\left[\mathsf{D}^{\mathsf{P}(\omega,\cdot,(\cdot,\cdot))}(\omega) = 1 : \ \omega \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)\right] - \mathbb{P}\left[\mathsf{D}^{\mathcal{O}_{\mathsf{sim\text{-}zk}}(\cdot,\cdot,\cdot)}(\omega) = 1 : \ (\omega, \tau_{\mathsf{sim}}) \leftarrow\!\!\$\ \mathsf{S}_0(1^\lambda)\right] \Big|$$

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

Groth [39] introduced the concept of simulation-extractable NIZK, which informally states that knowledge soundness should hold even if the adversary can see simulated proofs for possibly false statements of its choice.

**Definition 9** (Simulation extractability)**.** Let $\Pi$ be a NIZK proof systems for a relation $\mathcal{R}$, that satisfies adaptive multi-theorem zero-knowledge w.r.t. a simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$. We say that $\Pi$ is *simulation extractable* if there exists a PPT algorithm $\mathsf{Ext}$ such that every PPT adversary $\mathsf{A}$ has a negligible probability of winning in the following game:

- The challenger runs $(\omega, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}}) \leftarrow\!\!\$\ \mathsf{S}_0(1^\lambda)$, and gives $\omega$ to $\mathsf{A}$.
- Adversary $\mathsf{A}$ is given access to the oracle $\mathcal{O}^*_{\mathsf{sim\text{-}zk}}(\cdot,\cdot)$, which upon input a pair $(\phi, x)$ returns $\mathsf{S}_1(\tau_{\mathsf{sim}}, \phi, x)$; this oracle can be queried only once.
- Adversary $\mathsf{A}$ outputs a tuple $(\phi^*, x^*, \pi^*)$.
- The challenger runs $w \leftarrow\!\!\$\ \mathsf{Ext}(\tau_{\mathsf{ext}}, \phi^*, (x^*, \pi^*))$.

We say that $\mathsf{A}$ wins iff: (a) $(\phi^*, x^*)$ was not queried to the oracle $\mathcal{O}^*_{\mathsf{sim\text{-}zk}}(\cdot,\cdot)$; (b) $\mathsf{V}(\omega, \phi^*, (x^*, \pi^*)) = 1$; (c) $(x^*, w) \notin \mathcal{R}$.

### A.2.1 Collision-Resistant Hash Functions

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

**Definition 10** (Collision resistance)**.** Let $\Pi = (\mathsf{GenHK}, \mathsf{H})$ be a hash function. We say that $\Pi$ is collision resistant if for all PPT adversaries $\mathsf{A}$ there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that:

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

## A.3 Code Description

We recall the coding scheme $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ presented in [33], based on an auxiliary code $\Sigma' = (\mathsf{LREnc}, \mathsf{LRDec})$, on a NIZK proof system $\Pi' = (\mathsf{I}, \mathsf{P}, \mathsf{V})$ and on a hash function $\Pi = (\mathsf{GenHK}, \mathsf{H})$.

$\mathsf{Init}(1^\lambda)$**:** Sample $\omega \leftarrow\!\!\$\ \mathsf{I}(1^\lambda)$, $hk \leftarrow\!\!\$\ \mathsf{GenHK}(1^\lambda)$, and return $\overline{\omega} := (\omega, hk)$.

$\mathsf{Enc}(\overline{\omega}, s)$**:** Parse $\overline{\omega} := (\omega, hk)$, sample $(c'_0, c'_1) \leftarrow\!\!\$\ \mathsf{LREnc}(s)$, and for all $\beta \in \{0,1\}$ compute $h_\beta := \mathsf{H}(hk, c'_\beta)$ and $\pi_\beta \leftarrow\!\!\$\ \mathsf{P}(\omega, h_{1-\beta}, (h_\beta, c'_\beta))$. Output $c := (c_0, c_1)$ where for $\beta \in \{0,1\}$ we have $c_\beta := (c'_\beta, h_{1-\beta}, \pi_{1-\beta})$.

$\mathsf{Dec}(\overline{\omega}, c)$**:** Parse $\overline{\omega} := (\omega, hk)$ and $c := (c_0, c_1)$, where $c_\beta := (c'_\beta, h_{1-\beta}, \pi_{1-\beta})$ for $\beta \in \{0,1\}$. Compute $s = \mathsf{LRDec}(c'_0, c'_1)$, and $\phi_0 := \mathsf{H}(hk, c'_1)$ and $\phi_1 := \mathsf{H}(hk, c'_0)$. If the following conditions hold return $s$, and else return $\perp$.

(a) Left check: $\mathsf{V}(\omega, \phi_1, (h_1, \pi_1)) = 1$;
(b) Right check: $\mathsf{V}(\omega, \phi_0, (h_0, \pi_0)) = 1$;
(c) Cross check: $h_0 = \mathsf{H}(hk, c'_0)$ and $h_1 = \mathsf{H}(hk, c'_1)$.

Faust *et al.* [33] showed that if $\Sigma'$ is a leakage-resilient code, the NIZK $\Pi'$ satisfies zero-knowledge and simulation extractability, and the hash function $\Pi$ is collision resistant, then the coding scheme $\Sigma$ from above is continuously non-malleable in the split-state model. Below we prove a stronger statement, namely, that under the same assumptions the code $\Sigma$ is *augmented* continuously non-malleable in the split-state model.

**Theorem 6.** *Let $\Sigma'$ be a $(k, n')$-code that is $(\lambda + k + \omega(\log(\lambda)))$-noisy-leakage resilient, and assume that, for all $\beta \in \{0,1\}$ and for all $s \in \{0,1\}^k$, the following holds for the conditional average min-entropy of the random variable $(\mathbf{C}'_0, \mathbf{C}'_1)$ corresponding to $\mathsf{LREnc}(s)$: $\widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta | \mathbf{C}'_{1-\beta}) \geq \mathbb{H}_\infty(\mathbf{C}'_\beta) - k$. Additionally, let $\Pi$ be a collision resistant hash function with range $\{0,1\}^\lambda$, and $\Pi'$ be NIZK proof systems for the NP-relation $\mathcal{R}_{\mathsf{hash}} = \{((hk, h), x) : \mathsf{H}(hk, x) = h\}$, satisfying adaptive multi-theorem zero-knowledge and simulation extractability.*

*Then the code $\Sigma$ described above is an augmented continuously $\mathcal{F}^{n/2,n/2}_{\mathsf{split}}$-non-malleable $(k, n)$-code, where $n(\lambda) = n'(\lambda) + 2(\lambda + n_{\mathsf{nizk}}(\lambda))$, and where $n_{\mathsf{nizk}}(\lambda)$ is the size of a proof.*

By plugging in the parameters from Theorem 5, we see that it is sufficient to have $\log |\mathbb{F}| = k$ and $\epsilon = 2^{-\lambda}$ and set $n'(\lambda) = 16k(\lambda) + 9\lambda$ when we instantiate the leakage-resilient code using the inner-product extractor. The additional condition on the average min-entropy of $(\mathbf{C}'_0, \mathbf{C}'_1)$ follows by Lemma 13.

### A.4 Security Proof

We start by describing the simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$. Recall that $\mathsf{S}_0$ has to fake the CRS and additionally the right share of a real encoding, while $\mathsf{S}_1$ has to answer tampering queries without knowing the message, and in a way that is consistent with the right share simulated by $\mathsf{S}_0$. Let $\mathsf{S}' := (\mathsf{S}'_0, \mathsf{S}'_1)$ be the zero-knowledge simulator for the NIZK proof system $\Pi$. Intuitively, $\mathsf{S}_0$ will fake the CRS using $\mathsf{S}'_0$, and will emulate the right share of the target codeword by encoding a dummy message and simulating the zero-knowledge proofs.

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

- Sample $(\omega, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}}) \leftarrow\!\!\$\ \mathsf{S}'_0(1^\lambda)$, and $hk \leftarrow\!\!\$\ \mathsf{GenHK}(1^\lambda)$.
- Run $(c'_0, c'_1) \leftarrow\!\!\$\ \mathsf{LREnc}(0^k)$, compute $h_0 = \mathsf{H}(hk, c'_0)$, $h_1 = \mathsf{H}(hk, c'_1)$, $\pi_0 \leftarrow\!\!\$\ \mathsf{S}'_1(\tau_{\mathsf{sim}}, h_1, (hk, h_0))$, and $\pi_1 \leftarrow\!\!\$\ \mathsf{S}'_1(\tau_{\mathsf{sim}}, h_0, (hk, h_1))$.
- Let $c := (c_0, c_1)$, where $c_0 = (c'_0, h_1, \pi_1)$ and $c_1 = (c'_1, h_0, \pi_0)$.
- Output $(\overline{\omega}, \sigma, \hat{c}_1)$, where $\overline{\omega} := (\omega, hk)$, $\sigma := (\tau_{\mathsf{ext}}, c_0, c_1)$, and $\hat{c}_1 := c_1$.

**The tampering and leakage simulators.** We now turn to defining the simulator $\mathsf{S}_1$. To facilitate the description, in Fig. 6, we formalize an algorithm $\mathsf{T}$ that we call the *tampering simulator* and an algorithm $\mathsf{L}$ that we call the *leakage simulator*. Intuitively, algorithm $\mathsf{T}$ simulates the outcome corresponding to a tampering query $(f_0, f_1)$ from the adversary, by using only $c_\beta$ (i.e., one share of the simulated codeword). On a very high level, as long as both $\mathsf{T}(c_0, f_0)$ and $\mathsf{T}(c_1, f_1)$ return the same message $\tilde{s}$, the simulator $\mathsf{S}_1$ uses $\tilde{s}$ to answer the tampering queries from the adversary; otherwise, in case of disagreement, it emulates a decoding error (and subsequent self-destruct). The leakage simulator, instead, computes the hash values corresponding to a tampering query $(f_0, f_1)$. Intuitively, this is the leakage from the other share of the codeword that $\mathsf{T}$ needs in order emulate the "cross check" step of the decoding algorithm.

Simulator $\mathsf{S}_1(\sigma, (f_0, f_1))$:

Algorithm $\mathsf{T}(c_\beta, f_\beta, \tilde{h}^{\mathsf{leak}}_{1-\beta})$:

1. Parse $c_\beta := (c'_\beta, h_{1-\beta}, \pi_{1-\beta})$.
2. Compute $\tilde{c}_\beta := (\tilde{c}'_\beta, \tilde{h}_{1-\beta}, \tilde{\pi}_{1-\beta}) = f(c_\beta)$ and $\tilde{\phi}_{1-\beta} := \mathsf{H}(hk, c'_\beta)$.
3. If $\tilde{h}^{\mathsf{leak}}_{1-\beta} \neq \tilde{h}_{1-\beta}$, set $\tilde{s}_\beta := \perp$ ("cross check").
4. Else if $\mathsf{V}(\omega, \tilde{\phi}_{1-\beta}, (hk, \tilde{h}_{1-\beta}), \tilde{\pi}_{1-\beta}) = 0$, set $\tilde{s}_\beta := \perp$ ("left/right check").
5. Else if $(\tilde{c}'_\beta, \tilde{h}_{1-\beta}) = (c'_\beta, h_{1-\beta})$, set $\tilde{s}_\beta := \diamond$.
6. Else run $\tilde{c}'_{1-\beta} \leftarrow\!\!\$\ \mathsf{Ext}(\tau_{\mathsf{ext}}, \tilde{\phi}_{1-\beta}, (hk, \tilde{h}_{1-\beta}), \tilde{\pi}_{1-\beta})$.
   (a) In case $\tilde{c}'_{1-\beta} = \perp$, set $\tilde{s}_\beta := \perp$;
   (b) Otherwise set $\tilde{s}_\beta := \mathsf{LRDec}(\tilde{c}'_0, \tilde{c}'_1)$.
7. Return $(\tilde{s}_\beta, \tilde{h}_{1-\beta})$.

Algorithm $\mathsf{L}(c_\beta, f_\beta)$:

1. Parse $c_\beta := (c'_\beta, h_{1-\beta}, \pi_{1-\beta})$.
2. Compute $\tilde{c}_\beta := (\tilde{c}'_\beta, \tilde{h}_{1-\beta}, \tilde{\pi}_{1-b}) = f(c_\beta)$.
3. Return $\mathsf{H}(hk, \tilde{c}'_\beta)$.

**Figure 6:** The tampering and leakage simulators.

1. Parse $\sigma := (\tau_{\mathsf{ext}}, (c_0, c_1))$.
2. Let $\tilde{h}^{\mathsf{leak}}_0 = \mathsf{L}(c_0, f_0)$ and $\tilde{h}^{\mathsf{leak}}_1 = \mathsf{L}(c_1, f_1)$.
3. Let $(\tilde{s}_0, \tilde{h}_0) = \mathsf{T}(c_0, f_0, \tilde{h}^{\mathsf{leak}}_1)$ and $(\tilde{s}_1, \tilde{h}_1) \leftarrow \mathsf{T}(c_1, f_1, \tilde{h}^{\mathsf{leak}}_0)$:
   (a) If either $\tilde{s}_0 = \perp$ or $\tilde{s}_1 = \perp$, or $\tilde{s}_0 \neq \tilde{s}_1$, return $\perp$ and self-destruct;
   (b) Else return $\tilde{s} := \tilde{s}_0 = \tilde{s}_1$.

In order to prove that the above simulator works, we define a sequence of hybrid experiments starting with the ideal experiment and ending with the real experiment.

**Hybrid $\mathbf{H}_0(\lambda)$:** This is the ideal experiment, with simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$ as defined above.

**Hybrid $\mathbf{H}_1(\lambda)$:** This hybrid is identical to the experiment $\mathbf{Simu}^{+}_{\mathsf{S},\mathsf{A},\mathcal{F}_{\mathsf{split}}}(\lambda, n/2, n/2)$, except that we change the distribution of the target codeword as follows.

1. Sample $(\omega, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}}) \leftarrow\!\!\$\ \mathsf{S}'_0(1^\lambda)$ and $hk \leftarrow\!\!\$\ \mathsf{GenHK}(1^\lambda)$, and set $\overline{\omega} := (\omega, hk)$.
2. Run $(s, \alpha_0) \leftarrow\!\!\$\ \mathsf{A}_0(\overline{\omega})$.
3. Sample $(c'_0, c'_1) \leftarrow\!\!\$\ \mathsf{LREnc}(s)$. For each $\beta \in \{0,1\}$, compute $h_\beta = \mathsf{H}(hk, c'_\beta)$ and $\pi_\beta \leftarrow\!\!\$\ \mathsf{S}'_1(\tau_{\mathsf{sim}}, h_{1-\beta}, (hk, h_\beta))$.
4. Return $c := (c_0, c_1)$, where $c_0 = (c'_0, h_1, \pi_1)$ and $c_1 = (c'_1, h_0, \pi_0)$.

**Hybrid $\mathbf{H}_2(\lambda)$:** We change the way tampering queries are answered. Namely, instead of answering tampering query $(f_{0,j}, f_{1,j})$ by running $\tilde{s}_j \leftarrow\!\!\$\ \mathsf{S}_1(\sigma, (f_{0,j}, f_{1,j}))$, we now instead return $\tilde{s}_j = \mathsf{Dec}(\overline{\omega}, f_{0,j}(c_0), f_{1,j}(c_1))$.

**Hybrid $\mathbf{H}_3(\lambda)$:** We modify the distribution of the CRS. Namely, we now sample $\overline{\omega} \leftarrow \mathsf{Init}(1^\lambda)$ and also compute the proofs $\pi_0, \pi_1$ of the target codeword by running the prover of the underlying zero-knowledge proof system.

**Lemma 14.** *For all (even unbounded) adversaries* $\mathsf{A}$*, we have that* $\{\mathbf{H}_0(\lambda)\}_{\lambda\in\mathbb{N}} \approx_s \{\mathbf{H}_1(\lambda)\}_{\lambda\in\mathbb{N}}$.

*Proof.* The proof is down to the security of the leakage-resilient code $\Sigma'$, as the only difference between the two experiments is that the former runs the simulator $\mathsf{S}_1$ on a dummy encoding

of zero, whereas the latter uses an encoding of the message $s$ chosen by the adversary. By contradiction, assume that there exists an unbounded distinguisher $\mathsf{D}$ and an unbounded $\mathsf{A}$ such that $\mathsf{D}$ can tell apart the two experiments with non-negligible probability. Consider the following attacker $\mathsf{B}$ against the leakage-resilient code.

Reduction $\mathsf{B}$:

1. Sample $(\omega, \tau_{\mathsf{sim}}, \tau_{\mathsf{ext}}) \leftarrow\!\!\$\ \mathsf{S}'_0(1^\lambda)$ and $hk \leftarrow\!\!\$\ \mathsf{GenHK}(1^\lambda)$, and set $\overline{\omega} := (\omega, hk)$.
2. Run $(s, \alpha_0) \leftarrow\!\!\$\ \mathsf{A}_0(\overline{\omega})$.
3. Forward the pair $(0^k, s)$ to the challenger and obtain oracle access to $\mathcal{O}_{\mathsf{leak}}(c', \cdot)$, where $c' = (c'_0, c'_1)$ is the target codeword for the leakage-resilient code.
4. Query the leakage oracle with $\mathsf{L}(\cdot, \mathsf{id})$ and $\mathsf{L}(\cdot, \mathsf{id})$), where $\mathsf{id}$ is the identity function. Denote by $h_0, h_1$ the answer from the oracle; compute $\pi_0 \leftarrow\!\!\$\ \mathsf{S}'_1(\tau_{\mathsf{sim}}, \phi_0, (hk, h_0))$ and $\pi_1 \leftarrow\!\!\$\ \mathsf{S}'_1(\tau_{\mathsf{sim}}, \phi_1, (hk, h_1))$, where $\phi_0 = h_1$ and $\phi_1 = h_0$.
5. Run $\mathsf{A}_1(\alpha_0)$. Upon input a tampering query $(f_0, f_1) \in \mathcal{F}^{n/2, n/2}_{\mathsf{split}}$ from $\mathsf{A}_1$, define $f'_0 := f_0(\cdot, h_1, \pi_1)$ and $f'_1 := f_1(\cdot, h_0, \pi_0)$, and answer as follows:
   (a) Query the leakage oracle with $\mathsf{L}(\cdot, f'_0)$ and $\mathsf{L}(\cdot, f'_1)$. Denote by $\tilde{h}^{\mathsf{leak}}_0$ and $\tilde{h}^{\mathsf{leak}}_1$ the answers from the oracle.
   (b) Query the leakage oracle with $\mathsf{T}(\cdot, f'_0, \tilde{h}^{\mathsf{leak}}_1)$ and $\mathsf{T}(\cdot, f'_1, \tilde{h}^{\mathsf{leak}}_0)$); denote by $(\tilde{s}_0, \tilde{h}_1)$ and $(\tilde{s}_1, \tilde{h}_0)$ the answers from the oracle.
      - If either $\tilde{s}_0 = \perp$ or $\tilde{s}_1 = \perp$, or $\tilde{s}_0 \neq \tilde{s}_1$, return $\perp$ and self-destruct;
      - Else return $\tilde{s} := \tilde{s}_0 = \tilde{s}_1$.
6. Let $\alpha_1$ be the state information returned by adversary $\mathsf{A}_1$. Upon receiving the right share $c'_1$, define $c_1 := (c'_1, h_0, \pi_0)$ and run $\mathsf{A}_2(\alpha_2, c_1)$ obtaining state $\alpha_2$.
7. Return the same as $\mathsf{D}(\alpha_2)$.

We first notice that $\mathsf{B}$ perfectly simulates the experiment $\mathbf{H}_0(\lambda)$ when the challenge bit $b$ is 0, and perfectly simulates the experiment $\mathbf{H}_1(\lambda)$ when the challenge bit $b$ is one. In fact, by inspection, we see that the common reference string is computed exactly as in both experiments, the target codeword is perfectly simulated inside the leakage oracle (due to the fact that the target encoding $(c'_0, c'_1)$ is either distributed as in $\mathsf{LREnc}(0^k)$ or $\mathsf{LREnc}(s)$), and finally tampering queries are answered exactly as $\mathsf{S}_1$ would do, except that the output of the functions $\mathsf{L}$ and $\mathsf{T}$ is obtained via leakage queries.

Hence, it suffices to prove that $\mathsf{B}$ is $\ell'$-admissible, for $\ell'$ as in the statement of the theorem. Note that adversary $\mathsf{B}$ makes leakage queries at steps 4, 5a, and 5b, but the leakage query of step 4 is executed only once. Let $\mathbf{j}^* \in \mathbb{N}$ be the random variable corresponding to the the index of the tampering query where a self-destruct is triggered (if any). Clearly, the leakage queries of steps 5a and 5b are executed exactly $\mathbf{j}^*$ times. Denote by $\mathbf{Leak}_\beta$, for $\beta \in \{0, 1\}$, the random variable corresponding to the leakage performed by the reduction on each share of the target encoding $(\mathbf{C}'_0, \mathbf{C}'_1)$; note that $\mathbf{Leak}_\beta$ consists of the hash value $h_\beta$ and a tuple $\Lambda_{\beta,j} := (\tilde{h}^{\mathsf{leak}}_{\beta,j}, \tilde{s}_{\beta,j}, \tilde{h}_{1-\beta,j})$ for each tampering query $j \leq \mathbf{j}^*$ as defined in Fig. 6. We claim that, for each $j \leq \mathbf{j}^*$, $\Lambda_{0,j}$ and $\Lambda_{1,j}$ are identical up to re-ordering of the elements. In fact, there exists a bijection $\varphi$ such that $\varphi(\Lambda_{1-\beta,j}) = \Lambda_{\beta,j}$ for all $\beta \in \{0, 1\}$. The latter holds because, by definition of the self-destruct index $\mathbf{j}^*$, it must be the case that $\tilde{s}_{0,j} = \tilde{s}_{1,j}$, and moreover $\tilde{h}^{\mathsf{leak}}_\beta = \tilde{h}_\beta$. Hence, we can take $\varphi(\Lambda_{1-\beta,j}) = \varphi(\tilde{h}^{\mathsf{leak}}_{1-\beta,j}, \tilde{s}_{1-\beta,j}, \tilde{h}_{\beta,j}) = (\tilde{h}_{\beta,j}, \tilde{s}_{1-\beta,j}, \tilde{h}^{\mathsf{leak}}_{1-\beta,j})$.

Assume that the adversary $\mathsf{A}_1$ makes at most $q(\lambda) \in \mathtt{poly}(\lambda)$ tampering queries. For all

$\beta \in \{0,1\}$, we can write:

$$\widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\mathbf{Leak}_\beta) \geq \widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\mathbf{\Lambda}_{\beta,1},\ldots,\mathbf{\Lambda}_{\beta,\mathbf{j}^*}) - \lambda \quad (1)$$

$$= \widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\varphi(\mathbf{\Lambda}_{1-\beta,1}),\ldots,\varphi(\mathbf{\Lambda}_{1-\beta,\mathbf{j}^*})) - \lambda \quad (2)$$

$$\geq \widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\mathbf{C}'_{1-\beta},\mathbf{j}^*) - \lambda \quad (3)$$

$$\geq \widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\mathbf{C}'_{1-\beta}) - \lambda - O(\log(\lambda)), \quad (4)$$

where Eq. (1) follows by definition of $\mathbf{Leak}_\beta$ and by Lemma 11, Eq. (2) follows by definition of the map $\varphi(\cdot)$, Eq. (3) comes from Lemma 10 and by interpreting the tuple $(\varphi(\mathbf{\Lambda}_{1-\beta,1}),\ldots,\varphi(\mathbf{\Lambda}_{1-\beta,\mathbf{j}^*}))$ as a function of $(\mathbf{C}'_{1-\beta},\mathbf{j}^*)$, and finally Eq. (4) uses the assumption that $q(\lambda) \in \mathtt{poly}(\lambda)$. The lemma now follows by plugging in the bound on $\widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\mathbf{C}'_{1-\beta})$ from the statement of the theorem. □

**Lemma 15.** *For all PPT adversaries* $\mathsf{A}$*, we have that* $\{\mathbf{H}_1(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_2(\lambda)\}_{\lambda\in\mathbb{N}}$.

*Proof.* For each tampering query $(f_{0,j}, f_{1,j})$ asked by the adversary, let $\tilde{\mathbf{s}}_j$ and $\tilde{\mathbf{s}}^{\mathsf{dec}}$ be, respectively, the random variables corresponding to the decoding of the tampered codeword $(f_{0,j}(c_0), f_{1,j}(c_1))$ as computed by simulator $\mathsf{S}_1$ and by the decoding algorithm $\mathsf{Dec}$; we also write $\tilde{\mathbf{s}}_{0,j}$, $\tilde{\mathbf{s}}_{1,j}$ for the intermediate values computed by $\mathsf{S}_1$. Define the event $\mathtt{Bad}_j$ that $\tilde{\mathbf{s}}_j \neq \tilde{\mathbf{s}}^{\mathsf{dec}}_j$; clearly, conditioned on $\neg\bigvee_j \mathtt{Bad}_j$ not happening, the two experiments are identical, and so it suffices to upper bound the probability of $\mathtt{Bad}_j$ for all $j \in [q]$, where $q(\lambda) \in \mathtt{poly}(\lambda)$ is the number of tampering queries.

We claim that for all $j \in [q]$, there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $\mathbb{P}[\mathtt{Bad}_j] \leq \nu(\lambda)$. Below, we omit to write the index $j$ for clarity. Consider the following cases:

**Case 1:** $\exists\beta \in \{0,1\}$ s.t. $\tilde{\mathbf{s}}_\beta = \perp$ but $\tilde{\mathbf{s}}^{\mathsf{dec}} \neq \perp$. The second condition implies that $\mathsf{V}(\omega, \tilde{h}_\beta, (hk, \tilde{h}_{1-\beta}), \tilde{\pi}_{1-b}) = 1$, while the first condition implies that $(\tilde{c}'_\beta, \tilde{h}_{1-\beta}) \neq (c'_\beta, h_{1-\beta})$, as otherwise $\tilde{s}_\beta = \diamond$. Hence, the adversary has created a mauled proof for which the extractor fails to extract a valid witness, which can only happen with negligible probability by simulation extractability of the NIZK. The reduction is straightforward, and therefore omitted.

**Case 2:** $\forall\beta \in \{0,1\}, \tilde{\mathbf{s}}_\beta \neq \perp$, but $\tilde{\mathbf{s}}_0 \neq \tilde{\mathbf{s}}_1$ and $\tilde{\mathbf{s}}^{\mathsf{dec}} \neq \perp$. For all $\beta \in \{0,1\}$, let us write $\tilde{c}''_\beta$ for the share extracted by the simulator, and $\tilde{c}'_\beta$ for the mauled share processed by the decoding algorithm. The third condition implies that $\mathsf{H}(hk, \tilde{c}'_\beta) = \tilde{h}_\beta$, while the first condition implies that $\mathsf{H}(hk, \tilde{c}''_\beta) = \tilde{h}_\beta$ (this is because we assume that if the extractor did not output $\perp$, then the output is a valid witness). Finally, the second condition implies that $\mathsf{LRDec}(\tilde{c}''_0, \tilde{c}'_1) \neq \mathsf{Dec}(\tilde{c}'_0, \tilde{c}''_1)$, which yields that either $\tilde{c}''_0 \neq \tilde{c}'_0$ or $\tilde{c}'_1 \neq \tilde{c}''_1$. So, we found a collision in the hash function $\Pi$, which only happens with negligible probability. The reduction is straightforward, and therefore omitted.

**Case 3:** $\tilde{\mathbf{s}}_0 = \tilde{\mathbf{s}}_1 \neq \perp$, but $\tilde{\mathbf{s}}_0 \neq \tilde{\mathbf{s}}^{\mathsf{dec}} \neq \perp$. Recall that $\tilde{s}^{\mathsf{dec}} = \mathsf{LRDec}(\tilde{c}'_0, \tilde{c}'_1)$, while $\tilde{s}_0 = \mathsf{LRDec}(\tilde{c}'_0, \tilde{c}''_1)$. By the same argument as the previous case, we have that $\mathsf{H}(hk, \tilde{c}''_1) = \mathsf{H}(hk, \tilde{c}'_1)$, and if $\tilde{s}_0 \neq \tilde{s}^{\mathsf{dec}}$ we have that $\tilde{c}'_1 \neq \tilde{c}''_1$ so that we have found a collision. Thus, this event happens only with negligible probability.

**Case 4:** $\tilde{\mathbf{s}}_0 = \tilde{\mathbf{s}}_1 \neq \perp$, but $\tilde{\mathbf{s}}^{\mathsf{dec}} = \perp$. The third condition implies that either the "left check" or the "right check" or the "cross check" failed. However, the simulator performs exactly the same checks, so this event never happens.

Note that the above cases fully partition the event $\mathtt{Bad}_j$, and thus a standard union bound implies the lemma. □

**Lemma 16.** *For all PPT adversaries* $\mathsf{A}$*, we have that* $\{\mathbf{H}_2(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_3(\lambda)\}_{\lambda\in\mathbb{N}}$.

*Proof.* Follows directly by adaptive multi-theorem zero-knowledge. The reduction is obvious, so we omit it. □

The theorem follows by the lemmas above, and by observing that $\mathbf{H}_3(\lambda)$ is identically distributed to the real experiment.

# B Split-State Rate-1/2 Compiler with Adaptive Security

## B.1 Description of the Compiler

Let $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a rate-zero $(d, n)$-code, and $\Pi = (\mathsf{KGen}, \mathsf{AEnc}, \mathsf{ADec})$ be a $(d, k, m)$-SKE scheme. Consider the following construction of a $(k, n')$-code $\Sigma' = (\mathsf{Init}', \mathsf{Enc}', \mathsf{Dec}')$, where $n' := 2m + n$.

$\mathsf{Init}'(1^\lambda)$**:** Upon input $\lambda \in \mathbb{N}$, return the same as $\mathsf{Init}(1^\lambda)$.
$\mathsf{Enc}'(\omega, s)$**:** Upon input $\omega$ and a value $s \in \{0,1\}^k$, sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, compute $c \leftarrow\!\!\$\ \mathsf{Enc}(\omega, \kappa)$ and $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, s)$; parse $c$ as $c_0||c_1$, and return $c' := (c'_0, c'_1) = ((c_0, \gamma), (c_1, \gamma))$.
$\mathsf{Dec}'(\omega, c')$**:** Parse $c' := ((c_0, \gamma_0), (c_1, \gamma_1))$. If $\gamma_0 \neq \gamma_1$, return $\perp$ and self destruct; else let $\tilde{\kappa} = \mathsf{Dec}(\omega, c_0||c_1)$. If $\tilde{\kappa} = \perp$, return $\perp$ and self destruct; else return the same as $\mathsf{ADec}(\tilde{\kappa}, \gamma_0)$.

The main difference with the compiler presented in §3.1 is that the ciphertext $\gamma$ is stored in both the left and the right share of the codeword, and the decoding algorithm additionally checks that the two ciphertexts are the same.

## B.2 Leakage-Resilient Continuously Non-Malleable Codes

In order to prove security of the above rate compiler, we will need an underlying continuously non-malleable code that is also leakage resilient, i.e. non-malleability should hold even against an adversary that can leak independently from the two shares of an encoding.

Let $\mathcal{O}_{\mathsf{leak}}$ be the leakage oracle of Fig. 5, and consider a modified version **LKReal** of experiment **Real** (see Fig. 1) where the adversary $\mathsf{A}_2$, in addition to the tampering oracle $\mathcal{O}_{\mathsf{maul}}$, has also access to a leakage oracle $\mathcal{O}_{\mathsf{leak}}(c, \cdot)$. Similarly, we consider a modified version **LKSimu** of experiment **Simu** where the simulator replies to the leakage queries of the adversary. More in details, the simulator receives both queries of the form $(\texttt{tamp}, f)$ (i.e., tampering queries), and queries of the form $(\texttt{leak}, g)$) (i.e., leakage queries).

Intuitively, we would like to say that a code is leakage-resilient continuously non-malleable if there exists a simulator such that the real and ideal experiments are computationally close as long as the adversary is $\ell$-admissible (cf. Def. 6). However, defining this is a bit tricky in the computational setting due to the fact that the bound $\ell$ is an information theoretic measure that depends on the distribution of the codeword, but such a distribution can be statistically far within the real and ideal experiment. We therefore consider a more stringent definition with a so-called "canonical simulator."

**Definition 11** (Canonical simulator)**.** A simulator $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$ is *canonical* if and only if the following conditions hold.

1. Let $(\omega, \sigma) \leftarrow \mathsf{S}_0(1^\lambda)$, then we can parse $\sigma$ as $(\hat{\sigma}, (\hat{c}_0, \hat{c}_1))$.
2. The simulator $\mathsf{S}_1$ answers a leakage query $(\texttt{leak}, g := (g_0, g_1))$ by returning $g_0(\hat{c}_0), g_1(\hat{c}_1)$.

We call $(\hat{c}_0, \hat{c}_1)$ the *simulated codeword*.

Observe that the simulator of the augmented continuously non-malleable code from §A is, indeed, canonical. We can now generalize the notion of $\ell$-admissible adversary by requiring that, for all $\beta \in \{0,1\}$,

$$\widetilde{\mathbb{H}}_\infty(\hat{\mathbf{C}}_\beta|\mathbf{Leak}_\beta) \geq \mathbb{H}_\infty(\hat{\mathbf{C}}_\beta) - \ell,$$

where $\mathbf{Leak}_\beta$ is the random variable corresponding to the leakage performed by the adversary $\mathsf{A}$, and where $(\hat{\mathbf{C}}_0, \hat{\mathbf{C}}_1)$ is the random variable corresponding to the simulated codeword in the ideal experiment with a canonical simulator. The above leads to the following definition.

**Definition 12** (Leakage-resilient continuous non-malleability)**.** Let $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a $(k,n)$-code in the CRS model, and let $n_0(\lambda) = n_0 \in \mathbb{N}$ and $n_1(\lambda) = n_1 \in \mathbb{N}$ be such that $n = n_0 + n_1$. We say that $\Sigma$ is $\ell$-leakage-resilient continuously $\mathcal{F}_{\mathsf{split}}^{n_0,n_1}$-non-malleable if for all $\ell$-admissible PPT adversaries $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1)$ there exists a *canonical* simulator $\mathsf{S} := (\mathsf{S}_0, \mathsf{S}_1)$ such that

$$\{\mathbf{LKReal}_{\Sigma,\mathsf{A},\mathcal{F}_{\mathsf{split}}^{n_0,n_1}}(\lambda, n_0, n_1)\}_{\lambda\in\mathbb{N}} \approx_{\mathrm{c}} \{\mathbf{LKSimu}_{\mathsf{S},\mathsf{A},\mathcal{F}_{\mathsf{split}}^{n_0,n_1}}(\lambda, n_0, n_1)\}_{\lambda\in\mathbb{N}}.$$

One can prove that the code from §A is already leakage resilient. In fact, the leakage resilience is inherited from the underlying leakage-resilient code. In particular, when proving an equivalent of Lemma 14, the reduction $\mathsf{B}$ can use the leakage queries $(\mathtt{leak}, (g_0, g_1))$ from the adversary $\mathsf{A}$, to define its own leakage queries $(g_0(\cdot, h_1, \pi_1), g_1(\cdot, h_0, \pi_0))$ that are forwarded to the challenger. Importantly, this step of the proof still involves a simulated codeword, and thus allows to exploit the fact that $\mathsf{A}$ is admissible.

The above discussion can be summarized in the following statement.

**Theorem 7.** *Let $\Sigma'$ be a $(k,n')$-code that is $(\lambda + k + \omega(\log(\lambda)) + \ell')$-noisy-leakage resilient, and assume that, for all $\beta \in \{0,1\}$ and for all $s \in \{0,1\}^k$, the following holds for the conditional average min-entropy of the random variable $(\mathbf{C}'_0, \mathbf{C}'_1)$ corresponding to $\mathsf{LREnc}(s)$: $\widetilde{\mathbb{H}}_\infty(\mathbf{C}'_\beta|\mathbf{C}'_{1-\beta}) \geq \mathbb{H}_\infty(\mathbf{C}'_\beta) - k$. Additionally, let $\Pi$ be a collision resistant hash function with range $\{0,1\}^\lambda$, and $\Pi'$ be NIZK proof systems for the NP-relation $\mathcal{R}_{\mathsf{hash}} = \{((hk,h),x) : \mathsf{H}(hk,x) = h\}$, satisfying adaptive multi-theorem zero-knowledge and simulation extractability.*

*Then the code $\Sigma$ described in §A is an $\ell'$-leakage-resilient continuously $\mathcal{F}_{\mathsf{split}}^{n/2,n/2}$-non-malleable $(k,n)$-code, where $n(\lambda) = n'(\lambda) + 2(\lambda + n_{\mathsf{nizk}}(\lambda))$, and where $n_{\mathsf{nizk}}(\lambda)$ is the size of a proof. Additionally, the distribution $(\hat{\mathbf{C}}_0, \hat{\mathbf{C}}_1)$ of the simulated codeword output by the canonical simulator is $(\lambda+k)$-correlated, meaning that, for all $\beta \in \{0,1\}$, one has $\widetilde{\mathbb{H}}_\infty(\hat{\mathbf{C}}_\beta|\hat{\mathbf{C}}_{1-\beta}) \geq \mathbb{H}_\infty(\hat{\mathbf{C}}_\beta) - (\lambda+k)$.*

The last part of the statement follows readily by the fact that the codewords of the underlying leakage-resilient code are $k$-correlated, whereas the hash values $h_0, h_1$ are $\lambda$-bit long.

### B.3 Security Analysis

We establish the following result.

**Theorem 8.** *Assume that $\Sigma$ is an $(\lambda + \ell)$-leakage-resilient continuously $\mathcal{F}_{\mathsf{split}}^{n_0,n_1}$-non-malleable $(d,n)$-code with $\ell$-correlated simulated codewords, and that $\Pi$ is a* secure *$(d,k,m)$-SKE scheme. Then $\Sigma'$ as defined in §B.1 is a continuously $\mathcal{F}_{\mathsf{split}}^{n_0+m,n_1+m}$-non-malleable $(k, 2m+n)$-code.*

*Proof sketch.* Since the proof follows very closely that proof of Theorem 3, we only highlight the main differences between the two proofs. Let $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1)$ be the canonical simulator guaranteed by the underlying leakage-resilient continuously non-malleable code. The simulator $\mathsf{S}' = (\mathsf{S}'_0, \mathsf{S}'_1)$ works as follows.

$\underline{\mathsf{S}'_0(1^\lambda)}$:

1. Run $(\omega, \sigma) \leftarrow\!\!\$\ \mathsf{S}_0(1^\lambda)$.
2. Sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, and let $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, 0^k)$.
3. Set $\sigma' := (\sigma, \kappa, \gamma)$ and return $(\omega, \sigma')$

$\underline{\mathsf{S}'_1(\sigma', (f_0, f_1))}$:

1. Parse $\sigma' := (\sigma, \kappa, \gamma)$;
2. For all $i \in [m]$ sample $(c_{i,0}, c_{i,1}) \leftarrow\!\!\$\ \mathsf{S}_1(\sigma, (\mathtt{leak}, (\mathsf{L}_0^{\gamma,f_0,i}(\cdot), \mathsf{L}_1^{\gamma,f_1,i}(\cdot))$, where for all $\beta \in \{0,1\}$ the function $\mathsf{L}_\beta^{\gamma,f_\beta,i}(c_\beta)$ computes $(\tilde{c}_\beta, \tilde{\gamma}) = f_\beta(c_\beta, \gamma)$ and then outputs the $i$-th bit of $\tilde{\gamma}$. In case $(c_{i,0} \neq c_{i,1})$ break the cycle, and set $\tilde{\gamma} := \bot$.
3. If $\tilde{\gamma} = \bot$, return $\bot$ and self-destruct. Else, set $\tilde{\gamma} := (c_{0,0}, \ldots, c_{m,0})$;
4. Let $\tilde{\kappa} \leftarrow\!\!\$\ \mathsf{S}_1(\sigma, (\mathtt{tamp}, f'_0, f'_1), \hat{c}_1)$, where $f'_\beta(\cdot) := f_\beta(\cdot, \gamma)$.
5. If $\tilde{\kappa} = \diamond$ and $\tilde{\gamma} \neq \gamma$, let $\tilde{s} := \bot$ (and self-destruct).
6. Else, if $\tilde{\kappa} = \diamond$ and $\tilde{\gamma} = \gamma$, let $\tilde{s} := \diamond$.
7. Else, let $\tilde{s} := \mathsf{ADec}(\tilde{\kappa}, \tilde{\gamma})$.

We consider exactly the same sequence of hybrid experiments $\mathbf{H}_0(\lambda)$–$\mathbf{H}_3(\lambda)$ as for the proof of Theorem 3, where $\mathbf{H}_0(\lambda) \equiv \mathbf{Real}_{\Sigma',\mathsf{A}',cF'}$ and $\mathbf{H}_3(\lambda) \equiv \mathbf{Simu}_{\mathsf{S}',\mathsf{A}',\mathcal{F}'}$ (for the above defined simulator $\mathsf{S}'$).

The proof of the lemmas below follows closely that of, respectively, Lemma 3 and Lemma 4, and is therefore omitted.

**Lemma 17.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{1,2} \in \mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda)) = 1\right]\right| \leq \nu_{1,2}(\lambda).$$

**Lemma 18.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{2,3} \in \mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_3(\lambda)) = 1\right]\right| \leq \nu_{2,3}(\lambda).$$

**Lemma 19.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{0,1} \in \mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_0(\lambda)) = 1\right] - \mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda)) = 1\right]\right| \leq \nu_{0,1}(\lambda).$$

*Proof.* We reduce to the security of the $(\lambda+\ell)$-leakage-resilient continuously $\mathcal{F}_{\mathsf{split}}$-non-malleable code. Consider the following adversary $\mathsf{B} = (\mathsf{B}_0, \mathsf{B}_1)$, where $\mathsf{B}_0(\omega)$ runs $(s, \alpha) \leftarrow \mathsf{A}_0(\omega)$ and sets $\alpha' := (s, \alpha)$:

$\underline{\text{Adversary } \mathsf{B}_1(\alpha')}$

1. Parse $\alpha'$ as $(s, \alpha)$.
2. Sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, and let $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, s)$.
3. Run the adversary $\mathsf{A}_1(\alpha)$, and upon each adaptive query $(f_0, f_1)$:
   (a) For all $i \in [m]$ make a leakage query $(\mathtt{leak}, (\mathsf{L}_0^{\gamma,f_0,i}(\cdot), \mathsf{L}_1^{\gamma,f_1,i}(\cdot)))$, obtaining an answer $(c_{i,0}, c_{i,1})$; in case $(c_{i,0} \neq c_{i,1})$ break the cycle and set $\tilde{\gamma} := \bot$.
   (b) If $\tilde{\gamma} = \bot$, return the message $\bot$ to $\mathsf{A}_1$ and self-destruct; else, let $\tilde{\gamma} := (c_{0,0}, \ldots, c_{m,0})$.
   (c) Let $\tilde{\kappa} \leftarrow\!\!\$\ \mathsf{S}_1(\sigma, (\mathtt{tamp}, f'_0, f'_1), \hat{c}_1)$, where $f'_\beta(\cdot) := f_\beta(\cdot, \gamma)$.
   (d) If $\tilde{\kappa} = \diamond$ and $\tilde{\gamma} \neq \gamma$, return $\bot$ to $\mathsf{A}_1$ and self-destruct.

(e) Else, if $\tilde{\kappa} = \diamond$ and $\tilde{\gamma} = \gamma$, return $\diamond$ to $\mathsf{A}_1$.
(f) Else, return $\mathsf{ADec}(\tilde{\kappa}, \tilde{\gamma})$ to $\mathsf{A}_1$.

By inspection, the above simulation is perfect in the sense that $\mathsf{B}$ perfectly simulates the experiment $\mathbf{H}_1$ when running in the ideal experiment **LKSimu**, whereas it simulates perfectly the experiment $\mathbf{H}_0$ when running in the real experiment **LKReal**. Hence, it remains to prove that $\mathsf{B}$ is a $(\lambda + \ell)$-admissible according to Definition 12.

Let $q(\lambda) \in \mathtt{poly}(\lambda)$ be an upper bound on the number of tampering queries asked by $\mathsf{A}$. Furthermore, denote by $c^j_{i,\beta}$ the value $c_{i,\beta}$ as computed by $\mathsf{B}_1$ while answering the $j$-th tampering query of $\mathsf{A}_1$. We write $\mathbf{j}^*$ for the random variable that indicates the index $j^* \leq q$ corresponding to the tampering query where a self-destruct is triggered (if any), and $\mathbf{i}^*$ for the random variable corresponding to the first index $i^* \leq m$ such that $c^j_{i,0} \neq c^j_{i,1}$ for some $j \leq \mathbf{j}^*$ (in case no such index exists, we assume $i^* = m$). Finally, let $\mathbf{Leak}_\beta$ be the random variable corresponding to the leakage performed by the reduction, which consists of a sequence of ciphertexts $((c^0_{0,0}, c^0_{0,1}, \ldots, c^{\mathbf{j}^*}_{\mathbf{i}^*,0}, c^{\mathbf{j}^*}_{\mathbf{i}^*,1})$.

For all $\beta \in \{0,1\}$, we can write

$$\begin{aligned}
\widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta | \mathbf{Leak}_\beta\right) &= \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta | \mathbf{C}^0_{0,0}, \mathbf{C}^0_{0,1}, \ldots, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,1}\right) \\
&= \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta | \mathbf{C}^0_{0,1-\beta}, \ldots, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*-1,1-\beta}, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}\right) && (5) \\
&\geq \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta | \hat{\mathbf{C}}_{1-\beta}, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}, \mathbf{j}^*, \mathbf{i}^*\right) && (6) \\
&\geq \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta | \hat{\mathbf{C}}_{1-\beta}\right) - 1 - \log q(\lambda) - \log m(\lambda) && (7)
\end{aligned}$$

where Eq. (5) comes from the definition of the simulator $\mathsf{S}'$, Eq. (6) follows by Lemma 10, and Eq. (7) comes from Lemma 11 together with the fact that $(\mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}, \mathbf{C}^{\mathbf{j}^*}_{\mathbf{i}^*,0}) \in \{01, 10\}$. The statement now follows by the fact that simulated codewords are $\ell$-correlated, and moreover, for large enough $\lambda \in \mathbb{N}$, we have $1 + \log m + \log q \leq \lambda$. □

The statement of the theorem now follows by the above lemmas and the triangle inequality. □

# C Split-State Rate-One Compiler with Adaptive Security

## C.1 Description of the Compiler

Let $\Sigma = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ be a rate-zero $(d, n)$-code, and $\Pi = (\mathsf{KGen}, \mathsf{AEnc}, \mathsf{ADec})$ be a $(d, k, m)$-SKE scheme. Consider the following construction of a $(k, n')$-code $\Sigma' = (\mathsf{Init}', \mathsf{Enc}', \mathsf{Dec}')$ in the random oracle model, where $n' := m + n + \lambda$.

$\mathsf{Init}'(1^\lambda)$**:** run the same as $\mathsf{Init}(1^\lambda)$.
$\mathsf{Enc}'(\omega', s)$**:** Upon input $\omega'$ and a value $s \in \{0,1\}^k$, sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, compute $c \leftarrow\!\!\$\ \mathsf{Enc}(\omega, \kappa)$, $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, s)$, and $h = H(c)$; parse $c$ as $c_0||c_1$, and return $c' := (c'_0, c'_1) = ((c_0, h), (c_1, \gamma))$.
$\mathsf{Dec}'(\omega', c')$**:** Parse $c' := ((c_0, h), (c_1, \gamma))$. If $h \neq H(\gamma)$, return $\perp$ and self destruct; else let $\tilde{\kappa} = \mathsf{Dec}(\omega', c_0||c_1)$. If $\tilde{\kappa} = \perp$, return $\perp$ and self destruct; else, let $\tilde{\mu} = \mathsf{ADec}(\tilde{\kappa}, \gamma)$. If $\tilde{\mu} = \perp$ return $\perp$ and self destruct, else return $\tilde{\mu}$.

The only difference with the compiler presented in §B is that the ciphertext $\gamma$ is stored on the right share of the codeword, whereas a digest of the ciphertext $h$ (computed via the random oracle) is stored on the left share.

## C.2 Security Analysis

We establish the following result.

**Theorem 9.** *Assume that $\Sigma$ is a $(2\lambda+\ell)$-leakage-resilient continuously $\mathcal{F}_{\mathsf{split}}^{n_0,n_1}$-non-malleable $(d,n)$-code with a canonical simulator outputting simulated codewords that are $\ell$-correlated, and that $\Pi$ is a* secure *$(d,k,m)$-SKE scheme. Then $\Sigma'$ as defined in §C.1 is a continuously $\mathcal{F}_{\mathsf{split}}^{n_0+m,n_1+m}$-non-malleable $(k,m+n+\lambda)$-code in the non-programmable random oracle model.*

*Proof.* Since the proof follows very closely that proof of Theorem 3, we only highlight the main differences. Let $\mathsf{S}=(\mathsf{S}_0,\mathsf{S}_1)$ be the canonical simulator guaranteed by the underlying leakage-resilient continuously non-malleable code. The simulator $\mathsf{S}'=(\mathsf{S}'_0,\mathsf{S}'_1)$ works as follows.

$\underline{\mathsf{S}'_0(1^\lambda)}$:

1. Run $(\omega,\sigma)\leftarrow\!\!\$\ \mathsf{S}_0(1^\lambda)$;
2. Sample $\kappa\leftarrow\!\!\$\ \{0,1\}^d$, and let $\gamma\leftarrow\!\!\$\ \mathsf{AEnc}(\kappa,0^k)$ and $h=H(\gamma)$;
3. Set $\sigma':=(\sigma,\kappa,\gamma,h)$ and return $(\omega,\sigma')$;

$\underline{\mathsf{S}'_1(\sigma',(f_0,f_1))}$:

1. Parse $\sigma':=(\sigma,\kappa,\gamma,h)$;
2. Sample $(\tilde{h}_0,\tilde{h}_1)\leftarrow\!\!\$\ \mathsf{S}_1(\sigma,(\mathtt{leak},(\mathsf{L}_0^{\gamma,f_0}(\cdot),\mathsf{L}_1^{\gamma,f_1}(\cdot))))$, where the function $\mathsf{L}_0^{\gamma,f_0}(c_0)$ computes $(\tilde{c}_0,\tilde{h})=f_0(c_0,h)$ and outputs $\tilde{h}$, whereas the function $\mathsf{L}_1^{\gamma,f_1}(c_1)$ computes $(\tilde{c}_1,\tilde{\gamma})=f_1(c_1,\gamma)$ and outputs $H(\tilde{\gamma})$.
3. In case $(\tilde{h}_0\neq\tilde{h}_1)$ set $\tilde{\gamma}:=\perp$, else sample $(\varepsilon,\tilde{\gamma})\leftarrow\!\!\$\ \mathsf{S}_1(\sigma,(\mathtt{leak},(\mathsf{null},\mathsf{L}_2^{\gamma,f_1}(\cdot))$, where the function $\mathsf{L}_2^{\gamma,f_1}(c_1)$ computes $(\tilde{c}_1,\tilde{\gamma})=f_1(c_1,\gamma)$ and then outputs $\tilde{\gamma}$ while the function $\mathsf{null}$ simply outputs the empty string $\varepsilon$.
4. Let $\tilde{\kappa}\leftarrow\!\!\$\ \mathsf{S}_1(\sigma,(\mathtt{tamp},f'_0,f'_1),\hat{c}_1)$, where $f'_0(\cdot)$ computes $(\tilde{c}_0,\tilde{h})=f_0(\cdot,h)$ and outputs $\tilde{c}_0$, whereas $f'_1(\cdot)$ computes $(\tilde{c}_1,\tilde{\gamma})=f_1(\cdot,\gamma)$ and outputs $\tilde{c}_1$.
5. If $\tilde{\kappa}=\diamond$ and $\tilde{\gamma}\neq\gamma$, let $\tilde{s}:=\perp$ (and self-destruct).
6. Else, if $\tilde{\kappa}=\diamond$ and $\tilde{\gamma}=\gamma$, let $\tilde{s}:=\diamond$.
7. Else, let $\tilde{s}:=\mathsf{ADec}(\tilde{\kappa},\tilde{\gamma})$.

We consider exactly the same sequence of hybrid experiments $\mathbf{H}_0(\lambda)$–$\mathbf{H}_3(\lambda)$ as for the proof of Theorem 3, where $\mathbf{H}_0(\lambda)\equiv\mathbf{Real}_{\Sigma',\mathsf{A}',cF'}$ and $\mathbf{H}_3(\lambda)\equiv\mathbf{Simus}_{\mathsf{S}',\mathsf{A}',\mathcal{F}'}$ (for the above defined simulator $\mathsf{S}'$).

The proof of the lemmas below follows closely that of, respectively, Lemma 3 and Lemma 4, and is therefore omitted.

**Lemma 20.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{1,2}\in\mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda))=1\right]-\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda))=1\right]\right|\leq\nu_{1,2}(\lambda).$$

**Lemma 21.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{2,3}\in\mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_2(\lambda))=1\right]-\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_3(\lambda))=1\right]\right|\leq\nu_{2,3}(\lambda).$$

**Lemma 22.** *For all PPT adversaries $\mathsf{A}'$ there exists a function $\nu_{0,1}\in\mathtt{negl}(\lambda)$ such that for all PPT distinguishers $\mathsf{D}'$ we have:*

$$\left|\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_0(\lambda))=1\right]-\mathbb{P}\left[\mathsf{D}'(\mathbf{H}_1(\lambda))=1\right]\right|\leq\nu_{0,1}(\lambda).$$

*Proof.* We augment the underlying rate-zero code by allowing tampering functions $(f_0, f_1)$ to make random-oracle queries. This is fine since $\Sigma$ is independent of the random oracle, and additionally the code admits a canonical simulator, which can easily answer any random-oracle query asked by the tampering functions on its own (e.g., by using a PRF with a known key).

The proof is down to the security of the $(2\lambda + \ell)$-leakage-resilient continuously $\mathcal{F}_{\mathsf{split}}$-non-malleable code. Consider the following adversary $\mathsf{B} = (\mathsf{B}_0, \mathsf{B}_1)$, where $\mathsf{B}_0(\omega)$ runs $(s, \alpha) \leftarrow \mathsf{A}_0(\omega)$ and sets $\alpha' := (s, \alpha)$:

Adversary $\mathsf{B}_1(\alpha')$

1. Parse $\alpha'$ as $(s, \alpha)$.
2. Sample $\kappa \leftarrow\!\!\$\ \{0,1\}^d$, and let $\gamma \leftarrow\!\!\$\ \mathsf{AEnc}(\kappa, s)$.
3. Run the adversary $\mathsf{A}_1(\alpha)$; answer any random-oracle query by forwarding it to the random oracle $H$. Upon input a tampering query $(f_0, f_1)$ from $\mathsf{A}_1$, answer as follows:
   (a) Make a leakage query $(\mathtt{leak}, (\mathsf{L}_0^{\gamma, f_0}(\cdot), \mathsf{L}_1^{\gamma, f_1}(\cdot)))$, obtaining an answer $(\tilde{h}_0, \tilde{h}_1)$;
   (b) In case $\tilde{h}_0 \neq \tilde{h}_1$, set $\tilde{\gamma} := \perp$, return $\perp$ to $\mathsf{A}_1$, and self destruct.
   (c) Let $\mathcal{Q}_H$ be the set of random-oracle queries asked by $\mathsf{A}_1$; check that there exists a query $x$ such that $H(x) = \tilde{h}_0$. If no such value is found, return $\perp$ to $\mathsf{A}_1$ and self destruct.
   (d) Make the leakage query $(\mathtt{leak}, (\mathsf{null}, \mathsf{L}_2^{\gamma, f_1}(\cdot)))$, obtaining an answer $(\varepsilon, \tilde{\gamma})$.
   (e) If $H(\tilde{\gamma}) = H(x)$, but $x \neq \tilde{\gamma}$ then abort.
   (f) Make the tampering query $(\mathtt{tamp}, (f_0', f_1'), \hat{c}_1)$, obtaining an answer $\tilde{\kappa}$.
   (g) If $\tilde{\kappa} = \kappa$ and $\tilde{\gamma} \neq \gamma$, return $\perp$ to $\mathsf{A}_1$ and self-destruct.
   (h) Else, if $\tilde{\kappa} = \kappa$ and $\tilde{\gamma} = \gamma$, return $\mu$ to $\mathsf{A}_1$.
   (i) Else, return $\mathsf{ADec}(\tilde{\kappa}, \tilde{\gamma})$ to $\mathsf{A}_1$.

We claim that the above simulation is almost perfect, in the sense that with all but a negligible probability the reduction correctly emulates either the view in experiment $\mathbf{H}_1$ (if it is running in experiment **LKSimu**), or the view in experiment $\mathbf{H}_0$ (if it is running in experiment **LKReal**). The only differences are in the checks performed by the reduction in step 3c and step 3e. In particular, the reduction outputs $\perp$ and simulates a self-destruct whenever $\mathsf{A}_1$ sets correctly $H(x) = \tilde{h}_0$ without querying $x$ to the random oracle; since the latter can only happen with probability at most $2^{-\lambda}$, the simulation performed by the reduction is correct with probability at least $1 - 2^{-\lambda}$. As a consequence, we must have that $x = \tilde{\gamma}$ in step 3e since otherwise $\mathsf{A}_1$ found a collision in the random oracle, which can only happen with negligible probability.

It remains to prove that $\mathsf{B}$ is a $(\lambda + \ell)$-admissible according to Definition 12. Let $q(\lambda) \in \mathtt{poly}(\lambda)$ be an upper bound on the number of tampering queries asked by $\mathsf{A}$. Furthermore, denote by $\tilde{h}_\beta^j$ (resp. $\tilde{\gamma}^j$) the value $\tilde{h}_\beta$ (resp. $\tilde{\gamma}^j$) as computed by $\mathsf{B}_1$ while answering the $j$-th tampering query of $\mathsf{A}_1$. We write $\mathbf{j}^*$ for the random variable that indicates the index $j^* \leq q$ corresponding to the tampering query where a self-destruct is triggered (if any). Finally, let **Leak** be the random variable corresponding to the leakage performed by the reduction, which consists of values $(\tilde{h}_0^1, \tilde{h}_1^1, \tilde{\gamma}^1, \ldots, \tilde{h}_0^{\mathbf{j}^*}, \tilde{h}_1^{\mathbf{j}^*})$, and let $\mathbf{Leak}_\beta$ correspond to $(\tilde{h}_\beta^1, \tilde{\gamma}^1, \ldots, \tilde{h}_\beta^{\mathbf{j}^*})$.

We compute the conditional average min-entropy of each share of a codeword conditioned on the leakage performed by the reduction and on the list of random oracle queries (and answers

to these queries) $\mathcal{Q}_H$. For all $\beta \in \{0,1\}$, we can write

$$\begin{aligned}
\widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta|\mathbf{Leak}_\beta, \mathcal{Q}^H\right) &= \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta|(\tilde{h}^1_\beta, \tilde{\gamma}^1), \ldots, (\tilde{h}^{\mathbf{j}^*-\mathbf{1}}_\beta, \tilde{\gamma}^{\mathbf{j}^*-\mathbf{1}}), \tilde{h}^{\mathbf{j}^*}_\beta, \mathcal{Q}^H, \mathbf{j}^*\right) \\
&= \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta|\tilde{h}^1_\beta, \ldots, \tilde{h}^{\mathbf{j}^*-\mathbf{1}}_\beta, \tilde{h}^{\mathbf{j}^*}_\beta, \mathcal{Q}^H, \mathbf{j}^*\right) && (8) \\
&= \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta|\tilde{h}^1_\beta, \ldots, \tilde{h}^{\mathbf{j}^*-\mathbf{1}}_\beta, \tilde{h}^{\mathbf{j}^*}_\beta, \mathbf{j}^*\right) && (9) \\
&\geq \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta|\tilde{h}^1_{\beta-1}, \ldots, \tilde{h}^{\mathbf{j}^*-\mathbf{1}}_{\beta-1}, \tilde{h}^{\mathbf{j}^*}_\beta, \mathcal{Q}^H, \mathbf{j}^*\right) && (10) \\
&\geq \widetilde{\mathbb{H}}_\infty\left(\hat{\mathbf{C}}_\beta|\hat{\mathbf{C}}_{1-\beta}\right) - \log q(\lambda) - \lambda. && (11)
\end{aligned}$$

Eq. (8) holds because in step 3c the adversary $\mathsf{B}$ checks that $\mathsf{A}_1$ asks a query $x$ to the random oracle such that $H(x) = \tilde{h}_\beta$, and in step 3e the reduction ensures that $x = \tilde{\gamma}$, so that the values $\tilde{\gamma}^1, \ldots \tilde{\gamma}^{\mathbf{j}^*-1}$ are redundant. Eq. (9) comes from the fact that the queries $\mathcal{Q}_H$ are independent from the codeword $\hat{\mathbf{C}}_\beta$. Eq. (10) follows by definition of the simulator $\mathsf{S}'$, as for $j < \mathbf{j}^*$ we have $\tilde{h}^j_0 = \tilde{h}^j_1$. Eq. (10) holds by applying Lemma 10 and Lemma 11.

The statement now follows by the fact that simulated codewords are $\ell$-correlated, and moreover, for large enough $\lambda \in \mathbb{N}$, we have $\lambda + \log q \leq 2\lambda$. □

The theorem follows by the above lemmas and the triangle inequality. □