Explicit Rate-1 Non-malleable Codes for Local Tampering

Divya Gupta[1], Hemanta K. Maji[2], and Mingyuan Wang[2(✉)]

[1] Microsoft Research, Bangalore, India
divya.gupta@microsoft.com
[2] Department of Computer Science, Purdue University, West Lafayette, USA
{hmaji,wang1929}@purdue.edu

Abstract. This paper constructs high-rate non-malleable codes in the information-theoretic plain model against tampering functions with bounded locality. We consider δ-local tampering functions; namely, each output bit of the tampering function is a function of (at most) δ input bits. This work presents the first explicit and efficient rate-1 non-malleable code for δ-local tampering functions, where $\delta = \xi \lg n$ and $\xi < 1$ is any positive constant. As a corollary, we construct the first explicit rate-1 non-malleable code against NC^0 tampering functions.

Before our work, no explicit construction for a constant-rate non-malleable code was known even for the simplest 1-local tampering functions. Ball et al. (EUROCRYPT–2016), and Chattopadhyay and Li (STOC–2017) provided the first explicit non-malleable codes against δ-local tampering functions. However, these constructions are rate-0 even when the tampering functions have 1-locality. In the CRS model, Faust et al. (EUROCRYPT–2014) constructed efficient rate-1 non-malleable codes for $\delta = O(\log n)$ local tampering functions.

Our main result is a general compiler that bootstraps a rate-0 non-malleable code against leaky input and output local tampering functions to construct a rate-1 non-malleable code against $\xi \lg n$-local tampering functions, for any positive constant $\xi < 1$. Our explicit construction instantiates this compiler using an appropriate encoding by Ball et al. (EUROCRYPT–2016).

1 Introduction

Dziembowski, Pietrzak, and Wichs [18] introduced the notion of non-malleable codes as an extension of the standard objective of error-correction. Non-malleable codes provide message-integrity assurances even when error-detection, let alone error-correction, is impossible. Suppose a sender encodes a message $m \in \{0,1\}^\ell$ and transmits the codeword over a channel. If the channel adds an error that

H. K. Maji—The research effort is supported in part by an NSF CRII Award CNS–1566499, an NSF SMALL Award CNS–1618822, and an REU CNS–1724673.
H. K. Maji and M. Wang—The research effort is supported in part by a Purdue Research Foundation grant.

A. Boldyreva and D. Micciancio (Eds.): CRYPTO 2019, LNCS 11692, pp. 435–466, 2019.
https://doi.org/10.1007/978-3-030-26948-7_16

has a small Hamming weight, then the sender can encode the message using an error-correcting code and the receiver can error-correct and retrieve the original message. Algebraic Manipulation Detection codes [16] help the receiver detect if the transmitted codeword is tampered using algebraic operations. For more sophisticated classes of tampering function $\mathcal{F}$, demanding manipulation detection or error-correction might be far-fetched. For example, suppose the channel replaces the original codeword with a fixed valid codeword. In this case, error-correction or error-detection is impossible. Non-malleable codes provide a meaningful message integrity assurance against sophisticated tampering families.

Let us fix an encoding and a decoding scheme $(\mathsf{Enc}, \mathsf{Dec})$, and a tampering function family $\mathcal{F}$. Non-malleable codes ensure that for any message $m \in \{0,1\}^{\ell}$ and tampering function $f \in \mathcal{F}$, the tampered message $\mathsf{Dec}(f(\mathsf{Enc}(m)))$ is either identical to the original message m or a simulator Sim_f can simulate this distribution (that is, it is independent of the original message). Even such a weak message integrity assurance turns out to be cryptographically useful, for example, in storing secret-keys for cryptographic primitives [18,28] and non-malleable messaging [22,23]. Naturally, we measure the quality of non-malleable codes using the following two parameters.

1. **Rate.** The ratio of the length of the message to the length of its encoding.
2. **Sophistication of the tampering family.** The complexity of the tampering attacks captured by the tampering functions in this family.

Constructing explicit non-malleable codes with high rate against sophisticated tampering function families is the guiding principle for the research in non-malleable codes. However, both these objectives, even independently, have been significantly non-trivial to achieve. Only recently, using elegant probabilistic arguments, [13,20] constructed rate-1 non-malleable codes in the *CRS model* for tampering families of bounded size.[1]

In this paper, for any positive constant $\xi < 1$, we present the first *rate-1* explicit non-malleable codes against any tampering function that has $\xi \lg n$ *output locality*, i.e., at most $\xi \lg n$ input-bits influence any output bit of the tampering function. Note that there is no bound on the input locality, i.e., the number of output positions one input bit can influence during tampering. Here lg represents the logarithm with base 2, and n represents the length of the codeword. Notably, our construction is in the information-theoretic plain model. We emphasize that our construction does not rely on any computational hardness assumption or a CRS.

1.1 Prior Relevant Works

Note that it is impossible to construct a non-malleable code (NMC) that is secure for all tampering functions. Consider a tampering function that obtains an advantage in predicting the first bit m_1 of the message m. Then, the tampering function f overwrites the codeword with a *fixed* encoding of m_1^{ℓ}, which we

[1] Tampering functions can access the CRS; however, they cannot tamper the CRS.

hardwire into it. This result indicates that, given the codeword c, no tampering function $f \in \mathcal{F}$ should have an advantage in predicting any bit of the message $m \in \{0,1\}^{\ell}$. Consequently, we fix the class of tampering function family $\mathcal{F}$, and construct a non-malleable encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ for that tampering family.

Monte-Carlo Constructions. Dziembowski et al. [18], introduced the notion of non-malleable codes and showed the existence of rate-1 NMC against bit-wise tampering (each output bit is a function of the corresponding input bit). Next, Faust et al. [20] showed the existence of rate-1 non-malleable codes against any tampering family of size $2^{\mathrm{poly}(n)}$.[2] Cheraghchi and Guruswami [13] proved that there exists (possibly inefficient) $(1-\alpha)$-rate NMC against any tampering function family of size $2^{2^{\alpha n}}$. These results are probabilistic in nature and it is unknown whether we can derandomize them to obtain explicit constructions in the plain model. Note that there are (roughly) $\left(\binom{n}{\delta}2^{2^{\delta}}\right)^{n} \approx 2^{n2^{\delta}}$ distinct δ-local tampering functions. If $\delta = O(\log n)$ then [20] indicates the existence of an efficient rate-1 non-malleable code. Further, for $\delta = o(n)$, [13] implies the existence of a (possibly inefficient) rate-1 non-malleable code.

Explicit Constructions. A famous line of research explores designing NMCs against k-split-state tampering functions where k different locations store the k shares of the encoding. The tampering of each share is performed arbitrarily, albeit independently. The maximum achievable rate in this setting is $R \leqslant 1 - 1/k$ [13]. Dziembowski et al. [17] constructed NMC for one-bit messages in the 2-split-state model. A sequence of highly influential works have constructed near-optimal constant-rate NMCs using only $k = 3$ shares of the encoding [2,3,10,12,24,27,28]. Currently, Li's construction [29,30] achieves the highest rate $R = O(\log\log\log n/\log\log n)$ for 2-split-state tampering.

Another research direction allows the tampering function to tamper the entire codeword but constrains its computational power. For example, local tampering functions have an a priori upper bound of how many input bits influence each output bit. [7,11] construct explicit NMC for local tampering functions. The rate of [7] is at most the product of inverse of locality and rate of the 2-split-state NMC. Hence, their construction has rate-0 even for constant locality. Recently, [6] consider constant depth circuits and construct the first explicit NMC against AC^0 tampering. Both [11] and [6] have inverse-polynomial rate.

Explicit Rate-1 Constructions. Explicit rate-1 NMC constructions are even more scarce. Cheraghchi and Guruswami [14,18] construct a rate-1 NMC for bit-wise tampering. Agrawal et al. [4,5] provide an explicit rate-1 NMC against tampering functions that perform bit-wise tampering after permuting the input bits. Both these constructions amplify the rate of a base NMC (possibly, with additional properties) that has sub-optimal rate into a rate-1 NMC using a compiler. We emphasize that these two particular tampering families are not only 1-local but also have the constraint that each input bit influences at most

[2] This construction is also an efficient rate-1 construction in the CRS model.

one output bit (that is, 1-input local). Note that the focus of this work is δ-local tampering functions with $\delta > 1$ and no bound on input locality.

In the computationally bounded setting, [1] construct a rate-1 NMC against 2-split-state tampering based on the existence of one-way functions. Furthermore, there are constructions of NMC that rely on a CRS [8,20,31].

Non-malleable codes have also been considered in the continual tampering model, for example, [19,26,33]; covering which is beyond the scope of this work.

1.2 Our Contribution

Our work focuses on constructing non-malleable codes, in the information-theoretic plain model, against tampering functions that are δ-local, i.e., at most δ input bits influence any output bit. We emphasize that δ can be a function of n, the size of the codeword. Our work, for any positive constant $\xi < 1$, constructs explicit rate-1 NMC against δ-local tampering functions, where $\delta = \xi \lg n$, which has a tampering family of size $2^{n^{1+o(1)}}$. In our case, the locality $\delta = \omega(1)$ and, hence, the set of all δ-local tampering functions subsumes the family of NC^0 tampering functions.

We present a general black-box compiler that takes three ingredients as input and constructs a non-malleable code for local functions. At an intuitive level, we prove the following result.

Informal Theorem 1. *For any positive constant $\xi < 1$, there exists an explicit and efficient rate-1 NMC against $\xi \lg n$-local tampering functions using the following primitives in a black-box manner (refer to Fig. 2).*

1. *Rate-1 linear error-correcting code*[3] *with (near) linear distance and dual-distance (see Definition 7),*
2. *Rate-$1/\eta^{o(1)}$ NMC against leaky input and output local tampering for message length η (referred to as the base NMC) (see Definition 6), and*
3. *A pseudorandom generator for finite state machines with super-polynomial stretch (see Definition 9).*

The compiler (refer to Fig. 1 for an outline) encodes the message m using the error-correcting code. Then, it samples a few entries of the codeword (at a suitable rate) and adds errors at half of them. The compiler tabulates all the sampled entries (both the erroneous and unaltered ones) along with their respective locations. The erroneous codeword forms the primary payload of the message m. The list of tabulated entries is appropriately encoded using a combination of the base NMC and the PRG and is juxtaposed (at the end) for consistency checks during decoding. If the rate of subsampling is sufficiently low, then the overall construction is rate-1. The security argument proceeds by demonstrating that if the subsampling rate is sufficiently high, then any local function cannot change

[3] Error-correcting codes can be converted into error-correcting secret sharing schemes using standard share-packing techniques [9,21,34].

the payload without being inconsistent with the tabulated entries themselves. Section 1.3 provides an intuitive overview of our compiler's construction.

Finally, we instantiate the respective primitives using (1) Reed-Solomon Codes over characteristic 2 fields, (2) An appropriate encoding introduced by Ball et al. [7], and (3) Nisan's PRG [32]. As a consequence, we construct explicit efficient rate-1 NMC against $\xi \lg n$-local tampering functions, for any positive constant $\xi < 1$, with negligible simulation error (refer Theorem 2).

Remark. We note that the resulting decoding function for our construction is randomized. However, the randomization stems solely from the randomized decoding function of the base NMC construction of [7]. Given an appropriate NMC against leaky input and output local tampering with deterministic decoding, our construction will have deterministic decoding.

Remark. If the base NMC is only rate-$1/\operatorname{poly} n$, then our compiler with suitably modified parameters, constructs an explicit rate-1 NMC against $o(\log n)$-local tampering functions. We defer this modification to the full version.

1.3 Technical Overview

As a starting point, it is instructive to understand the construction of Agrawal et al. [5] for a rate-1 NMC against tampering functions with input and output locality 1. The conceptual hurdles in generalizing this approach to δ-local functions, we believe, motivates the components used in our construction.

Construction of Agrawal et al. [5]. The construction of Agrawal et al. [5] encodes the message m with an error correcting secret sharing (ECSS) scheme to obtain a. Then, it samples a small number of bits from a indexed by E, which are represented by a_E, and replaces a_E with a (uniformly random) error e. This creates an erroneous codeword c. Observe that half of the bits of e match the original entries in a_E and the remaining do not. Next, an NMC of rate-$1/\operatorname{poly}$ encodes the consistency checks (E, e) as c_{err}, and the final encoding is (c, c_{err}). The decoding algorithm error-corrects c to obtain a (and hence, m) and checks the *consistency* between a, c, c_{err}. For an appropriately chosen size of the set E, the encoding (c, c_{err}) is non-malleable and has rate-1.

We represent the tampered codeword and error, respectively, by $\widetilde{c}$ and $\widetilde{c_{\text{err}}}$. The security argument proceeds, roughly, as follows.

(1) The tampering on c_{err} is independent of the message m. This argument crucially relies on the output-locality of the tampering function. The independence[4] of the ECSS is sufficiently high to permit the simulation of the tampering on c_{err} independent of the message m.

(2) The non-malleability of the encoding c_{err} ensures that $\widetilde{c_{\text{err}}}$ encodes either (a) the original (E, e), or (b) an entirely unrelated (E^*, e^*). The case of the tampering function creating an invalid encoding is not particularly insightful.

[4] An ECSS of independence t has the property that any t shares are uniformly and independently random.

(3.a.) Consider the case where the tampering function preserves error; namely, the same* case. In this case, they argue that the only way to get a valid tampered codeword is by keeping $\widetilde{c}$ identical to c and that the probability of encoding being valid independent of the original message m. For this argument, they perform a case analysis based on the number of bits that the tampering function *does not* directly copy from the codeword (a.k.a., the *not-copied-bits*). The tampering function, by definition, directly *copies* the remaining bits from the codeword into the tampered codeword.

If the number of not-copied-bits in the tampering function is small, then the simulation proceeds as follows. Since the tampering function has a small number of not-copied-bits, most bits in $\widetilde{c}$ are identical to their corresponding bits in c. These copied bits define a unique codeword (using the high distance property of ECSS[5]). Decoding succeeds if every not-copied-bit of $\widetilde{c}$ matches the corresponding bit in c. Moreover, decoding fails if any not-copied bit of $\widetilde{c}$ does not match the corresponding input bit in c. Since, the number of the not-copied-bits is small and they have output locality 1, we can simulate this check independent of the original message m by leveraging the (sufficiently large) independence of the ECSS.

On the other hand, if the number of not-copied-bits is large, then they argue that the tampered codeword is invalid (w.h.p.). The following intuition underlies their argument. Due to the input-locality 1 of the tampering functions, the error c_{err} can influence only a few bits in $\widetilde{c}$. Consequently, there still remains a large number of bits in $\widetilde{c}$ that are not-copied-bits and are not influenced by c_{err}. Therefore, the subset of these bits that is sampled in E is also large (over the random choice of E). Among these indices, leveraging the high independence of the ECSS and input locality 1 of the tampering function, there is a large subset where each indexed bit in the tampered codeword independently disagrees with the tabulated (E, e) with probability (at least)[6] 1/2. So, with high probability, the tampered codeword fails the consistence check.

(3.b.) Consider the case where the tampering function replaces the error with an unrelated (E^*, e^*). In this case, they argue that the only way to get valid tampered codeword is by replacing c by an unrelated c^* that is consistent with (E^*, e^*). For this argument, they perform a case analysis based on the number of output-bits of the tampering function that are non-constant (a.k.a., the *non-constant-bits*). If the number of non-constant-bits is small, then the tampered message is simulatable independent of the message due to the high independence of the ECSS and output locality 1 of tampering function. On the other hand, if the number of non-constant-bits is large, then the decoding fails with high probability. In this case, each bit in $\widetilde{c}$ that is influenced by a bit in c risks creating an independent inconsistency with (E^*, e^*) with probability 1/2. Hence, if there is a large number of these bits where each of them is inconsistent

[5] An ECSS with distance d ensures that, for two different secrets, at least d secret shares are different.

[6] If the tampering function flips the input bit then the probability of disagreement is 1; otherwise, the probability of disagreement is 1/2.

with (E^*, e^*) independently with probability 1/2, then the overall codeword will be invalid with high probability. Similar to case 3.a., this argument relies on leveraging the high independence of the ECSS, input locality 1 of the tampering function, and the fact that E is randomly chosen.

To summarize, two key properties are crucial to our arguments.

(A) Being non-committal to the errors. We rely on randomness of errors to argue inconsistency with tabulated errors in c_{err}.
(B) Independence of failure. Our objective is to identify output bits that cause decoding failure independently.

Consequently, we have the following objective.

> "Find a *large subset* of bits in $\widetilde{c}$ that *independently* fail the consistency check" while, simultaneously, "remaining *noncommittal* to (most of) the error (E, e)"

In the sequel, we elaborate the unique challenges to achieve this objective against δ-local functions, with $\delta > 1$, and no a priori bound on the input-locality.

Intuition underlying Our Construction. For a tampering function with output locality δ (referred to as a δ-local function), intuitively, every bit in the tampered codeword is influenced by some bits in c and some bits in c_{err}. The 2-local tampering functions suffice to capture these two influences and we use these to illustrate some primary challenges and key components of our construction.

Using the output locality of the tampering function, we can argue that tampering on c_{err} would be independent of the message m. Next, we use non-malleablity of encoding c_{err} to simulate whether $\widetilde{c_{\text{err}}}$ encodes (a) the original errrors, (2) an unrelated (E^*, e^*), or (3) $\perp$. Let us consider the case when the tampering function preserves the original errors. In this case, we perform a case analysis on the number of *not-copied-bits*. So the first (somewhat minor) hurdle is how to define not-copied-bits for δ-local functions. Since a bit in $\widetilde{c}$ can be influenced by δ bits, it is a not-copied-bit if it is not a copy for (at least) 1 out of the 2^δ possible inputs. Hence, in the final argument, this bit shall fail the consistency check with probability $1/2^\delta$. Thus, as δ increases, we need to find *exponentially more* bits that *independently* fail to be consistent.

The second hurdle is that, unlike Agrawal et al. [5], our tampering functions are *not* input-local. So, for instance, one bit in the (c, c_{err}) can influence every bit of the tampered codeword. Therefore, even though there might be many not-copied-bits, their probability of being inconsistent is possibly correlated. To resolve this challenge, Viola [35] proposed a technique to fix the values of the *highly influential* input bits (sampled from an appropriate distribution) of the tampering function. This technique, intuitively, transforms an output local tampering function into a convex combination of tampering function that are both input and output local. We use this technique to fix the highly influential bits in c to be uniform random bits (relying on output locality of tampering function and independence of ECSS). However, as we discuss below, many challenges remain related to the bits in c_{err} that are highly influential for $\widetilde{c}$.

Consider the following representative 2-local tampering function. Each bit is $\widetilde{c}$ is influenced by corresponding bit in c and a bit in c_{err} while ensuring that all bits in c_{err} have an identical number of output neighbors.

(1) If the threshold to identify "highly influential" input bits is set too low, then the procedure mentioned above might fix the entire c_{err}, because the size of c_{err} is very small. Consequently, the error (E, e) gets fixed. Thereafter, it is unclear how to proceed and catch any non-trivial tampering of c. So, the threshold to identify "highly influential" *cannot* be too low. Therefore, in this case, it is possible that no bit in c_{err} is fixed and c_{err} cumulatively influences a lot of bits in $\widetilde{c}$.

(2) Ideally, we would like that the bits we pick from $\widetilde{c}$ to argue failure do not depend on c_{err}. However, in this case, all the bits in $\widetilde{c}$ depend on c_{err}.

(3) Furthermore, there is another subtle issue. Conditioning on the fact that the tampered $\widetilde{c_{\text{err}}}$ encodes the same error or a fixed (E^*, e^*) distorts the distribution of c_{err}, which, in turn, influences the distribution of the tampered $\widetilde{c}$. To summarize, the distributions $\widetilde{c}$ and (E, e) are correlated when conditioned on whether the $\widetilde{c_{\text{err}}}$ encodes the same c_{err} or a fixed c_{err}.

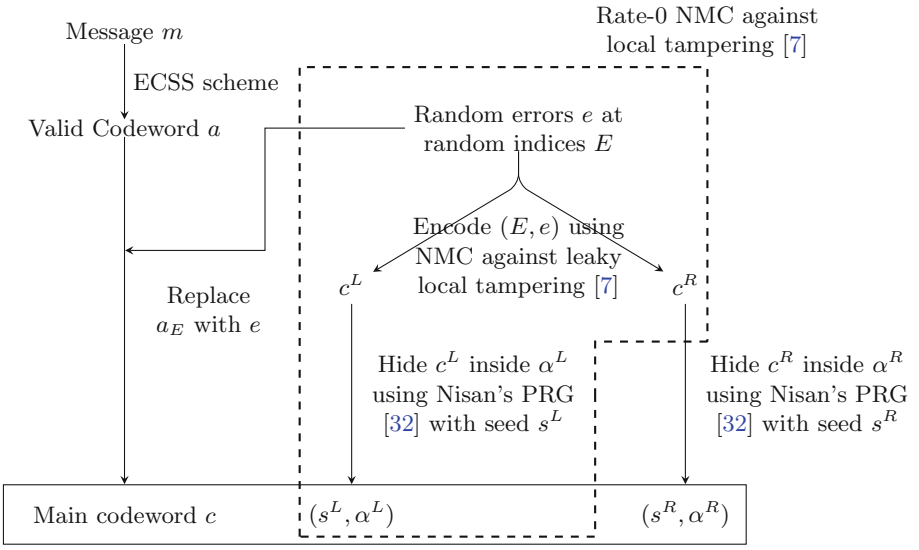

Fig. 1. Block diagram of the compiler to construct NMC against local tampering.

To resolve these concerns simultaneously, the high level idea is to hide the informative bits about (E, e), i.e., c_{err}, in a polynomially larger string, say α (refer to Fig. 1 for a block diagram of our compiler). We use a PRG with a super-polynomial stretch to determine the positions with informative bits inside α and store the PRG seed s along with α as the new payload. So our final

codeword is (c, s, α).[7] We argue that for any tampering function, the number of bits from c_{err} that are highly influential for $\widetilde{c}$ is small. To simulate these bits, we perform a small leakage on c_{err}. Since our base NMC from [8] is resilient to small leakage, we stay non-committal to (E, e) even conditioned on this leakage. Note that the rest of the bits in c_{err} have a bounded input locality onto $\widetilde{c}$ and hence, c_{err} influences only a small subset of bits in $\widetilde{c}$.

Now, if we had a large number of not-copied-bits in $\widetilde{c}$, we have a large number of not-copied-bits in $\widetilde{c}$ that are not influenced by c_{err}. But these bits might share input neighbors in c and have correlated probability of failing consistency checks. Recall that we have already fixed the highly influential bits in c. Finally, we can use the bounded input and output locality to identify independent bits in $\widetilde{c}$ (using the greedy neighbor-of-neighbor argument of Viola [35]).

This section presents only the intuitive rationale underlying the cryptographic primitives needed for our construction. There are further subtleties involved in the security arguments. Section 5.1 presents the full proof of our compiler using a hybrid argument.

Remark: Limit of Our Approach. We present a simple rationale for why our construction works for δ-local functions, where $\delta = \xi \lg n$ and $\xi < 1$ is a positive constant. Note that in steps 3.a. and 3.b., the probability of inconsistency with the tabulated error was at least $1/2$ in a 1-local tampering function. However, the probability of inconsistency in a δ-local tampering function can be as low as $2^{-\delta}$. The probability of u independent consistency checks to simultaneously pass is $(1 - 2^{-\delta})^u$. We need $u = \omega(2^{\delta} \log n)$ for this quantity to be negligible. On the other hand, we have $u \leqslant n$. Consequently, we must have $2^{\delta} \ll n/\log n$, or, in particular, $\delta \ll \lg n$.

2 Preliminaries

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For $x = (x_1, x_2, \ldots, x_n)$ and $S \subseteq [n]$, we use x_S to denote $(x_{s_1}, x_{s_2}, \ldots, x_{s_k})$, where $S = \{s_1, s_2, \ldots, s_k\}$ and $s_1 < s_2 < \cdots < s_k$. For brevity, we write x_{-i} for $x_{[n]\setminus\{i\}}$. We use U_S to represent the uniform distribution over the set S. If $\mathcal{D}$ is a distribution, we write $x \sim \mathcal{D}$ to denote that x is sampled according to distribution $\mathcal{D}$. The support of a distribution $\mathcal{D}$, represented by $\mathsf{Supp}(\mathcal{D})$, is the set $\{x\colon \Pr[\mathcal{D} = x] > 0\}$. For any binary strings $x, y \in \{0,1\}^n$, we use $\mathsf{HD}(x, y)$ to denote their Hamming distance defined by $\mathsf{HD}(x, y) := |\{i\colon x_i \neq y_i \text{ and } 1 \leqslant i \leqslant n\}|$.

2.1 Local Functions

Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a deterministic function. We write f as $(f_1, f_2, \ldots, f_n)$ such that $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$, where each

[7] Similar to [6], hash function families with sufficiently high independence also suffice in this context.

$f_i\colon \{0,1\}^n \to \{0,1\}$ and $1 \leqslant i \leqslant n$. We say that the *i-th bit (of the input) has influence on the j-th bit (of the output)* if there exists an x^*_{-i} such that

$$f_j(x^*_1, \ldots, x^*_{i-1}, 0, x^*_{i+1}, \ldots, x^*_n) \neq f_j(x^*_1, \ldots, x^*_{i-1}, 1, x^*_{i+1}, \ldots, x^*_n)$$

For every output position $1 \leqslant j \leqslant n$, we define the input neighbors $\mathrm{Inp}_f(j)$ to be $\{i | 1 \leqslant i \leqslant n, i \text{ has influence on } j\}$. Similarly, for an input position $1 \leqslant i \leqslant n$, we define its output neighbors $\mathrm{Out}_f(i)$ to be $\{j | 1 \leqslant j \leqslant n, i \text{ has influence on } j\}$. We extend this notion naturally to a set of indices. We write $\mathrm{Inp}_f(S) = \cup_{s \in S} \mathrm{Inp}_f(s)$ and $\mathrm{Out}_f(S) = \cup_{s \in S} \mathrm{Out}_f(s)$.

A function f has input locality δ, if, for all $1 \leqslant i \leqslant n$, we have $|\mathrm{Out}_f(i)| \leqslant \delta$. Similarly, a function f has output locality δ, if for all $1 \leqslant j \leqslant n$, we have $|\mathrm{Inp}_f(j)| \leqslant \delta$.

Definition 1 (Local Functions). *A function $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ is called a δ-local function if it has output locality δ.*

We use Local^δ to represent the set of all such functions because n shall be implicit from our context.

Recall that NC^0 is the set of all functions f such that for all i, f_i can be computed by a circuit of fan-in 2 and constant depth. Trivially, $\mathsf{NC}^0 \subseteq \mathrm{Local}^{O(1)}$.

We follow the convention in the literature and define the restriction of boolean functions as follows.

Definition 2 (Restriction). *Let $g : \{0,1\}^n \to \{0,1\}$ be a boolean function and $(I, \bar{I})$ be a partition of $[n]$. Let $x \in \{0,1\}^I$. Then, we write $g_{I|x} : \{0,1\}^n \to \{0,1\}$ for function g with input of indices in I being restricted to x. For function $f : \{0,1\}^n \to \{0,1\}^n$ such that $f = (f_1, f_2, \ldots, f_n)$ we write $f_{I|x}$ to denote $((f_1)_{I|x}, (f_2)_{I|x}, \ldots, (f_n)_{I|x})$. We say that $i \in \bar{I}$ has influence on j if there exists a x^*_{-i} such that $x^*_I = x$ and*

$$(f_{I|x})_j(x^*_1, \ldots, x^*_{i-1}, 0, x^*_{i+1}, \ldots, x^*_n) \neq (f_{I|x})_j(x^*_1, \ldots, x^*_{i-1}, 1, x^*_{i+1}, \ldots, x^*_n)$$

Note that for all $j \in [n]$, $\mathrm{Inp}_{f_{I|x}}(j) = \{i | 1 \leqslant i \leqslant n, i \text{ has influence on } j\} \subseteq \bar{I}$.

2.2 Non-malleable Codes

We define non-malleable codes below similar to previous works.

Definition 3 (Coding Schemes). *Let $\mathrm{Enc}\colon \{0,1\}^\ell \to \{0,1\}^n$ and $\mathrm{Dec}\colon \{0,1\}^n \to \{0,1\}^\ell \cup \{\perp\}$ be randomized functions (that is, they has access to private randomness). The pair $(\mathrm{Enc}, \mathrm{Dec})$ defines an encoding scheme with block length n and message length ℓ if it satisfies perfect (resp., statistical) correctness. That is, for all $m \in \{0,1\}^\ell$, over the randomness of Enc and Dec, $\Pr[\mathrm{Dec}(\mathrm{Enc}(m)) = m] = 1$ (resp., $\Pr[\mathrm{Dec}(\mathrm{Enc}(m)) = m] = 1 - \mathsf{negl}(\ell)$). The rate R of this encoding scheme is defined as $R = \ell/n$.*

Let $\mathcal{F}_n$ denote the set of all functions $f\colon \{0,1\}^n \to \{0,1\}^n$. Non-malleable codes are defined w.r.t. a family of tampering functions, say $\mathcal{F} \subseteq \mathcal{F}_n$, as follows.

Definition 4 ((n, ℓ, ε)-Non-malleable Codes). *A coding scheme* $(\mathrm{Enc}, \mathrm{Dec})$ *with block length* n *and message length* ℓ *is said to be non-malleable against tampering family* $\mathcal{F} \subseteq \mathcal{F}_n$ *with (simulation) error* ε*, if for all functions* $f \in \mathcal{F}$*, there exists a distribution* $\mathrm{Sim}(f)$ *over* $\{0,1\}^\ell \cup \{\perp\} \cup \{\text{same}^*\}$ *such that for all messages* $m \in \{0,1\}^\ell$*,*

$$\mathrm{Tamper}_f^m \approx_\varepsilon \mathrm{copy}\left(\mathrm{Sim}(f), m\right)$$

where Tamper_f^m *stands for the following distribution of the tampered message*

$$\mathrm{Tamper}_f^m := \left\{ \begin{array}{c} c \sim \mathrm{Enc}(m), \widetilde{c} = f(c),\ \widetilde{m} = \mathrm{Dec}(\widetilde{c}) \\ \textit{Output: } \widetilde{m}. \end{array} \right\}$$

and

$$\mathrm{copy}(x, y) = \begin{cases} y, & \textit{if } x = \text{same}^*; \\ x, & \textit{otherwise.} \end{cases}$$

2.3 Hypergeometric Distribution

Consider a universe of size N with K success samples. An (N, K, n)-hypergeometric distribution is the probability distribution of number of success samples picked when n random samples are picked from the universe without replacement. Specifically, we define the distribution as follows.

Definition 5. *A distribution* $\mathcal{D}$ *over the sample space* $[n]$ *is an* (N, K, n)*-hypergeometric distribution if, for any* $k \in [n]$*, we have*

$$\Pr[\mathcal{D} = k] = \binom{K}{k}\binom{N-K}{n-k}\binom{N}{n}^{-1}$$

Using standard coupling arguments, it is known that the hypergeometric distribution is more concentrated than the corresponding Bernoulli distribution. Consequently, we have the following tail bound.

Lemma 1. *([15,25]) Let* X *be a random variable sampled from a* (N, K, n)*-hypergeometric distribution. Then for any* $\varepsilon \in (0, \frac{K}{N})$*,*

$$\Pr\left[X \leqslant (K/N - \varepsilon) \cdot n\right] \leqslant \exp\left(-2\varepsilon^2 n\right)$$

The following corollary suffices for our proof.

Corollary 1. *Let* $A \subseteq [n]$ *be an arbitrary subset of size* a*. Let* $B \subseteq [n]$ *be a random subset of size* b*. Then*

$$\Pr\left[|A \cap B| \leqslant ab/2n\right] \leqslant \exp\left(-a^2 b/2n^2\right)$$

Note that $|A \cap B|$ is an (n, a, b)-hypergeometric distribution. The corollary follows from the previous lemma with $\varepsilon = a/2n$.

3 Building Blocks

In this section we describe the building blocks of our compiler.

3.1 Non-malleable Codes Against Leaky Input and Output Local Tampering

Our construction relies on an encoding scheme that satisfies non-malleability against leaky input and output local tampering that we define below.

Definition 6. *Let* $(\mathrm{Enc}, \mathrm{Dec})$ *be a coding scheme such that* $\mathrm{Enc} : \{0,1\}^k \to \{0,1\}^{n_L} \times \{0,1\}^{n_R}$ *and* $\mathrm{Dec} : \{0,1\}^{n_L} \times \{0,1\}^{n_R} \to \{0,1\}^k$. *We call* $(\mathrm{Enc}, \mathrm{Dec})$ *a* $(\lambda, \mu, \ell_i, \ell_o)$-non-malleable code against leaky input and output local tampering *with simulation error* ε *if it satisfies the following conditions.*

Let $\mathcal{L}^L \subseteq [n_L]$ *and* $\mathcal{L}^R \subseteq [n_R]$ *be arbitrary subsets of size at most* λn_L *and* λn_R, *respectively. Consider any function* F *with domain* $\{0,1\}^{|\mathcal{L}^L|} \times \{0,1\}^{|\mathcal{L}^R|}$ *that outputs a tampering function* $g : \{0,1\}^{n_L} \times \{0,1\}^{n_R} \to \{0,1\}^{n_L} \times \{0,1\}^{n_R}$ *such that for any* $x \in \{0,1\}^{|\mathcal{L}^L|}$, $y \in \{0,1\}^{|\mathcal{L}^R|}$, *and* $g = F(x, y)$

1. *The output locality of the tampering function* g *is at most* ℓ_o, *and*
2. *All but (at most)* μn_L *input-bits of the first* n_L *input-bits of* g *have input locality (at most)* ℓ_i.

Then, there exists a distribution $\mathrm{Sim}(\mathcal{L}^L, \mathcal{L}^R, F)$ *over* $\left(\{0,1\}^k \cup \{\perp, \text{same}^*\}\right) \times \{0,1\}^{|\mathcal{L}^L|} \times \{0,1\}^{|\mathcal{L}^R|}$ *such that for any message* $m \in \{0,1\}^k$,

$$\mathrm{Tamper}^m_{\mathcal{L}^L,\mathcal{L}^R,F} \approx_\varepsilon \mathrm{copy}(\mathrm{Sim}(\mathcal{L}^L, \mathcal{L}^R, F),\ m),\ \textit{where}$$

$$\mathrm{Tamper}^m_{\mathcal{L}^L,\mathcal{L}^R,F} := \left\{ \begin{array}{c} (c^L, c^R) \sim \mathrm{Enc}(m), x := c^L_{\mathcal{L}^L}, y := c^R_{\mathcal{L}^R} \\ g := F(x, y) \\ (\widetilde{c^L}, \widetilde{c^R}) = g(c^L, c^R), \widetilde{m} = \mathrm{Dec}(\widetilde{c^L}, \widetilde{c^R}) \\ \textit{Output } (\widetilde{m}, x, y) \end{array} \right\}$$

Intuitively, leaky input and output local tampering allows the adversary to first pick a subset of indices and peek into the codeword at those places, then use this leakage as an advice to select a output-local, (almost) input-local tampering function. Then, non-malleability against leaky input and output local tampering guarantees that the tampered message and the leakage are simulatable independent of the original message only given the position of leaked indices and the map F from leakage to the tampering function. Ball et al. [7] construct this non-malleable code as an intermediate step toward their final rate-0 non-malleable codes against local tampering. As a corollary of their results, we have the following lemma, which suffices for our construction.

Lemma 2 ([7]). *There exist constants λ, μ such that, for any $\ell_i, \ell_o = O(\log k)$, there exists an explicit and efficient $(\lambda, \mu, \ell_i, \ell_o)$-non-malleable code against leaky input and output local tampering with simulation error $\varepsilon = \mathsf{negl}(k)$ and rate $1/k^{o(1)}$, where k is the length of the message.*

Remark 1. Note that [7] reduces the problem of constructing non-malleable codes against leaky input and output local tampering to the problem of constructing non-malleable codes against 2-split-state tampering family. The rate of their final construction will be the product of the rate of the reduction, which is inverse of the locality (i.e., $1/\max(\ell_i, \ell_o)$) and the rate of the given 2-split-state non-malleable code. Instantiated with the state-of-the-art 2-split-state construction by Li [29,30], which has rate $\Omega(\log\log\log k/\log\log k)$, the final rate of [7]'s construction can be as high as $1/\operatorname{polylog}(k)$, which is $1/k^{o(1)}$ and satisfies this lemma.

3.2 Error-Correcting Secret-Sharing Schemes

Definition 7. *An encoding scheme* $(\mathrm{Enc}, \mathrm{Dec})$ *with block length n and message length ℓ is said to be an* (n, ℓ, d, t)-error-correcting secret sharing scheme *(ECSS scheme) if it satisfies the following conditions.*

1. ***Distance*** *d. For any two codewords c, c', $\mathsf{HD}(c, c') > d$.*
2. ***Independence*** *t. For any message $m \in \{0,1\}^\ell$ and a subset $S \subseteq [n]$ such that $|S| \leqslant t$, the distribution of $\mathrm{Enc}(m)_S$ is identical to the uniform distribution $U_{\{0,1\}^{|S|}}$.*
3. ***Error Correction*** *$d/2$. There exists an error-correcting function* ECorr *such that for any $c \in \{0,1\}^n$, $\mathrm{ECorr}(c)$ outputs a codeword c^* such that $\mathsf{HD}(c, c^*) \leqslant d/2$. If no such codeword exists, then it outputs $\perp$.*

Lemma 3. *For every $\zeta \in (0,1)$, there exists an explicit (n, ℓ, d, t)-ECSS scheme with $n = (1 + o(1))\ell$ and $d, t \geqslant n^{1-\zeta}$.*

Standard Reed-Solomon codes over characteristic 2 fields achieve the properties required by Lemma 3. We defer such a construction to the full version.

3.3 Pseudorandom Generator for Finite State Machines

Definition 8 (Finite State Machine). *A finite state machine (FSM) Q with space w over the alphabet Σ satisfies the following properties.*

1. *There exists a state-transition function $q\colon \{0,1\}^w \times \Sigma \to \{0,1\}^w$ that takes as input the current state $s \in \{0,1\}^w$ and an alphabet $x \in \Sigma$, and outputs the new state $q(s, x)$.*
2. *There exists a subset $S \subseteq \{0,1\}^w$ such that if the final state $s \in S$ then the FSM accepts the input and outputs 1. Otherwise, it outputs 0.*

Definition 9. *A function* $G\colon \{0,1\}^p \to \Sigma^u$ *is a* pseudorandom generator for FSMs *with space* w *and alphabet* Σ *with error* ε *if for any distinguisher FSM* Q *with space* w *and alphabet* Σ *we have*

$$\left|\Pr\left[Q\left(U_{\Sigma^u}\right)=1\right]-\Pr\left[Q\left(G(U_{\{0,1\}^p})\right)=1\right]\right| \leqslant \varepsilon$$

Lemma 4 ([32]). *There exists a constant* $\kappa > 0$ *such that for all integers* $d > 0$ *and* $u \leqslant \kappa d$*, there is an explicit pseudorandom generator* $G\colon \Sigma^{3u} \to \Sigma^{2^u}$ *for FSMs with alphabet* $\Sigma = \{0,1\}^d$ *and space* κd *with error* $2^{-\kappa d}$*.*

4 Our Compiler

In this section, we will present our compiler. That is, for all constants $\xi < 1$, given a rate-1 ECSS scheme, a rate $1/\eta^{o(1)}$ non-malleable code against leaky input and output local tampering (for η length messages) and a PRG secure against finite state machines with appropriate parameters, we construct a rate-1 non-malleable coding scheme against all δ-local tampering family Local^δ for $\delta = \xi \cdot \lg n$. Here n is the length of the codeword. We begin by giving some notation, specifying the building blocks used followed by our construction overview.

Notation: Throughout our construction and proof, we use the notation that after the tampering is done, any variable of original codeword, for example, a, will have a tilde on it, i.e., $\widetilde{a}$. For example, c is the original main codeword and $\widetilde{c}$ would be the tampered version of the main codeword. Thus, when we talk about bits from c, it refers to the input-bits of the tampering function and on the other hand, bits from $\widetilde{c}$ are output-bits of the tampering function.

Building blocks used. We use the following three building blocks. Let $\delta = \xi \cdot \lg n$ for $\xi < 1$ be the locality of the tampering function.

1. An (n, ℓ, d, t)-ECSS scheme with $d, t \geqslant n^{1-\zeta}$ and $n = (1 + o(1))\ell$ for an appropriate constant ζ to be fixed later. This is provided by Lemma 3.
2. For any constant λ, μ and $\eta = n^{\Theta(1)}$, a $(\lambda, \mu, \ell_i, \ell_o)$-NMC against leaky input and output local tampering for messages in $\{0,1\}^\eta$, rate $1/\eta^{o(1)}$, $\ell_o = \delta = O(\log \eta)$, $\ell_i = 4\delta/\mu = O(\log \eta)$, simulation error negligible in η. This is provided by Lemma 2. We denote the corresponding simulator by Sim_0.
3. A PRG $G : (\{0,1\}^{\log^2 n})^{3\Lambda \log n} \longrightarrow (\{0,1\}^{\log^2 n})^{n^\Lambda}$ that is secure against all FSMs with alphabet $\Sigma = \{0,1\}^{\log^2 n}$ and space $\kappa \log^2 n$ with error $2^{-\kappa \log^2 n}$ for an appropriate constant Λ to be fixed later. Here, κ is a constant provided by Lemma 4 for $u = \Lambda \log n$ and $d = \log^2 n$.

Building blocks:

- $(\mathsf{ECSS.Enc}, \mathsf{ECSS.Dec})$ is an (n, ℓ, d, t) ECSS scheme.
- $(\mathsf{NMEnc}_0, \mathsf{NMDec}_0)$ is a $(\lambda, \mu, \ell_i, \ell_o)$-non-malleable code against leaky input and output local tampering.
- $G : (\{0,1\}^{\log^2 n})^{3\Lambda \log n} \longrightarrow (\{0,1\}^{\log^2 n})^{n^{\Lambda}}$ is a PRG that fools all FSMs with space $\kappa \log^2 n$. We set Λ below.

$\mathsf{NMEnc}_1(m)$:

1. Sample a random $E \subseteq [n]$ of size $n^{1-\varepsilon_1}$, where ε_1 is a small constant.
2. For all $i \in E$, sample $e_i \sim U_{\{0,1\}}$.
3. Sample $a \sim \mathsf{ECSS.Enc}(m)$
4. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
5. Let $(c^L, c^R) \sim \mathsf{NMEnc}_0(E, e)$
6. Pick seeds $s^L, s^R \xleftarrow{\$} \{0,1\}^{3\Lambda \cdot \log^3 n}$.
7. Let $\mathsf{Embed}^L, \mathsf{Embed}^R$ be as below.
   - $\alpha^L = \mathsf{Embed}^L(s^L, c^L)$
   - $\alpha^R = \mathsf{Embed}^R(s^R, c^R)$
8. Output $(c, s^L, \alpha^L, s^R, \alpha^R)$

$\mathsf{NMDec}_1(\widetilde{c}, \widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R})$:

1. Let $\mathsf{Recover}^L, \mathsf{Recover}^R$ be as below.
   - $\widetilde{c^L} = \mathsf{Recover}^L(\widetilde{s^L}, \widetilde{\alpha^L})$
   - $\widetilde{c^R} = \mathsf{Recover}^R(\widetilde{s^R}, \widetilde{\alpha^R})$
2. If $\mathsf{NMDec}_0(\widetilde{c^L}, \widetilde{c^R}) = \perp$, output $\perp$
3. (Else) $(\widetilde{E}, \widetilde{e}) = \mathsf{NMDec}_0(\widetilde{c^L}, \widetilde{c^R})$
4. If $\mathsf{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$
5. (Else) $\widetilde{a} = \mathsf{ECSS.ECorr}(\widetilde{c})$
6. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
7. if $c' \neq \widetilde{c}$, output $\perp$
8. (Else) $\widetilde{m} = \mathsf{ECSS.Dec}(\widetilde{a})$
9. Output $\widetilde{m}$

Let lengths of c^L and c^R be n^{β_1} and n^{β_2}, respectively. First, pick[a] a constant γ s.t. $\max(\beta_1, \beta_2) < \gamma < 1$. Next, let $\tau > 0$ be a constant s.t. $\Lambda = \gamma + 2\tau < 1$.

$\mathsf{Embed}^L, \mathsf{Recover}^L$: Let $\rho^L : \{0,1\}^{\log^2 n} \longrightarrow \{0,1\}$ be any function with bias[b] $2n^{-(\Lambda - \beta_1)}$. First, compute $G(s^L) = (y_1, y_2, \ldots, y_{n^\Lambda})$ s.t. each $y_i \in \{0,1\}^{\log^2 n}$ and $\mathrm{Adv}^L = (\rho^L(y_1), \rho^L(y_2), \ldots, \rho^L(y_{n^\Lambda}))$. Then, $\alpha^L = \mathsf{Embed}^L(s^L, c^L)$ is defined as:

$$\alpha_i^L := \begin{cases} c_j^L & \text{If } \mathrm{Adv}_i^L \text{ is the } j^{th} \text{ 1 in } \mathrm{Adv}^L \\ 0 & \text{Otherwise.} \end{cases}$$

To recover during decoding, compute $G(\widetilde{s^L}) = (\widetilde{y_1}, \widetilde{y_2}, \ldots, \widetilde{y_{n^\Lambda}})$ and $\widetilde{\mathrm{Adv}}^L = (\rho^L(\widetilde{y_1}), \ldots, \rho^L(\widetilde{y_{n^\Lambda}}))$. Then, if $\widetilde{\mathrm{Adv}}^L$ does not contain $\geqslant n^{\beta_1}$ many 1's, quit decoding by outputing $\perp$. Otherwise, $\widetilde{c^L} = \mathsf{Recover}^L(\widetilde{s^L}, \widetilde{\alpha^L})$ is defined as:

$$\widetilde{c_j^L} := \widetilde{\alpha_i^L} \quad \text{where } \widetilde{\mathrm{Adv}_i}^L \text{ is the } j^{th} \text{ 1 in } \widetilde{\mathrm{Adv}}^L$$

$\mathsf{Embed}^R, \mathsf{Recover}^R$: Let $\rho^R : \{0,1\}^{\log^2 n} \longrightarrow \{0,1\}$ be any function with bias $2n^{-(\Lambda - \beta_2)}$. Now $\mathsf{Embed}^R, \mathsf{Recover}^R$ are defined analogously to above using ρ^R.

[a] This is possible because (E, e) has length $\eta = n^{1-\varepsilon_1}(\log n + 1)$ and $(\mathsf{NMEnc}_0, \mathsf{NMDec}_0)$ is a $1/\eta^{o(1)}$ rate coding scheme.

[b] Bias of a function is the probability that output is 1 for a uniformly sampled input.

Fig. 2. Our rate-1 non-malleable codes against δ-local functions

Construction Overview. Our construction starts with encoding the message $m \in \{0,1\}^{\ell}$ using ECSS scheme $a \sim \text{ECSS.Enc}(m)$ such that $a \in \{0,1\}^{n}$. Next, we sample a random subset $E \subseteq [n]$ of size $n^{1-\varepsilon_1}$ for a small constant ε_1 specified later. Next, for each index $i \in E$, we sample a random bit e_i. These will be our planted errors. Then, all bits at E in codeword a are replaced by these random bits e_i to produce c. We refer to this an erroneous codeword c as the main codeword. We note that a bit at index in E has probability $1/2$ of being an error.

Next, for the second part of our codeword, we record the error indices E as well as planted errors $e = (e_1, e_2, \ldots, e_{|E|})$ using (poor-rate) non-malleable codes against leaky input and output local tampering. We sample $(c^L, c^R) \sim \text{NMEnc}_0(E, e)$. Finally, we hide the codeword (c^L, c^R) inside a larger code (α^L, α^R) at pseudorandom locations as follows: We will sample two seeds s^L, s^R of appropriate length (See Fig. 2). And invoke our pseudorandom generator G on s^L (resp., s^R) and use appropriate bias function ρ^L (resp., ρ^R) to generate advice string Adv^L (resp., Adv^R). At a high level, positions having a 1 in the advice string will store an actual bit of the code, and positions with 0 will store a redundant 0. Intuitively, this step ensures that when bits from α^L or α^R are used for tampering, most of these bits would be redundant 0's. Our final codeword is $(c, s^L, \alpha^L, s^R, \alpha^R)$.

Conversely, to decode, we use seeds $\widetilde{s^L}, \widetilde{s^R}$ to determine the indices of $\widetilde{c^L}, \widetilde{c^R}$ in $\widetilde{\alpha^L}, \widetilde{\alpha^R}$. Then, we decode $(\widetilde{c^L}, \widetilde{c^R})$ to get the error index set $\widetilde{E}$ and error bits $\widetilde{e}$. Next, we compare $\widetilde{c}$ with planted errors $(\widetilde{E}, \widetilde{e})$ to check (1) whether all the bits from $\widetilde{c}$ with index in $\widetilde{E}$ and $\widetilde{e}$ are equal; (2) we error correct $\widetilde{c}$ to obtain correct codeword $\widetilde{a}$ and check whether all the errors in $\widetilde{c}$ were recorded in $\widetilde{E}$. If both conditions are satisfied, we will consider the codeword valid and output the decoding of $\widetilde{a}$ as the decoded message.

Setting the parameters. Next, we will set the various constants used in our construction (as well as proof of non-malleability).

- λ, μ: We pick constants λ, μ arbitrarily.
- Λ, γ, τ: Let $|c^L| = n^{\beta_1}$ and $|c^R| = n^{\beta_2}$. Since $\eta = |(E, e)| = n^{1-\varepsilon_1}(\log n)$ and rate of NMEnc_0 is $1/\eta^{o(1)}$, we have that $\max(\beta_1, \beta_2) < 1$. We pick positive constants γ, τ such that $\max(\beta_1, \beta_2) < \gamma < 1$ and $\gamma + 2\tau < 1$. Set $\Lambda = \gamma + 2\tau$.
- $\varepsilon_1, \varepsilon_2$: The number of erroneous indices $|E| = n^{1-\varepsilon_1}$. In our security hybrids, we have another small constant ε_2 and we require $\varepsilon_1 + 2\varepsilon_2 < 1 - \xi$, where ξ is defined by the tampering family. Hence, given ξ, we pick two positive constants satisfying the condition.
- ζ: In our construction, we use an (n, ℓ, d, t)-ECSS scheme with $d, t \geqslant n^{1-\zeta}$. In our security proof, we require $\zeta < \min(\varepsilon_1, \varepsilon_2, \tau, 1 - \Lambda)$ and hence, ζ can be picked as a sufficiently small positive constant satisfying the constraint.

Theorem 1. *Let $\{0,1\}^{\ell}$ be the message space and $\delta = \xi \cdot \lg n$, for some constant $\xi < 1$. There exists an explicit and efficient rate-1 NMC against Local^{δ} with simulation error that is negligible in n and uses the following primitives in a black-box manner.*

1. *For appropriate* $\zeta > 0$*, an* (n, ℓ, d, t)*-ECSS scheme with* $d, t \geqslant n^{1-\zeta}$ *and* $n = (1 + o(1))\ell$*.*
2. *For some constant* λ, μ *and* $\eta = n^{\Theta(1)}$*, a* $(\lambda, \mu, \ell_i, \ell_o)$*-NMC against leaky input and output local tampering for messages in* $\{0,1\}^{\eta}$*, rate* $1/\eta^{o(1)}$*,* $\ell_o = O(\log \eta)$*,* $\ell_i = O(\log \eta)$*, simulation error negligible in* η*.*
3. *For some constant* $\Lambda > 0$*, a PRG* $G : (\{0,1\}^{\log^2 n})^{3\Lambda \log n} \longrightarrow (\{0,1\}^{\log^2 n})^{n^{\Lambda}}$ *that is secure against FSM with alphabet size* $\log^2 n$ *and space* $\Theta(\log^2 n)$ *with error that is negligible in* n*.*

The above theorem when instantiated with Lemmas 3, 2 and 4 gives following theorem.

Theorem 2. *For all constants* $\xi < 1$*, there exists an explicit rate-1 non-malleable code against* $\text{Local}^{\xi \cdot \lg n}$ *with negligible in* n *simulation error, where* n *is the length of the codeword.*

In particular, this implies an explicit rate-1 non-malleable code against NC^0 tampering.

4.1 Proof of Theorem 1

Here, we will prove that the our construction has rate-1 and perfect correctness. We provide proof of non-malleability in the next section.

Rate of our construction. Our codeword is $(c, s^L, \alpha^L, s^R, \alpha^R)$. Note that our main codeword c has length $n = \ell + o(\ell)$. Next, $|s^L| = |s^R| = 3\Lambda \log^3 n$. And, $—\alpha^L| = |\alpha^R| = n^{\Lambda}$. Since, $\Lambda = \gamma + 2\tau < 1$ (see parameter setting above), the overall codeword has length $\ell + o(\ell)$.

Correctness. We first argue that our scheme has statistical correctness, and then show how the scheme in Fig. 2 can be tweaked slightly to give perfect correctness. It is easy to see that the correctness of our scheme in Fig. 2 is broken only when Adv^L does not have enough number of 1's to store all of c^L in α^L or similarly, when Adv^R does not have enough number of 1's to store all of c^R in α^R. If this happens, the decoding algorithm would output $\perp$. Note that whether this event happens or not depends on the choice of seeds s^L and s^R only. We prove the following lemma that states that probability of this event happening is negligible.

Lemma 5. *With probability at least* $1 - 2^{-\Omega(\log^2 n)}$ *over the random choice of* s_L *and* s_R*,* α^L *and* α^R *will contain all the bits from* c^L *and* c^R*.*

Proof. We will prove the lemma for (s^L, α^L) and same argument holds for (s^R, α^R). We first show that the lemma holds when G is a random function. Next, we argue that if lemma does not hold for a PRG G, then there exists a distinguisher FSM Q with space $\kappa \log^2 n$ that breaks PRG security with non-negligible probability in n.

Firstly, when $G(s^L)$ outputs uniform random string, the expected number of 1's in Adv^L is $n^{\Lambda} \cdot 2n^{-(\Lambda-\beta_1)} = 2n^{\beta_1}$. Next, using Chernoff bound, with probability at least $1 - \exp(-\Theta(n^{\beta_1}))$, there are at least n^{β_1} many 1's in Adv^L and hence, α^L will contain all the bits from c^L.

Now suppose that the lemma does not hold when we use PRG G that fools FSMs with space $\kappa \log^2 n$. Consider the following FSM Q that takes $(y_1, y_2, \ldots, y_{n^{\Lambda}})$ as input and a state in Q stores ctr, which denotes number of indices i for which $\rho^L(y_i)$ output 1. The final output of Q is 1 when $\mathsf{ctr} \geqslant n^{\beta_1}$. Clearly, by our argument above, on a true uniform string, Q will output 1 with probability at least $1 - \exp(-\Theta(n^{\beta_1}))$. If this lemma is incorrect for a PRG G, Q will output 1 with probability at most $1 - 2^{-\Omega(\log^2 n)}$ and hence Q will break the underlying PRG with success probability greater than $2^{-\Omega(\log^2 n)}$. Finally, note that Q only needs $\Lambda \log n < \kappa \log^2 n$ space to record A. This completes the proof. □

Getting perfect correctness. We can tweak our scheme slightly to give perfect correctness as follows: If s^L or s^R is bad, i.e., (α^L, α^R) will not contain all bits in (c^L, c^R), then we ignore $\text{Adv}^L, \text{Adv}^R$ and store the codeword in default location. More precisely, we store c^L in first $|c^L|$ locations in α^L and similarly for c^R. It is easy to see that this gives perfect correctness. In the proof of non-malleability, our simulator can simply give up when this case happens. (Since s^L, s^R are uniform seeds independent of the message, it is easy to check for this case.) This would increase the simulation error by the probability of this event occurring. But, above Lemma 5 proves that this happens with negligible probability. Hence, this only increases the simulation error by $\mathsf{negl}(n)$.

5 Proof of Non-malleability of Our Compiler

Non-malleability. Recall that to prove non-malleability of the resulting scheme against δ-local tampering family Local^{δ}, we need to show that for any $f \in \text{Local}^{\delta}$, there exists a simulator $\text{Sim}_1(f)$ such that, for all message $m \in \{0,1\}^{\ell}$, we have the following

$$\left\{ \begin{array}{l} (c, s^L, \alpha^L, s^R, \alpha^R) \sim \text{NMEnc}_1(m) \\ (\widetilde{c}, \widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R}) = f(c, s^L, \alpha^L, s^R, \alpha^R) \\ \quad \widetilde{m} = \text{NMDec}_1(\widetilde{c}, \widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R}) \\ \quad\quad \text{Output} \widetilde{m} \end{array} \right\} = \text{Tamper}_f^m \approx_{\varepsilon} \text{copy}(\text{Sim}_1(f), m)$$

Our simulator is formally defined in Fig. 3. In the simulator and the hybrids, $n_e = |(s^L, \alpha^L, s^R, \alpha^R)|$. A detailed proof using a sequence of indistinguishable hybrids is presented in the next section. We shall use the following lemma in our hybrid argument. We defer the proof of this lemma to the full version.

1. Let $P = \{i | i \in [n], |\mathrm{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \mathrm{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample a random $E_1 \subseteq X$ s.t. $|E_1| \sim (n, |X|, n^{1-\varepsilon_1})$-hypergeometric distribution.
5. For all $i \in E_1$, sample $e_i \sim U_{\{0,1\}}$.
6. For all $i \in E_1$, replace a_i with e_i, we get c_X.
7. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3A \log^3 n}$.
8. Given s^L (resp., s^R), indices of c^L (resp., c^R) in α^L (resp., α^R) are determined.
   Let Bad^L=\{Indices of c^L with more than $n^{1-\gamma-\tau}$ output neighbors in $\widetilde{c}$\},
   Leak^L=\{Indices of c^L with output neighbors in either $\widetilde{s^L}$ or $\widetilde{s^R}$\},
   Bad^R=\{Indices in c^R with more than $n^{1-\gamma-\tau}$ output neighbors in $\widetilde{c}$\}, and
   Leak^R=\{Indices in c^R with output neighbors in either $\widetilde{s^L}$ or $\widetilde{s^R}$\}
9. Let $\mathcal{L}^L = \mathrm{Bad}^L \cup \mathrm{Leak}^L$ and $\mathcal{L}^R = \mathrm{Bad}^R \cup \mathrm{Leak}^R$.
10. Let f_0 be the following mapping from leakage at $(\mathcal{L}^L, \mathcal{L}^R)$ to tampering function g for NMEnc_0: First, use (s^L, s^R), leakage at $(\mathrm{Leak}^L, \mathrm{Leak}^R)$ and c_Q to compute $\widetilde{s^L}$ and $\widetilde{s^R}$. These determine indices of $\widetilde{c^L}$ and $\widetilde{c^R}$ in $\widetilde{\alpha^L}$ and $\widetilde{\alpha^R}$. Then, define g to be the tampering function from indices of (c^L, c^R) to indices of $(\widetilde{c^L}, \widetilde{c^R})$.
11. If $(|\mathcal{L}^L| \geqslant \lambda n^{\beta_1})$ or $(|\mathcal{L}^R| \geqslant \lambda n^{\beta_2})$ or (f_0 does not satisfy Definition 6), output $\perp$
12. (Else) $(\mathrm{ans}, x, y) = \mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0)$.
13. Let S^L, S^R denote indices of s^L, s^R. Define function h as a restriction of f_1:

$$h := (f_1)_{(X, S^L, S^R, \mathcal{L}^L, \mathcal{L}^R)|(c_X, s^L, s^R, x, y)} \qquad \text{(See Definition 2)}$$

14. $V := \{i | i \in [n], \mathrm{Inp}_h(i) \neq \emptyset\}$
15. $W := \{i | i \in [n], \mathrm{Inp}_h(i) \backslash [n] \neq \emptyset\}$
16. $Z := \{i \in [n] | \exists z \in \{0,1\}^{n+n_e}, z_{(X, S^L, S^R, \mathcal{L}^L, \mathcal{L}^R)} = (c_X, s^L, s^R, x, y), h_i(\mathrm{z}) \neq z_i\}$
17. Sample $a \sim \mathrm{ECSS.Enc}(0^\ell) | (\mathrm{ECSS.\,Enc}(0^\ell))_X = a_X$
18. Sample a random $E_2 \subseteq [n] \backslash X$ of size $n^{1-\varepsilon_1} - |E_1|$, let $E = E_1 \cup E_2$
19. For all $i \in E_2$, sample $e_i \sim U_{\{0,1\}}$
20. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
21. $(\widetilde{E}, \widetilde{e}) = \mathrm{copy}(\mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0), (E, e))$
22. $(c^L, c^R) \sim \mathrm{NMEnc}_0(E, e)$ s.t. $\mathrm{NMDec}_0(g(c^L, c^R)) = (\widetilde{E}, \widetilde{e})$ and $c^L_{\mathcal{L}^L} = x$, $c^R_{\mathcal{L}^R} = y$
23. $\alpha^L = \mathrm{Embed}^L(s^L, c^L)$, $\alpha^R = \mathrm{Embed}^R(s^R, c^R)$
24. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
25. If ans =
    - $\perp$: Output $\perp$
    - same*: If $|Z \backslash (W \cup X)| \geqslant n^{1-\varepsilon_2}$, output $\perp$
      (Else) If $\widetilde{c}_Z = c_Z$, output same*; (Else) Output $\perp$.
    - (E^*, e^*): If $|V \backslash W| \geqslant n^{1-\varepsilon_2}$, output $\perp$
      (Else) If $\mathrm{ECSS.\,ECorr}(\widetilde{c}) \perp$, output $\perp$;
      (Else) $\widetilde{a} = \mathrm{ECSS.\,ECorr}(\widetilde{c})$
      Define c' as $c'_i = \begin{cases} \widetilde{a_i}, & i \notin \widetilde{E} \\ \widetilde{e_i}, & i \in \widetilde{E} \end{cases}$
      If $c' \neq \widetilde{c}$, output $\perp$; (Else) Output $\widetilde{m} = \mathrm{ECSS.\,Dec}(\widetilde{a})$

Fig. 3. Simulator $\mathrm{Sim}_1(f)$

Lemma 6. *For any δ-local tampering function, with probability at least $1 - 2^{-\Omega(\log^2 n)}$ over the random choice of s_L and s_R, the following conditions hold.*
(1) At most μn^{β_1} bits from c^L will have input locality higher than $4\delta/\mu$ onto $\widetilde{\alpha^R}$;
(2) Number of bits in c^L and c^R that have greater than $n^{1-\gamma-\tau}$ input locality onto $\widetilde{c}$ are bounded by $4\delta n^{\beta_1-\tau}$ and $4\delta n^{\beta_2-\tau}$, respectively.
And as a consequence, we have
(3) Number of bits in $\widetilde{c}$ that are influenced by low input locality bits from c^L and c^R are bounded by $n^{\beta_1} \cdot n^{1-\gamma-\tau} = o(n^{1-\tau})$ and $n^{\beta_2} \cdot n^{1-\gamma-\tau} = o(n^{1-\tau})$, respectively.

5.1 Detailed Hybrid Argument

In this section, we are going to use a series of statistically close hybrids to prove that Tamper_f^m and copy $\big(\mathrm{Sim}_1(f), m\big)$ are indistinguishable. Throughout this subsection, we use the following color/highlight notation. In a current hybrid, the text in red denotes the changes from the previous hybrid. The text in shaded part represents the steps that will be replaced by red part of the next hybrid. We call c (resp., $\widetilde{c}$) the main codeword and $(s^L, \alpha^L, s^R, \alpha^R)$ (resp., $(\widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R})$) the error codeword.

$H_1(f, m)$: Our first hybrid is the real world Tamper_f^m, we simply open up the definition of NMEnc_1 and NMDec_1 and write tampering function f as (f_1, f_2). Both functions are given as input the entire codeword and f_1 is doing the tampering on the main codeword, i.e., outputs $\widetilde{c}$, while f_2 is doing the tampering on the error codeword, i.e., outputs $(\widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R})$. This way of writing f would be useful in later hybrids.

$H_2(f, m)$: In the next hybrid H_2, we change the way we sample ECSS codeword of m. We define two subsets of indices P and Q. Intuitively, P is the *popular* input bits of the main codeword, i.e., bits in c that influence more than n^{ε_2} bits of $\widetilde{c}$. And Q is the set of bits in main codeword c that influence the error codeword $(\widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R})$. Now, let $X = P \cup Q$. We first sample a uniform string a_X of length $|X|$ and then sample $a \sim \mathrm{ECSS.Enc}(m)$ condition on that $\mathrm{ECSS.Enc}(m)_X = a_X$. We argue that this does not change the distribution of a and hence it is identical to the previous hybrid.

To argue this we use the independence property of our ECSS scheme. In particular, since $t \geqslant n^{1-\zeta}$, the distribution of a_X is indeed uniform as long as $|X| = o(n^{1-\zeta})$. Now, $|P|$ can be bound as follows: The total number of input neighbors of $\widetilde{c}$ is δn and at most $\delta n^{1-\varepsilon_2}$ many bits in c can influence more than n^{ε_2} bits from $\widetilde{c}$. Hence $|P| = o(n^{1-\zeta})$ as long as we pick $\boxed{\zeta < \varepsilon_2}$. Next, the length of the error codeword is $|s^L| + |\alpha^L| + |s^R| + |\alpha^R| = O(n^{\Lambda})$ and hence, by output locality δ, the size of Q is at most $\delta \cdot O(n^{\Lambda}) = o(n^{1-\zeta})$ as long as $\boxed{\zeta < 1 - \Lambda}$.

$H_1(f, m)$:

1. Sample a random $E \subseteq [n]$ of size $n^{1-\varepsilon_1}$
2. For all $i \in E$, sample $e_i \sim U_{\{0,1\}}$
3. Sample $a \sim \text{ECSS.Enc}(m)$
4. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
5. Let $(c^L, c^R) \sim \text{NMEnc}_0(E, e)$
6. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
7. $\alpha^L = \text{Embed}^L(s^L, c^L)$ and $\alpha^R = \text{Embed}^R(s^R, c^R)$
8. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
9. $(\widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R}) = f_2(c, s^L, \alpha^L, s^R, \alpha^R)$
10. $\widetilde{c^L} = \text{Recover}^L(\widetilde{s^L}, \widetilde{\alpha^L})$ and $\widetilde{c^R} = \text{Recover}^R(\widetilde{s^R}, \widetilde{\alpha^R})$
11. If $\text{NMDec}_0(\widetilde{c^L}, \widetilde{c^R}) = \perp$, output $\perp$; (Else) $(\widetilde{E}, \widetilde{e}) = \text{NMDec}_0(\widetilde{c^L}, \widetilde{c^R})$
12. If $\text{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \text{ECSS.ECorr}(\widetilde{c})$
13. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
14. If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \text{ECSS.Dec}(\widetilde{a})$
15. Output $\widetilde{m}$

$H_2(f, m)$:

1. Let $P = \{i | i \in [n], |\text{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \text{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample $a \sim \text{ECSS.Enc}(m) | (\text{ECSS.Enc}(m))_X = a_X$
5. Sample a random $E \subseteq [n]$ of size $n^{1-\varepsilon_1}$
6. For all $i \in E$, sample $e_i \sim U_{\{0,1\}}$
7. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
8. Let $(c^L, c^R) \sim \text{NMEnc}_0(E, e)$
9. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
10. $\alpha^L = \text{Embed}^L(s^L, c^L)$ and $\alpha^R = \text{Embed}^R(s^R, c^R)$
11. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
12. $(\widetilde{s^L}, \widetilde{\alpha^L}, \widetilde{s^R}, \widetilde{\alpha^R}) = f_2(c, s^L, \alpha^L, s^R, \alpha^R)$
13. $\widetilde{c^L} = \text{Recover}^L(\widetilde{s^L}, \widetilde{\alpha^L})$ and $\widetilde{c^R} = \text{Recover}^R(\widetilde{s^R}, \widetilde{\alpha^R})$
14. If $\text{NMDec}_0(\widetilde{c^L}, \widetilde{c^R}) = \perp$, output $\perp$; (Else) $(\widetilde{E}, \widetilde{e}) = \text{NMDec}_0(\widetilde{c^L}, \widetilde{c^R})$
15. If $\text{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \text{ECSS.ECorr}(\widetilde{c})$
16. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
17. If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \text{ECSS.Dec}(\widetilde{a})$
18. Output $\widetilde{m}$

$H_3(f, m)$:

1. Let $P = \{i | i \in [n], |\mathrm{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \mathrm{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample $a \sim \mathrm{ECSS.Enc}(m) | (\mathrm{ECSS.Enc}(m))_X = a_X$
5. Sample a random $E \subseteq [n]$ of size $n^{1-\varepsilon_1}$
6. For all $i \in E$, sample $e_i \sim U_{\{0,1\}}$
7. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
8. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
9. Given s^L (resp., s^R), indices of c^L (resp., c^R) in α^L (resp., α^R) are determined. Let Bad^L={Indices of c^L with more than $n^{1-\gamma-\tau}$ output neighbors in $\widetilde{c}$}, Leak^L={Indices of c^L with output neighbors in either $\widetilde{s^L}$ or $\widetilde{s^R}$}, Bad^R={Indices in c^R with more than $n^{1-\gamma-\tau}$ output neighbors in $\widetilde{c}$}, and Leak^R={Indices in c^R with output neighbors in either $\widetilde{s^L}$ or $\widetilde{s^R}$}
10. Let $\mathcal{L}^L = \mathrm{Bad}^L \cup \mathrm{Leak}^L$ and $\mathcal{L}^R = \mathrm{Bad}^R \cup \mathrm{Leak}^R$.
11. Let f_0 be the following mapping from leakage at $(\mathcal{L}^L, \mathcal{L}^R)$ to tampering function g for NMEnc_0: First, use (s^L, s^R), leakage at $(\mathrm{Leak}^L, \mathrm{Leak}^R)$ and c_Q to compute $\widetilde{s^L}$ and $\widetilde{s^R}$. These determine indices of $\widetilde{c^L}$ and $\widetilde{c^R}$ in $\widetilde{\alpha^L}$ and $\widetilde{\alpha^R}$. Then, define g to be the tampering function from indices of (c^L, c^R) to indices of $(\widetilde{c^L}, \widetilde{c^R})$.
12. If $(|\mathcal{L}^L| \geqslant \lambda n^{\beta_1})$ or $(|\mathcal{L}^R| \geqslant \lambda n^{\beta_2})$ or (f_0 does not satisfy Definition 6), output $\perp$
13. (Else) $(\widetilde{E}, \widetilde{e}, x, y) = \mathrm{Tamper}^{(E,e)}_{\mathcal{L}^L, \mathcal{L}^R, f_0}$
14. If $(\widetilde{E}, \widetilde{e}) = \perp$, output $\perp$
15. $(c^L, c^R) \sim \mathrm{NMEnc}_0(E, e)$ s.t. $\mathrm{NMDec}_0(g(c^L, c^R)) = (\widetilde{E}, \widetilde{e})$ and $c^L_{\mathcal{L}^L} = x$, $c^R_{\mathcal{L}^R} = y$
16. $\alpha^L = \mathrm{Embed}^L(s^L, c^L)$, $\alpha^R = \mathrm{Embed}^R(s^R, c^R)$
17. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
18. If $\mathrm{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \mathrm{ECSS.ECorr}(\widetilde{c})$
19. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
20. If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \mathrm{ECSS.Dec}(\widetilde{a})$
21. Output $\widetilde{m}$

$H_3(f, m)$: In the next hybrid H_3, we rewrite the way how $(\widetilde{E}, \widetilde{e})$ is generated from (E, e) given seeds s^L and s^R. Here, we would generate $(\widetilde{E}, \widetilde{e})$ as output of a tampering experiment on (E, e) with an appropriate tampering function from the leaky input and output local tampering family. Note that (E, e) is first encoded to (c^L, c^R) and then is hidden among (α^L, α^R) using seeds s^L, s^R. We note that if we are given the seed s^L and s^R, the places where c^L are c^R are stored among α^L and α^R is known. Similarly, if we know $\widetilde{s^L}$ and $\widetilde{s^R}$, the places where $\widetilde{c^L}$ and $\widetilde{c^R}$ are stored among $\widetilde{\alpha^L}$ and $\widetilde{\alpha^R}$ are also known. Therefore, we define Leak^L and Leak^R as the input neighbors of both $\widetilde{s^L}$ and $\widetilde{s^R}$ from c^L and c^R respectively. Now let f_0 be the mapping that given the leakage Leak^L and Leak^R,

first computes[8] $\widetilde{s^L}$ and $\widetilde{s^R}$, and then outputs the tampering function g. Now that we know indices of (c^L, c^R) and $(\widetilde{c^L}, \widetilde{c^R})$, function g maps (c^L, c^R) to $(\widetilde{c^L}, \widetilde{c^R})$.[9] We note that leaking bits at Bad^L and Bad^R from c^L and c^R would be used in later hybrids. So the total leakage from c^L and c^R are $\mathcal{L}^L = \text{Leak}^L \cup \text{Bad}^L$ and $\mathcal{L}^R = \text{Leak}^R \cup \text{Bad}^R$. Now we need to argue that the tampering f_0 and leakage $\mathcal{L}^L, \mathcal{L}^R$ forms a valid tampering experiment onto our base NMC against leaky input and output local tampering. It is easy to see that if it is valid, then the two hybrids are identical. When they are not valid we output $\perp$ in this hybrid and we need to argue that probability of output $\perp$ due this is negligible.

$H_4(f, m)$:

1. Let $P = \{i | i \in [n], |\text{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \text{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample $a \sim \text{ECSS.Enc}(m) | (\text{ECSS.Enc}(m))_X = a_X$
5. Sample a random $E \subseteq [n]$ of size $n^{1-\varepsilon_1}$
6. For all $i \in E$, sample $e_i \sim U_{\{0,1\}}$
7. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
8. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
9. Given s^L, define: $\text{Bad}^L, \text{Leak}^L$ as in $H_3(f, m)$
   Given s^R, define: $\text{Bad}^R, \text{Leak}^R$ as in $H_3(f, m)$
10. Let $\mathcal{L}^L = \text{Bad}^L \cup \text{Leak}^L$ and $\mathcal{L}^R = \text{Bad}^R \cup \text{Leak}^R$.
11. Define mapping f_0 and its output g as in $H_3(f, m)$
12. If $(|\mathcal{L}^L| \geqslant \lambda n^{\beta_1})$ or $(|\mathcal{L}^R| \geqslant \lambda n^{\beta_2})$ or (f_0 does not satisfy Definition 6), output $\perp$
13. (Else) $(\text{ans}, x, y) = \text{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0)$
14. If ans $= \perp$, output $\perp$; (Else) $(\widetilde{E}, \widetilde{e}) = \text{copy}(\text{ans}, (E, e))$
15. $(c^L, c^R) \sim \text{NMEnc}_0(E, e)$ s.t. $\text{NMDec}_0(g(c^L, c^R)) = (\widetilde{E}, \widetilde{e})$ and $c^L_{\mathcal{L}^L} = x$, $c^R_{\mathcal{L}^R} = y$
16. $\alpha^L = \text{Embed}^L(s^L, c^L)$, $\alpha^R = \text{Embed}^R(s^R, c^R)$
17. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
18. If $\text{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \text{ECSS.ECorr}(\widetilde{c})$
19. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
20. If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \text{ECSS.Dec}(\widetilde{a})$
21. Output $\widetilde{m}$

Firstly, f_0 might not satisfy Definition 6 if one of the following happens: (i) Not all the bits from c^L, c^R are contained in α^L and α^R, respectively and thus, f_0 cannot produce function g; (ii) g has output locality higher than $\ell_o = \delta$; (iii) under g, more than μn^{β_1} many bits from c^L have input locality higher than

[8] Note that at this point, the original seed s^L and s^R and their input neighbors c_Q from main codeword c is already fixed.

[9] If $\widetilde{c^L}$ or $\widetilde{c^R}$ are not contained in $\widetilde{\alpha^L}$ or $\widetilde{\alpha^R}$, f_0 will simply set g to be a $\perp$ function.

$\ell_i = 4\delta/\mu$ to $\widetilde{c^R}$. Note that our tampering function f is δ-local and therefore, the output function g will also be δ-local, thus (ii) will never happen. And the probability of (i) or (iii) happening is negligible as guaranteed by Lemma 5 and (1) from Lemma 6, respectively.

We bound the size of the leakage $|\mathcal{L}^L| = |\,\mathrm{Leak}^L \cup \mathrm{Bad}^L\,|$ by $o(n^{\beta_1})$. First, we observe that our seeds s^L and s^R are of length $O(\log^3 n)$ and hence $|\mathrm{Leak}^L|$ is at most $O(\delta \log^3 n) = o(n^{\beta_1})$. And the size of Bad^L is $o(n^{\beta_1})$ is guaranteed by (2) of Lemma 6. The argument for $\mathcal{L}^R$ is analogous to $\mathcal{L}^L$. This proves that this hybrid and the previous one are $2^{-\Omega(\log^2 n)}$-close.

Note that we still need the error codeword $(s^L, \alpha^L, s^R, \alpha^R)$ to do the tampering f_1 onto $\widetilde{c}$. Hence, we sample c^L and c^R under the condition that the tampering experiment outputs $(\widetilde{E}, \widetilde{e}, x, y)$ and construct the error codeword as defined by our compiler.

$H_5(f, m)$:

1. Let $P = \{i | i \in [n], |\mathrm{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \mathrm{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample a random $E_1 \subseteq X$ s.t. $|E_1| \sim (n, |X|, n^{1-\varepsilon_1})$-hypergeometric distribution
5. For all $i \in E_1$, sample $e_i \sim U_{\{0,1\}}$
6. For all $i \in E_1$, replace a_i with e_i, we get c_X
7. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
8. Given s^L, define: $\mathrm{Bad}^L, \mathrm{Leak}^L$ as in $H_3(f, m)$
   Given s^R, define: $\mathrm{Bad}^R, \mathrm{Leak}^R$ as in $H_3(f, m)$
9. Let $\mathcal{L}^L = \mathrm{Bad}^L \cup \mathrm{Leak}^L$ and $\mathcal{L}^R = \mathrm{Bad}^R \cup \mathrm{Leak}^R$.
10. Define mapping f_0 and its output g as in $H_3(f, m)$
11. If $(|\mathcal{L}^L| \geqslant \lambda n^{\beta_1})$ or $(|\mathcal{L}^R| \geqslant \lambda n^{\beta_2})$ or (f_0 does not satisfy Definition 6), output $\perp$
12. (Else) $(\mathrm{ans}, x, y) = \mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0)$.
13. If $\mathrm{ans} = \perp$, output $\perp$
14. Sample $a \sim \mathrm{ECSS.Enc}(m) | (\mathrm{ECSS.Enc}(m))_X = a_X$
15. Sample a random $E_2 \subseteq [n] \backslash X$ of size $n^{1-\varepsilon_1} - |E_1|$, Let $E = E_1 \cup E_2$
16. For all $i \in E_2$, sample $e_i \sim U_{\{0,1\}}$
17. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
18. $(\widetilde{E}, \widetilde{e}) = \mathrm{copy}(\mathrm{ans}, (E, e))$
19. $(c^L, c^R) \sim \mathrm{NMEnc}_0(E, e)$ s.t. $\mathrm{NMDec}_0(g(c^L, c^R)) = (\widetilde{E}, \widetilde{e})$ and $c^L_{\mathcal{L}^L} = x$, $c^R_{\mathcal{L}^R} = y$
20. $\alpha^L = \mathrm{Embed}^L(s^L, c^L)$, $\alpha^R = \mathrm{Embed}^R(s^R, c^R)$
21. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
22. If $\mathrm{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \mathrm{ECSS.ECorr}(\widetilde{c})$
23. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
24. If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \mathrm{ECSS.Dec}(\widetilde{a})$
25. Output $\widetilde{m}$

$H_4(f,m)$: In the next hybrid H_4, we simply replace the tampering experiment onto our base non-malleable codes with its corresponding simulator Sim_0 and incur a negligible error by Lemma 2.

$H_5(f,m)$: In this hybrid, we break the error indices E into two parts: $E_1 = E \cap X$ and $E_2 = E \backslash E_1$. Next, we note that c_Q is needed to define the tampering on the error codeword. Hence, we sample E_1 and the error bits from E_1 early before defining tampering on error codeword. However, rest of errors, i.e., E_2 is not used before we invoke simulator Sim_0. Based on these observations, we re-arrange parts of the hybrids and this hybrid is identical to the previous one. Note that the size of E_1 and E_2 are distributed according to $(n, |X|, n^{1-\varepsilon_1})$-hypergemetric distribution and $(n, n-|X|, n^{1-\varepsilon_1})$-hyper geometric distribution, respectively. By Corollary 1, it is easy to see that with probability $1-\exp(-\Theta(n^{1-\varepsilon_1}))$, the size of E_2 is at least $n^{1-\varepsilon_1}/2$.

$H_6(f,m)$: In this hybrid, we only introduce some new notation to be used in later hybrids and hence, this hybrid is identical to the previous one.

We focus on the tampering of the main codeword using function f_1. Note that so far in the previous hybrid, we have already fixed certain bits in the input main codeword c (that is, c_X), picked PRG seeds s^L, s^R and also leaked certain parts of c^L, c^R, i.e., $\mathcal{L}^L, \mathcal{L}^R$.[10] Using this information, we define a restriction h of function f_1 that fixes all the above bits in the input.

We next define three subsets of $[n]$ corresponding to h, namely, V, W and Z as follows. V is the subset of bits i such that $\widetilde{c_i}$ is not fixed given the fixing of bits done so far. And W is the subset of bits that are influenced by some bits in the error codeword (that have not been leaked and fixed so far). And Z is the subset of bits i, such that the output of h_i is not always the i-th input bit (In the definition of Z, recall that $n_e = |(s^L, \alpha^L, s^R, \alpha^R)|$).

Intuitively, Z is the set of bits that are not-copied-bits under the tampering function h, V is the set of non-constant-bits and W is the set of bits that are influenced by the error codeword. As we explained in technical overview Sect. 1.3, if ans = same* and the size of $Z \backslash W$ is large or if ans $= (E^*, e^*)$ and the size of $V \backslash W$ is large, then the tampered codeword will be invalid with probability $1 - \mathsf{negl}(n)$. This intuition is formally proved in the next hybrid.

$H_7(f,m)$: In the next hybrid H_7, we add a sanity check right after we define V, W, Z. (a) When ans $= \perp$, we will output $\perp$ immediately. This is the same as the previous hybrid. (b) When ans = same*, we check the size of $Z \backslash (W \cup X)$. If it is larger than $n^{1-\varepsilon_2}$, we directly output $\perp$ without any further computation. On the other hand, if it is less than $n^{1-\varepsilon_2}$, we only compare c and $\widetilde{c}$ at locations Z. If they are the same, we output same*, otherwise, we output $\perp$. (c) When ans $= (E^*, e^*)$, we check the size of $V \backslash W$. If $|V \backslash W| \geqslant n^{1-\varepsilon_2}$, we directly output $\perp$ without further computation. Below, we prove that the previous hybrid $H_6(f,m)$ and $\mathrm{copy}(H_7(f,m), m)$ are statistically close. We break the proof into two parts: ans = same* case and ans $= (E^*, e^*)$ case.

[10] Note that those places in α^L, α^R that are not used to store c^L and c^R are also fixed (to be 0 by the compiler).

$H_6(f, m)$:

1. Let $P = \{i | i \in [n], |\mathrm{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \mathrm{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample a random $E_1 \subseteq X$ s.t. $|E_1| \sim (n, |X|, n^{1-\varepsilon_1})$-hypergeometric distribution.
5. For all $i \in E_1$, sample $e_i \sim U_{\{0,1\}}$
6. For all $i \in E_1$, replace a_i with e_i, we get c_X
7. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
8. Given s^L, define: $\mathrm{Bad}^L, \mathrm{Leak}^L$ as in $H_3(f, m)$
   Given s^R, define: $\mathrm{Bad}^R, \mathrm{Leak}^R$ as in $H_3(f, m)$
9. Let $\mathcal{L}^L = \mathrm{Bad}^L \cup \mathrm{Leak}^L$ and $\mathcal{L}^R = \mathrm{Bad}^R \cup \mathrm{Leak}^R$.
10. Define mapping f_0 and its output g as in $H_3(f, m)$
11. If $(|\mathcal{L}^L| \geqslant \lambda n^{\beta_1})$ or $(|\mathcal{L}^R| \geqslant \lambda n^{\beta_2})$ or (f_0 does not satisfy Definition 6), output $\perp$
12. (Else) $(\mathrm{ans}, x, y) = \mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0)$.
13. Let S^L, S^R denote indices of s^L, s^R. Define function h as a restriction of f_1 (Definition 2): $h := (f_1)_{(X, S^L, S^R, \mathcal{L}^L, \mathcal{L}^R)|(c_X, s^L, s^R, x, y)}$
14. $V := \{i \in [n] | \mathrm{Inp}_h(i) \neq \emptyset\}$.
15. $W := \{i \in [n] | \mathrm{Inp}_h(i) \backslash [n] \neq \emptyset\}$.
16. $Z := \{i \in [n] | \exists z \in \{0,1\}^{n+n_e}, z_{(X, S^L, S^R, \mathcal{L}^L, \mathcal{L}^R)} = (c_X, s^L, s^R, x, y), h_i(z) \neq z_i\}$.
17. If ans $= \perp$, output $\perp$
18. Sample $a \sim \mathrm{ECSS.Enc}(m) | (\mathrm{ECSS.Enc}(m))_X = c_X$
19. Sample a random $E_2 \subseteq [n] \backslash X$ of size $n^{1-\varepsilon_1} - |E_1|$, let $E = E_1 \cup E_2$
20. For all $i \in E_2$, sample $e_i \sim U_{\{0,1\}}$
21. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
22. $(\widetilde{E}, \widetilde{e}) = \mathrm{copy}(\mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0), (E, e))$
23. $(c^L, c^R) \sim \mathrm{NMEnc}_0(E, e)$ s.t. $\mathrm{NMDec}_0(g(c^L, c^R)) = (\widetilde{E}, \widetilde{e})$ and $c^L_{\mathcal{L}^L} = x$, $c^R_{\mathcal{L}^R} = y$
24. $\alpha^L = \mathrm{Embed}^L(s^L, c^L)$, $\alpha^R = \mathrm{Embed}^R(s^R, c^R)$
25. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
26. If $\mathrm{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \mathrm{ECSS.ECorr}(\widetilde{c})$
27. Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
28. If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \mathrm{ECSS.Dec}(\widetilde{a})$
29. Output $\widetilde{m}$

Case ans = same*: Let us first look at that the case when $|Z \backslash (W \cup X)| < n^{1-\varepsilon_2}$. Note that by the definition of Z, all the bits of $\widetilde{c}$ in $[n] \backslash Z$ are identical to those in c. Recall c is obtained by planting $|E| = n^{1-\varepsilon_1}$ errors into a valid ECSS codeword a. We have $\mathsf{HD}(\widetilde{c}, a) \leqslant \mathsf{HD}(\widetilde{c}, c) + \mathsf{HD}(c, a) = |Z| + |E| \leqslant (|Z \backslash (W \cup X)| + |W| + |X|) + |E|$. Using $|W| = o(n^{1-\tau})$ from (3) of Lemma 6, $|X| = o(n^{1-\zeta})$ from hybrid 2, and $|E| = n^{1-\varepsilon_1}$, we get $\mathsf{HD}(\widetilde{c}, a) \leqslant n^{1-\varepsilon_2} + o(n^{1-\tau}) + o(n^{1-\zeta}) + n^{1-\varepsilon_1} = o(n^{1-\zeta})$ by setting $\boxed{\zeta < \varepsilon_2}$, $\boxed{\zeta < \tau}$ and $\boxed{\zeta < \varepsilon_1}$. Hence, using the fact that the distance of the ECSS scheme, $d \geqslant n^{1-\zeta}$, we get

$\mathsf{ECSS.ECorr}(\widetilde{c}) = a$. Consequently, if we error-correct $\widetilde{c}$ and plant in the original errors (E, e), we get c. Hence, experiment would output $\perp$ iff $\widetilde{c} \neq c$. This happens only when $\widetilde{c}_Z \neq c_Z$.

Now consider the case when $|Z \backslash (W \cup X)| \geqslant n^{1-\varepsilon_2}$. We begin by computing a lower bound on number of error indices in $Z \backslash (W \cup X)$, i.e., size of set $A = (Z \backslash (W \cup X)) \cap E_2$. First, note that E_2 is a random subset of $[n] \backslash X$ of size at least $n^{1-\varepsilon_1}/2$ with probability $1 - \exp(-\Omega(n^{1-\varepsilon_1}))$ by Corollary 1. Next, we observe that sets Z, W, X are defined independent of E_2 and hence, by Corollary 1, $|A| \geqslant \frac{1}{4} \cdot n^{1-\varepsilon_1-\varepsilon_2}$ with probability at least $1 - \exp(-\Omega(n^{1-\varepsilon_1-2\varepsilon_2}))$.

Next, we pick a subset $A' \subseteq A$ such that bits in A' have disjoint input neighbors. That is, $\forall i, j \in A', \mathrm{Inp}_h(i) \cap \mathrm{Inp}_h(j) = \emptyset$. We use following two properties to ensure that we can pick A' of sufficiently large size. First, for every bit $i \in A$, $\mathrm{Inp}_h(i) \subseteq [n]$ (because $A \cap W = \emptyset$). Second, all the bits in $[n]$ with more than n^{ε_2} output neighbors in $[n]$ belong to subset P and have already been fixed. This implies that for any bit $i \in A$, all bits in $\mathrm{Inp}_h(i)$ have at most n^{ε_2} output neighbours in $[n]$. Therefore, it is guaranteed that we can pick a set $A' \subseteq A$ s.t. $|A'| \geqslant \frac{|A|}{\delta n^{\varepsilon_2}} = \frac{n^{1-\varepsilon_1-2\varepsilon_2}}{4\delta}$. (This can be done greedily by picking an arbitrary index $i \in A$ and discarding all the bits in A that are influenced by $\mathrm{Inp}_h(i)$. Since h has at output locality δ and each bit in $\mathrm{Inp}_h(i)$ influences at most n^{ε_2}-many bits in A, we discard at most δn^{ε_2} indices from A for picking one index in A'. Now, we recurse on the remaining indices in A.)

For the rest of the proof, we consider such a set A' of size exactly $\frac{n^{1-\varepsilon_1-2\varepsilon_2}}{4\delta}$. We note that for all indices $i \in A'$ following conditions are satisfied (1) c_i is a planted error e_i ($A' \subseteq E_2$); (2) h_i does not always output e_i ($A' \subseteq Z$); (3) the input neighbors of i are all in $[n]$ ($A' \cap W = \emptyset$). For the tampered main codeword to be consistent with recorded errors, we need that for all $i \in A'$, The i-th bit after tampering, i.e. $\widetilde{c}_i$ needs to be equal to e_i. We show that this happens with probability at most $(1 - 1/2^{\delta})^{n^{1-\varepsilon_1-2\varepsilon_2}8\delta}$, which is negligible for $\delta = \xi \cdot \lg n$ when $\boxed{\varepsilon_1 + 2\varepsilon_2 < 1 - \xi}$. Hence, it suffices to output $\perp$ always.

We first argue that all of A' input neighbors are independent uniform bits. We use the fact that A' is of size $\frac{n^{1-\varepsilon_1-2\varepsilon_2}}{4\delta}$ and its (at most $\delta \cdot \frac{n^{1-\varepsilon_1-2\varepsilon_2}}{4\delta}$-many) input neighbors are all from our ECSS codeword with planted errors. Since we have only fixed X of size $o(t)$ from c so far and our ECSS has independence $t \geqslant n^{1-\zeta}$ and $n^{1-\varepsilon_1-2\varepsilon_2} = o(t)$, all the input neighbors of A' are indeed independent uniform bits. Given the uniformly random input, we examine the bits from A' one by one. For any $i \in A'$, there are following two possibilities

- If c_i (i.e., e_i) is the input neighbor of $\widetilde{c}_i$, then since h_i does not always output c_i, there exists a setting of the other (at most) $\delta - 1$ neighbors, such that $\widetilde{c}_i$ is either fixed 0, fixed 1, or flipped e_i. Because of uniformity of value at input neighbors, this setting happens with probability at least $\frac{1}{2^{\delta-1}}$ and when it happens, with probability at least $1/2$, $\widetilde{c}_i \neq e_i$. Hence, $\widetilde{c}_i = e_i$ with probability at most $1 - \frac{1}{2^{\delta}}$. We remove i from A' and recurse on remaining bits.
- If c_i (i.e., e_i) is not the input neighbor of $\widetilde{c}_i$, then since all of the input neighbors are independent of uniform bit e_i, the probability $\widetilde{c}_i = e_i$ is at most $1/2$.

However, we need to address a small subtlety here. Since e_i is not the input neighbor of itself, it can be in the input neighbor of another bit in A'. To keep failure probabilities independent, if such a bit j exists (s.t. e_i is an input neighbor of $\widetilde{c_j}$), we only include i in our witness set of failed indices but we remove both indices i and j before recursing to remaining bits in A'.

H_7:

1. Let $P = \{i | i \in [n], |\mathrm{Out}_f(i) \cap [n]| \geqslant n^{\varepsilon_2}\}$
2. Let $Q = \{i | i \in [n], \mathrm{Out}_f(i) \backslash [n] \neq \emptyset\}$
3. Let $X = P \cup Q$. Sample $a_X \sim U_{\{0,1\}^{|X|}}$
4. Sample a random $E_1 \subseteq X$ s.t. $|E_1| \sim (n, |X|, n^{1-\varepsilon_1})$-hypergeometric distribution
5. For all $i \in E_1$, sample $e_i \sim U_{\{0,1\}}$
6. For all $i \in E_1$, replace a_i with e_i, we get c_X
7. Sample seeds s^L, s^R uniformly from $\{0,1\}^{3\Lambda \log^3 n}$
8. Given s^L, define: $\mathrm{Bad}^L, \mathrm{Leak}^L$ as in $H_3(f, m)$
   Given s^R, define: $\mathrm{Bad}^R, \mathrm{Leak}^R$ as in $H_3(f, m)$
9. Let $\mathcal{L}^L = \mathrm{Bad}^L \cup \mathrm{Leak}^L$ and $\mathcal{L}^R = \mathrm{Bad}^R \cup \mathrm{Leak}^R$.
10. Define mapping f_0 and its output g as in $H_3(f, m)$
11. If $(|\mathcal{L}^L| \geqslant \lambda n^{\beta_1})$ or $(|\mathcal{L}^R| \geqslant \lambda n^{\beta_2})$ or (f_0 does not satisfy Definition 6), output $\perp$
12. (Else) $(\mathrm{ans}, x, y) = \mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0)$.
13. Let S^L, S^R denote indices of s^L, s^R. Then, $h := (f_1)_{(X, S^L, S^R, \mathcal{L}^L, \mathcal{L}^R)|(c_X, s^L, s^R, x, y)}$
14. $V := \{i | i \in [n], \mathrm{Inp}_h(i) \neq \emptyset\}$
15. $W := \{i | i \in [n], \mathrm{Inp}_h(i) \backslash [n] \neq \emptyset\}$
16. $Z := \{i \in [n] | \exists z \in \{0,1\}^{n+n_e}, z_{(X, S^L, S^R, \mathcal{L}^L, \mathcal{L}^R)} = (c_X, s^L, s^R, x, y), h_i(\mathrm{z}) \neq z_i\}$
17. If $\mathrm{ans} = \perp$, output $\perp$
    If $\mathrm{ans} = \text{same*}$ and $|Z \backslash (W \cup X)| \geqslant n^{1-\varepsilon_2}$, output $\perp$
    If $\mathrm{ans} = (E^*, e^*)$ and $|V \backslash W| \geqslant n^{1-\varepsilon_2}$, output $\perp$
18. Sample $a \sim$ ECSS.Enc(m) $|(\mathrm{ECSS.Enc}(m))_X = c_X$
19. Sample a random $E_2 \subseteq [n] \backslash X$ of size $n^{1-\varepsilon_1} - |E_1|$, let $E = E_1 \cup E_2$
20. For all $i \in E_2$, sample $e_i \sim U_{\{0,1\}}$
21. Define c as $c_i = \begin{cases} a_i, & i \notin E \\ e_i, & i \in E \end{cases}$
22. $(\widetilde{E}, \widetilde{e}) = \mathrm{copy}(\mathrm{Sim}_0(\mathcal{L}^L, \mathcal{L}^R, f_0), (E, e))$
23. $(c^L, c^R) \sim \mathrm{NMEnc}_0(E, e)$ s.t. $\mathrm{NMDec}_0(g(c^L, c^R)) = (\widetilde{E}, \widetilde{e})$ and $c^L_{\mathcal{L}^L} = x$, $c^R_{\mathcal{L}^R} = y$
24. $\alpha^L = \mathrm{Embed}^L(s^L, c^L)$, $\alpha^R = \mathrm{Embed}^R(s^R, c^R)$
25. $\widetilde{c} = f_1(c, s^L, \alpha^L, s^R, \alpha^R)$
26. If ans =
    - same*: If $\widetilde{c}_Z = c_Z$, output same*
      (Else) Output $\perp$.
    - (E^*, e^*): If $\mathrm{ECSS.ECorr}(\widetilde{c}) = \perp$, output $\perp$; (Else) $\widetilde{a} = \mathrm{ECSS.ECorr}(\widetilde{c})$
      Define c' as $c'_i = \begin{cases} \widetilde{a}_i, & i \notin \widetilde{E} \\ \widetilde{e}_i, & i \in \widetilde{E} \end{cases}$
      If $c' \neq \widetilde{c}$, output $\perp$; (Else) $\widetilde{m} = \mathrm{ECSS.Dec}(\widetilde{a})$
      Output $\widetilde{m}$

Now, we have shown that either a bit has probability at most $1 - \frac{1}{2^\delta}$ to be consistent or two bits have probability at most $1/2$ to be consistent at the same time. And all of those events are independent, hence, the probability that all the bits are consistent with errors (E, e) is at most $(1 - \frac{1}{2^\delta})^{|A'|/2}$.

Case ans $= (E^*, e^*)$*:* For the case when ans $= (E^*, e^*)$, this hybrid is only different from previous one when $|V \backslash W| \geqslant n^{1-\varepsilon_2}$. We show that if this happens, the output of previous hybrid is not $\perp$ with only negligible probability.

We first pick a $B \subseteq (V \backslash W)$ such that $\forall i, j \in B, \mathrm{Inp}_h(i) \cap \mathrm{Inp}_h(j) = \emptyset$. Similar to above, all the input neighbors of $V \backslash W$ are contained in $[n]$ and have output locality at most n^{ε_2} in $[n]$. Hence, it is guaranteed that we could pick B such that $|B| = \frac{n^{1-2\varepsilon_2}}{\delta}$. (Similar to same* case, this can be done greedily by picking an arbitrary index from $V \backslash W$ into B and removing all the bits its input neighbors have influence on. We only discard at most δn^{ε_2} bits for picking one bit.)

Note that $B \subseteq V$ implies that for all $i \in B$, $\mathrm{Inp}_h(i) \neq \emptyset$ and since all the bits in B has disjoint input neighbors, we have $|\mathrm{Inp}_h(B)| \geqslant |B|$. Now, consider a subset $B' \subseteq B$ such that each bit in B' has an input neighbour in errors E_2. That is,

$$B' = \left\{ i \middle| i \in B, \mathrm{Inp}_h(i) \cap E_2 \neq \emptyset \right\}$$

Again E_2 is a random subset of size at least $n^{1-\varepsilon_1}/2$ with probability $1 - \exp(-\Omega(n^{1-\varepsilon_1}))$ and is independent of B. Thus, by Corollary 1, with probability at least $1 - \exp(-\Omega(n^{1-\varepsilon_1-4\varepsilon_2}))$, $|\mathrm{Inp}_h(B) \cap E_2| \geqslant \frac{1}{4\delta} n^{1-\varepsilon_1-2\varepsilon_2}$. Hence, $|B'| \geqslant \frac{n^{1-\varepsilon_1-2\varepsilon_2}}{4\delta^2}$ (Because of δ-locality).

For the rest of proof, we consider such a set B' of size exactly $\frac{n^{1-\varepsilon_1-2\varepsilon_2}}{4\delta^2}$. Next, we argue that input neighbors of B' (at most $\delta \cdot n^{1-\varepsilon_1-2\varepsilon_2}/(4\delta^2)$ in number) are independently uniformly distributed. This is because they are all from our ECSS codeword with planted errors. Since we have only fixed X of size $o(t)$ from c so far and our ECSS has independence $t = n^{1-\zeta}$ with $\boxed{\zeta < \varepsilon_1 + 2\varepsilon_2}$, all the input neighbors of B' are indeed independent uniform bits. So, bits in B' satisfy the following conditions: its input neighbors (1) are disjoint; (2) contain at least one bit from E_2; (3) are contained in $[n]$; (4) are independently uniform bits.

Next, we define $M = \mathrm{Out}_h(\mathrm{Inp}_h(B'))$. This is the set of all indices that is being influenced by the input neighbor of B'. Obviously $B' \subseteq M$. And the size of M is bounded by $n^{\varepsilon_2} \cdot \delta \cdot n^{1-\varepsilon_1-2\varepsilon_2}/(4\delta^2) = n^{1-\varepsilon_1-\varepsilon_2}/(4\delta)$. We first observe that fix any $c^*_{[n]\backslash M}$, there is at most one c^*_M that is consistent with $c^*_{[n]\backslash M}$ and the fixed errors E^*, e^*. This is because if there exist two $c^{(1)}, c^{(2)}$ s.t. $c^{(1)}_{[n]\backslash M} = c^{(2)}_{[n]\backslash M}$, their distance is bounded by $n^{1-\varepsilon_1-\varepsilon_2}/(4\delta)$ which is smaller than the distance $d \geqslant n^{1-\zeta}$ as long as $\boxed{\zeta < \varepsilon_1 + \varepsilon_2}$. Therefore, those two codewords will be error-corrected to the same correct codeword and after being reconstructed from errors (E^*, e^*), they will be the same. Therefore, for every fixing $c^*_{[n]\backslash M}$, there is at most

one codeword c^* (equivalently, one c^*_M), which is consistent with (E^*, e^*). Since $B' \subseteq M$, there is at most one choice for $c^*_{B'}$ as well.

Finally, we prove that the probability that $c^*_{B'}$ takes the fixed value needed to be consistent is negligible. Now,for any $i \in B'$, we know some bit E_j is the input neighbors of i. Therefore, at least one out of at most $2^{\delta-1}$ possible settings of all the other input neighbors $\mathrm{Inp}_h(i)\backslash[j]$, flipping the value of e_j will flip the output of h_i. Note that by definition of M, E_j cannot be the input neighbors of any bits in $[n]\backslash M$, hence e_j is independent of $c^*_{[n]\backslash M}$. And thus, whenever this setting happens, with probability 1/2, the output at i will not be consistent with (E^*, e^*). Therefore, since the input neighbors of i are uniformly distributed, the probability that $\widetilde{c_i}$ is not consistent with fixed errors (E^*, e^*) is at least $\frac{1}{2^\delta}$. Since all the input neighbors of B' are all independent uniform bits, the probability that all the bits from B' are consistent is at most $(1 - \frac{1}{2^\delta})^{n^{1-\varepsilon_1-2\varepsilon_2}/(4\delta^2)}$, which is negligible when $\delta = \xi \cdot \lg n$ with $\boxed{\varepsilon_1 + 2\varepsilon_2 < 1 - \xi}$.

$H_8(f, m)$: Our final hybrid is identical to our simulator Fig. 3. In this final hybrid, we simply switch message m with 0^ℓ.

Note that the only bits from $\mathrm{ECSS.Enc}(m)$ that affect the output of the hybrid is (1) the neighbors of Z and also c_Z when ans = same* and $|Z\backslash(W \cup X)| < n^{1-\varepsilon_2}$; (2) the neighbors of V, when ans $\notin \{\text{same}^*, \perp\}$ and $|V\backslash W| < n^{1-\varepsilon_2}$.[11] For (1), as shown in hybrid 7, the size of Z is $o(t)$ when $|Z\backslash(W \cup X)| < n^{1-\varepsilon_2}$ and hence the neighbor of $|Z|$ is of size at most $\delta \cdot |Z| = o(t)$. For (2), $|V| \leqslant |V\backslash W| + |W|$. Both are $o(t)$ as require in hybrid 7 and therefore so is $|V|$ and the size of the neighbors of V. Hence the number of bits in c that influence the hybrid output is at most $o(t)$. Any $o(t)$ bits from $\mathrm{ECSS.Enc}(m)$ condition on c_X is uniformly distributed. Hence, we can switch the encoding of m with encoding of 0^ℓ.

This completes our hybrid argument.

References

1. Aggarwal, D., Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Optimal computational split-state non-malleable codes. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 393–417. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_15
2. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. In: STOC (2015)
3. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: STOC (2014)
4. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Explicit non-malleable codes against bit-wise tampering and permutations. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 538–557. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_26

[11] Note that, by the definition of V, all the output bits from $[n]\backslash V$ are fixed to some values with no input neighbors. Hence, it suffices to have the neighbor of V to finish the hybrid completely.

5. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 375–397. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_16
6. Ball, M., Dachman-Soled, D., Guo, S., Malkin, T., Tan, L.-Y.: Non-malleable codes for small-depth circuits. In: FOCS (2018)
7. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes for bounded depth, bounded fan-in circuits. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 881–908. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_31
8. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes from average-case hardness: AC^0, decision trees, and streaming space-bounded tampering. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 618–650. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_20
9. Blakley, G.R., Meadows, C.: Security of ramp schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 242–268. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_20
10. Chattopadhyay, E., Goyal, V., Li, X.: Non-malleable extractors and codes, with their many tampered extensions. In: STOC (2016)
11. Chattopadhyay, E., Li, X.: Non-malleable codes and extractors for small-depth circuits, and affine functions. In: STOC (2017)
12. Chattopadhyay, E., Zuckerman, D.: Non-malleable codes against constant split-state tampering. In: FOCS (2014)
13. Cheraghchi, M., Guruswami, V.: Capacity of non-malleable codes. In: ITCS (2014)
14. Cheraghchi, M., Guruswami, V.: Non-malleable coding against bit-wise and split-state tampering. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 440–464. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_19
15. Chvátal, V.: The tail of the hypergeometric distribution. Discret. Math. **25**(3), 285–287 (1979)
16. Cramer, R., Dodis, Y., Fehr, S., Padró, C., Wichs, D.: Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 471–488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_27
17. Dziembowski, S., Kazana, T., Obremski, M.: Non-malleable codes from two-source extractors. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 239–257. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_14
18. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: ICS (2010)
19. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: Continuous non-malleable codes. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 465–488. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_20
20. Faust, S., Mukherjee, P., Venturi, D., Wichs, D.: Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 111–128. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_7
21. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: STOC (1992)
22. Goyal, V., Kumar, A.: Non-malleable secret sharing. In: STOC (2018)

23. Goyal, V., Kumar, A.: Non-malleable secret sharing for general access structures. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 501–530. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_17
24. Gupta, D., Maji, H.K., Wang, M.: Non-malleable codes against lookahead tampering. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 307–328. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_17
25. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
26. Jafargholi, Z., Wichs, D.: Tamper detection and continuous non-malleable codes. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 451–480. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_19
27. Kanukurthi, B., Obbattu, S.L.B., Sekar, S.: Four-State non-malleable codes with explicit constant rate. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 344–375. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_11
28. Kanukurthi, B., Obbattu, S.L.B., Sekar, S.: Non-malleable randomness encoders and their applications. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 589–617. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_19
29. Li, X.: Improved non-malleable extractors, non-malleable codes and independent source extractors. In: STOC (2017)
30. Li, X.: Pseudorandom correlation breakers, independence preserving mergers and their applications. In: ECCC 25 (2018)
31. Liu, F.-H., Lysyanskaya, A.: Tamper and leakage resilience in the split-state model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 517–532. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_30
32. Nisan, N.: Pseudorandom generators for space-bounded computation. In: STOC (1990)
33. Ostrovsky, R., Persiano, G., Venturi, D., Visconti, I.: Continuously non-malleable codes in the split-state model from minimal assumptions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 608–639. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_21
34. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
35. Viola, E.: Extractors for circuit sources. In: FOCS (2011)