Continuously Non-Malleable Codes from Authenticated Encryptions in 2-Split-State Model

Anit Kumar Ghosal [1] and Dipanwita Roychowdhury [1]

Department of Computer Science and Engineering, IIT Kharagpur, India

Abstract. Tampering attack is the act of deliberately modifying the codeword to produce another codeword of a related message. The main application is to find out the original message from the codeword. Non-malleable codes are introduced to protect the message from such attack. Any tampering attack performed on the message encoded by non-malleable codes, guarantee that output is either *completely unrelated or original message*. It is useful mainly in the situation when privacy and integrity of the message is important rather than correctness. Unfortunately, standard version of non-malleable codes are used for *one-time* tampering attack. In literature, it is shown that non-malleable codes can be designed from authenticated encryption. But, such construction does not provide security when an adversary tampers the codeword more than once. Later, *continuously non-malleable codes* are constructed where an attacker can tamper the message for *polynomial number of times*. In this work, we propose a construction of continuously non-malleable code from authenticated encryption in 2-split-state model. Our construction provides security against polynomial number of tampering attacks and non-malleability property is preserved. The security of proposed continuously non-malleable code reduces to the security of underlying leakage resilient storage when tampering experiment triggers *self-destruct*.

Keywords: Authenticated encryption · Non-malleable codes · 2-Split-State model · Tamper-resilient cryptography.

1 Introduction

In the era of digital evaluation, various kind of attacks on the hardware devices are the most threatening aspects for the crypto designers. The adversary wants to exploit the weakness of physical implementation mechanism by injecting some faults during runtime of the cryptographic algorithm. Then, it can analyze the faulty and fault free output to get partial information about the internal state of the algorithm. Tampering attack is one of the attack where an adversary modifies the internal state of the device and manipulates some parameters of the underlying algorithm. Such attack can be performed by a fault injection or heating up the device. In case of software platform, a virus in the computer can carry out such tampering attack on the storage device by corrupting some regions of the memory. The ultimate goal of the adversary is to find out the keys so that they can destroy the cryptosystem completely. Boneh et al. [2] show such an devastating attack where an adversary can make a minor modification in the cryptographic device and the signing key can be recovered completely. A line of research work have focused how to secure any cryptographic implementation from such tampering attacks [4,8,9,11,13,27,28].

Non-malleable codes are introduced by Dziembowski et al. [5] as one of the applications of tamper-resilient cryptography. It ensures with *high probability* that if an adversary tampers any message encoded with non-malleable codes, output is either *completely unrelated* or *original message*, when tampering has no effect. Let k be the secret message, i.e., key of any cryptographic algorithm and f be a tampering function. The secret message is encoded as $\mathsf{Encode}(k)$. An adversary can apply the tampering function f on the encoded message as $f(\mathsf{Encode}(k))$. Then, it tries to decode the message in the following way $\mathsf{Decode}(f(\mathsf{Encode}(k)))$. The property of non-malleability ensures that $\mathsf{Decode}(f(\mathsf{Encode}(k))) = k$ with probability one, when tampering has no effect or in case of successful tampering attempt $\mathsf{Decode}(f(\mathsf{Encode}(k))) = k^{'}$, where k and $k^{'}$ both are computationally independent. Let $f_{increment}$ be a tampering function which tampers the encoded data as $f_{increment}(\mathsf{Encode}(k) + 1)$. After decoding output is $k + 1$, which is highly related to the original secret message. Hence, non-malleable codes can be constructed for some classes of tampering function only. In literature, the most widely used model is 2-split-state where the codeword is split into two different parts of the memory $\mathcal{M}_L$, $\mathcal{M}_R$ and two different tampering functions $f = (f_1(\mathcal{M}_L), f_2(\mathcal{M}_R))$ modify the codeword in an arbitrary and independent way [15,18]. Standard notion of non-malleability deals with *one-time* tampering attack only. It cannot handle the situation when an adversary tampers the codeword $polynomial\ number\ of\ times$. Later, Faust et al. [14] propose a stronger version of non-malleability called *continuous non-malleable codes* $(CNMC)$ where an adversary can perform the tampering attack for $polynomial\ number\ of\ times$ and still non-malleability is preserved.

There are various flavours of continuous non-malleability. The original message is denoted as m whereas $m^{'}$ is the decoded tampered message. Moreover, c represents the original codeword and $c^{'}$ represents the tampered codeword in a continuous tampering experiment. Usually, *standard* version of continuous non-malleability refers to the situation where the decoded tampered message $m^{'}$ and the original message m are completely independent but an attacker can create an encoding such that $c^{'}$ is not equal to c but $c^{'}$ decodes to m as discussed in [5]. In case of *strong* continuous non-malleability, when $c^{'}$ is not equal to c, it is guaranteed that both $m^{'}$ and m are independent. The more stronger flavour is *super-strong* continuous non-malleability, where $c^{'}$ is not equal to c implies that $c^{'}$ and c are independent [14,16,17]. We consider *stronger* version of continuous non-malleability. Again, based on the situation that how tampering functions are applied to the codeword, tampering experiment of continuous non-malleability has two versions as shown in [17]. In case of *non-persistent* tampering, the adversary applies the tampering functions on initial encoding of the codeword. In *persistent* version, tampering functions are applied to the previous version of tampered codeword rather than initial encoding. An adversary can tamper two different parts of the memory until decoding error is triggered. Continuously non-malleable code constructions are broadly categorized into two domains as information-theoretic [25] and computational [14, 22, 24]. Information-theoretic continuous non-malleability is impossible to achieve in 2-split-state model as mentioned in [14] due to the generic attack. Later, Aggarwal et al. [21] show that information-theoretic continuous non-malleability is possible when tampering is persistent in 2-split-state model. Ostrovsky et al. propose a more relaxed version of $CNMC$ from computational assumption in the plain model (i.e., without $common\ reference\ string$ (CRS) based setup) but it provides weaker security guarantee. To achieve stronger security, it is *necessary* to rely on CRS based setup assumptions as described in [26]. Hence, our construction relies on authenticated encryption, *robust non interactive zero knowledge* (NIZK) [3] proof and a commitment scheme with CRS based setup.

Scheme	Model	Security Assumption	Tampering Attempt	Security against Tampering and Leakage Attacks
[14]	Computational, CRS	NIZK, Collision resistant hash, Leakage resilient storage	Non-persistent with self-destruct	Polynomial number of tampering attacks and bounded leakage attacks
[21]	Information theoretic	NA	Persistent with self-destruct	Unbounded Adversary with polynomial number of tampering attacks and bounded leakage attacks
[22]	Computational, CRS	NIZK, Non-interactive commitment Leakage resilient public key encryption	Non-persistent with self-destruct	Polynomial number of tampering attacks and bounded leakage attacks
[24]	Computational	Only one-to-one one-way function	Non-persistent with self-destruct	Unbounded Adversary with polynomial number of tampering attacks and bounded leakage attacks
Our work	Computational, CRS	NIZK, Non-interactive commitment, Leakage resilient storage	Non-persistent with self-destruct	Polynomial number of tampering attacks and bounded leakage attacks

Table 1: Comparative results of various continuously non-malleable codes in the 2-split-state model.

Motivation of the construction. The initial construction of non-malleable codes are *keyless* in nature. Further research work shows that such codeword can be constructed from symmetric-key primitives, i.e., Authenticated Encryption (AE) [19], [28], [15, 20], related-key secure cipher [23] etc. Unfortunately, the codeword of [15, 20] and [19], [23], [28] are secure against *one-time* tampering attack only. It can not provide security when an adversary tampers the codeword more than once. Moreover, an adversary can tamper the right part of a codeword M_1 and produce $M_1^{'}$. Such attack can create two valid codewords (M_0, M_1) and $(M_0, M_1^{'})$ such that their decoding does not return $\perp$, i.e., $\perp \neq \mathsf{Decode}_k(\alpha, (M_0, M_1)) \neq \mathsf{Decode}_k(\alpha, (M_0, M_1^{'}\ )) \neq \perp$, where $M_1 \neq M_1^{'}$. The goal of the adversary is to produce two valid messages m, $m^{'}$. Further, the adversary may not activate the *self-destruct* feature and it can leak all the bits of M_1 with the assumption that the underlying tampering function is *non-persistent*. In general, for any continuously non-malleable codes, finding two valid codewords (M_0, M_1) and $(M_0, M_1^{'})$ such that $\mathsf{Decode}_k(\alpha, (M_0, M_1)) \neq \mathsf{Decode}_k(\alpha, (M_0, M_1^{'}))$ should be computationally hard to the adversary. This property is called *message uniqueness* as described in [14]. Our goal is to design non-malleable codes from authenticated encryption (i.e., *Encrypt then MAC*)

that is secure against *polynomial number of tampering attempts*. Table 1 shows various constructions of continuous non-malleable codes in 2-split-state model.

Our Contribution. In this work, we propose a continuous version of non-malleable code in 2-split-state model from authenticated encryption (i.e., *Encrypt then MAC*) along with *robust* NIZK proof and a commitment scheme, instantiated with one-to-one one-way function [1]. Initially, the message is encoded into leakage resilient storage (lrs) to protect from leakage attacks. Further, it is encoded with authenticated encryption along with robust NIZK and a commitment scheme. The authenticated encryption used in our construction should satisfy the following assumption:

a) If the decryption algorithm of an authenticated encryption with a key k succeeds, it should return $\perp$ when it is decrypted with a different key $k^{'}$, where $k \neq k^{'}$.

Organization. The paper is structured as follows. Section 2 describes some preliminaries whereas Section 3 provides a brief description about continuous non-malleability. Code construction, limitations and future enhancements are illustrated in Section 4. Finally, we conclude the paper in Section 5.

2 Preliminaries

Basic Notations. We describe a summary of notations in Table 2.

Notation	Terminology
m	Original message
M_0, M_1	Left and right half of a codeword
$\mathcal{M}_L$, $\mathcal{M}_R$	Left and right half of the memory
$\mathcal{O}^{T}_{cnmc}(.,.)$	Tampering oracle
f_1, f_2	Tampering functions
$\mathcal{K}$	Key set
$k \xleftarrow{\$} \mathcal{K}$	A particular key is selected
n	Security parameter
$\mathcal{O}^{l}(s)$	Leakage oracle with s as input
α	Common reference string
S_0, S_1	Two simulators
$\epsilon(n)$	A negligible function
$\mathbb{E} \underset{s}{\approx} \mathbb{F}$	Statistical indistinguishability
$\tau()$	Leakage function
λ	Public label
π	Proof of a statement
pk, sk	Public and private key pair
r	Randomness

Table 2: Summary of notations

2.1 Leakage Resilient Storage

The purpose of leakage resilient storage (lrs) scheme is to encode the message in such a way that an adversary with access to some additional leakage information is unable to guess the original message from the encoded one. The security of leakage resilient storage is preserved until some bounded information is available to the adversary [14]. It consists of a pair of algorithms (Enc^{lrs}, Dec^{lrs}) described as follows:

- Enc^{lrs} algorithm inputs a message m and produces the output p_0, p_1.
- Dec^{lrs} algorithm inputs p_0, p_1 and generates m as output.

The leakage experiment is defined below:

$$leak^{\beta}_{A,m} = \left\{ \begin{array}{c} (p_0, p_1) \leftarrow Enc^{lrs}(m); \mathcal{L} \leftarrow A^{\mathcal{O}^l(p_0,.),\mathcal{O}^l(p_1,.)} \\ output : (p_\beta, \mathcal{L}_A), \beta \in \{0, 1\} \end{array} \right\}$$

Initially, a counter ctr is initialized to 0. When strings are passed into the oracle $\mathcal{O}^l(p_0, .)$, $\mathcal{O}^l(p_1, .)$, the leakage function $\tau(p_0)$, $\tau(p_1)$ are used to calculate the value and finally, it is added to ctr, until $ctr \leq l$ from each part. Oracle terminates if $ctr > l$, and further query would return $\perp$. The storage scheme is said to be *strong lrs* if an adversary should not be able to distinguish between two arbitrarily chosen messages m and $m^{'}$ except with negligible probability, i.e., $leak^{\beta}_{A,m} \underset{c}{\approx} leak^{\beta}_{A,m^{'}}$

2.2 Robust Non-interactive Zero Knowledge

Let L be the language with relation $\mathcal{R}$, denoted as $L^{\mathcal{R}} = \{\ m\colon \exists\ w$ such that $\mathcal{R}(m, w) = 1\}$ and $m \in \mathcal{M}$. *Robust non-interactive zero knowledge* (NIZK) proof system for the language $L^{\mathcal{R}}$ consists of a set of algorithms ($CRSGen, Prove, Vrfy, S = (S_0, S_1), Xtr$), defined as follows. $CRSGen$ algorithm inputs a security parameter 1^n and generates $\alpha \in \{0, 1\}^n$ as $common\ reference\ string$ (CRS). $Prove$ algorithm inputs α, a label λ, $(m, w) \in \mathcal{R}$ and produces the proof $\pi = Prove^{\lambda}(\alpha, m, w)$ as output. The deterministic verification algorithm $Vrfy$ outputs $true$ in case of successful statement verification, i.e., $Vrfy^{\lambda}(\alpha, m, Prove^{\lambda}(\alpha, m, w)) = 1$. S algorithm consists of two simulators, i.e., S_0 and S_1. The simulator S_0 generates a CRS and the *trapdoor key* whereas S_1 performs simulated game with an adversary A. Xtr outputs the hidden value of the relation $\mathcal{R}(m, w)$. It satisfies all the below properties as mentioned in [3]:

- **Completeness.** For every $m \in L^{\mathcal{R}}$ and all w such that $\mathcal{R}(m, w) = 1$, for all $\alpha \leftarrow CRSGen(1^n)$, we require that $Pr[Vrfy(\alpha, m, Prove(\alpha, w, m)) = 1]$ should be satisfied.

- **Multi-Theorem zero knowledge.** The honestly computed proof does not reveal anything except the validity of the statement. Formally, we can define it as follows. For every PPT adversary A, the real experiment and the simulated experiment are indistinguishable, i.e., $Real(n) \approx Simulated(n)$. $Real(n)$ and $Simulated(n)$ are described below:

$$Real(n) = \left\{ \begin{array}{c} \alpha \leftarrow CRSGen(1^n); \mathcal{L} \leftarrow A^{Prove(\alpha,.,.)}(\alpha) \\ output : \mathcal{L} \end{array} \right\}$$

$$Simulated(n) = \left\{ \begin{array}{c} (\alpha, pk) \leftarrow S_0(1^n); \mathcal{L} \leftarrow A^{S_1(\alpha,.,pk)}(\alpha) \\ output : \mathcal{L} \end{array} \right\}$$

– **Extractability.** Extractability property describes that for every PPT adversary A, there exists a PPT algorithm Xtr, a negligible function ϵ and a security parameter n such that $Pr[G^{Xtr} = 1] \leq \epsilon(n)$, where G^{Xtr} is described below:

$$G^{Xtr} = \left\{ \begin{array}{c} (\alpha, pk, sk) \leftarrow S_0(1^n) \\ (m, \pi) \leftarrow A^{S_1(\alpha,.,pk)}(\alpha); w \leftarrow Xtr(\alpha, (m, \pi), sk) \\ (m, \pi) \notin \mathcal{Q} \wedge \mathcal{R}(m, w) \neq 1 \wedge Vrfy(\alpha, m, \pi) = 1 \end{array} \right\},$$

The query set $\mathcal{Q}$ stores (m, π) pairs that an adversary A asks to S_1.

Our assumption is that if any statement is modified, proof of verification should be unsuccessful as illustrated in [10,14]. Also, the proof system supports public label λ and this label is appended to the statement m during calculation of all the above properties, i.e., $Prove^{\lambda}(.,.,.)$, $Vrfy^{\lambda}(.,.,.)$, $Xtr^{\lambda}(.,.,.)$, $S_1^{\lambda}(.,.,.)$ etc.

2.3 Authenticated Encryption

An *authenticated encryption* (AE) scheme[1] consists of following algorithms ($k = \{k_{enc}||k_{mac}\}, Encypt, Decrypt$) such that

- $Encrypt$: Encryption algorithm takes a key $k_{enc} \in \mathcal{K}$, message $m \in M$ and produces a ciphertext $c \in \mathcal{C}$. We write it as $c \leftarrow Encrypt(k_{enc}, m)$. Then, it produces $\mathsf{tag} \leftarrow tag(k_{mac}, c)$. Finally, it outputs $(c||\mathsf{tag})$.
- $Decrypt$: Decryption algorithm checks first the tag. If it matches, the plaintext is retrieved as $m \leftarrow Decrypt(k_{enc}, c)$ or $\perp$ if decryption fails.

Moreover, the *correctness* property $Decrypt(k, Encrypt(k, m)) = m$, for all $k \in \mathcal{K}$, $m \in M$ and $c \in \mathcal{C}$ should be satisfied.

2.4 Non-interactive Commitment Scheme

A Non-interactive Commitment Scheme consists of two algorithms, i.e., $CRSGen$ and $Commit$. $CRSGen$ takes input security parameter 1^n and generates $\alpha \in \{0,1\}^n$ as a commitment key. $Commit$ algorithm takes the commitment key α, message $m \in \{0,1\}^n$, randomness $r \in \{0,1\}^n$ and generates γ as output. It satisfies the following properties:

– **Computationally binding.** The commitment scheme is said to satisfy statistically binding property if there does not exist messages $m^0, m^1 \in \{0,1\}^n$ such that $m^0 \neq m^1$ and pair (m^0, r_0), (m^1, r_1) produces $Commit(\alpha, m^0, r_0) = Commit(\alpha, m^1, r_1)$.

[1] We refer only *Encrypt then MAC* scheme .

– **Statistically hiding.** A Non-interactive Commitment Scheme is said to satisfy computationally hiding property if for messages $m^0, m^1 \in \{0,1\}^n$, the equation $Commit(\alpha, m^0) \underset{s}{\approx} Commit(\alpha, m^1)$ should be satisfied.

3 Continuously Non-malleable Codes

Leakage Oracle. The purpose of stateful leakage oracle $\mathcal{O}^l(.)$ is to calculate the total leakage through arbitrary leakage function $\tau()$. The complete leakage experiment is defined in Algorithm 1. Initially, the value of counter ctr is initialized to 0. When a new string is passed through the oracle, leakage value is calculated and the result is added with the ctr, until $ctr \leq l$. Otherwise, it returns $\perp$.

Algorithm 1 Leakage Oracle $\mathcal{O}^l(s, .)$

```
Set ctr = 0
Apply leakage function τ() on s and calculate leakage
Update ctr = ctr + |τ(s)|
if  ctr ≤ l then
    return ctr
else
    return ⊥
end if
```

Tampering Oracle. The tampering Oracle $\mathcal{O}^T_{cnmc}(., .)$ in 2-split-state model is a stateful oracle that takes input two codewords M_0, M_1 and tampering function $f = (f_0, f_1) \in \mathcal{F}$ with initial $state = alive$. The tampering oracle experiment is defined in Algorithm 2.

Coding Scheme. Let $CNMC = (\mathsf{CRSGen}, \mathsf{Encode}_k, \mathsf{Decode}_k)$ be a split-state coding scheme in the CRS model.

– *$CRSGen$ algorithm takes security parameter 1^n as input and generates output $\alpha \in \{0,1\}^n$ as CRS.*
– *Encode_k algorithm takes key $k \in \mathcal{K}$, CRS α, message $m \in \mathcal{M}$ and produces the codeword (M_0, M_1).*
– *Decode_k algorithm takes the codeword (M_0, M_1), key $k \in \mathcal{K}$, CRS α and generates message m or special symbol $\perp$.*

Algorithm 2 Tampering Oracle $\mathcal{O}^{T}_{cnmc}((M_0, M_1), (f_0, f_1))$

if $state = self\text{-}destruct$ **then**
  return $\perp$
end if
$(M_0^{'}, M_1^{'}) = (f_0(M_0), f_1(M_1))$
if $(M_0, M_1) = (M_0^{'}, M_1^{'})$ **then**
  return *same**
end if
if $\mathsf{Decode}_k(\alpha, (M_0^{'}, M_1^{'})) = \perp$ **then**
  set $state = self\text{-}destruct$ and return $\perp$
else
  return $\mathsf{Decode}_k(\alpha, (M_0^{'}, M_1^{'}))$
end if

Continuous Non-malleability. The coding scheme $CNMC$ is said to be l leakage resilient, q continuously non-malleable code in split-state model if for all messages $m, m^{'} \in \{0,1\}^n$ and for all *probabilistic polynomial-time* adversaries A, $\mathbf{Tamper}^{A,m}_{cnmc}$ and $\mathbf{Tamper}^{A,m^{'}}_{cnmc}$ are computationally indistinguishable, i.e.,

$\mathbf{Adv}^{Strong}_{Tamper^{A}_{cnmc}}(A) = [Pr[A(\mathbf{Tamper}^{A,m}_{cnmc}) = 1]$ - $Pr[A\ (\mathbf{Tamper}^{A,m^{'}}_{cnmc}) = 1]] \leq \epsilon(n)$, where $m,\ m^{'} \in \{0,1\}^n$ and

$$\mathbf{Tamper}^{A,m}_{cnmc} = \left\{ \begin{array}{c} \alpha \leftarrow CRSGen(1^n); i = 0; (M_0, M_1) \leftarrow \mathsf{Encode}_k(\alpha, m) \\ while\ i \leq q \\ \mathcal{L}^{i}_{A} \leftarrow A^{\mathcal{O}^{l}(M_0^{i}), \mathcal{O}^{l}(M_1^{i}), \mathcal{O}^{T}_{cnmc}(M_0^{i}, M_1^{i})} \\ i = i + 1 \\ end\ while \\ output : \mathcal{L}^{i}_{A}. \end{array} \right\},$$

The complete view of an adversary is stored into $\mathcal{L}^{i}_{A}$ with two parameters μ and δ, where i denotes the number of tampering queries ($i \leq q$). The array μ captures the value of all leakage queries ($\mu \leq 2l$) whereas δ array stores the value of tampering queries ($\delta \leq q$) from $\mathcal{O}^{T}_{cnmc}()$. In case, the value $i = 1$ denotes that the codeword can handle *one-time* tampering attack only. Further, the value $i = 0$ denotes that the codeword is capable of handling *leakage* attacks [6].

Message Uniqueness. Let $CNMC = (\mathsf{CRSGen}, \mathsf{Encode}_k, \mathsf{Decode}_k)$ be a 2-split-state (l, q) continuously non-malleable code. The codeword is said to satisfy message uniqueness property if there does not exist a valid pair (M_0, M_1), $(M_0, M_1^{'})$ such that $\perp \neq \mathsf{Decode}_k(\alpha,\ (M_0, M_1)) \neq \mathsf{Decode}_k(\alpha, (M_0, M_1^{'})) \neq \perp$, where $M_1 \neq M_1^{'}$ and the experiment produces two valid messages $m,\ m^{'}$. A continuously non-malleable code should not violate uniqueness property as mentioned in [14].

4 Code Construction

To construct continuously non-malleable codes, we use authenticated encryption along with robust NIZK and a commitment scheme. The complete codeword construction is described as follows:

1. $\mathsf{CRSGen}(1^n)$. The CRS generation algorithm inputs 1^n as a security parameter and produces the common reference string α as output.

2. $\mathsf{Encode}_k(\alpha, m)$. To encode the message $m \in \mathcal{M}$, a uniformly random key $k \in \mathcal{K}$ ($k = \{k_{enc}||k_{mac}\}$) is selected with CRS α. The algorithm first computes $(p_0, p_1) \leftarrow Enc^{lrs}(m||r)$ with some randomness $r \leftarrow \{0,1\}^n$. Further, p_0, p_1 (i.e., $c_0 \leftarrow Encrypt(k_{enc}, p_0)$, $c_1 \leftarrow Encrypt(k_{enc}, p_1)$) are encrypted by encryption algorithm of the authenticated encryption. The tag_{p_0} and tag_{p_1} are generated as $\mathsf{tag}_{p_0} \leftarrow tag(k_{mac}, c_0)$, $\mathsf{tag}_{p_1} \leftarrow tag(k_{mac}, c_1)$. Commitment scheme is used to check uniqueness of the key $k = \{k_{enc}, k_{mac}\}$, i.e., $com = commit(\alpha, k; r)$. The next step is to calculate the proof of statement, i.e., $\pi_0 = Prove^{c_1}(\alpha, k, (com, c_0))$, $\pi_1 = Prove^{c_0}(\alpha, k, c_1)$. Finally, the codeword $(M_0, M_1) = ((k, p_0, (\mathsf{tag}_{p_1}, com, c_1), \pi_0, \pi_1), (k, p_1, (\mathsf{tag}_{p_0}, c_0), \pi_0, \pi_1))$ is stored into $\mathcal{M}_L$ and $\mathcal{M}_R$ respectively.

3. $\mathsf{Decode}_k(\alpha, (M_0, M_1))$. To decode the codeword, π_0, π_1 are parsed and the following steps are performed:
   (a) *Left & Right verification.* If the verification algorithm $Vrfy^{c_1}(\alpha, (com, c_0), \pi_0)$ and $Vrfy^{c_0}(\alpha, c_1, \pi_1)$ return $false$ in (M_0, M_1), output $\perp$. Otherwise, go to the next step.
   (b) *Uniqueness check.* If $com = commit(\alpha, k; r)$, go to the next step. Otherwise, return $\perp$.
   (c) *Cross check & Decode.* If tag_{p_0} and tag_{p_1} are not matched, return $\perp$. Otherwise, compare $p_0 \neq Decrypt(k_{enc}, c_0)$, $p_1 \neq Decrypt(k_{enc}, c_1)$. Whenever the following proofs π_0, π_1 are not matched, return $\perp$. Finally, the equality of p_0, p_1 are checked in M_0 and M_1, if it is satisfied, call decode $Dec^{lrs}(p_0, p_1)$.

Lemma 1. *$CNMC = (\mathsf{CRSGen}, \mathsf{Encode}_k, \mathsf{Decode}_k)$ satisfies message uniqueness property if implemented with a commitment scheme.*

Proof. The binding property of the commitment scheme implies message uniqueness. Let us consider an adversary A has the capability to generate a pair (M_0, M_1), $(M_0, M_1^{'})$ such that both are valid and $M_1 \neq M_1^{'}$. Therefore, the adversary is able to generate the following equation: $\perp \neq \mathsf{Decode}_k(\alpha, (M_0, M_1)) \neq \mathsf{Decode}_k(\alpha, (M_0, M_1^{'})) \neq \perp$. It is only possible if an adversary generates a valid key pair $(k, k^{'})$ in such a way that satisfies $commit(\alpha, k, r) = com = commit(\alpha, k^{'}, r)$, where $k \neq k^{'}$. Unfortunately, such equation violates the binding property of the commitment scheme. Hence, $commit(\alpha, k, r) = com \neq commit(\alpha, k^{'}, r)$. Therefore, we can conclude that integrity of the key is violated and decoding should return $\perp$ as per property (a) of the underlying authenticated encryption.

Correctness and Security. To prove the security of the proposed construction, we need to use reduction. Informally, we can say that when the tampering experiment triggers *self-destruct*, the security of continuously non-malleable code reduces to the security of underlying leakage resilient storage. Alternatively, we can say that if the underlying leakage resilient storage is secure, the proposed continuously non-malleable code is secure. Our future work is to analyse the proof in detail.

Application to Tamper-Resilient Cryptography. In cryptography, the main assumption is that an adversary only has black-box view of the cryptosystem. Further, it can only observe the input-output behavior to the system. Unfortunately, such model does not provide security when an adversary has physical access to the cryptosystem. It can attack the hardware or software module where the actual implementation of the algorithm is present. An adversary can have some arbitrary leakage function to get partial information about the cryptosystem (i.e., using timing, radiation, heating, power consumption etc. of the device). The other way, it can physically tamper the device by heating up to introduce some random errors in the memory or cut the wires. The goal of an adversary is to learn the secret key. Our proposed codeword can be used to protect sensitive information against both leakage and tampering attacks against *polynomial number of* times until *self-destruct* occurs. The codeword takes any *secret key* $< \mathsf{K} >$ and converts into $< \mathsf{K}^{encoded} >$, i.e., key encoded with continuously non-malleable codes secured against leakage and tampering attacks.

Limitations and Future Directions. Our codeword provides security against *non-persistent* tampering attacks only until *self-destruct* state is triggered. The proposed construction is capable of handling *polynomial number of tampering attacks* in computational domain. The future work is to design continuously non-malleable codes for *persistent* tampering attacks with *self-destruct* feature from authenticated encryptions (i.e., *Encrypt then MAC* and others generic AE scheme) in *common reference string* model or plain model. Also it is not known whether continuously non-malleable codes can be designed from authenticated encryption for *persistent* tampering attacks in information-theoretic domain for computationally unbounded adversary.

5 Conclusion

In this work, we show a construction of continuously non-malleable code from authenticated encryption (i.e., *Encrypt then MAC*) in *common reference string* model. The codeword is capable of handing *non-persistent* tampering attacks with *self-destruct* feature only. To the best of our knowledge, this work is the first one that considers authenticated encryption to design continuously non-malleable codes and handles polynomial number of tampering attacks.

References

1. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. J. ACM 38(3), 691–729 (1991)
2. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. J. Cryptology 14(2), 101–119 (2001)
3. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001)
4. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
5. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: Yao, A.C.-C. (ed.) ICS 2010, Beijing, China, January 5-7, pp. 434–452. Tsinghua University Press (2010)
6. Davì, F., Dziembowski, S., Venturi, D.: Leakage-resilient storage. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 121–137. Springer, Heidelberg (2010)
7. Dziembowski, S., Faust, S.: Leakage-resilient cryptography from the inner-product extractor. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 702–721. Springer, Heidelberg (2011)
8. Bellare, M., Cash, D., Miller, R.: Cryptography secure against related-key attacks and tampering. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 486–503. Springer, Heidelberg (2011)
9. Kalai, Y.T., Kanukurthi, B., Sahai, A.: Cryptography with Tamperable and Leaky Memory. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 373–390. Springer, Heidelberg (2011)
10. 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)
11. Bellare, M., Paterson, K.G., Thomson, S.: RKA security beyond the linear barrier: IBE, encryption and signatures. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 331–348. Springer, Heidelberg (2012)
12. 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)
13. Damgård, I., Faust, S., Mukherjee, P., Venturi, D.: Bounded tamper resilience: How to go beyond the algebraic barrier. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 140–160. Springer, Heidelberg (2013)
14. 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)
15. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: STOC, pp. 774–783 (2014)
16. Faust, S., Mukherjee, P., Venturi, D., Wichs, D.: Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: EUROCRYPT. pp. 111–128 (2014)
17. 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)

18. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, pp. 459–468. ACM (2015)
19. Kiayias, A., Liu, F.H., Tselekounis, Y.: Practical non-malleable codes from l-more extractable hash functions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1317–1328. ACM Press, October 2016
20. 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)
21. Aggarwal, D., Kazana, T., Obremski, M.: Inception makes non-malleable codes stronger. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 319–343. Springer, Cham (2017)
22. Faonio, A., Nielsen, J.B., Simkin, M., Venturi, D.: Continuously non-malleable codes with split-state refresh. In: Preneel, B., Vercauteren, F. (eds.) ACNS 2018. LNCS, vol. 10892, pp. 1–19. Springer, Cham (2018)
23. Fehr, S., Karpman, P., Mennink, B.: Short Non-Malleable Codes from Related-Key Secure Block Ciphers. IACR Transactions on Symmetric Cryptology, 336-352, (2018)
24. 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, Part III. LNCS, vol. 10993, pp. 608–639. Springer, Cham (2018)
25. Aggarwal, D., Döttling, N., Nielsen, J.B., Obremski, M., Purwanto, E.: Continuous non-malleable codes in the 8-split-state model. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 531–561. Springer, Cham (2019)
26. D. Dachman-Soled, M. Kulkarni, Upper and lower bounds for continuous non-malleable codes, in PKC (2019), pp. 519–548
27. B. Chen, Y. Chen, K. Hostáková, P. Mukherjee, Continuous space-bounded non-malleable codes from stronger proofs-of-space, in CRYPTO (2019), pp. 467–495
28. Ghosal, A.K., Ghosh, S., Roychowdhury, D.: Practical Non-malleable Codes from Symmetric-Key Primitives in 2-Split-State Model. In: Ge, C., Guo, F. (eds) Provable and Practical Security, (2022).