Constant-rate Three-state Non-malleable Code

Divya Gupta* Hemanta K. Maji†, ‡ Mingyuan Wang§, ‡

October 24, 2017

Abstract

Dziembowski, Pietrzak, and Wichs (ICS–2010) introduced the notion of non-malleable codes as a relaxation of the error-correcting codes. Intuitively, a non-malleable code ensures that the tampered codeword encodes the original message or a message that is entirely independent of the original message.

One of the fundamental research directions in the field of non-malleable code construction considers encoding the message into k separates states, where $k \geqslant 2$, such that each state can be tampered separately by an arbitrary function. The goal of this research effort is to reduce the number of states k, while achieving high encoding rate, i.e., the ratio of the message length to the cumulative size of all the k encoded states. The current state-of-the-art provides non-malleable code constructions for 2-states at $1/\log \ell$ rate (Li, STOC–2017), where ℓ is the length of the encoded message, and, very recently, for 4-states at $\approx 1/3$ rate (Kanukurthi, Obbattu, and Sekar, TCC–2017). However, there are no known non-malleable code constructions for $k = 2$ or $k = 3$ with a constant rate. This work contributes to this research endeavor by providing a rate $\approx 1/3$ non-malleable code for $k = 3$.

Reducing the number of states k in a non-malleable code is notoriously hard. However, our work succeeds in reducing the number of states required by the construction of Kanukurthi et al. by leveraging a unique characteristic of the non-malleable code for 2-states provided by Aggarwal, Dodis, and Lovett (STOC, 2014).

*Microsoft Research, Bangalore, India. t-digu@microsoft.com

†Department of Computer Science, Purdue University. hmaji@purdue.edu.

‡The research effort is supported in part by a Purdue Research Foundation grant.

§Department of Computer Science, Purdue University. wang1929@purdue.edu.

Contents

1 Introduction

In the seminal paper [DPW10], Dziembowski, Pietrzak, and Wichs introduced the notion of *non-malleable codes* as a relaxation of error-correcting codes. Intuitively, error-correcting codes allow tampering of a small fraction of the codeword. While, on the other hand, non-malleable codes are resilient to more sophisticated tampering families where the tampering function may replace the entire codeword. Non-malleable codes ensure that the tampered codeword either leaves the original message intact or encodes a message that is entirely independent of the original message.

If we do not restrict the power of the tampering functions, then we cannot achieve non-malleability. In particular, the tampering function can decode the codeword, retrieve the original codeword, and re-encode a new message that is one more than the original message. However, limiting the power of the tampering function family enables the construction of non-malleable codes for that particular function family. For example, tampering functions that perform bit-level perturbation and permutations [DPW10, CG14b, AGM+15], AC^0 tampering functions [CL17], or functions that tamper disjoint parts of the codeword independently [ADL14, CZ14, ADKO15, Li17] are a representative few.

In split-state tampering, we split the codeword into $k \geqslant 2$ states, and the tampering function tampers each state independently. There has been a large body of work on the explicit construction of non-malleable codes against split-state tampering, for $k \geqslant 2$. The priority in this line of research is to reduce k while, simultaneously, achieving a high rate, the ratio of the length of the message to the cumulative length of the codeword. For each k, the maximum possible rate is at most $1 - \frac{1}{k}$ [CG14a]. The ideal goal of this research direction is to achieve rate close to this upper-bound for small values of k. In particular, the reduction of the number of states k is notoriously difficult.

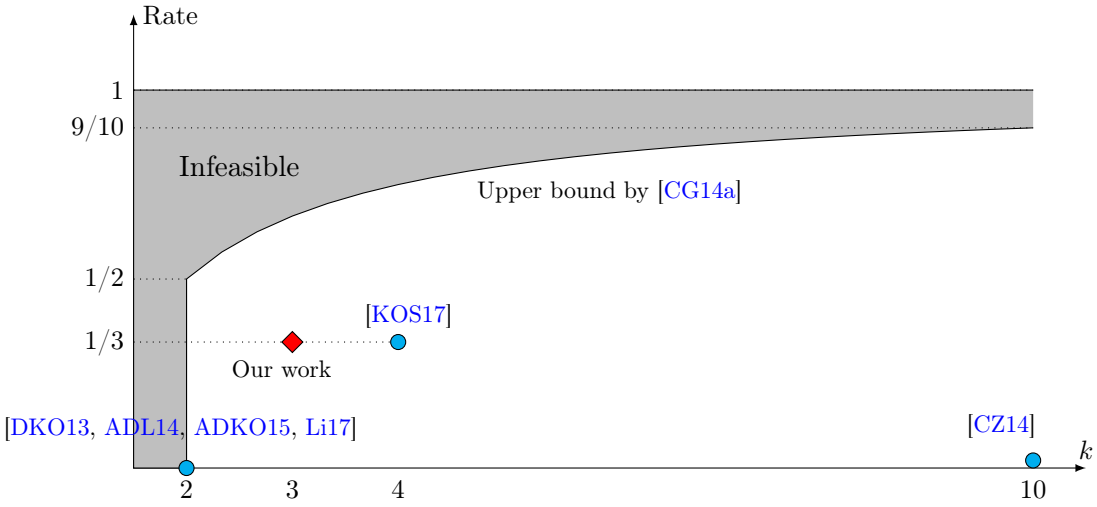

Figure 1: A summary of the research progress.

The current state-of-the-art constructions for $k = 2$ achieve only rate $1/\log \ell$ [Li17] and for $k = 4$, only recently, there is a construction that achieves rate $1/3$ [KOS17]. There is no known constant rate non-malleable code construction for $k = 2$ and $k = 3$. Our work contributes to this research endeavor by constructing the first non-malleable codes for $k = 3$ with rate $1/3$. Note that the rate of our construction is only a multiplicative factor of 2 less than the upper-bound of the maximum achievable rate.

1.1 Our Contribution

In this work, we give the first construction of three-state non-malleable codes that achieves constant rate ($\approx \frac{1}{3}$). Intuitively, we prove the following theorem.

Theorem 1 (Informal)**.** *There exists a three-state non-malleable code with rate arbitrarily close to* $1/3$*.*

For the exact statement of our result, refer to Theorem 2. Our construction leverages a unique property of the 2-split-state non-malleable code construction proposed by [ADL14]. This property, namely, *augmented non-malleability*, allows us to simulate the joint distribution of the left-state of the non-malleable encoding along with the tampered message. Now, using the property mentioned above, we merge two suitable states among the four states of the recent construction of [KOS17], while preserving the rate of their construction. Section 1.3 presents the central intuition underlying our construction.

1.2 Prior Relevant Works

Constructions of non-malleable codes are known against a large number of interesting and sophisticated families of tampering functions. If the size of the tampering function class is upper-bounded, then Monte Carlo constructions of rate-1 non-malleable codes are known, cf., [FMVW14] and [CG14a]. However, explicit constructions are known only for a few tampering families.

Among these families of tampering functions, a widely studied family is the k split-state model. In this model, we encode the message into $k \geqslant 2$ states, and the tampering function tampers the two states independently. It is evident that the goal of reducing k increases the complexity of designing such non-malleable codes, which remains the primary bottleneck of this research direction. The case of $k = 2$ has been well explored [DKO13, ADL14, ADKO15, Li17] and, currently, the most efficient encoding schemes achieve rate $1/\log n$. For higher values of k, Chattopadhyay and Zuckerman [CZ14] construct a (minuscule) constant-rate encoding for $k = 10$, and, recently, Kanukurthi et al. [KOS17] construct a rate-1/3 non-malleable code for $k = 4$.

The search of constant-rate non-malleable codes for $k = 2$ and $k = 3$ remains open. In this work, we settle this problem for $k = 3$ and achieve rate 1/3.

1.3 Technical Overview

$\mathrm{Enc}(m)$:	$\mathrm{Dec}(c_1, c_2, c_3)$:
1. $w \sim U_n$, $s \sim U_d$, $k_1 \sim U_\gamma$, $k_2 \sim U_{\gamma'}$.	1. $\widetilde{c} := c_1$, $(\widetilde{w}, \widetilde{L}) := c_2$, $\widetilde{R} := c_3$.
2. $r = \mathrm{Ext}(w, s)$, $c = m \oplus r$.	2. $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \mathrm{Dec}^*(\widetilde{L}, \widetilde{R})$.
3. $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.	3. If $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \perp$, output $\perp$.
4. $(L, R) \sim \mathrm{Enc}^*(k_1, k_2, t_1, t_2, s)$.	4. If $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
5. Output $\Big(c, (w, L), R\Big)$.	5. Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

Figure 2: Compiling an augmented 2-state non-malleable code $(\mathrm{Enc}^*, \mathrm{Dec}^*)$ into a 3-state non-malleable code.

At a high-level, we design a compiler that compiles a bad rate two-state non-malleable code into a constant rate three-state non-malleable code. Compared to the original two-state non-malleable code, our complier significantly increases the rate by introducing one additional state.

Our design is similar to the construction of [KOS17]. We use a one-time pad to mask the original message m, i.e., we compute the cipher-text $c = m \oplus r$. The randomness r itself is extracted from a larger source w and a seed s using an (average) min-entropy extractor Ext, i.e., we have $r = \mathrm{Ext}(w, s)$. Furthermore, we sample a short key k and generate a short tag t using an (information-theoretic) message authentication code for both the source w and cipher-text c. Finally, we encode the keys and tags together with the seed by a two-state non-malleable code $(\mathrm{Enc}^*, \mathrm{Dec}^*)$, i.e., $(L, R) = \mathrm{Enc}^*(\mathrm{key}, \mathrm{tag}, \mathrm{seed})$. Our key observation is that instead of storing the cipher-text c, the source w and the two-states L, R of the encoding into 4 separate states, we can

store the source w along with one of the states, say, L in one state. Using the *augmented* non-malleability of the split state encoding, we prove the non-malleability of our compiler using hybrid-arguments. So, our construction uses only three states, namely, c, (w, L) and R.

To help better understand the coding scheme, we draw Figure 3. In this outline, the length of blue nodes are long (linear in message length) and the length of red nodes are short (sub-linear in message length). Hence, we use bad rate augmented 2-state non-malleable code to encode all the information in the red nodes, and store w and c directly.

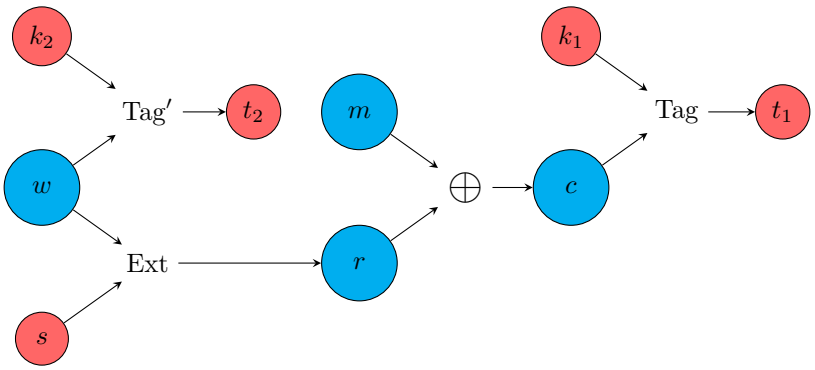

Figure 3: A pictorial summary of our non-malleable code construction.

To argue the non-malleability of our construction, we need to show that for all tampering functions f, g, h, such that $\widetilde{c} = f(c)$, $(\widetilde{w}, \widetilde{L}) = g(w, L)$, $\widetilde{R} = g(R)$, the decoding of $\widetilde{c}, \widetilde{w}, \widetilde{L}, \widetilde{R}$ can be simulated without m. The primary bottleneck we face is the combining of w and L into one state, because the tampering on w and L will be correlated with each other (i.e., the tampered source $\widetilde{w}$ depends on both w and L, and so does the tampered left-state of the 2-state encoding $\widetilde{L}$). To solve this, we decompose the function f into two functions f_w and f_L, which are functions defined to be f with the first (resp. second) input restricted to be w (resp. L). First, note that w is independent of m and, hence, we can simulate f_w independent of m, which will give us a pair of tampering functions onto the two-state encoding. Moreover, by the non-malleability of $(\mathsf{Enc}^*, \mathsf{Dec}^*)$, we can replace this step with its simulator Sim^*.

However, the left state of the 2-state non-malleable code L is still correlated with m and we cannot simulate it without some additional guarantees. We recall a *augmented* property of the two-state construction in [ADL14] that is observed and introduced in [AAG+16]. This augmented property of two-state non-malleable codes ensures that not only could we simulate the decoding Ans of the tampered codeword, we can even simulate the joint distribution of the left state and decoding of the tampered codeword, i.e., the joint distribution (L, Ans). Note that the two-state non-malleable code requires that Ans can be simulated without the message, which actually implies that L can be simulated without the message as well. However, their joint distribution is not guaranteed by the definition of two-state non-malleable codes to be simulatable. Now, using this augmented property, we can simulate the output (L, Ans) with only the knowledge of tampering functions (g_w, h) and therefore, we could further use the output L to simulate the tampering f_L onto w.[1]

Now, we break the result of Ans into three categories based on whether it is invalid, the original message, or some fixed messages. That is, we check whether Ans is $\perp$, same* or $(k_1^*, k_2^*, t_1^*, t_2^*, s^*)$. If $\mathsf{Ans} = \perp$ then this is an easy case, because we can just output $\perp$, which is independent of the message m. However, for both same* and $(k_1^*, k_2^*, t_1^*, t_2^*, s^*)$ cases, we still need to further check if the tags are valid, extract the one-time pad $\widetilde{r} = \mathsf{Ext}(\widetilde{w}, \widetilde{s})$, and decode the message $\widetilde{m} = \widetilde{c} \oplus \widetilde{r}$. These steps are correlated with the message m.

We first notice that by the property of message authentication code, in the $\mathsf{Ans} = \text{same}^*$ case, we do not need to actually check if the tag is correct and instead we can simply check if the message is still the same or not. This step removes the dependence of the tampering experiment with the original "seed, keys, and tags" (since they are also not used in fixed cases).

Next, we observe that we only require a small amount of information from w in our hybrid. In fact, we only need a small amount of leakage $\mathcal{L}(w)$ of w to continue with our hybrid. And this leakage $\mathcal{L}(w)$

[1] The hybrid-argument is not clear when we decide to merge the cipher text c with the left-state L. So, it was crucial for us to merge the source w with the left-state L.

provides the following information. (1) the output Ans of the simulator Sim^* that simulates the tampering experiment of $(\mathrm{Enc}^*, \mathrm{Dec}^*)$ tampered by (g_w, h). (2) whether w remains the same after it is tampered by g_L when Ans = same*. (3) whether the tampered w could pass the check of message authentication code when Ans is the fixed case. (4) the new mask $\widetilde{r} = \mathrm{Ext}(\widetilde{w}, s^*)$ in fixed cases. By the property of our min-entropy extractor, we argue that even conditioned on this leakage $\mathcal{L}(w)$, $\mathrm{Ext}(w, s)$ is close to uniform, i.e., the joint distribution $(\mathcal{L}(w), \mathrm{Ext}(w, s)) \approx (\mathcal{L}(w), U)$. So, we can replace the extraction step with independent uniform bits. Finally, by the perfect secrecy of the one-time pad, this last step removes the dependence on m and completes our hybrid argument from tampering experiment to our simulator.

In order for our last step the go through, we requires that w has length more than 2ℓ, where ℓ is the message length. Intuitively, this is because the dominant part of the leakage is $\widetilde{r} = \mathrm{Ext}(\widetilde{w}, \widetilde{s})$, which is as long as the message. To extract ℓ-bit randomness for the one-time pad. Hence, we will need roughly 2ℓ-bit randomness.

2 Preliminaries

We use $\mathrm{SD}(\mathcal{X}, \mathcal{Y})$ to denote the statistical distance between $\mathcal{X}$ and $\mathcal{Y}$. We write $x \sim \mathcal{X}$ when x is draw according to distribution $\mathcal{X}$. For all $n \in \mathbb{N}$, we use U_n to denote the uniform distribution on $\{0,1\}^n$. For function $f : \{0,1\}^p \times \{0,1\}^q \longrightarrow \{0,1\}^p \times \{0,1\}^q$ and every $x \in \{0,1\}^p$, define function $f_x : \{0,1\}^q \longrightarrow \{0,1\}^q$ to be $f_x(y) = \widetilde{y}$ if and only if $f(x, y) = (\widetilde{x}, \widetilde{y})$. Similarly we define $f_y(x) = \widetilde{x}$.

Definition 1 ((n, k) coding scheme)**.** *A coding scheme with message length k and code length n consists of two functions. A possibly probabilistic encryption function* $\mathrm{Enc} : \{0,1\}^k \longrightarrow \{0,1\}^n$ *and a deterministic decryption function* $\mathrm{Dec} : \{0,1\}^n \longrightarrow \{0,1\}^k \cup \{\perp\}$ *such that for all $m \in \{0,1\}^k$,* $\mathrm{Pr}_r[\mathrm{Dec}(\mathrm{Enc}(m; r))] = 1$*, where r is the private randomness of* Enc*.*

Definition 2. *A family of hash function $\left\{h_k : \{0,1\}^\alpha \longrightarrow \{0,1\}^\beta\right\}_{k \in K}$ is said to be μ-almost pairwise independent hash function if for all $m \in \{0,1\}^\alpha, t \in \{0,1\}^\beta$,*

$$\Pr_{k \sim U_K}[h_k(m) = t] = 2^{-\beta}$$

and for all $m, m' \in \{0,1\}^\alpha$ such that $m \neq m'$ and $t, t' \in \{0,1\}^\beta$

$$\Pr_{k \sim U_K}\left[h_k(m) = t \wedge h_k(m') = t'\right] \leqslant \mu \cdot 2^{-\beta}$$

Definition 3. *Function* $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^\ell$ *is called (k, ε)-extractor if for all distribution $\mathcal{X}$ such that $H_\infty(\mathcal{X}) \geqslant k$, we have* $\mathrm{Ext}(\mathcal{X}, U_d) \approx_\varepsilon U_\ell$*.*

Definition 4 (Average conditional min-entropy)**.** *The average conditional min-entropy of a distribution $\mathcal{X}$ condition on $\mathcal{W}$ is defined to be*

$$\widetilde{H}_\infty(\mathcal{X}|\mathcal{W}) = -\log\left(\mathbb{E}_{w \sim \mathcal{W}}\left[2^{-H_\infty(\mathcal{X}|\mathcal{W}=w)}\right]\right)$$

We will need the following lemma on average conditional min-entropy.

Lemma 1. *If the size of the support of $\mathcal{W}$ does not exceed 2^λ, then $\widetilde{H}_\infty(\mathcal{X}|\mathcal{W}) \geqslant H_\infty(\mathcal{X}) - \lambda$.*

2.1 Non-malleable codes

Let $\mathrm{copy}(x, y)$ to be the following function

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

Definition 5 (Non-malleable code)**.** *A coding scheme* $(\mathrm{Enc}, \mathrm{Dec})$ *with message length* k *is said to be non-malleable against tampering family* $\mathcal{F}$ *with error* ε *if for all function* $f \in \mathcal{F}$*, there exists a distribution* Sim_f *over* $\{0,1\}^k \cup \{\perp\} \cup \{\text{same}^*\}$ *such that for every* $m \in \{0,1\}^k$*,*

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

where Tamper_f^m *stands for the following tampering experiment*

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

Definition 6 (Split-state Tampering family)**.** *The tampering family* $\mathcal{F}_{p_1,p_2,\ldots,p_q}$ *such that* $p_1 + p_2 + \cdots + p_q = n$ *is a* q*-state tampering family on* $\{0,1\}^n$ *that consists of all functions* $f = (f_1, f_2, \ldots, f_q)$ *where* $f_i : \{0,1\}^{p_i} \longrightarrow \{0,1\}^{p_i}$ *and tampering on codeword* $c = (c_1, c_2, \ldots, c_q)$ *is done by*

$$f(c) = \Big(f_1(c_1), f_2(c_2), \ldots, f_q(c_q)\Big)$$

When there are just two states, we usually write (c_1, c_2) *as* (L, R)*.*

3 Construction of 3-state Non-malleable code

In this section, we will present our construction for 3-state non-malleable code and prove the following theorem.

Theorem 2. *For everiy constant* $\alpha > 0$*, there exists a constant* c *and a 3-state non-malleable coding scheme with rate* $\frac{1}{3+\alpha}$ *and error* 2^{-n^c}*.*

Before we jump into construction, we will need the following building blocks and results.

3.1 Building blocks

Augmented 2-state non-malleable code

In our construction, we need a stronger version of 2-state non-malleable code, namely augmented 2-state non-malleable code. This notion is introduced by [AAG+16].

Definition 7. *Suppose* $(\mathrm{Enc}, \mathrm{Dec})$ *is a coding scheme with message length* k *and code length* $n = p + q$*. We say it is an augmented 2-state non-malleable code against* $\mathcal{F}_{p,q}$ *with error* ε*, if for all* $(f, g) \in \mathcal{F}_{p,q}$*, there exists a distribution* $\mathrm{Sim}_{f,g}^*$ *over* $\{0,1\}^p \times \Big(\{0,1\}^k \cup \{\text{same}^*, \perp\}\Big)$ *such that for all* $m \in \{0,1\}^k$*,*

$$\mathrm{TamperPlus}_{f,g}^m \approx_\varepsilon \mathrm{copy}(\mathrm{Sim}_{f,g}^*, m)$$

where TamperPlus *is defined to be*

$$\mathrm{TamperPlus}_{f,g}^m := \left\{ \begin{array}{c} (L, R) \sim \mathrm{Enc}(m),\ \widetilde{L} = f(L),\ \widetilde{R} = g(R) \\ \textit{Output } \Big(L, \mathrm{Dec}(\widetilde{L}, \widetilde{R})\Big) \end{array} \right\}$$

Notice that we abuse function $\mathrm{copy}(\cdot,\cdot)$ here. $\mathrm{Sim}_{f,g}^* = (x_1, x_2)$ has two entries and $\mathrm{copy}(\mathrm{Sim}_{f,g}^*, m)$ will output (x_1, m) if $x_2 = \text{same}^*$ and (x_1, x_2) if $x_2 \neq \text{same}^*$.

It is proven in [AAG+16] that there exists an augmented 2-state non-malleable code with $1/\operatorname{poly}$ rate and negligible error.

Theorem 3 (Theorem 1 in [AAG+16])**.** *There exists a fixed polynomial* p *such that for all* k*, there exists a* (n, k) *coding scheme that is augmented 2-state non-malleable code with error* $2^{-\lambda}$ *and* $n \leqslant p(k, \lambda)$*.*

Average min-entropy extractor

Definition 8. *We say* $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^\ell$ *is a* (k,ε)*-average min-entropy extractor if for every joint distribution* $(\mathcal{X},\mathcal{Y})$ *such that* $\widetilde{H}_\infty(\mathcal{X}|\mathcal{Y}) \geqslant k$*, we have that* $(\mathrm{Ext}(\mathcal{X},U_d),\mathcal{Y}) \approx_\varepsilon (U_\ell,\mathcal{Y})$*.*

It is proved in [Vad12] that any extractor is also a average min-entropy extractor with only a loss of constant factor on error.

Lemma 2 (Problem 6.8 in [Vad12])**.** *If* $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^\ell$ *is a* (k,ε) *extractor, it is also a* $(k,3\varepsilon)$*-average min-entropy extractor.*

Combined this results with the following known construction for extractors, we have that there exists average min-entropy extractor that require seed length $O(\log n + \log(1/\varepsilon))$ and extracts randomness with length arbitrary close to conditional min-entropy.

Theorem 4 ([GUV07])**.** *For all constant* $\alpha > 0$ *and all integers* $n \geqslant k$*, there exists an efficient* (k,ε)*-extractor* $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^\ell$ *with* $d = O(\log n + \log(1/\varepsilon))$ *and* $\ell = (1-\alpha)k$*.*

Message authentication code

Definition 9. *A* μ*-secure message authentication code (MAC) is a family of pairs of function*

$$\left\{\begin{array}{c}\mathrm{Tag}_k : \{0,1\}^\alpha \longrightarrow \{0,1\}^\beta \\ \mathrm{Verify}_k : \{0,1\}^\alpha \times \{0,1\}^\beta \longrightarrow \{0,1\}\end{array}\right\}_{k\in K}$$

such that
(1) For all m,k, $\mathrm{Verify}_k(m,\mathrm{Tag}_k(m)) = 1$.
(2) For all $m \neq m'$ *and* t,t', $\Pr_{k\sim U_K}[\mathrm{Tag}_k(m) = t \mid \mathrm{Tag}_k(m') = t'] \leqslant \mu$.

Message authentication code could be constructed from μ-almost pairwise hash function family with the key length only as long as $2\log(1/\mu)$. For the completeness of our proof, we give a construction in the Appendix D.

3.2 Construction

Now we are ready to construct a constant rate 3-state non-malleability code. We need the following tools for our construction.

- $(\mathrm{Tag},\mathrm{Verify})$ is a μ-secure message authentication code with message length ℓ, Tag length β and key length γ.
- $(\mathrm{Tag}',\mathrm{Verify}')$ is a μ'-secure message authentication code with message length n, Tag length β' and key length γ'.
- Ext is a (k,ε_1) average min-entropy extractor with source length n, seed length d and output length ℓ.
- $(\mathrm{Enc}^*,\mathrm{Dec}^*)$ is a ε_2-secure augmented 2-state non-malleable code. This coding scheme has message length $\ell^* = (\beta+\beta'+\gamma+\gamma'+d)$ and code length p_1+p_2. We denote its code word by (L,R) and its augmented simulator by Sim^*, which given tampering functions will output (L,Ans).

Construction Overview: Our construction will first draw at random the source w and seed s. Next, we will use the extracted randomness $r = \mathrm{Ext}(w,s)$ as a one-time pad to mask the message $c = m \oplus r$. Then draw at random two keys k_1,k_2 and compute the Tag for the source and cipher text, $t_1 = \mathrm{Tag}_{k_1}(c)$ and $t_2 = \mathrm{Tag}'_{k_2}(w)$. Finally we will encode (k_1,k_2,t_1,t_2,s) using Enc^*, i.e. $(L,R) \sim \mathrm{Enc}^*(k_1,k_2,t_1,t_2,s)$. Our three state will be c, (w,L) and R respectively. A formal description of our coding scheme is showed in Figure 2.

4 Proof of Non-malleability

The perfect correctness of our coding scheme is trivial. We shall prove its non-malleability in this section. To prove the non-malleability of our coding scheme, we shall first define our simulator in Figure 4. In our simulator, we define a leakage function $\mathcal{L}(w)$ that captures all the leakage we need of w to proceed with our hybrid. It consists of four parts, $\text{Ans}, \text{flag}_1, \text{flag}_2$ and mask. Ans is the output we sample from the simulator of $(\text{Enc}^*, \text{Dec}^*)$ which depends on w because the tampering function g_w depends on w. (Recall that g_w is the function $g(w, L) = (\widetilde{w}, \widetilde{L})$ with the first input restricted as w and only output the second part) Next when $\text{Ans} = \text{same}^*$, we need a one-bit leakage flag_1 that tells us whether $\widetilde{w} = w$. And when $\text{Ans} = (\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$, we need another one-bit leakage flag_2 that tells us whether $\widetilde{w}$ can pass the MAC verification check and when it does, we further need leakage mask, which is the tampered one-time pad for the decoding of message.

1. $w \sim U_n$.
2. Define leakage function $\mathcal{L}(w) : \{0,1\}^n \longrightarrow \{0,1\}^{\beta+\beta'+\gamma+\gamma'+d+1} \times \{0,1\} \times \{0,1\} \times \{0,1\}^\ell$ as the following function:
   (a) $(L, \text{Ans}) \sim \text{Sim}^*_{g_w,h}$, $\widetilde{w} = g_L(w)$.
   (b) If Ans =
      Case $\perp$: $\text{flag}_1 = 0$, $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
      Case same*: If $(\widetilde{w} = w)$, $\text{flag}_1 = 1$; Else $\text{flag}_1 = 0$.
      $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
      Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: $\text{flag}_1 = 0$.
      If $\left(\text{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\right) = 1$, $\text{flag}_2 = 1$; Else $\text{flag}_2 = 0$.
      Let $\text{mask} = \text{Ext}(\widetilde{w}, \widetilde{s})$.
   (c) $\mathcal{L}(w) := (\text{Ans}, \text{flag}_1, \text{flag}_2, \text{mask})$.
3. $r \sim U_\ell$, $c = 0^\ell \oplus r$, $\widetilde{c} = f(c)$.
4. If Ans =
   Case $\perp$: Output $\perp$.
   Case same*: If $\left(\widetilde{c} = c \text{ and } \text{flag}_1\right) = 1$, output same*.
   Else output $\perp$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\text{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1}) \text{ and } \text{flag}_2\right) = 0$, output $\perp$.
   Else output $\widetilde{c} \oplus \text{mask}$.

Figure 4: Simulator $\text{Sim}_{f,g,h}$

We will use a series of statistically close hybrids to show that

$$\left\{ \begin{array}{c} (c, (w, L), R) \sim \text{Enc}(m). \\ \widetilde{c} = f(c),\ (\widetilde{w}, \widetilde{L}) = g(w, L),\ \widetilde{R} = h(R). \\ \text{Output: } \tilde{m} = \text{Dec}\left(\widetilde{c}, (\widetilde{w}, \widetilde{L}), \widetilde{R}\right). \end{array} \right\} = \text{Tamper}^m_{f,g,h} \approx \text{copy}\left(\text{Sim}_{f,g,h}, m\right)$$

Our first hybrid is exactly the same as $\text{Tamper}^m_{f,g,h}$. We just open up the definition of Enc and Dec.

$H_0(f, g, h, m)$:

1. $w \sim U_n,\ s \sim U_d,\ k_1 \sim U_\gamma,\ k_2 \sim U_{\gamma'}$.
2. $r = \mathrm{Ext}(w, s)$, $c = m \oplus r$, $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.
3. $(L, R) \sim \mathrm{Enc}^*(k_1, k_2, t_1, t_2, s)$.
4. $\widetilde{c} = f(c)$, $(\widetilde{w}, \widetilde{L}) = g(w, L)$, $\widetilde{R} = h(R)$.
5. $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \mathrm{Dec}^*(\widetilde{L}, \widetilde{R})$.
6. If $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \perp$, output $\perp$.
7. Else if $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
8. Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

Now, we want to rewrite $(\widetilde{w}, \widetilde{L}) = g(w, L)$ as $\widetilde{w} = g_L(w)$ and $\widetilde{L} = g_w(L)$, so that we could create a tampering experiment onto our 2-state non-malleable code. This gives hybrid $H_1(f, g, h, m)$. Since we do not change anything essentially, this hybrid the same as $H_0(f, g, h, m)$.

$H_1(f, g, h, m)$:

1. $w \sim U_n,\ s \sim U_d,\ k_1 \sim U_\gamma,\ k_2 \sim U_{\gamma'}$.
2. $r = \mathrm{Ext}(w, s)$, $c = m \oplus r$, $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.
3. $\widetilde{c} = f(c)$.
4. $(L, R) \sim \mathrm{Enc}^*(k_1, k_2, t_1, t_2, s)$.
5. $\widetilde{L} = g_w(L)$, $\widetilde{R} = h(R)$.
6. $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \mathrm{Dec}^*(\widetilde{L}, \widetilde{R})$.
7. $\widetilde{w} = g_L(w)$.
8. If $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \perp$, output $\perp$.
9. Else if $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
10. Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

Notice that step 4,5,6 in $H_1(f, g, h, m)$ is exactly $\mathrm{TamperPlus}_{g_w, h}^{(k_1, k_2, t_1, t_2, s)}$, replace this with simulator $\mathrm{Sim}^*_{g_w, h}$ gives us $H_2(f, g, h, m)$ and we have the following lemma.

Lemma 3. *If* $(\mathrm{Enc}^*, \mathrm{Dec}^*)$ *is a* ε_2*-secure augmented 2-state non-malleable codes,* $H_1(f, g, h, m) \approx_{\varepsilon_2} H_2(f, g, h, m)$.

$H_2(f, g, h, m)$:

1. $w \sim U_n,\ s \sim U_d,\ k_1 \sim U_\gamma,\ k_2 \sim U_{\gamma'}$.
2. $r = \mathrm{Ext}(w, s)$, $c = m \oplus r$, $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.
3. $\widetilde{c} = f(c)$.
4. $(L, \mathrm{Ans}) \sim \mathrm{Sim}^*_{g_w, h}$.
5. $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \mathrm{copy}\Big(\mathrm{Ans}, (k_1, k_2, t_1, t_2, s)\Big)$.
6. $\widetilde{w} = g_L(w)$.
7. If $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s}) = \perp$, output $\perp$.
8. Else if $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
9. Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

Now in hybrid $H_3(f, g, h, m)$, instead of doing copy(), we specifically write down what the hybrid would do for each case of Ans. Since we do not change anything, the two hybrid are exactly the same $H_2(f, g, h, m) = H_3(f, g, h, m)$.

$H_3(f,g,h,m)$:

1. $w \sim U_n,\ s \sim U_d,\ k_1 \sim U_\gamma,\ k_2 \sim U_{\gamma'}$.
2. $r = \mathrm{Ext}(w,s)$, $c = m \oplus r$, $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.
3. $\widetilde{c} = f(c)$.
4. $(L, \mathrm{Ans}) \sim \mathrm{Sim}^*_{g_w,h}$.
5. $\widetilde{w} = g_L(w)$.
6. If Ans =

Case $\perp$: Output $\perp$.

Case same*: If $\Big(\mathrm{Verify}_{k_1}(\widetilde{c}, t_1)$ and $\mathrm{Verify}'_{k_2}(\widetilde{w}, t_2)\Big) = 0$, output $\perp$.
Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, s)$.

Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

Consider the case when Ans = same*. In this case, the key and tag remain the same and we want to use the property of message authentication code to say that the probability that f and g_L could successfully forge a different message that has the same tag under the same key is small. Hence, instead of using Verify, we would just check if the ciphertext c and source w are the same. This gives us $H_4(f,g,h,m)$ and we have the following lemma.

Lemma 4. *If* $(\mathrm{Tag}, \mathrm{Verify})$ *and* $(\mathrm{Tag}', \mathrm{Verify}')$ *are* μ *and* μ'*-secure message authentication code, then* $H_3(f,g,h,m) \approx_{\mu+\mu'} H_4(f,g,h,m)$.

$H_4(f,g,h,m)$:

$\mathrm{copy}\Big($

1. $w \sim U_n,\ s \sim U_d,\ k_1 \sim U_\gamma,\ k_2 \sim U_{\gamma'}$.
2. $r = \mathrm{Ext}(w,s)$, $c = m \oplus r$, $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.
3. $\widetilde{c} = f(c)$.
4. $(L, \mathrm{Ans}) \sim \mathrm{Sim}^*_{g_w,h}$.
5. $\widetilde{w} = g_L(w)$.
6. If Ans =

Case $\perp$: Output $\perp$.

Case same*: If $\Big(\widetilde{c} = c$ and $\widetilde{w} = w\Big) = 1$, output same*.
Else output $\perp$.

Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

$, m\Big)$

Now notice that the blue part k_1, k_2, t_1, t_2, s are no longer used in the hybrid, we could remove those steps and clean up the hybrid to get $H_5(f,g,h,m)$. Obviously, $H_4(f,g,h,m) = H_5(f,g,h,m)$.

$H_5(f,g,h,m)$:

$\mathrm{copy}\Big($

1. $w \sim U_n,\ s \sim U_d,\ r = \mathrm{Ext}(w,s),\ c = m \oplus r$.
2. $(L, \mathrm{Ans}) \sim \mathrm{Sim}^*_{g_w,h}$.
3. $\widetilde{c} = f(c),\ \widetilde{w} = g_L(w)$.
4. If Ans =

Case $\perp$: Output $\perp$.

Case same*: If $\Big(\widetilde{c} = c$ and $\widetilde{w} = w\Big) = 1$, output same*.
Else output $\perp$.

Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

$, m\Big)$

Now, we wish to use the property of average min-entropy extractor to remove the dependence between c and w. Before we do the trick, we shall first rearrange the steps in $H_5(f,g,h,m)$ to get $H_6(f,g,h,m)$. We process all the leakage we need at the first part of our hybrid and use only the leakage of w in the remaining. Intuitively, when Ans = same*, flag_1 is the flag that whether $\widetilde{w} = w$ and when Ans = $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$, flag_2 is the flag whether $\widetilde{w}$ can pass the MAC verification and mask is the one-time pad we need for the decoding. Again $H_5(f,g,h,m) = H_6(f,g,h,m)$.

$H_6(f,g,h,m)$:

copy $\Big($

1. $w \sim U_n$.
2. $(L, \text{Ans}) \sim \text{Sim}^*_{g_w,h}$, $\widetilde{w} = g_L(w)$.
3. If Ans =
   Case same*: If $(\widetilde{w} = w)$, $\text{flag}_1 = 1$; Else $\text{flag}_1 = 0$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\text{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\right) = 1$, $\text{flag}_2 = 1$; Else $\text{flag}_2 = 0$.
   Let mask = $\text{Ext}(\widetilde{w}, \widetilde{s})$.
4. $s \sim U_d$, $r = \text{Ext}(w,s)$, $c = m \oplus r$, $\widetilde{c} = f(c)$.
5. If Ans =
   Case $\perp$: Output $\perp$.
   Case same*: If $\left(\widetilde{c} = c \text{ and } \text{flag}_1\right) = 1$, output same*.
   Else output $\perp$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\text{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1}) \text{ and } \text{flag}_2\right) = 0$, output $\perp$.
   Else output $\widetilde{c} \oplus \text{mask}$.

$, m\Big)$

In the next hybrid, we formalize the red part in $H_6(f,g,h,m)$ as a leakage function onto w.

$H_7(f,g,h,m)$:

copy $\Big($

1. $w \sim U_n$.
2. Define leakage function $\mathcal{L}(w) : \{0,1\}^n \longrightarrow \{0,1\}^{\beta+\beta'+\gamma+\gamma'+d+1} \times \{0,1\} \times \{0,1\} \times \{0,1\}^\ell$ as the following function:
   (a) $(L, \text{Ans}) \sim \text{Sim}^*_{g_w,h}$, $\widetilde{w} = g_L(w)$.
   (b) If Ans =
   Case $\perp$: $\text{flag}_1 = 0$, $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
   Case same*: If $(\widetilde{w} = w)$, $\text{flag}_1 = 1$; Else $\text{flag}_1 = 0$.
   $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: $\text{flag}_1 = 0$.
   If $\left(\text{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\right) = 1$, $\text{flag}_2 = 1$; Else $\text{flag}_2 = 0$.
   Let mask = $\text{Ext}(\widetilde{w}, \widetilde{s})$.
   (c) $\mathcal{L}(w) := (\text{Ans}, \text{flag}_1, \text{flag}_2, \text{mask})$.
3. $s \sim U_d$, $r = \text{Ext}(w,s)$, $c = m \oplus r$, $\widetilde{c} = f(c)$.
4. If Ans =
   Case $\perp$: Output $\perp$.
   Case same*: If $\left(\widetilde{c} = c \text{ and } \text{flag}_1\right) = 1$, output same*.
   Else output $\perp$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\text{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1}) \text{ and } \text{flag}_2\right) = 0$, output $\perp$.
   Else output $\widetilde{c} \oplus \text{mask}$.

$, m\Big)$

Notice that $\mathcal{L}(w)$ is all the leakage on the source w we need to carry on with our hybrid. By the property of min-entropy extractor, as long as the conditional min-entropy is large enough, the extracted string is uniformly random even condition on the leakage. This gives us hybrid $H_8(f,g,h,m)$ and the following lemma.

Lemma 5. *If* Ext *is a* (k, ε_1) *min-entropy extractor,* $H_7(f,g,h,m) \approx_{\varepsilon_1} H_8(f,g,h,m)$.

$H_8(f,g,h,m)$:

copy $\Big($

1. $w \sim U_n$.
2. Define leakage function $\mathcal{L}(w) : \{0,1\}^n \longrightarrow \{0,1\}^{\beta+\beta'+\gamma+\gamma'+d+1} \times \{0,1\} \times \{0,1\} \times \{0,1\}^\ell$ as the following function:
   (a) $(L, \text{Ans}) \sim \text{Sim}^*_{g_w,h}$, $\widetilde{w} = g_L(w)$.
   (b) If Ans =
   Case $\perp$: $\text{flag}_1 = 0$, $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
   Case same*: If $(\widetilde{w} = w)$, $\text{flag}_1 = 1$; Else $\text{flag}_1 = 0$.
   $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: $\text{flag}_1 = 0$.
   If $\left(\text{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\right) = 1$, $\text{flag}_2 = 1$; Else $\text{flag}_2 = 0$.
   Let $\text{mask} = \text{Ext}(\widetilde{w}, \widetilde{s})$.
   (c) $\mathcal{L}(w) := (\text{Ans}, \text{flag}_1, \text{flag}_2, \text{mask})$.
3. $r \sim U_\ell$, $c = m \oplus r$, $\widetilde{c} = f(c)$.
4. If Ans =
   Case $\perp$: Output $\perp$.
   Case same*: If $\left(\widetilde{c} = c \text{ and } \text{flag}_1\right) = 1$, output same*.
   Else output $\perp$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\text{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1}) \text{ and } \text{flag}_2\right) = 0$, output $\perp$.
   Else output $\widetilde{c} \oplus \text{mask}$.

$, m\Big)$

Finally, notice that the distribution of c is independent of m and we can fix m to be 0 message. This gives us our simulator. Clearly $H_8(f,g,h,m) = H_9(f,g,h,m)$. Notice that $H_9(f,g,h,m) = \text{copy}\left(\text{Sim}_{f,g,h}, m\right)$.

$H_9(f,g,h,m)$:

copy $\Big($

1. $w \sim U_n$.
2. Define leakage function $\mathcal{L}(w) : \{0,1\}^n \longrightarrow \{0,1\}^{\beta+\beta'+\gamma+\gamma'+d+1} \times \{0,1\} \times \{0,1\} \times \{0,1\}^\ell$ as the following function:
   (a) $(L, \text{Ans}) \sim \text{Sim}^*_{g_w,h}$, $\widetilde{w} = g_L(w)$.
   (b) If Ans =
   Case $\perp$: $\text{flag}_1 = 0$, $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
   Case same*: If $(\widetilde{w} = w)$, $\text{flag}_1 = 1$; Else $\text{flag}_1 = 0$.
   $\text{flag}_2 = 0$, $\text{mask} = 0^\ell$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: $\text{flag}_1 = 0$.
   If $\left(\text{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\right) = 1$, $\text{flag}_2 = 1$; Else $\text{flag}_2 = 0$.
   Let $\text{mask} = \text{Ext}(\widetilde{w}, \widetilde{s})$.
   (c) $\mathcal{L}(w) := (\text{Ans}, \text{flag}_1, \text{flag}_2, \text{mask})$.
4. $r \sim U_\ell$, $c = 0^\ell \oplus r$, $\widetilde{c} = f(c)$.
5. If Ans =
   Case $\perp$: Output $\perp$.
   Case same*: If $\left(\widetilde{c} = c \text{ and } \text{flag}_1\right) = 1$, output same*.
   Else output $\perp$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\text{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1}) \text{ and } \text{flag}_2\right) = 0$, output $\perp$.
   Else output $\widetilde{c} \oplus \text{mask}$.

$, m\Big)$

We prove our lemmata in the appendix. Now by the closeness of hybrids, we have

$$\begin{aligned}\mathrm{Tamper}^{m}_{f,g,h} &= H_0(f,g,h,m) = H_1(f,g,h,m) \approx_{\varepsilon_2} H_2(f,g,h,m) = H_3(f,g,h,m) \approx_{\mu+\mu'} H_4(f,g,h,m)\\ &= H_5(f,g,h,m) = H_6(f,g,h,m) = H_7(f,g,h,m) \approx_{\varepsilon_1} H_8(f,g,h,m) = H_9(f,g,h,m)\\ &= \mathrm{copy}\left(\mathrm{Sim}_{f,g,h}, m\right)\end{aligned}$$

This shows that our coding scheme is a 3-state non-malleable code with error $\mu + \mu' + \varepsilon_1 + \varepsilon_2$.

4.1 Setting the parameters

We will use λ as our security parameter. By Corollary 5, we will let k_1, k_2 be of length 2λ, i.e. $\gamma = \gamma' = 2\lambda$ and t_1, t_2 will have length λ, i.e. $\beta = \beta' = \lambda$ and both $(\mathrm{Tag}, \mathrm{Verify})$ and $(\mathrm{Tag}', \mathrm{Verify}')$ will have error $2^{-\lambda}$.

Since we will need to extract ℓ bits as a one-time pad to mask the message, by Lemma 2 and Theorem 4, we will set min-entropy k to be $(1+\alpha')\ell$ and let Ext be a $((1+\alpha')\ell, 2^{-\lambda})$-average min-entropy extractor that extract ℓ-bit randomness with seed length $O(\log n + \lambda)$. By our analysis in Appendix C, it is suffice to have $n - 6\lambda - \ell - O(\log n + \lambda) - 3 \geqslant (1+\alpha')\ell$. Hence, we will set $n = (2+\alpha)\ell$ for some $\alpha > \alpha'$.

Now the message length for our augmented 2-state non-malleable code will be $2\lambda + 2\lambda + \lambda + \lambda + O(\log n + \lambda) = O(\log n + \lambda)$. Now by Theorem 3, we will let c be the constant such that $p(n^c) = o(n)$ and set $\lambda = O(n^c)$. Hence, the length of (L, R) will be $o(n)$. Therefore, the total length of our coding scheme will be $\ell + (2+\alpha)\ell + o(n)$ and the rate is $\frac{1}{3+\alpha}$ with error $O(2^{-n^c})$. This completes the proof for Theorem 2.

References

[AAG+16] Divesh Aggarwal, Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. Optimal computational split-state non-malleable codes. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 393–417, 2016.

[ADKO15] Divesh Aggarwal, Yevgeniy Dodis, Tomasz Kazana, and Maciej Obremski. Non-malleable reductions and applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 459–468, 2015.

[ADL14] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 774–783, 2014.

[AGM+15] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 375–397, 2015.

[CG14a] Mahdi Cheraghchi and Venkatesan Guruswami. Capacity of non-malleable codes. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 155–168, 2014.

[CG14b] Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 440–464, 2014.

[CL17] Eshan Chattopadhyay and Xin Li. Non-malleable codes and extractors for small-depth circuits, and affine functions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1171–1184, 2017.

[CZ14] Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 306–315, 2014.

[DKO13] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 239–257, 2013.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 434–452, 2010.

[FMVW14] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 111–128, 2014.

[GUV07] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 96–108, 2007.

[KOS17] Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, and Sruthi Sekar. Four-state non-malleable codes with explicit constant rate. Cryptology ePrint Archive, Report 2017/930, 2017. http://eprint.iacr.org/2017/930.

[Li17] Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1144–1156, 2017.

[Vad12] S.P. Vadhan. *Pseudorandomness*. Foundations and trends in theoretical computer science. Now Publishers, 2012.

A Proof of Lemma 3

Assume there exists f, g, h, m such that $\mathrm{SD}\Big(H_1(f,g,h,m), H_2(f,g,h,m)\Big) > \varepsilon_2$. Notice that

$$\mathrm{SD}\Big(H_1(f,g,h,m), H_2(f,g,h,m)\Big) = \mathbb{E}_{w,s,k_1,k_2}\Big[\mathrm{SD}\Big(H_1(f,g,h,m)\Big|w,s,k_1,k_2\ ,\ H_2(f,g,h,m)\Big|w,s,k_1,k_2\Big)\Big]$$

Therefore, there exists a w^*, s^*, k_1^*, k_2^* such that

$$\mathrm{SD}\Big(H_1(f,g,h,m)\Big|w^*,s^*,k_1^*,k_2^*\ ,\ H_2(f,g,h,m)\Big|w^*,s^*,k_1^*,k_2^*\Big) > \varepsilon_2$$

And hence there exists a distinguisher $\mathcal{A}$ such that

$$\left|\Pr\Big[\mathcal{A}\Big(H_1(f,g,h,m)\Big|w^*,s^*,k_1^*,k_2^*\Big)=1\Big] - \Pr\Big[\mathcal{A}\Big(H_2(f,g,h,m)\Big|w^*,s^*,k_1^*,k_2^*\Big)=1\Big]\right| > \varepsilon_2$$

Let $c^* = m \oplus \mathrm{Ext}(w^*, s^*)$, $t_1^* = \mathrm{Tag}_{k_1^*}(c^*)$, $t_2^* = \mathrm{Tag}'_{k_2^*}(w^*)$. We shall use $\mathcal{A}$ to design $\mathcal{A}^*$ to break the augmented non-malleability of $(\mathrm{Enc}^*, \mathrm{Dec}^*)$ for message $u^* = (k_1^*, k_2^*, t_1^*, t_2^*, s^*)$ with tampering function $(g^*, h^*) = (g_{w^*}, h)$. Let $\mathcal{A}^*(L, \mathrm{Ans})$ do the following

$\mathcal{A}^*(L, \mathrm{Ans})$:

1. $\widetilde{c} = f(c^*)$, $\widetilde{w} = g_L(w^*)$.
2. If $\mathrm{Ans} = \perp$, output $\mathcal{A}(\perp)$. Else assume $\mathrm{Ans} = (\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$.
3. If $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\mathcal{A}(\perp)$.
4. Else output $\mathcal{A}\Big(\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})\Big)$.

It is easily seen that $\mathcal{A}\Big(H_1(f,g,h,m)\Big|w^*,s^*,k_1^*,k_2^*\Big)$ and $\mathcal{A}\Big(H_2(f,g,h,m)\Big|w^*,s^*,k_1^*,k_2^*\Big)$ are $\mathcal{A}^*\Big(\mathrm{TamperPlus}^{u^*}_{g^*,h^*}\Big)$ and $\mathcal{A}^*\Big(\mathrm{copy}\Big(\mathrm{Sim}^*_{g^*,h^*}, u^*\Big)\Big)$ respectively. And thus

$$\left|\Pr\Big[\mathcal{A}^*\Big(\mathrm{TamperPlus}^{u^*}_{g^*,h^*}\Big)=1\Big] - \Pr\Big[\mathcal{A}^*\Big(\mathrm{copy}\Big(\mathrm{Sim}^*_{g^*,h^*}, u^*\Big)\Big)=1\Big]\right| > \varepsilon_2$$

This breaks the augmented non-malleability and completes the proof.

B Proof of Lemma 4

To prove this lemma, consider the following intermediate hybrid.

$H'(f, g, h, m)$:

1. $w \sim U_n$, $s \sim U_d$, $k_1 \sim U_\gamma$, $k_2 \sim U_{\gamma'}$.
2. $r = \mathrm{Ext}(w, s)$, $c = m \oplus r$, $t_1 = \mathrm{Tag}_{k_1}(c)$, $t_2 = \mathrm{Tag}'_{k_2}(w)$.
3. $\widetilde{c} = f(c)$.
4. $(L, \mathrm{Ans}) = \mathrm{Sim}^*_{g_w,h}$.
5. $\widetilde{w} = g_L(w)$.
6. If Ans =

Case $\perp$: Output $\perp$.

Case same*: If $\Big((\widetilde{c} = c)$ and $\mathrm{Verify}'_{k_2}(\widetilde{w}, t_2)\Big) = 0$, output $\perp$.
Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, s)$.

Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\Big(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1})$ and $\mathrm{Verify}'_{\widetilde{k_2}}(\widetilde{w}, \widetilde{t_2})\Big) = 0$, output $\perp$.
Else output $\widetilde{c} \oplus \mathrm{Ext}(\widetilde{w}, \widetilde{s})$.

If $\mathrm{SD}(H_3(f,g,h,m), H_4(f,g,h,m)) > \mu + \mu'$, then one of the following will be true

$$\mathrm{SD}\left(H_3(f,g,h,m), H'(f,g,h,m)\right) > \mu \text{ or } \mathrm{SD}\left(H'(f,g,h,m), H_4(f,g,h,m)\right) > \mu'$$

We are going to use the property of $(\mathrm{Tag}, \mathrm{Verify})$ to prove the first one is impossible. (the other case could be proved similarly). Notice the only difference between $H_3(f,g,h,m)$ and $H'(f,g,h,m)$ is when $\widetilde{c} \neq c$ but $\mathrm{Tag}_{k_1}(\widetilde{c}) = \mathrm{Tag}_{k_1}(c)$. Therefore, we could write

$$\begin{aligned}
\mathrm{SD}\left(H_3(f,g,h,m), H'(f,g,h,m)\right) \leqslant & \Pr\left[\widetilde{c} \neq c \wedge \mathrm{Tag}_{k_1}(\widetilde{c}) = \mathrm{Tag}_{k_1}(c)\right] \\
= & \mathbb{E}_{\widetilde{c}\neq c}\left[\Pr_{k_1 \sim U_\gamma}\left[\mathrm{Tag}_{k_1}(\widetilde{c}) = \mathrm{Tag}_{k_1}(c)\right]\right] \\
\leqslant & \mathbb{E}_{\widetilde{c}\neq c}[\mu] = \mu
\end{aligned}$$

where the last inequality is because $(\mathrm{Tag}, \mathrm{Verify})$ is μ-secure message authentication code. This completes the proof of Lemma 4.

C Proof of Lemma 5

Consider the following function $F(\mathcal{L}(w), r)$, where $\mathcal{L}(w)$ is interpreted as $(\mathrm{Ans}, \mathrm{flag}_1, \mathrm{flag}_2, \mathrm{mask})$.

$F(\mathcal{L}(w), r)$:

1. $c = m \oplus r,\ \widetilde{c} = f(c)$
2. If Ans =
   Case $\perp$: Output $\perp$.
   Case same*: If $\left(\widetilde{c} = c \text{ and } \mathrm{flag}_1\right) = 1$, output same*.
   Else output $\perp$.
   Case $(\widetilde{k_1}, \widetilde{k_2}, \widetilde{t_1}, \widetilde{t_2}, \widetilde{s})$: If $\left(\mathrm{Verify}_{\widetilde{k_1}}(\widetilde{c}, \widetilde{t_1}) \text{ and } \mathrm{flag}_2\right) = 0$, output $\perp$.
   Else output $\widetilde{c} \oplus \mathrm{mask}$.

Notice that $H_7(f,g,h,m)$ and $H_8(f,g,h,m)$ are $F(\mathcal{L}(w), \mathrm{Ext}(w, U_d))$ and $F(\mathcal{L}(w), U_l)$ respectively. Since $\mathcal{L}(w)$ is of length $\ell + \beta + \beta' + \gamma + \gamma' + d + 3$, we have

$$\widetilde{H}_\infty\left(\mathcal{D}(w) \middle| \mathcal{L}(w)\right) \geqslant n - (\ell + \beta + \beta' + \gamma + \gamma' + d + 3)$$

Here we use $\mathcal{D}(w)$ to denote the distribution of w. By the property of our average min-entropy extractor, if $n - (\ell + \beta + \beta' + \gamma + \gamma' + d + 3) \geqslant k$, we have

$$\left(\mathcal{L}(w), \mathrm{Ext}\left(w, U_d\right)\right) \approx_{\varepsilon_1} \left(\mathcal{L}(w), U_\ell\right)$$

Therefore, $F(\mathcal{L}(w), \mathrm{Ext}(w, U_d)) \approx_{\varepsilon_1} F(\mathcal{L}(w), U_\ell)$ and this is equivalent to $H_7(f,g,h,m) \approx_{\varepsilon_1} H_8(f,g,h,m)$.

D Message authentication code

Lemma 6. *Suppose $\{h_k : \{0,1\}^\alpha \longrightarrow \{0,1\}^\beta\}$ is a μ-almost pairwise independent hash family. Then the following family of pair of functions is a μ-secure message authentication code.*

$$\left\{\begin{array}{c} \mathrm{Tag}_k(x) = h_k(x) \\ \mathrm{Verify}_k(x,y) = 1 \textit{ if and only if } y = h_k(x) \end{array}\right\}_{k \in K}$$

Proof. Obviously, for all m, k, $\text{Verify}(m, h_k(m)) = 1$. Also, for all $m \neq m'$ and t, t',

$$\Pr_{k\sim U_K}\Big[\,\text{Tag}_k(m') = t' \,\Big|\, \text{Tag}_k(m) = t\Big] = \frac{\Pr_{k\sim U_K}[\text{Tag}_k(m') = t' \wedge \text{Tag}_k(m) = t]}{\Pr_{k\sim U_K}[\text{Tag}_k(m) = t]} \leqslant \frac{\mu \cdot 2^{-\beta}}{2^{-\beta}} = \mu \quad \blacksquare$$

Lemma 7. *Suppose $\alpha = \ell \cdot \beta$ and write m as $(m_1, m_2, \ldots, m_\ell)$ where $m_i \in \{0,1\}^\beta$. Let $K = \{0,1\}^\beta \times \{0,1\}^\beta$ and write k as (k_1, k_2). Define $h_{k_1,k_2}(m) = k_1 + m_1 k_2 + m_2 k_2^2 + \cdots + m_l k_2^\ell$, which is seen as a polynomial in $\mathbb{GF}[2^\beta]$. Then $\{h_k\}$ is a $\frac{\alpha}{\beta \cdot 2^\beta}$-almost pairwise independent hash family.*

Proof. For all m, t,

$$\Pr_{k\sim U_{2\beta}}[h_k(m) = t] = \Pr_{k_2\sim U_\beta}\left[\Pr_{k_1\sim U_\beta}\left[k_1 + m_1 k_2 + m_2 k_2^2 + \cdots + m_l k_2^\ell = t\right]\right] = \Pr_{k_2\sim U_\beta}\left[2^{-\beta}\right] = 2^{-\beta}$$

For all $m \neq m'$ and t, t',

$$\begin{aligned}
&\Pr_{k\sim U_{2\beta}}[h_k(m) = t \wedge h_k(m') = t'] \\
&= \Pr_{k_1\sim U_\beta, k_2\sim U_\beta}\left[k_1 + m_1 k_2 + m_2 k_2^2 + \cdots + m_\ell k_2^\ell = t \wedge k_1 + m_1' k_2 + m_2' k_2^2 + \cdots + m_\ell' k_2^\ell = t'\right] \\
&= \Pr_{k_2\sim U_\beta}\left[\sum_{i=1}^{\ell}(m_i - m_i')k_2^i = t - t'\right] \cdot \Pr_{k_1\sim U_\beta}\left[k_1 + m_1 k_2 + m_2 k_2^2 + \cdots + m_\ell k_2^\ell = t\right] \\
&\leqslant \frac{\ell}{2^\beta} \cdot 2^{-\beta}
\end{aligned}$$

where the last inequality is because a degree ℓ polynomial in a field can have at most ℓ many zeros. Since $\ell = \alpha/\beta$, this completes the proof. $\blacksquare$

Corollary 5. *For all message length α and Tag length β, there exists a $\frac{\alpha}{2^\beta}$-secure message authentication code scheme with key length 2β.*