A New Look at Counters: Don't Run Like Marathon in a Hundred Meter Race

Avijit Dutta, Ashwin Jha and Mridul Nandi

Abstract—In cryptography, counters (classically encoded as bit strings of fixed size for all inputs) are employed to prevent collisions on the inputs of the underlying primitive which helps us to prove the security. In this paper we present a unified notion for counters, called *counter function family*. Based on this generalization we identify some necessary and sufficient conditions on counters which give (possibly) simple proof of security for various counter-based cryptographic schemes. We observe that these conditions are trivially true for the classical counters. We also identify and study two variants of the classical counter which satisfy the security conditions. The first variant has message length dependent counter size, whereas the second variant uses universal coding to generate message length independent counter size. Furthermore, these variants provide better performance for shorter messages. For instance, when the message size is 2^{19} bits, AES-LightMAC with 64-bit (classical) counter takes 1.51 cycles per byte (cpb), whereas it takes 0.81 cpb and 0.89 cpb for the first and second variant, respectively. We benchmark the software performance of these variants against the classical counter by implementing them in MACs and HAIFA hash function.

Index Terms—counter, XOR-MAC, protected counter sum, LightMAC, HAIFA, universal hash, cryptography, AES-NI.

1 Introduction

In cryptography, counters are classically viewed as sequences of binary strings, $\langle 0 \rangle_s$, $\langle 1 \rangle_s, \ldots, \langle 2^s - 1 \rangle_s$, where $\langle i \rangle_s$ is the s-bit binary representation of i for some fixed integer s. The integer s is an application parameter which remains fixed for all invocations of a scheme. In general counters are employed to prevent collisions among the inputs to the underlying cryptographic primitive. This in turn helps in proving the security of the overall scheme. Counter-based schemes can be broadly classified into two categories, **(1) Counter-as-Input (or CaI) Schemes**: where the counter value is used as a standalone input to the underlying primitive; **(2) Counter-as-Encoding (or CaE) Schemes**: where the counter is encoded within the input to the underlying primitive. CTR [1], GCM [2], and SIV [3], are some schemes that fall under the former category, while XOR-MACs [4], PCS [5], and LightMAC [6] fall under the later category. There are some schemes, such as HAIFA [7], which can be included in both the categories. In this paper we solely focus on CaE schemes.

1.1 Some Counter-as-Encoding Schemes

Counter-based encoding is used in HAIFA [7] (illustrated in Fig. 1(a)), to enhance the (second preimage) security of iterated hash functions. Notably, the popular second preimage attacks [8], [9] on Merkle-Damgård (or MD) hash function [10], [11] do not apply on HAIFA. In fact Bouillagauet and Fouque [12] have shown that HAIFA has full second preimage security in the random oracle model.

In 1995, Bellare et al. proposed the so-called XOR-MACs [4], to construct stateful and probabilistic MACs. At the highest level, XOR-MAC consist of three steps: (1) counter-based encoding of message into a collection of blocks; (2) application of a pseudorandom function (or PRF) f to each of the blocks and XORing them together into a hash value, h; and (3) XORing h with $f'(R)$ where f' is a PRF independent of f, and R is either a nonce (non-repeating internal state) or a random salt. The original security proof of XOR-MACs is based on the assumption that f and f' are (pseudo)random functions. Later Bernstein [13] provided an improved analysis when f and f' are (pseudo)random permutations.

In 1999, Bernstein proposed the protected counter sum (or PCS) [5], to construct stateless and deterministic MACs. Similar to XOR-MACs, PCS also computes a hash value h using the first two steps. But in the last step, it computes $f'(h)$, where f' is a PRF independent of f. Recently Luykx et al. have extended the idea of PCS, where they replaced PRFs with PRPs, to construct LightMAC [6].

All these MAC constructions have a common underlying function, that we call h (illustrated in Fig. 1(b)). In fact XOR-MACs are based on the Hash-then-Mask (or HtM) paradigm [14] (illustrated in Fig. 1(c)) by Wegman and Carter. More precisely, if h is an AXU universal hash [15] (defined later in subsection 2.3) and f' is a random function (or permutation) independent of H, the hash-then-mask paradigm is defined as $h(M) \oplus f'(R)$. The security of these schemes mainly rely on the non-repeating nature of R and the universal property of h. PCS and LightMAC are based on the Hash-then-PRF (HtPRF) paradigm [14], [16] (illustrated in Fig. 1(d)), which is simply defined as $f'(h(M))$. The security of schemes based on this paradigm rely on the universal property of h.

1.2 Motivation

Consider the three MAC schemes discussed in the preceding subsection. All these schemes apply a counter-based input

• *A. Dutta, A. Jha and M. Nandi are with the Indian Statistical Institute, Kolkata, India.*
E-mail: ashwin.jha1991@gmail.com

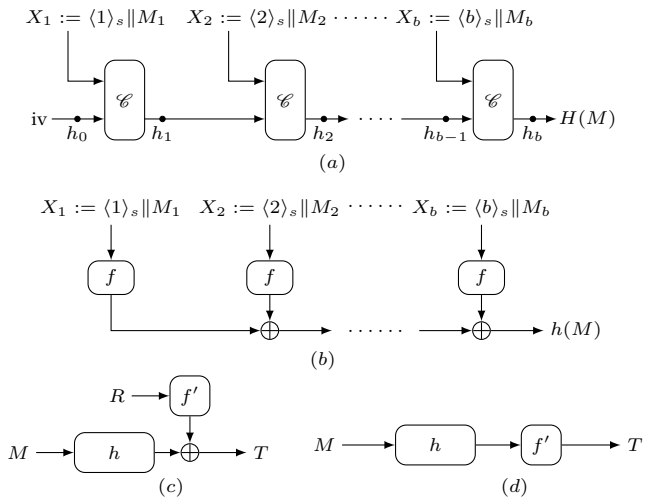

Fig. 1: Some CaE schemes. (a) HAIFA Hash Function H based on a compression function $\mathscr{C}$ (primitive for this construction); (b) Counter based AXU hash h based on a random function f (primitive for this construction); (c) Hash-then-Mask (HtM) MAC; and (d) Hash-then-PRF (HtPRF) MAC based on counter-based AXU hash h as shown in (b); Note that the input message $M := (M_1, \ldots, M_b)$ is encoded into b blocks, where each block $X_i := \langle i\rangle_s \| M_i$ is a concatenation of the s-bits binary representation of i and the message block M_i of size $n - s$.

encoding to the message. Given a message $M \in \{0,1\}^\ell$, $\ell \leq L$ (for fixed integers s and L), the encoding works as follows: Let $M' = M\|1\|0^d$ where d is the smallest integer such that $n - s$ divides the bit-length (number of bits) of M'. Suppose $M' = (M_1, M_2, \ldots, M_b)$, such that for all $i \in \{1, \ldots, b\}$, M_i is of bit-length $n - s$. Then $X := (X_1 := \langle 1\rangle_s \| M_1, \ldots, X_b := \langle b\rangle_s \| M_b)$ is the encoded input of M.

Recall that, the encoding uses a classical counter with a fixed parameter s. The choice of s may depend on the nature of the application. More precisely, s should be chosen in such a way that the number of encoded blocks generated for the longest permissible message should not be more than 2^s, to avoid the possible reseting of counter values.

A typical choice for the maximum permissible message length would allow 2^{64} encoded blocks, i.e., s is exactly 64 bits. Suppose AES cipher (with block size 128 bits) is used to instantiate the LightMAC [6] scheme. Then each call of AES will process 64 bits of message, as each encoded block contains 64 bits of counter value and 64 bits of message. So it would take 2 AES calls per 128 bits of message, independent of whether the message size is 1 KB (2^{13} bits), or 1 GB (2^{33} bits). In short for a 1 KB message, LightMAC makes about 128 AES calls. On the other hand when $s = 8$, the number of AES calls reduces to 69 (almost twice faster). But this limits the maximum number of encoded blocks to 2^8. So a classical counter cannot be efficient over a wide range of message lengths.

When the message size is known beforehand, one can simply choose the best choice of s for the given message length, instead of fixing it throughout all the invocations. This will definitely speed up the performance for shorter messages and provide very similar performance when the message size approaches the maximum size. Although this idea is very natural, it has not been applied in any algorithms so far.

One possible reason is that the security guarantee is not obvious. One has to analyze the counter-as-encoding schemes with variable length messages. In this case a general view of counters could have made this analysis much easier.

In [4] the authors have given a general framework for constructing XOR-MACs. But even this general framework is somewhat lacking and does not give a unified yet simple treatment for all possible counter-based encodings. In the main theorem of [5], it is assumed that given a message, the encoding produces distinct encoded blocks for distinct primitive calls. Although this points towards a general requirement from any counter-based encoding, the indication is rather implicit. Thus, a formal treatment of counters is required. Furthermore, this generalization should allow a generic proof technique that applies to all counter-based encodings.

A disadvantage of the aforementioned message length dependent counter scheme is the requirement of prior knowledge of the message length. In many scenarios this may not be possible. What can be a good approach when the message length is not known beforehand?

Let us consider an analogous problem from athletics. Consider a race over an unknown distance, where the athletes would only know the finishing point once they actually reach there. What should be a good strategy for running this race? One may consider to run like a marathoner hoping that it is a marathon. In this case, clearly the athlete loses if the race is a hundred meter sprint. The better solution would be to start like a sprinter and then gradually reduce speed as the race progresses. This would definitely be optimal for short runs.

Our problem is similar to the one just described. We want to find a variable length counter that offers near optimal counter size. Although similar problems are well-studied in the field of prefix codes [17], [18] and data compression [19], to the best of our knowledge it has not seen any interest in cryptography. It would be interesting to investigate the techniques used in prefix coding and construct a near optimal counter scheme when the message length is not known.

1.3 Our Contributions

In this paper we define a general notion of counters and closely study the security of CaE schemes based on this general view. More precisely, we view a counter as a function from non-negative integers to bits-strings (may not be of same sizes). An appropriate part of a message and the counters are fed into the primitive. We describe a **sufficient condition** for counters to ensure security guarantee (see section 3). We further show that these properties give straightforward generic security proofs (see section 5) for counter-based hash function (HAIFA), counter-based universal hash functions and MACs (see section 4).

Our generalization of counters, can serve as a general guideline for designers as well as developers on how to choose or create a secure counter-based encoding scheme for various applications. In other words, this work can be used as a theoretical starting point in creating/choosing a counter-based encoding scheme for practical purposes.

For instance, based on our generalizations we are able to identify two efficient alternatives (described in section 3)

for the standard counter scheme. We have implemented all these counters (see section 6) for (1) Davis-Meyer based HAIFA, (2) LightMAC, and (3) XOR-MACs. In this work we solely focus on software implementation based comparison. Our implementations use Intel's AES-NI and SSE4 instruction set [20]. We have observed that the practical results support the theoretical speed up. For instance, for 1 MB message size, MAC implementations based on new counter schemes use around 35% less clock cycles per byte as compared to the standard MAC implementations with 64-bit counter. Similarly HAIFA based on new counter schemes uses approximately 32% less clock cycles per byte as compared to the standard HAIFA implementation with 64-bit counter. These performance characteristics indicate that the two new counter schemes are indeed much more flexible and efficient as compared to the classical counter.

As noted earlier, we do not consider CaI schemes [1], [2], [3], in this paper. In these schemes the counter size does not really affects the number of primitive calls. So the counter schemes considered in this paper may not give any improvements.

2 Preliminaries

2.1 Notation

We write the set of non-negative numbers as $\mathbb{N}$. For a binary string $x = x_1x_2\cdots x_s \in \{0,1\}^s$, we call $|x| := s$ the length or bit-size of x. For $1 \le r \le s$, let $\mathsf{lsb}_r(x) := x_{s-r+1}x_{s-r+2}\cdots x_s$ (or $\mathsf{msb}_r(x) := x_1x_2\cdots x_r$) denote the least significant (respectively most significant) r bits of x. For an integer $i < 2^s$, $\langle i\rangle_s$ represents the canonical s-bit binary representation of i. For two binary strings x and y, $x||y$ represents the concatenation of two strings x and y. For any positive integer m, we write $\{0,1\}^{\le m} = \cup_{i=1}^{m}\{0,1\}^i$. Given a binary string $x \in \{0,1\}^s$ we can also view it as a nonnegative integer less than 2^s. Suppose $x \ne 1^s$ then we define $(x+1)$ to denote the s-bit binary string after adding one to x (viewed as an integer).

Convention. Throughout this paper, we fix two positive integers n and L.

1) The integer n represents the block size (bit-size) of the underlying primitive (e.g., a blockcipher), and
2) L represents the length (bit-size) of the largest message which can appear for a specific application.

All elements of $\{0,1\}^n$ are called blocks. $\{0,1\}^*, (\{0,1\}^n)^*$ denote the set of all finite length binary strings, and block strings respectively. For any $x \in \{0,1\}^*$, if $\ell = |x|$, then $\hat{\ell} = \lceil \ell/n \rceil$ denotes the number of blocks in x. B, KB and MB represent bytes, kilo bytes and mega bytes, respectively.

2.2 HAIFA Framework in Hash Function

Hash function H is a function from $\{0,1\}^*$ to $\{0,1\}^c$. Hash functions are usually constructed by means of iterating a cryptographic compression function $f : \{0,1\}^c \times \{0,1\}^n \to \{0,1\}^c$, while trying to maintain the following three requirements:

1) **Preimage resistance**: Given a challenge y, it is hard to find x such that $H(x) = y$. Here x is called *preimage* of y.
2) **Collision resistance**: It is hard to find a pair (x, x') of distinct elements, called *collision pair* for H, such that $H(x) = H(x')$.
3) **Second preimage (2PI) resistance**: Given a challenge x, it is hard to find a collision pair of the form (x, x') (in this case x' is also called a *second preimage* of x).

The most widely used mode of iteration is Merkle-Damgård hash [10], [11], which is also known as the MD hash function. The simple iteration method maintains the collision resistance and preimage resistance of the compression function. But there are multiple second preimage attacks [8], [9] on MD hash function. In [7], Biham and Dunkelman suggested the HAsh Iterative FrAmework (HAIFA) to replace the MD hash function, which would resist the earlier attack strategies. The main idea behind HAIFA is the use of counter-based (number of bits hashed so far) encoded input in the iterated hash mode. More formally, let $f : \{0,1\}^c \times \{0,1\}^n \to \{0,1\}^c$ be a compression function, $w \le n$ be a fixed integer such that $2^w \le L$, and $h_0 := \mathsf{IV}$ be a fixed initial value. Then, given a message $M \in \{0,1\}^\ell$, we first parse $M\|10^d\|\langle \ell\rangle_w$ as $(M_1, \ldots, M_b) \in (\{0,1\}^{n-w})^b$ where d is the smallest nonnegative integer such that $\ell + w + 1 + d$ is divisible by $(n-w)$. The hash output is defined as h_b where h_i's are defined recursively as $h_i = f(h_{i-1}, \langle i\rangle_w\|M_i)$, $1 \le i \le b-1$, and $h_b = f(h_{b-1}, \langle 0\rangle_w\|M_b)$ (see Fig. 2).[1] For a fixed invocation of the hash function, this method differentiates the two inputs of the underlying compression function. This fact is used by Bouillaguet and Fouque [12] to show that HAIFA has full (i.e. 2^n) second preimage security in the random oracle model.

2.2.1 Davis-Meyer Compression Function

Both MD hash function and HAIFA require an underlying compression function. An old but popular method of constructing a compression function out of a blockcipher is due to Davis and Meyer [21], [22]. The Davis-Meyer function has been proved [23] to be a secure compression function in the ideal cipher model (discussed below). Let $e : \{0,1\}^n \times \{0,1\}^c \to \{0,1\}^c$ be a blockcipher, i.e., for any $K \in \{0,1\}^n$, $e_K := e(K,\cdot)$ is a permutation over $\{0,1\}^c$. Davis-Meyer compression function $\mathsf{DM}_e : \{0,1\}^c \times \{0,1\}^n \to \{0,1\}^c$ is defined as $\mathsf{DM}_e(x,K) = e_K(x) \oplus x$.

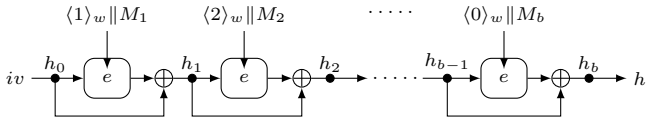

Fig. 2: HAIFA based on Davis-Meyer compression function.

The ideal cipher model by Coron et al. [24], [25] is widely used as the de facto model for a blockcipher based hash function. In this model, the adversary has access to e and it's inverse e^{-1} (defined as $e^{-1}(K,y) = e_K^{-1}(y)$). For any *fresh* query (K,x) to e (i.e. (K,x) has not been queried to e before, and there is no e^{-1}-query (K,y) whose response is x), the adversary obtains a response which is chosen uniformly

1. For administrative reason, we present a simpler variant of counter in HAIFA. We refer [7] for the actual definition.

from $\{0,1\}^c \setminus (S_1 \cup S_2)$ where S_1 and S_2 denote the set of all outputs of e_K-query (i.e. key is K), and the set of all inputs of e_K^{-1}-query, respectively. Similarly a fresh query to e^{-1} can be defined. Thus, a second preimage adversary $\mathcal{A}$, against the HAIFA hash H based on the Davis-Meyer function can make some e and e^{-1} queries, and finally it has to return a second preimage for a previously given challenge message. We write,

$$\mathbf{Adv}^{\mathrm{2PI}}(\mathcal{A}) = \mathsf{Pr}[M' \leftarrow \mathcal{A}^{e,e^{-1}}(M) : H(M') = H(M)]$$

to denote the second preimage advantage of $\mathcal{A}$ against H.

2.3 (Almost-XOR) Universal Hash Function

An n-bit hash function h is a $(\mathcal{K},\mathcal{D})$-family of functions $\{h_k := h(k,\cdot) : \mathcal{D} \to \{0,1\}^n\}_{k\in\mathcal{K}}$ defined over its domain or message space $\mathcal{D}$ and indexed by the **key space** $\mathcal{K}$.

Definition 1 (ϵ-AXU hash function [26]). A $(\mathcal{K},\mathcal{D})$-family h is called **ϵ-Almost-XOR Universal** (or AXU) hash function, if for any two distinct x and x' in $\mathcal{D}$ and a $\delta \in \{0,1\}^n$, the *δ-differential probability*

$$\mathsf{diff}_{h,\delta}[x,x'] := \mathsf{Pr}_K[h_K(x) \oplus h_K(x') = \delta] \le \epsilon$$

where the random variable K is uniformly distributed over the set $\mathcal{K}$.

The *maximum δ-differential probability*, over all possible pairs of distinct inputs x, x', is denoted by $\Delta_{h,\delta}$. The *maximum differential probability* $\Delta_h := \max_\delta \Delta_{h,\delta}$. If $\Delta_h \le \epsilon$ then we say that h is an ϵ-AXU hash. Multi-linear hash [27], [28], and pseudo-dot-product or PDP hash [27], [29], [30], [31] are some examples of AXU hash functions. In this paper we will see an example of a CaE AXU hash function (similar to PHASH [32]).

Universal Hash Function. When $\delta = 0$, the 0-differential event is equivalent to collision. So we write $\mathsf{diff}_{h,0}[x,x']$ and $\Delta_{h,0}$ by $\mathsf{coll}_h[x,x']$ and coll_h, respectively, and we call them collision probabilities.

Definition 2 (ϵ-universal hash function). A hash family h is called **ϵ-universal** (or ϵ-U) if

$$\mathsf{coll}_h := \max_{x\neq x'} \mathsf{Pr}_K[h_K(x) = h_K(x')] \le \epsilon.$$

2.4 Pseudorandom Function and Permutation

An n-bit random function $\$_{\mathcal{D}}$ is a function chosen uniformly from the set of all functions from $\mathcal{D}$ to $\{0,1\}^n$. When, $\mathcal{D} = \{0,1\}^n$, we simply write $\$_n$. Similarly, an n-bit random permutation Π_n is a permutation chosen uniformly from the set of all permutations over $\{0,1\}^n$. Given an oracle adversary $\mathcal{A}$, we define the PRF-advantage of $\mathcal{A}$ against a keyed function F_K as

$$\mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{A}) = |\mathsf{Pr}[\mathcal{A}^{F_K} = 1] - \mathsf{Pr}[\mathcal{A}^{\$_n} = 1]|.$$

Let $\mathbf{Adv}^{\mathrm{prf}}_F(t,q,\hat{\ell})$ denote $\max_{\mathcal{A}} \mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{A})$ where the maximum is taken over all $\mathcal{A}$ running in time at most t, making at most q queries such that the longest query has at most $\hat{\ell}$ many blocks. We similarly define the PRP-advantage of $\mathcal{A}$ as

$$\mathbf{Adv}^{\mathrm{prp}}_F(\mathcal{A}) = |\mathsf{Pr}[\mathcal{A}^{F_K} = 1] - \mathsf{Pr}[\mathcal{A}^{\Pi_n} = 1]|.$$

The maximum PRP advantage is denoted as $\mathbf{Adv}^{\mathrm{prp}}_F(t,q)$ (note that for a keyed permutation the domain is $\{0,1\}^n$, so all queries have exactly one block). The prf-prp switching lemma [33], [34] says that for a keyed function (or permutation) F_K over the domain $\{0,1\}^n$,

$$|\mathbf{Adv}^{\mathrm{prp}}_F(t,q) - \mathbf{Adv}^{\mathrm{prf}}_F(t,q)| \le \frac{q^2}{2^{n+1}}. \quad (1)$$

A Composition Theorem: PRF(U) $\equiv$ PRF. It is well-known [14], [16] that composition of an ϵ universal hash function h and a PRF F is a PRF. More precisely, we have the following result for HtPRF due to Wegman and Carter [14], and Shoup [16],

Theorem 1 ([14], [16]). Let $G_{K_1,K_2} := F_{K_2} \circ h_{K_1} : \mathcal{D} \to \{0,1\}^n$ where h is an ϵ-universal hash over $\mathcal{D}$. Then,

$$\mathbf{Adv}^{\mathrm{prf}}_G(t,q) \le \mathbf{Adv}^{\mathrm{prf}}_F(t',q,\widetilde{\ell}) + \binom{q}{2} \times \epsilon,$$

where $t' = t + \mathcal{O}(qT)$ and T denotes the maximum time for computing h.

2.5 Message Authentication Code

Message authentication codes or MACs are symmetric-key primitives which are used to ensure data integrity. The working principle of MAC is simple. Whenever Alice wants to send a message M to Bob, she sends (M,T) where the *tag* $T = F_K(M)$ is the output of tag-generation algorithm. In case of deterministic MAC, when Bob receives a message tag pair (M',T'), he verifies the equality $T' = F_K(M')$ (if verified, the pair is called a *valid pair*). We denote this verification algorithm by $V_K(M,T)$ which returns 1 on valid pair and 0 for invalid. Usually, an adversary may attempt (multiple times) to forge a tag for a message after having seen other tagged messages. We define the forgery advantage of a forger $\mathcal{A}$ against F_K as

$$\mathbf{Adv}^{\mathrm{forge}}_F(\mathcal{A}) = \mathsf{Pr}[\mathcal{A}^{F_K,V_K} \text{ wins}] \quad (2)$$

where $\mathcal{A}$ wins if it submits a valid and fresh (not obtained through previous F_K queries) (M,T) pair to the verification oracle. The maximum advantage in forging F is defined as

$$\mathbf{Adv}^{\mathrm{forge}}_F(t,q_m,q_v) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathrm{forge}}_F(\mathcal{A})$$

where the maximum is taken over all $\mathcal{A}$ which makes at most q_m tag-generation queries and q_v verification queries, and runs in time at most t. The tag generation algorithm is called (t,q_m,q_v,ϵ)-mac if $\mathbf{Adv}^{\mathrm{forge}}_F(t,q_m,q_v) \le \epsilon$. Note that the tag-generation algorithm F_K can be probabilistic in nature. When it is deterministic, the unforgeability is implied by the PRF property of F_K.

Lemma 1. [35] If F_K is a deterministic keyed function then $\mathbf{Adv}^{\mathrm{forge}}_F(t,q_m,q_v) \le \mathbf{Adv}^{\mathrm{prf}}_F(t,q_m+q_v) + q_v 2^{-n}$.

2.5.1 Categories of MAC

CBC-MAC [36], PMAC [37], EMAC [38], HMAC [39], PCS [5], LightMAC [6] etc. are some examples of deterministic MACs. A probabilistic tag-generation algorithm takes an additional random input R along with the message M. The tag of the probabilistic MAC, denoted as $F^{\$}_K(M) := (R, F_K(R,M))$, is the pair (R,T). The *verification algorithm* $V_K(M,(R,T))$ checks whether $F_K(R,M) = T$ or not.

XMACR [4], RMAC [40], MACRX$_3$ [41], EHtM [42], RW-MAC [42] and FRMAC [43] are some examples of probabilistic MACs. Stateful MACs can be viewed as variants of probabilistic MACs. In this case, R is treated as a nonce (unique for each invocation of the tag-generation algorithm) and stored in an internal memory. We denote the output of a stateful MAC as $F^{st}(M)$. WMAC [44], XMACC [4], and EWCDM [45] are some examples of stateful MACs. Many probabilistic/stateful MAC schemes are based on the Hash-then-Mask paradigm by Wegman and Carter [14]. For these schemes the MAC security can be bounded by the universal property of the underlying AXU hash function.

Theorem 2 ([14]). Let $h_K : \mathcal{D} \to \{0,1\}^n$ be an ϵ-AXU hash and $F_{K'}$ be a keyed function over $\{0,1\}^n$. Let

$$G_{K,K'}(R, M) = F_{K'}(R) \oplus h_K(M).$$

Then, we have

1) $\mathbf{Adv}^{\text{forge}}_{G^{\$}}(t, q_m, q_v) \leq \mathbf{Adv}^{\text{prf}}_{F}(t', q_m + q_v) + q_v\epsilon + \frac{q^2}{2^{n+1}}$.
2) $\mathbf{Adv}^{\text{forge}}_{G^{st}}(t, q_m, q_v) \leq \mathbf{Adv}^{\text{prf}}_{F}(t', q_m + q_v) + q_v\epsilon$.

3 A New Look at Counters

In computer science, counters are used to count the number of times a particular event or process has occurred. In addition with counting, counters have a very special role in cryptography. In CaE schemes, the counter values are used to encode the message into distinct blocks. This freshness of inputs actually helps in proving (possibly improved) the security of CaE schemes. For example, the input block of the i-th execution of the underlying compression function of a HAIFA [7] hash function contains an encoding of i. Similarly, in counter-based MACs, e.g. XOR-MACs [4], PCS [5], LightMAC [6], the i-th input of the underlying pseudorandom function or permutation contains an encoding of i. The classical encoding of i is $\langle i \rangle_s$ which is the s-bit binary representation of i for a fixed parameter s. Whenever $i < 2^s$, we have distinct binary string corresponding to each counter value. For counter-based algorithms, the parameter s can be chosen as per the needs of the application domain. For example, if we know beforehand that for an application, the message size can not be more than 2^{32}, then one can choose $s = 32$. However, it must remain constant throughout all the executions of the algorithm. This might affect the performance for smaller length messages. In this section we introduce a new and general way of looking at counters which will provide some tools to improve the performances of CaE schemes without compromising with the security.

Definition 3 (Counter function family). A counter function family (we also use CFF) CTR is a family of counter functions $\{\mathsf{ctr}_\ell : \ell \leq L\}$ where for all $\ell \leq L$, $\mathsf{ctr}_\ell : \mathbb{N} \to \{0,1\}^{<n}$. We say that CTR is prefix-free if for all $\ell \leq L$, ctr_ℓ is prefix-free (i.e., for all $i \neq j$, $\mathsf{ctr}_\ell(i)$ is not a prefix of $\mathsf{ctr}_\ell(j)$).

The classical (or standard) encoding, as mentioned above, can be viewed as a CFF STD^s where $\mathsf{std}^s_\ell(i) = \langle i \rangle_s$, for a fixed positive integer $s < n$. We use capital letters to denote a function family and small letters for individual functions. Note that the standard counter function std_ℓ is actually independent of ℓ and hence for all ℓ, the counter functions are same. We call such CFFs *message length independent*. For a message length independent CFF CTR, we simply write ctr to denote the counter function ctr_ℓ. Note that the standard counter is a prefix-free counter. Prefix-free CFF is necessary to avoid repetitions among the inputs to the primitive (see Lemma 2 below). We also note that the size of the output of std_ℓ is fixed for all counter values. In general for a CFF CTR, if $\forall \ell\ \forall i, j\ \ |\mathsf{ctr}_\ell(i)| = |\mathsf{ctr}_\ell(j)|$, then we say that the CFF has *fixed size*. Otherwise, we call it a *variable size CFF*. We will see some examples of message length dependent and variable size CFFs later in the section.

3.1 Counter Function Family Based Message Encoding

Recall the encoding scheme discussed in section 1.2. When we use the standard counter STD^s with $s2^{n-s} \leq L$, we first parse a message $M \in \{0,1\}^\ell$ as $(M_1, \ldots, M_{b-1}, M'_b)$ where $b = \lceil \frac{\ell+1}{n-s} \rceil$, $|M_1| = \cdots |M_{b-1}| = n - s$ and $|M'_b| < (n - s)$ such that $M = M_1 \| \cdots \| M_{b-1} \| M'_b$. Let $M_b = M'_b \| 10^d$ where $d = n - s - 1 - |M'_b|$. Thus, $|M_b| = n - s$. We also denote the parsing of M as

$$(M_1, \ldots, M_b) \xleftarrow{n-s} M \text{ or } (M_1, \ldots, M_b) \xleftarrow{\mathsf{STD}^s} M$$

to emphasize the standard CFF. Now we define the b blocks encoding as $X_i = \mathsf{std}^s(i) \| M_i$ for all $1 \leq i \leq b$. These blocks are used as inputs to the underlying primitive, such as blockcipher or compression function.

We extend the same methodology to define the encoding for any other CFF CTR. For this, we first define the block function $b^{\mathsf{CTR}}(\ell)$, which associates each message size to a unique number of blocks. We have seen that for standard counter, $b^{\mathsf{STD}}(\ell) = \lceil (\ell + 1)/(n - s) \rceil$.

Definition 4. For any CFF CTR, we define the **block function**, $\boldsymbol{b^{\mathsf{CTR}}(\ell)}$ as the least integer b such that

$$\ell + 1 \ \leq \ \sum_{i=1}^{b} (n - |\mathsf{ctr}_\ell(i)|) \ \leq \ \ell + n.$$

It is easy to see that such a b exists and it is unique. Now given a message $M \in \{0,1\}^\ell$, $\ell \leq L$, we parse it as $M = M_1 \| \cdots \| M_{b-1} \| M'_b$ where $|M_i| = n - |\mathsf{ctr}_\ell(i)|$ for all $i < b$ and $0 \leq |M'_b| < n - |\mathsf{ctr}_\ell(b)|$. We similarly pad the last block to make it compatible with the corresponding counter. More precisely, we define $M_b = M'_b \| 10^d$ where $d = n - |\mathsf{ctr}_\ell(b)| - 1 - |M'_b|$. Thus, $|M_b| = n - |\mathsf{ctr}_\ell(b)|$. Finally, we define the encoding

$$\mathsf{CTR}(M) := (X_1, \ldots, X_b)$$

where $X_i = \mathsf{ctr}_\ell(i) \| M_i$ for all $1 \leq i \leq b$. Thus, we also view the CFF as an encoding function $\mathsf{CTR} : \{0,1\}^{\leq L} \to (\{0,1\}^n)^+$.

Recall that one of the main purpose of counters in cryptography is to generate distinct blocks. Mathematically, a CFF CTR is called *block-wise collision-free* if for all $M \in \{0,1\}^\ell$, $\ell \leq L$, X_i's are distinct where $\mathsf{CTR}(M) = (X_1, \ldots, X_b)$. Now we provide a characterization of block-wise collision-free counters.

Lemma 2. CTR is block-wise collision-free if and only if it is prefix-free.

Proof: We first prove the "only if" direction. Suppose there exists $i \neq j$ such that $i, j \leq b$ and $\mathsf{ctr}_\ell(i)$ is a prefix of $\mathsf{ctr}_\ell(j)$. Thus, we can find x and y such that $\mathsf{ctr}_\ell(i)\|x = \mathsf{ctr}_\ell(j)\|y \in \{0,1\}^n$. Let $(M_1, \ldots, M_b) \overset{\mathsf{CTR}}{\leftarrow} M \in \{0,1\}^\ell$. Then we define $M' \in \{0,1\}^\ell$ by replacing M_i and M_j by x and y respectively. Thus, it is easy to see that $X'_i = X'_j$ where $\mathsf{CTR}(M') = (X'_1, \ldots, X'_b)$.

To prove the other direction, let us assume $X_i = X_j$ for some $i \neq j$. Therefore, $\mathsf{ctr}_\ell(i)||M_i = \mathsf{ctr}_\ell(j)||M_j$. This clearly proves that either $\mathsf{ctr}_\ell(i)$ is a prefix of $\mathsf{ctr}_\ell(j)$, or $\mathsf{ctr}_\ell(j)$ is a prefix of $\mathsf{ctr}_\ell(i)$. ■

3.2 Some (Efficient) Alternatives to the STD CFF

We have already demonstrated the classical or standard CFF STD^s. It is a message-length independent and fixed size CFF. In this section we study two new examples of CFFs and see their advantages over the standard one.

3.2.1 STD^{opt}: A Message Length Dependent CFF

Counter function of our first CFF is an s bit classical encoding of its argument, where s depends on the message size instead of a pre-determined fixed parameter. In fact, s is defined as the smallest possible value such that we can represent all the counter values distinctly for the given message length. Formally, the counter function is defined as

$$\mathsf{std}_\ell^{opt}(i) = \langle i \rangle_s, \text{ where } s = \min\{g : 2^g(n-g) > \ell\}.$$

It is an example of message-length dependent and fixed size CFF. Clearly, it is a prefix-free counter.

3.2.2 VAR^r: A Variable Size CFF

Till now we have seen counter functions which map integers to their s bits binary representation. In case of STD^s, s is fixed (approx. $\lg L$), whereas in case of STD^{opt}, s depends on the message length ℓ (approx. $\lg \ell$). Both STD^s and STD^{opt} are fixed size CFFs. Now we will construct a CFF which maps integers to binary strings of monotonically increasing lengths. A similar problem is well known in the field of source coding. We briefly discuss these techniques assuming that the set of symbols is $\mathbb{N}$. Readers may refer to the references cited for a more detailed exposition.

Prefix codes. A mapping $\alpha : \mathbb{N} \to \{0,1\}^*$, is called a binary prefix code [18] over integers, if for all $x \neq y \in \mathbb{N}$, $\alpha(x)$ is not a prefix of $\alpha(y)$ and vice-versa. Here $\alpha(x)$ is called the codeword corresponding to x. Huffman codes [17], [18], Shannon-Fano codes [18], [19], [46] and Universal codes [18], [47] are some popular techniques for getting prefix codes.

Huffman codes and Shanon-Fano codes are in general better than universal codes, when the probability (or frequency) distribution over the integers is known. This is generally the case in static data compression settings, such as JPEG File Interchange Format [48] which employs a modified form of Huffman coding, and the ZIP file format [49] which uses a modified version of Shannon-Fano coding. Although Huffman codes and Shannon-Fano codes are better than universal codes, but they require exact probability (or frequency) distribution over the integers.

Universal Codes. A universal code [18], [47], [50] is a binary prefix code, with the additional property that if the probability (or frequency) distribution over integers is monotonic, then the expected lengths of the codewords are within a constant factor of the optimal expected lengths for that distribution. Note that a universal code works for any countably infinite set, and it only requires a monotonic distribution. This is one of the main motivations behind the study of universal codes. Our problem falls precisely in this case, as we can always assume a natural monotonic distribution over the number of blocks. For instance all messages must have at least one block, so 1 has the maximum frequency, followed by 2, and so on.

In 1974, Elias proposed the first prefix code with universal property [47]. In 1975, he followed it up by proposing several new examples of universal codes [50]. Some of the popular examples of universal codes are Elias gamma coding [18], [50], Elias delta coding [18], [50], Elias omega coding [18], [50], Fibonacci coding [18], [51], Levenshtein coding [18] and Exp-Golomb coding [18]. Now we describe two examples of universal codes, viz., Elias gamma and delta codings.

Elias Gamma Coding, γ. To code a number $i \geq 1$:

1) Let $j = \lfloor \log_2 i \rfloor$, i.e., $2^j \leq i < 2^{j+1}$.
2) $\gamma(i) := 0^j \| \langle i \rangle_{j+1}$.

Elias Delta Coding, δ. To code a number $i \geq 1$:

1) Let $j = \lfloor \log_2 i \rfloor$, i.e., $2^j \leq i < 2^{j+1}$.
2) $\delta(i) := \gamma(j+1) \| \mathsf{lsb}_j(\langle i \rangle_{j+1})$.

To represent an integer i, Elias gamma uses $2\log_2 i + 1$ bits, and Elias delta uses $\log_2 i + 2\log_2(\log_2 i + 1) + 1$ bits. Clearly, Elias delta is more compact as compared to Elias gamma.

CFF from Universal Coding. Theoretically, any universal code U can be used to construct a length-independent variable size prefix-free CFF $\mathsf{U\text{-}CTR}$. Suppose U is a universal code, then we define $\mathsf{U\text{-}CTR}$ as follows:

$$\mathsf{U\text{-}CTR} = \{\mathsf{u\text{-}ctr}_\ell : \ell \leq L\} \text{ where } \mathsf{u\text{-}ctr}_\ell = \mathsf{U},\ 1 \leq \ell \leq L.$$

Clearly, $\mathsf{U\text{-}CTR}$ is prefix-free as U is universal. Although one might be tempted to use $\mathsf{U\text{-}CTR}$ straightaway, there is still some scope of improvement in this generic scheme. Note that the motivation for universal coding is quite different than the cryptographic settings.

First, we are restricting the domain to some L, whereas a universal code is defined over $\mathbb{N}$. Intuitively it seems that the universal codes must have some redundant information that we can avoid. Second, all the CFFs discussed so far have efficient counter update (generating $\mathsf{u\text{-}ctr}_\ell(i+1)$ from $\mathsf{u\text{-}ctr}_\ell(i)$) mechanism, which is not a mandatory requirement for universal codes. So we should restrict our focus to only those universal codes which support efficient update mechanism, such as Elias delta code. Now we present our second CFF candidate, called VAR^r, which is a modified version of δ-CTR.

VAR^r Counter Function Family. We first fix r, an application parameter chosen suitably (we will see very soon how to choose r). As VAR^r would be message length independent, it would be sufficient to define a function over the domain $\{1, 2, \ldots, L\}$. To begin with, we define $\mathsf{var}^r(0) = 0^r$. Now, for all $i \leq L$, we will recursively define $\mathsf{var}^r(i)$ given that $\mathsf{var}^r(j)$ has been defined for all $j < i$. Like STD^s

and STD^{opt}, we increment the counter by one at every step. In other words, we first define $y = \mathsf{var}^r(i-1)+1$. If the r most significant bits get altered (this is easy to check, as the remaining bits would all be zero), then we define $\mathsf{var}^r(i) = y\|0$, otherwise we define $\mathsf{var}^r(i) = y$. Mathematically, we can write the recursive definition of var^r for $i \geq 1$, as

$$\mathsf{var}^r(i) = \begin{cases} x+1 & \text{if } msb_r(x) = msb_r(x+1), \\ (x+1)\|0 & \text{if } msb_r(x) \neq msb_r(x+1). \end{cases}$$

where $x = \mathsf{var}^r(i-1)$.

Clearly the size of the counter function output increases slowly but monotonically with i. So VAR^r is an example of message length independent and variable size CFF. To understand this counter, we demonstrate how the counter values are computed for $r = 3$. A boldface zero at the least significant bit represents the added zero bit on expansion. The underlined bits are the third most significant bits.

$$00\underline{0}, 00\underline{1}\mathbf{0}, 00\underline{1}1, 01\underline{0}0\mathbf{0}, 01\underline{0}01, 01\underline{0}10, 01\underline{0}11, 01\underline{1}00\mathbf{0}, \ldots$$

Now we will describe how one should choose the parameter r given that the limit on the message length is L. Note that, when the size of the counter reaches $r+i$ for the first time, the i least significant bits of the counter value are all zero. So the counter will be incremented 2^i many times before the next expansion in counter size. Since we need to keep the r most significant bits non-zero (to avoid repetitions), the following equation must be satisfied:

$$\sum_{i=0}^{2^r-1} (n-r-i)\cdot 2^i > L.$$

After doing some algebraic simplifications one can show that the smallest r that satisfies the above equation is approx. $\log_2 \log_2 L$ for $L < 2^{c(n)\cdot n}$ where $\frac{1}{2} \leq c(n) < 1$. Note that for any integer i,

$$\mathsf{var}^r_\ell(i) = \langle \lfloor log_2 i \rfloor \rangle_r \| \mathsf{lsb}_{\log_2 i}(\langle i \rangle_{\log_2 i+1}),$$

whereas

$$\delta(i) = \gamma(\lfloor \log_2 i + 1 \rfloor) \| \mathsf{lsb}_{\log_2 i}(\langle i \rangle_{\log_2 i+1}).$$

Clearly $|\mathsf{var}^r_\ell(i)|$ is less than $|\delta(i)|$ when

$$i \geq 2^{2^{\frac{r-1}{2}}-1}.$$

We generally fix the maximum message length to be 2^{64} bits, which gives $r \approx 6$. So, the average counter size in VAR^r will be less than the average counter size in δ-CTR, when the number of generated blocks is at least 2^6. Specifically for around 2^{16} blocks, the average counter size in VAR^r is shorter than the average counter size in δ-CTR by more than 3 bits. Now we show that VAR^r is a prefix-free CFF.

Theorem 3. Let r be defined as above then VAR^r is prefix-free.

Proof: Let $i \neq j$ be non-zero and $x = \mathsf{msb}_r(\mathsf{var}^r(i))$ and $y = \mathsf{msb}_r(\mathsf{var}^r(j))$. If $x \neq y$ then x can not be prefix of y. So assume $x = y$. Because of our choice of r, the r most significant bits does not become 0^r (except for the input 0). So, $\mathsf{var}^r(i)$ and $\mathsf{var}^r(j)$ have same size. As i and j are distinct, the rest of the bits must be different. This proves the prefix-free property. ■

3.2.3 Word Oriented Adaptation of Our Counters

The counter functions described in the preceding section are aimed to minimize the counter size as much as possible, keeping all the counter values distinct (i.e., prefix-free). However, there are different practical issues while implementing these counter functions. The most important issue is to parse the message into small chunks which are compatible with the counter-based encoding. In practice, we mostly receive messages as a sequence of words (e.g., 8-bit (byte), or 32-bit words). It would be easier for implementation if we can parse the messages in multiples of word size. More formally, let us fix a parameter w (elements of $\{0,1\}^w$ are called words). We define $\mathsf{STD}^{opt,w}$ (the word oriented adaption of STD^{opt}) as follows:

$$\mathsf{std}^{opt,w}_\ell(i) = \langle i \rangle_s, \text{ where } s = \min\{g : w|g,\ (n-g)2^g > \ell\}.$$

Note that $\mathsf{STD}^{opt} = \mathsf{STD}^{opt,1}$. Now we describe how we can generalize our second proposal to $\mathsf{VAR}^{r,w}$ which fits into word oriented implementation, for $r \leq w$. We define the counter function recursively as before except that we add 0^w instead of single 0. More formally, $\mathsf{var}^{r,w}(0) = 0^w$. For $i \geq 1$, let $x = \mathsf{var}^{r,w}(i-1)$. Then,

$$\mathsf{var}^{r,w}(i) = \begin{cases} x+1 & \text{if } msb_r(x) = msb_r(x+1), \\ (x+1)\|0^w & \text{if } msb_r(x) \neq msb_r(x+1). \end{cases}$$

We choose $r = 4$ for $w = 8$ in our implementation.[2] For this choice of r, the counter function is prefix-free. In the following example we describe how the counter expands for $r = 4$ and $w = 8$.

$$000\underline{0}0000, \ldots, 000\underline{1}0000\mathbf{0^8}, \ldots, 001\underline{0}0000000000000\mathbf{0^8}, \ldots$$

To the best of our knowledge such word-oriented definitions are not available for delta codes (for obvious reasons). Even if we use our word-oriented definition, the resulting code will not give better counter function. This can be argued by the simple fact that in case of $\mathsf{VAR}^{r,w}$, r is fixed, so we can simply start with some fixed w and then move as it is. But in case of delta code we have to consider the underlying gamma code also which is variable in nature. Handling two variable components may result in overheads in the counter size as well as the update mechanism.

3.3 Comparison of Rates of Counter Function Families

Recall that we have defined the number of blocks in a message of length ℓ as $\hat{\ell} = \lceil \ell/n \rceil$. As we execute some costly primitive function on these blocks, it would be good to minimize the number of blocks as much as possible.

Definition 5. We define the rate function $\mathsf{rate}^{\mathsf{CTR}}(\ell)$ associated with a counter-based encoding CTR, as the ratio of the number of blocks in the message to the total number of blocks in the encoded message produced by CTR, i.e., $\mathsf{rate}^{\mathsf{CTR}}(\ell) = \frac{\hat{\ell}}{b^{\mathsf{CTR}}(\ell)}$.

As $\hat{\ell} \geq \ell/n$, we have $\mathsf{rate}^{\mathsf{CTR}}(\ell) \leq \frac{\ell}{bn}$. So, one can achieve maximum rate one (in this case there is no counter). Now we provide a comparison between the rate functions for the three CFFs defined above.

2. Our implementation uses a bit different step size to leverage Intel's AES-NI and SSE instructions.

1) For the standard counter STD^s, the rate function is
$$\frac{\lceil \ell/n \rceil}{\lceil (\ell+1)/(n-s) \rceil} \approx \frac{n - \lg L}{n}.$$
When L is large, we need to choose large $s \approx \log_2 L$, which reduces the rate.
2) For STD^{opt}, the rate function is
$$\frac{\lceil \ell/n \rceil}{\lceil (\ell+1)/(n-\log_2 \ell) \rceil} \approx \frac{n - \log_2 \ell}{n}.$$
Clearly, the rate of this counter is better than STD^s for all choices of $\ell < L$.
3) For VAR^r, the rate function is a complex function in n and r. We skip the exact algebraic expression here and give an approximation,
$$\frac{\lceil \ell/n \rceil}{\lceil (\ell+n-r+2)/(n-r+2-\log_2 \ell) \rceil} \approx \frac{n-r+2-\log_2 \ell}{n} \quad \Big(\text{for } \ell \gg n\Big).$$
Clearly, the rate of this counter is better than STD for small messages and large s. Further the rate is comparable with STD^{opt} for $n \gg r$.

3.4 Word Oriented Rates of Counter Function Families

We will assume that $n, \ell, L \in \mathbb{M}_w$, where $\mathbb{M}_w$ denotes the set of all multiples of w. We assume that $L = o(2^n)$.

The word oriented rate functions for STD^s and STD^{opt} are pretty similar to their bit oriented counterparts. For instance, in STD^s, we can simply choose s to be some multiple of w. So we only derive the word oriented rate function for VAR^r.

3.4.1 Estimating Parameter r in VAR^{r,w}

Let $c = \lceil r/w \rceil$. We know that the following inequality holds for the correct value of r,
$$\sum_{i=0}^{2^r-1} (n - cw - iw) 2^{iw+cw-r} \geq L \tag{3}$$
Let $n' := n - cw$, $r' := cw - r$, $\dot{r} = 2^r$, and $\dot{w} = 2^w$. Now we will get a lower bound on r,
$$\sum_{i=0}^{\dot{r}-1} (n' - iw) 2^{iw+r'} \geq L$$
$$2^{r'} \left(n' \sum_{i=0}^{\dot{r}-1} \dot{w}^i - w \sum_{i=0}^{\dot{r}-1} i \cdot \dot{w}^i \right) \geq L$$
$$2^{r'} \left[\left(n' + \frac{w\dot{w}}{\dot{w}-1} \right) \left(\frac{\dot{w}^{\dot{r}}-1}{\dot{w}-1} \right) - \frac{w\dot{r}\dot{w}^{\dot{r}}}{\dot{w}-1} \right] \geq L \tag{4}$$
$$2^{r'} \left[(n'+2w)\left(2\dot{w}^{\dot{r}-1}\right) - w\dot{w}^{\dot{r}-1} \right] \geq L \tag{5}$$
$$\dot{w}^{\dot{r}} \left(2n - w\right) \geq 2L \tag{6}$$
where (4) can be obtained by simple algebraic simplifications. As $W \geq 2$, we have $\frac{1}{\dot{w}-1} \leq \frac{2}{\dot{w}}$. Using this we get (5). Also $c \geq 1$ and $r' \leq w - 1$, which gives us (6). Now simple algebraic simplifications give us the upper bound on r,
$$r \geq \log_2 \left\lceil \frac{\log_2 \left(\frac{2L}{2n-w} \right)}{w} \right\rceil \tag{7}$$
For $w \ll n \ll L$, we get $r \approx \log_2 \log_2 L - \log_2 w$.

3.4.2 Estimating the Block Function of VAR^{r,w}

For a given message length ℓ, we are interested in a lower bound on $b(\ell)$.[3] For simplicity we also assume $c = r/w$, i.e., $r' = 0$. We restrict our analysis to only those ℓ, for which $\exists\, k \leq 2^r - 1$ such that the number of blocks,
$$b(\ell) = 2^{r'} \sum_{i=0}^{k-1} \dot{w}^i = \left(\frac{\dot{w}^k - 1}{\dot{w} - 1} \right). \tag{8}$$
We find an upper bound for $b(\ell)$ in the following derivation,
$$\left(n' \sum_{i=0}^{k-1} \dot{w}^i - w \sum_{i=0}^{k-1} i \cdot \dot{w}^i \right) \leq \ell + n$$
$$\left(n' + \frac{w\dot{w}}{\dot{w}-1} - \frac{wk\dot{w}^k}{\dot{w}^k - 1} \right) \left(\frac{\dot{w}^k - 1}{\dot{w} - 1} \right) \leq \ell + n \tag{9}$$
$$(n - r + w - wR) \left(\frac{\dot{w}^k - 1}{\dot{w} - 1} \right) \lesssim \ell + n \tag{10}$$
$$(n - r + w - \log_2(\ell + n)) \left(\frac{\dot{w}^k - 1}{\dot{w} - 1} \right) \lesssim \ell + n \tag{11}$$
For $W \geq 2$, $\frac{1}{\dot{w}-1} \geq \frac{1}{\dot{w}}$, and for moderately large k, we get $\frac{\dot{w}^k}{\dot{w}^k - 1} \approx 1$. Using these facts we get (10) from (9). Similarly $wk \lesssim \log_2(\ell + n)$ (experimental results show that $wk \approx \log_2 \ell$), which gives (11) from (10). Using (8) and (11) we get,
$$b(\ell) \lesssim \frac{\ell + n}{n - r + w - \log_2(\ell + n)} \tag{12}$$

Rate Function for Word Oriented VAR^{r,w}. Finally we get the following approximation for lower bound on the rate function of word oriented VAR^{r,w},
$$\text{rate}^{\text{VAR}^{r,w}}(\ell) \gtrsim \frac{n + w - r - \log_2(\ell + n)}{n}$$
for $w \ll n \ll \ell$. Note that we derived this function for some restricted ℓ values. But our experimental results show that this holds for all ℓ. In Table 1 we provide the experimental results for the word oriented rate functions of the three candidate counter functions. Fig. 3 gives a graphical comparison between the rates of the three CFFs.

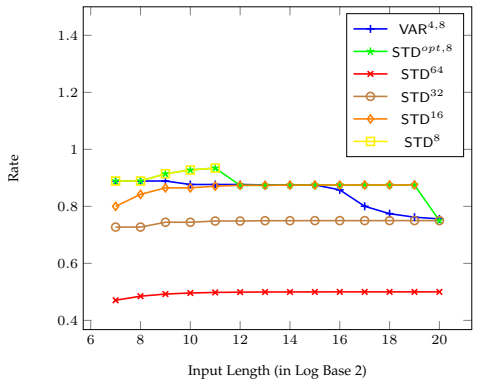

Fig. 3: Rate plot for the three CFFs.

3. Note that we have dropped VAR^{r,w} from the superscript as it is obvious.

TABLE 1: Comparison between the rates offered by the three candidate CFFs for block size, $n = 128$.

Length	STD^s				$\mathsf{STD}^{opt,8}$	$\mathsf{VAR}^{4,8}$
	s = 8 bits	s = 16 bits	s = 32 bits	s = 64 bits		
128B	0.89	0.80	0.73	0.47	0.89	0.89
256B	0.89	0.84	0.73	0.48	0.89	0.89
512B	0.91	0.86	0.74	0.49	0.91	0.89
1KB	0.93	0.86	0.74	0.50	0.93	0.88
2KB	0.93	0.87	0.75	0.50	0.93	0.88
4KB	-	0.87	0.75	0.50	0.87	0.88
8KB	-	0.87	0.75	0.50	0.87	0.88
16KB	-	0.87	0.75	0.50	0.87	0.88
32KB	-	0.87	0.75	0.50	0.87	0.88
64KB	-	0.87	0.75	0.50	0.87	0.86
128KB	-	0.87	0.75	0.50	0.87	0.80
256KB	-	0.87	0.75	0.50	0.87	0.77
512KB	-	0.87	0.75	0.50	0.87	0.76
1MB	-	-	0.75	0.50	0.75	0.76

4 Counter-as-Encoding Constructions

Now we will analyze various CaE schemes, such as AXU hash, MAC and HAIFA hash function based on our CFFs. In the last section, we defined prefix-free counter. For any such prefix-free counter CTR and for all M, we know that $X_1, \ldots, X_b$ are distinct where $\mathsf{CTR}(M) = (X_1, \ldots, X_b)$. With a slight abuse of notation, we let $\mathsf{CTR}(M)$ to denote the set $\{X_1, \ldots, X_b\}$. Observe that prefix-free property does not say anything about the collisions between the encoded blocks of two distinct messages. Clearly we will have some intersection between $\mathsf{CTR}(M)$ and $\mathsf{CTR}(M')$ (viewed as a set) for any counter function family.

Definition 6. A prefix-free CFF CTR is called **injective** if for all $M \neq M'$, $\mathsf{CTR}(M) \neq \mathsf{CTR}(M')$ (as a set).

Now we provide a sufficient condition for injective counter function families.

Lemma 3. Let CTR be a prefix-free CFF. It is injective if it satisfies the following condition

$$\forall\ \ell, \ell' \leq L,\ b(\ell) = b(\ell') \Rightarrow \mathsf{ctr}_\ell = \mathsf{ctr}_{\ell'}.$$

Proof: Suppose for some $M \in \{0,1\}^\ell, M' \in \{0,1\}^{\ell'}$, $\mathsf{CTR}(M) = \mathsf{CTR}(M')$. We require to show that $M = M'$. Note that, $\mathsf{CTR}(M) = \mathsf{CTR}(M') \Rightarrow \{X_1, \ldots, X_{b(\ell)}\} = \{X'_1, \ldots, X'_{b(\ell')}\}$. So $b(\ell) = b(\ell') = b$. By given condition, we have $\mathsf{ctr}_\ell = \mathsf{ctr}_{\ell'}$ and we denote it simply by ctr. Now we first show that $X_i = X'_i$ for all i. As two sets are equal for any $i \leq b$, there must exist j so that $X_i = X'_j$. This implies that one of $\mathsf{ctr}(i)$ and $\mathsf{ctr}(j)$ is prefix of the other, and hence $i = j$ (as the counter function is block-wise collision-free). Since counter functions for ℓ and ℓ' are same we have $M_i = M'_i$ for all i. This proves that $M = M'$. ■

Recall that all our candidate CFFs are prefix-free. Further it is obvious to see that they also satisfy the condition for injectiveness. So we get the following corollary.

Corollary 1. $\mathsf{CTR} \in \{\mathsf{STD}, \mathsf{STD}^{opt}, \mathsf{VAR}^r\}$ is a prefix-free and injective CFF.

In the following subsections we present various CaE schemes. The security of all these schemes follow directly from the prefix-free and injective property of the underlying CFF. Specifically, all the security results in the following subsections hold, when we instantiate these schemes with $\mathsf{CTR} \in \{\mathsf{STD}, \mathsf{STD}^{opt}, \mathsf{VAR}^r\}$.

4.1 Counter Based HAIFA

Let CTR be a CFF. We present a counter based HAIFA hash function, called $\mathsf{CtHAIFA}$, based on an n-bit (both blocksize and keysize is n bits) Davis-Meyer compression function DM. For a message $M \in \{0,1\}^{\leq L}$, let $h_0 = \mathsf{IV}$ (an n bit constant) and

$$h_i = \mathsf{DM}(h_{i-1}, X_i),\ \ \forall\ i \in \{1, \ldots, b\}$$

where X_i is the i-th element of $\mathsf{CTR}(M)$. We define $\mathsf{CtHAIFA}$ based on CTR as,

$$\mathsf{CtHAIFA}(M) = \mathsf{DM}(h_b, \langle |M| \rangle_n).$$

Note that this definition is a bit different than the one described earlier. Here we use $\langle |M| \rangle_n$ as the last block instead of $\langle 0 \rangle_w \| M_b$. This has been done just for the sake of simplicity. The security result can also be proved for the original definition with some additional notations. We present our main result on $\mathsf{CtHAIFA}$ in the following theorem. We postpone the proof of this theorem to the next section.

Theorem 4. Let CTR be a prefix-free and injective CFF. Then, $\mathsf{CtHAIFA}$ has full second preimage security. More specifically, for any second preimage adversary $\mathcal{A}$ that makes at most q queries, we have

$$\mathbf{Adv}^{\mathsf{2PI}}_{\mathsf{CtHAIFA}}(\mathcal{A}) \leq \frac{3q}{2^n}.$$

Remark 1. Although the fixed points in Davis-Meyer compression functions can be easily computed, this does not help in inverting any arbitrary hash value. This can be argued as follows: Let h_i be some hash value and the adversary aims to compute a pair (h_{i-1}, m) such that $\mathsf{DM}_e(h_{i-1}, m) = h_i$. Suppose the adversary queries (m, y) to the decryption oracle, then the probability that $e_m^{-1}(y) = x$ such that $e_m(x) \oplus x = h_i$ is at most $1/(2^n - i + 1)$.

Remark 2. Dean [52] showed an attack on the MD hash function when the underlying compression function has efficiently computable fixed points for random message blocks. That attack cannot be extended to $\mathsf{CtHAIFA}$ as the underlying CFF has block-wise collision free property.

Remark 3. In Crypto 2005, Coron et al. [24] proved that MD hash with prefix-free encoding is indifferentiable from a random oracle. Here prefix-free refers to the property that the encoding of M is not a prefix of the encoding of M', if $M \neq M'$. Recall that a prefix-free CFF means $\mathsf{ctr}_\ell(i)$ is not a prefix of $\mathsf{ctr}_\ell(j)$ if $i \neq j$. Not all encoding schemes based on prefix-free counters are prefix-free encodings. In fact length padding is necessary to get prefix-free encodings.

4.2 AXU Hash Function

Let CTR be a CFF. We define a counter based AXU-hash function, denoted CtH, based on an n-bit pseudorandom permutation E_K and CTR. Let $M \in \{0,1\}^\ell$. We define

$$\mathsf{CtH}_{E_K,\mathsf{CTR}}(M) = E_K(X_1) \oplus \cdots E_K(X_b)$$

where $\mathsf{CTR}(M) = (X_1, \ldots, X_b)$. Note that this hash function is a generalization of the hash functions used in [4], [5], [6]. It has a simple design that requires only E_k computation and XOR operations. Furthermore the hash can be computed completely in parallel. Now we show that if we replace E_K by a random permutation, we get an AXU hash function. The proof is postponed to the next section.

Theorem 5. Let CTR be a prefix-free and injective CFF. Then, $\mathsf{CtH}_{\Pi_n,\mathsf{CTR}}$ is $2/2^n$-AXU hash if $b^{\mathsf{CTR}}(L) \leq 2^{n-1}$, where L is the length of the largest message.

4.3 MAC

Now that we have an AXU hash function, we can apply standard methods, such as Hash-then-PRF and Hash-then-Mask, to obtain MAC schemes from CtH. Let CTR be a CFF. Given any $M \in \{0,1\}^\ell$ with $\ell > n$, we write $M = M'\|m$ where m is a block. We define

$$\mathsf{CtMac1}_{\Pi_n,\Pi'_n}(M) = \Pi'_n(\mathsf{CtH}_{\Pi_n}(M') \oplus m)$$

CtH is an AXU hash function when CTR is prefix-free and injective. So the modified scheme $\mathsf{CtH}_{\Pi_n}(M') \oplus m$ is a universal hash when CTR is prefix-free and injective. Hence we can apply composition theorem to show that CtMac1 is a pseudorandom function. We have the following theorem.

Theorem 6. Let $\mathsf{CtMac1} := \mathsf{CtMac1}_{E_{K_1},E_{K_2}}$ be defined based on two independently chosen keyed blockcipher, and let CTR be a prefix-free and injective CFF. Then,

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{CtMac1}}(t,q,\hat{\ell}) \leq \frac{q^2}{2^{n-1}} + \mathbf{Adv}^{\mathrm{prp}}_E(t',\hat{\ell}q)$$

The proof of theorem 6 follows directly from theorem 1 and 5. The above construction is deterministic.

CtMAC1 is based on HtPRF paradigm. Now we use HtM paradigm to define probabilistic and stateful MACs that are motivated from XOR-MACs. Let R be a seed (either random or nonce). We define $\mathsf{CtMac2}_{E_{K_1},E_{K_2}}(R,M) = E_{K_2}(R) \oplus \mathsf{CtH}_{E_{K_1}}(M)$. By using Theorem 2 we can have following results on the MAC security if probabilistic and stateful CtMac2.

Theorem 7. Let $\mathsf{CtMac2} := \mathsf{CtMac2}_{E_{K_1},E_{K_2}}$ be defined based on two independently chosen keyed blockcipher. Then,

$$\mathbf{Adv}^{\mathrm{forge}}_{\mathsf{CtMac2}^{st}}(t,q_m,q_v,\hat{\ell}) \leq \frac{0.5q^2}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_E(t',\hat{\ell}(q_m+q_v)) + \frac{q_v}{2^n}$$

$$\mathbf{Adv}^{\mathrm{forge}}_{\mathsf{CtMac2}^{\$}}(t,q_m,q_v,\hat{\ell}) \leq \frac{q^2}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_E(t',\hat{\ell}(q_m+q_v)) + \frac{q_v}{2^n}$$

Remark 4. Note that CtMAC2 can be shown to have full MAC security [4] when instantiated with a random function. The security reduces to $q^2/2^n$ when block ciphers are used for instantiation. This is due to the PRF-PRP switching lemma (Equation 1). Since we give a concrete implementation of stateful MAC based on AES block cipher, we have to switch from PRF to PRP model.

4.4 Necessity of Prefix-free and Injectivity

In all the security results discussed in this section, the security proof is implied by the prefix-free and injective property of the CFF CTR. In other words, prefix-free and injective properties are sufficient for security. Now we show that these properties are also necessary in case of CtH, CtMAC1 and CTMAC2.

> **Case 1:** Suppose CTR doesn't have injective property. As CTR is a deterministic function (encodings are deterministic), one can easily find two distinct messages M, M' such that $\mathsf{CTR}(M) = \mathsf{CTR}(M')$.
> **Case 2:** Suppose CTR is not prefix-free, i.e., for some $\ell \leq L$, we have $i < j \in \mathbb{N}$ such that $\mathsf{ctr}_\ell(i)$ is a prefix of $\mathsf{ctr}_\ell(j)$. Thus, we can find two pairs (c,c') and (d,d') such that $\mathsf{ctr}_\ell(i)\|c = \mathsf{ctr}_\ell(j)\|c'$ and $\mathsf{ctr}_\ell(i)\|d = \mathsf{ctr}_\ell(j)\|d'$. Using these two pairs we can construct two distinct messages M and M' such that the i-th encoded block collides with the j-th encoded block for both M and M', i.e., they cancel out each other in $\mathsf{CtH}(M)$ and $\mathsf{CtH}(M')$.

Therefore in both the cases we can find M, M', such that $\mathsf{Pr}[\mathsf{CtH}(M) \oplus \mathsf{CtH}(M') = 0] = 1$. This leads to trivial forgery attacks on CtMAC1 and CtMAC2. Hence prefix-free and injective property are necessary.

5 Security Proofs of CtHAIFA and CtH

5.1 Proof of Theorem 4 [2PI Security of CtHAIFA]

We use a similar approach as used in [12] for HAIFA based on random oracle. Let $\mathcal{A}$ be a second preimage adversary for CtHAIFA that makes q distinct queries to the underlying ideal cipher. Note that in the ideal cipher model, $\mathcal{A}$ can make both encryption and decryption queries to the underlying block cipher. We also assume that the adversary computes $\mathsf{CtHAIFA}(M)$. The adversary can do this by making encryption query (M_i, h_{i-1}) and storing $(h_{i-1}, M_i, h_i := e_{M_i}(h_{i-1}) \oplus h_{i-1})$ for all $i \in [b]$. Actually $\mathcal{A}$ can compute any $\mathsf{DM}(h,m)$ by making encryption query (m,h) to e. Let the sequence of intermediate chaining values for the challenge message M be $(h_0, h_1, \ldots, h_b, h_{b+1})$.

We denote the i-th query of $\mathcal{A}$ by the tuple $(x_i, m_i, c_i, y_i, o_i)$ such that c_i is the counter value, m_i is the message value, x_i is some chaining value, and $y_i = \mathsf{DM}(x_i, c_i\|m_i) \oplus x_i$. Further,

1) if the i-th query is an encryption query then $o_i = 0$ and $\mathcal{A}$ queried $c_i\|m_i, x_i$ and got $y_i := e_{c_i\|m_i}(x_i)$.
2) if the i-th query is a decryption query then $o_i = 1$ and $\mathcal{A}$ queried $c_i\|m_i, y_i$ and got $x_i := e^{-1}_{c_i\|m_i}(y_i)$.

Now suppose that $\mathcal{A}$ produces a valid second preimage M'. We denote this event by Win. Let Win_1 and Win_2 denote the events $\mathsf{Win} \wedge |M| \neq |M'|$ and $\mathsf{Win} \wedge |M| = |M'|$, respectively. Clearly $\mathsf{Win} = \mathsf{Win}_1 \cup \mathsf{Win}_2$. Therefore,

$$\mathsf{Pr}[\mathsf{Win}] \leq \mathsf{Pr}[\mathsf{Win}_1] + \mathsf{Pr}[\mathsf{Win}_2].$$

We bound the two events as follows:

1) If Win_1 is true then $|M| \neq |M^{'}|$. In this case we must have $\langle |M| \rangle_n \neq \langle |M^{'}| \rangle_n$. Therefore the adversary found a second preimage for the last block,

i.e., there exist a query $i \in \{1, \ldots, q\}$ such that $c_i \| m_i = \langle |M'| \rangle_n \neq \langle |M| \rangle_n$ and $y_i \oplus x_i = h_{b+1}$. This is possible with probability $\frac{1}{2^n - i + 1}$. Bounding over q queries we have $\mathsf{Pr}[\mathsf{Win}_1] \leq \frac{q}{2^n - q}$.

2) If Win_2 is true then $|M| = |M^{'}|$. In this case, if $\mathcal{A}$ succeeds, then $\mathcal{A}$ was successful in creating a connection to some intermediate chaining value h_j. Suppose this connection is established at the i-th query, then c_i must be equal to $\mathsf{ctr}_\ell(j)$ and $y_i \oplus x_i = h_j$. This probability is again $\frac{1}{2^n - i + 1}$. Bounding over q queries we have $\mathsf{Pr}[\mathsf{Win}_2] \leq \frac{q}{2^n - q}$.

Combining 1 and 2 we have,

$$\mathsf{Pr}[\mathsf{Win}] \leq \frac{2q}{2^n - q} \leq \frac{3q}{2^n}.$$

The result follows. ■

5.2 Proof of Theorem 5 [AXU security of CtH]

Let $M \neq M'$ and $\mathsf{CTR}(M) = S = \{X_1, \ldots, X_b\}$, $\mathsf{CTR}(M') = S' = \{X_1, \ldots, X_{b'}\}$. As CTR is injective, there exists at least one block X which appears exactly once in S or S'. Thus, for any $\delta \in \{0,1\}^n$,

$$\begin{aligned}&\mathsf{Pr}[\mathsf{CtH}_{\Pi_n,\mathsf{CTR}}(M) \oplus \mathsf{CtH}_{\Pi_n,\mathsf{CTR}}(M') = \delta] \\ &\quad = \mathsf{Pr}[\Pi_n(X) = \oplus_{x \in S_1 \cup S_2 \setminus \{X\}} \Pi_n(x)].\end{aligned}$$

The result follows by conditioning on the outputs of the random permutation on all blocks except X. ■

Remark 5. Note that the above result has been proved implicitly in [4], [5], [6] for the standard counter. Our result generalizes their results for any prefix-free and injective counter function family.

6 Comparison and Empirical Result

In this section, we compare the performance of the three candidate counter function families via their application in different CaE schemes. In particular, we present pipelined software implementation of counter based MACs, and serial implementation of HAIFA based on DM compression function. We instantiate the MAC and DM compression function using AES-128 [53] block cipher. AES cipher is used to exploit the AES-NI instruction set available on new generation Intel microprocessors. The source code is publicly available at [54].

6.1 Platform Setup

As mentioned earlier we use AES-128 (key size 128 bits) as the underlying block cipher. We use performance data for all inputs of length $\ell = 2^k$ bits where $7 \leq k \leq 20$. We implemented all the schemes on Intel's Sandy Bridge microarchitecture using AES-NI and SSE4 intructions. All measurements were taken on a single core of Intel Xeon E5-2640 processor at 2.5Ghz with Turbo Boost disabled. The warmup parameter is 250000 and the data is averaged over 1000000 repetitions. All results will be either in number of cycles per byte or number of cycles. The baseline performance for AES using AES-NI instruction set is presented in Table 2.

TABLE 2: Baseline CPB of AES using AES-NI.

	encryption (cycles/byte)	**key scheduling** (cycles)
(AES, serial)	6.7	117
(AES, parallel)	0.69	117

6.2 Implementing Counter Function Families

The three counter function families basically differ in their selection of counter sizes. STD^s is a standard counter with a fixed size s; STD^{opt} computes the (optimal) counter size s based on the length of the input; and VAR^r dynamically changes the counter size s based on the number of input blocks processed so far.

In a practical application STD^s requires a beforehand heuristic on the typical input lengths to be encountered. In situations where the input lengths are inconstant, this scheme may not be that efficient. If the input length is known beforehand, then STD^{opt} is the best option as it fixes the optimal (machine-dependent) counter size. VAR^r has an edge over the other two in the worst case scenario, i.e., when neither the input length is known nor the inputs have predictable lengths. This scenario is quite frequent in server-side applications.

Although STD^{opt} and VAR^r are much more flexible in terms of the changes in the input length, they do require some book-keeping operations. STD^{opt} requires the computation of a suitable counter size for the given input length. Similarly, VAR^r requires some kind of mechanism to update the counter size when the current counter reaches its limit.

STD^s does not incur such book-keeping overheads, as it is fixed for the construction. Thus, STD^s with $s = 32$ uses 32-bit counter irrespective of whether the input size is 2^{10} or 2^{20} bits. For STD^{opt}, we pre-store the set of counter sizes ($\{8, 16, 32, 64\}$) along with their respective maximum input lengths. Although this incurs a small increment in code size, it reduces the number of micro-operations. For VAR^r, the overhead can be reduced significantly by reducing the number of times the counter size is increased. For example, we increase the size in steps of multiple of $8 \cdot 2^i$ (i.e. 8,16,32,64) (instead of $8 \cdot i$). All three counters were implemented in 64-bit registers.

Table 3 gives the book-keeping cost incurred by the three counters. The cost for STD^{opt} is significantly lower than the other two candidates as in this case the major book-keeping operation is executed only once per message. For VAR^r and STD^s the check for last message block accounts for most of the book-keeping cost. VAR^r has a complex counter update mechanism as compared to STD^s. This leads to a further increase in its book-keeping cost. Fig. 4 presents these characteristics graphically.

6.3 Performance of MACs

We summarise the performance results of the deterministic MACs in Table 4 and the stateful MACs in Table 5. We deliberately leave out the performance results for probabilistic MAC, as apart from the random number generation phase, it is exactly similar to the stateful MAC. Fig. 5 and 6 give graphical characteristics of the performance data. STD^s is implemented for $s = w \cdot 2^i$, where $w = 8$ bits is the word size, and i is varied over $\{0, 1, 2, 3\}$. Similar values are used

TABLE 3: Cycles per byte comparison between the book-keeping operations performed by the three CFF candidates.

Length	STD^s				STD^{opt,8}	VAR^{4,8}
	s = 8 bits	s = 16 bits	s = 32 bits	s = 64 bits		
128B	0.17	0.20	0.22	0.27	0.33	0.18
256B	0.15	0.17	0.18	0.27	0.18	0.17
512B	0.14	0.15	0.16	0.23	0.13	0.15
1KB	0.13	0.14	0.16	0.22	0.09	0.15
2KB	0.14	0.13	0.15	0.22	0.09	0.15
4KB	-	0.14	0.15	0.22	0.10	0.16
8KB	-	0.14	0.16	0.22	0.09	0.16
16KB	-	0.13	0.15	0.22	0.09	0.15
32KB	-	0.14	0.15	0.22	0.09	0.15
64KB	-	0.14	0.16	0.22	0.09	0.16
128KB	-	0.14	0.16	0.22	0.09	0.17
256KB	-	0.16	0.18	0.23	0.09	0.18
512KB	-	0.17	0.19	0.24	0.09	0.19
1MB	-	-	0.19	0.24	0.08	0.19

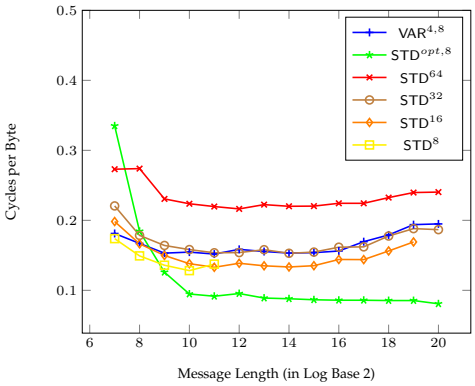

Fig. 4: CPB plot for book-keeping operations.

in STD^{opt} and VAR^r while choosing (or expanding) the counter size.

TABLE 4: Cycles per byte comparison between the software performance of the three CtMAC1 candidates.

Length	STD^s				STD^{opt,8}	VAR^{4,8}
	s = 8 bits	s = 16 bits	s = 32 bits	s = 64 bits		
128B	1.26	1.74	1.79	1.96	1.37	1.24
256B	1.25	1.26	1.29	1.74	1.27	1.26
512B	1.00	1.02	1.18	1.63	1.00	1.03
1KB	0.88	0.98	1.07	1.57	0.88	0.91
2KB	0.83	0.92	1.05	1.54	0.79	0.91
4KB	-	0.90	1.02	1.52	0.86	0.89
8KB	-	0.88	1.01	1.51	0.83	0.87
16KB	-	0.88	1.00	1.51	0.83	0.88
32KB	-	0.88	1.01	1.51	0.82	0.87
64KB	-	0.88	1.00	1.51	0.81	0.89
128KB	-	0.87	1.00	1.51	0.82	0.95
256 KB	-	0.87	1.01	1.51	0.82	0.98
512KB	-	0.88	1.01	1.51	0.82	1.00
1MB	-	-	1.01	1.51	0.95	1.01

Some notes on the characteristics. It is evident from Fig. 5 and 6 that STD^{opt} gives the fastest MACs among all the three candidates. Also observe that when the input length is comparable to the counter size both STD^s and VAR^r closely resemble the performance curve for STD^{opt}. For instance, in Table 4, look at the entries corresponding to the range 8 KB (2^{16}) to 64 KB (2^{19} bits). In this range all three counters offer

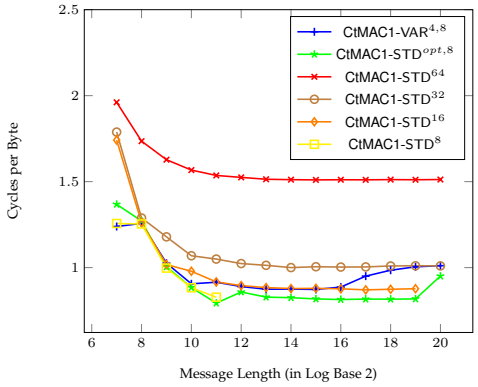

Fig. 5: CPB plot for the three CtMAC1 candidates.

similar cpb of around 0.8-0.9.

For a fixed value of s in STD^s, VAR^r outperforms STD^s, until the input length is significantly smaller than 2^s, or the counter size in VAR^r is smaller than s. For example, for $s = 32$ VAR^r is faster than STD^s, when the input length $\ell \leq 128$KB (2^{20} bits). At 256 KB, the counter size in VAR^r expands to 32 bits which causes a change in the slope of the curve. Even when the input length lies in the optimal range the gain in using STD^s is marginal when compared to the flexibility offered by VAR^r over a diverse range of input lengths.

Note that the effective counter size in VAR^r is always r bits less than the actual counter size. This is because the first r bits are reserved. For example when the counter size is 16 bits, only 12 bits are used. This reduces the maximum message length for 16-bit counter from 2^{23} bits to 2^{19} bits.

TABLE 5: Cycles per byte comparison between the software performance of the three CtMAC2st candidates.

Length	STD^s				STD^{opt,8}	VAR^{4,8}
	s = 8 bits	s = 16 bits	s = 32 bits	s = 64 bits		
128B	1.21	1.28	1.47	2.05	1.31	1.23
256B	1.03	1.11	1.30	1.74	1.07	1.04
512B	0.94	1.04	1.11	1.62	0.94	1.00
1KB	0.91	0.92	1.08	1.58	0.87	0.94
2KB	0.84	0.90	1.03	1.54	0.79	0.89
4KB	-	0.89	1.02	1.53	0.85	0.88
8KB	-	0.88	1.01	1.52	0.82	0.87
16KB	-	0.88	1.01	1.51	0.82	0.87
32KB	-	0.87	1.01	1.51	0.82	0.87
64KB	-	0.87	1.00	1.51	0.81	0.89
128KB	-	0.87	1.00	1.52	0.81	0.95
256KB	-	0.87	1.00	1.52	0.81	0.99
512KB	-	0.88	1.01	1.52	0.82	1.01
1MB	-	-	1.01	1.52	0.95	1.02

6.4 Performance of HAIFA

We summarise the performance results of CtHAIFA in Table 6. The graphical representation is illustrated in Fig. 7. The comparison results are similar to those obtained earlier, in case of deterministic MACs and stateful MACs.

7 Conclusion

In this paper we presented a generalized notion of counter function families. We also tried to formalize the cryptographically significant properties required from a counter

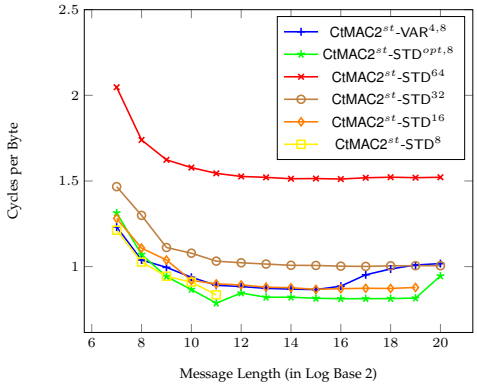

Fig. 6: CPB plot for the three CtMAC2st candidates.

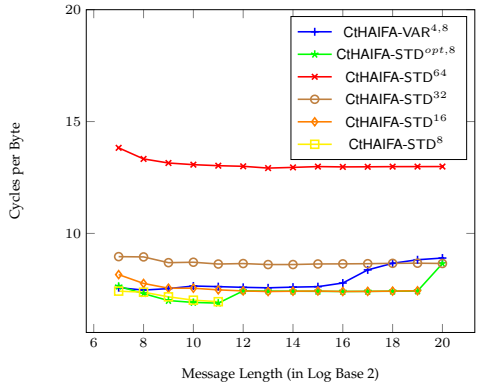

Fig. 7: CPB plot for the three CtHAIFA candidates.

TABLE 6: Cycles per byte comparison between the software performance of the three CtHAIFA candidates.

Length	STD^s				STD^{opt,8}	VAR^{4,8}
	s = 8 bits	s = 16 bits	s = 32 bits	s = 64 bits		
128B	7.42	8.16	8.96	13.83	7.64	7.58
256B	7.37	7.76	8.95	13.33	7.31	7.47
512B	7.17	7.55	8.69	13.15	7.00	7.53
1KB	7.02	7.55	8.71	13.07	6.92	7.65
2KB	6.95	7.48	8.63	13.03	6.88	7.62
4KB	-	7.43	8.65	13.00	7.43	7.59
8KB	-	7.40	8.61	12.92	7.42	7.57
16KB	-	7.42	8.61	12.95	7.42	7.60
32KB	-	7.43	8.63	12.99	7.41	7.62
64KB	-	7.41	8.64	12.97	7.41	7.78
128KB	-	7.41	8.65	12.98	7.41	8.37
256KB	-	7.42	8.66	12.99	7.42	8.66
512KB	-	7.44	8.67	12.99	7.43	8.82
1MB	-	-	8.66	12.99	8.65	8.90

scheme. Based on these properties we gave straightforward proofs for some CaE schemes such as HAIFA hash function, XOR universal hash and MACs.

Further we presented two efficient alternatives for the standard counter scheme. One of these alternatives, namely, the VAR^r counter function family is a nice application of universal codes in cryptography. To the best of knowledge this relationship has not been studied before. Finally we gave software comparison of MAC and HAIFA constructions based on the three counter function families. Based on our findings we can set the following guidelines for choosing one among the three counter function families for a specific area of application:

1) STD^{opt} is the best option when the input length is known beforehand.
2) If the input length is not known:
   a) VAR^r is better when the message length can vary arbitrarily over a wide range.
   b) STD^s works best when the average input length is close to the application limit.

In general, VAR^r seems to be the best candidate that works for wide range of input lengths with comparable performance curve.

The paper shows following new research directions.

1) **Parallel vs Serial mode.** In this paper we defined a general notion of counter function families and then studied prefix-free and injective counters. Although we studied HAIFA hash function which is a serial mode CaE scheme, our study was mainly motivated by those parallel mode CaE schemes such as XOR-MAC [4] and LightMAC [6]. We showed that prefix-free and injective properties are necessary for these parallel mode of operations. It would be interesting to investigate whether the prefix-free property can be relaxed in case of serial mode of operation.
2) **New CFF schemes.** We have shown a general way of using universal codes as CFFs. Can we have other such generic techniques? Also, it would be interesting to further investigate the CFFs mentioned in this paper.
3) **Application in CaI schemes.** Although counter size does not affect the rate of CaI schemes, it would be interesting to see whether we can get some advantage by replacing the standard counter function with one of STD^{opt} or VAR^r. One can also investigate generic proof techniques for CaI schemes based on CFF properties.

REFERENCES

[1] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication," NIST, U. S. Department of Commerce, NIST Special Publication 800-38B, 2005.
[2] D. A. McGrew and J. Viega, "The security and performance of the galois/counter mode (GCM) of operation," in *Proc. INDOCRYPT 2004*, 2004, pp. 343–355.
[3] P. Rogaway and T. Shrimpton, "Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem," in *Proc. EUROCRYPT 2006*, 2006, pp. 373–390.
[4] M. Bellare, R. Guérin, and P. Rogaway, "XOR macs: New methods for message authentication using finite pseudorandom functions," in *Proc. CRYPTO 1995*, 1995, pp. 15–28.
[5] D. J. Bernstein, "How to Stretch Random Functions: The Security of Protected Counter Sums," *J. Cryptol.*, vol. 12, pp. 185–192, 1999.
[6] A. Luykx, B. Preneel, E. Tischhauser, and K. Yasuda, "A MAC mode for lightweight block ciphers," in *Proc. Fast Software Encryption - FSE 2016*, 2016, pp. 43–59.
[7] E. Biham and O. Dunkelman, "A Framework for Iterative Hash Functions - HAIFA," *IACR Cryptology ePrint Archive*, 2007.
[8] J. Kelsey and B. Schneier, "Second preimages on n-bit hash functions for much less than 2^n work," in *Proc. EUROCRYPT 2005*, 2005, pp. 474–490.
[9] E. Andreeva, C. Bouillaguet, O. Dunkelman, P. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer, "New second-preimage attacks on hash functions," *J. Cryptol.*, vol. 29, pp. 657–696, 2016.
[10] R. C. Merkle, "One way hash functions and DES," in *Proc. CRYPTO 1989*, 1989, pp. 428–446.

[11] I. Damgård, "A design principle for hash functions," in *Proc. CRYPTO 1989*, 1989, pp. 416–427.
[12] C. Bouillaguet and P.-A. Fouque, "Practical Hash Functions Constructions Resistant to Generic Second Preimage Attacks Beyond the Birthday Bound," Submitted for publication, (2010).
[13] D. J. Bernstein, "Stronger Security Bounds for Wegman-Carter-Shoup Authenticators," in *Proc. EUROCRYPT 2005*, 2005, pp. 164–180.
[14] M. N. Wegman and L. Carter, "New hash functions and their use in authentication and set equality," *J. Comput. Syst. Sci.*, vol. 22, pp. 265–279, 1981.
[15] L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, pp. 143–154, 1979.
[16] V. Shoup, "A Composition Theorem for Universal One-Way Hash Functions," in *Proc. EUROCRYPT 2000*, 2000, pp. 445–452.
[17] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, pp. 1098–1101, 1952.
[18] D. Salomon, *Variable-length Codes for Data Compression*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
[19] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423,623–656, 1948.
[20] S. Gueron, "Intel® advanced encryption standard (aes) new instructions set," Intel Corporation, Tech. Rep., 2010.
[21] B. Preneel, R. Govaerts, and J. Vandewalle, "Hash Functions Based on Block Ciphers: A Synthetic Approach," in *Proc. CRYPTO 1993*, 1993, pp. 368–378.
[22] B. Preneel, "Davies–Meyer," in *Encyclopedia of Cryptography and Security*. Springer-Verlag, 2011.
[23] J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block-cipher-based hash-function constructions from PGV," in *Proc. CRYPTO 2002*, 2002, pp. 320–335.
[24] J. Coron, Y. Dodis, C. Malinaud, and P. Puniya, "Merkle-Damgård Revisited: How to Construct a Hash Function," in *Proc. CRYPTO 2005*, 2005, pp. 430–448.
[25] J. Coron, J. Patarin, and Y. Seurin, "The random oracle model and the ideal cipher model are equivalent," in *Proc. CRYPTO 2008*, 2008, pp. 1–20.
[26] P. Rogaway, "Bucket Hashing and Its Application to Fast Message Authentication," *J. Cryptol.*, vol. 12, pp. 91–115, 1999.
[27] S. Halevi and H. Krawczyk, "MMH: Software Message Authentication in the Gbit/Second Rates," in *Proc. Fast Software Encryption - FSE 1997*, 1997, pp. 172–189.
[28] P. Sarkar, "A New Multi-Linear Universal Hash Family," *Des. Codes Cryptography*, vol. 69, pp. 351–367, 2013.
[29] S. Winograd, "A New Algorithm for Inner Product," *IEEE Trans. Comput.*, vol. 17, pp. 693–694, 1968.
[30] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and Secure Message Authentication," in *Proc. CRYPTO 1999*, 1999, pp. 216–233.
[31] T. Krovetz, "Message Authentication on 64-bit Architectures," *IACR Cryptology ePrint Archive*, no. 37, 2006.
[32] K. Minematsu and T. Matsushima, "New Bounds for PMAC, TMAC, and XCBC," in *Proc. Fast Software Encryption - FSE 2007*, 2007, pp. 434–451.
[33] M. Bellare and P. Rogaway, "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs," in *Proc. EUROCRYPT 2006*, 2006, pp. 409–426.
[34] D. Chang and M. Nandi, "A Short Proof of the PRP/PRF Switching Lemma," *IACR Cryptology ePrint Archive*, no. 78, 2008.
[35] M. Bellare, J. Kilian, and P. Rogaway, "The Security of The Cipher Block Chaining Message Authentication Code," *J. Comput. Syst. Sci.*, vol. 61, pp. 362–399, 2000.
[36] W. F. Ehrsam, C. H. W. Meyer, J. L. Smith, and W. L. Tuchman, "Message Verification and Transmission Error Detection by Block Chaining," U.S. Patent 4 074 066, 1976.
[37] J. Black and P. Rogaway, "A Block-Cipher Mode of Operation for Parallelizable Message Authentication," in *Proc. EUROCRYPT 2002*, 2002, pp. 384–397.
[38] A. Berendschot, B. den Boer, J. Boly, A. Bosselaers, J. Brandt, D. Chaum, I. Damgård, M. Dichtl, W. Fumy, M. van der Ham, C. Jansen, P. Landrock, B. Preneel, G. Roelofsen, P. de Rooij, and J. Vandewalle, "Final Report of Race Integrity Primitives," *Lecture Notes in Computer Science, Springer-Verlag, 1995*, vol. 1007, 1995.
[39] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proc. CRYPTO 1996*, 1996, pp. 1–15.
[40] É. Jaulmes, A. Joux, and F. Valette, "On the Security of Randomized CBC-MAC Beyond the Birthday Paradox Limit: A New Construction," in *Proc. Fast Software Encryption - FSE 2002*, 2002, pp. 237–251.
[41] M. Bellare, O. Goldreich, and H. Krawczyk, "Stateless evaluation of pseudorandom functions: Security beyond the birthday barrier," in *Proc. CRYPTO 1999*, 1999, pp. 270–287.
[42] K. Minematsu, "How to thwart birthday attacks against macs via small randomness," in *Proc. Fast Software Encryption - FSE 2010*, 2010, pp. 230–249.
[43] É. Jaulmes and R. Lercier, "FRMAC, a proc. fast randomized message authentication code," *IACR Cryptology ePrint Archive*, no. 166, 2004.
[44] J. Black and M. Cochran, "MAC reforgeability," in *Proc. Fast Software Encryption - FSE 2009*, 2009, pp. 345–362.
[45] B. Cogliati and Y. Seurin, "EWCDM: an efficient, beyond-birthday secure, nonce-misuse resistant MAC," in *Proc. CRYPTO 2016*, 2016, pp. 121–149.
[46] R. M. Fano, "The transmission of information," Research Laboratory of Electronics at MIT, Technical Report No. 65, 1949.
[47] P. Elias, "Minimum times and memories needed to compute the values of a function," *J. Comput. Syst. Sci.*, vol. 9, pp. 196–212, 1974.
[48] *ISO/IEC 10918-5:2013, Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)*, ISO/IEC, 2013.
[49] *ZIP File Format Specification*, PKWARE Inc., 2014.
[50] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inf. Theory*, vol. 21, pp. 194–203, 1975.
[51] A. S. Fraenkel and S. T. Klein, "Robust universal complete codes for transmission and compression," *Discrete Applied Mathematics*, vol. 64, pp. 31–55, 1996.
[52] R. Dean, "Formal Aspects of Mobile Code Security," Ph.D. dissertation, Princeton University, 1999.
[53] *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, NIST, U. S. Department of Commerce Fedral Information Processing Standards Publication 197, 2001.
[54] A. Dutta, A. Jha, and M. Nandi. [Online]. Available: https://github.com/ashwinj/counter-as-encodings