UdMAC: Efficiently Updatable Message Authentication Codes

Debrup Chakraborty[1] and Avishek Majumder[2]

[1] Indian Statistical Institute, Kolkata, India
[2] Krea University, Sri City, India

Abstract

Message authentication codes (MAC) are ubiquitous and are considered to be the most important tool employed to ensure authenticity of messages in the symmetric key setting. In this work, we aim to empower MACs with a newly added functionality of updatablility, i.e., the goal is to construct a MAC scheme where the authentication tag for a message can be updated with every update to the message without recomputing the tag for the entire message. Such a functionality can be useful in several scenarios, primarily where the storage of a frequently changing large message is delegated to an un-trusted server. In such a scenario it may be infeasible for an user to download the entire message and recompute the tag for every update.

Bellare et al. [CRYPTO'94, STOC'95] introduced a new class of history-free updatable MACs, which can update a message tag without requiring any previous information. However, their update algorithm takes the entire document as an input, which can be impractical in many real-world scenarios. We introduce a new class of MACs called updatable message authentication codes (UdMACs). An UdMAC allows an existing message tag to be updated using only the previous tag, the message update, and a small state, without accessing the previous message. We systematically develop UdMACs, provide syntax for UdMAC, propose formal security notion. We then present two constructions: ConCatU and XoRU, which support two distinct message updates, namely, concatenation and xor difference. We analyze both the constructions in details and prove security of the construction in the newly proposed security model.

Keywords: Message Authentication Codes, Updatable MAC, Provable Security, Incremental Cryptography

1 Introduction

Message authentication codes (MACs) are algorithms traditionally used to provide integrity of a message. A MAC scheme consists of a pair of algorithms namely *MAC generation*, *MAC Verification*). Given a message and a key, the MAC generation algorithm outputs a *tag*, which the sender transmits along with the message. Given a message, a key, and a tag, the Verification algorithm outputs 1 if the message is authentic and 0 otherwise.

For our purpose, we need to extend the basic functionality of a MAC. Consider a scenario where a client outsources a message M to be stored on a server. To ensure integrity, the client computes a MAC t for M and transmits M along with t to the server. Over time, if the client needs to update M by some change m, existing MAC schemes

Emails: debrup.chakraborty@gmail.com (Debrup Chakraborty), avishek.majumder1991@gmail.com (Avishek Majumder)

Date of this document: 2026-09-02.

require the client to reconstruct the full updated message and recompute the MAC from scratch. If M is large and updates are frequent, this is inefficient, since the client must handle the entire message just to authenticate a small change.

We address this problem by introducing a new class of MACs, which we call *updatable MACs* (UdMAC). Let $\mathcal{M}$ be a message space and let $\Delta : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ be an associative binary operator with an identity element $\emptyset \in \mathcal{M}$, which we call the update function; for two messages $M, m \in \mathcal{M}$, we write $\Delta(M, m)$ as $M\Delta m$. (For example, Δ may be concatenation, so that $M\Delta m$ denotes $M\|m$, with identity the empty string; or Δ may be bitwise xor, with identity the all-zero string.) A Δ-UdMAC is a stateful scheme with the following property: given a message M with tag t, and given only the update m, the tag t, and a small retained state, the client can compute the tag t' for the updated message $M' = M\Delta m$ — without access to M itself. The client can therefore send only the update m and the new tag t' to the server, which stores M' and t'; the client need only retain t' (and its associated state) for future updates. This functionality is not available in traditional MACs.

We formalize the syntax and security notion for Δ-UdMACs. We then give two constructions supporting two different, practically motivated choices of Δ:

- ConCatU converts any variable input length pseudorandom function (PRF) into a $\|$-updatable MAC, i.e., one supporting concatenation updates. The client's retained state grows only with the *number* of updates applied to a message, not with the size of the message itself. For example, in a motivating scenario where a client uploads an hour of surveillance video to a server every hour, the client's state grows by a small, fixed number of bits per hour, regardless of how large the video itself is. We prove ConCatU secure against forgery under the sole assumption that the underlying PRF is secure.
- XoRU uses a block cipher (a fixed length PRF) and a polynomial hash into a $\oplus$-updatable MAC, supporting updates that XOR a difference into the current message. XoRU uses a nonce-based design, which allows us to prove a security bound that does not degrade with the number of update queries made by an adversary.

For both constructions, we give concrete efficiency analyses in terms of field operations and block-cipher/PRF evaluations, showing that each update can be processed with cost independent of the size of the message accumulated so far, and we prove security against adaptive forgery adversaries under standard pseudorandomness assumptions.

1.1 Related Works

The closest line of research providing similar functionality is *Incremental Cryptography* [BGG94, BGG95, BM97, Fis97b, BKY01, MPRS12, SY15, MGS15, GP17, KV18, ABK20]. This line began with [BGG94, BGG95], which introduced incremental collision-resistant hash functions and signatures, later refined and formalized in [ABK20].

The concept of incremental cryptography was first introduced in [BGG94], focusing on two primitives: collision-free hashing and digital (public-key) signatures. This was extended in [BGG95] to include encryption and private-key variants of all primitives, along with a MAC scheme based on pseudorandom functions. An incremental authentication scheme supporting single-block insertion and deletion was introduced in [Fis97a], with lower bounds for such constructions given in [Fis97b]. Incremental authenticated encryption with associated data, maintaining a history to generate new tags, was explored in [SY15]. Vulnerabilities in the incremental MAC scheme of [BGG95] were identified in [KV18], along with a proposed fix. However, the schemes in [BGG95, KV18] both require the entire message as input during updates.

More recently, the first definitions and constructions for incremental pseudo-random functions (IPRFs) were introduced in [ABK20], built from an incremental hash function

with weak collision resistance together with a symmetric encryption scheme. A scheme secure in a single-document setting is first constructed, then extended to a multi-document setting, adapting the Carter-Wegman paradigm traditionally used for non-incremental MACs. Like the schemes above, the constructions in [BGG94, BGG95, ABK20] are history-free, i.e., the update function requires no previous state to generate the updated tag and operate in a nonce-based setting with non-repeating nonces. Their update function also requires the entire document as an argument. This is impractical in settings such as an encrypted database maintained by a server, where a client performs frequent updates: requiring the full database as input to authenticate each update defeats the purpose of outsourcing storage in the first place.

A separate line of work considers key-updatable encryption, where the key holder updates keys and computes a token that lets a party holding ciphertexts update them to the new key, without decrypting [BLMR13, BEKS20, EPRS17]. Key-updatable signatures and MACs have similarly been studied in [CRS+21], where the latter are termed updatable MACs (UMAC). We emphasize that this is a different notion from ours: [CRS+21] updates the *key* used to authenticate a fixed message, whereas we update the *message* itself under a fixed key. To avoid confusion with their terminology we use a different acronym, UdMAC. Our formulation is closer in spirit to [BGG94, BGG95, ABK20], though our syntax and constructions differ.

2 Preliminaries

For a finite set X, $|X|$ denotes the cardinality of X. Let X be any set, then $x \xleftarrow{\$} X$ means x is sampled uniformly at random from the set X. Let X, Y be two finite sets then, $X \times Y$ denotes the Cartesian products of X and Y. For a non-negative integer n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$, and for non-negative integers i, j, $i < j$, $[i, j]$ represents the set $\{i, i+1, ..., j\}$. The set $\{0,1\}^*$ represents the set of all binary strings, including the empty string, and for a positive integer n, $\{0,1\}^n$ denotes the set of all n bit strings. For $x, y \in \{0,1\}^*$, $x \| y$ denote the concatenation of the strings x and y. Let $\mathbb{F}_{2^n}$ denote the field with 2^n elements, n-bit strings can be seen as elements in $\mathbb{F}_{2^n}$. Thus additions and multiplication of n-bit strings are addition and multiplications in $\mathbb{F}_{2^n}$. For $x, y \in \{0,1\}^n$, we denote their addition and multiplication by $x \oplus y$ and xy respectively. For two string $x, y \in \{0,1\}^*$, $|x| \leq |y|$ by $x \oplus y$ we mean bitwise xor of $(x \| 0^{|y|-|x|})$ and y. For any $x \in \{0,1\}^*$, $\mathsf{parse}_n(x) = (x_1, x_2, \ldots, x_m)$ where each $|x_i| = n$ for $i < m$ and $|x_m| \leq n$. Also, for $x \in \{0,1\}^*$, $|x| \leq n$, $\mathsf{Pad}_n(x) = x \| 0^{n-|x|}$. We treat adversaries as algorithms. Sometimes we describe adversaries with access to oracles. For an adversary $\mathcal{A}$, by $\mathcal{A}^{\mathcal{O}} \Rightarrow 1$, we mean that $\mathcal{A}$ has oracle access to $\mathcal{O}$, and after interacting with $\mathcal{O}$, $\mathcal{A}$ outputs a 1.

Pseudorandom Function. Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a keyed function from $\mathcal{X}$ to $\mathcal{Y}$, which is denoted by $F(K, x)$ or $F_K(x)$, where $\mathcal{K}$ is called the key space, $\mathcal{X}$ is called the input space and $\mathcal{Y}$ is called the output space. Now consider an adversary $\mathcal{A}$, who has oracle access to a function from $\mathcal{X}$ to $\mathcal{Y}$. The task of $\mathcal{A}$ is to distinguish a function $f \in \{F_K\}_{K \in \mathcal{K}}$ from a random function. We consider the PRF advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{prf}}_F(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{F_K} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{RF}} \Rightarrow 1\right] \right|,$$

where $K \xleftarrow{\$} \mathcal{K}$ and $\mathsf{RF} \xleftarrow{\$} \mathsf{Func}(\mathcal{X}, \mathcal{Y})$, where $\mathsf{Func}(\mathcal{X}, \mathcal{Y})$ is the set of all functions mapping $\mathcal{X}$ to $\mathcal{Y}$.

2.1 Message Authentication Codes

A message authentication code (MAC) is a map $F : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$, where $\mathcal{K}$ is the key space and $\mathcal{M}$ the message space. We often write $F_K(\cdot)$ to denote $F(K, \cdot)$. The output

of a MAC is called the tag, and τ is called the tag length. The security of a MAC F is defined using an interaction of F with an adversary $\mathcal{A}$. It is assumed that $\mathcal{A}$ has an oracle access to $F_K(\cdot)$, where $K \xleftarrow{\$} \mathcal{K}$. For a query $x \in \mathcal{M}$ of $\mathcal{A}$ the oracle responds by sending $y = F_K(x)$. Let, $\mathcal{A}$ query $x_1, x_2, \ldots, x_q$ and gets $y_1, y_2, \ldots, y_q$ as responses from the oracle. These queries are performed adaptively. Finally, $\mathcal{A}$ outputs a pair (x^*, y^*), where $x^* \notin \{x_1, x_2, \ldots, x_q\}$. This pair is called a forgery and it is said that $\mathcal{A}$ has successfully *forged* F if $F_K(x^*) = y^*$. The auth-advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{\mathsf{auth}}_F(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A} \text{ forges}\right].$$

We say that F is (ϵ, t) secure if for every adversary $\mathcal{A}$, which runs for time at most t, $\mathsf{Adv}^{\mathsf{auth}}_F(\mathcal{A}) \leq \epsilon$. It is well known [BS23] that for any arbitrary adversary $\mathcal{A}$ for the MAC F there exists a PRF adversary $\mathcal{B}$ for F such that

$$\mathsf{Adv}^{\mathsf{auth}}_F(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}_F(\mathcal{B}) + \frac{1}{2^\tau}, \tag{1}$$

where $\mathcal{B}$ and $\mathcal{A}$ both run almost for the same time and ask almost the same number of queries.

2.2 Polynomial MACs

For $x \in \{0,1\}^*$ and $h \in \{0,1\}^n$, let $(x_1, x_2, \ldots, x_m) = \mathsf{parse}_n(x)$. We define the function

$$\mathsf{Poly}_h(x) = x_1 h \oplus x_2 h^2 \oplus \cdots \oplus \mathsf{Pad}_n(x_m) h^m$$

where the additions and multiplications are in $\mathbb{F}_{2^n}$. A related function is

$$\mathsf{PPoly}_h(x) = \mathsf{Poly}_h(x) \oplus |x| h^{m+1}.$$

The following result is well known.

Proposition 1. *Let $x_1, x_2 \in \{0,1\}^*$, $x_1 \neq x_2$ and let $\ell_1 = \lceil |x_1|/n \rceil$ and $\ell_2 = \lceil |x_2|/n \rceil$ then for any $c \in \{0,1\}^n$*

$$\Pr[h \xleftarrow{\$} \{0,1\}^n : \mathsf{PPoly}_h(x_1) \oplus \mathsf{PPoly}_h(x_2) = c] \leq \frac{\max\{\ell_1, \ell_2\} + 1}{2^n}.$$

The nonce-based Wegman Carter MAC [CW77, WC79] is a widely used MAC algorithm. A popular variant of this MAC takes in a message $x \in \{0,1\}^*$, two keys $K, h \in \{0,1\}^n$ and a nonce $N \in \{0,1\}^n$ and produces an n-bit tag as

$$\mathsf{WC}_{K,h}(x; N) = \mathsf{PPoly}_h(x) \oplus E_K(N), \tag{2}$$

where $E_K()$ is a pseudo-random function family. The Theorem below regarding the security of Wegman Carter MAC is well known (for example see [BS23]).

Theorem 1. *Let Wegman Carter MAC be as defined in Equation (2) and $\mathcal{A}$ be a nonce respecting[3] forgery adversary who asks at most q queries. Among the q queries, let the message queried of the largest length be of length ℓ. Then there exists a PRF adversary $\mathcal{B}$ for $E_K()$, such that*

$$\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{WC}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}_E(\mathcal{B}) + \frac{\ell + 1}{2^n}.$$

We will also need the following result regarding the function $\mathsf{Poly}_h()$.

[3] A nonce respecting adversary is an adversary who never repeats a nonce in its queries.

Proposition 2. *Let $x \in \{0,1\}^{n\ell_1}$ and $y \in \{0,1\}^{n\ell_2}$, and let $(x_1, x_2, \ldots, x_{\ell_1}) = \mathsf{parse}_n(x)$ and $(y_1, y_2, \ldots, y_{\ell_2}) = \mathsf{parse}_n(y)$ and $x_{\ell_1} \neq 0^n$ and $y_{\ell_2} \neq 0^n$ and $x \neq y$. Then, for any $c \in \{0,1\}^n$,*

$$\Pr[h \xleftarrow{\$} \{0,1\}^n : \mathsf{Poly}_h(x) \oplus \mathsf{Poly}_h(y) = c] \leq \frac{\max\{\ell_1, \ell_2\}}{2^n}.$$

Proof. Without loss of generality, let $\ell_1 \geq \ell_2$. Thus, we have

$$\begin{aligned} \mathsf{Poly}_h(x) &= x_1 h \oplus x_2 h^2 \oplus \cdots \oplus x_{\ell_2} h^{\ell_2} \oplus \cdots \oplus x_{\ell_1} h^{\ell_1} \\ \mathsf{Poly}_h(y) &= y_1 h \oplus y_2 h^2 \oplus \cdots \oplus y_{\ell_2} h^{\ell_2}. \end{aligned}$$

If $\ell_1 = \ell_2$ we have

$$\mathsf{Poly}_h(x) \oplus \mathsf{Poly}_h(y) = (x_1 \oplus y_1)h \oplus (x_2 \oplus y_2)h^2 \oplus \cdots \oplus (x_{\ell_1} \oplus y_{\ell_1})h^{\ell_1},$$

which is a non-zero polynomial of degree at most ℓ_1 as $x \neq y$. If $\ell_1 > \ell_2$ we have

$$\mathsf{Poly}_h(x) \oplus \mathsf{Poly}_h(y) = (x_1 \oplus y_1)h \oplus (x_2 \oplus y_2)h^2 \oplus \cdots \oplus (x_{\ell_2} \oplus y_{\ell_2})h^{\ell_2} \oplus \cdots \oplus x_{\ell_1} h^{\ell_1},$$

which is a non-zero polynomial of degree at most ℓ_1 as $x_{\ell_1} \neq 0^n$. Thus, in both cases we have $\mathsf{Poly}_h(x) \oplus \mathsf{Poly}_h(y) \oplus c = 0$ have at most ℓ_1 roots in $\mathbb{F}_{2^n} = \{0,1\}^n$.

Similar argument holds if $\ell_2 > \ell_1$ and in that case we have $\mathsf{Poly}_h(x) \oplus \mathsf{Poly}_h(y) \oplus c = 0$ have at most ℓ_2 roots in $\mathbb{F}_{2^n} = \{0,1\}^n$. Thus we have

$$\Pr[h \xleftarrow{\$} \{0,1\}^n : \mathsf{Poly}_h(x) \oplus \mathsf{Poly}_h(y) = c] \leq \frac{\max\{\ell_1, \ell_2\}}{2^n}.$$

□

3 Updatable MACs

An updatable MAC works on four associated sets $\mathcal{K}$, $\mathcal{M}$, Σ and $\mathcal{T}$ called the key space, message space, state space and tag space, respectively. We will consider $\mathcal{M} \subset \{0,1\}^*$ and an associative binary operation $\Delta : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$, which we will further call as an update function. An updatable MAC Ψ is a triple of algorithms $\Psi = (\mathsf{KeyGen}, \mathsf{MacUpdate}, \mathsf{MacVerify})$ described as follows:

> $\mathsf{KeyGen}(1^\lambda)$: takes as input the security parameter λ outputs a key K, selected uniformly at random from the key space $\mathcal{K}$.
>
> $\mathsf{MacUpdate}(K, m, \sigma)$: takes as input $K \in \mathcal{K}$, $m \in \mathcal{M}$ and $\sigma \in \Sigma \cup \{\perp\}$ and outputs a $t \in \mathcal{T}$ and an updated state $\widetilde{\sigma}$.
>
> $\mathsf{MacVerify}(K, m, \sigma, t)$: takes in the key K, a $m \in \mathcal{M}$, $\sigma \in \Sigma$ and $t \in \mathcal{T}$ and outputs a bit $b \in \{0,1\}$.

Correctness: Let $m_1, m_2, \ldots, m_s \in \mathcal{M}$ be a sequence of messages and $\Psi = (\mathsf{KeyGen}, \mathsf{MacUpdate}, \mathsf{MacVerify})$. For an updatable MAC Ψ, we define a procedure $\Psi.\mathsf{SeqUpdt}$ in Figure 1 which takes in a key $K \in \mathcal{K}$ generated by $\Psi.\mathsf{KeyGen}(1^\lambda)$, a sequence of messages $(m_1, m_2, \ldots, m_s)$ and outputs $(\sigma, t) \in \Sigma \times \mathcal{T}$. We say that Ψ is correct, if for any message sequence $(m_1, \ldots, m_s)$ and any K generated by $\Psi.\mathsf{KeyGen}$,

$$\Psi.\mathsf{MacVerify}(K, M, \Psi.\mathsf{SeqUpdt}(K, (m_1, m_2, \ldots, m_s))) = 1,$$

where $M = m_1 \ \Delta \ m_2 \ \Delta \ \cdots \ \Delta \ m_s$.

$\underline{\Psi.\mathsf{SeqUpdt}(K, (m_1, m_2, \ldots, m_s))}$
01. $(t_1, \sigma_1) \leftarrow \Psi.\mathsf{MacUpdate}(K, m_1, \perp)$
02. **for** $i = 2$ to s,
03. $\quad (\sigma_i, t_i) \leftarrow \Psi.\mathsf{MacUpdate}(K, m_i, \sigma_{i-1})$
04. **end for**
05. **return** (σ_s, t_s)

Figure 1: The procedure SeqUpdt used to define correctness of a updatable MAC scheme Ψ.

The definition of correctness specifies the usefulness of updatable MAC. Consider the MacUpdate procedure is applied on the messages $m_1, m_2, \ldots, m_s$, one after the other in the same sequence as in the procedure SeqUpdt of Figure 1. Thus, we start with the message m_1 and an empty state and in the subsequent messages the input to MacUpdate is the current message and the state obtained for the previous message. The final state-tag pair (σ_s, t_s) obtained should be a valid MAC for the message $M = m_1 \ \Delta \ m_2 \ \Delta \ \cdots \ \Delta \ m_s$.

3.1 Security of Updatable MACs

For defining security of an updatable MAC $\Psi = (\mathsf{KeyGen}, \mathsf{MacUpdate}, \mathsf{MacVerify})$, we give $\mathcal{A}$ oracle access of $\Psi.\mathsf{MacUpdate}(K, \cdot, \cdot)$, instantiated with a key K generated by $\Psi.\mathsf{KeyGen}(1^\lambda)$. $\mathcal{A}$ can query its oracle with $(m, \sigma) \in \mathcal{M} \times \Sigma$ and get as response $(\widetilde{\sigma}, t) \in \Sigma \times \mathcal{T}$. Let $\mathbb{Q} = \{(m_1, \sigma_1), (m_2, \sigma_2), \ldots (m_q, \sigma_q)\}$ be the set of queries made by $\mathcal{A}$ and for the ith query (m_i, σ_i), let the response of the oracle be $(\widetilde{\sigma}_i, t_i)$. We denote the set of messages queried by $\mathcal{A}$ by $\mathbb{M} = \{m : (m, \sigma) \in \mathbb{Q}\}$ and we call the set

$$\mathbb{P}_{\mathcal{A}} = \{(\sigma_i, m_i, \widetilde{\sigma}_i, t_i) : 1 \leq i \leq q\}$$

as the query profile of the adversary $\mathcal{A}$. Finally, after $\mathcal{A}$ stops querying, it outputs a forgery (M^*, σ^*, t^*).

We call a forgery (M^*, σ^*, t^*) of an adversary $\mathcal{A}$ with query profile $\mathbb{P}_{\mathcal{A}}$ as *"invalid"* if there exists a sequence of query-responses $(\sigma^{(1)}, m^{(1)}, \widetilde{\sigma}^{(1)}, t^{(1)}), (\sigma^{(2)}, m^{(2)}, \widetilde{\sigma}^{(2)}, t^{(2)}), \ldots,$ $(\sigma^{(\ell)}, m^{(\ell)}, \widetilde{\sigma}^{(\ell)}, t^{(\ell)})$, such that the following are true simultaneously

1. $\{(\sigma^{(1)}, m^{(1)}, \widetilde{\sigma}^{(1)}, t^{(1)}), (\sigma^{(2)}, m^{(2)}, \widetilde{\sigma}^{(2)}, t^{(2)}), \ldots, (\sigma^{(\ell)}, m^{(\ell)}, \widetilde{\sigma}^{(\ell)}, t^{(\ell)})\} \subseteq \mathbb{P}_{\mathcal{A}}$.
2. $m^{(1)} \ \Delta \ m^{(2)} \ \Delta \ \cdots \ \Delta \ m^{(\ell)} = M^*$
3. $\sigma^{(1)} = \perp$, $\widetilde{\sigma}^{(\ell)} = \sigma^*$ and $\sigma^{(i)} = \widetilde{\sigma}^{(i-1)}$ for $1 \leq i \leq \ell$.

A forgery (M^*, σ^*, t^*) is valid if it is not invalid. An adversary $\mathcal{A}$ is successful if its forgery (M^*, σ^*, t^*) is valid and $\Psi.\mathsf{MacVerify}(M^*, \sigma^*, t^*) = 1$. Let $\mathsf{Succ}(\mathcal{A})$ be the event that $\mathcal{A}$ is successful. We define the forging advantage of an adversary $\mathcal{A}$ for a Δ-updatable MAC as

$$\mathsf{Adv}^{\mathsf{Uauth}}_{\Psi}(\mathcal{A}) = \Pr[\mathsf{Succ}(\mathcal{A})]. \tag{3}$$

The probability is taken over the selection of the key K with which the oracle is instantiated, the (possible) randomness in Ψ, and the (possible) randomness in $\mathcal{A}$.

In the subsequent sections we describe the construction of two updatable MACs which support two different update functions. The first one is called ConCatU which converts any given PRF $f : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ to an $\|$-updatable MAC, i.e., where the update function is concatenation. The second construction converts a Wegman Carter MAC [CW77] to an $\oplus$-updatable MAC, where the update function is xor differences.

4 ConCatU: An Updatable MAC for Concatenation

We introduce few more notations for convenience. Let, $(\mathbb{N})^*$ denote the set of all lists whose elements are natural numbers including the empty list. Thus a finite list $L \in (\mathbb{N})^*$ can be seen as an array, and its ith element will be denoted by $L[i]$, and its length, i.e., the number of elements present in it, will be denoted by $||L||$. An empty list is denoted by $\langle\rangle$. For an $a \in \mathbb{N}$, $\mathsf{append}(L, a)$ appends a at the end of L. By $\mathsf{sum}(L)$ we will mean the sum of all the elements in L. Let, $L \in (\mathbb{N})^*$ be a finite length list of length $||L|| = \ell$ where $L = \langle x_1, x_2, \ldots, x_\ell\rangle$ and let $M \in \{0,1\}^*$ be a string of length $\mathsf{sum}(L)$. Then, $\mathsf{Parse}(M, L)$ returns $(m_1, m_2, \ldots, m_\ell)$ such that $|m_i| = L[i]$ and $m_1\|m_2\|\ldots\|m_\ell = M$. For a list $L = \langle x_1, x_2, \ldots, x_\ell\rangle$, by $\mathsf{str}(L)$ we mean the $n\ell$ bit string $\mathsf{bin}_n(x_1)\|\mathsf{bin}_n(x_2)\|\cdots\|\mathsf{bin}_n(x_\ell)$. If $L = \langle\rangle$, $||L|| = 0$, $\mathsf{sum}(L) = 0$, and $\mathsf{str}(L)$ is the empty string.

With this new notations in hand, we present the description of our first updatable MAC construction ConCatU. Let $M \subset \{0,1\}^*$, $\mathcal{K} = \{0,1\}^k \times \{0,1\}^n$, $\Sigma \subset (\mathbb{N})^* \times \{0,1\}^n \cup \{\perp\}$, $\mathcal{T} = \{0,1\}^n$ be the message space, key space, state space, and tag space of ConCatU respectively. Let $f : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ be a PRF family. We use f to construct a $\|$-updatable MAC ConCatU, where update function is concatenation.

A state σ of ConCatU is either $\perp$ or it contains two distinct fields denoted as $\sigma = (L_\sigma, \mathsf{pTag}_\sigma)$, where L_σ is a finite list of positive integers, and pTag_σ is a n-bit binary string. ConCatU.KeyGen() outputs a pair (K, h), where $K \xleftarrow{\$} \{0,1\}^k$ and $h \xleftarrow{\$} \{0,1\}^n$. The procedures ConCatU.MacUpdate and ConCatU.MacVerify are shown in Figure 2. To understand the workings of the procedures ConCatU.MacUpdate and ConCatU. MacVerify it is important see the role of the state σ.

Consider an initial message m_1, and let subsequent updates on it be $m_1, m_2, \ldots, m_k$. As the update function is a concatenation function, hence the message after the ith update, $1 \leq i \leq k$, will be $M_i = m_1\|m_2\|\cdots\|m_i$. We see the initial message also as an update on an empty message. The intended usage of ConCatU.MacUpdate is to obtain a correct authentication tag of the updated message after each update without knowing the whole message. To accomplish this, each update m is given as an input to the ConCatU.MacUpdate procedure is associated with a state σ. The state σ carries some information regarding the message on which the current update m is to be applied. In particular, σ keeps a record of the history of updates on the message. The procedure produces an authentication tag for the updated message and also updates the state with the information of the current update.

For an input (m, σ), if $\sigma = \perp$, then it signifies m is the initial message, i.e., an update m on an empty message is being sought. The ConCatU.MacUpdate procedure on such an input should produce a tag for the message m, which in this case is the updated message. Further, the procedure should populate the initially empty list L with the length of m and set the current state to (L, tag) and output it along with the tag.

When an input (m, σ) is received with $\sigma \neq \perp$, then a MAC is being sought for the message $M\|m$, where M is some message which is not being explicitly provided to the algorithm. But, the information regarding M is encoded in the state $\sigma = (L, \mathsf{pTag})$, which carries the following information:

1. The message M, over which the update is being sought was obtained after $k = ||L||$ updates.
2. The initial update on the empty message, i.e., the initial message was of length $L[1]$ bits.
3. For $2 \leq i \leq k$, the ith update increased the length of the message by $L[i]$ bits. Thus, for a correct usage, $|M| = \mathsf{sum}(L)$.
4. pTag was the tag computed by ConCatU.MacUpdate for the message M, i.e., the last call to ConCatU.MacUpdate for this specific message returned pTag.

$\underline{\mathsf{ConCatU.MacUpdate}((K,h),m,\sigma)}$

1: **if** $\sigma = \perp$ **then**
2: $\quad \ell \leftarrow 0$
3: $\quad \mathsf{pTag} \leftarrow 0^n$
4: $\quad L \leftarrow [\,]$
5: **else**
6: $\quad (L, \mathsf{pTag}) \leftarrow \sigma$
7: $\quad \ell \leftarrow ||L||$
8: **end if**
9: $X \leftarrow 1^n$
10: **for** $i = 1$ to ℓ **do**
11: $\quad X \leftarrow X \parallel \mathsf{bin}_n(L[i])$
12: **end for**
13: $p \leftarrow \mathsf{Poly}_h(X)$
14: $\mathsf{tag} \leftarrow f_K(p \parallel \mathsf{pTag} \parallel m)$
15: $L \leftarrow \mathsf{append}(L, |m|)$
16: $\sigma \leftarrow (L, \mathsf{tag})$
17: **return** (σ, tag)

$\underline{\mathsf{ConCatU.MacVerify}((K,h),M,\sigma,t)}$

21: **if** $\sigma = \perp$ **then**
22: $\quad$ **return** 0
23: **else**
24: $\quad (L, \mathsf{pTag}) \leftarrow \sigma$
25: $\quad$ **if** $\mathsf{sum}(L) \neq |M|$ **or** $\mathsf{pTag} \neq t$ **then**
26: $\quad\quad$ **return** 0
27: $\quad$ **end if**
28: **end if**
29: $\ell \leftarrow ||L||$
30: $(m_1, m_2, \ldots, m_\ell) \leftarrow \mathsf{Parse}(M, L)$
31: $X \leftarrow 1^n$; $\mathsf{tag} \leftarrow 0^n$
32: **for** $i = 1$ to ℓ **do**
33: $\quad p \leftarrow \mathsf{Poly}_h(X)$
34: $\quad \mathsf{tag} \leftarrow f_K(p \parallel \mathsf{tag} \parallel m_i)$
35: $\quad X \leftarrow X \parallel \mathsf{bin}_n(L[i])$
36: **end for**
37: **if** $\mathsf{tag} = t$ **then**
38: $\quad$ **return** 1
39: **else**
40: $\quad$ **return** 0
41: **end if**

Figure 2: Specification of ConCatU using a PRF $f_K : \{0,1\}^* \to \{0,1\}^n$.

As specified in Figure 2, the update procedure ConCatU.MacUpdate on an input (m, σ), parses $\sigma = (L, \mathsf{pTag})$ if $\sigma \neq \perp$ and if $\sigma = \perp$ it sets L to an empty list and pTag to 0^n. Further, it computes a string X as the concatenation of the n-bit binary encodings of the entries in L. In particular, if $\sigma = \perp$, then X is set to 1^n and otherwise it is set as $X = 1^n \| \mathsf{bin}_n(L[1]) \| \cdots \| \mathsf{bin}_n(L[\ell]) = 1^n \| \mathsf{str}(L)$, where $\ell = ||L||$. Then, p is computed as $\mathsf{Poly}_h(X)$ and finally the tag is produced as $f_K(p \| \mathsf{pTag} \| m)$. The new state σ is constructed as $(\mathsf{append}(L, |m|), \mathsf{tag})$.

The verification process takes a message M, a state σ and a tag t. First, it does some checks to verify if the state σ and the message M are compatible. In particular, if the following are true, then the verification procedure rejects by returning 0.

1. $\sigma = \perp$. This is an incompatible state for any message for verification, as $\sigma = \perp$ signifies that the message has never gone through the ConCatU.MacUpdate procedure.

2. $\sigma \neq \perp$ and $\sigma = (L, \mathsf{pTag})$ but $\mathsf{sum}(L) \neq |M|$. This also represents an incompatible state for the message M. As the list L keeps track of the updates performed on a message. The size of the list $||L||$ represents the number of updates that have taken place and $L[i]$ denotes the number of bits concatenated to the message in the i-th update. Hence, if $\mathsf{sum}(L) \neq |M|$, then σ represents an incompatible state for M.

3. $\sigma \neq \perp$ and $\sigma = (L, \mathsf{pTag})$ but $\mathsf{pTag} \neq t$. This is incompatible, as the ConCatU.MacUpdate always writes the current computed tag in the pTag field of σ.

After these checks of compatibility, it sets $\ell = ||L||$, and parses the message M as $(m_1, m_2, \ldots, m_\ell)$ such that $M = m_1 \| m_2 \| \cdots \| m_\ell$ and $|m_i| = L[i]$. Then, in lines 29 to 33, it basically runs the procedure $\mathsf{ConCatU.SeqUpdt}((K,h),(m_1, m_2, \ldots, m_\ell))$ as described in Figure 1, i.e., it sequentially computes updates of $m_1, m_2, \ldots, m_\ell$ starting from an empty message. Finally, if the computed tag matches the input tag t, then it accepts by returning 1, else it rejects by returning 0. Based on the verification procedure described in Figure 2, it is easy to verify that ConCatU is a correct UdMAC.

4.1 Usecase, Instantiations, and Efficiency

A practical usecase: We have motivated UdMACs to be used in client-server scenarios, where a client stores a message M along with a tag t in an un-trusted server and it requires updating the message. In the context of ConCatU, the allowed update is concatenation. The client wants to compute the authentication tag for the updated message without downloading the previous message from the server. We will view the procedures of ConCatU in this context as this would help us to evaluate the efficiency of the procedure.

A motivating use case may be the following. Suppose the client records real time video footage through a surveillance camera and every hour uploads the raw video data to a server. Thus, every hour, the message in the server is updated by concatenation and by using ConCatU, one can compute the tag using just the data of one hour without needing any access to the whole data.

In such a client-server scenario. If a single message is being updated multiple times, it is best for the client to store the state of the message with itself. In the case of ConCatU, the state contains the tag and the list of updates. The size of the state for a message which has undergone k updates would be $n(k+1)$ bits, n-bits for the tag and nk bits for the list L. For our example, the size of the state grows just n-bits per hour, whereas the size of the message grows by about a GB per hour (this would depend on the resolution of the video etc). Note, n being the tag length, an acceptable value for it would be just 128.

Efficient instantiation: Storing the state, along with some extra information with the client would have efficiency implications while computing the tag. As per the procedure depicted in Figure 2, an update with a message m and state (L, pTag) requires computing $p = \mathsf{Poly}_h(X)$, and one computation of $f_k(p\|\mathsf{pTag}\|m)$. The other computations required are not significant. Note, $X = 1^n\|\mathsf{str}(L)$ and thus $|X| = (||L|| + 1)n$, hence $\mathsf{Poly}_h(X)$ is of degree at most $(||L|| + 1)$, and computing $\mathsf{Poly}_h(X)$ will require $(||L|| + 1)$ additions and multiplications in the field $\mathbb{F}_{2^n}$. But, the client may keep a bit more information to speed up this computation. Along with the state, it also stores the current value of p (see line 13 of Figure 2) for each update. When a message m_1 is first uploaded then p is computed as $(1^n)h$ and L is initialized as a list containing a single element, where $L[1] = |m_1|$. The client stores both p and L. When m_1 is updated with m_2, then instead of recomputing p the client just updates it as $p = (p + |m_2|)h$, and uses this value in line no 13 of Figure 2. This update incurs just one multiplication and one addition. This can be continued, and each update would require just one multiplication and one addition for computing p irrespective of the current state of the message.

In addition to p, the client also needs to compute $f_k(p\|\mathsf{pTag}\|m)$. The only requirement for $f_k()$ is that it has to be a variable input length PRF. Thus, $f_k()$ can be instantiated with any block-cipher based secure deterministic message authentication code like PMAC [BR02, Rog04], PAuth [CS16], secure variants of CBC-MAC [BR00, IK03, Nan09, KI03] etc. The parallelizable MACs like PMAC and PAuth are particularly suited as they are efficient across several platforms and considering block ciphers with a block size of n, they require just $\lceil |M|/n \rceil$ many block-cipher calls to authenticate a message M. Thus, a single update call for a message m will require $2 + \lceil |m|/n \rceil$ block-cipher calls, one addition, and one multiplication in $\mathbb{F}_{2^n}$.

4.2 Security of ConCatU

First, let us consider a small modification in the procedure ConCatU.MacUpdate where the procedure only returns the tag, i.e., it does not output the updated state, though it computes it. We call this modified construction as ConCatU.MU1. Our first observation is that if f is a PRF, then ConCatU.MU1 is also a PRF. Before we formalize this fixing a notation would be useful.

Definition 1. A (q, r, ℓ)-adversary is an adversary who asks q queries in the query phase where the queries are (σ_i, m_i), $1 \leq i \leq q$, each $\sigma_i = (L_i, \mathsf{pTag}_i) \in (\mathbb{N})^* \times \{0,1\}^n$ and $r = \max_i\{||L_i||\}$ and $\ell = \max_i\{|m_i|\}$. Note if $\sigma = \perp$ we consider $\sigma = (\langle\rangle, 0^n)$.

Proposition 3. *Let $\mathcal{A}$ be an arbitrary (q, r, ℓ)-PRF adversary for* ConCatU.MU1*. Then there exists a* PRF *adversary $\mathcal{B}$ for f_K such that*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{ConCatU.MU1}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{B}) + \binom{q}{2}\frac{r}{2^n}.$$

Proof. We assume that $\mathcal{A}$ does not repeat queries. As it is a (q, r, ℓ)-PRF adversary, it asks q distinct queries.

Consider the i-th query (σ_i, m_i) of $\mathcal{A}$. If $\sigma_i = \perp$ we set $L_i = \langle\rangle$, $X_i = 1^n$ and $\mathsf{pTag}_i = 0^n$. If $\sigma_i \neq \perp$ then $\sigma_i = (L_i, \mathsf{pTag}_i)$, and we set $X_i = 1^n \| \mathsf{str}(L_i)$. For the i-th query of $\mathcal{A}$ we define the string $Z_i = p_i \| \mathsf{pTag}_i \| m_i$, where

$$p_i = \mathsf{Poly}_h(X_i).$$

Note that on the i-th query, the function $f_K()$ is called with input Z_i. Let the set of all inputs to $f_K()$ during the queries of the adversary be

$$\mathbb{S} = \{Z_i : 1 \leq i \leq q\}.$$

Let us define the following events:

G0: $\mathcal{A}$ interacting with ConCatU.MU1 instantiated with $K \xleftarrow{\$} \{0,1\}^k$ and $h \xleftarrow{\$} \{0,1\}^n$ and outputting a 1.

G1: $\mathcal{A}$ interacting with ConCatU.MU1, where $f_K()$ is replaced by a uniform random function ρ drawn from the set of all functions mapping $\{0,1\}^*$ to $\{0,1\}^n$, and outputting 1.

G2: $\mathcal{A}$ interacting with ConCatU.MU1, where $f_K()$ is replaced by \$(), which on any input returns a uniform random n bit string, and outputting a 1. Note that ConCatU.MU1 where $f_K()$ is replaced by \$() essentially outputs a n-bit random string irrespective of its input.

COLL: While interacting with ConCatU.MU1, where $f_K()$ is replaced by a uniform random function ρ, $\mathcal{A}$ asks two distinct queries $i, j \in [q]$, $i \neq j$ such that $Z_i = Z_j$.

By definition of PRF advantage, we have

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{ConCatU.MU1}}(\mathcal{A}) = |\Pr[\mathsf{G0}] - \Pr[\mathsf{G2}]|\,. \tag{4}$$

By a standard reduction, we can construct an adversary $\mathcal{B}$ such that

$$|\Pr[\mathsf{G0}] - \Pr[\mathsf{G1}]| \leq \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{B}), \tag{5}$$

Also, the events G1 and G2 are same unless the event COLL occurs, hence by the difference lemma,we have

$$\Pr[\mathsf{G1}] - \Pr[\mathsf{G2}]| \leq \Pr[\mathsf{COLL}]. \tag{6}$$

Using the above equations, we have the following sequence of inequalities.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{ConCatU.MU1}}(\mathcal{A}) &= |\Pr[\mathsf{G0}] - \Pr[\mathsf{G2}]| \\
&= |(\Pr[\mathsf{G0}] - \Pr[\mathsf{G1}]) + (\Pr[\mathsf{G1}] - \Pr[\mathsf{G2}])| \\
&\leq |(\Pr[\mathsf{G0}] - \Pr[\mathsf{G1}])| + |(\Pr[\mathsf{G1}] - \Pr[\mathsf{G2}])| \\
&\leq \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{B}) + \Pr[\mathsf{COLL}].
\end{aligned}$$

We are only left to bound $\Pr[\mathsf{COLL}]$. For which we consider two distinct queries (σ_i, m_i), (σ_j, m_j) for which $\rho()$ gets called with Z_i and Z_j respectively. We consider the following cases:

Case 1. $m_i \neq m_j$. In this case $\Pr[Z_i = Z_j] = 0$.

Case 2. $m_i = m_j$. As $(\sigma_i, m_i) \neq (\sigma_j, m_j)$, thus $\sigma_i \neq \sigma_j$. We have to consider two sub-cases here. If $\mathsf{pTag}_i \neq \mathsf{pTag}_j$, then $\Pr[Z_i = Z_j] = 0$. If $\mathsf{pTag}_i = \mathsf{pTag}_j$, then $L_i \neq L_j$. Which implies $X_i \neq X_j$ and by Proposition 2 we have $\Pr[Z_i = Z_j] = \Pr[p_i = p_j] \leq (\max(||L_i||, ||L_j||))/2^n$.

Thus, by the above two cases, the fact that $\mathbb{S}$ contains q elements, and the union bound, we have

$$\Pr[\mathsf{COLL}] \leq \binom{q}{2}\frac{r}{2^n}. \tag{7}$$

This completes the proof. □

The following Theorem claims security of $\mathsf{ConCatU}$ against a forgery adversary.

Theorem 2. *Let $\mathcal{A}$ be an arbitrary (q, r, ℓ)-forgery adversary attacking $\mathsf{ConCatU}$, also let the forgery produced by $\mathcal{A}$ be (σ, m, t), where $\sigma = (L, \mathsf{ptag})$, where $||L|| = r^*$. Then there exists a PRF adversary $\mathcal{B}$ for f such that*

$$\mathsf{Adv}^{\mathsf{Uauth}}_{\mathsf{ConCatU}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{B}) + \frac{1 + (q+1)r^*}{2^n} + \frac{q\max\{r, r^*\}}{2^n} + \binom{q}{2}\frac{r}{2^n}.$$

Proof. We denote the real scheme instantiated with $(K, h) \xleftarrow{\$} \{0,1\}^k \times \{0,1\}^n$ by $\mathsf{ConCatU}(f_K, h)$. By $\mathsf{ConCatU}(\rho, h)$ we denote the update and verification algorithms of $\mathsf{ConCatU}$, where the function $f_K()$ is replaced by a uniform random function ρ drawn from the set of all functions mapping $\{0,1\}^*$ to $\{0,1\}^n$.

We call the event of $\mathcal{A}$ attacking $\mathsf{ConCatU}(f_K, h)$ and producing a valid forgery which passes verification by $\mathsf{Succ}^{(f_K,h)}(\mathcal{A})$. Similarly, we call the event of $\mathcal{A}$ attacking $\mathsf{ConCatU}(\rho, h)$ and producing a valid forgery which passes verification by $\mathsf{Succ}^{(\rho,h)}(\mathcal{A})$.

By an easy reduction, we can construct a PRF adversary $\mathcal{B}$ such that

$$\mathsf{Adv}^{\mathsf{Uauth}}_{\mathsf{ConCatU}}(\mathcal{A}) = \Pr\left[\mathsf{Succ}^{(f_K,h)}(\mathcal{A})\right] \leq \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{B}) + \Pr\left[\mathsf{Succ}^{(\rho,h)}(\mathcal{A})\right]. \tag{8}$$

Now we try to bound $\Pr[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}]$.

We assume that $\mathcal{A}$ does not repeat any query and makes q distinct queries. Let its ith query be (σ_i, m_i), and for such a query it gets a response $(\widetilde{\sigma}_i, t_i)$. Thus, the query profile of $\mathcal{A}$ is

$$\mathbb{P}_{\mathcal{A}} = \{(\sigma_i, m_i, \widetilde{\sigma}_i, t_i) : 1 \leq i \leq q\}.$$

Also, we denote

$$\mathbb{Q}_{\mathcal{A}} = \{(\sigma_i, m_i) : 1 \leq i \leq q\},$$

the set of distinct queries made by $\mathcal{A}$.

As in the proof of Proposition 3, consider the i-th query (σ_i, m_i). If $\sigma_i = \perp$ we set $L_i = \langle\rangle$, $X_i = 1^n$ and $\mathsf{pTag}_i = 0^n$. If $\sigma_i \neq \perp$ then $\sigma_i = (L_i, \mathsf{pTag}_i)$, and we set $X_i = 1^n \| \mathsf{str}(L_i)$. For the i-th update query of $\mathcal{A}$ we define the string $Z_i = p_i \| \mathsf{pTag}_i \| m_i$, where

$$p_i = \mathsf{Poly}_h(X_i).$$

Note that on the i-th query, the random function ρ is called on the string Z_i. Let the set of all inputs to ρ during the queries of the adversary be

$$\mathbb{S} = \{Z_i : 1 \leq i \leq q\}.$$

Claim. *Let* $\mathsf{E0}$*, be the event that there is a collision in the set* $\mathbb{S}$*. Then,*

$$\Pr[\mathsf{E0}] \leq \binom{q}{2}\frac{r}{2^n}. \tag{9}$$

As the event E_0 is exactly the event COLL defined in the proof of Proposition 3, hence Equation (7) proves the above claim.

Let the valid forgery produced by $\mathcal{A}$ be (σ^*, m^*, t^*). It is required that $\sigma^* \neq \perp$, as otherwise the success probability of $\mathcal{A}$ will be zero. Let $\sigma^* = (L^*, \mathsf{pTag}^*)$, where $L^* = \langle \ell_1^*, \ell_2^*, \ldots, \ell_{r^*}^* \rangle$, and

$$\ell_1^* + \ell_2^* + \cdots + \ell_{r^*}^* = \ell^* = |m^*|.$$

Also, let $(m_1^*, m_2^*, \ldots, m_{r^*}^*) = \mathsf{Parse}(m^*, L^*)$. Also define $L_0^* = \langle\rangle$ and for $1 \leq i \leq r^*$, $L_i^* = \mathsf{append}(L_{i-1}^*, \ell_i^*)$.

For ease of exposition, we define some variables. For $0 \leq i \leq r^*$ we define

$$X^{(i)} = \begin{cases} 1^n & \text{if } i = 0 \\ X^{(i-1)} \| \mathsf{bin}_n(\ell_{i-1}) & \text{otherwise,} \end{cases} \tag{10}$$

and for $1 \leq j \leq r^*$

$$p^{(j)} = \mathsf{Poly}_h\left(X^{(j-1)}\right).$$

For $0 \leq i \leq r^*$

$$\mathsf{tag}^{(i)} = \begin{cases} 0^n & \text{if } i = 0 \\ f_K\left(p^{(i)} \| \mathsf{tag}^{(i-1)} \| m_i^*\right) & \text{otherwise,} \end{cases} \tag{11}$$

and for $1 \leq i \leq r^*$,

$$Y^{(i)} = p^{(i)} \| \mathsf{tag}^{(i-1)} \| m_i^*.$$

and let $\mathbb{Y} = \{Y^{(i)} : 1 \leq i \leq r^*\}$. Note that while verification of the forgery produced by $\mathcal{A}$, ρ gets evaluated on the strings in $\mathbb{Y}$.

Now, we define a set of queries $\mathbb{Q} = \{Q_i : 1 \leq i \leq r^*\}$, where

$$Q_i = \begin{cases} (\perp, m_i^*) & \text{if } i = 1 \\ \left((L_{i-1}^*, \mathsf{tag}^{(i-1)}), m_i^*\right) & \text{if } i > 1. \end{cases}$$

Note that, if Q_i is given as an update query, then the function ρ gets evaluated on the string $Y^{(i)}$. Also this query set $\mathbb{Q}$ is defined by the forgery attempt of $\mathcal{A}$.

As the forgery produced by $\mathcal{A}$ is a valid forgery, hence according to the definition of a valid forgery, $\mathcal{A}$ has not asked the queries $Q_1, Q_2, \ldots, Q_{r^*}$ in the query phase in the same order.

Definition 2. Given the forgery of $\mathcal{A}$ the query set $\mathbb{Q}_{\mathcal{A}}$ is called out of order, if there exists a $i \in [r^* - 1]$, such that query Q_{i+1} was asked before query Q_i.

Definition 3. Given the forgery of $\mathcal{A}$ the query set $\mathbb{Q}_{\mathcal{A}}$ is called incomplete, if there exists a $i \in [r^*]$, such that query $Q_i \notin \mathbb{Q}_{\mathcal{A}}$.

Thus, for the forgery produced by $\mathcal{A}$ to be valid, its query set $\mathbb{Q}_{\mathcal{A}}$ has to be either out of order or incomplete.

Claim. *Let* $\mathsf{E1}$ *be the event that the query set* $\mathbb{Q}_{\mathcal{A}}$ *is out of order. Then,*

$$\Pr[\mathsf{E1}] \leq \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n}. \tag{12}$$

Proof of Claim: Let for $i \in [r^* - 1]$ the query Q_{i+1} was asked before Q_i. For these queries, the random function ρ was evaluated on Z_{i+1} and Z_i respectively in the same order. Assuming that there was no collision on $\mathbb{S}$, we have $Z_i \neq Z_{i+1}$. Also,

$$\begin{aligned} Z_{i+1} &= p_i + 1 \| \mathsf{pTag}_{i+1} \| m_{i+1}, \\ Z_i &= p_i \| \mathsf{pTag}_i \| m_i, \end{aligned}$$

and $\mathsf{pTag}_{i+1} = \rho(Z_i)$. As all Z_i's are distinct thus $\mathcal{A}$ has not seen the output of $\rho(Z_i)$ before it asked the query Q_i, as ρ is a random function. Thus the probability that $\mathcal{A}$ could have asked the query $Q_{i+1} = ((L_i, t), m_i)$ where $t = \rho(Z_i)$ is at most $1/2^n$. Hence using the union bound over all possible $i \in [r^*]$ which are out of order, we have $\Pr[\mathsf{E1}|\neg\mathsf{E0}] \leq r^*/2^n$. Hence,

$$\begin{aligned} \Pr[\mathsf{E1}] &\leq \Pr[\mathsf{E1}|\neg\mathsf{E0}] + \Pr[\mathsf{E0}] \\ &\leq \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n}. \end{aligned} \tag{13}$$

This completes the proof of the claim.

Now, we assume that the query set $\mathbb{Q}_{\mathcal{A}}$ is not out of order, then if the forgery of $\mathcal{A}$ is valid, then $\mathbb{Q}_{\mathcal{A}}$ must be incomplete, i.e., there exists a $j^* \in [r^*]$ such that Q_{j^*} is not in $\mathbb{Q}_{\mathcal{A}}$. Thus we have the following claim.

Claim. *For $j^* \leq k \leq r^*$, let $\mathsf{E2}^{(k)}$ be the event that $Y^{(k)} \in \mathbb{S}$. Then*

$$\Pr\left[\mathsf{E2}^{(j^*)}\right] \leq \frac{q \max\{r, j^*\}}{2^n} + \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n}. \tag{14}$$

and for $j^* + 1 \leq k \leq r^*$,

$$\Pr\left[\mathsf{E2}^{(k)}|\neg\mathsf{E2}^{(k-1)}\right] \leq \frac{q}{2^n}. \tag{15}$$

For a proof of Equation (14), notice that, if $Q_{j^*} = ((L_j^*, \mathsf{pTag}_{j^*}), m_{j^*})$ was never asked in the query phase then $\Pr\left[Y^{(j^*)} = Z_i\right]$ for some $i \in [q]$ is at most $\max\{r, j^*\}/2^n$. This follows from Proposition 2. Thus as there are q many Z_i's in $\mathbb{S}$, by the union bound we have, $\Pr\left[\mathsf{E2}^{(j^*)}|\neg\mathsf{E1}\right]$ is at most $q\max\{r, j^*\}/2^n$. Hence,

$$\begin{aligned} \Pr\left[\mathsf{E2}^{(j^*)}\right] &\leq \Pr\left[\mathsf{E2}^{(j^*)}|\neg\mathsf{E1}\right] + \Pr\left[\mathsf{E1}\right] \\ &\leq \frac{q \max\{r, j^*\}}{2^n} + \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n}. \end{aligned}$$

For Equation (15), we observe that if $Y^{(k-1)}$, not in $\mathbb{S}$, then $\mathsf{tag}^{(k-1)}$ is a n-bit uniform random string, and thus $Y^{(k)} = p^{(k)} \| \mathsf{tag}^{(k-1)} \| m_k^*$ is equal to any $Z_i \in \mathbb{S}$ with probability at most $1/2^n$, and hence

$$\Pr\left[\mathsf{E2}^{(k)}|\neg\mathsf{E2}^{(k-1)}\right] \leq \frac{q}{2^n}.$$

as claimed.

Finally, we would like to bound the probability of success of $\mathcal{A}$, i.e., we need to bound $\Pr\left[\mathsf{Succ}_{\mathcal{A}}^{(\rho,h)}\right]$.

$$\begin{aligned} \Pr\left[\mathsf{Succ}_{\mathcal{A}}^{(\rho,h)}\right] &= \Pr\left[t^* = \mathsf{tag}^{(r^*)}\right] \\ &\leq \Pr\left[t^* = \mathsf{tag}^{(r^*)}|\neg\mathsf{E2}^{(r^*)}\right] + \Pr\left[\mathsf{E2}^{(r^*)}\right] \\ &\leq \frac{1}{2^n} + \Pr\left[\mathsf{E2}^{(r^*)}\right]. \end{aligned} \tag{16}$$

The last inequality is due to the fact that if $Y^{(r^*)} \notin \mathbb{S}$, then $\mathsf{tag}^{(r^*)}$ is an uniform random n-bit string and t^* is a fixed string fixed by $\mathcal{A}$ before $\rho()$ was evaluated on $Y^{(r^*)}$.

Now, by using repeated conditioning, we have

$$\begin{aligned}\Pr\left[\mathsf{E2}^{(r^*)}\right] &= \sum_{k=0}^{r^*-j^*} \Pr\left[\mathsf{E2}^{(r^*-k)} | \neg \mathsf{E2}^{(r^*-(k-1))}\right] + \Pr\left[\mathsf{E2}^{(j^*)}\right] \\ &\leq \frac{(r^*-j^*)q}{2^n} + \frac{q\max\{r,j^*\}}{2^n} + \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n} \\ &< \frac{r^*q}{2^n} + \frac{q\max\{r,r^*\}}{2^n} + \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n}. \end{aligned} \tag{17}$$

Finally using Equations (8),(16) and (17), we get

$$\begin{aligned}\Pr\left[\mathsf{Succ}_{\mathcal{A}}^{(f_K,h)}\right] &\leq \mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{B}) + \frac{1}{2^n} + \frac{r^*q}{2^n} + \frac{q\max\{r,r^*\}}{2^n} + \frac{r^*}{2^n} + \binom{q}{2}\frac{r}{2^n} \\ &= \mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{B}) + \frac{1+(q+1)r^*}{2^n} + \frac{q\max\{r,r^*\}}{2^n} + \binom{q}{2}\frac{r}{2^n}.\end{aligned}$$

□

5 XoRU: An Updatable MAC for ⊕ Difference

Here we discuss a scheme for UdMAC where the update function is $\oplus$, i.e., on a message M, if an update m is applied then if results in the message $M \oplus m$. We always assume that the length of the updated message is at least the size of the original message. In particular, it is important to note the following points:

1. If the current message is M and the intended updated message is M', where $|M| = |M'|$, then the update request should be with the string $m = M \oplus M'$, and here $|m| = |M|$.
2. If the current message is M and the intended updated message is M' where $|M| < |M'|$, then an update request should be issued with $m = (M \| 0^{|M'|-|M|}) \oplus M'$. In this case $|m| = |M'|$.
3. If the current message is M and the intended updated message is M' where $|M| > |M'|$, such an update is not allowed.

If a sequence of updates $m_1, m_2, \ldots, m_s \in \mathcal{M}$ is applied to a given message $M \in \mathcal{M}$, then $|M| \leq |m_1| \leq |m_2| \leq \cdots \leq |m_s|$.

Unlike ConCatU, XoRU is nonce based, i.e., its MacUpdate and MacVerify procedures takes in a nonce in addition to the inputs as specified in the syntax described in Section 3. In particular we would refer to the update and verification procedures of XoRU by $\mathsf{XoRU.MacUpdate}(K, m, N, \sigma)$ and $\mathsf{XoRU.MacVerify}(K, m, N, \sigma, t)$ respectively.

For XoRU we consider the state space to be $\Sigma = \{0,1\}^{3n} \cup \{\perp\}$, where $\sigma \in \Sigma$ is either $\perp$ or can be parsed as $\sigma = (\mathsf{pNonce}, \mathsf{pLen}, \mathsf{pTag})$, where $\mathsf{pNonce}, \mathsf{pLen}, \mathsf{pTag} \in \{0,1\}^n$.

The MacUpdate and Verify procedures for XoRU are described in Figure 3. Next we note down some elementary observations regarding the construction.

Proposition 4. *Let* $m_1, m_2, \ldots, m_s \in \mathcal{M}$ *and* $\mathsf{XoRU.SeqUpdt}((K,h), m_1, m_2, \ldots, m_s)) = (\sigma_s, t_s)$. *Let* $\sigma_s = (\mathsf{N}, \mathsf{L}, \mathsf{T})$. *Then*

1. $\mathsf{L} = |m_s|$
2. $t_s = \mathsf{T} = \mathsf{Poly}_h(m_1 \oplus m_2 \oplus \cdots \oplus m_s) \oplus \mathsf{L}h^{\lceil \mathsf{L}/n \rceil + 1} \oplus E_K(\mathsf{N})$.

Note that, point (2) in Proposition 4 essentially gives the correctness of XoRU.

$\underline{\mathsf{XoRU.MacUpdate}((K,h),m,N,\sigma)}$

01: **if** $\sigma = \perp$ **then**
02: $\quad \mathsf{tag} \leftarrow \mathsf{Poly}_h(m) \oplus |m| h^{\lceil |m|/n \rceil + 1} \oplus E_K(N)$
03: $\quad \mathsf{pLen} \leftarrow 0$
04: **else**
05: $\quad (\mathsf{pNonce}, \mathsf{pLen}, \mathsf{pTag}) \leftarrow \sigma$
06: $\quad \ell_1 \leftarrow \lceil \frac{\mathsf{pLen}}{n} \rceil + 1$
07: $\quad \ell_2 \leftarrow \left\lceil \frac{|m|}{n} \right\rceil + 1$
08: $\quad$ **if** $\mathsf{pLen} = |m|$ **then**
09: $\quad\quad \mathsf{tag} \leftarrow \mathsf{pTag} \oplus \mathsf{Poly}_h(m) \oplus E_K(\mathsf{pNonce})$
10: $\quad\quad \mathsf{tag} \leftarrow \mathsf{tag} \oplus E_K(N)$
11: $\quad$ **else**
12: $\quad\quad \mathsf{tag} \leftarrow \mathsf{pTag} \oplus h^{\ell_1}\mathsf{pLen} \oplus E_K(\mathsf{pNonce})$
13: $\quad\quad \mathsf{tag} \leftarrow \mathsf{tag} \oplus \mathsf{Poly}_h(m) \oplus h^{\ell_2}|m| \oplus E_K(N)$
14: $\quad$ **end if**
15: **end if**
16: $\sigma \leftarrow (N, \max\{\mathsf{pLen}, |m|\}, \mathsf{tag})$
17: **return** (σ, tag)

$\underline{\mathsf{XoRU.MacVerify}((K,h),m,N,\sigma,t)}$

21: **if** $\sigma = \perp$ **then**
22: $\quad$ **return** 0
23: **end if**
24: $(\mathsf{pNonce}, \mathsf{pLen}, \mathsf{pTag}) \leftarrow \sigma$
25: **if** $\mathsf{pLen} \neq |m|$ **then**
26: $\quad$ **return** 0
27: **end if**
28: $\ell \leftarrow \lceil \frac{\mathsf{pLen}}{n} \rceil + 1$
29: $\mathsf{tag} \leftarrow \mathsf{Poly}_h(m) \oplus h^{\ell}\mathsf{pLen}$
30: $\mathsf{tag} \leftarrow \mathsf{tag} \oplus E_K(\mathsf{pNonce})$
31: **if** $\mathsf{tag} \neq t$ **then**
32: $\quad$ **return** 0
33: **else**
34: $\quad$ **return** 1
35: **end if**

Figure 3: Specification of XoRU.MacUpdate and XoRU.MacVerify.

5.1 Efficiency of XoRU

The bulk computation for update takes place in lines 09 and line 12 or line 13 of the procedure described in Figure 3. We first give a general estimate of the computation required.

First, we observe that if the condition in line number 08 of Figure 3 is satisfied then the updated tag would be computed as

$$\mathsf{tag} = \mathsf{pTag} \oplus \mathsf{Poly}_h(m) \oplus E_K(\mathsf{pNonce}) \oplus E_K(N).$$

Thus, an update satisfying the condition in Line 08 will require the computation of $\mathsf{Poly}_h(m)$, which is just an evaluation of a degree ℓ_2 polynomial with no constant term. If the Horner's rule is used for evaluating the polynomial, it will take ℓ_2 multiplications and $\ell_2 - 1$ additions in $\mathbb{F}_{2^n}$. In addition to the computation of $\mathsf{Poly}_h(m)$ three more additions in $\mathbb{F}_{2^n}$ and two calls to the PRF $E_K()$ (which can be instantiated with a block-cipher like AES) would be required. If we assume the cost of one block-cipher call to be T_{AES} and the cost of one multiplication and one addition in $\mathbb{F}_{2^n}$ to be T_{add} and T_{mult} respectively, then the total cost if $\mathsf{pLen} > |m|$ (the condition in line 08) will be

$$T_{\mathsf{add}}(\ell_2 + 2) + T_{\mathsf{mult}}\ell_2 + 2T_{\mathsf{AES}}.$$

Now, let us analyze the case where the condition in line 08 is not satisfied, i.e., we have $\mathsf{pLen} < |m|$. Note, in this case computation in line 08 will require two additions in $\mathbb{F}_{2^n}$, a computation of h^{ℓ_1} and a multiplication in $\mathbb{F}_{2^n}$. If the square and multiply algorithm for exponentiation is used then computation of h^{ℓ_1} will require at most $\log(\ell_1)$ squarings and $\log(\ell_1)$ multiplications in $\mathbb{F}_{2^n}$. Though squares in fields of characteristic 2 can be computed much more efficiently than multiplications in various platforms, for simplicity, we consider the cost of squaring to be the same as that of multiplication. With such an assumption line 12 can be computed with $2\log(\ell_1)$ multiplications. For the computation in line 12, $\mathsf{Poly}_h(m) \oplus h^{\ell_2}|m|$ can be computed with ℓ_2 additions and $\ell_2 + 1$ multiplications in $\mathbb{F}_{2^n}$. Thus summarizing, we have the total cost

$$(\ell_2 + 4)\, T_{\mathsf{add}} + (2\log(\ell_1) + \ell_2 + 1)\, T_{\mathsf{mult}} + 2T_{\mathsf{AES}}.$$

A Special Case: XoRU can be used to support the concatenation of messages by proper update requests. For example, if the current message is M and the intended message is $M\|m$, then M can be updated with an update request of $0^{|M|}\|m$. Any scheme, which will require such kinds of updates, in that case, both ConCatU and XoRU can be used.

Note, for each update, if $|m| = kn$ where k is a small constant, then the client can store h^{ℓ_1} where $\ell_1 = \lceil \mathsf{pLen}/n \rceil + 1$ for each updated message, and also store $\mathsf{Poly}_h(M)$ for the current message M. This will require just $(2k+2)$ multiplications for computing each updated tag and also for updating h^{ℓ_1} (as ℓ_1 increases by k in each update), and the polynomial of the current message.

5.2 Security of XoRU

We claim security of XoRU against a *nonce respecting* adversary, i.e., for the mac-update queries, an adversary never repeats a nonce.

For technical convenience, we consider a little change in the XoRU.MacUpdate construction, where the procedure returns only the tag instead of (σ, tag). We call this construction as $\mathsf{XoRU.MacUpdate}^{(1)}$. This change has no effect on security as the updated state σ can be constructed by the adversary.

We define the rnd-advantage for a nonce respecting adversary $\mathcal{A}$ attacking XoRU as the following:

$$\mathsf{Adv}^{\mathsf{rnd}}_{\mathsf{XoRU}}(\mathcal{A}) = \left| \Pr\left[K, h \xleftarrow{\$} \{0,1\}^n : \mathcal{A}^{\mathsf{XoRU.MacUpdt}^{(1)}}(K, h, \cdot, \cdot, \cdot) = 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} = 1\right] \right|,$$

where the $\$(\cdot,\cdot,\cdot)$ oracle on invocation with any $(\sigma, N, m) \in \Sigma \cup \{\perp\} \times \{0,1\}^n \times \mathcal{M}$ returns a uniform random n-bit string.

We claim that an efficient nonce-respecting adversary interacting with the XoRU. $\mathsf{MacUpdate}^{(1)}$ oracle instantiated with keys $(K, h) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^n$ cannot distinguish it from the oracle \$ if $E_K : \{0,1\}^n \to \{0,1\}^n$ is a pseudorandom function family. In other words,

Proposition 5. *For any efficient nonce respecting* rnd *adversary* $\mathcal{A}$ *attacking* XoRU*, there exists an (almost) equally efficient PRF adversary* $\mathcal{B}$ *for the function family* $E_K : \{0,1\}^n \to \{0,1\}^n$*, such that*

$$\mathsf{Adv}^{\mathsf{rnd}}_{\mathsf{XoRU.MacUpdate}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{E}(\mathcal{B}).$$

The proof of the above proposition is an easy reduction. The main observation that leads to the claim is that if $E_K()$ in the construction of $\mathsf{XoRU.MacUpdate}^{(1)}$ is replaced by a uniform random function then the procedure essentially outputs uniform random strings.

Now, we are in a position to state the security of XoRU.

Theorem 3. *Let* $\mathcal{A}$ *be any efficient forging adversary for* XoRU *who asks* q *many update queries and finally outputs a valid forgery* (M^*, σ^*, t^*)*, then there exists an (almost) equally efficient PRF adversary* $\mathcal{B}$ *attacking* E_K *such that*

$$\mathsf{Adv}^{\mathsf{Uauth}}_{\mathsf{XoRU}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}(\mathcal{B}) + \frac{1}{2^n}\left(\frac{|M^*|}{n} + 2\right).$$

Proof. Let $\mathsf{XoRU}(K, h)$ be the real update and verification algorithms described in Figure 3 instantiated with uniform random n-bit keys K, h. Let $\mathsf{XoRU}(\rho, h)$ denote the update and verification algorithms where $E_K()$ in Figure 3 is replaced by a function ρ sampled uniformly at random from the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$.

We call the event of $\mathcal{A}$ attacking $\mathsf{XoRU}(K, h)$ and producing a valid forgery which passes verification by $\mathsf{Succ}^{(K,h)}_{\mathcal{A}}$. Similarly, we call the event of $\mathcal{A}$ attacking $\mathsf{XoRU}(\rho, h)$ and producing a valid forgery which passes verification by $\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}$.

By an easy reduction, we can construct a PRF adversary $\mathcal{B}$ such that

$$\mathsf{Adv}^{\mathsf{Uauth}}_{\mathsf{XoRU}}(\mathcal{A}) \;=\; \Pr\left[\mathsf{Succ}^{(K,h)}_{\mathcal{A}}\right] \leq \mathsf{Adv}^{\mathsf{prf}}(\mathcal{B}) + \Pr\left[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}\right]. \tag{18}$$

Now we try to bound $\Pr\left[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}\right]$.

First, note that according to Proposition 5, when $\mathcal{A}$ asks update queries to $\mathsf{XoRU}(\rho, h)$ (s)he gets as response random strings in place of the real tags. Thus, in the query phase, irrespective of the queries it asks, $\mathcal{A}$ gets no information regarding the real scheme.

Finally $\mathcal{A}$ outputs a valid forgery (M^*, σ^*, t^*). We consider $\sigma^* = (N^*, L^*, \mathsf{tag}^*)$. We have the following cases to consider:

Case I: There was no query with the nonce N^*: Consider the verification procedure described in Figure 3 where $E_K()$ is replaced by $\rho()$. Thus, in this case, in the verification procedure, $\rho(N^*)$ would be a uniform random string. Thus, the tag computed in line 29 and 30 of the procedure described in Figure 3 will in turn be a uniform random string. Thus, as t^* is a fixed sting, $\Pr[\mathsf{tag} = t^*] = 1/2^n$, Thus,

$$\Pr\left[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}\right] = \frac{1}{2^n}.$$

Case II: There was a query with the nonce N^*: In this case, $\rho(N^*)$ was fixed, say to t when a query with nonce N^* was asked by $\mathcal{A}$. Thus the tag computed by the verification algorithm is

$$\mathsf{tag} = \mathsf{Poly}_h(M^*) \oplus h^{\lceil L^*/n \rceil + 1} L^* \oplus t.$$

For $\mathcal{A}$ to be successful it is required that $\mathsf{tag} = t^*$. Thus, in this case, we have

$$\begin{aligned} \Pr\left[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}\right] &= \Pr\left[\mathsf{tag} = t^*\right] \\ &= \Pr\left[\mathsf{Poly}_h(M^*) \oplus h^{\lceil L^*/n \rceil + 1} L^* \oplus t \oplus t^* = 0\right]. \end{aligned}$$

The above probability is computed on the randomness of the choice of h. As,

$$p(h) = \mathsf{Poly}_h(M^*) \oplus h^{\lceil L^*/n \rceil + 1} L^* \oplus t \oplus t^*,$$

is a nonzero polynomial of degree $\lceil L^*/n \rceil + 1$, hence there are at most $\lceil L^*/n \rceil + 1$ many values of h which makes $p(h) = 0$. Thus, we have

$$\begin{aligned} \Pr\left[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}\right] &= \Pr[\mathsf{tag} = t^*] \\ &\leq \frac{1}{2^n}\left(\left\lceil \frac{L^*}{n} \right\rceil + 1\right) \\ &= \frac{1}{2^n}\left(\left\lceil \frac{|M^*|}{n} \right\rceil + 1\right). \end{aligned}$$

Now, combining cases I and II and using the union bound, we have

$$\Pr\left[\mathsf{Succ}^{(\rho,h)}_{\mathcal{A}}\right] \leq \frac{1}{2^n}\left(\frac{|M^*|}{n} + 1\right) + \frac{1}{2^n} \tag{19}$$

Finally, using Equations (18) and (19), we get

$$\mathsf{Adv}^{\mathsf{Uauth}}_{\mathsf{XoRU}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}(\mathcal{B}) + \frac{1}{2^n}\left(\frac{|M^*|}{n} + 2\right),$$

as desired. □

6 Conclusion

We proposed a new type of state-full MAC: UdMAC, which can be used to update the authentication tag of a message and compute the correct tag for the updated message only with access to the update and a small state about the message. We defined the security of UdMACs and finally proposed two constructions. This class of MAC can be used in a client server model where at the time of updating the message, the client does not have access to the current message stored with the server.

Acknowledgement

Authors like to thank Jean-Philippe Aumasson for pointing out an error in the previous version of the paper.

References

[ABK20] Vivek Arte, Mihir Bellare, and Louiza Khati. Incremental cryptography revisited: Prfs, nonces and modular design. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020, 21st International Conference on Cryptology in India, Bangalore, India, December 13–16, 2020, Proceedings*, volume 12578 of *Lecture Notes in Computer Science*, pages 576–598, Berlin, Heidelberg, 2020. Springer. doi:10.1007/978-3-030-65277-7_26.

[BEKS20] Dan Boneh, Saba Eskandarian, Sam Kim, and Maurice Shih. Improving speed and security in updatable encryption schemes. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020, 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 559–589, Berlin, Heidelberg, 2020. Springer. doi:10.1007/978-3-030-64840-4_19.

[BGG94] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 216–233, Berlin, Heidelberg, 1994. Springer. doi:10.1007/3-540-48658-5_22.

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC 1995), Las Vegas, Nevada, USA, May 29–June 1, 1995*, pages 45–56, New York, 1995. ACM. doi:10.1145/225058.225080.

[BKY01] Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental unforgeable encryption. In Mitsuru Matsui, editor, *Fast Software Encryption, 8th International Workshop, FSE 2001, Yokohama, Japan, April 2–4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 109–124, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-45473-X_9.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A.

Garay, editors, *Advances in Cryptology - CRYPTO 2013, 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2013, Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-40041-4_23.

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11–15, 1997, Proceedings*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192, Berlin, Heidelberg, 1997. Springer. doi:10.1007/3-540-69053-0_13.

[BR00] John Black and Phillip Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20–24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215, Berlin, Heidelberg, 2000. Springer. doi:10.1007/3-540-44598-6_12.

[BR02] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28–May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 384–397, Berlin, Heidelberg, 2002. Springer. doi:10.1007/3-540-46035-7_25.

[BS23] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. 2023. URL: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf.

[CRS+21] Valerio Cini, Sebastian Ramacher, Daniel Slamanig, Christoph Striecks, and Erkan Tairi. Updatable signatures and message authentication codes. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021, 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10–13, 2021, Proceedings, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 691–723, Berlin, Heidelberg, 2021. Springer. doi:10.1007/978-3-030-75245-3_25.

[CS16] Debrup Chakraborty and Palash Sarkar. On modes of operations of a block cipher for authentication and authenticated encryption. *Cryptogr. Commun.*, 8(4):455–511, 2016. URL: https://doi.org/10.1007/s12095-015-0153-6, doi:10.1007/S12095-015-0153-6.

[CW77] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, 1977.

[EPRS17] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017, 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 98–129, Berlin, Heidelberg, 2017. Springer. doi:10.1007/978-3-319-63697-9_4.

[Fis97a] Marc Fischlin. Incremental cryptography and memory checkers. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11–15, 1997, Proceedings*, volume 1233 of *Lecture Notes in Computer Science*, pages 293–408, Berlin, Heidelberg, 1997. Springer. doi:10.1007/3-540-69053-0_27.

[Fis97b] Marc Fischlin. Lower bounds for the signature size of incremental schemes. In *38th Annual Symposium on Foundations of Computer Science (FOCS '97), Miami Beach, Florida, USA, October 19–22, 1997*, pages 438–447, Piscataway, NJ, 1997. IEEE Computer Society. doi:10.1109/SFCS.1997.646132.

[GP17] Sanjam Garg and Omkant Pandey. Incremental program obfuscation. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017, 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 193–223, Berlin, Heidelberg, 2017. Springer. doi:10.1007/978-3-319-63715-0_7.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: one-key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24–26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153, Berlin, Heidelberg, 2003. Springer. doi:10.1007/978-3-540-39887-5_11.

[KI03] Kaoru Kurosawa and Tetsu Iwata. TMAC: two-key CBC MAC. In Marc Joye, editor, *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13–17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*, pages 33–49, Berlin, Heidelberg, 2003. Springer. doi:10.1007/3-540-36563-X_3.

[KV18] Louiza Khati and Damien Vergnaud. Analysis and improvement of an authentication scheme in incremental cryptography. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018, 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 50–70, Berlin, Heidelberg, 2018. Springer. doi:10.1007/978-3-030-10970-7_3.

[MGS15] Hristina Mihajloska, Danilo Gligoroski, and Simona Samardjiska. Reviving the idea of incremental cryptography for the zettabyte era use case: Incremental hash functions based on SHA-3. In Jan Camenisch and Dogan Kesdogan, editors, *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, volume 9591 of *Lecture Notes in Computer Science*, pages 97–111, Berlin, Heidelberg, 2015. Springer. doi:10.1007/978-3-319-39028-4_8.

[MPRS12] Ilya Mironov, Omkant Pandey, Omer Reingold, and Gil Segev. Incremental deterministic public-key encryption. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012, 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15–19, 2012, Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 628–644, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-29011-4_37.

[Nan09] Mridul Nandi. Fast and secure CBC-Type MAC algorithms. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009,*

Leuven, Belgium, February 22–25, 2009, Revised Selected Papers, volume 5665 of *Lecture Notes in Computer Science*, pages 375–393, Berlin, Heidelberg, 2009. Springer. doi:10.1007/978-3-642-03317-9_23.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pages 16–31, 2004.

[SY15] Yu Sasaki and Kan Yasuda. A new mode of operation for incremental authenticated encryption with associated data. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015, 22nd International Conference, Sackville, NB, Canada, August 12–14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 397–416, Berlin, Heidelberg, 2015. Springer. doi:10.1007/978-3-319-31301-6_23.

[WC79] Mark N Wegman and J Lawrence Carter. New classes and applications of hash functions. In *20th Annual Symposium on Foundations of Computer Science (SFCS 1979)*, pages 175–182. IEEE, 1979.