Message-locked Encryption with File Update

Suyash Kandele[0000−0002−5887−2907] and Souradyuti Paul[0000−0001−5404−9975]

Indian Institute of Technology Bhilai, C.G., India
{suyashk,souradyuti}@iitbhilai.ac.in

Abstract. *Message-locked encryption (MLE)* (formalized by Bellare, Keelveedhi and Ristenpart, 2013) is an important cryptographic primitive that supports deduplication in the *cloud*. *Updatable block-level message-locked encryption (UMLE)* (formalized by Zhao and Chow, 2017) adds the *update* functionality to the *MLE*. In this paper, we formalize and extensively study a new cryptographic primitive *file-updatable message-locked encryption (FMLE)*. *FMLE* can be viewed as a generalization of the *UMLE*, in the sense that unlike the latter, the former does not require the existence of *BL-MLE (block-level message-locked encryption)*. *FMLE* allows more flexibility and efficient methods for updating the ciphertext and tag.

Our second contribution is the design of two efficient *FMLE* constructions, namely, RevD-1 and RevD-2, whose design principles are inspired from the very unique *reverse decryption* functionality of the *FP* hash function (designed by Paul, Homsirikamol and Gaj, 2012) and the *APE* authenticated encryption (designed by Andreeva *et al.*, 2014). With respect to *UMLE* – which provides so far the most efficient update function – RevD-1 and RevD-2 reduce the total update time by at least 50%, on average. Additionally, our constructions are storage efficient; in particular, *ciphertext expansion* and *tag storage* for our constructions are constant, while they are logarithmic and linear for the *UMLE*, making it difficult to implement in various practical applications. We also provide detailed proofs of security for our constructions and give extensive comparison between our and the existing constructions. Being randomized, our constructions are secure against *dictionary attacks*. On the other hand, they are vulnerable to *strong tag consistency* (STC) attacks (as is the case with any randomized *MLE*).

1 Introduction

MLE. *Message-locked encryption (MLE)* is a special type of encryption, where the decryption key is derived from the message itself. The main application of *MLE* is in the secure deduplication of data in the *cloud*, where *MLE* removes the need for storing multiple copies of identical data, without compromising their privacy and, thereby, helps to reduce the storage costs. Given the cloud services being on the rise, this primitive is gaining importance.

The first attempt to solve the problem of deduplication was made in 2002 by Douceur *et al.*[15], who came up with the idea of *Convergent Encryption*

(CE). Bellare, Keelveedhi and Ristenpart [6] studied this subject in a formal way and named it *message-locked encryption (MLE)*. They also gave efficient constructions of *MLE*.
UMLE. As seen before, *MLE* does not inherently support the *file-update* and the *proof of ownership* functionalities in its definition. *UMLE* solves this issue by adding three functionalities – file-update, PoW algorithms for prover and verifier – to the existing definition of *MLE*. The main drawback of *UMLE* is that the functionalities are constructed from another cryptographic primitive, namely, *BL-MLE*, which is nothing but *MLE* executed on a fixed-sized block. Such a *BL-MLE*-based *UMLE* entails degradation of performance for encryption and decryption [20]. Another drawback of *UMLE* is that the update of file-tag is an expensive operation.

Motivation for studying FMLE. From the high level, both *UMLE* and *FMLE* have identical functionalities; the main difference, however, is that the former is necessarily based on *BL-MLE*, but the latter may or may not be. Therefore, the definition of *FMLE* can be viewed as a generalisation of *UMLE*, where we remove the constraint of using *BL-MLE*. The motivation for studying *FMLE* is clear from the drawbacks of *UMLE* mentioned above. These motivations are:

Does there exist an FMLE *scheme which is not based on* BL-MLE*? If such a construction exists, is it more efficient than* UMLE*?*

Studying *FMLE* is a futile exercise, if both the answers are in the negative. A moment's reflection suggests that the answer to the first question is actually 'Yes'. A trivial *FMLE* construction – not based on *BL-MLE* – always exists, where the file-update function is designed the following way: apply decryption to the ciphertext to recover the original plaintext; edit/modify the original message; and finally encrypt the updated message. Note that this trivial file-update function does not need any *BL-MLE*, therefore, it is an *FMLE*, but certainly not a *UMLE* scheme. The main drawback of this *FMLE* scheme is that this is several orders of magnitude slower than a *UMLE* scheme. Therefore, the main challenge is:

Does there exists an FMLE *scheme more efficient than* UMLE*?*

Searching for such a construction is the main motivation of this paper.

Our Contribution. Our first contribution is formalizing the new cryptographic notion *file-updatable message-locked encryption (FMLE)*. We also propose two efficient *FMLE* constructions RevD-1 and RevD-2: their update functions are at least 50% faster (on average).[1] Also, our constructions are more space efficient than the so-far best *MLE* variants; in particular, the *ciphertext expansion* and *tag storage* in RevD-1 and RevD-2 are constant, while they are logarithmic and linear (or may be worse) for other similar time-efficient cases. In

[1] The term RevD is a shorthand for ***Rev****erse* ***D****ecryption*.

order to obtain this improvement in the performance, our constructions critically exploit a very unique feature – what we call *reverse decryption* – of the hash function *FP* and the authenticated encryption *APE*. We also present proofs of security of our constructions. Extensive comparison of our constructions with the others – in terms of time complexity, storage requirements and security properties – have also been provided (see Table 1). Being randomized, our constructions are secure against the *dictionary attacks*, however, they lack STC security, like all other randomized *MLE*s.

Related Work. We now describe various pieces of work done by several researchers that are related to *FMLE*. Douceur *et al.* are the first to come up with the idea of *Convergent Encryption (CE)* in 2002, where the key was calculated as a hash of the message, and then this key was used for encryption [15]. Bellare, Keelveedhi and Ristenpart formalized *CE* in the form of *message-locked encryption (MLE)* [6]. They also provided a systematic discussion of various *MLE* schemes. In a separate paper, these authors also designed a new system DupLESS that supports deduplication even if the message entropy is low [5].

Beelare and Keelveedhi extended *message-locked encryption* to *interactive message-locked encryption*, and have addressed the cases when the messages are correlated as well as dependent on the public parameters, leading to weakened privacy [4]. Abadi *et al.* gave two new constructions for the *i-MLE*; these fully randomized schemes also supported equality-testing algorithm for finding ciphertexts derived from identical messages [1]. Jiang *et al.* gave an efficient logarithmic-time deduplication scheme that substantially reduces the equality-testing in the *i-MLE* schemes [17].

Canard, Laguillaumie and Paindavoine introduced *deduplication consistency* – a new security property – that prevents the clients from bypassing the deduplication protocol. This is accomplished by introducing a new feature named *verifiability (of the well-formation) of ciphertext at the server* [13]. They also proposed a new ElGamal-based construction satisfying this property. Wang *et al.* proposed a stronger security notion PRV-CDA3 and showed that their new construction *ME* is secure in this model [19].

Chen *et al.* proposed the *block-level message-locked encryption (BL-MLE)*, which is nothing but breaking a big message into smaller chunks – called *blocks* – and then applying *MLE* on the individual *blocks* [14]. Huang, Zhang and Wang showed how to integrate the functionality *proof of storage (PoS)* with *MLE* by using a new data structure *Quadruple Tags* [16]. Zhao and Chow proposed the use of *BL-MLE* to design *Efficiently Updatable Block-Level Message-Locked Encryption (UMLE)* scheme which has an additional functionality of updating the ciphertext that costs sub-linear time [20].

Organization of the paper. In Sect. 2, we discuss the preliminaries including the notation and basic definitions. In Sect. 3, we give the formal definition of *FMLE*. Section 4 describes the application of *FMLE* in the deduplication protocol. In Sects. 5 and 6, we construct the *FMLE* schemes by tweaking the existing *MLE* and *UMLE* schemes. In Sect. 7, we describe the two new efficient *FMLE*

schemes and we compare them with the various *FMLE* schemes and conclude our paper in Sect. 8.

2 Preliminaries

2.1 Notation

The expression $M := x$ denotes that the value of x is assigned to M, and $M := \mathcal{D}(x)$ denotes that the value returned by function $\mathcal{D}(\cdot)$, on input x, is assigned to M. $M = x$ denotes the equality comparison of the two variables M and x, and $M = \mathcal{D}(x)$ denotes the equality comparison of the variable M with the output of $\mathcal{D}(\cdot)$, on input x. The XOR or $\oplus$ denotes the bit-by-bit *exclusive-or* operation on two binary strings of same length. The concatenation operation of $p \geq 1$ strings $s_1, s_2, \cdots, s_p$ and assignment to the variable s is denoted by $s := s_1||s_2||\cdots||s_p$. The parsing of string s into $p \geq 1$ strings $s_1, s_2, \cdots, s_p$ is denoted by $s_1||s_2||\cdots||s_p := s$. The length of string M is denoted by $|M|$. The set of all binary strings of length ℓ is denoted by $\{0,1\}^\ell$. The set of all binary strings of any length is denoted by $\{0,1\}^*$. A vector of strings is denoted by $\boldsymbol{M}$ and i-th string in $\boldsymbol{M}$ is denoted by $\boldsymbol{M}^{(i)}$. The number of strings in $\boldsymbol{M}$ is denoted by $\|\boldsymbol{M}\|$. The infinite set of all binary strings of any length is denoted by $\{0,1\}^{**}$. The set of all Natural numbers is denoted by $\mathbb{N}$. We denote that M is assigned a binary string of length k chosen randomly and uniformly by $M \xleftarrow{\$} \{0,1\}^k$. To mark any invalid string (may be input string or output string), the symbol $\perp$ is used. $(\boldsymbol{M}, Z) \xleftarrow{\$} \mathcal{S}(1^\lambda)$ denotes the assignment of outputs given randomly and uniformly by $\mathcal{S}$ to M and Z.

2.2 Dictionary Attack

A *dictionary attack* is defined to be a *brute-force attack*, where the adversary first builds a dictionary off-line, and then processes every element of the dictionary to determine the correct solution against an online challenge. For example, suppose that the hash of a message is given as a challenge to the adversary for her to determine the correct message. If the entropy of the message is low, then the adversary generates the dictionary of all possible messages and their corresponding hash values off-line; and given the online challenge, she selects the message whose hash value matches the challenge. *Any deterministic* MLE *with low message entropy is broken by dictionary attack.*

2.3 Proof of Ownership

Proof-of-ownership (PoW) is an interactive protocol where the owner of file proves the ownership of a file to the cloud storage. This protocol assumes that the adversary does not have access to the entire ciphertext which was uploaded onto the cloud by some previous (or first) owner, but he may know the tag, which is a small fraction of the entire information. In this protocol, the cloud

storage provider generates a challenge Q and sends it to the client, along with some other information. The client computes the proof P corresponding to the given challenge Q and sends it back to the cloud. The cloud verifies it and if the verification is successful, then the client is granted access, otherwise the access is denied.

2.4 Ideal Permutation

Let $\pi/\pi^{-1}\colon \{0,1\}^n \mapsto \{0,1\}^n$ be a pair of oracles. The pair π/π^{-1} is called an *ideal permutation* if the following three properties are satisfied.
1. $\pi^{-1}(\pi(x)) = x$ and $\pi(\pi^{-1}(x)) = x$, for all $x \in \{0,1\}^n$.
2. Suppose, x_k is the k-th query ($k \geq 1$), submitted to the oracle π, and $y \in \{0,1\}^n$. Then, for the current query x_i:

$$\Pr\left[\pi(x_i) = y \middle| \pi(x_1) = y_1, \pi(x_2) = y_2, \cdots, \pi(x_{i-1}) = y_{i-1}\right]$$

$$= \begin{cases} 1, & \text{if } x_i = x_j, y = y_j, j < i. \\ 0, & \text{if } x_i = x_j, y \neq y_j, j < i, \\ 0, & \text{if } x_i \neq x_j, y = y_j, j < i, \\ \frac{1}{2^n - i + 1}, & \text{if } x_i \neq x_j, y \neq y_j, j < i. \end{cases}$$

3. Suppose, y_k is the k-th query ($k \geq 1$), submitted to the oracle π^{-1}, and $x \in \{0,1\}^n$. Then, for the current query y_i:

$$\Pr\left[\pi^{-1}(y_i) = x \middle| \pi^{-1}(y_1) = x_1, \pi^{-1}(y_2) = x_2, \cdots, \pi^{-1}(y_{i-1}) = x_{i-1}\right]$$

$$= \begin{cases} 1, & \text{if } y_i = y_j, x = x_j, j < i. \\ 0, & \text{if } y_i = y_j, x \neq x_j, j < i, \\ 0, & \text{if } y_i \neq y_j, x = x_j, j < i, \\ \frac{1}{2^n - i + 1}, & \text{if } y_i \neq y_j, x \neq x_j, j < i. \end{cases}$$

2.5 Random Function

Let $\mathsf{rf}\colon \{0,1\}^n \mapsto \{0,1\}^n$. Then rf is called a *random function* if the following property is satisfied. Suppose, x_k is the k-th query ($k \geq 1$), submitted to the rf, and $y \in \{0,1\}^n$. Then, for the current query x_i:

$$\Pr\left[\mathsf{rf}(x_i) = y \middle| \mathsf{rf}(x_1) = y_1, \mathsf{rf}(x_2) = y_2, \cdots, \mathsf{rf}(x_{i-1}) = y_{i-1}\right]$$

$$= \begin{cases} 1, & \text{if } x_i = x_j, y = y_j, j < i. \\ 0, & \text{if } x_i = x_j, y \neq y_j, j < i, \\ \frac{1}{2^n}, & \text{if } x_i \neq x_j, j < i. \end{cases}$$

2.6 Unpredictable Sources

We are modelling the security based on an unpredictable message source which is a PT algorithm, denoted $\mathcal{S}(\cdot)$, that returns $(\boldsymbol{M}, Z)$ on input 1^λ, where *vector of messages* $\boldsymbol{M} \in \{0,1\}^{**}$ and *auxiliary information* $Z \in \{0,1\}^*$. We consider that $\mathcal{S}(\cdot)$ is a Public source, that is, it is known to all the parties including the adversary. Here, $\boldsymbol{M}$ has $m(1^\lambda)$ number of strings, i.e., $\|\boldsymbol{M}\| = m(1^\lambda)$ and the length of each string $\boldsymbol{M}^{(i)}$ is $l(1^\lambda, i)$, i.e., $|\boldsymbol{M}^{(i)}| = l(1^\lambda, i)$ for $i \in \{1, 2, \cdots, m(1^\lambda)\}$. Here, m and l are two functions. We require that the two strings $\boldsymbol{M}^{(i_1)} \neq \boldsymbol{M}^{(i_2)}$, for $i_1 \neq i_2$ and $i_1, i_2 \in \{1, 2, \cdots, m(1^\lambda)\}$. Associated with the *source* $\mathcal{S}(\cdot)$ is a real number $GP_{\mathcal{S}}$, namely, the *Guessing Probability of source*, which is the maximum of all the probabilities of guessing a single string in $\boldsymbol{M}$, given the auxiliary information. The formal definition is $GP_{\mathcal{S}}(1^\lambda) \stackrel{def}{=} \max_{i \in \{1,2,\cdots,m(1^\lambda)\}} GP(\boldsymbol{M}^{(i)}|Z)$. The source $\mathcal{S}(\cdot)$ is said to be unpredictable if the value of $GP_{\mathcal{S}}$ is negligible. We now define the *min-entropy* $\mu_{\mathcal{S}}(\cdot)$ of the source $\mathcal{S}(\cdot)$ as $\mu_{\mathcal{S}}(1^\lambda) = -\log(GP_{\mathcal{S}}(1^\lambda))$. The source $\mathcal{S}(\cdot)$ is said to be a valid source for an *MLE* scheme Π if $\boldsymbol{M}^{(i)} \in \mathcal{M}, \forall i \in \{1, 2, \cdots, m(1^\lambda)\}$.

2.7 Message-locked Encryption

The definition of *message-locked encryption (MLE)* has already been described in [6]. We briefly re-discuss it below, with a few suitable changes in the notation to suit the present context.

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. An *MLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ is a pair of algorithms over a PPT setup $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$, $\mathcal{M}^{(\Pi)}$, $\mathcal{C}^{(\Pi)}$ and $\mathcal{T}^{(\Pi)}$, denoting the *key*, *message*, *ciphertext* and *tag spaces* respectively.
2. The PPT encryption algorithm $\Pi.\mathcal{E}$ takes as inputs $params^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and returns a 3-tuple $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, where $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$.
3. The decryption algorithm $\Pi.\mathcal{D}$ is a deterministic algorithm that takes as inputs $params^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$, and returns $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) \in \mathcal{M}^{(\Pi)} \cup \{\perp\}$. The decryption algorithm $\Pi.\mathcal{D}$ returns $\perp$ if the key K, ciphertext C and tag T are not generated from a valid message.
4. We restrict $|C|$ to be a linear function of $|M|$.

Key Correctness. Let $M, M' \in \mathcal{M}^{(\Pi)}$. Suppose:

- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{E}(params^{(\Pi)}, M')$.

Then *key correctness* of Π requires that if $M = M'$, then $K = K'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

DECRYPTION CORRECTNESS. Let $M \in \mathcal{M}^{(\Pi)}$. Suppose:

- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$.

Then *decryption correctness* of Π requires that $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) = M$, for all $\lambda \in \mathbb{N}$ and all $M \in \mathcal{M}^{(\Pi)}$.

TAG CORRECTNESS. Let $M, M' \in \mathcal{M}^{(\Pi)}$. Suppose:

- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{E}(params^{(\Pi)}, M')$.

Then *tag correctness* of Π requires that if $M = M'$, then $T = T'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

The two security games are written in the form of a challenger-adversary framework.

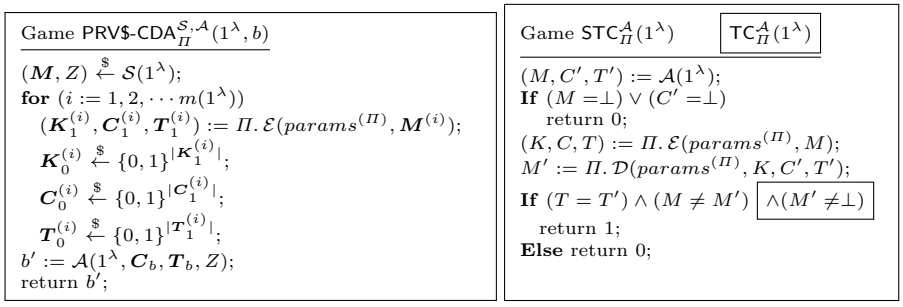


Fig. 1. Games defining PRV$-CDA, STC and TC security of *MLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$.

PRIVACY. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an *MLE* scheme. Since, no *MLE* scheme can provide PRV$-CDA security for predictable messages (even if the scheme is randomized), we use an unpredictable message source $\mathcal{S}$, as defined in Sect. 2.6, to design our security notion. For an *MLE* scheme, we design the *privacy against chosen distribution attack* PRV$-CDA security game in Fig. 1. Here, the challenger generates a vector of messages $\boldsymbol{M}$ and some auxiliary information Z using the source $\mathcal{S}(1^\lambda)$, encrypts the string $\boldsymbol{M}^{(i)}$, where $i \in \{1, 2, \cdots, m(1^\lambda)\}$, using $\Pi.\mathcal{E}$ to obtain $(\boldsymbol{K}_1^{(i)}, \boldsymbol{C}_1^{(i)}, \boldsymbol{T}_1^{(i)})$, computes the random strings $\boldsymbol{K}_0^{(i)}$, $\boldsymbol{C}_0^{(i)}$ and $\boldsymbol{T}_0^{(i)}$ of length $|\boldsymbol{K}_1^{(i)}|$, $|\boldsymbol{C}_1^{(i)}|$ and $|\boldsymbol{T}_1^{(i)}|$ respectively, and sends $(\boldsymbol{C}_b, \boldsymbol{T}_b, Z)$ to the adversary. The adversary has to return a bit b' indicating whether the ciphertext $\boldsymbol{C}_b$ and tag $\boldsymbol{T}_b$ corresponds to message $\boldsymbol{M}$ or is it a collection of random strings. If the values of b and b' coincide, then the adversary wins the game.

Now, we define the advantage of a PRV$-CDA adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Big| \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b=1)=1] - \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b=0)=1] \Big|.$$

An *MLE* scheme Π is said to be $\mathsf{PRV\$\text{-}CDA}$ secure over a set of valid PT sources for *MLE* scheme Π, $\overline{\mathcal{S}} = \{\mathcal{S}_0, \mathcal{S}_1, \cdots\}$, for all PT adversaries $\mathcal{A}$ and for all $\mathcal{S}_i \in \overline{\mathcal{S}}$, if $Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S}_i,\mathcal{A}}(\cdot)$ is negligible. An *MLE* scheme Π is said to be $\mathsf{PRV\$\text{-}CDA}$ secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(\cdot)$ is negligible, for all valid PT source $\mathcal{S}$ for Π.

Tag Consistency. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ be an *MLE* scheme. For an *MLE* scheme, we design the STC and TC security games in Fig. 1, which aims to provide security against duplicate faking attacks. In a duplicate faking attack, two unidentical messages – one fake message produced by an adversary and a legitimate one produced by an honest client – produce the same tag, thereby causing loss of message and hampers the integrity. In an erasure attack, the adversary replaces the ciphertext with a fake message that decrypts successfully.

The adversary returns a message M, a ciphertext C' and a tag T'. If the message or ciphertext is invalid, the adversary loses the game. Otherwise, the challenger computes encryption key K, ciphertext C and tag T corresponding to message M using $\Pi.\mathcal{E}$, and computes the message M' corresponding to key K, ciphertext C' and tag T' using $\Pi.\mathcal{D}$. If the two tags are equal, i.e. $T = T'$, the message M' is valid, i.e. $M' \neq \perp$, and the two messages are unequal, i.e. $M \neq M'$, then the adversary wins the TC game.

Now, we define the advantage of a TC adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{TC}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{TC}^{\mathcal{A}}_{\Pi}(1^\lambda) = 1].$$

Now, we define the advantage of an STC adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{STC}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{STC}^{\mathcal{A}}_{\Pi}(1^\lambda) = 1].$$

An *MLE* scheme Π is said to be TC (or STC) secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{TC}}_{\Pi,\mathcal{A}}(\cdot)$ (or $Adv^{\mathsf{STC}}_{\Pi,\mathcal{A}}(\cdot)$) is negligible.

2.8 Updatable block-level message-locked encryption

The definition of *updatable block-level message-locked encryption (UMLE)* has already been described in [20]. We briefly re-discuss it below, with a few suitable changes in the notation to suit the present context.

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. A *UMLE* scheme $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TagGen}, \Pi.\mathsf{Dec}, \Pi.\mathsf{Update}, \Pi.\mathsf{UpdateTag}, \Pi.\mathsf{PoWPrf}, \Pi.\mathsf{PoWVer})$ is eight-tuple of algorithms over a PPT setup $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$, $\mathcal{M}^{(\Pi)}$, $\mathcal{C}^{(\Pi)}$ and $\mathcal{T}^{(\Pi)}$, denoting the *key*, *message*, *ciphertext* and *tag spaces* respectively.
2. The PPT key-generation algorithm $\Pi.\mathsf{KeyGen}$ takes as inputs $params^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and returns a set of keys $(k_{mas}, k_1, k_2, \cdots, k_n) := \Pi.\mathsf{KeyGen}(params^{(\Pi)}, M)$, where $k_{mas}, k_1, k_2, \cdots, k_n \in \mathcal{K}^{(\Pi)}$. It uses two routines: $\Pi.\mathsf{B\text{-}KeyGen}$ that takes as input i-th block of message $M[i]$, and returns the block key k_i; and $\Pi.\mathsf{M\text{-}KeyGen}$ that takes as input message M, and returns the master key k_{mas}.
3. The PPT encryption algorithm $\Pi.\mathsf{Enc}$ takes as inputs $params^{(\Pi)}$, a set of keys $k_{mas}, k_1, k_2, \cdots, k_n \in \mathcal{K}^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and returns the ciphertext $C := \Pi.\mathsf{Enc}(params^{(\Pi)}, (k_{mas}, k_1, k_2, \cdots, k_n), M)$, where $C \in \mathcal{C}^{(\Pi)}$. It uses two routines: $\Pi.\mathsf{B\text{-}Enc}$ that takes as input i-th block of message $M[i]$ and corresponding key k_i, and returns the block ciphertext $C[i]$; and $\Pi.\mathsf{BK\text{-}Enc}$ that takes as input block keys $k_1, k_2, \cdots, k_n$ and returns the encrypted block keys $C[n+1], C[n+2], \cdots, C[n']$, where $n' \in \mathcal{O}(n)$.
4. The PPT tag-generation algorithm $\Pi.\mathsf{TagGen}$ takes as inputs $params^{(\Pi)}$ and $C \in \mathcal{C}^{(\Pi)}$, and returns the tag $T := \Pi.\mathsf{TagGen}(params^{(\Pi)}, C)$, where $T \in \mathcal{T}^{(\Pi)}$. It uses two routines: $\Pi.\mathsf{B\text{-}TagGen}$ that takes as input i-th block of ciphertext $C[i]$ and returns the block tag $T[i]$; and $\Pi.\mathsf{M\text{-}TagGen}$ that takes as input ciphertext C and returns the file tag $T[0]$. Then $T = T[0]||T[1]||T[2]||\cdots||T[n']$.
5. The decryption algorithm $\Pi.\mathsf{Dec}$ is a deterministic algorithm that takes as inputs $params^{(\Pi)}, k_{mas} \in \mathcal{K}^{(\Pi)}$ and $C \in \mathcal{C}^{(\Pi)}$, and returns the $\Pi.\mathsf{Dec}(params^{(\Pi)}, k_{mas}, C) \in \mathcal{M}^{(\Pi)} \cup \{\perp\}$. It uses two routines: $\Pi.\mathsf{BK\text{-}Dec}$ that takes as input master key k_{mas} and encrypted block keys $C[n+1], C[n+2], \cdots, C[n']$, and returns a set of block keys $k_1, k_2, \cdots, k_n$; and $\Pi.\mathsf{B\text{-}Dec}$ that takes as input ciphertext $C[i]$ and corresponding key k_i and returns the file block $M[i]$
6. The update-ciphertext algorithm $\Pi.\mathsf{Update}$ takes as inputs $params^{(\Pi)}$, master key $k_{mas} \in \mathcal{K}^{(\Pi)}$, block number to be updated $i \in \mathbb{N}$, new message block $M_{new} \in \mathcal{M}^{(\Pi)}$ and the ciphertext $C \in \mathcal{C}^{(\Pi)}$, and returns the pair $(K', C') := \Pi.\mathsf{Update}(params^{(\Pi)}, (k_{mas}, i, M_{new}), C)$, where $K' \in \mathcal{K}^{(\Pi)}$ and $C' \in \mathcal{C}^{(\Pi)}$.
7. The update-tag algorithm $\Pi.\mathsf{UpdateTag}$ takes as inputs $params^{(\Pi)}$, $T \in \mathcal{T}^{(\Pi)}$, and $C, C' \in \mathcal{C}^{(\Pi)}$, and returns $T' := \Pi.\mathsf{UpdateTag}(params^{(\Pi)}, T, C, C')$, where $T' \in \mathcal{T}^{(\Pi)}$.
8. The PPT PoW algorithm for prover $\Pi.\mathsf{PoWPrf}$ takes as inputs $params^{(\Pi)}$, challenge Q and $M \in \mathcal{M}^{(\Pi)}$, and returns the proof $P := \Pi.\mathsf{PoWPrf}(params^{(\Pi)}, Q, M)$.
9. The PPT PoW algorithm for verifier $\Pi.\mathsf{PoWVer}$ takes as inputs $params^{(\Pi)}$, challenge Q, $T \in \mathcal{T}^{(\Pi)}$ and proof P, and returns the value $val := \Pi.\mathsf{PoWVer}(params^{(\Pi)}, Q, T, P)$, where $val \in \{\mathsf{TRUE}, \mathsf{FALSE}\}$.
10. We restrict $|C|$ to be a linear function of $|M|$.

Decryption Correctness. Let $M = M[1]||M[2]||\cdots||M[n]$. For block message $M[i]$, where $1 \leq i \leq n$, suppose:

- $k_i := \Pi.\mathsf{B\text{-}KeyGen}(M[i])$, and
- $C[i] := \Pi.\mathsf{B\text{-}Enc}(k_i, M[i])$.

Then *decryption correctness* of Π requires that $\Pi.\mathsf{B\text{-}Dec}(k_i, C[i]) = M[i]$, for all $\lambda \in \mathbb{N}$ and all $M[i] \in \mathcal{M}^{(\Pi)}$.

Block Key Retrieval Correctness. Let $M = M[1]||M[2]||\cdots||M[n]$. Suppose:

- $k_i := \Pi.\mathsf{B\text{-}KeyGen}(M[i])$, for block message $M[i]$, where $1 \leq i \leq n$,
- $k_{mas} := \Pi.\mathsf{M\text{-}KeyGen}(M)$, and
- $(C[n+1], C[n+2], \cdots, C[n']) := \Pi.\mathsf{BK\text{-}Enc}(k_1, k_2, \cdots, k_n)$.

Then *block key retrieval correctness* of Π requires that $\Pi.\mathsf{BK\text{-}Dec}(k_{mas}, C[n+1], C[n+2], \cdots, C[n']) = (k_1, k_2, \cdots, k_n)$.

Tag Correctness. Let $M, M' \in \mathcal{M}^{(\Pi)}$, Suppose:

- $k_{mas} := \Pi.\mathsf{KeyGen}(params^{(\Pi)}, M)$,
- $C := \Pi.\mathsf{Enc}(params^{(\Pi)}, k_{mas}, M)$,
- $T := \Pi.\mathsf{TagGen}(params^{(\Pi)}, C)$
- $k'_{mas} := \Pi.\mathsf{KeyGen}(params^{(\Pi)}, M')$,
- $C' := \Pi.\mathsf{Enc}(params^{(\Pi)}, k'_{mas}, M')$, and
- $T' := \Pi.\mathsf{TagGen}(params^{(\Pi)}, C')$.

Then *tag correctness* of Π requires that $\mathsf{if}\ M = M'$, then $T = T'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

Update Correctness. Let $M = M[1]||M[2]||\cdots||M[n]$. Suppose:

- $k_{mas} := \Pi.\mathsf{KeyGen}(params^{(\Pi)}, M)$,
- $C := \Pi.\mathsf{Enc}(params^{(\Pi)}, k_{mas}, M)$, and
- $(K', C') := \Pi.\mathsf{Update}(params^{(\Pi)}, (k_{mas}, i, M'[i]), C)$.

Then *update correctness* of Π requires that $\Pi.\mathsf{Dec}(params^{(\Pi)}, K', C') = M[1]||M[2]||\cdots||M[i-1]||M'[i]||M[i+1]||\cdots||M[n]$.

PoW Correctness. Let $M = M[1]||M[2]||\cdots||M[n]$. Suppose:

- $k_{mas} := \Pi.\mathsf{KeyGen}(params^{(\Pi)}, M)$,
- $C := \Pi.\mathsf{Enc}(params^{(\Pi)}, k_{mas}, M)$, and
- $T := \Pi.\mathsf{TagGen}(params^{(\Pi)}, C)$.

Then *PoW correctness* of Π requires that for any challenge Q, and for the proof $P := \Pi.\mathsf{PoWPrf}(params^{(\Pi)}, Q, M)$, we have the probability $\Pr[\Pi.\mathsf{PoWVer}(params^{(\Pi)}, Q, T, P) = \mathsf{TRUE}] = 1$.

The four security games are written in the form of a challenger-adversary framework.

Privacy. Let $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TagGen}, \Pi.\mathsf{Dec}, \Pi.\mathsf{Update}, \Pi.\mathsf{UpdateTag}, \Pi.\mathsf{PoWPrf}, \Pi.\mathsf{PoWVer})$ be a *UMLE* scheme. Since, no *UMLE* scheme can provide security for predictable messages, we are modelling the security based on the unpredictable message source $\mathcal{S}(\cdot)$ (for details see Sect. 2.6). According to

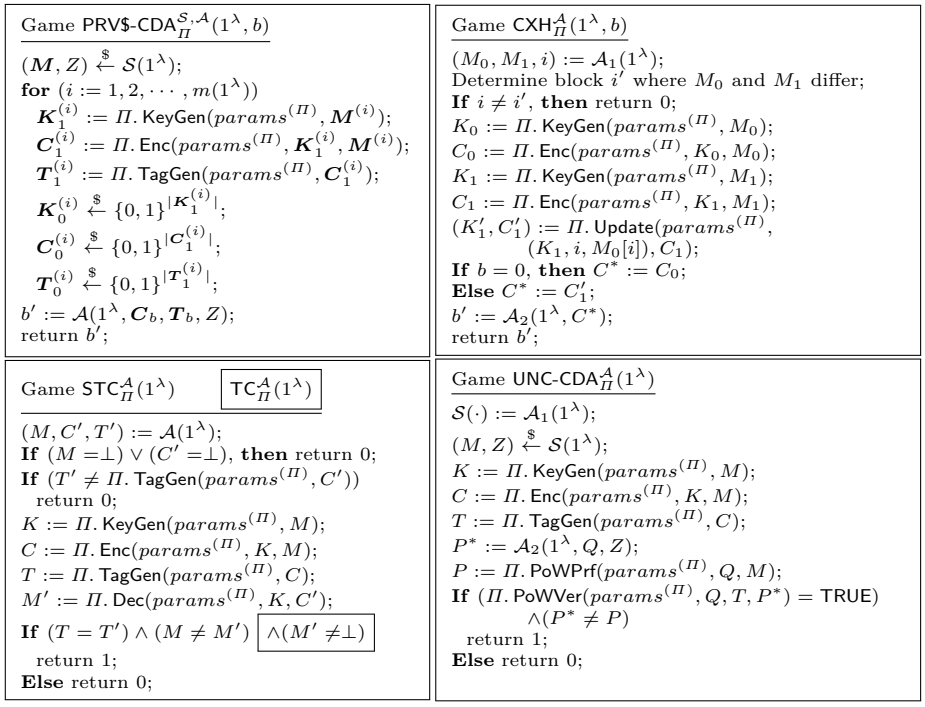


Fig. 2. Games defining PRV$-CDA, STC, TC, CXH and UNC-CDA security of *UMLE* scheme $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TagGen}, \Pi.\mathsf{Dec}, \Pi.\mathsf{Update}, \Pi.\mathsf{UpdateTag}, \Pi.\mathsf{PoWPrf}, \Pi.\mathsf{PoWVer})$. In CXH and UNC-CDA games, adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

the PRV$-CDA game, as in Fig. 2, the challenger gets a vector of messages, $\boldsymbol{M}$ and the auxiliary information Z, from the source $\mathcal{S}(\cdot)$. The challenger does the following operations: computes the decryption key $\boldsymbol{K}_1^{(i)}$, ciphertext $\boldsymbol{C}_1^{(i)}$ and tag $\boldsymbol{T}_1^{(i)}$ for each message string $\boldsymbol{M}^{(i)}$, where $i \in \{1, 2, \cdots, m(1^\lambda)\}$; computes the random strings $\boldsymbol{K}_0^{(i)}$, $\boldsymbol{C}_0^{(i)}$ and $\boldsymbol{T}_0^{(i)}$ of length $|\boldsymbol{K}_1^{(i)}|$, $|\boldsymbol{C}_1^{(i)}|$ and $|\boldsymbol{T}_1^{(i)}|$ respectively; and returns $(\boldsymbol{C}_b, \boldsymbol{T}_b, Z)$ to the adversary. The adversary has to return a bit b' indicating whether the ciphertext $\boldsymbol{C}_b$ and tag $\boldsymbol{T}_b$ corresponds to message $\boldsymbol{M}$ or is it a collection of random strings. If the values of b and b' coincide, then the adversary wins the game.

We define the advantage of a PRV$-CDA adversary $\mathcal{A}$ against Π for the message source $\mathcal{S}(\cdot)$ as:

$$Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Big| \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b = 1) = 1] - \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b = 0) = 1] \Big|.$$

A *UMLE* scheme Π is said to be PRV$-CDA secure over a set of valid PT sources for *UMLE* scheme Π, $\overline{\mathcal{S}} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots\}$, for all PT adversaries $\mathcal{A}$ and

for all $\mathcal{S}_i \in \overline{\mathcal{S}}$, if $Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S}_i,\mathcal{A}}(\cdot)$ is negligible. A *UMLE* scheme Π is said to be $\mathsf{PRV\$\text{-}CDA}$ secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(\cdot)$ is negligible, for all valid PT source $\mathcal{S}$ for Π.

Tag Consistency. Let $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TagGen}, \Pi.\mathsf{Dec}, \Pi.\mathsf{Update}, \Pi.\mathsf{UpdateTag}, \Pi.\mathsf{PoWPrf}, \Pi.\mathsf{PoWVer})$ be a *UMLE* scheme. For a *UMLE* scheme, we have designed the STC and TC security game in Fig. 2, which aims to provide security against duplicate faking attacks. In addition, STC provides guards against erasure attack. In a duplicate faking attack, two unidentical messages – one fake message produced by an adversary and a legitimate one produced by an honest client – produce the same tag, thereby causing loss of message and hampers the integrity. In an erasure attack, the adversary replaces the ciphertext with a fake message that decrypts successfully.

The adversary returns a message M, a ciphertext C' and a tag T'. If the message or ciphertext is invalid, the adversary looses the game. If the tag T' is not computed from C', then also, the adversary loses the game. Otherwise, the challenger computes key K, ciphertext C and tag T corresponding to message M, and computes the message M' corresponding to ciphertext C' using key K. If the two tags are equal, i.e. $T = T'$, the message M' is valid, i.e. $M' \neq \perp$, and the two messages are unequal, i.e. $M \neq M'$, then the adversary wins the TC game.

Now, we define the advantage of a TC adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{TC}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{TC}^{\mathcal{A}}_{\Pi}(1^\lambda) = 1].$$

Now, we define the advantage of an STC adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{STC}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{STC}^{\mathcal{A}}_{\Pi}(1^\lambda) = 1].$$

A *UMLE* scheme Π is said to be TC (or STC) secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{TC}}_{\Pi,\mathcal{A}}(\cdot)$ (or $Adv^{\mathsf{STC}}_{\Pi,\mathcal{A}}(\cdot)$) is negligible.

Context Hiding. Let $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TagGen}, \Pi.\mathsf{Dec}, \Pi.\mathsf{Update}, \Pi.\mathsf{UpdateTag}, \Pi.\mathsf{PoWPrf}, \Pi.\mathsf{PoWVer})$ be a *UMLE* scheme. For a *UMLE*, we have designed the CXH game in Fig. 2, which aims to provide security against distinguishing between an updated ciphertext and a ciphertext encrypted from scratch, to ensure that the level of privacy is not compromised during update process.

According to the CXH game, as in Fig. 2, the adversary returns two messages M_0 and M_1 such that M_0 and M_1 are identical for all bits except block i. The challenger determines the messages block i' where M_0 and M_1 differ, and the adversary loses if the messages M_0 and M_1 differ at any place other than i-th block, i.e. $i \neq i'$. The challenger encrypts the two messages M_0 and M_1 to generate K_0 & C_0 and K_1 & C_1 and updates the i-th block of C_1 with $M_0[i]$ to obtain K'_1 and C'_1. The challenger then sends either C_0 or C'_1, depending on the value of b, to the adversary. The adversary has to return a bit b' indicating

whether the ciphertext is built from scratch or is an updated ciphertext. If the values of b and b' coincide, then the adversary wins the game.

Now, we define the advantage of a CXH adversary $\mathcal{A}$ for 1-block update in message, against Π as:

$$Adv^{\mathsf{CXH}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Big|\Pr[\mathsf{CXH}^{\mathcal{A}}_{\Pi}(1^\lambda, b=1)=1] - \Pr[\mathsf{CXH}^{\mathcal{A}}_{\Pi}(1^\lambda, b=0)=1]\Big|.$$

A *UMLE* scheme Π is said to be CXH secure, for 1-block update in message, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{CXH}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible.

Proof of ownership. Let $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TagGen}, \Pi.\mathsf{Dec}, \Pi.\mathsf{Update}, \Pi.\mathsf{UpdateTag}, \Pi.\mathsf{PoWPrf}, \Pi.\mathsf{PoWVer})$ be a *UMLE* scheme. For a *UMLE*, we have designed the $\mathsf{UNC\text{-}CDA}$ game in Fig. 2, which aims to provide security against the adversary in proving that they possess the entire file when they actually have only a partial information about the file. This is to block the unauthorised ownership of the file.

According to the $\mathsf{UNC\text{-}CDA}$ game, as in Fig. 2, the adversary returns an unpredictable message source $\mathcal{S}(\cdot)$. The challenger gets a message M and the auxiliary information Z, from this source. The challenger then send the challenge Q and the auxiliary information Z to the adversary and the adversary returns a proof P^*. The challenger generates the proof P for the same challenge. If P^* is successfully verified by the PoW verifier algorithm $\Pi.\mathsf{PoWVer}$, i.e. $\Pi.\mathsf{PoWVer}(params^{(\Pi)}, Q, T, P^*) = \mathsf{TRUE}$, and P is different from P^*, i.e. $P^* \neq P$, then the adversary wins the game.

Now, we define the advantage of a $\mathsf{UNC\text{-}CDA}$ adversary $\mathcal{A}$ against an uncheatable chosen distribution attack against Π for a message source source $\mathcal{S}(\cdot)$ as:

$$Adv^{\mathsf{UNC\text{-}CDA}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{UNC\text{-}CDA}^{\mathcal{A}}_{\Pi}(1^\lambda)=1].$$

A *UMLE* scheme Π is said to be $\mathsf{UNC\text{-}CDA}$ secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{UNC\text{-}CDA}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible.

2.9 Hash Function

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. A *hash function* H is an algorithm $\mathsf{H}.\mathcal{H}$ over a PPT setup $\mathsf{H}.\mathsf{Setup}$. H satisfies the following conditions.

1. The PPT setup algorithm $\mathsf{H}.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\mathsf{H})}$ and the sets $\mathcal{M}^{(\mathsf{H})}$ and $\mathcal{T}^{(\mathsf{H})}$ denoting the *message* and *digest spaces* respectively.
2. The deterministic hash algorithm $\mathsf{H}.\mathcal{H}$ takes as inputs $params^{(\mathsf{H})}$ and $M \in \mathcal{M}^{(\mathsf{H})}$, and returns $h := \mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M)$, where $h \in \mathcal{T}^{(\mathsf{H})}$.

Correctness. Let $M, M' \in \mathcal{M}^{(\mathsf{H})}$. Suppose:
- $h := \mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M)$, and
- $h' := \mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M')$.

Then *correctness* of H requires that if $M = M'$, then $h = h'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\mathsf{H})}$.

The security game is written in the form of a challenger-adversary framework.

Game $\mathsf{CRHF}^{\mathcal{A}}_{\mathsf{H}}(1^\lambda)$

$(M_0, M_1) := \mathcal{A}(1^\lambda)$;
If $(\mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M_0) = \mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M_1)) \wedge (M_0 \neq M_1)$
return 1;
Else return 0;

Fig. 3. Game defining CRHF security of Hash Function $\mathsf{H} = \mathsf{H}.\mathcal{H}$.

Collision-Resistance. Let $\mathsf{H} = \mathsf{H}.\mathcal{H}$ be a *hash function*. We design the Collision-Resistance security game in Fig. 3. According to the CRHF game, the adversary sends two messages M_0 and M_1 to the challenger. The challenger checks, if the hash value of the two messages M_0 and M_1 are equal, i.e. $\mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M_0) = \mathsf{H}.\mathcal{H}(params^{(\mathsf{H})}, M_1)$ under $M_0 \neq M_1$.

Now, we define the advantage of a CRHF adversary $\mathcal{A}$ against H as:

$$Adv^{\mathsf{CRHF}}_{\mathsf{H},\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{CRHF}^{\mathcal{A}}_{\mathsf{H}}(1^\lambda) = 1].$$

A hash function H is said to be CRHF secure, for all PPT adversaries $\mathcal{A}$, if $Adv^{\mathsf{CRHF}}_{\mathsf{H},\mathcal{A}}(\cdot)$ is negligible.

2.10 One-time Symmetric Encryption

Syntax. Suppose $\lambda \in \mathbb{N}$ is the security parameter. A *one-time symmetric encryption* scheme $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$ is three-tuple of algorithms over a PPT setup $\mathsf{SE}.\mathsf{Setup}$. SE satisfies the following conditions.

1. The PPT setup algorithm $\mathsf{SE}.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\mathsf{SE})}$ and the sets $\mathcal{K}^{(\mathsf{SE})}$, $\mathcal{M}^{(\mathsf{SE})}$ and $\mathcal{C}^{(\mathsf{SE})}$ denoting the *key*, *message* and *ciphertext* spaces respectively.
2. The PPT key generation algorithm $\mathsf{SE}.\mathcal{GEN}$ takes as inputs $params^{(\mathsf{SE})}$, and returns key $K := \mathsf{SE}.\mathcal{GEN}(params^{(\mathsf{SE})})$, where $K \in \mathcal{K}^{(\mathsf{SE})}$.
3. The encryption $\mathsf{SE}.\mathcal{SE}$ is a deterministic algorithm takes as inputs $params^{(\mathsf{SE})}$, encryption key $K \in \mathcal{K}^{(\mathsf{SE})}$ and the message $M \in \mathcal{M}^{(\mathsf{SE})}$, returns ciphertext $C := \mathsf{SE}.\mathcal{SE}(params^{(\mathsf{SE})}, K, M)$, where $C \in \mathcal{C}^{(\mathsf{SE})}$. In order to make this scheme *one-time* symmetric encryption, each execution of $\mathsf{SE}.\mathcal{SE}(\cdot)$ requires that the key be freshly generated using $\mathsf{SE}.\mathcal{GEN}(\cdot)$.
4. The decryption $\mathsf{SE}.\mathcal{SD}$ is a deterministic algorithm that takes as inputs $params^{(\mathsf{SE})}$, key $K \in \mathcal{K}^{(\mathsf{SE})}$ and ciphertext $C \in \mathcal{C}^{(\mathsf{SE})}$, and returns the message $M := \mathsf{SE}.\mathcal{SD}(params^{(\mathsf{SE})}, K, C)$, where $M \in \mathcal{M}^{(\mathsf{SE})}$.

5. We restrict $|C|$ to be a linear function of $|M|$.

CORRECTNESS. Let $M \in \mathcal{M}^{(\mathsf{SE})}$. Suppose:
- $K := \mathsf{SE}.\mathcal{GEN}(params^{(\mathsf{SE})})$, and
- $C := \mathsf{SE}.\mathcal{SE}(params^{(\mathsf{SE})}, K, M)$.

Then correctness of SE requires that $\mathsf{SE}.\mathcal{SD}(params^{(\mathsf{SE})}, K, C) = M$, for all $\lambda \in \mathbb{N}$ and all $M \in \mathcal{M}^{(\mathsf{SE})}$.

The two security games are written in the form of a challenger-adversary framework.

Game $\mathsf{KR}^{\mathcal{A}}_{\mathsf{SE}}(1^\lambda)$	Game $\mathsf{IND\text{-}PRV}^{\mathcal{A}}_{\mathsf{SE}}(1^\lambda, b)$
$M := \mathcal{A}_1(1^\lambda)$; $K := \mathsf{SE}.\mathcal{GEN}(params^{(\mathsf{SE})})$; $C := \mathsf{SE}.\mathcal{SE}(params^{(\mathsf{SE})}, K, M)$; $K' := \mathcal{A}_2(1^\lambda, C)$; **If** $K' = K$, **then** return 1; **Else** return 0;	$M := \mathcal{A}_1(1^\lambda)$; $K := \mathsf{SE}.\mathcal{GEN}(params^{(\mathsf{SE})})$; $C_0 := \mathsf{SE}.\mathcal{SE}(params^{(\mathsf{SE})}, K, M)$; $C_1 \xleftarrow{\$} \{0,1\}^{\lvert C_0 \rvert}$; $b' := \mathcal{A}_2(1^\lambda, C_b)$; return b';

Fig. 4. Game defining KR and $\mathsf{IND\text{-}PRV}$ security of one-time symmetric encryption $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$. Here, $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

KEY RECOVERY. Let $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$ be a one-time symmetric encryption scheme. We design Key-Recovery security game in Fig. 4. According to the KR game, the adversary sends a message M to the challenger. The challenger generates a key $K := \mathsf{SE}.\mathcal{GEN}(params^{(\mathsf{SE})})$, encrypts M using K to obtain the ciphertext $C := \mathsf{SE}.\mathcal{SE}(params^{(\mathsf{SE})}, K, M)$ and sends C to the adversary. The adversary has to return a key K' as the encryption key of M that resulted into C. If the values of K and K' coincide, i.e. $K = K'$, then the adversary wins the game.

Now, we define the advantage of a KR adversary $\mathcal{A}$ against SE as:

$$Adv^{\mathsf{KR}}_{\mathsf{SE},\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{KR}^{\mathcal{A}}_{\mathsf{SE}}(1^\lambda) = 1].$$

A one-time symmetric encryption scheme SE is said to be KR secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{KR}}_{\mathsf{SE},\mathcal{A}}(\cdot)$ is negligible.

PRIVACY. Let $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$ be a one-time symmetric encryption scheme. We design the Privacy security game in Fig. 4. According to the $\mathsf{IND\text{-}PRV}$ game, the adversary sends a message M to the challenger. The challenger generates a key $K := \mathsf{SE}.\mathcal{GEN}(params^{(\mathsf{SE})})$, encrypts M using K to obtain the ciphertext $C_0 := \mathsf{SE}.\mathcal{SE}(params^{(\mathsf{SE})}, K, M)$, computes the random string C_1 of length $|C_0|$, and sends C_b to the adversary. The adversary has to

return a bit b' indicating whether the ciphertext C_b corresponds to message M or is it a random string. If the values of b and b' coincide, then the adversary wins the game.

Now, we define the advantage of a IND-PRV adversary $\mathcal{A}$ against SE as:

$$Adv^{\mathsf{IND\text{-}PRV}}_{\mathsf{SE},\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Big|\Pr[\mathsf{IND\text{-}PRV}^{\mathcal{A}}_{\mathsf{SE}}(1^\lambda, b=1)=1] - \Pr[\mathsf{IND\text{-}PRV}^{\mathcal{A}}_{\mathsf{SE}}(1^\lambda, b=0)=1]\Big|.$$

A one-time symmetric encryption scheme SE is said to be IND-PRV secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{IND\text{-}PRV}}_{\mathsf{SE},\mathcal{A}}(\cdot)$ is negligible.

2.11 Sponge hash function

The pictorial and algorithmic descriptions of Sponge construction [3, 8–12] are given in Figs. 5 and 6; all wires are λ-bit long. Let M denote the message to be hashed, $M[i]$ denote the i-th block of message. Below we describe the algorithm $\mathsf{H}.\mathcal{H}$ for Sponge hash function in detail.

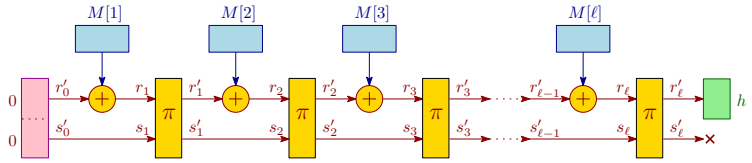


Fig. 5. Diagrammatic description of Sponge hash function $h := \mathsf{H}.\mathcal{H}(1^\lambda, M)$, where $M := M[1]||M[2]||\cdots||M[\ell]$, and $M[1], M[2], \cdots M[\ell]$ are the λ-bit message blocks.

$\underline{\mathsf{H}.\mathcal{H}(1^\lambda, M)}$
$\ell := |M|/\lambda$, $r'_0 := 0^\lambda$, $s'_0 := 0^\lambda$;
$M[1]||M[2]||\cdots||M[\ell] := M$;
for $(j := 1, 2, \cdots, \ell)$
  $r_j := M[j] \oplus r'_{j-1}$, $s_j := s'_{j-1}$;
  $r'_j||s'_j := \pi(r_j||s_j)$;
$h := r'_\ell$;
return h;

Fig. 6. Algorithmic description of Sponge hash function $h := \mathsf{H}.\mathcal{H}(1^\lambda, M)$.

Hash $\mathsf{H}.\mathcal{H}$. Fig. 5 shows the hashing of message M. The hash takes the parameter 1^λ and message M and breaks it into several λ-bit blocks, namely, $M[1], M[2], \cdots, M[\ell]$. Hash of M is composed of hash of individual blocks in

sequence. Two variables r'_0 and s'_0 are assigned 0^λ. Now we give the details of how the message block $M[j]$, for $j := 1, 2, \cdots \ell$ is hashed: r_j is assigned the XOR of $M[j]$ and r'_{j-1}; s_j is assigned the value of s'_{j-1}; and we compute $(r'_j||s'_j)$ as $\pi(r_j||s_j)$. We assign r'_ℓ as the hash h.

Security of Sponge construction Sponge construction is already proven to be CRHF secure. CRHF game is given in Fig. 3.

3 *FMLE*: A New Cryptographic Primitive

The *File-updatable Message-Locked Encryption (FMLE)* is a generalisation of *Efficiently Updatable Block-Level Message-Locked Encryption (UMLE)* as given by Zhao and Chow[20]. The difference between the definitions of *UMLE* and *FMLE* is that the former requires the existence of a *BL-MLE* scheme, while the latter does not.[2] Therefore, any *UMLE* scheme can be viewed as an *FMLE* scheme too, not the other way round.

Below we elaborately discuss the syntax, correctness and security definition of the new notion *FMLE*.

3.1 Syntax

Suppose $\lambda \in \mathbb{N}$ is the security parameter. An *FMLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$ is five-tuple of algorithms over a PPT setup $\Pi.\mathsf{Setup}$. Π satisfies the following conditions.

1. The PPT setup algorithm $\Pi.\mathsf{Setup}(1^\lambda)$ outputs the parameter $params^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$, $\mathcal{M}^{(\Pi)}$, $\mathcal{C}^{(\Pi)}$ and $\mathcal{T}^{(\Pi)}$, denoting the *key*, *message*, *ciphertext* and *tag spaces* respectively.
2. The PPT encryption algorithm $\Pi.\mathcal{E}$ takes as inputs $params^{(\Pi)}$ and $M \in \mathcal{M}^{(\Pi)}$, and returns a 3-tuple $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, where $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$.
3. The decryption algorithm $\Pi.\mathcal{D}$ is a deterministic algorithm that takes as inputs $params^{(\Pi)}$, $K \in \mathcal{K}^{(\Pi)}$, $C \in \mathcal{C}^{(\Pi)}$ and $T \in \mathcal{T}^{(\Pi)}$, and returns $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) \in \mathcal{M}^{(\Pi)} \cup \{\perp\}$. The decryption algorithm $\Pi.\mathcal{D}$ returns $\perp$ if the key K, ciphertext C and tag T are not generated from a valid message.
4. The PPT update algorithm $\Pi.\mathcal{U}$ takes as inputs $params^{(\Pi)}$, the index of starting and ending bits i_{st} and i_{end}, new message bits $M_{\text{new}} \in \mathcal{M}^{(\Pi)}$, the decryption key $K \in \mathcal{K}^{(\Pi)}$, the ciphertext to be updated $C \in \mathcal{C}^{(\Pi)}$, the tag to be updated $T \in \mathcal{T}^{(\Pi)}$ and the bit $app \in \{0, 1\}$ indicating change in length of new message, and returns a 3-tuple $(K', C', T') := \Pi.\mathcal{U}(params^{(\Pi)}, i_{st}, i_{end}, M_{\text{new}}, K, C, T, app)$, where $K' \in \mathcal{K}^{(\Pi)}$, $C' \in \mathcal{C}^{(\Pi)}$ and $T' \in \mathcal{T}^{(\Pi)}$.

[2] A *block-level message-locked encryption (BL-MLE)* is an *MLE* that works on the fixed-sized messages, called *blocks*.

5. The PPT proof-of-ownership (PoW) algorithm for prover $\Pi.\mathcal{P}$ takes as inputs parameter $params^{(\Pi)}$, challenge Q, a file $M \in \mathcal{M}^{(\Pi)}$, the decryption key $K \in \mathcal{K}^{(\Pi)}$, the ciphertext $C \in \mathcal{C}^{(\Pi)}$, the tag $T \in \mathcal{T}^{(\Pi)}$, and returns the proof $P := \Pi.\mathcal{P}(params^{(\Pi)}, Q, M, K, C, T)$.
6. The PPT proof-of-ownership (PoW) algorithm for verifier $\Pi.\mathcal{V}$ takes as inputs parameter $params^{(\Pi)}$, challenge Q, ciphertext $C \in \mathcal{C}^{(\Pi)}$, tag $T \in \mathcal{T}^{(\Pi)}$ and proof P, and returns the value $val := \Pi.\mathcal{V}(params^{(\Pi)}, Q, C, T, P)$, where $val \in \{\mathsf{TRUE}, \mathsf{FALSE}\}$.
7. We restrict $|C|$ to be a linear function of $|M|$.

3.2 Correctness

Key Correctness. Let $M, M' \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{E}(params^{(\Pi)}, M')$.

Then *key correctness* of Π requires that if $M = M'$, then $K = K'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

Decryption Correctness. Let $M \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$.

Then *decryption correctness* of Π requires that $\Pi.\mathcal{D}(params^{(\Pi)}, K, C, T) = M$, for all $\lambda \in \mathbb{N}$ and all $M \in \mathcal{M}^{(\Pi)}$.

Tag Correctness. Let $M, M' \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{E}(params^{(\Pi)}, M')$.

Then *tag correctness* of Π requires that if $M = M'$, then $T = T'$, for all $\lambda \in \mathbb{N}$ and all $M, M' \in \mathcal{M}^{(\Pi)}$.

Update Correctness. Let $M \in \mathcal{M}^{(\Pi)}$. Suppose:
- $\ell = |M|$,
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$, and
- $(K', C', T') := \Pi.\mathcal{U}(params^{(\Pi)}, i_{st}, i_{end}, M_{\text{new}}, K, C, T, app)$.

Then *update correctness* of Π requires that, for all $\lambda \in \mathbb{N}$, all $M \in \mathcal{M}^{(\Pi)}$, $1 \leq i_{st} \leq \ell$ and $i_{st} < i_{end}$:
• for $app = 1$, $\Pi.\mathcal{D}(params^{(\Pi)}, K', C', T') = M[1] \,||M[2]\, ||\cdots||\, M[i_{st} - 1] \,||M_{\text{new}}$, and
• for $app = 0$, $\Pi.\mathcal{D}(params^{(\Pi)}, K', C', T') = M[1] \,||M[2]\, ||\cdots||\, M[i_{st} - 1] \,||M_{\text{new}}\, ||M[i_{end} + 1]||M[i_{end} + 2]||\cdots||M[\ell]$.

PoW Correctness. Let $M \in \mathcal{M}^{(\Pi)}$. Suppose:
- $(K, C, T) := \Pi.\mathcal{E}(params^{(\Pi)}, M)$,
- Q is any challenge, and
- $P := \Pi.\mathcal{P}(params^{(\Pi)}, Q, M, K, C, T)$.

Then *PoW correctness* of Π requires that $\Pr[\Pi.\mathcal{V}(params^{(\Pi)}, Q, C, T, P) = \mathsf{TRUE}] = 1$, for all $\lambda \in \mathbb{N}$ and all $M \in \mathcal{M}^{(\Pi)}$.

3.3 Security Definitions

Security definitions of *FMLE* are naturally adapted from those of *UMLE*. For the sake of completeness, we describe them below in full detail. As usual, all the games are written in the form of challenger-adversary framework.

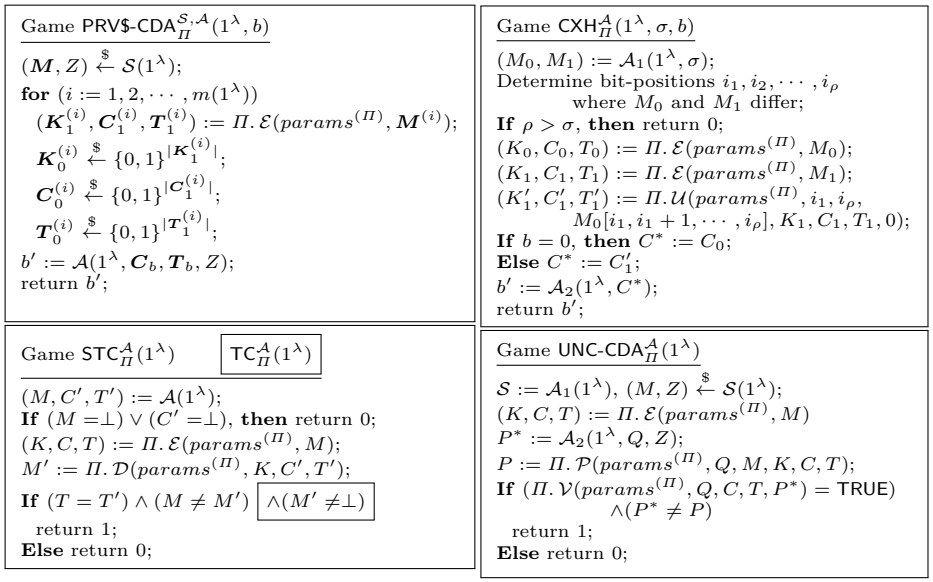


Fig. 7. Games defining PRV$-CDA, STC, TC, CXH and UNC-CDA security of *FMLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$. In CXH and UNC-CDA games, adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

Privacy. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$ be an *FMLE* scheme. Since, no *FMLE* scheme can provide security for predictable messages, we are modelling the security based on the unpredictable message source $\mathcal{S}(\cdot)$. According to the PRV$-CDA game, as in Fig. 7, the challenger gets a vector of messages, $\boldsymbol{M}$ and the auxiliary information Z, from the source $\mathcal{S}(\cdot)$. The challenger does the following operations: computes the decryption key $\boldsymbol{K}_1^{(i)}$, ciphertext $\boldsymbol{C}_1^{(i)}$ and tag $\boldsymbol{T}_1^{(i)}$ for each message string $\boldsymbol{M}^{(i)}$ using $\Pi.\mathcal{E}$, where $i \in \{1, 2, \cdots, m(1^\lambda)\}$; computes the random strings $\boldsymbol{K}_0^{(i)}$, $\boldsymbol{C}_0^{(i)}$ and $\boldsymbol{T}_0^{(i)}$ of length $|\boldsymbol{K}_1^{(i)}|$, $|\boldsymbol{C}_1^{(i)}|$ and $|\boldsymbol{T}_1^{(i)}|$ respectively; and returns $(\boldsymbol{C}_b, \boldsymbol{T}_b, Z)$ to the adversary. The adversary has to return a bit b' indicating whether the ciphertext $\boldsymbol{C}_b$ and tag $\boldsymbol{T}_b$ corresponds to message $\boldsymbol{M}$ or is it a collection of random strings. If the values of b and b' coincide, then the adversary wins the game.

We define the advantage of an $\mathsf{PRV\$\text{-}CDA}$ adversary $\mathcal{A}$ against Π for the message source $\mathcal{S}(\cdot)$ as:

$$Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Big| \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b=1)=1] - \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b=0)=1]\Big|.$$

An *FMLE* scheme Π is said to be $\mathsf{PRV\$\text{-}CDA}$ secure over a set of valid PT sources for *FMLE* scheme Π, $\overline{\mathcal{S}} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots\}$, for all PT adversaries $\mathcal{A}$ and for all $\mathcal{S}_i \in \overline{\mathcal{S}}$, if $Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S}_i,\mathcal{A}}(\cdot)$ is negligible. An *FMLE* scheme Π is said to be $\mathsf{PRV\$\text{-}CDA}$ secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{PRV\$\text{-}CDA}}_{\Pi,\mathcal{S},\mathcal{A}}(\cdot)$ is negligible, for all valid PT source $\mathcal{S}$ for Π.

Tag Consistency. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$ be an *FMLE* scheme. For an *FMLE* scheme, we have designed the STC and TC security games in Fig. 7, which aim to provide security against duplicate faking attacks. In addition, STC provides safeguards against erasure attack. In a duplicate faking attack, two unidentical messages – one fake message produced by an adversary and a legitimate one produced by an honest client – produce the same tag, thereby cause loss of message and hamper the integrity. In an erasure attack, the adversary replaces the ciphertext with a fake message that decrypts successfully.

The adversary returns a message M, a ciphertext C' and a tag T'. If the message or ciphertext is invalid, the adversary loses the game. Otherwise, the challenger computes decryption key K, ciphertext C and tag T corresponding to message M, and computes the message M' corresponding to ciphertext C' and tag T' using key K. If the two tags are equal, i.e. $T = T'$, the message M' is valid, i.e. $M' \neq \perp$, and the two messages are unequal, i.e. $M \neq M'$, then the adversary wins the TC game.

Now, we define the advantage of a TC adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{TC}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{TC}^{\mathcal{A}}_{\Pi}(1^\lambda)=1].$$

Now, we define the advantage of an STC adversary $\mathcal{A}$ against Π as:

$$Adv^{\mathsf{STC}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{STC}^{\mathcal{A}}_{\Pi}(1^\lambda)=1].$$

An *FMLE* scheme Π is said to be TC (or STC) secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{TC}}_{\Pi,\mathcal{A}}(\cdot)$ (or $Adv^{\mathsf{STC}}_{\Pi,\mathcal{A}}(\cdot)$) is negligible.

Context Hiding. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$ be an *FMLE* scheme. For an *FMLE*, we have designed the CXH game in Fig. 7, which aims to provide security against distinguishing between an updated ciphertext and a ciphertext encrypted from scratch, to ensure that the level of privacy is not compromised during update process.

According to the CXH game, as in Fig. 7, the adversary returns two messages M_0 and M_1 such that M_0 and M_1 are identical for all bits except σ bits. The challenger calculates the bit-positions $i_1, i_2, \cdots, i_\rho$ where M_0 and M_1 differ, and

the adversary loses if $\rho > \sigma$. The challenger encrypts the two messages M_0 and M_1 to generate (K_0, C_0, T_0) and (K_1, C_1, T_1) and updates the C_1 with $M_0[i_1, i_1+1, \cdots, i_\rho]$ to obtain (K_1', C_1', T_1'). The challenger then sends either C_0 or C_1', depending on the value of b, to the adversary. The adversary has to return a bit b' indicating whether the ciphertext is built from scratch or is an updated ciphertext. If the values of b and b' are equal, then the adversary wins the game.

Now, we define the advantage of a CXH adversary $\mathcal{A}$ for σ-bit update in message, against Π as:

$$Adv^{\mathsf{CXH}}_{\Pi,\mathcal{A}}(1^\lambda, \sigma) \stackrel{def}{=} \Big| \Pr[\mathsf{CXH}^{\mathcal{A}}_{\Pi}(1^\lambda, \sigma, b = 1) = 1] - \Pr[\mathsf{CXH}^{\mathcal{A}}_{\Pi}(1^\lambda, \sigma, b = 0) = 1] \Big|.$$

An *FMLE* scheme Π is said to be CXH secure, for σ-bit update in message, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{CXH}}_{\Pi,\mathcal{A}}(\cdot, \cdot)$ is negligible.

Proof of ownership. Let $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$ be an *FMLE* scheme. For an *FMLE*, we have designed the $\mathsf{UNC\text{-}CDA}$ game in Fig. 7, which aims to provide security against the adversary in proving that they possess the entire file when they actually have only a partial information about the file. This is to block the unauthorised ownership of the file.

According to the $\mathsf{UNC\text{-}CDA}$ game, as in Fig. 7, the adversary returns an unpredictable message source $\mathcal{S}(\cdot)$. The challenger gets a message M and the auxiliary information Z, from this source. The challenger then send the challenge Q and the auxiliary information Z to the adversary and the adversary returns a proof P^*. The challenger generates the proof P for the same challenge. If P^* is successfully verified by the PoW verifier algorithm $\Pi.\mathcal{V}$ and P is different from P^*, then the adversary wins the game.

Now, we define the advantage of a $\mathsf{UNC\text{-}CDA}$ adversary $\mathcal{A}$ against an uncheatable chosen distribution attack against Π for a message source source $\mathcal{S}(\cdot)$ as:

$$Adv^{\mathsf{UNC\text{-}CDA}}_{\Pi,\mathcal{A}}(1^\lambda) \stackrel{def}{=} \Pr[\mathsf{UNC\text{-}CDA}^{\mathcal{A}}_{\Pi}(1^\lambda) = 1].$$

An *FMLE* scheme Π is said to be $\mathsf{UNC\text{-}CDA}$ secure, for all PT adversaries $\mathcal{A}$, if $Adv^{\mathsf{UNC\text{-}CDA}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible.

4 Deduplication: An Application of *FMLE*

Deduplication is a mechanism by which a protocol removes the requirement for storing multiple copies of an identical file in memory. This is highly beneficial for the better utilization of space in the *cloud*, where multiple users often store identical files. Loosely speaking, it does so, by identifying the identical files, removing all the copies except one, and then attaching a special file called the *list of owners* to it. In Fig. 8, we give the details of the deduplication protocol supporting *file-update* and *proof of ownership (PoW)* functionalities, implemented using *FMLE* scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D}, \Pi.\mathcal{U}, \Pi.\mathcal{P}, \Pi.\mathcal{V})$. Although, intuitively clear, we would like to point out that the *authentication of users* is not a part of this

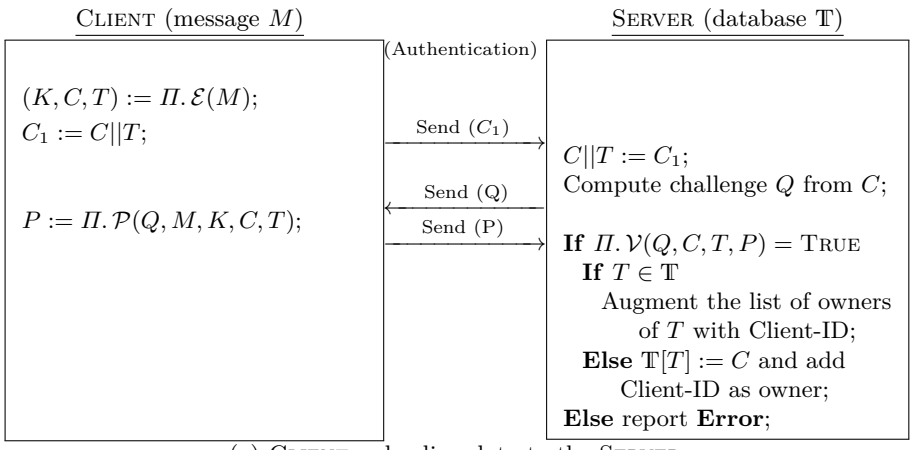
protocol; the system, otherwise, takes care of that through various well-known means such as password-based/bio-metric authentications, etc.

The deduplication protocol with file-update and PoW functionalities, has three functions: client uploading data to the server, as shown in Fig. 8(a); client downloading data from the server, as shown in Fig. 8(b); and client updating data to the server, as shown in Fig. 8(c). Below we give the textual description of the three functionalities of the deduplication protocol.

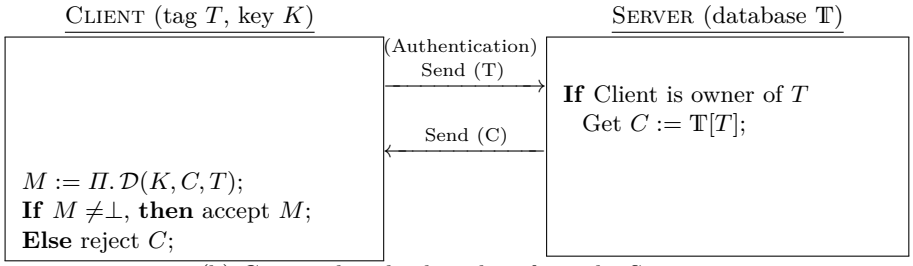
Uploading. For uploading data to the server, the client authenticates its identity to the server, calculates the key K, ciphertext C and tag T using the encryption function $\Pi.\mathcal{E}$ on message M, and sends $C_1 := C||T$ to the server. The server computes tag T and ciphertext C from C_1, generates a challenge Q and sends Q to the client. The client generates and sends to the server, the proof P based on the challenge Q, message M, key K, ciphertext C and tag T using $\Pi.\mathcal{P}$. The server verifies if the proof is correct or not using $\Pi.\mathcal{V}$ function and if it is correctly verified, then stores the (C, T) pair in the database $\mathbb{T}$ along with the Client-ID as owner. If the (C, T) pair already exists in the database, then only the owner information is appended.

Downloading. For downloading data from the server, the client authenticates its identity to the server and sends tag T corresponding to the required ciphertext, to the server. Then, the server checks if the client is the owner of ciphertext corresponding to tag T in the database $\mathbb{T}$ or not, and on successful verification, sends ciphertext C. The client decrypts the C using the decryption function $\Pi.\mathcal{D}$ with key K and tag T, to obtain message M. If the message is valid, i.e. if $M \neq \perp$, then the message is accepted, otherwise the ciphertext C is rejected.

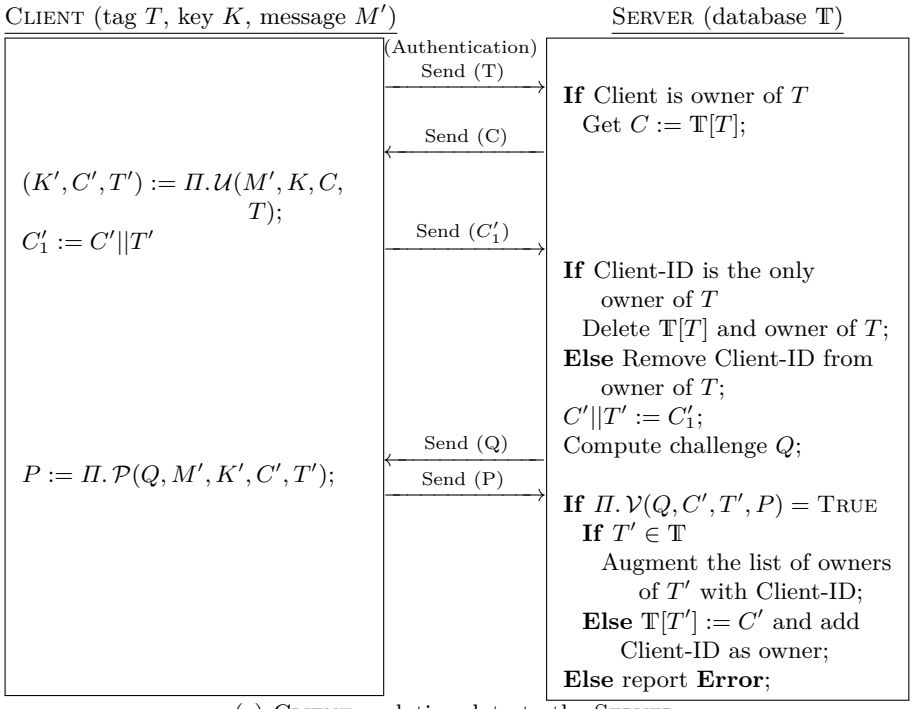
Updating. For updating data to the server, the client authenticates its identity to the server and sends tag T corresponding to the required ciphertext, to the server. Then, the server checks if the client is the owner of ciphertext corresponding to tag T in the database $\mathbb{T}$ or not, and on successful verification, sends ciphertext C. The client updates the ciphertext using the new message M' and the key K, ciphertext C and tag T, to obtain the new key K', ciphertext C' and tag T' and sends $C'_1 := C'||T'$ to the server. The server removes the ownership of the client from the ciphertext C, computes a challenge Q corresponding to C'_1 and sends Q to the client. The client generates and sends to the server, the proof P based on the challenge Q, message M', key K', ciphertext C' and tag T' using $\Pi.\mathcal{P}$. The server verifies if the proof is correct or not using $\Pi.\mathcal{V}$ function and if it is correctly verified, then stores the (C', T') pair in the database $\mathbb{T}$ along with the Client-ID as owner. If the (C', T') pair already exists in the database, then, only the owner information is appended.



(a) Client uploading data to the Server.



(b) Client downloading data from the Server.



(c) Client updating data to the Server.

Fig. 8. Upload, Download and Update Protocols in the Deduplication Protocol.

5 Practical *FMLE* Constructions from existing *MLE* schemes

In this section, we give three *FMLE* schemes built by augmenting the existing *MLE* schemes.

Ingredients. All three *FMLE* schemes Π of this section are built from a one-time symmetric encryption $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$, and a hash function family $\mathsf{H} = (\mathsf{H}.\mathcal{H})$, as discussed in Sects. 2.10 and 2.9 respectively. Here, we note that the setup function of SE, H and Π are compatible such that:

- $\mathcal{M}^{(\mathsf{H})} = \mathcal{K}^{(\mathsf{SE})} = \mathcal{M}^{(\mathsf{SE})} = \mathcal{C}^{(\mathsf{SE})}$
- $\mathcal{K}^{(\Pi)} = \mathcal{T}^{(\mathsf{H})} = \mathcal{K}^{(\mathsf{SE})}$
- $\mathcal{M}^{(\Pi)} = \mathcal{M}^{(\mathsf{H})} = \mathcal{M}^{(\mathsf{SE})}$
- $\mathcal{C}^{(\Pi)} = \mathcal{M}^{(\mathsf{H})} = \mathcal{C}^{(\mathsf{SE})}$
- $\mathcal{T}^{(\Pi)} = \mathcal{T}^{(\mathsf{H})}$

As defined in Sect. 3, any *FMLE* is associated with an implicit setup algorithm that outputs $params^{(\Pi)}$ and the sets $\mathcal{K}^{(\Pi)}$, $\mathcal{M}^{(\Pi)}$, $\mathcal{C}^{(\Pi)}$ and $\mathcal{T}^{(\Pi)}$. Construction is all about clearly specifying the five algorithms namely $\Pi.\mathcal{E}$, $\Pi.\mathcal{D}$, $\Pi.\mathcal{U}$, $\Pi.\mathcal{P}$ and $\Pi.\mathcal{V}$. All five algorithms have, as input, $params^{(\Pi)}$ given by the associated setup algorithm. The setup function $\Pi.\mathsf{Setup}$ is designed in such a way that it invokes $\mathsf{H}.\mathsf{Setup}$ and $\mathsf{SE}.\mathsf{Setup}$, to generate $params^{(\mathsf{H})}$ and $params^{(\mathsf{SE})}$, and the output of $\Pi.\mathsf{Setup}$ is $params^{(\Pi)} = (params^{(\mathsf{H})}, params^{(\mathsf{SE})})$.

5.1 F-CE

The *MLE* construction CE, proposed by Douceur et al.[15], that uses the hash function $\mathsf{H} = (\mathsf{H}.\mathcal{H})$ and the symmetric-key algorithm $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$, can be converted into an *FMLE* scheme, denoted $\mathsf{F\text{-}CE}$, by adding three algorithms $\mathsf{F\text{-}CE}.\mathcal{U}$, $\mathsf{F\text{-}CE}.\mathcal{P}$ and $\mathsf{F\text{-}CE}.\mathcal{V}$ to the existing algorithms $\mathsf{F\text{-}CE}.\mathcal{E}$ and $\mathsf{F\text{-}CE}.\mathcal{D}$, as shown in Fig. 9.

5.2 F-HCE2

The *MLE* construction $\mathsf{HCE2}$, proposed by Bellare, Keelveedhi and Ristenpart [6], that uses the hash function $\mathsf{H} = (\mathsf{H}.\mathcal{H})$ and the symmetric-key algorithm $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$, can be modified into an *FMLE* scheme $\mathsf{F\text{-}HCE2}$ as shown in Fig. 10.

5.3 F-RCE

The *MLE* construction RCE, proposed by Bellare, Keelveedhi and Ristenpart [6], that uses the hash function $\mathsf{H} = (\mathsf{H}.\mathcal{H})$ and the symmetric-key algorithm $\mathsf{SE} = (\mathsf{SE}.\mathcal{GEN}, \mathsf{SE}.\mathcal{SE}, \mathsf{SE}.\mathcal{SD})$, can be modified into an *FMLE* scheme $\mathsf{F\text{-}RCE}$ as shown in Fig. 11.

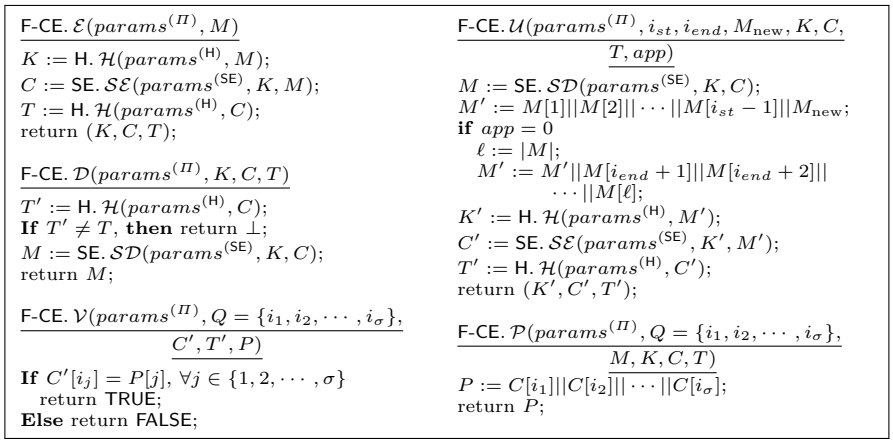


Fig. 9. Algorithmic Description of the *FMLE* scheme F-CE. Here, $params^{(\Pi)} = (params^{(\mathsf{H})}, params^{(\mathsf{SE})})$.

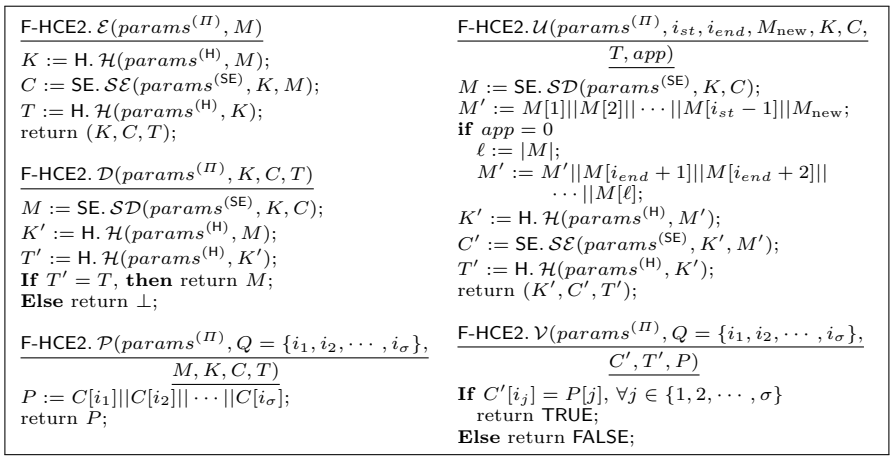


Fig. 10. Algorithmic Description of the *FMLE* scheme F-HCE2. Here, $params^{(\Pi)} = (params^{(\mathsf{H})}, params^{(\mathsf{SE})})$.

5.4 Security of F-CE, F-HCE2 and F-RCE

The F-CE, F-HCE2 and F-RCE schemes are constructed by adding functionalities to the CE, HCE2 and RCE schemes respectively. Therefore, if the CE, HCE2 and RCE schemes are PRV$-CDA, STC and TC secure, so are the F-CE, F-HCE2 and F-RCE schemes.

The CXH and UNC-CDA security of the schemes F-CE and F-HCE2 follow from the fact that they are deterministic.

The CXH security of the scheme F-RCE follows from the fact that the new key is generated for the updated message and the whole updated message is re-

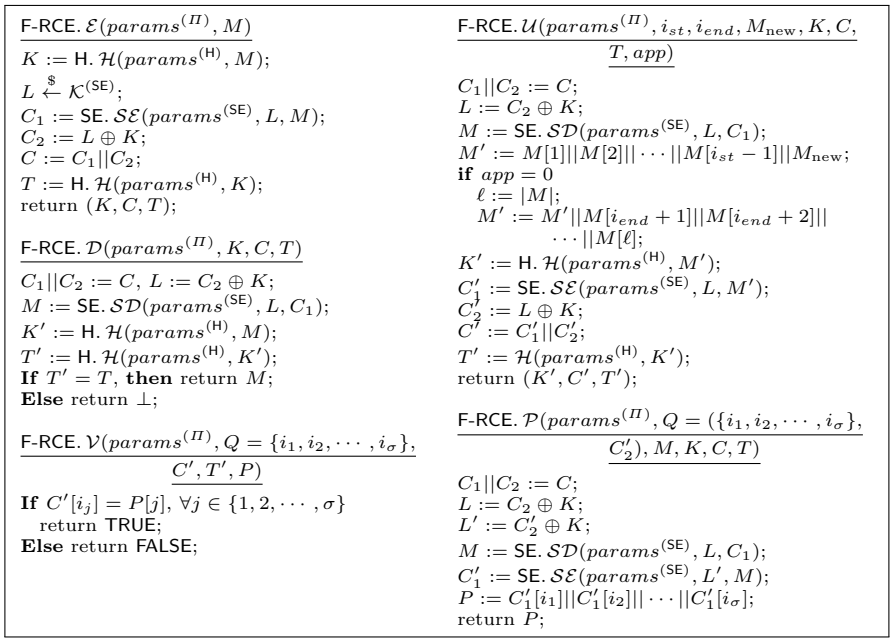


Fig. 11. Algorithmic Description of the *FMLE* scheme F-RCE. Here, $params^{(\Pi)} = (params^{(\mathsf{H})}, params^{(\mathsf{SE})})$.

encrypted (as described in the Fig. 11), therefore, the advantage of the adversary is zero.

The UNC-CDA security is achieved in the F-RCE scheme because the proof P^* produced by the adversary has to be identical to the real proof P, which already stored as the ciphertext in the cloud, because the verification part is just the equality check.

6 Practical FMLE Construction from existing UMLE scheme

In this section, we give one *FMLE* scheme built by augmenting the existing *UMLE* scheme.

6.1 The F-UMLE scheme

The F-UMLE scheme Π is constructed, as shown in Fig. 12, from a *UMLE* scheme $\pi = (\pi.\mathsf{KeyGen}, \pi.\mathsf{Enc}, \pi.\mathsf{TagGen}, \pi.\mathsf{Dec}, \pi.\mathsf{Update}, \pi.\mathsf{UpdateTag}, \pi.\mathsf{PoWPrf}, \pi.\mathsf{PoWVer})$, defined over the setup algorithm $\pi.\mathsf{Setup}$ that outputs parameters $params^{(\pi)}$ and the sets $\mathcal{K}^{(\pi)}$, $\mathcal{M}^{(\pi)}$, $\mathcal{C}^{(\pi)}$ and $\mathcal{T}^{(\pi)}$. Here, we note that the setup function $\Pi.\mathsf{Setup}$ is designed in such a way that it invokes $\pi.\mathsf{Setup}$, to generate

$params^{(\pi)}$, and the output of $\Pi.\mathsf{Setup}$ is $params^{(\Pi)} = params^{(\pi)}$. Also the sets generated by $\Pi.\mathsf{Setup}$ and $\pi.\mathsf{Setup}$ are compatible such that:

- $\mathcal{K}^{(\Pi)} = \mathcal{K}^{(\pi)}$
- $\mathcal{M}^{(\Pi)} = \mathcal{M}^{(\pi)}$
- $\mathcal{C}^{(\Pi)} = \mathcal{C}^{(\pi)}$
- $\mathcal{T}^{(\Pi)} = \mathcal{T}^{(\pi)}$

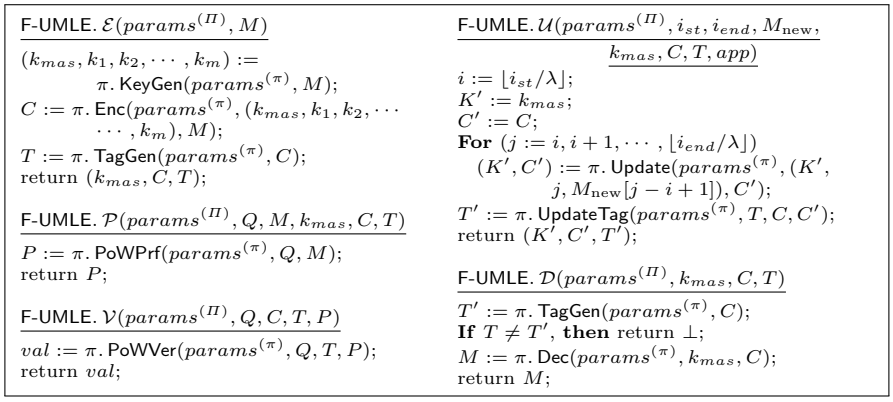


Fig. 12. Algorithmic Description of the *FMLE* scheme F-UMLE. Here, $params^{(\Pi)} = params^{(\pi)}$

6.2 Security of F-UMLE scheme

The F-UMLE scheme is a *UMLE* as well as an *FMLE* scheme. Since, the security targets of *FMLE* and *UMLE* are identical, and F-UMLE satisfies the *UMLE* security targets, hence, F-UMLE is a secure *FMLE* scheme.

7 New Efficient *FMLE* Schemes

In this section we present the two new efficient constructions for *FMLE* – namely RevD-1 and RevD-2 – which are based on a 2λ-bit easy-to-invert permutation π. We assume that the length of message is a multiple of λ; λ is the security parameter.

7.1 The RevD-1 scheme

We describe our first *FMLE* scheme, namely, RevD-1. This construction is motivated by the design of the hash function mode of operation *FP* [18].

Description of RevD-1 The pictorial and algorithmic descriptions are given in Figs. 13 and 14; all wires are λ-bit long. Let M denote the message to be encrypted, $M[i]$ denote the i-th block of message, and $M[i][j]$ denote the j-th bit of i-th block of message. It is worth noting that the decryption is executed in the reverse direction of encryption. Below, we describe the 5-tuple of algorithms $(\mathsf{RevD\text{-}1}.\mathcal{E}, \mathsf{RevD\text{-}1}.\mathcal{D}, \mathsf{RevD\text{-}1}.\mathcal{U}, \mathsf{RevD\text{-}1}.\mathcal{P}, \mathsf{RevD\text{-}1}.\mathcal{V})$ for RevD-1, in detail.

Encryption RevD-1.$\mathcal{E}$. Fig. 13(a) shows the encryption of message M. The encryption takes the parameter 1^λ and message M and breaks it into several λ-bit blocks, namely, $M[1], M[2], \cdots, M[\ell]$. Encryption of M is composed of encryptions of individual blocks in sequence. Two variables s_1 and t_1 are assigned 0^λ. Two numbers R and R' are chosen randomly from $\{0,1\}^\lambda$, and u_1 and v_1 are assigned R and R' respectively. Now we give the details of how the message block $M[j]$, for $j := 1, 2, \cdots \ell - 1$ is encrypted: r_j is assigned $M[j]$; we compute $(r'_j||s'_j)$ as $\pi(r_j||s_j)$; we compute $u_{j+1}||v_{j+1}$ as $\pi(u_j||v_j)$; we obtain $C[j]$ as the XOR of v_{j+1} and r'_j; s_{j+1} is assigned the XOR of s'_j and t_j; and t_{j+1} is assigned the value r'_j.

For the last block $M[\ell]$, we perform the following operations: r_ℓ is assigned $M[\ell]$; we compute $(r'_\ell||s'_\ell)$ as $\pi(r_\ell||s_\ell)$; we compute $u_{\ell+1}||v_{\ell+1}$ as $\pi(u_\ell||v_\ell)$; $s_{\ell+1}$ is assigned the XOR of s'_ℓ and t_ℓ; u' is assigned the value of $u_{\ell+1}$; and $v_{\ell+1}$ is XORed with $s_{\ell+1}$ to obtain v'.

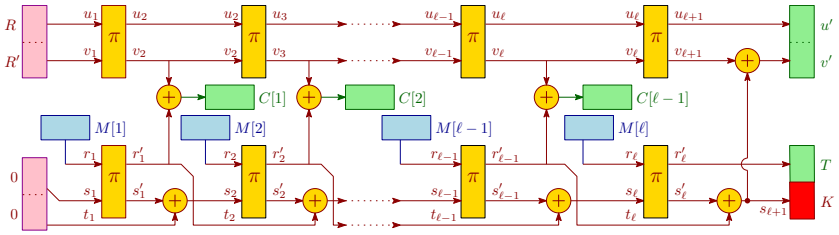
We assign $s_{\ell+1}$ as the decryption key K, we concatenate $C[1], C[2], \cdots, C[\ell-1]||u'||v'$ to obtain the ciphertext C, and r'_ℓ as the tag T.

Decryption RevD-1.$\mathcal{D}$. The decryption takes the parameter 1^λ, decryption key K, ciphertext C and tag T. Fig. 13(b) shows the decryption of ciphertext C. The decryption takes the ciphertext C and breaks it into several λ-bit blocks, namely, $C[1], C[2], \cdots, C[\ell-1]||u'||v'$. Decryption of C is composed of decryptions of individual blocks executed in the direction opposite to the encryption. For decrypting the last block, we first perform following operations: we assign K to $s_{\ell+1}$; we assign T to r'_ℓ; we assign u' to $u_{\ell+1}$; v' is XORed with $s_{\ell+1}$ to obtain $v_{\ell+1}$; we compute $u_\ell||v_\ell$ as $\pi^{-1}(u_{\ell+1}||v_{\ell+1})$; if $\ell > 1$, we compute t_ℓ as XOR of $C[\ell-1]$ and v_ℓ, otherwise as 0^λ; $s_{\ell+1}$ is XORed with t_ℓ to obtain s'_ℓ; $r_\ell||s_\ell$ as $\pi^{-1}(r'_\ell||s'_\ell)$; and we assign r_ℓ to $M[\ell]$.

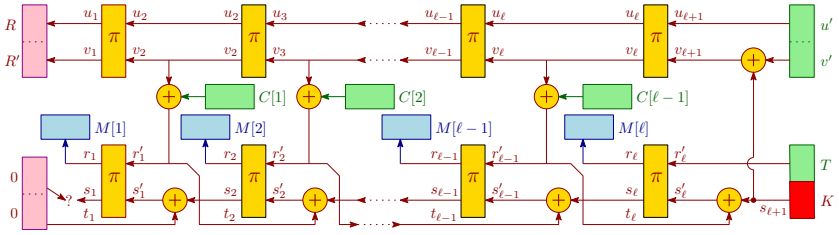
Now we give the details of how the ciphertext block $C[j]$ for $j := \ell-1, \ell-2, \cdots, 1$ is decrypted: we compute $u_j||v_j$ as $\pi^{-1}(u_{j+1}||v_{j+1})$; if the value of j is 1, then we assign t_j as 0^λ, otherwise t_j is assigned the XOR of $C[j-1]$ and v_j; r'_j is assigned t_{j+1}; s'_j is assigned the XOR of s_{j+1} and t_j; we compute $r_j||s_j$ as $\pi^{-1}(r'_j||s'_j)$; and $M[j]$ is assigned the value of r_j.

We concatenate $M[1], M[2], \cdots, M[\ell]$ to obtain the message M. The message is only returned if the value of variable s_1 is equal to 0^λ.

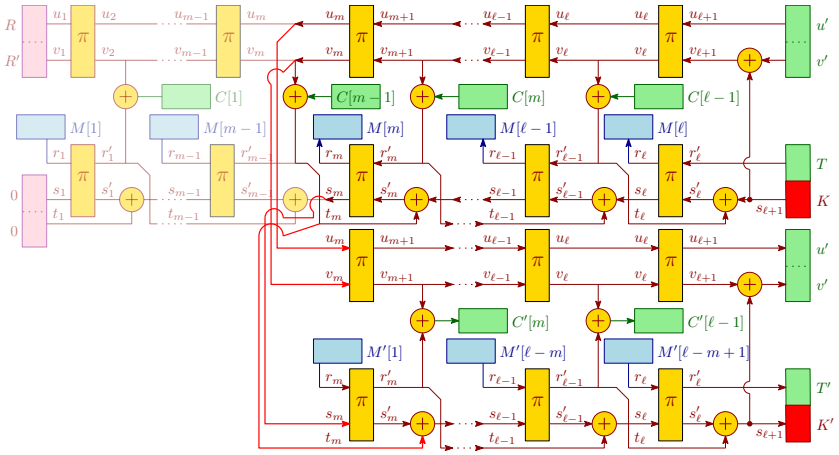
Update RevD-1.$\mathcal{U}$. The update takes the parameter 1^λ, the index of starting and ending bits i_{st} and i_{end}, new message bits $M_{\text{new}} \in \mathcal{M}$, the decryption key K, the ciphertext to be updated C, the tag T and the bit *app*. Fig. 13(c) shows



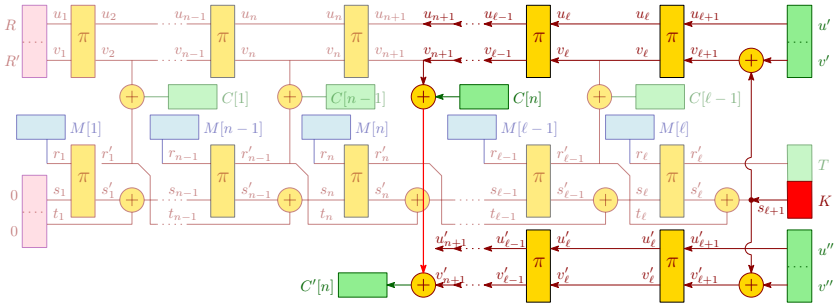
(a) Pictorial description of RevD-1. $\mathcal{E}(1^\lambda, M)$.



(b) Pictorial description of RevD-1. $\mathcal{D}(1^\lambda, K, C, T)$.



(c) Pictorial description of RevD-1. $\mathcal{U}(1^\lambda, i_{st}, i_{end}, M_{\text{new}}, K, C, T, app)$.



(d) Pictorial description of RevD-1. $\mathcal{P}(1^\lambda, Q = (n, u'', v''), M, K, C, T)$.

Fig. 13. Diagrammatic description of RevD-1.

RevD-1. $\mathcal{E}(1^\lambda, M)$

$\ell := |M|/\lambda$, $s_1 := 0^\lambda$, $t_1 := 0^\lambda$;
$M[1]||M[2]||\cdots||M[\ell] := M$;
$R \xleftarrow{\$} \{0,1\}^\lambda$, $R' \xleftarrow{\$} \{0,1\}^\lambda$;
$u_1 := R$, $v_1 := R'$;
for $(j := 1, 2, \cdots, \ell - 1)$
  $r_j := M[j]$, $r'_j||s'_j := \pi(r_j||s_j)$;
  $u_{j+1}||v_{j+1} := \pi(u_j||v_j)$, $C[j] := v_{j+1} \oplus r'_j$;
  $s_{j+1} := s'_j \oplus t_j$, $t_{j+1} := r'_j$;
$r_\ell := M[\ell]$, $r'_\ell||s'_\ell := \pi(r_\ell||s_\ell)$;
$u_{\ell+1}||v_{\ell+1} := \pi(u_\ell||v_\ell)$, $s_{\ell+1} := s'_\ell \oplus t_\ell$;
$u' := u_{\ell+1}$, $v' := v_{\ell+1} \oplus s_{\ell+1}$;
$K := s_{\ell+1}$, $T := r'_\ell$;
$C := C[1]||C[2]||\cdots||C[\ell-1]||u'||v'$;
return (K, C, T);

RevD-1. $\mathcal{D}(1^\lambda, K, C, T)$

$\ell := |C|/\lambda - 1$, $s_{\ell+1} := K$, $r'_\ell := T$;
$C[1]||C[2]||\cdots||C[\ell-1]||u'||v' := C$;
$u_{\ell+1} := u'$, $v_{\ell+1} := v' \oplus s_{\ell+1}$;
$u_\ell||v_\ell := \pi^{-1}(u_{\ell+1}||v_{\ell+1})$;
if $\ell = 1$, **then** $t_\ell := 0^\lambda$;
Else $t_\ell := C[\ell-1] \oplus v_\ell$;
$s'_\ell := s_{\ell+1} \oplus t_\ell$, $r_\ell||s_\ell := \pi^{-1}(r'_\ell||s'_\ell)$;
$M[\ell] := r_\ell$;
for $(j := \ell - 1, \ell - 2, \cdots, 1)$
  $u_j||v_j := \pi^{-1}(u_{j+1}||v_{j+1})$;
  If $j = 1$, **then** $t_j := 0^\lambda$;
  Else $t_j := C[j-1] \oplus v_j$;
  $r'_j := t_{j+1}$, $s'_j := s_{j+1} \oplus t_j$;
  $(r_j||s_j) := \pi^{-1}(r'_j||s'_j)$, $M[j] := r_j$;
$M := M[1]||M[2]||\cdots||M[\ell]$;
if $s_1 = 0^\lambda$ **then** return M;
else return $\perp$;

RevD-1. $\mathcal{P}(1^\lambda, Q = (n, u'', v''), M, K, C, T)$

$\ell := |C|/\lambda - 1$, $s_{\ell+1} := K$;
$C[1]||C[2]||\cdots||C[\ell-1]||u'||v' := C$;
$u_{\ell+1} := u'$, $v_{\ell+1} := v' \oplus s_{\ell+1}$;
$u'_{\ell+1} := u''$, $v'_{\ell+1} := v'' \oplus s_{\ell+1}$;
for $(j := l, l - 1, \cdots, n + 1)$
  $u_j||v_j := \pi^{-1}(u_{j+1}||v_{j+1})$;
  $u'_j||v'_j := \pi^{-1}(u'_{j+1}||v'_{j+1})$;
$C'[n] := C[n] \oplus v_{n+1} \oplus v'_{n+1}$;
return $C'[n]$;

RevD-1. $\mathcal{U}(1^\lambda, i_{st}, i_{end}, M_{\text{new}}, K, C, T, app)$

$\ell := |C|/\lambda - 1$, $m := \lceil i_{st}/\lambda \rceil$;
$C[1]||C[2]||\cdots||C[\ell-1]||u'||v' := C$;
$s_{\ell+1} := K$, $r'_\ell := T$, $u_{\ell+1} := u'$;
$v_{\ell+1} := v' \oplus s_{\ell+1}$;
$u_\ell||v_\ell := \pi^{-1}(u_{\ell+1}||v_{\ell+1})$;
if $\ell = 1$, **then** $t_\ell := 0^\lambda$;
Else $t_\ell := C[\ell-1] \oplus v_\ell$;
$s'_\ell := s_{\ell+1} \oplus t_\ell$, $r_\ell||s_\ell := \pi^{-1}(r'_\ell||s'_\ell)$;
$M[\ell] := r_\ell$;
for $(j := \ell - 1, \ell - 2, \cdots, m)$
  $u_j||v_j := \pi^{-1}(u_{j+1}||v_{j+1})$;
  If $j = 1$, **then** $t_j := 0^\lambda$;
  Else $t_j := C[j-1] \oplus v_j$;
  $r'_j := t_{j+1}$, $s'_j := s_{j+1} \oplus t_j$;
  $(r_j||s_j) := \pi^{-1}(r'_j||s'_j)$, $M[j] := r_j$;
$idx_1 := i_{st} \mod \lambda$;
if $idx_1 = 0$, **then** $idx_1 := \lambda$;
$M' = M[m][1]||M[m][2]||\cdots$
  $\cdots||M[m][idx_1 - 1]||M_{\text{new}}$;
if $app = 0$
  $n := \lceil i_{end}/\lambda \rceil$, $idx_2 := i_{end} \mod \lambda$;
  If $idx_2 = 0$, **then** $idx_2 := \lambda$;
  $M' = M'||M[n][idx_2 + 1]||M[n][idx_2 + 2]$
    $||\cdots||M[n][\lambda]||M[n+1]$
    $||M[n+2]||\cdots||M[\ell]$;
Else $\ell := (m-1) + (|M'|/\lambda)$;
$M'[1]||M'[2]||\cdots||M'[\ell - m + 1] := M'$;
for $(j := m, m + 1, \cdots, \ell - 1)$
  $r_j := M'[j - m + 1]$;
$r'_j||s'_j := \pi(r_j||s_j)$;
  $u_{j+1}||v_{j+1} := \pi(u_j||v_j)$;
  $C'[j] := v_{j+1} \oplus r'_j$;
  $s_{j+1} := s'_j \oplus t_j$, $t_{j+1} := r'_j$;
$r_\ell := M'[\ell - m + 1]$, $r'_\ell||s'_\ell := \pi(r_\ell||s_\ell)$;
$u_{\ell+1}||v_{\ell+1} := \pi(u_\ell||v_\ell)$, $s_{\ell+1} := s'_\ell \oplus t_\ell$;
$u' := u_{\ell+1}$, $v' := v_{\ell+1} \oplus s_{\ell+1}$;
$K' := s_{\ell+1}$, $T' := r'_\ell$;
$C' := C[1]||C[2]||\cdots||C[m-1]||C'[m]$
  $||C'[m+1]||\cdots||C'[\ell-1]||u'||v'$;
return (K', C', T');

RevD-1. $\mathcal{V}(1^\lambda, Q = n, C, T, P)$

if $C[n] = P$, **then** return TRUE;
Else return FALSE;

Fig. 14. Algorithmic description of the *FMLE* scheme RevD-1.

the update on ciphertext C. The update takes the ciphertext C and breaks it into several λ-bit blocks, namely, $C[1], C[2], \cdots, C[\ell-1]||u'||v'$ and computes the index of block from where the update starts m as the ceil value of i_{st}/λ. Update of C is composed of decryptions of individual blocks executed in the direction opposite to the encryption upto m-th block, calculating s_m and t_m, and then encrypting the new updated message from m-th block to the last block. The decryption and encryption processes are already explained above.

We assign the new $s_{\ell+1}$ as the decryption key K', we concatenate $C[1], C[2], \cdots, C[m-1], C'[m], C'[m+1], \cdots, C'[\ell-1]||u'||v'$ to obtain the ciphertext C', and r'_ℓ as the tag T'.

Proof of ownership by Prover RevD-1.$\mathcal{P}$. The proof of ownership by prover takes the parameter 1^λ, the challenge $Q = (n, u'', v'')$, the message M, the decryption key K, the ciphertext C and the tag T. Fig. 13(d) shows the proof of ownership by prover for the ciphertext C. The algorithm takes the ciphertext C and breaks it into several λ-bit blocks, namely, $C[1], C[2], \cdots, C[\ell-1]||u'||v'$, assigns K to $s_{\ell+1}$; the values of u' and u'' are assigned to $u_{\ell+1}$ and $u'_{\ell+1}$, and the values of v' and v'' are XORed with $s_{\ell+1}$ to obtain $v_{\ell+1}$ and $v'_{\ell+1}$. Now, for $j := \ell, \ell-1, \cdots, n+1$, we calculate the values of $u_j||v_j$ as $\pi^{-1}(u_{j+1}||v_{j+1})$ and $u'_j||v'_j$ as $\pi^{-1}(u'_{j+1}||v'_{j+1})$. The values of $C[n]$, v_{n+1} and v'_{n+1} are XORed to get the proof $C'[n]$.

Proof of ownership by Verifier RevD-1.$\mathcal{V}$. The proof of ownership by verifier takes the parameter 1^λ, the challenge $Q = n$, the ciphertext C, the tag T and the proof P as inputs, and checks if $C[n]$ equals the proof P and returns TRUE if the equality holds, otherwise returns FALSE.

Security of RevD-1 Our assumption is that the 2λ-bit permutation used in the RevD-1 is an ideal permutation. On a distinct input, an Ideal permutation outputs a uniformly chosen element from the set of remaining elements in the range that had not been already outputted in previous queries. Similar properties are retained for the inverse permutation.

Theorem 1. *Let λ be the security parameter. If the π in the design of RevD-1 is assumed to be the 2λ-bit ideal permutation, then for all adversaries $\mathcal{A}$,*

$$Adv^{\mathsf{PRV\$\text{-}CDA}}_{\mathsf{RevD\text{-}1},\mathcal{S},\mathcal{A}}(1^\lambda) \leq \Big(\frac{m(m-1)}{2^{2\lambda}}\Big) + \Big(\frac{2\cdot(\ell+1)^2m^2}{2^\mu} + \frac{2\cdot(\ell+1)^2m^2}{2^\lambda}\Big).$$

Here, $\mathcal{S}$ is a message source that outputs the message vector $\boldsymbol{M}$ such that $\|\boldsymbol{M}\| \stackrel{def}{=} m = m(1^\lambda)$, with min-entropy *$\mu = \mu_{\mathcal{S}}(1^\lambda)$ and $\ell = \max_{i\in\{1,2,\cdots,m\}} |\boldsymbol{M}_i|$. The PRV\$-CDA game is defined in Fig. 7.*

Proof. We prove the security by constructing successive games (or hybrids) and finding adversarial advantages in distinguishing between them. In summary, we need to bound the adversarial advantage Δ between the Games G_S and G_L (denoted $G_S \stackrel{\Delta}{\to} G_L$). In order to do so, we construct two intermediate games: $G_S \stackrel{\Delta_1}{\to} G_1 \stackrel{\Delta_2}{\to} G_2 \stackrel{\Delta_3}{\to} G_L$. Using Triangle Inequality [7], $\Delta \leq \Delta_1 + \Delta_2 + \Delta_3$.

In what follows, we compute the values from Δ_1 to Δ_3.

Game $\mathbf{G}_S$: This game is identical to PRV\$-CDA for $b = 1$, that is, when the challenger sends to the adversary $\mathcal{A}$, the encryption of the vector of messages $\boldsymbol{M}$

(derived from source $\mathcal{S}$) using the encryption algorithm $\mathcal{E}$ of the RevD-1 scheme.

Game $\mathbf{G}_1$: This game is identical to Game G_S, except that it uses a modified version of $\mathcal{E}$, namely $\mathcal{E}'$, as described in Fig. 15. In $\mathcal{E}'$, the 2λ-bit permutation π of $\mathcal{E}$ is replaced by the 2λ-bit random function rf. Therefore,

$$\Delta_1 \stackrel{def}{=} \Big| \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\mathsf{RevD\text{-}1}^{\pi},\mathcal{S}}(1^\lambda, b=1)=1] - \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\mathsf{RevD\text{-}1}^{\mathsf{rf}},\mathcal{S}}(1^\lambda, b=1)=1]\Big|$$

Using PRP/PRF Switching Lemma [7], for an adversary $\mathcal{A}$ with the vector of messages $\boldsymbol{M}$, such that $\|\boldsymbol{M}\| = m$, we obtain:

$$\Delta_1 \leq \frac{m(m-1)}{2^{2\lambda}} \tag{1}$$

Game $\mathbf{G}_2$: This game is identical to Game G_1, except that it uses a modified version of $\mathcal{E}'$, namely $\mathcal{E}''$, as described in Fig. 15. In $\mathcal{E}''$, the *bad* flag is set, when there is a collision in the latter λ bits of the 2λ-bit input to the random function rf. Therefore,

$$\Delta_2 \stackrel{def}{=} \Big| \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\mathsf{RevD\text{-}1}^{\mathsf{rf}},\mathcal{S}}(1^\lambda, b=1)=1] - \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\mathsf{RevD\text{-}1}^{\mathsf{rf}'},\mathcal{S}}(1^\lambda, b=1)=1]\Big|$$

Using Code-Based Game Playing Technique [7], for an adversary $\mathcal{A}$ with the vector of messages $\boldsymbol{M}$, such that $\|\boldsymbol{M}\| = m$, we obtain:

$$\Delta_2 \leq \Pr[\mathcal{A} \text{ sets } bad_1 \text{ in } \mathrm{G}_2] + \Pr[\mathcal{A} \text{ sets } bad_2 \text{ in } \mathrm{G}_2] \tag{2}$$

The event that $\mathcal{A}$ sets bad_1 in G_2 happens when $s_{i,j}$ matches with one of the previous lower-half-inputs to the random function rf, that is, $s_{i,j}$ can match with $v_{i',j'}$ or $s_{i',j'}$ for all ($i' < i$ and $j' < \ell$) or (if $i' = i$, then $j' < j$).

The event that $\mathcal{A}$ sets bad_2 in G_2 happens when $v_{i,j}$ matches with one of the previous lower-half-inputs to the random function rf, that is, $v_{i,j}$ can match with $v_{i',j'}$ or $s_{i',j'}$ for all ($i' < i$ and $j' < \ell$) or ($i' = i$ and $j' < j$).

Calculation of $\Pr[\mathcal{A}$ sets bad_1 in $\mathbf{G}_2]$: Suppose that we are using the construction $\mathcal{E}''$, and we define the following events:

- A is the event that $\mathcal{A}$ sets bad_1 in G_2, that is, at least one of the $s_{i,j}$ collides with some $s_{i',j'}$ or $v_{i',j'}$ for $i, i' \in \{1, 2, \cdots, m\}$ and $j, j' \in \{1, 2, \cdots, \ell + 1\}$.

- $A_{i,j}$ is the event that all the $s_{i',j'}$'s and $v_{i',j'}$'s are all distinct for the values $j' \leq \ell$, when $i' < i$, and $j' < j$, when $i' = i$.

• $B_{i,j}$ is the event that the value $s_{i,j}$, for $i \in \{1, 2, \cdots, m\}$ and $j \in \{1, 2, \cdots, \ell + 1\}$, collides with some $s_{i',j'}$ or $v_{i',j'}$ for the values $j' \leq \ell$, when $i' < i$, and $j' < j$, when $i' = i$.

So, we calculate the Probability of event A as follows:

$$\begin{aligned}\Pr[A] &\leq \Pr[B_{1,2}|A_{1,2}] + \Pr[B_{1,3}|A_{1,3}] + \cdots + \Pr[B_{m,\ell+1}|A_{m,\ell+1}] \\ &\leq \Big(\frac{1}{2^\mu} + \frac{1}{2^\lambda}\Big) + \Big(\frac{2}{2^\mu} + \frac{2}{2^\lambda}\Big) + \cdots \Big(\frac{m \times (\ell+1)}{2^\mu} + \frac{m \times (\ell+1)}{2^\lambda}\Big) \\ &\leq \frac{(\ell+1)^2 m^2}{2^\mu} + \frac{(\ell+1)^2 m^2}{2^\lambda}\end{aligned}$$

$$\therefore \Pr[\mathcal{A} \text{ sets } bad_1 \text{ in } \mathrm{G}_2] \leq \frac{(\ell+1)^2 m^2}{2^\mu} + \frac{(\ell+1)^2 m^2}{2^\lambda} \tag{3}$$

Similarly, we can calculate:

$$\Pr[\mathcal{A} \text{ sets } bad_2 \text{ in } \mathrm{G}_2] \leq \frac{(\ell+1)^2 m^2}{2^\mu} + \frac{(\ell+1)^2 m^2}{2^\lambda} \tag{4}$$

Using the Eqs. 2, 3 and 4, we obtain the following:

$$\Delta_2 \leq \frac{2 \cdot (\ell+1)^2 m^2}{2^\mu} + \frac{2 \cdot (\ell+1)^2 m^2}{2^\lambda} \tag{5}$$

Game $\mathbf{G}_L$: This game is identical to PRV$-CDA for $b = 0$, that is, when the challenger sends to the adversary $\mathcal{A}$, a random string of length identical to the length of ciphertexts and tags obtained by the encryption of the vector of messages $\boldsymbol{M}$ (derived from source $\mathcal{S}$) using the encryption algorithm $\mathcal{E}$ of the RevD-1 scheme.

$$\begin{aligned}\Delta_3 \stackrel{def}{=} \Big| &\Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi'',\mathcal{S}}(1^\lambda, b = 1) = 1] \\ &- \Pr[\mathsf{PRV\$\text{-}CDA}^{\mathcal{A}}_{\Pi,\mathcal{S}}(1^\lambda, b = 0) = 1]\Big|\end{aligned}$$

In Game G_2, each value of $v_{i,j}$ is generated randomly and is distinct from previous values, therefore the ciphertext generated $\boldsymbol{C}_i[j-1]$ will be also random, for $(i, j) \in \{1, 2, \cdots m\} \times \{1, 2, \cdots, \ell + 1\}$, where $\|\boldsymbol{M}\| = m$ and $\ell = \max_{i \in \{1,2,\cdots,m\}} |\boldsymbol{M}[i]|$. Thus, the games G_2 and G_L output a completely random ciphertext $\boldsymbol{C}$, without releasing any non-trivial information to the adversary and are, therefore, indistinguishable. Hence, we obtain:

$$\Delta_3 = 0 \tag{6}$$

Using Triangle Inequality [7] and the Eqs.1, 5 and 6, the following equation can be obtained.

$$\begin{aligned}\Delta &\leq \Delta_1 + \Delta_2 + \Delta_3 \\ &\leq \Big(\frac{m(m-1)}{2^{2\lambda}}\Big) + \Big(\frac{2\cdot(\ell+1)^2m^2}{2^{\mu}} + \frac{2\cdot(\ell+1)^2m^2}{2^{\lambda}}\Big) + 0\end{aligned}$$

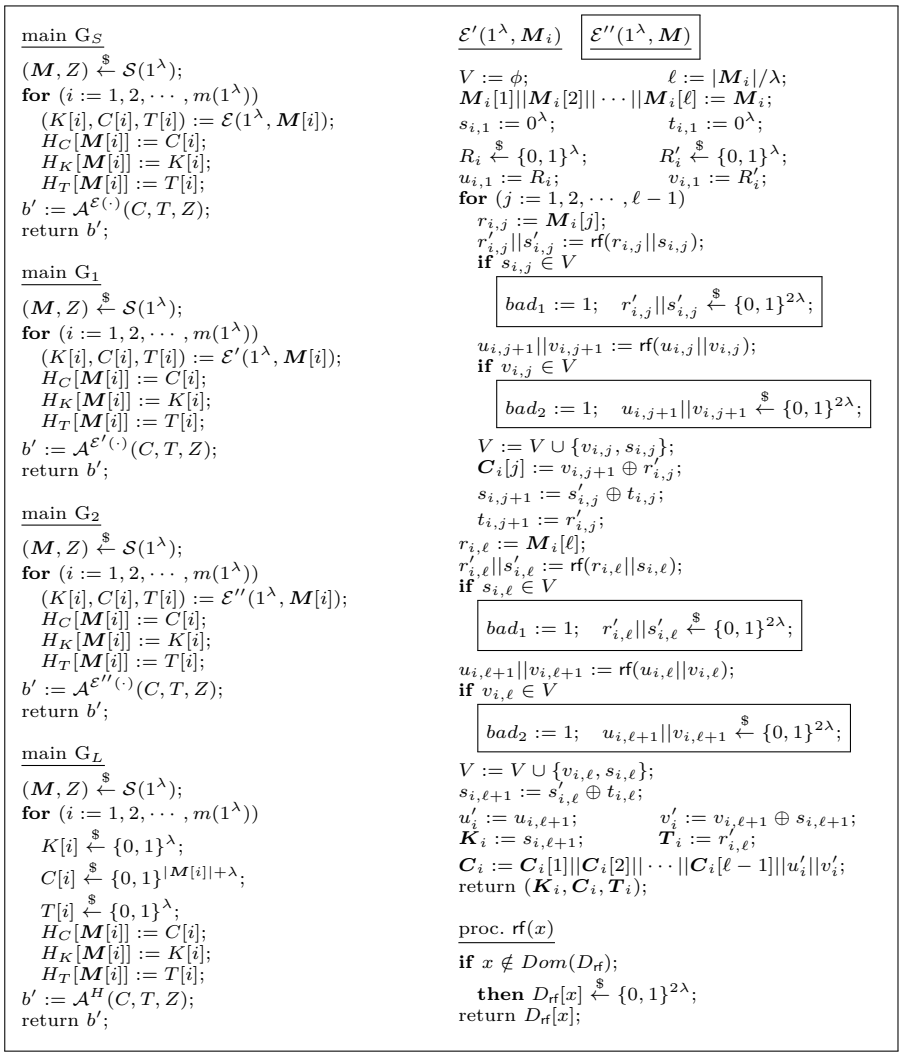


Fig. 15. Games used in PRV$-CDA of RevD-1.

Theorem 2. *Let λ be the security parameter. For all TC adversaries $\mathcal{A}$ against RevD-1, there exists a poly-reducible CRHF adversary $\mathcal{B}$ against FP, such that:*

$$Adv^{\mathsf{TC}}_{\mathsf{RevD\text{-}1},\mathcal{A}}(1^\lambda) \leq Adv^{\mathsf{CRHF}}_{\mathsf{FP},\mathcal{B}}(1^\lambda)$$

The TC game is defined in Fig. 7 and the CRHF game is defined in Fig. 3.

Proof. We poly-reduce any TC adversary $\mathcal{A}$ against RevD-1 into a CRHF adversary $\mathcal{B}$ against FP. The explicit reduction is shown in Fig. 16 and the proof readily follows from it.

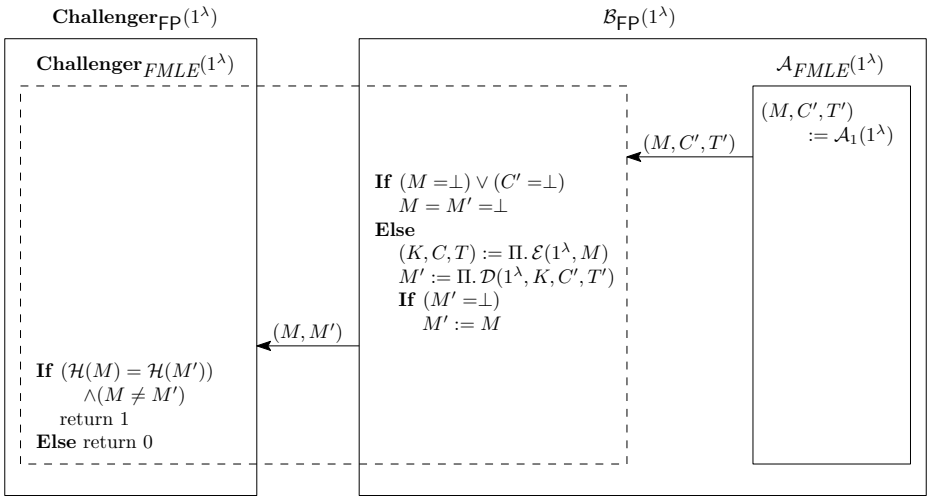


Fig. 16. Reducing an TC adversary $\mathcal{A}$ against RevD-1 into a CRHF adversary $\mathcal{B}$ against FP (see the TC game for RevD-1 in Fig. 7 and CRHF game for FP in Fig. 3.

Theorem 3. *Let λ be the security parameter. Then for all adversaries $\mathcal{A}$,*

$$Adv^{\mathsf{CXH}}_{\mathsf{RevD\text{-}1},\mathcal{A}}(1^\lambda) = 0.$$

The CXH game is defined in Fig. 7.

Proof. The function RevD-1.$\mathcal{E}(1^\lambda, M)$ can be rewritten as RevD-1.$\mathcal{E}_r(1^\lambda, M)$ because internally it generates and uses a random number r. So, the function RevD-1.$\mathcal{U}(1^\lambda, i_1, i_\rho, M_0[i_1, i_1+1, \cdots, i_\rho], \text{RevD-1}.\mathcal{E}_{r_2}(1^\lambda, M_1), 0)$ is equivalent to RevD-1.$\mathcal{E}_{r_2}(1^\lambda, M_0)$ (from the algorithms described in Fig. 14). Since, RevD-1.$\mathcal{E}_{r_1}(1^\lambda, M_0)$ and RevD-1.$\mathcal{E}_{r_2}(1^\lambda, M_0)$ differ in the choice of random number only. Therefore, RevD-1.$\mathcal{E}_{r_1}(1^\lambda, M_0)$ is indistinguishable from RevD-1.$\mathcal{U}(1^\lambda, i_1, i_\rho, M_0[i_1, i_1+1, \cdots, i_\rho], \text{RevD-1}.\mathcal{E}_{r_2}(1^\lambda, M_1), 0)$.

Theorem 4. *Let λ be the security parameter. Then for all adversaries $\mathcal{A}$,*

$$Adv^{\mathsf{UNC\text{-}CDA}}_{\mathsf{RevD\text{-}1},\mathcal{A}}(1^\lambda) = 0$$

The UNC-CDA *game is defined in Fig. 7.*

Proof. The UNC-CDA game requires the proof P^* generated by adversary $\mathcal{A}$ to be different from the actual proof P, and then P^* needs to be correctly verified by the function RevD-1.$\mathcal{V}$. However, this is not possible in RevD-1, because for each challenge Q, the corresponding unique proof P is already stored in the cloud (as the ciphertext) and some other value of $P^* \neq P$ will never be accepted by the function RevD-1.$\mathcal{V}$.

7.2 The RevD-2 scheme

We describe our second *FMLE* scheme, namely, RevD-2. This construction is motivated by the design of the authenticated encryption *APE* [2].

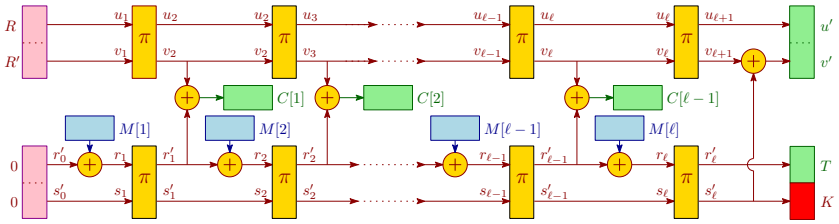
Description of RevD-2 The pictorial and algorithmic descriptions are given in Figs. 17 and 18; all wires are λ-bit long. Let M denote the message to be encrypted, $M[i]$ denote the i-th block of message, and $M[i][j]$ denote the j-th bit of i-th block of message. It is worth noting that the decryption is executed in the reverse direction of encryption. Below, we describe the 5-tuple of algorithms $(\mathcal{E}, \mathcal{D}, \mathcal{U}, \mathcal{P}, \mathcal{V})$ for RevD-2, in detail.

Encryption RevD-2.$\mathcal{E}$. Fig. 17(a) shows the encryption of message M. The encryption takes the parameter 1^λ and message M and breaks it into several λ-bit blocks, namely, $M[1], M[2], \cdots, M[\ell]$. Encryption of M is composed of encryptions of individual blocks in sequence. Two variables r'_0 and s'_0 are assigned 0^λ. Two numbers R and R' are chosen randomly from $\{0,1\}^\lambda$, and u_1 and v_1 are assigned R and R' respectively. Now we give the details of how the message block $M[j]$, for $j := 1, 2, \cdots \ell - 1$ is encrypted: r_j is assigned the XOR of $M[j]$ and r'_{j-1}; s_j is assigned the value of s'_{j-1}; we compute $(r'_j||s'_j)$ as $\pi(r_j||s_j)$; we compute $u_{j+1}||v_{j+1}$ as $\pi(u_j||v_j)$; and we obtain $C[j]$ as the XOR of v_{j+1} and r'_j.

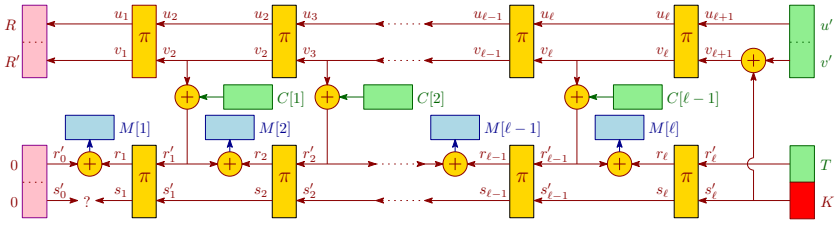
For the last block $M[\ell]$, we perform the following operations: r_ℓ is assigned the XOR of $M[\ell]$ and $r'_{\ell-1}$; s_ℓ is assigned the value of $s'_{\ell-1}$; we compute $(r'_\ell||s'_\ell)$ as $\pi(r_\ell||s_\ell)$; we compute $u_{\ell+1}||v_{\ell+1}$ as $\pi(u_\ell||v_\ell)$; u' is assigned the value of $u_{\ell+1}$; and $v_{\ell+1}$ is XORed with s'_ℓ to obtain v'.

We assign s'_ℓ as the decryption key K, we concatenate $C[1], C[2], \cdots, C[\ell - 1]||u'||v'$ to obtain the ciphertext C, and r'_ℓ as the tag T.

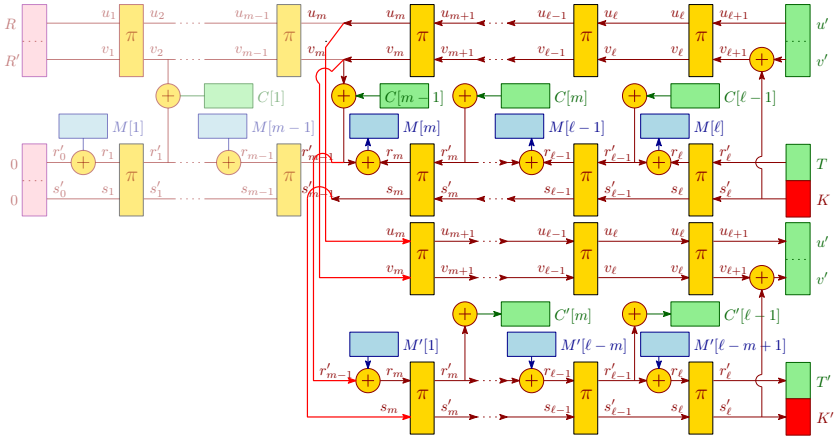
Decryption RevD-2.$\mathcal{D}$. The decryption takes the parameter 1^λ, decryption key K, ciphertext C and tag T. Fig. 17(b) shows the decryption of ciphertext C. The decryption takes the ciphertext C and breaks it into several λ-bit blocks,



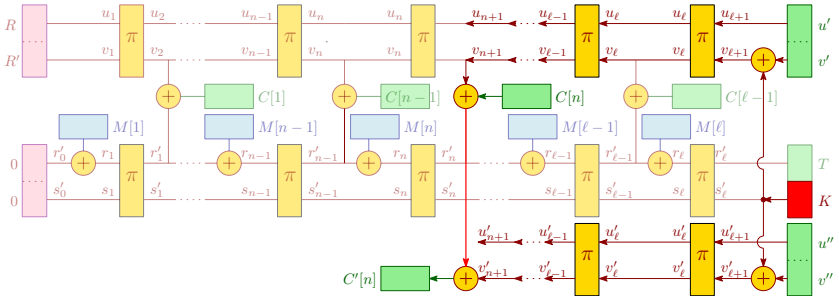
(a) Pictorial description of RevD-2. $\mathcal{E}(1^\lambda, M)$.



(b) Pictorial description of RevD-2. $\mathcal{D}(1^\lambda, K, C, T)$.



(c) Pictorial description of RevD-2. $\mathcal{U}(1^\lambda, i_{st}, i_{end}, M_{\text{new}}, C, T, K, app)$.



(d) Pictorial description of RevD-2. $\mathcal{P}(1^\lambda, Q = (n, u'', v''), C, T, M, K)$.

Fig. 17. Diagrammatic description of RevD-2.

RevD-2. $\mathcal{E}(1^\lambda, M)$

$\ell := |M|/\lambda$, $r'_0 := 0^\lambda$, $s'_0 := 0^\lambda$;
$M[1]||M[2]||\cdots||M[\ell] := M$;
$R \xleftarrow{\$} \{0,1\}^\lambda$, $R' \xleftarrow{\$} \{0,1\}^\lambda$;
$u_1 := R$, $v_1 := R'$;
for $(j := 1, 2, \cdots, \ell - 1)$
  $r_j := M[j] \oplus r'_{j-1}$, $s_j := s'_{j-1}$;
  $r'_j||s'_j := \pi(r_j||s_j)$;
  $u_{j+1}||v_{j+1} := \pi(u_j||v_j)$;
  $C[j] := v_{j+1} \oplus r'_j$;
$r_\ell := M[\ell] \oplus r'_{\ell-1}$, $s_\ell := s'_{\ell-1}$;
$r'_\ell||s'_\ell := \pi(r_\ell||s_\ell)$, $u_{\ell+1}||v_{\ell+1} := \pi(u_\ell||v_\ell)$;
$u' := u_{\ell+1}$, $v' := v_{\ell+1} \oplus s'_\ell$;
$K := s'_\ell$, $T := r'_\ell$;
$C := C[1]||C[2]||\cdots||C[\ell-1]||u'||v'$;
return (K, C, T);

RevD-2. $\mathcal{D}(1^\lambda, K, C, T)$

$\ell := |C|/\lambda - 1$, $r'_\ell := T$, $s'_\ell := K$;
$C[1]||C[2]||\cdots||C[\ell-1]||u'||v' := C$;
$u_{\ell+1} := u'$,$v_{\ell+1} := v' \oplus s'_\ell$;
$u_\ell||v_\ell := \pi^{-1}(u_{\ell+1}||v_{\ell+1})$;
$r_\ell||s_\ell := \pi^{-1}(r'_\ell||s'_\ell)$;
If $\ell = 1$, **then** $r'_{\ell-1} := 0^\lambda$;
Else $r'_{\ell-1} := C[\ell-1] \oplus v_\ell$;
$s'_{\ell-1} := s_\ell$, $M[\ell] := r'_{\ell-1} \oplus r_\ell$;
for $(j := \ell - 1, \ell - 2, \cdots, 1)$
  $r_j||s_j := \pi^{-1}(r'_j||s'_j)$;
  $u_j||v_j := \pi^{-1}(u_{j+1}||v_{j+1})$;
  If $j = 1$, **then** $r'_{j-1} := 0^\lambda$;
  Else $r'_{j-1} := C[j-1] \oplus v_j$;
  $M[j] := r_j \oplus r'_{j-1}$, $s'_{j-1} := s_j$;
$M := M[1]||M[2]||\cdots||M[\ell]$;
if $s'_0 = 0^\lambda$, **then** return M;
else return $\perp$;

RevD-2. $\mathcal{P}(1^\lambda, n, u'', v'', C, T, M)$

$\ell := |C|/\lambda - 1$, $s'_\ell := K$;
$C[1]||C[2]||\cdots||C[\ell-1]||u'||v' := C$;
$u_{\ell+1} := u'$, $v_{\ell+1} := v' \oplus s'_\ell$;
$u'_{\ell+1} := u''$, $v'_{\ell+1} := v'' \oplus s'_\ell$;
for $(j := l, l - 1, \cdots, n + 1)$
$u_j||v_j := \pi^{-1}(u_{j+1}||v_{j+1})$;
$u'_j||v'_j := \pi^{-1}(u'_{j+1}||v'_{j+1})$;
$C'[n] := C[n] \oplus v_{n+1} \oplus v'_{n+1}$;
return $C'[n]$;

RevD-2. $\mathcal{U}(1^\lambda, i_{st}, i_{end}, M_{\text{new}}, C, T, K, app)$

$\ell := |C|/\lambda - 1$, $m := \lceil i_{st}/\lambda \rceil$;
$C[1]||C[2]||\cdots||C[\ell-1]||u'||v' := C$;
$s'_{\ell+1} := K$, $r'_\ell := T$;
$u_{\ell+1} := u'$, $v_{\ell+1} := v' \oplus s'_\ell$;
$u_\ell||v_\ell := \pi^{-1}(u_{\ell+1}||v_{\ell+1})$;
$r_\ell||s_\ell := \pi^{-1}(r'_\ell||s'_\ell)$;
If $\ell = 1$, **then** $r'_{\ell-1} := 0^\lambda$;
Else $r'_{\ell-1} := C[\ell-1] \oplus v_\ell$;
$s'_{\ell-1} := s_\ell$, $M[\ell] := r'_{\ell-1} \oplus r_\ell$;
for $(j := \ell - 1, \ell - 2, \cdots, m)$
  $r_j||s_j := \pi^{-1}(r'_j||s'_j)$;
  $u_j||v_j := \pi^{-1}(u_{j+1}||v_{j+1})$;
  If $j = 1$, **then** $r'_{j-1} := 0^\lambda$;
  Else $r'_{j-1} := C[j-1] \oplus v_j$;
  $M[j] := r_j \oplus r'_{j-1}$, $s'_{j-1} := s_j$;
$idx_1 := i_{st} \mod \lambda$;
if $idx_1 = 0$, **then** $idx_1 := \lambda$;
$M' = M[m][1]||M[m][2]||\cdots$
  $\cdots||M[m][idx_1 - 1]||M_{\text{new}}$;
if $app = 0$
  $n := \lceil i_{end}/\lambda \rceil$, $idx_2 := i_{end} \mod \lambda$;
  If $idx_2 = 0$, **then** $idx_2 := \lambda$;
  $M' = M'||M[n][idx_2 + 1]||M[n][idx_2 + 2]$
    $||\cdots||M[n][\lambda]||M[n+1]$
    $||M[n+2]||\cdots||M[\ell]$;
Else $\ell := (m-1) + (|M'|/\lambda)$;
$M'[1]||M'[2]||\cdots||M'[\ell - m + 1] := M'$;
for $(j := m, m + 1, \cdots, \ell - 1)$
  $r_j := M'[j - m + 1] \oplus r'_{j-1}$, $s_j := s'_{j-1}$;
  $r'_j||s'_j := \pi(r_j||s_j)$;
  $u_{j+1}||v_{j+1} := \pi(u_j||v_j)$;
  $C'[j] := v_{j+1} \oplus r'_j$;
$r_\ell := M'[\ell - m + 1] \oplus r'_{\ell-1}$, $s_\ell := s'_{\ell-1}$;
$r'_\ell||s'_\ell := \pi(r_\ell||s_\ell)$;
$u_{\ell+1}||v_{\ell+1} := \pi(u_\ell||v_\ell)$;
$u' := u_{\ell+1}$, $v' := v_{\ell+1} \oplus s'_\ell$;
$K' := s'_\ell$, $T' := r'_\ell$;
$C' := C[1]||C[2]||\cdots||C[m-1]||C'[m]$
  $||C'[m+1]||\cdots||C'[\ell-1]||u'||v'$;
return (K', C', T');

RevD-2. $\mathcal{V}(1^\lambda, n, C, T, P)$

if $C[n] = P$, **then** return TRUE;
Else return FALSE;

Fig. 18. Algorithmic description of the *FMLE* scheme RevD-2.

namely, $C[1], C[2], \cdots, C[\ell-1]||u'||v'$. Decryption of C is composed of decryptions of individual blocks executed in the direction opposite to the encryption. For decrypting the last block, we first perform following operations: we assign T to r'_ℓ; we assign K to s'_ℓ; we assign u' to $u_{\ell+1}$; v' is XORed with s'_ℓ to obtain $v_{\ell+1}$; we compute $u_\ell||v_\ell$ as $\pi^{-1}(u_{\ell+1}||v_{\ell+1})$; $r_\ell||s_\ell$ as $\pi^{-1}(r'_\ell||s'_\ell)$; $C[\ell-1]$ is

XORed with v_ℓ to obtain $r'_{\ell-1}$, s_ℓ is assigned to $s'_{\ell-1}$; and $r'_{\ell-1}$ is XORed with r_ℓ to obtain $M[\ell]$.

Now we give the details of how the ciphertext block $C[j]$ for $j := \ell-1, \ell-2, \cdots, 1$ is decrypted: we compute $r_j||s_j$ as $\pi^{-1}(r'_j||s'_j)$; we compute $u_j||v_j$ as $\pi^{-1}(u_{j+1}||v_{j+1})$; if the value of j is 1, then we assign r'_{j-1} as 0^λ, otherwise r'_{j-1} is assigned the XOR of $C[j-1]$ and v_j; $M[j]$ is assigned the XOR of r_j and r'_{j-1}; and s'_{j-1} is assigned the value of s_j.

We concatenate $M[1], M[2], \cdots, M[\ell]$ to obtain the message M. The message is only returned if the value of variable s_1 is equal to 0^λ.

Update RevD-2.$\mathcal{U}$. The update takes the parameter 1^λ, the index of starting and ending bits i_{st} and i_{end}, new message bits $M_{\text{new}} \in \mathcal{M}$, the decryption key K, the ciphertext to be updated C, the tag T and the bit *app*. Fig. 17(c) shows the update on ciphertext C. The update takes the ciphertext C and breaks it into several λ-bit blocks, namely, $C[1], C[2], \cdots, C[\ell-1]||u'||v'$ and computes the index of block from where the update starts m as the ceil value of i_{st}/λ. Update of C is composed of decryptions of individual blocks executed in the direction opposite to the encryption upto m-th block, calculating r'_{m-1} and s'_{m-1}, and then encrypting the new updated message from m-th block to the last block. The decryption and encryption processes are already explained above.

We assign the new s'_ℓ as the decryption key K', we concatenate $C[1], C[2], \cdots, C[m-1], C'[m], C'[m+1], \cdots, C'[\ell-1]||u'||v'$ to obtain the ciphertext C', and r'_ℓ as the tag T'.

Proof of ownership by Prover RevD-2.$\mathcal{P}$. The proof of ownership by prover takes the parameter 1^λ, the challenge $Q = (n, u'', v'')$, the message M, the decryption key K, the ciphertext C and the tag T. Fig. 17(d) shows the proof of ownership by prover for the ciphertext C. The algorithm takes the ciphertext C and breaks it into several λ-bit blocks, namely, $C[1], C[2], \cdots, C[\ell-1]||u'||v'$; assigns K to s'_ℓ; the values of u' and u'' are assigned to $u_{\ell+1}$ and $u'_{\ell+1}$, and the values of v' and v'' are XORed with s'_ℓ to obtain $v_{\ell+1}$ and $v'_{\ell+1}$. Now, for $j := \ell, \ell-1, \cdots, n+1$, we calculate the values of $u_j||v_j$ as $\pi^{-1}(u_{j+1}||v_{j+1})$ and $u'_j||v'_j$ as $\pi^{-1}(u'_{j+1}||v'_{j+1})$. The values of $C[n]$, v_{n+1} and v'_{n+1} are XORed to get the proof $C'[n]$.

Proof of ownership by Verifier RevD-2.$\mathcal{V}$. The proof of ownership by verifier takes the parameter 1^λ, the challenge $Q = n$, the ciphertext C, the tag T and the proof P as inputs, and checks if $C[n]$ equals the proof P and returns TRUE if the equality holds, otherwise returns FALSE.

Security of RevD-2 The security proofs for the RevD-1 and RevD-2 are similar.

Resistance of RevD-1 and RevD-2 against Dictionary Attack Since, RevD-1 and RevD-2 have randomized encryption algorithms, they are not vulnerable to dictionary attacks (see Sect. 2.2 for a definition of dictionary attack).

7.3 Comparing RevD-1 and RevD-2 with the other *FMLE* schemes

In Table 1, we compare the RevD-1 and RevD-2 with the other *FMLE* constructions described in Sects. 5 and 6, on the basis of time and space complexities, and the security properties.

In summary, the *FMLE* schemes RevD-1 and RevD-2 possess the randomization property of *MLE* construction RCE, and the efficient update property of *UMLE*. It is also noted that it outperforms all the constructions in terms of the number of passes.

8 Conclusion

In this paper, we present a new cryptographic primitive *FMLE* and two new constructions of it: RevD-1 and RevD-2. We showed that these constructions perform better – both with respect to time and memory – than the existing constructions. The high performance is attributed to a unique property named *reverse decryption* of the *FP* hash function and the *APE* authenticated encryption, on which these new constructions are based. The only disadvantage is, perhaps, that these constructions are not STC secure. We leave as an open problem construction of an STC secure efficient *FMLE*.

References

1. Abadi, M., Boneh, D., Mironov, I., Raghunathan, A., Segev, G.: Message-Locked Encryption for Lock-Dependent Messages. In: Canetti, R., Garay, J. A. (eds.) CRYPTO 2013, LNCS, vol. 8042, pp. 374–391. Springer, Santa Barbara (2013). https://doi.org/10.1007/978-3-642-40041-4_21
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014, Revised Selected Papers, LNCS, vol. 8540, pp. 168–186. Springer, London (2014). https://doi.org/10.1007/978-3-662-46706-0_9
3. Andreeva, E., Daemen, J., Mennink, B., Assche, G. V.: Security of Keyed Sponge Constructions Using a Modular Proof Approach. In: Leander, G. (ed.) Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers, LNCS, vol. 9054, pp. 364–384. Springer, Istanbul (2015). https://doi.org/10.1007/978-3-662-48116-5_18
4. Bellare, M., Keelveedhi, S.: Interactive Message-Locked Encryption and Secure Deduplication. In: Katz, J. (ed.) Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings, LNCS, vol. 9020, pp. 516–538. Springer, Gaithersburg (2015). https://doi.org/10.1007/978-3-662-46447-2_23
5. Bellare, M., Keelveedhi, S., Ristenpart, T.: DupLESS: Server-Aided Encryption for Deduplicated Storage. In: King, S. (ed.) USENIX 2013, pp. 179–194.

Const. / Param.	**F-CE** [15]	**F-HCE2** [6]	**F-RCE** [6]	**F-UMLE** [20]	**RevD-1/RevD-2 This Paper**
Time of $\mathcal{E}$					
• Key Gen	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\left(\frac{\lvert F\rvert}{B}+\log_{B/\lambda}\lvert F\rvert\right)\mathsf{c}_{\mathcal{H}_B}$	–
• Tag Gen	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_\lambda}$	$\mathsf{c}_{\mathcal{H}_\lambda}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}+\left(\frac{\lvert F\rvert}{B}+\log_{B/\lambda}\lvert F\rvert\right)\mathsf{c}_{\mathcal{H}_B}$	–
• Cipher	$\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	$\left(\frac{\lvert F\rvert}{B}+\log_{B/\lambda}\lvert F\rvert\right)\mathsf{c}_{\mathcal{SE}_B}$	$\mathbf{2\cdot\frac{\lvert F\rvert}{\lambda}\mathsf{c}_{\pi_{2\lambda}}}$
# of passes	3	2	1	$1+\log_{B/\lambda}\lvert F\rvert$	**1**
Time of $\mathcal{D}$					
• Plaintext	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}$	$\left(\frac{\lvert F\rvert}{B}+\log_{B/\lambda}\lvert F\rvert\right)\mathsf{c}_{\mathcal{SD}_B}$	$\mathbf{2\cdot\frac{\lvert F\rvert}{\lambda}\mathsf{c}_{\pi^{-1}_{2\lambda}}}$
• Tag Verif	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}+\mathsf{c}_{\mathcal{H}_\lambda}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}+\mathsf{c}_{\mathcal{H}_\lambda}$	$\left(\frac{\lvert F\rvert}{B}+\log_{B/\lambda}\lvert F\rvert\right)\mathsf{c}_{\mathcal{H}_B}$	–
# of passes	2	2	1	$1+\log_{B/\lambda}\lvert F\rvert$	**1**
Time of $\mathcal{U}$					
• Decryp	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}$	$\log_{B/\lambda}\lvert F\rvert.\,\mathsf{c}_{\mathcal{SD}_B}$	$\mathbf{2\cdot\frac{\lvert F'\rvert}{\lambda}\mathsf{c}_{\pi^{-1}_{2\lambda}}}$
• Encryp	$\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	$\log_{B/\lambda}\lvert F\rvert.\,\mathsf{c}_{\mathcal{SE}_B}$	$\mathbf{2\cdot\frac{\lvert F'\rvert}{\lambda}\mathsf{c}_{\pi_{2\lambda}}}$
• Key Gen	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\log_{B/\lambda}\lvert F\rvert.\,\mathsf{c}_{\mathcal{H}_B}$	–
• Tag Gen	$\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	$\mathsf{c}_{\mathcal{H}_\lambda}$	$\mathsf{c}_{\mathcal{H}_\lambda}$	$\log_{B/\lambda}\lvert F\rvert.\,\mathsf{c}_{\mathcal{H}_B}+\mathsf{c}_{\mathcal{H}_{\lvert F\rvert}}$	–
# of passes	4	3	1	$1+\frac{\log_{B/\lambda}\lvert F\rvert}{\lvert F\rvert}$	$\mathbf{2\frac{\lvert F'\rvert}{\lvert F\rvert}}$
Time of $\mathcal{P}$					
• Gen Proof	*negl.*	*negl.*	$\mathsf{c}_{\mathcal{SD}_{\lvert F\rvert}}+\mathsf{c}_{\mathcal{SE}_{\lvert F\rvert}}$	*negl.*	$\mathbf{2\frac{\lvert F\rvert-k}{\lambda}\mathsf{c}_{\pi^{-1}_{2\lambda}}}$
# of passes	0	0	1	0	$\mathbf{1-\frac{k}{\lvert F\rvert}}$
Storage					
• Key	λ	λ	λ	λ	$\boldsymbol{\lambda}$
• Ciphertext	$\lvert F\rvert$	$\lvert F\rvert$	$\lvert F\rvert+\lambda$	$\lvert F\rvert+\log_{B/\lambda}\lvert F\rvert\cdot B$	$\boldsymbol{\lvert F\rvert+\lambda}$
• Tag	λ	λ	λ	$\left(\frac{\lvert F\rvert}{B}+\log_{B/\lambda}\lvert F\rvert\right)\lambda$	$\boldsymbol{\lambda}$
Security					
• TC	✓	✓	✓	✓	✓
• STC	✓	×	×	✓	×
• PRV-CDA	✓	✓	✓	✓	✓
• PRV$-CDA	✓	✓	✓	✓	✓
• CXH	✓	✓	✓	✓	✓
• UNC-CDA	✓	✓	✓	✓	✓
• Dict. Attack	×	×	✓	×	✓

Table 1. RevD-1 and RevD-2 are compared with the other *FMLE* schemes. Here, λ is the security parameter; F is the file to be encrypted; $|F|$ denotes the bit-length of F; B is the block-size used in the *UMLE* scheme; F' is the shortest suffix of F containing all modified bits; k is the index of the first bit in the challenge in the PoW protocol; $\mathsf{c}_{\mathcal{H}_{|F|}}$, $\mathsf{c}_{\mathcal{H}_B}$ and $\mathsf{c}_{\mathcal{H}_\lambda}$ are the costs of computing hashes on inputs of lengths $|F|$, B and λ respectively; $\mathsf{c}_{\mathcal{SE}_{|F|}}$ and $\mathsf{c}_{\mathcal{SE}_B}$ are the costs of encryption (in a one-time symmetric encryption) of the messages of lengths $|F|$ and B respectively; $\mathsf{c}_{\mathcal{SD}_{|F|}}$ and $\mathsf{c}_{\mathcal{SD}_B}$ are the costs of decryption of the messages of lengths $|F|$ and B respectively; $\mathsf{c}_{\pi_{2\lambda}}$ and $\mathsf{c}_{\pi^{-1}_{2\lambda}}$ are the costs of computing the 2λ-bit permutation and its inverse respectively.

6. Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-Locked Encryption and Secure Deduplication. In: Johansson, T., Nguyen, P. Q. (eds.) EUROCRYPT 2013, LNCS, vol. 7881, pp. 296–312. Springer, Athens (2013). https://doi.org/10.1007/978-3-642-38348-9_18
7. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006, LNCS, vol. 4004, pp. 409-426. Springer, Saint Petersburg (2006). https://doi.org/10.1007/11761679_25
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G. V.: Sponge Functions. ECRYPT 2007, `http://sponge.noekeon.org/SpongeFunctions.pdf`. Last accessed 1 March 2012
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G. V.: On the Indifferentiability of the Sponge Construction. In: Smart, N. P. (ed.) EUROCRYPT 2008, Istanbul, Turkey, LNCS, vol. 4965, pp. 181-197. Springer, Istanbul (2008). https://doi.org/10.1007/978-3-540-78967-3_11
10. Bertoni, G., Daemen, J., Peeters, M., Assche, G. V.: The Keccak Hash Function. In: The 1st SHA-3 Candidate Conference 2009, Leuven, Belgium, (2009).
11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Miri, A., Vaudenay, S. (eds.) Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers, LNCS, vol. 7118, pp. 320–337. Springer, Toronto (2011). https://doi.org/10.1007/978-3-642-28496-0_19
12. Bertoni, G., Daemen, J., Peeters, M., Assche, G. V., Keer, R. V.: CAESAR submission: KETJE v1, (2014). `https://competitions.cr.yp.to/round1/ketjev1.pdf`
13. Canard, S., Laguillaumie, F., Paindavoine, M.: Verifiable Message-Locked Encryption. In: Foresti, S., Persiano, G. (eds.) Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings, LNCS, vol. 10052, pp. 299–315. Springer, Milan (2016). https://doi.org/10.1007/978-3-319-48965-0_18
14. Chen, R., Mu, Y., Yang, G., Guo, F.: BL-MLE: Block-Level Message-Locked Encryption for Secure Large File Deduplication. In: IEEE Trans. Information Forensics and Security 2015, vol. 10, no. 12, pp. 2643–2652 (2015). https://doi.org/10.1109/TIFS.2015.2470221
15. Douceur, J. R., Adya, A., Bolosky, W. J., Simon, D., Theimer, M.: Reclaiming Space from Duplicate Files in a Serverless Distributed File System. In: ICDCS 2002, pp. 617-624 (2002). https://doi.org/10.1109/ICDCS.2002.1022312
16. Huang, K., Zhang, X., Wang, X.: Block-Level Message-Locked Encryption with Polynomial Commitment for IoT Data. In: Journal of Information Science and Engineering (JISE) 2017, vol. 33, no. 4, pp. 891–905 (2017). `http://jise.iis.sinica.edu.tw/JISESearch/pages/View/PaperView.jsf?keyId=157_2047`
17. Jiang, T., Chen, X., Wu, Q., Ma, J., Susilo, W., Lou, W.: Towards Efficient Fully Randomized Message-Locked Encryption. In: Liu, J. K., Steinfeld, R. (eds.) Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part I, LNCS, vol. 9722, pp. 361–375. Springer, Melbourne (2016). https://doi.org/10.1007/978-3-319-40253-6_22
18. Paul, S., Homsirikamol, E., Gaj, K.: A Novel Permutation-based Hash Mode of Operation FP and the Hash Function SAMOSA. In: Galbraith, S., Nandi, M. (eds.) Progress in Cryptology - INDOCRYPT 2012, Kolkata, WB, India, LNCS, vol. 7668, pp. 514–532. Springer, Verlag (2012).

19. Wang, H., Chen, K., Qin, B., Lai, X., Wen, Y.: A new construction on randomized message-locked encryption in the standard model via UCEs. In: SCIENCE CHINA Information Sciences 2017, vol. 60, no. 5, pp. 52101 (2017). https://doi.org/10.1007/s11432-015-1037-2
20. Zhao, Y., Chow, S. S. M: Updatable Block-Level Message-Locked Encryption. In: Karri, R., Sinanoglu O., Sadeghi A., Yi, X. (eds.) Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017, pp. 449–460. ACM (2017). https://doi.org/10.1145/3052973.3053012