Ghosal and Roychowdhury *Cybersecurity (2023)6:25*
https://doi.org/10.1186/s42400-023-00152-1

Cybersecurity

RESEARCH

Open Access

Continuously non-malleable codes from block ciphers in split-state model

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

Abstract

Non-malleable code is an encoding scheme that is useful in situations where traditional error correction or detection is impossible to achieve. It ensures with high probability that decoded message is either completely unrelated or the original one, when tampering has no effect. Usually, standard version of non-malleable codes provide security against one time tampering attack. Block ciphers are successfully employed in the construction of non-malleable codes. Such construction fails to provide security when an adversary tampers the codeword more than once. Continuously non-malleable codes further allow an attacker to tamper the message for polynomial number of times. In this work, we propose continuous version of non-malleable codes from block ciphers in split-state model. Our construction provides security against polynomial number of tampering attacks and it preserves non-malleability. When the tampering experiment triggers self-destruct, the security of continuously non-malleable code reduces to security of the underlying leakage resilient storage.

Keywords Block cipher, Non-malleable code, Split-state model, Tamper-resilient cryptography

Introduction

Physical attacks on the implementation of various cryptographic schemes are the most threatening aspects for the crypto designer. In theoretical cryptography, the algorithm under consideration is modeled as a blackbox with which an adversary can interact via the input–output interface of the system. Such blackbox security notions do not incorporate an adversary that can change the secret message into some related value through tampering attack, and analyse the outcomes. The adversary can perform tampering attack by heating up the devices, fault injections (Sergei and Ross 2002) etc. In software module, viruses or malwares can also carry out the attack on storage device by corrupting some regions of the memory. Boneh et al. (2001) show that a single bit flip of the signing key is enough to extract the secret information of RSA signature completely. This is one of the most devastating attack where an adversary makes minor modification in the cryptographic device and the sensitive information can be recovered. A line of research have focused on how to secure any cryptographic implementation from such tampering attacks (Bellare and Kohno 2003; Bellare et al. 2011; Kalai et al. 2011; Bellare et al. 2012; Damgård et al. 2013; Chen et al. 2019; Ghosal et al. 2022).

Non-malleable codes, introduced by Dziembowski et al. (2010, 2018), are used as one of the applications of tamper resilient cryptography. It is required when correction of the message is not the main concern but privacy and integrity are more important. Further, the guarantee is that if an adversary tampers any message encoded by non-malleable codes, output is either completely *unrelated* or the *original* one. Let k be the secret message (e.g., key of any cryptographic algorithm) and f be the tampering function. An adversary encodes the secret message k as $Enc(k)$. It uses the tampering function f on the encoded message $Enc(k)$ and performs decoding, i.e., $Dec(f(Enc(k)))$. Non-malleability property guarantees that

*Correspondence:
Anit Kumar Ghosal
anit.ghosal@gmail.com
[1] Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur, West Bengal, India

$Dec(f(Enc(k))) = k$, for every k with probability 1, when tampering has no effect or $Dec(f(Enc(k))) = k^{'}$, in case of tampering, where k and $k^{'}$ are computationally independent. Generally, non-malleability cannot be achieved for arbitrary classes of tampering functions. Let $f_{increment}$ be the tampering function. An adversary uses this function on the encoded message as $f_{increment}(Enc(k) + 1)$ and tries to decode it as $Dec(f_{increment}(Enc(k) + 1))$. After decoding, the adversary gets output as $k + 1$. It is highly related to the original secret message, i.e., k. Hence, non-malleable codes can be constructed for some classes of tampering functions only. In literature, most widely used model is split-state, where the codeword is divided into two different parts M_0, M_1, and it is stored into memory M_L, M_R respectively (Liu and Lysyanskaya 2012; Dziembowski et al. 2013; Jafargholi and Wichs 2015; Aggarwal et al. 2015; Kiayias et al. 2016; Aggarwal et al. 2016; Fehr et al. 2018). Two different tampering functions $f = (f_0(M_0), f_1(M_1))$ modifies the codeword in an arbitrary and independent way. One important functionality is that both tampering functions cannot run the decoding procedure because two shares are needed in order to decode a codeword whereas each of the functions $f_0(M_0)$, $f_1(M_1)$ can access only one share. Standard notion of non-malleability protects message for *one time* tampering attack only. Such codeword is called *one-shot* non-malleable code. It cannot handle the situation when an adversary tampers the codeword more than once. A stronger version of non-malleability, called *continuously non-malleable codes* (*CNMC*) is proposed in Faust et al. (2014a), where the following attack $f = (f_i(M_0), f_{i+1}(M_1))$ $(i \in q \wedge q \in poly(n))$ is performed polynomial number of times for each $f_i \in \mathcal{F}$, and still non-malleability is preserved.

Continuous non-malleability has various flavours. Let m be the original message and $m^{'}$ be the decoded tampered message. Moreover, c denotes the codeword and $c^{'}$ denotes the tampered codeword in a continuous tampering experiment. *Standard* version of continuous non-malleability or *default* version refers to the situation where decoded tampered message $m^{'}$ and original message m are completely independent but it is possible for an attacker to create an encoding such that $c^{'}$ is not equal to c but $c^{'}$ decodes to m as discussed in Dziembowski et al. (2010). In case of *strong* continuous non-malleability, when $c^{'}$ is not equal to c, it is guaranteed that both $m^{'}$ and m completely are independent. Another stronger flavour is *super-strong* continuous non-malleability, where $c^{'}$ is not equal to c implies that $c^{'}$ and c are independent (Faust et al. 2014a, b; Jafargholi and Wichs 2015). Our construction considers *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 Jafargholi and Wichs (2015). When tampering functions are applied always to initial encoding of the codeword, it is called *non-persistent* tampering. Here, an auxiliary memory is required beyond n bits of *active-memory* to store the codeword. An attacker can make a copy of the original codeword to the auxiliary memory. Further, the attacker can tamper the original version of the codeword from the auxiliary memory and place it to the *active-memory*. In *persistent* version, tampering functions are applied to the previous version of tampered codeword rather than initial encoding. So, the extra memory requirement is not present here. An adversary can tamper two different parts of the memory until decoding error is triggered. Additional feature of continuous non-malleability is to handle leakage attacks while tampering attacks are being performed. The adversary can gain leakage values as a partial information. Earlier constructions of continuously non-malleable codes are built on top of some leakage resilient primitives which can handle some bounded amount of leakages (Faust et al. 2014a; Aggarwal et al. 2014, 2015) independently from two different parts of the memory. Continuously non-malleable code constructions are broadly categorized into two domains as information-theoretic (Aggarwal et al. 2019) and computational (Faust et al. 2014a; Faonio et al. 2018; Ostrovsky et al. 2018). In Faust et al. (2014a), it is shown that information-theoretic continuous non-malleability is not possible to achieve in split-state model due to the generic attack. Later, Aggarwal et al. (2017) show that in case of persistent tampering in split-state model, information-theoretic continuous non-malleability can be achieved. Further research work shows a more relaxed version of *CNMC* from computational assumption in the plain model (i.e., without *common reference string* based setup) but it provides weaker security guarantee (Ostrovsky et al. 2018). In Dachman-Soled and Kulkarni (2019), authors describe that it is *necessary* to rely on setup assumptions, i.e., *common reference string* (CRS) to achieve stronger security. Hence, the proposed construction relies on block cipher and *robust non interactive zero knowledge* (NIZK) (De Santis et al. 2001) proof in CRS based trusted setup environment. In Table 1, we describe various constructions of continuously non-malleable codes in split-state model as available in the literature.

Limitations of the Existing Work and Our Motivation. Usually, non-malleable codes are *keyless* encoding scheme in nature. The first construction of a continuously non-malleable code is proposed in Faust et al. (2014a). Their work is based on collision resistant hash function with *robust non interactive zero knowledge* (NIZK) proof. Later, Fehr et al. (2018) show that *one-shot*

Table 1 Comparison of various continuously non-malleable codes in the split-state model

Reference	Model	Assumption	Types of tampering	Adversarial capability
Faust et al. (2014a, 2020)	Comp., CRS	Collision resistant hash, NIZK Leakage resilient storage	Non-persistent with self-destruct	Polynomial number of tampering attacks & bounded leakage attacks
Aggarwal et al. (2017)	I.T.	–	Persistent with self-destruct	Unbounded adversary with polynomial number of tampering attacks & bounded leakage attacks
Faonio et al. (2018)	Comp., CRS	NIZK, Non-interactive commitment Leakage resilient public key encryption	Non-persistent with self-destruct	Polynomial number of tampering attacks & bounded leakage attacks
Ostrovsky et al. (2018)	Comp.	One-to-one One-way function	Non-persistent with self-destruct	Unbounded adversary with polynomial number of tampering attacks & bounded leakage attacks
Our work	Comp., CRS	NIZK, Strong PRP, Leakage resilient storage	Non-persistent with self-destruct	Polynomial number of tampering attacks & bounded leakage attacks

In the table IT stands for *information-theoretic*, Comp. stands for *computational*

non-malleable codes can be constructed from *related-key* secure block ciphers. Such construction does not satisfy security against continuous attacks. An attacker can create two valid codewords (M_0, M_1) and $(M_0, M_1^{'})$ such that their decoding does not return $\perp$, i.e., $\perp \neq Dec_k(\alpha, (M_0, M_1)) \neq Dec_k(\alpha, (M_0, M_1^{'})) \neq \perp$, where $M_1 \neq M_1^{'}$. It produces two valid messages $m, m^{'}$. Moreover, assuming the tampering function is *non-persistent*, an adversary can leak all the bits of M_1 without activating the *self-destruct* feature. In general, for any continuously non-malleable codes, it should be hard to find two valid codewords (M_0, M_1) and $(M_0, M_1^{'})$ such that $Dec_k(\alpha, (M_0, M_1)) \neq Dec_k(\alpha, (M_0, M_1^{'}))$. This property is called *mess age uniqueness* as described in Faust et al. (2014a). Our goal is to design non-malleable codes from any kind of block cipher such as AES (Joan and Vincent 2002), SHACAL (Handschuh and Naccache 2002), Midori (Banik et al. 2015) etc. that is secure against *polynomial number of tampering attempts*. The block ciphers used in our construction should satisfy the following properties:

a) The output produced by the underlying block cipher should be *strong pseudorandom permutation* (*sprp*).
b) If decryption of a ciphertext c with a key k succeeds, it should return $\perp$ if it is decrypted with a different key $k^{'}$ (Subsection 2.5).

Our Contribution. In this work, we propose the construction of continuously non-malleable codes in split-state model from any block cipher in computational domain with trusted setup, i.e., CRS. We remove the restriction of *related-key* secure block cipher as used in Fehr et al. (2018). The codeword is capable of handling *non-persistent* tampering attempts until *self-destruct* occurs. Initially, the message is encoded into leakage resilient storage (*lrs*). Further, it is encoded with block cipher along with *robust non interactive zero knowledge* (NIZK) proof. The key k of the block cipher is divided into two shares k_0, k_1. Left part of a codeword M_0 stores k_0 whereas M_1 stores k_1. During decoding, it is reconstructed as $k \leftarrow k_0 \oplus k_1$.

Organization. The paper is organized as follows. Section 2 describes some preliminaries whereas Sect. 3 provides a brief description about continuous non-malleability. Code construction and basic proof ideas are illustrated in Sect. 4. Thereafter, proof of security is given in Sect. 5. Finally, we conclude the paper in Sect. 6.

Preliminaries

Notations and basic results

Let m be the original message. M_0 and M_1 are the left and right half of a codeword in split-state model, stored in memory M_L and M_R respectively. $\mathcal{O}^{T}_{cnmc}(.,.)$ represents the tampering oracle. Two tampering functions are f_0 and f_1 working in M_0 and M_1 respectively. Moreover, f_0^i (or f_1^i) denotes the tampering function used by an adversary at i^{th} round. $\mathcal{K}$ is the usable key set after removing the *weak* and *semi-weak* keys of the block cipher. If $\mathcal{K}$ is the key set, $|\mathcal{K}|$ represents the number of key elements in $\mathcal{K}$. When k is uniformly chosen at random from $\mathcal{K}$, we write $k \xleftarrow{\$} \mathcal{K}$. n is the security parameter. $\mathcal{O}^{l}(s)$ denotes the leakage oracle that takes string s as input and performs leakage function $\tau_b()$ on s, and it returns at most l bits. $r \in \{0,1\}^n$ denotes the randomness. α represents an untamperable *common reference string* (CRS). A function $\epsilon(n)$ is called *negligible* in n if it vanishes faster than the inverse of any polynomial in n. $P(x; r)$ is a randomized algorithm which takes $x \in \{0,1\}^n$, randomness $r \in \{0,1\}^n$ as input and produces the output $y \in \{0,1\}^n$. An algorithm P is called

probabilistic polynomial-time (PPT) if P is allowed to make random choices, and the computation of $P(x; r)$ terminates in a polynomial number of steps ($|x|$) at most for $x \in \{0,1\}^n$, $r \in \{0,1\}^n$. Let $\mathbb{E} = \{E_k\}_{k\in N}$, $\mathbb{F} = \{F_k\}_{k\in N}$ be two ensembles and $\mathbb{E} \underset{c}{\approx} \mathbb{F}$ represents the *computational indistinguishability* that for every PPT distinguisher D, $|\Pr[D(E_k) = 1] - \Pr[D(F_k) = 1]| \leq \epsilon(n)$. In similar way, $\mathbb{E} \underset{s}{\approx} \mathbb{F}$ denotes the *statistical indistinguishability* for computationally unbounded scenario. $\mathcal{H}_\infty(X)$ and $\tilde{\mathcal{H}}_\infty(X|Y)$ denote the min-entropy and conditional average min-entropy of the random variable X. $\delta_0[i]$, $\delta_1[i]$ are two arrays used to store the result of tampering queries whereas $\mu_0[i]$, $\mu_1[i]$ are used to store leakage queries result at each invocation ($i \in q \wedge q \in poly(n)$) in Algorithm 3 and Algorithm 4. In Table 2, we describe a summary of notations. We now define some definitions and lemmas related to the code construction.

Definition 2.1.1 *(Split-State Model)* Let M be a codeword which consist of two shares $M = (M_0, M_1)$, and they are stored into two different parts of the memory $\mathsf{M}_L, \mathsf{M}_R$ respectively. Each tampering attempt $f = (f_0, f_1)$ is described by two arbitrary chosen functions that can be applied to the codeword $f = (f_0(M_0), f_1(M_1))$ in an independent way. The model which satisfies the above property is said to be `split-state` model.

Definition 2.1.2 *(Non-persistent Tampering)* Let $f = (f_0, f_1)$ be the tampering function and M be a codeword which is split into two shares $M = (M_0, M_1)$. The tampering experiment is said to be `non-persistent` if the tampering functions are applied to initial encoding of the codeword always. Moreover, such model considers the scenario when an adversary has access to an n-bit auxiliary memory beyond the active memory, and it can copy the original codeword to the auxiliary memory. Later, the subsequent attack can be performed on the auxiliary memory and the tampered codeword can be placed to the original memory.

Lemma 2.1.1 *A random variable X has min-entropy over the set $\mathcal{X}$, denoted as $\mathcal{H}_\infty(X) = -log\ max_{x\in\mathcal{X}} \Pr[X = x]$. It represents the probability of guessing X by an unbounded adversary.*

Lemma 2.1.2 *A random variable X has conditional average min-entropy given some information Y over the set $\mathcal{X}$, $\mathcal{Y}$, denoted as $\tilde{\mathcal{H}}_\infty(X|Y) = -log\mathbb{E}_{y\in\mathcal{Y}}\ max_{x\in\mathcal{X}} \Pr[X = x|Y = y]$. It represents the probability of guessing X when some related information of X is available to the adversary through side channel leakage.*

Table 2 Summary of notations

Notation	Terminology
m	Original message
M_0, M_1	Left and right half of a codeword
$\mathsf{M}_L, \mathsf{M}_R$	Left and right half of the memory
$\mathcal{O}^T_{cnmc}(.,.)$	Tampering oracle
$\delta_0[], \delta_1[]$	Stores tampering queries data
f_0, f_1	Tampering functions
$\mathcal{K}$	Key set
$poly(n)$	Polynomial function on input n
$k \xleftarrow{\$} \mathcal{K}$	A key is selected
n	Security parameter
$\mathcal{O}^l(s,.)$	Leakage oracle with s and τ as input
$\mu_0[], \mu_1[]$	Stores leakage queries data
α	Common reference string
$\epsilon(n)$	A negligible function
$P(x; r)$	A randomized algorithm
$\mathbb{E} \underset{s}{\approx} \mathbb{F}$	Statistical indistinguishability
$\mathcal{H}_\infty(X)$	Min-entropy
$\tilde{\mathcal{H}}_\infty(X\|Y)$	Conditional average min-entropy
$\tau()$	Arbitrary leakage function
λ	Label
π	Proof of a statement
S_0, S_1	Two simulators
pk, sk	Public and private key pair
r	Randomness
$\mathcal{R}(m, w)$	Relation
$\mathfrak{D}_k()$	Block cipher decryption algorithm
$\mathfrak{E}_k()$	Block cipher encryption algorithm
$Enc_k()$	CNMC encoding algorithm
$Dec_k()$	CNMC decoding algorithm
$\mathcal{O}_{prp}()$	Pseudorandom permutation oracle
$\mathcal{O}_R()$	Random permutation oracle
$\mathfrak{Enc}^{lrs}, \mathfrak{Dec}^{lrs}$	*lrs* encoding and decoding algorithm

Lemma 2.1.3 *For a random variable X and another random variable Y, $\tilde{\mathcal{H}}_\infty(X|Y) \geq \mathcal{H}_\infty(X) - l$, where Y takes 2^l possible values ($l \in \{0,1\}^n$).*

Lemma 2.1.4 *For a random variable X and other two correlated random variables Y_1, Y_2, we get $\tilde{\mathcal{H}}_\infty(X|Y_1, Y_2) \geq \tilde{\mathcal{H}}_\infty(X|Y_1) - l$, where Y_2 takes 2^l possible values ($l \in \{0,1\}^n$).*

Lemma 2.1.5 *Let τ be the leakage function (possibly randomized) used by an adversary on variable X. Then,*

$\tilde{\mathcal{H}}_\infty(X|\tau(X)) \geq \mathcal{H}_\infty(X) - l$, *where* $\tau(X)$ *generates* l *bits of leakage through the side channel* ($l \in \{0,1\}^n$).

Lemma 2.1.6 *Let* X, Y *be the correlated random variables and* τ *be the leakage function used by an adversary* A. *Then,* $\tilde{\mathcal{H}}_\infty(X|\tau(Y)) \geq \tilde{\mathcal{H}}_\infty(X|Y)$.

Leakage resilient storage

Leakage Resilient Storage (*lrs*) scheme encodes message in such a way that secures the underlying message against leakage attacks. It consists of a pair of algorithms $(\mathfrak{Enc}^{lrs}, \mathfrak{Dec}^{lrs})$ with the following properties:

- $\mathfrak{Enc}^{lrs}$ algorithm takes input a message m, randomness r and produces the output p_0, p_1.
- $\mathfrak{Dec}^{lrs}$ algorithm takes p_0, p_1 as input and generates m as output.

Original idea of $(\mathfrak{Enc}^{lrs}, \mathfrak{Dec}^{lrs})$ algorithm is used in literature (Davì et al. 2010; Dziembowski and Faust 2011) for computationally unbounded adversary. In our construction, it is used for computationally bounded adversary (Faust et al. 2014a). Leakage experiment is defined below:

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

Initially, a counter *ctr* is set to 0. When strings are passed into $\mathcal{O}^l(p_0,.)$, $\mathcal{O}^l(p_1,.)$, along with leakage function $\tau(.)$, leakage values are calculated through $\tau(p_0)$, $\tau(p_1)$, and it is added to *ctr*, until $ctr \leq l$ from each part. Oracle terminates if $ctr > l$, and further query would return $\perp$.

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.,

$$\mathbf{Adv}^{strong}_{\mathfrak{leak}^{\beta}_{A}}(A) = [Pr[A(\mathfrak{leak}^{\beta}_{A,m}) = 1] - \Pr[A(\mathfrak{leak}^{\beta}_{A,m'}) = 1]] \leq \epsilon(n),$$

where $m, m^{'} \in \{0,1\}^n$ and $\epsilon(n)$ denotes a negligible function.

Robust non-interactive zero knowledge

Let $\mathcal{R}$ be a relation for the language $\mathfrak{L}$, denoted as $\mathfrak{L}^{\mathcal{R}} = \{ m : \exists\, w$ such that $\mathcal{R}(m,w) = 1\}$ and $m \in \mathcal{M}$. *Robust non-interactive zero knowledge* (NIZK) proof system for $\mathfrak{L}^{\mathcal{R}}$ consists of a set of algorithms $(CRSGen, Prove, Vrfy, S = (S_0, S_1), Xtr)$, defined as follows. *CRSGen* takes input a security parameter 1^n and generates $\alpha \in \{0,1\}^n$ as a *common reference string* (CRS). *Prove* takes α, a label λ, $(m,w) \in \mathcal{R}$ as input and produces proof $\pi = Prove^{\lambda}(\alpha, m, w)$ as output. The deterministic *Vrfy* algorithm outputs *true* when verification of statement is successful, i.e., $Vrfy^{\lambda}(\alpha, m, Prove^{\lambda}(\alpha, m, w)) = 1$. The algorithm S consists of two simulators, i.e., S_0 and S_1. 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 De Santis et al. (2001):

- **Completeness.** For every $m \in \mathfrak{L}^{\mathcal{R}}$ and all w such that $\mathcal{R}(m,w) = 1$, for all $\alpha \leftarrow CRSGen(1^n)$, we require that the following probability should be satisfied $\Pr[Vrfy(\alpha, m, Prove(\alpha, w, m)) = 1]$.
- **Multi-theorem zero knowledge.** It says that honestly computed proof does not reveal anything beyond the validity of the statement. Mathematically, it is represented as follows. For every *probabilistic polynomial-time* adversary A, real experiment, i.e., $Real(n)$ and simulated experiment, i.e., $Simulated(n)$ are completely 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.** For all 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 game G^{Xtr} is described below.

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

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

In Liu and Lysyanskaya (2012); Faust et al. (2014a), authors show that if the proof statement is modified, the verification algorithm should not proceed further. We use the same approach in our construction. Moreover, the proof algorithm supports public label λ and such label is incorporated with the statement of the message m to calculate the above algorithms, i.e., $Prove^{\lambda}(.,.,.)$, $Vrfy^{\lambda}(.,.,.)$, $Xtr^{\lambda}(.,.,.)$, $S_1^{\lambda}(.,.,.)$ etc.

Pseudorandom permutation

Let block cipher $\mathfrak{E}$: $\{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ be a mapping from message space $\mathcal{M}$ to ciphertext space $\mathcal{C}$ through a fixed k. An adversary A plays fixed pseudorandom security (*prp*) game with *prp* oracle $\mathcal{O}_{prp}()$ and

random permutation oracle $\mathcal{O}_R()$. The *pseudorandom permutation* security advantage is defined as follows: $\mathbf{Adv}^{prp}_{\mathfrak{E}}(A) = \Pr[A^{\mathcal{O}_{prp}()} = 1]$ - $Pr[A^{\mathcal{O}_R()} = 1]$.

$\mathbf{Adv}^{prp}_{\mathfrak{E}}(q,t) = \max_A\{\mathbf{Adv}^{prp}_{\mathfrak{E}}(A)\}$, where q is the maximum number of queries with time at most t.

An adversary A guesses the value of b, where $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, A proceeds with $\mathcal{O}_{prp}()$ and if $b = 1$, A proceeds with $\mathcal{O}_R()$. $\mathcal{O}_{prp}()$ returns encryption $\mathfrak{E}_k(m)$ and $\mathcal{O}_R()$ returns random keyed permutations $E_k(m)$, $k \xleftarrow{\$} \mathcal{K}$.

$\mathfrak{K}$ is the total key set whereas $\mathcal{K}$ is the usable key set after removing *weak* and *semi-weak* keys, i.e., $\mathcal{K} = \mathfrak{K}$ - $\{k^{weak} \cup k^{semi-weak}\}$. In a cipher, *weak* and *semi-weak* keys are such keys by which an encryption scheme can be broken more efficiently than usual keys.

Block cipher

A block cipher $\mathfrak{E}$: $\{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ is a keyed permutation which takes message $m \in \mathcal{M}$, key $k \in \mathcal{K}$ and outputs $c \in \mathcal{C}$, called encryption. Its inverse algorithm which takes $c \in \mathcal{C}$, $k \in \mathcal{K}$ and generates $m \in \mathcal{M}$, called decryption $\mathfrak{D}$. Classical security models for block ciphers are *pseudoran dompermutation* (*prp*) and *strong pseudorandom permutation* (*sprp*). In *prp* security model, an adversary has only access to encryption oracle whereas in *strong pseudorandom permutation* model the adversary has access to both encryption and decryption oracle.

Moreover, the block cipher used in our construction has the following property: If key is modified then decryption algorithm should return $\perp$. To achieve such property in our non-malleable code construction, we check the key in Algorithm 3 and Algorithm 4. The original key k of a cipher is stored into two parts of codeword M_0 and M_1. Whenever original key k and tampered key $k^{'}$ are completely different, i.e., $k^{'} \neq k$, decryption algorithm $\mathfrak{D}_k()$ should not be called and we return $\perp$ from the decoding algorithm of non-malleable code. Since the decryption algorithm of a block cipher with a different key $k^{'}$ returns some other message rather than original one, we need to restrict it in this way.

Continuously non-malleable codes

Leakage Oracle. Leakage Oracle $\mathcal{O}^l(.)$ is a stateful oracle that calculates total leakage through some arbitrary leakage function $\tau()$. Algorithm 1 shows the leakage experiment. Initially, a counter *ctr* is set to 0. When strings are passed into it, leakage values are calculated and its length is added with the *ctr*, until $ctr \leq l$. Otherwise, it returns $\perp$.

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

Coding Scheme. Let $CNMC = (CRSGen,\ Enc_k, Dec_k)$ 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.*
- Enc_k *algorithm takes key* $k \in \mathcal{K}$, *CRS* α, *message* $m \in \mathcal{M}$ *and produces the codeword* (M_0, M_1).
- Dec_k *algorithm takes the codeword* (M_0, M_1), *key* $k \in \mathcal{K}$, *CRS* α *and generates message m or special symbol* $\perp$.

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

$$\mathbf{Tamper}^{A,m}_{cnmc} = \left\{ \begin{array}{c} \alpha \leftarrow CRSGen(1^n);\ i = 0;\ (M_0, M_1) \leftarrow Enc_k(\alpha, m) \\ while \quad i \leq q \\ \mathcal{L}^i_A \leftarrow A^{\mathcal{O}^l(M^i_0), \mathcal{O}^l(M^i_1), \mathcal{O}^T_{cnmc}(M^i_0, M^i_1)} \\ i = i + 1 \\ end \quad while \\ output : \mathcal{L}^i_A. \end{array} \right\},$$

$\mathcal{L}_A^i$ contains the view of an adversary with two parameters μ and δ, for i number of tampering queries ($i \leq q \wedge q \in poly(n)$). μ stores the result of leakage queries ($\mu \leq 2\,l$) and δ stores the result of tampering queries ($\delta \leq q$) from $\mathcal{O}_{cnmc}^{T}()$. When $i = 1$, our code behaves as *one-shot* non-malleable code and without any tampering query, i.e., $i = 0$, it acts as *leakage resilient code* (Davì et al. 2010).

Message Uniqueness. Let $CNMC = (CRSGen, Enc_k, Dec_k)$ be a split-state (l, q) continuously non-malleable code. It 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 $\perp \neq Dec_k(\alpha, (M_0, M_1)) \neq Dec_k(\alpha, (M_0, M_1^{'})) \neq \perp$, where $M_1 \neq M_1^{'}$ and it produces two valid messages $m, m^{'}$. A continuously non-malleable code should not violate uniqueness property as mentioned in Faust et al. (2014a).

Code construction

We propose the construction of continuously non-malleable codes from block cipher along with *robust non interactive zero knowledge* (NIZK) proof. Then, we analyse the uniqueness property of the codeword and proof of security. Let $CNMC = (CRSGen, Enc_k, Dec_k)$ be split-state (l, q) continuously non-malleable code in the CRS model based on *leakage resilient storage* ($\mathfrak{Enc}^{lrs}$, $\mathfrak{Dec}^{lrs}$), on a block cipher $\mathfrak{E}$: $\{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ with some properties incorporated and on a *robust non-interactive zero knowledge* (NIZK) proof system (*CRSGen*, *Prove*, *Vrfy*) with label support for language $\mathfrak{L}^{\mathfrak{E}_{k_0}} = \{\, c_{key} : \exists\, k$ such that $c_{key} = \mathfrak{E}_{k_0}(k)\}$, where $k \in \mathcal{K}$, $k \leftarrow k_0 \oplus k_1$. The construction of our codeword is illustrated below:

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

```
Initialize ctr = 0
Use the arbitrary leakage function τ() on s and find the leakage
Adjust the counter ctr = ctr + |τ(s)|
if ctr ≤ l then
    return ctr
else
    return ⊥
end if
```

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

```
if state = self-destruct then
    return ⊥
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 Dec_k(α, (M_0', M_1')) = ⊥ then
    set state = self-destruct and return ⊥
else
    return Dec(α, (M_0', M_1'))
end if
```

I. $CRSGen(1^n)$. The algorithm takes 1^n as a security parameter and generates the common reference string α.
II. $Enc_k(\alpha, m)$. Encoding algorithm takes key $k \in \mathcal{K}$, CRS α and message $m \in \mathcal{M}$ as input. Initially, the message m with some randomness $r \leftarrow \{0,1\}^n$ is fed into *leakage resilient storage*, i.e., $(p_0, p_1) \leftarrow \mathfrak{Enc}^{lrs}(m||r)$. Next, it encrypts p_0, p_1 as $c_0 \leftarrow \mathfrak{E}_k(p_0)$, $c_1 \leftarrow \mathfrak{E}_k(p_1)$, where $\mathfrak{E}_k()$ is an encryption algorithm of a block cipher. The key k is divided into two shares k_0, k_1 and it is reconstructed as $k \leftarrow k_0 \oplus k_1$. Further, the master key k is encrypted as $c_{key} = \mathfrak{E}_{k_0}(k)$. Thereafter, proof of statements are calculated in the following way, i.e., $\pi_0 = Prove^{c_1}(\alpha, k_0, (c_{key}, c_0))$, $\pi_1 = Prove^{c_0}(\alpha, k_1, (c_{key}, c_1))$. Finally, it outputs the codeword $(M_0, M_1) = (((k_0, c_0), p_0, (c_{key}, c_1), \pi_0, \pi_1), ((k_1, c_1), p_1, (c_{key}, c_0), \pi_0, \pi_1))$. The codeword (M_0, M_1) is stored into the memory (M_L, M_R) respectively.
III. $Dec_k(\alpha, (M_0, M_1))$. Decoding algorithm starts by parsing π_0 and π_1. Then, it constructs the key $k \leftarrow k_0 \oplus k_1$ and performs the below steps:
IV. *Left & Right verification.* If the verification of statement in the codeword (M_0, M_1) are not successful, i.e., either $Vrfy^{c_1}(\alpha, (c_{key}, c_0))$ or $Vrfy^{c_0}(\alpha, (c_{key}, c_1), \pi_1)$ returns 0, it outputs $\perp$. Otherwise, go to the next step.
V. *Uniqueness check.* If $k = \mathfrak{D}_{k_0}(c_{key})$, go to the next step. Otherwise, it returns $\perp$.
VI. *Cross check & Decode.* If $p_0 \neq \mathfrak{D}_k(c_0)$, $p_1 \neq \mathfrak{D}_k(c_1)$ and proofs π_0, π_1 both are different, it returns $\perp$. Otherwise, check p_0, p_1, if both are equal in M_0 and M_1, call decode $\mathfrak{Dec}^{lrs}(p_0, p_1)$.

Lemma 1 *$CNMC = (CRSGen, Enc_k, Dec_k)$ satisfies message uniqueness property if implemented with the block cipher.*

Proof

Message uniqueness is based on the property (b) (Subsection 2.5) of the underlying block cipher, i.e., ciphetext generated by the cipher with a key k returns $\perp$ if it is decrypted with a different key k'. Hence, integrity of the key has to be maintained. Suppose, an adversary A generates a pair (M_0, M_1), (M_0, M_1') such that both are valid and $M_1 \neq M_1'$. It means $\perp \neq Dec_k(\alpha, (M_0, M_1)) \neq Dec_k(\alpha, (M_0, M_1')) \neq \perp$. The equation is only possible if an adversary is able to produce a valid key pair (k_0, k_1), (k_0, k_1') such that for (k_0, k_1), $\mathfrak{D}_{k_0}(c_{key}) = k_0 \oplus k_1$ (for M_0, M_1) which is equal to $k_0 \oplus k_1' = \mathfrak{D}_{k_0}(c_{key})$ for (k_0, k_1') (for M_0, M_1'), where $k_1 \neq k_1'$. Unfortunately, it violates the deterministic property of decryption algorithm as the decrypted key and newly formed key are same. So, $\mathfrak{D}_{k_0}(c_{key}) = (k_0 \oplus k_1)$ (for M_0, M_1) $\neq (k_0 \oplus k_1') = \mathfrak{D}_{k_0}(c_{key})$ (for M_0, M_1'). Therefore, the key is modified and decoding should return $\perp$.

Security proof idea of *CNMC*

Our hunch is to develop the *continuous version of non-malleable* codes from block ciphers with some additional properties incorporated on the cipher. As mentioned by Gennaro et al. (2004), certain strong cryptographic assumptions are necessary when an adversary tampers a portion of the memory. To prove that codeword is continuously non-malleable, a simulator for the $\mathbf{Tamper}^{A,m}_{cnmc}$ experiment is developed. In $\mathbf{Tamper}^{A,m}_{cnmc}$ experiment, an adversary A performs all leakage and tampering oracle queries in real environment on the codeword (M_0, M_1), stored in memory M_L and M_R respectively, whereas simulated experiment $\mathbf{SimTamper}^{A,0^n}_{cnmc}$ simulates the adversaries view of the tampering experiment in an ideal scenario. We need to show that both experiments are indistinguishable except with negligible probability, i.e., $|\Pr[\mathbf{Tamper}^{A,m}_{cnmc} = 1] - \Pr[\mathbf{SimTamper}^{A,0^n}_{cnmc} = 1]| \leq \epsilon(n)$. Simulated tampering experiment takes $r \leftarrow \{0,1\}^n$ and proceeds with encryption of message $0^n||r$. But the original tampering experiment proceeds with encryption of message $m||r$. Initially, $m||r$ is encoded using leakage resilient storage which splits the message into two halves, and it keeps the message secure as long as l bits are leaked at most from each parts of the memory. Given the codeword $M = (M_0, M_1)$, oracle continues until simulated output from left (Algorithm 3) and right (Algorithm 4) sides of (T_0, T_1) are equal. The experiment stops when decoding error is triggered, i.e., outputs are not equal. From that point further query would return $\perp$, and *self-destruct* state is invoked. Since non-persistent tampering is considered, a separate memory $\mathfrak{M}$ of polynomial length is used to store tampered versions of the codeword at each round along with leakage and tampering data.

Algorithm 3 Tampering Experiment for Left Split $T_0(M_0, f_0^i, r, i)$

Parse $M_0 = ((k_0, c_0), p_0, (c_{key}, c_1), \pi_0, \pi_1)$
Apply leakage function τ_0^i on M_0, i.e., $\mu_0[i] = \tau_0^i(M_0)$
Apply f_0^i on M_0. $M_0^{'} = f_0^i(M_0) = ((k_0^{'}, c_0^{'}), p_0^{'}, (c_{key}^{'}, c_1^{'}), \pi_0^{'}, \pi_1^{'})$
if $M_0 = M_0^{'}$ **then**
  set $\delta_0[i] = same^*$
end if
if $Vrfy^{c_1^{'}}(\alpha, (c_{key}^{'}, c_0^{'}), \pi_1^{'}) = 0$ **then**
  set $\delta_0[i] = \perp$ and return $\perp$
end if
if $\pi_0 = \pi_0^{'}$ **then**
  set $\delta_0[i] = \perp$ and return $\perp$
end if
Run Xtr to retrieve $k_1^{'}$. $k_1^{'} \leftarrow Xtr^{c_0^{'}}(\alpha, ((c_{key}^{'}, c_1^{'}), \pi_1^{'}), sk)$
if $k_1^{'} = \perp$ **then**
  set $\delta_0[i] = \perp$ and return $\perp$
end if
Form key $k^{'} \leftarrow k_0^{'} \oplus k_1^{'}$
if $k^{'} \neq \mathfrak{D}_{k_0}(c_{key})$ **then**
  set $\delta_0[i] = \perp$ and return $\perp$
end if
$p_1^{'} \leftarrow \mathfrak{D}_{k^{'}}(c_1^{'})$
Call decode $\mathfrak{Dec}^{lrs}(p_0^{'}, p_1^{'})$ to retrieve $m^{'}$
Set $\delta_0[i] = M_0^{'}$ and return $m^{'}$

The main difficulty of our experiment is to find *self-destruct* index, i.e., from the point experiment would return $\perp$ for further query. Let $\tau(M)$ be the leakage function on the codeword M. $\mathcal{H}_\infty(M|\tau(M))$ denotes the conditional average entropy of the codeword M when some information is available through side-channel, i.e., the best chance of guessing message m from the codeword M with some side-channel information by an adversary A. Leakage functions are applied in the interleaved way by an adversary A on (M_0, M_1) as $\tau_0^0(M_0)$, $\tau_1^0(M_1)$, $\tau_0^1(M_0)$, $\tau_1^1(M_1)$,... $\tau_0^{i-1}(M_0)$, $\tau_1^{i-1}(M_1)$. The $\mathbf{SimTamper}_{cnmc}^{A,0^n}$ experiment proceeds until output produced by two algorithms T_0 and T_1 are equal. From information-theoretic observation, it can be viewed as $\tilde{\mathcal{H}}_\infty(M_0|\tau_0^i(M_0)) = \tilde{\mathcal{H}}_\infty(M_1|\tau_1^i(M_1))$, i.e., best chance of guessing message m from the codeword $M = (M_0, M_1)$ is same when some information is available through side-channel leakage to the adversary A. At each query invocation, simulated experiment proceeds by checking tampered output from both halves of the memory. If it matches, leak the entire part so that total amount of leakage is upper bounded by $\mathcal{O}(n)$, where n represents the security parameter. The experiment triggers *self-destruct* when outputs are unequal. Simulated tampering experiment consists of $S = (S_0, S_1)$ and it works in the following way. The simulator S_0 generates an untamperable CRS and the key (α, pk, sk). Further, the key is passed to S_1 which takes $r \leftarrow \{0,1\}^n$, encoding of message $0^n||r$, and invokes (T_0, T_1) to simulate the tampering experiment until outputs are equal. The simulator S_1 makes simulated proof of statement $\pi_0 = S_1^{c_1}(\alpha, (c_{key}, c_0), pk)$ and $\pi_1 = S_1^{c_0}(\alpha, (c_{key}, c_1), pk)$. Then, it calls the algorithm T_0 and T_1 in an interleaved manner. Algorithm T_0 simulates left part of a codeword (simulated) M_0 and algorithm T_1 simulates right part of a codeword M_1. Both the algorithm proceeds by parsing M_0 and M_1. It calculates leakage through (τ_0^i, τ_1^i) and stores the value into $\mu_b[i]$. Then, it applies tampering function f_0^i on M_0 and f_1^i on M_1, and it compares tampered codeword $M^{'}$ with the original codeword M. If both are same, $\delta_b[i]$ is set to *same**. Next, it verifies the proof of the statement and if it is successful, T_b proceeds further. Otherwise, $\delta_b[i]$ is set to $\perp$. Further, the original and tampered proof of statement are compared, and the corresponding values are stored into $\delta_b[i]$. The extractor Xtr algorithm retrieves the key $k_0^{'}$ in algorithm T_1, $k_1^{'}$ in algorithm T_0 and the key $k^{'}$ is formed, i.e., $k^{'} \leftarrow k_0^{'} \oplus k_1^{'}$. Next, uniqueness condition of the key $k^{'}$ is checked with k, and if they are same, decoding is performed to retrieve the message $m^{'}$.

Now, we discuss why the known attacks are not possible to perform in the proposed construction. Firstly,

if an adversary tampers c_0 and changes it some related value $c_0^{'}$ in M_0, NIZK proof π_0 should be changed to $\pi_0^{'}$. Hence, both values π_0, $\pi_0^{'}$ should be different and by the property of robust NIZK, experiment should return $\perp$. Also the adversary has to make same changes in M_1, this should be hard without knowing a witness by robustness of the proof. Apart from that if an adversary tampers the key k, and make it to $k^{'}$, NIZK proof should be different and decryption with $k^{'}$ should return $\perp$ as per cipher property (b). Hence, the codeword is secure against continuous tampering attacks. In the next section, we discuss the security of the construction in detail.

Algorithm 4 Tampering Experiment for Right Split $T_1(M_1, f_1^i, r, i)$

Parse $M_1 = ((k_1, c_1), p_1, (c_{key}, c_0), \pi_0, \pi_1)$
Apply leakage function τ_1^i on M_1, i.e., $\mu_1[i] = \tau_1^i(M_1)$
Apply f_1^i on M_1. $M_1^{'} = f_1^i(M_1) = ((k_1^{'}, c_1^{'}), p_1^{'}, (c_{key}^{'}, c_0^{'}), \pi_0^{'}, \pi_1^{'})$
if $M_1 = M_1^{'}$ **then**
    set $\delta_0[i] = same^*$
end if
if $Vrfy^{c_0^{'}}(\alpha, (c_{key}^{'}, c_1^{'}), \pi_1^{'}) = 0$ **then**
    set $\delta_1[i] = \perp$ and return $\perp$
end if
if $\pi_1 = \pi_1^{'}$ **then**
    set $\delta_1[i] = \perp$ and return $\perp$
end if
Run Xtr to retrieve $k_0^{'}$. $k_0^{'} \leftarrow Xtr^{c_1^{'}}(\alpha, ((c_{key}^{'}, c_0^{'}), \pi_0^{'}), sk)$
if $k_0^{'} = \perp$ **then**
    set $\delta_1[i] = \perp$ and return $\perp$
end if
Form key $k^{'} \leftarrow k_0^{'} \oplus k_1^{'}$
$k_0 \leftarrow Xtr^{c_1^{'}}(\alpha, ((c_{key}, c_0), \pi_0), sk)$
if $k^{'} \neq \mathfrak{D}_{k_0}(c_{key})$ **then**
    set $\delta_1[i] = \perp$ and return $\perp$
end if
$p_0^{'} \leftarrow \mathfrak{D}_{k^{'}}(c_0^{'})$
Call decode $\mathfrak{Dec}^{lrs}(p_0^{'}, p_1^{'})$ to retrieve $m^{'}$
Set $\delta_1[i] = M_1^{'}$ and return $m^{'}$

Proof of security

Theorem 1 Let $\mathfrak{E}$: $\{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ be the block cipher with message space $\mathcal{M}$, key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$, $(\mathfrak{Enc}^{lrs}, \mathfrak{Dec}^{lrs})$ be $l^{'}$ leakage resilient storage, (*CRSGen, Prove, Vrfy*) is a robust NIZK proof for language $\mathfrak{L}^{\mathcal{R}}$ chosen from message space $\mathcal{M}$. Then CNMC $= (CRSGen, Enc_k, Dec_k)$ is $((l + \gamma + \eta), q)$ continuously non-malleable and l leakage resilient code under non-persistent tampering when instantiated with all the above primitives, where $q = poly(n)$, $\gamma = log(\mathcal{M})$, $\eta = log(\mathcal{K})$, $l^{'} \geq (2\,l + n)$ and n denotes the security parameter.

Proof

The proof of our theorem is quite involved. We develop a simulator that simulates the tampering experiment in an ideal scenario. It is shown that an adversary cannot distinguish between the real and simulated experiment except with negligible probability, i.e., $|\Pr[\mathbf{Tamper}_{cnmc}^{A,m} = 1]$ - $\Pr[\mathbf{SimTamper}_{cnmc}^{A,0^n} = 1]| \leq \epsilon(n)$. In $\mathbf{Tamper}_{cnmc}^{A,m}$ experiment, an adversary A proceeds with q number of leakage and tampering queries in real environment until the *self-destruct* state is invoked. $\mathbf{SimTamper}_{cnmc}^{A,0^n}$ experiment simulates the adversaries view in an ideal environment. Here, the simulator $S = (S_0, S_1)$ is constructed to execute the $\mathbf{SimTamper}_{cnmc}^{A,0^n}$ experiment. The simulator S_0 generates a triplet (α, pk, sk) and passes it to S_1. α is an untamperable CRS and *(pk, sk)* pair is used to make the simulated proof of statement in *Xtr* algorithm. The goal of S_1 is to simulate the actual tampering experiment. It consists of two algorithms (T_0, T_1) with tampering

functions f_0^i and f_1^i ($i \leq q \wedge q \in poly(n)$). Algorithm T_0 works on the codeword M_0 with tampering function f_0^i and T_1 works on the codeword M_1 with tampering function f_1^i. *S*imulated experiment proceeds with encoding of message $0^n||r$ whereas real experiment proceeds with message $m||r$ *(*$r \leftarrow \{0,1\}^n$*).* To show that simulation works in a proper way, distribution of simulated experiment is changed incrementally until we reach to the real tampering experiment $\mathbf{Tamper}_{cnmc}^{A,m}$. At each step, a negligible amount of error is introduced. Such change is not noticeable due to the security of *lrs s*cheme. In this way, encryption of 0^n switches to the codeword *M,* i.e., encoding of message *m.* S_1 calls (T_0, T_1) in the interleaved manner and experiment stops when outputs from both algorithms are unequal, i.e., $T_0(M_0, f_0^i, r, i) \neq T_1(M_1, f_1^i, r, i)$. Any further query would return $\perp$ and experiment leads to *self-destruct* in $\mathbf{SimTamper}_{cnmc}^{A,0^n}$. Whenever the experiment triggers *self-destruct, s*ecurity of continuous non-malleability reduces to the security of underlying *lrs* scheme. Alternatively, we can say that if an adversary A breaks the security of continuous non-malleability then there exists an efficient reduction that breaks the security of *lrs* which contradicts the fact that *lrs* scheme is secure. S_1 simulates the actual reduction with (T_0, T_1) in the following way.

Algorithm 3 illustrates the working strategy of the simulated tampering experiment T_0. It parses the left part of a codeword first and applies the leakage function $\tau_0^i()$. The maximum leakage bound tolerated by T_0 is *l*. All leakage values are stored in $\mu_0[i]$ array. Then, tampered codeword $M_0^{'}$ is obtained after applying f_0^i on M_0, i.e., $M_0^{'} = f_0^i(M_0) = ((k_0^{'}, c_0^{'}), p_0^{'}, (c_{key}^{'}, c_1^{'}), \pi_0^{'}, \pi_1^{'})$. If M_0 and $M_0^{'}$ are equal, $\delta_0[i]$ array is set to *same**. Next, the verification of statement is checked and in case, it is unsuccessful, $\delta_0[i]$ array is set to $\perp$ and the experiment stops. If the original proof of statement π and the tampered one $\pi^{'}$ are same, $\delta_0[i]$ array is set to $\perp$ and it returns $\perp$. Extractor algorithm *Xtr* is run to extract $k_1^{'}$ from the simulated proof of statement with the extractor key *sk*, i.e., $k_1^{'} \leftarrow Xtr^{c_0^{'}}(\alpha, ((c_{key}^{'}, c_1^{'}), \pi_1^{'}), sk)$. Further, the key $k_1^{'}$ in conjunction with $k_0^{'}$ is XORed to form the original key $k^{'}$ which is checked against $\mathfrak{D}_{k_0}(c_{key})$. If both are same, $p_1^{'} \leftarrow \mathfrak{D}_{k^{'}}(c_1^{'})$ is called. Next, the $\mathfrak{Dec}^{lrs}(p_0^{'}, p_1^{'})$ algorithm is invoked to retrieve the message $m^{'}$. Since tampering experiment is non-persistent, a separate memory $\mathfrak{M}$ stores all the tampered codeword along with leakage and tampering data, i.e., $\delta_0[i]$ and $\mu_0[i]$.

Algorithm 4 describes the simulated tampering experiment T_1. It starts by parsing right part of a codeword M_1 and calculates leakage through $\tau_1^i()$. The maximum leakage tolerated by T_1 is upper bounded to *l*. $\mu_1[i]$ array stores the leakage data and $\delta_1[i]$ stores all the tampering information. At each query invocation, tampering function f_1^i is applied on M_1. Next, if verification of the statement with label $c_0^{'}$ is successful, proof of statement is compared with the tampered one. In case of successful comparison, *Xtr* algorithm retrieves $k_0^{'}$ from the simulated proof of statement, i.e., $k_0^{'} \leftarrow Xtr^{c_1^{'}}(\alpha, ((c_{key}^{'}, c_0^{'}), \pi_0^{'}), sk)$. The original key $k^{'}$ is formed and compared with $\mathfrak{D}_{k_0}(c_{key})$. Finally, $p_0^{'}$ is recovered from *lrs* and $\mathfrak{Dec}^{lrs}(p_0^{'}, p_1^{'})$ is invoked. The $\mathfrak{Dec}^{lrs}(p_0^{'}, p_1^{'})$ algorithm returns $m^{'}$.

The simulator S_1 runs algorithm T_0 and T_1 alternatively as long as their outputs are same. Let $\tilde{\mathcal{H}}_\infty(M_0|\tau_0^i(M_0))$ be the average conditional entropy. It captures the scenario that best chance of guessing M_0 when some information is available through side channel leakages $\tau_0^i(M_0)$ to the adversary A. Information theoretically, we can write $\tilde{\mathcal{H}}_\infty(M_0|\tau_0^i(M_0)) = \tilde{\mathcal{H}}_\infty(M_1|\tau_1^i(M_1))$ from the working strategy of the simulator S_1. $\tilde{\mathcal{H}}_\infty(M_0|\tau_0^i(M_0))$ can be written as follows (**Lemma 2.1.3**).

$$\tilde{\mathcal{H}}_\infty(M_0|\tau_0^i(M_0)) = \mathcal{H}_\infty(M_0) - l$$

Similarly,

$$\tilde{\mathcal{H}}_\infty(M_1|\tau_1^i(M_1)) = \mathcal{H}_\infty(M_1) - l$$

Here, $\tau_0^i(M_0)$ or $\tau_1^i(M_1)$ can leak at most *l* bits as per security of the *lrs* scheme. The simulator S_1 runs until *self-destruct* is invoked or returns $\perp$. Let q be the maximum number of queries that are made by A in $\mathbf{Tamper}_{cnmc}^{A,m}$. It is assumed that the experiment stops at q^{th} query. In case of $\mathbf{SimTamper}_{cnmc}^{A,0^n}$, same number of queries are performed and the experiment returns $\perp$ whenever outputs from T_0 and T_1 are different. The algorithm $T_0(M_0, f_0^q, r, q)$ and $T_1(M_1, f_1^q, r, q)$ are *l* leaky. For 1 to $(q-1)^{th}$ query, we get the below equation with the assumption that function output cannot be more informative than its own input and last inequality comes from **Lemma 2.1.4**. Apart from that M_1, $T_0(M_0, f_0^q, r, q))$ do not give much useful information about M_0 to guess the message m and it decreases the min-entropy of M_0 by $\mathcal{O}(n)$, i.e., its size. Hence, the security of codeword reduces to the security of leakage resilient storage.

$$\tilde{\mathcal{H}}_\infty(M_0|T_0(M_0,f_0^1,r,1),..,T_0(M_0,f_0^q,r,q))$$
$$= \tilde{\mathcal{H}}_\infty(M_0|T_1(M_1,f_1^1,r,1),..,T_1(M_1,f_1^{q-1},r,q-1),T_0(M_0,f_0^q,r,q)).$$
$$\implies \tilde{\mathcal{H}}_\infty(M_0|T_1(M_1,f_1^1,r,1),..,T_1(M_1,f_1^{q-1},r,q-1),T_0(M_0,f_0^q,r,q))$$
$$\geq \tilde{\mathcal{H}}_\infty(M_0|M_1,q,T_0(M_0,f_0^q,r,q)).$$

At each query invocation, tampered output from both sides of (M_0, M_1) are compared and if it matches, leak the entire codeword. At last query invocation when output from both sides are not same (also $\tau_0^q(M_0) \neq \tau_1^q(M_1)$), leak the entire tampered codeword so that total leakage is upper bounded by $\mathcal{O}(n)$. Apart from that *lrs* in both parts of the codeword can tolerate leakages upto $2l$ (l bits from each side) bits. Combining the parameters, we need l' at least greater than $(2l + n)$ to work the simulator S_1 properly.

Conclusion

In this work, we propose a generic method to construct continuously non-malleable codes from any kind of block cipher in split-state model. The length of codeword depends on the block size of underlying cipher. A non-persistent version of tampering with *self-destruct* capability is considered here. Further research work can be pursued to construct *super-strong* continuously non-malleable codes with *self-destruct* or without *self-destruct* capability, and non-persistent tampering attempts from block ciphers in split-state model.

Acknowledgements
The authors did not receive support from any organization for the submitted work.

Authors' contributions
All the authors read and approved the final manuscript.

Funding
No funding is received for conducting this study.

Availability of data and materials
No such data is used in this research work.

Declarations

Competing interests
The authors have no relevant financial or non-financial interests to disclose. The authors have no financial or proprietary interests in any material discussed in thisarticle.

Received: 3 January 2023 Accepted: 16 March 2023
Published online: 01 October 2023

References

Aggarwal D, Agrawal S, Gupta D, Maji HK, Pandey O, Prabhakaran M (2016) Optimal computational split-state non-malleable codes. In: Kushilevitz E, Malkin T (eds) TCC 2016, vol 9563. LNCS. Springer, Heidelberg, pp 393–417

Aggarwal D, Kazana T, Obremski M (2017) Inception makes non-malleable codes stronger. In: Kalai Y, Reyzin L (eds) TCC 2017, vol 10678. LNCS. Springer, Cham, pp 319–343

Aggarwal D, Döttling N, Nielsen JB, Obremski M, Purwanto E (2019) Continuous non-malleable codes in the 8-split-state model. In: Ishai Y, Rijmen V (eds) EUROCRYPT 2019, Part I, vol 11476. LNCS. Springer, Cham, pp 531–561

Aggarwal D, Dodis Y, Kazana T, Obremski M (2015) Non-malleable reductions and applications. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, pp 459–468

Aggarwal D, Dodis Y, Lovett S (2014) Non-malleable codes from additive combinatorics. In: STOC, pp 774–783

Banik S, Bogdanov A, Isobe T, Shibutani K, Hiwatari H, Akishita T, Regazzoni F (2015) Midori: a block cipher for low energy. In: Iwata T et al (eds) ASIACRYPT 2015, vol 9453. LNCS. Springer, Heidelberg, pp 411–436

Bellare M, Kohno T (2003) A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In: Biham E (ed) EUROCRYPT 2003, vol 2656. LNCS. Springer, Heidelberg, pp 491–506

Bellare M, Cash D, Miller R (2011) Cryptography secure against related-key attacks and tampering. In: Lee DH, Wang X (eds) ASIACRYPT 2011, vol 7073. LNCS. Springer, Heidelberg, pp 486–503

Bellare M, Paterson KG, Thomson S (2012) RKA security beyond the linear barrier: IBE, encryption and signatures. In: Wang X, Sako K (eds) ASIACRYPT 2012, vol 7658. LNCS. Springer, Heidelberg, pp 331–348

Boneh D, DeMillo RA, Lipton RJ (2001) On the importance of eliminating errors in cryptographic computations. J Cryptol 14(2):101–119

Chen B, Chen Y, Hostáková K, Mukherjee P (2019) Continuous space-bounded non-malleable codes from stronger proofs-of-space. In: CRYPTO, pp 467–495

Dachman-Soled D, Kulkarni M (2019) Upper and lower bounds for continuous non-malleable codes. In: PKC, pp 519–548

Damgård I, Faust S, Mukherjee P, Venturi D (2013) Bounded tamper resilience: How to go beyond the algebraic barrier. In: Sako K, Sarkar P (eds) ASIACRYPT 2013, Part II, vol 8270. LNCS. Springer, Heidelberg, pp 140–160

Davì F, Dziembowski S, Venturi D (2010) Leakage-resilient storage. In: Garay JA, De Prisco R (eds) SCN 2010, vol 6280. LNCS. Springer, Heidelberg, pp 121–137

De Santis A, Di Crescenzo G, Ostrovsky R, Persiano G, Sahai A (2001) Robust non-interactive zero knowledge. In: Kilian J (ed) CRYPTO 2001, vol 2139. LNCS. Springer, Heidelberg, pp 566–598

Dziembowski S, Faust S (2011) Leakage-resilient cryptography from the inner-product extractor. In: Lee DH, Wang X (eds) ASIACRYPT 2011, vol 7073. LNCS. Springer, Heidelberg, pp 702–721

Dziembowski S, Kazana T, Obremski M (2013) Non-malleable codes from two-source extractors. In: Canetti R, Garay JA (eds) CRYPTO 2013, vol 8043. LNCS. Springer, Heidelberg, pp 239–257

Dziembowski S, Pietrzak K, Wichs D (2018) Non-malleable codes. J ACM 65(4):1–32

Dziembowski S, Pietrzak K, Wichs D (2010) Non-malleable codes. In: Yao AC-C (ed) ICS 2010, Tsinghua University Press, Beijing, pp 434-452

Faonio A, Nielsen JB, Simkin M, Venturi D (2018) Continuously non-malleable codes with split-state refresh. In: Preneel B, Vercauteren F (eds) ACNS 2018, vol 10892. LNCS. Springer, Cham, pp 1–19

Faust S, Mukherjee P, Nielsen JB, Venturi D (2014a) Continuous non-malleable codes. In: Lindell Y (ed) TCC 2014, vol 8349. LNCS. Springer, Heidelberg, pp 465–488

Faust S, Mukherjee P, Nielsen JB, Venturi D (2020) Continuously non-malleable codes in the split-state model. J Cryptol 33(4):2034–77

Faust S, Mukherjee P, Venturi D, Wichs D (2014b) Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: EUROCRYPT. pp 111–128
Fehr S, Karpman P, Mennink B (2018) Short non-malleable codes from related-key secure block ciphers. IACR Trans Symm Cryptol, 336–352
Gennaro R, Lysyanskaya A, Malkin T, Micali S, Rabin T (2004) Algorithmic tamper-proof (ATP) security: theoretical foundations for security against hardware tampering. In: Naor M (ed) TCC 2004, vol 2951. LNCS. Springer, Heidelberg, pp 258–277
Ghosal AK, Ghosh S, Roychowdhury D (2022) Practical non-malleable codes from symmetric-key primitives in 2-split-state model. In: Ge C, Guo F (eds) Provable and practical security
Goldreich O, Micali S, Wigderson A (1991) Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. J ACM 38(3):691–729
Handschuh H, Naccache D (2002) SHACAL: A Family of Block Ciphers. Submission to the NESSIE project
Jafargholi Z, Wichs D (2015) Tamper detection and continuous non-malleable codes. In: Dodis Y, Nielsen JB (eds) TCC 2015, vol 9014. LNCS. Springer, Heidelberg, pp 451–480
Joan D, Vincent R (2002) The Design of Rijndael. Springer-Verlag, New York Inc, Secaucus
Kalai YT, Kanukurthi B, Sahai A (2011) Cryptography with Tamperable and Leaky Memory. In: Rogaway P (ed) CRYPTO 2011, vol 6841. LNCS. Springer, Heidelberg, pp 373–390
Kiayias A, Liu FH, Tselekounis Y (2016) Practical non-malleable codes from l-more extractable hash functions. In: Weippl ER, Katzenbeisser S, Kruegel C, Myers AC, Halevi S (eds) ACM CCS 2016, ACM Press, pp 1317–1328
Liu F-H, Lysyanskaya A (2012) Tamper and leakage resilience in the split-state model. In: Safavi-Naini R, Canetti R (eds) CRYPTO 2012, vol 7417. LNCS. Springer, Heidelberg, pp 517–532
Ostrovsky R, Persiano G, Venturi D, Visconti I (2018) Continuously non-malleable codes in the split-state model from minimal assumptions. In: Shacham H, Boldyreva A (eds) CRYPTO 2018, Part III, vol 10993. LNCS. Springer, Cham, pp 608–639
Sergei P, Ross J (2002) Optical fault induction attacks. In: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems, Springer, Heidelberg, pp 2-12

Publisher's Note