OMD: A Compression Function Mode of Operation for Authenticated Encryption

Simon Cogliani[1], Diana-Ştefania Maimuţ[1], David Naccache[1], Rodrigo Portella do Canto[2], Reza Reyhanitabar[3](✉), Serge Vaudenay[3], and Damian Vizár[3]

[1] ENS, Paris, France
Diana.Maimut@ens.fr
[2] Université Paris II - Panthéon-Assas, Paris, France
[3] EPFL, Lausanne, Switzerland
reza.reyhanitabar@epfl.ch

Abstract. We propose the Offset Merkle-Damgård (OMD) scheme, a mode of operation to use a compression function for building a nonce-based authenticated encryption with associated data. In OMD, the parts responsible for privacy and authenticity are tightly coupled to minimize the total number of compression function calls: for processing a message of ℓ blocks and associated data of a blocks, OMD needs $\ell + a + 2$ calls to the compression function (plus a single call during the whole lifetime of the key). OMD is provably secure based on the standard pseudorandom function (PRF) property of the compression function. Instantiations of OMD using the compression functions of SHA-256 and SHA-512, called OMD-SHA256 and OMD-SHA512, respectively, provide much higher quantitative level of security compared to the AES-based schemes. OMD-SHA256 can benefit from the new Intel SHA Extensions on next-generation processors.

Keywords: Authenticated encryption · Provable security · Standard model · Intel SHA Extensions

1 Introduction

An authenticated encryption (AE) scheme delivers on two complementary data security goals: confidentiality (privacy) and integrity (authenticity). Historically, these goals were achieved by combining two separate cryptographic primitives, an encryption scheme to ensure privacy and a message authentication code (MAC) to guarantee authenticity [4,5]. This generic composition paradigm is neither most efficient (for instance, it requires processing the input stream at least twice) nor most robust to implementation errors [8,21,27]. The notion of AE, as a desirable primitive in its own right, was originally formalized in [4,6,17] (as a probabilistic algorithm) and later developed to include notions of nonce-based AE [25], nonce-based AE with associated data (AEAD) [22,24], deterministic AE

A. Joux and A. Youssef (Eds.): SAC 2014, LNCS 8781, pp. 112–128, 2014.
DOI: 10.1007/978-3-319-13051-4_7

(DAE) [26] (providing a solution to nonce-misuse resistance) and online DAE (online nonce-misuse resistant AE) [11].

Importance of useable AE to practice, and to some extent, difficulty of getting it right, is evident from the number of standards that were developed over the years, specifying different methods; for instance, the CCM method is specified in IEEE 802.11i, IPsec ESP and IKEv2 and NIST SP 800-38C; the GCM method is specified in NIST SP 800-38D; the EAX method is specified in ANSI C12.22; and ISO/IEC 19772:2009 defines six methods including five dedicated AE designs and one generic composition method, namely Encrypt-then-MAC. Surprisingly, the way that the latter generic method is specified in ISO/IEC 19772:2009 was very recently shown to be flawed, opening way for incorrect and insecure implementations [21].

AE schemes have been studied for over a decade, yet the topic remains a highly active and interesting area of research as evidenced by the recently initiated CAESAR competition by the cryptographic community. The competition aims to boost public discussions towards a better understanding of AE designs and to identify a portfolio of efficient and secure AE schemes by mid-December 2017 [7].

In this paper, we present a new nonce-based AEAD, called Offset Merkle-Damgård (OMD), offering several attractive features. Unlike the mainstream schemes which are either blockcipher-based or permutation-based schemes, OMD is designed as a mode of operation for a compression function. The motivation for this is manifold: (1) the cryptographic community has spent more than two decades on public research and standardization activities on hash functions resulting to development of a rich source of secure and efficient compression functions; (2) the standard SHA family of algorithms is heavily employed in many of the most common cryptographic applications and one can easily use off-the-shelf highly optimized implementations of these functions [12,13]; (3) Intel has recently introduced new instructions that support performance acceleration of SHA-1 and SHA-256 on next-generation processors [14]; (4) we believe that having a diverse set of AE schemes based on different primitives can be interesting from a practical viewpoint, providing the opportunity to choose among the AE algorithms based on what primitives have already been available and implemented and to reuse them.

Some of the interesting features of OMD, and its instantiations OMD-SHA256 and OMD-SHA512, are as follows:

Provable Security in the Standard Model. OMD achieves its security goals (privacy and authenticity) provably, based on the standard assumption that its underlying keyed compression function is PRF, an assumption which is among the most well-known and widely-used assumptions [2]. From a theoretical point of view, this is an advantage compared to permutation-based AE schemes whose security proofs rely on the *ideal permutation* assumption.

High Quantitative Security Level. When implemented with an off-the-shelf compression function such as those of the standard SHA family [1], OMD can achieve

much higher security level compare to AES-based schemes. For example, the proven security of OMD-SHA256 and OMD-SHA512 falls off in about $\frac{\sigma^2}{2^{256}}$ and $\frac{\sigma^2}{2^{512}}$, respectively, where σ is the total number of calls to the compression function. In comparison, for the same key size and tag size, the proven security of all the standardized blockcipher-based AE schemes using AES (e.g. all five dedicated schemes specified in ISO/IEC 19772:2009) falls off in about $\frac{\sigma'^2}{2^{128}}$ where σ' is the total number of calls to AES. We note that it is possible to get blockcipher-based AE schemes with (high) beyond birthday-bound security, but the existing schemes with beyond birthday-bound security have a degraded efficiency [15,16,19,20].

Online. OMD encryption is online; that is, it outputs a stream of ciphertext as a stream of plaintext arrives with a constant latency and using constant memory. After receiving an indication that the plaintext is over, the final part of ciphertext together with the tag is output. OMD decryption is *internally* online: one can generate a stream of plaintext bits as the stream of ciphertext bits comes in, but no part of the plaintext stream will be returned before the whole ciphertext stream is decrypted and the tag is verified to be correct.

Flexible Parameters. OMD-SHA256 can support any key length up to 256 bits, tag length up to 256 bits, and nonce length up to 255 bits. OMD-SHA512 can support any key length up to 512 bits, tag length up to 512 bits, and nonce length up to 511 bits. These upper bounds on the parameters' length will satisfy the required security level of almost any imaginable application today and well beyond. The lower bounds on the parameters' lengths should be selected based on the specific security level sought by an application; for instance, most applications would not use keys shorter than 128 bits, tags shorter than 32 bits and nonce shorter than 64 bits.

Organization of the paper. Notations and preliminary concepts are presented in Sect. 2. Security goals are defined in Sect. 3. Section 4 provides the specification of the OMD mode of operation. In Sect. 5, we provide the security analysis of OMD. Section 6 describes our recommended instantiations OMD-SHA256 and OMD-SHA512.

2 Preliminaries

Notations. If $\mathcal{S}$ is a finite set, $x \xleftarrow{\$} S$ means that x is chosen from S uniformly at random. $X \leftarrow Y$ is used for denoting the assignment statement where the value of Y is assigned to X. The set of all binary strings of length n bits (for some positive integer n) is denoted as $\{0,1\}^n$, the set of all binary strings whose lengths are variable but upper-bounded by L is denoted by $\{0,1\}^{\leq L}$ and the set of all binary strings of arbitrary but finite length is denoted by $\{0,1\}^*$. For two strings X and Y we use $X||Y$ and XY analogously to denote the string obtained by concatenating Y to X. For an m-bit binary string $X = X_{m-1} \cdots X_0$ we denote

the left-most bit by $\mathsf{msb}(X) = X_{m-1}$ and the right-most bit by $\mathsf{lsb}(X) = X_0$; let $X[i \cdots j] = X_i \cdots X_j$ denote a substring of X, for $0 \leq j \leq i \leq (m-1)$. Let $1^n 0^m$ denote concatenation of n ones by m zeros. For a non-negative integer i let $\langle i \rangle_m$ denote binary representation of i by an m-bit string.

For a binary string $X = X_{m-1} \cdots X_0$, let $X \ll n$ denote the left-shift operation, where the n left-most bits are discarded and the n vacated right bits are set to 0. We let $X \gg n$ denote the (unsigned) right-shift operation where the n right-most bits are discarded and the n vacated left bits are set to 0. We let $X \gg_s n$ denote the *signed* right-shift operation where the n right-most bits are discarded and the n vacated left bits are filled with the left-most bit (which is considered as the sign bit); for example, $1001100 \gg_s 3 = 1111001$. If the left-most bit of X is 0 then we have $X \gg_s n = X \gg n$. Let $\mathtt{ntz}(i)$ denote the number of trailing zeros (i.e. the number of rightmost bits that are zero) in the binary representation of a positive integer i.

The special symbol $\perp$ means that the value of a variable is undefined; we also overload this symbol and use it to signify an error. Let $|Z|$ denote the number of elements of Z if Z is a set, and the length of Z in bits if Z is a binary string. For $X \in \{0,1\}^*$ let $X[1]||X[2] \cdots ||X[m] \xleftarrow{b} X$ denote partitioning X into blocks $X[i]$ such that $|X[i]| = b$ for $1 \leq i \leq m-1$ and $|X[m]| \leq b$; let $m = |X|_b$ denote length of X in b-bit blocks.

For two binary strings $X = X_{m-1} \cdots X_0$ and $Y = Y_{n-1} \cdots Y_0$, the notation $X \oplus Y$ denotes bitwise xor of $X_{m-1} \cdots X_{m-1-\ell}$ and $Y_{n-1} \cdots Y_{n-1-\ell}$ where $\ell = min\{m-1, n-1\}$. Clearly, if X and Y have the same length then $X \oplus Y$ simply means their usual bitwise xor. The empty string is denoted by ε and we let $|\varepsilon| = 0$. For any string X, define $X \oplus \varepsilon = \varepsilon \oplus X = \varepsilon$.

The Finite Field with 2^n Points. Let $(GF(2^n), \oplus, \cdot)$ denote the Galois Field with 2^n points. When considering a point α in $GF(2^n)$ it can be represented in any of the following equivalent ways: (1) as an integer between 0 and 2^n, (2) as a binary string $\alpha_{n-1} \cdots \alpha_0 \in \{0,1\}^n$, or (3) as a formal polynomial $\alpha(X) = \alpha_{n-1} X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ with binary coefficients. The addition "$\oplus$" and multiplication "$\cdot$" of two field elements in $GF(2^n)$ are defined as usual (e.g., see [25]). For $GF(2^{256})$ we use $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, and for $GF(2^{512})$ we use $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$ as the irreducible polynomials used in the field multiplications. It is easy to multiply an arbitrary field element α by the element 2 (i.e. X). For example, in $GF(2^{256})$ using $P_{256}(X)$ the doubling operation can be described as follows:

$$2 \cdot \alpha = \begin{cases} \alpha \ll 1 & \text{if } \mathsf{msb}\ (\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{245}10000100101 & \text{if } \mathsf{msb}\ (\alpha) = 1 \end{cases} \quad (1)$$

$$= (\alpha \ll 1) \oplus ((\alpha \gg_s 255) \wedge 0^{245}10000100101) \quad (2)$$

We note that the results computed in (1) and (2) are the same but an implementation using (2) will not be susceptible to the timing attacks unlike one which uses (1).

3 Definitions and Security Goals

As usual in the concrete-security definitions, we measure the insecurity of a scheme Π using the resource parametrized function $\mathbf{Adv}^{\mathsf{xxx}}_{\Pi}(\mathbf{r})$, denoting the maximal value of the adversarial advantage, $\mathbf{Adv}^{\mathsf{xxx}}_{\Pi}(\mathbf{r}) = max_{\boldsymbol{A}}\{\mathbf{Adv}^{\mathsf{xxx}}_{\Pi}(\boldsymbol{A})\}$, over all adversaries $\boldsymbol{A}$, against the xxx property of a primitive or scheme Π, that use resources bounded by $\mathbf{r}$. Let $\boldsymbol{A}$ be an adversary that returns a binary value; by $\boldsymbol{A}^{f(.)}(X) \Rightarrow 1$ we refer to the event that $\boldsymbol{A}$ on input X and access to an oracle function $f(.)$ returns 1.

Pseudorandom Functions (PRFs) and Tweakable PRFs. Let $\mathrm{Func}(m,n)$ be the set of all functions from m-bit strings to n-bit strings; i.e., $\mathrm{Func}(m,n) = \{f : \{0,1\}^m \rightarrow \{0,1\}^n\}$. A random function (RF) R with m-bit input and n-bit output is a function selected uniformly at random from $\mathrm{Func}(m,n)$. We denote this by $R \xleftarrow{\$} \mathrm{Func}(m,n)$.

Let $\mathrm{Func}^{\mathcal{T}}(m,n)$ be the set of all functions $\left\{\widetilde{f} : \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n\right\}$, where $\mathcal{T}$ is a set of tweaks. A tweakable RF with the tweak space $\mathcal{T}$, m-bit input and n-bit output is a map $\widetilde{R} : \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n$ selected uniformly at random from $\mathrm{Func}^{\mathcal{T}}(m,n)$; i.e. $\widetilde{R} \xleftarrow{\$} \mathrm{Func}^{\mathcal{T}}(m,n)$. Clearly, if $\mathcal{T} = \{0,1\}^t$ then $|\mathrm{Func}^{\mathcal{T}}(m,n)| = |\mathrm{Func}(m+t,n)|$, and hence, $\widetilde{R}$ can be instantiated using a random function R with $(m+t)$-bit input and n-bit output. We use $\widetilde{R}^{\langle T\rangle}(.)$ and $\widetilde{R}(T,.)$ interchangeably, for every $T \in \mathcal{T}$. Notice that each tweak T names a random function $\widetilde{R}^{\langle T\rangle} : \{0,1\}^m \rightarrow \{0,1\}^n$ and distinct tweaks name distinct (independent) random functions.

Let $F : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a keyed function and let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a keyed and tweakable function, where the key space $\mathcal{K}$ is some nonempty set. Let $F_K(.) = F(K,.)$ and $\widetilde{F}^{\langle T\rangle}_K(.) = \widetilde{F}(K,T,.)$. Let $\boldsymbol{A}$ be an adversary. Then:

$$\mathbf{Adv}^{\mathrm{prf}}_{F}(\boldsymbol{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{F_K(.)} \Rightarrow 1\right] - \Pr\left[R \xleftarrow{\$} \mathrm{Func}(m,n) : \boldsymbol{A}^{R(.)} \Rightarrow 1\right]$$

$$\mathbf{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(\boldsymbol{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{\widetilde{F}^{\langle\cdot\rangle}_K(.)} \Rightarrow 1\right] - \Pr\left[\widetilde{R} \xleftarrow{\$} \mathrm{Func}^{\mathcal{T}}(m,n) : \boldsymbol{A}^{\widetilde{R}^{\langle\cdot\rangle}(.)} \Rightarrow 1\right]$$

The resource parametrized advantage functions are defined accordingly, considering that the adversarial resources of interest here are the time complexity (t) of the adversary and the total number of queries (q) asked by the adversary (note that we just consider fixed-input-length functions, so the lengths of queries are fixed and known). We say that F is $(t,q;\epsilon)$-PRF if $\mathbf{Adv}^{\mathrm{prf}}_{F}(t,q) \leq \epsilon$. We say that $\widetilde{F}$ is $(t,q;\epsilon)$-tweakable PRF if $\mathbf{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t,q) \leq \epsilon$.

Syntax of an AEAD Scheme. A nonce-based authenticated encryption with associated data, AEAD for short, is a symmetric key scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is some non-empty finite set. The encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{C} \cup \{\perp\}$ takes four arguments, a secret key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, an associated data (a.k.a. header data) $A \in \mathcal{A}$ and a message $M \in \mathcal{M}$,

and returns either a ciphertext $\mathbb{C} \in \mathcal{C}$ or a special symbol $\perp$ indicating an error. The decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ takes four arguments $(K, N, A, \mathbb{C})$ and either outputs a message $M \in \mathcal{M}$ or an error indicator $\perp$.

For correctness of the scheme, it is required that $\mathcal{D}(K, N, A, \mathbb{C}) = M$ for any $\mathbb{C}$ such that $\mathbb{C} = \mathcal{E}(K, N, A, M)$. It is also assumed that if algorithms $\mathcal{E}$ and $\mathcal{D}$ receive parameter not belonging to their specified domain of arguments they will output $\perp$. We write $\mathcal{E}_K(N, A, M) = \mathcal{E}(K, N, A, M)$ and similarly $\mathcal{D}_K(N, A, \mathbb{C}) = \mathcal{D}(K, N, A, \mathbb{C})$.

We assume that the message and associated data can be any binary string of arbitrary but finite length; i.e. $\mathcal{M} = \{0,1\}^*$ and $\mathcal{A} = \{0,1\}^*$, but the key and nonce are some fixed-length binary strings, i.e. $\mathcal{N} = \{0,1\}^{|N|}$ and $\mathcal{K} = \{0,1\}^k$, where the positive integers $|N|$ and k are respectively the nonce length and the key length of the scheme in bits. We assume that $|\mathcal{E}_K(N, A, M)| = |M| + \tau$ for some positive fixed constant τ; that is, we will have $\mathbb{C} = C||\mathsf{Tag}$ where $|C| = |M|$ and $|\mathsf{Tag}| = \tau$. We call C the core ciphertext and Tag the tag.

Nonce Respecting Adversaries. Let $\boldsymbol{A}$ be an adversary. We say that $\boldsymbol{A}$ is nonce-respecting if it never repeats a nonce in its *encryption* queries. That is, if $\boldsymbol{A}$ queries the encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ on $(N^1, A^1, M^1) \cdots (N^q, A^q, M^q)$ then $N^1, \cdots, N^q$ must be distinct.

Privacy of AEAD Schemes. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AEAD scheme. Let $\boldsymbol{A}$ be a nonce-respecting adversary. $\boldsymbol{A}$ is provided with an oracle which can be either a real encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ such that on input (N, A, M) returns $\mathbb{C} = \mathcal{E}_K(N, A, M)$, or a fake encryption oracle $\$(\cdot,\cdot,\cdot)$ which on any input (N, A, M) returns $|\mathbb{C}|$ fresh random bits. The advantage of $\boldsymbol{A}$ in mounting a chosen plaintext attack (CPA) against the privacy property of Π is measured as follows:

$$\mathbf{Adv}^{\mathrm{priv}}_{\Pi}(\boldsymbol{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\boldsymbol{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1].$$

This privacy notion, also called indistinguishability of ciphertext from random bits under CPA (IND$-CPA), is defined originally in [25] and is a stronger variant of the classical IND-CPA notion [3,4] for conventional symmetric-key encryption schemes.

Authenticity of AEAD Schemes. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AEAD scheme. Let $\boldsymbol{A}$ be a nonce-respecting adversary. We stress that nonce-respecting is only regarded for the encryption queries; that is, $\boldsymbol{A}$ can repeat nonces during its decryption queries and it can also ask an encryption query with a nonce that was already used in a decryption query. Let $\mathcal{A}$ be provided with the encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ and the decryption oracle $\mathcal{D}_K(\cdot,\cdot,\cdot)$; that is, we consider adversaries that can mount chosen ciphertext attacks (CCA). We say that $\boldsymbol{A}$ forges if it makes a decryption query $(N, A, \mathbb{C})$ such that $\mathcal{D}_K(N, A, \mathbb{C}) \neq \perp$ and no previous encryption query $\mathcal{E}_K(N, A, M)$ returned $\mathbb{C}$.

$$\mathbf{Adv}^{\mathrm{auth}}_{\Pi}(\boldsymbol{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \boldsymbol{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot),\, \mathcal{D}_K(\cdot,\cdot,\cdot)} \text{ forges}].$$

This authenticity notion, also called integrity of ciphertext (INT-CTXT) under CCA attacks, is defined originally in [4].

Resource parameters for the adversary. Let an adversary $\boldsymbol{A}$ make encryption queries $(N^1, A^1, M^1) \cdots (N^{q_e}, A^{q_e}, M^{q_e})$ and decryption queries $(N'^1, A'^1, \mathbb{C}'^1) \cdots (N'^{q_v}, A'^{q_v}, \mathbb{C}'^{q_v})$. We define the resource parameters of $\boldsymbol{A}$ as $(t, q_e, q_v, \sigma_A, \sigma_M, \sigma_{A'}, \sigma_{\mathbb{C}'}, L_{max})$, where t is the time complexity, q_e and q_v are respectively the total number of encryption queries and decryption queries, L_{max} is the maximum length of each query in bits, $\sigma_A = \sum_{i=1}^{q_e} |A^i|$, $\sigma_M = \sum_{i=1}^{q_e} |M^i|$, $\sigma_{A'} = \sum_{i=1}^{q_v} |A'^i|$ and $\sigma_{\mathbb{C}'} = \sum_{i=1}^{q_v} (|\mathbb{C}'^i| - \tau)$.

We remind that absence of a resource parameter means that the parameter is irrelevant in the context and hence omitted.

The use of the aforementioned privacy (IND$-CPA) and authenticity (INT-CTXT) goals to define security of AE schemes dates back to [4] where it was shown that if an AE scheme satisfies the combination of IND-CPA and INT-CTXT properties then it will also fulfill indistinguishability under the strongest form of chosen-ciphertext attack (IND-CCA) which, in turn, is equivalent to non-malleability under chosen-ciphertext attack (NM-CCA).

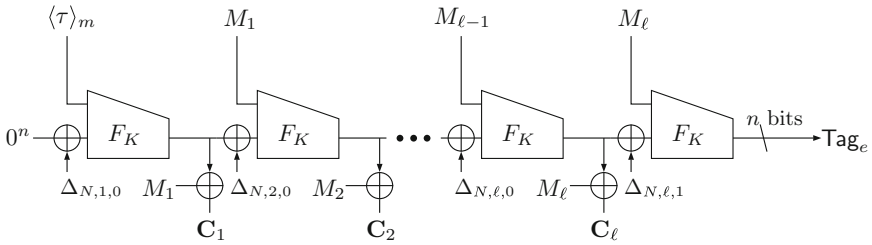
4 The OMD Mode of Operation

To use OMD one must specify a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and *fix* a tag length $\tau \leq n$; where the key space $\mathcal{K} = \{0,1\}^k$ and $m \leq n$. We let OMD$[F, \tau]$ denote the OMD mode of operation using the keyed compression function F_K and the fixed tag length τ.

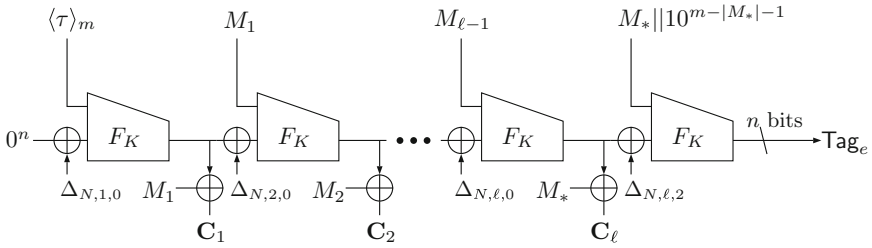
At first glance, imposing $m \leq n$ may look a bit odd as usually a compression function has a larger input block length than its output length, but we note that in practice, the compression function of standard hash functions (e.g. SHA-1 or the SHA-2 family) are keyless, therefore one will need to use k bits of their b-bit message block to get a keyed function. So, there will be no waste in each call to the compression function if $m = n$ and $b = n + k$; for example, when the key length is 256 bits and the compression function of SHA-256 is used.

Figure 1 depicts the encryption algorithm of OMD$[F, \tau]$. The construction of the decryption algorithm is straightforward and almost the same as the encryption algorithm except a tag comparison (verification) at the end of the decryption process. An algorithmic description of OMD$[F, \tau]$ is provided in Fig. 2.

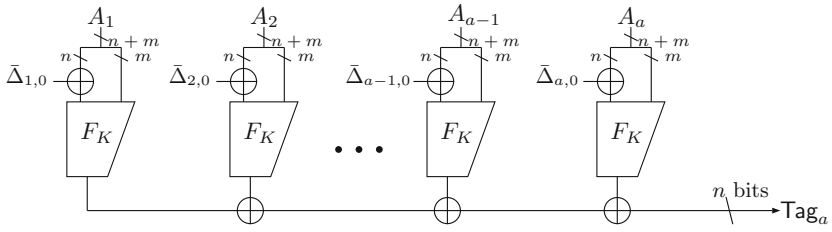
The encryption algorithm of OMD$[F, \tau]$ inputs four arguments (secret key $K \in \{0,1\}^k$, nonce $N \in \{0,1\}^{|N|}$, associated data $A \in \{0,1\}^*$, message $M \in \{0,1\}^*$) and outputs $\mathbb{C} = C||\mathsf{Tag} \in \{0,1\}^{|M|+\tau}$. The decryption algorithm of OMD$[F, \tau]$ inputs four arguments (secret key $K \in \{0,1\}^k$, nonce $N \in \{0,1\}^{|N|}$, associated data $A \in \{0,1\}^*$, ciphertext $C||\mathsf{Tag} \in \{0,1\}^*$) and either outputs the whole $M \in \{0,1\}^{|C|-\tau}$ at once or an error message ($\perp$). Note that we have either $C = C_1 \cdots C_\ell$ or $C = C_1 \cdots C_{\ell-1} C_*$ depending on whether the message length in bits is a multiple of the block length m or not, respectively.



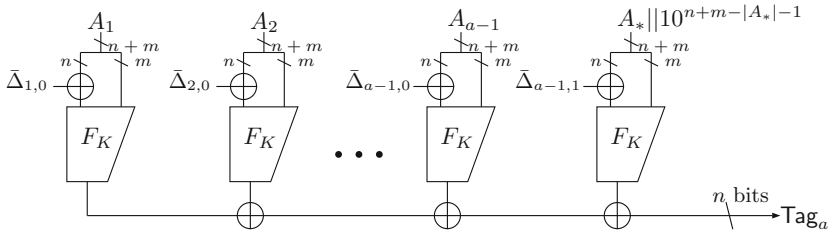
Encrypting a message whose length is a multiple of the block length. No padding is needed.



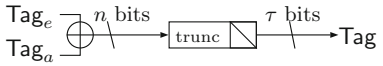
Encrypting a message whose length is not a multiple of the block length. The final message block is padded to make it a full block



Computing Tag_a for an associate data whose length is a multiple of the input length (i.e $|A_a| = n + m$).



Computing Tag_a for an associate data whose length is not a multiple of the input length. The final block is padded to make it a full block .



The Tag is computed as XOR of Tag_e and Tag_a truncated to τ bits.

Fig. 1. The encryption process of OMD$[F, \tau]$ using a keyed compression function $F_K : (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and a fixed tag length τ.

Computing the masking values. As seen from the description of OMD in Fig. 1, before each call to the underlying keyed compression function we xor a masking value denoted as $\Delta_{N,i,j}$ (the top and middle parts of Fig. 1) and $\bar{\Delta}_{i,j}$ (the bottom part of Fig. 1). In the following, we describe how these masks are generated.

There are different ways to compute the masking values to satisfy both the security and efficiency criteria; for example, we refer to [9,18,23]. We use the method proposed in [18]. In the following, all multiplications (denoted by "$\cdot$") are in $GF(2^n)$.

Initialization:
$\Delta_{N,0,0} = F_K(N||10^{n-1-|N|}, 0^m)$; $\bar{\Delta}_{0,0} = 0^n$; $L_* = F_K(0^n, 0^m)$; $L(0) = 4 \cdot L_*$, and $L(i) = 2 \cdot L(i-1)$ for $i \geq 1$. We note that the values $L(i)$ can be preprocessed and stored (for a fast implementation) in a table for $0 \leq i \leq \lceil \log_2(\ell_{max}) \rceil$, where ℓ_{max} is the bound on the maximum number of blocks in any input that can be encrypted or decrypted. Alternatively, (if there is a memory restriction) they can be computed on-the-fly for $i \geq 1$. It is also possible to precompute and store some values and then compute the others as needed on-the-fly.

Masking sequence for processing the message:
For $i \geq 1$: $\Delta_{N,i,0} = \Delta_{N,i-1,0} \oplus L(\mathtt{ntz}(i))$; $\Delta_{N,i,1} = \Delta_{N,i,0} \oplus 2 \cdot L_*$; and $\Delta_{N,i,2} = \Delta_{N,i,0} \oplus 3 \cdot L_*$.

Masking sequence for processing the associated data:
$\bar{\Delta}_{i,0} = \bar{\Delta}_{i-1,0} \oplus L(\mathtt{ntz}(i))$ for $i \geq 1$; and $\bar{\Delta}_{i,1} = \bar{\Delta}_{i,0} \oplus L_*$ for $i \geq 0$.

5 Security Analysis

Theorem 1 provides the security bounds of OMD.

Theorem 1. *Fix $n \geq 1$ and $\tau \in \{0, 1, \cdots, n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ be a PRF, where the key space $\mathcal{K} = \{0,1\}^k$ for $k \geq 1$ and $1 \leq m \leq n$. Then*

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{OMD}[F,\tau]}(t, q_e, \sigma_e, \ell_{max}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(t', 2\sigma_e) + \frac{3\sigma_e^2}{2^n}$$

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{OMD}[F,\tau]}(t, q_e, q_v, \sigma, \ell_{max}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(t', 2\sigma) + \frac{3\sigma^2}{2^n} + \frac{q_v \ell_{max}}{2^n} + \frac{q_v}{2^\tau}$$

where q_e and q_v are, respectively, the number of encryption and decryption queries, ℓ_{max} denotes the maximum number of m-bit blocks in an encryption or decryption query, $t' = t + cn\sigma$ for some constant c, and σ_e and σ are the total number of calls to the underlying compression function F in all queries asked by the CPA and CCA adversaries against the privacy and authenticity of the scheme, respectively.

The proof is obtained by combing Lemma 1 in Subsect. 5.1 with Lemmas 2 and 3 in Subsect. 5.2.

Algorithm $\textsc{Initialize}(K)$
$L_* \leftarrow F_K(0^n, 0^m)$
$L(0) \leftarrow 4 \cdot L_*$
for $i \leftarrow 1$ **to** $\lceil \log_2(\ell_{max}) \rceil$ **do**
  $L(i) = 2 \cdot L(i-1)$
return

Algorithm $\text{HASH}_K(A)$
$b \leftarrow n + m$
$A_1 || A_2 \cdots A_{\ell-1} || A_\ell \xleftarrow{b} A$
$\mathsf{Tag}_a \leftarrow 0^n$
$\Delta \leftarrow 0^n$
for $i \leftarrow 1$ **to** $\ell - 1$ **do**
  $\Delta \leftarrow \Delta \oplus L(\mathtt{ntz}(i))$
  $\mathsf{Left} \leftarrow A_i[b-1 \cdots m]$
  $\mathsf{Right} \leftarrow A_i[m-1 \cdots 0]$
  $\Phi \leftarrow F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
  $\mathsf{Tag}_a \leftarrow \mathsf{Tag}_a \oplus \Phi$
if $|A_\ell| = b$ **then**
  $\Delta \leftarrow \Delta \oplus L(\mathtt{ntz}(\ell))$
  $\mathsf{Left} \leftarrow A_\ell[b-1 \cdots m]$
  $\mathsf{Right} \leftarrow A_\ell[m-1 \cdots 0]$
  $\Phi \leftarrow F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
  $\mathsf{Tag}_a \leftarrow \mathsf{Tag}_a \oplus \Phi$
else
  $\Delta \leftarrow \Delta \oplus L_*$
  $A_{\text{pad}} \leftarrow A_\ell || 10^{b-|A_\ell|-1}$
  $\mathsf{Left} \leftarrow A_{\text{pad}}[b-1 \cdots m]$
  $\mathsf{Right} \leftarrow A_{\text{pad}}[m-1 \cdots 0]$
  $\Phi \leftarrow F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
  $\mathsf{Tag}_a \leftarrow \mathsf{Tag}_a \oplus \Phi$
return Tag_a

Algorithm $\mathcal{E}_K(N, A, M)$
if $|N| > n - 1$ **then**
  return $\perp$
$M_1 || M_2 \cdots M_{\ell-1} || M_\ell \xleftarrow{m} M$
$\Delta \leftarrow F_K(N || 10^{n-1-|N|}, 0^m)$
$H \leftarrow 0^n$
$\Delta \leftarrow \Delta \oplus L(0)$
$H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
for $i \leftarrow 1$ **to** $\ell - 1$ **do**
  $C_i \leftarrow H \oplus M_i$
  $\Delta \leftarrow \Delta \oplus L(\mathtt{ntz}(i+1))$
  $H \leftarrow F_K(H \oplus \Delta, M_i)$
$C_\ell \leftarrow H \oplus M_\ell$
if $|M_\ell| = m$ **then**
  $\Delta \leftarrow \Delta \oplus 2 \cdot L_*$
  $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
else if $|M_\ell| \neq 0$ **then**
  $\Delta \leftarrow \Delta \oplus 3 \cdot L_*$
  $M_{\text{pad}} \leftarrow M_\ell || 10^{m-|M_\ell|-1}$
  $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_{\text{pad}})$
else
  $\mathsf{Tag}_e \leftarrow H$
$\mathsf{Tag}_a \leftarrow \text{HASH}_K(A)$
$\mathsf{Tag} \leftarrow (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1 \cdots n-\tau]$
$\mathbb{C} \leftarrow C_1 || C_2 || \cdots || C_\ell || \mathsf{Tag}$
return $\mathbb{C}$

Algorithm $\mathcal{D}_K(N, A, \mathbb{C})$
if $|N| > n - 1$ **or** $|\mathbb{C}| < \tau$ **then**
  return $\perp$
$C_1 || C_2 \cdots C_{\ell-1} || C_\ell || \mathsf{Tag} \xleftarrow{m} \mathbb{C}$
$\Delta \leftarrow F_K(N || 10^{n-1-|N|}, 0^m)$
$H \leftarrow 0^n$
$\Delta \leftarrow \Delta \oplus L(0)$
$H \leftarrow F_K(H \oplus \Delta, \langle \tau \rangle_m)$
for $i \leftarrow 1$ **to** $\ell - 1$ **do**
  $M_i \leftarrow H \oplus C_i$
  $\Delta \leftarrow \Delta \oplus L(\mathtt{ntz}(i+1))$
  $H \leftarrow F_K(H \oplus \Delta, M_i)$
$M_\ell \leftarrow H \oplus C_\ell$
if $|C_\ell| = m$ **then**
  $\Delta \leftarrow \Delta \oplus 2 \cdot L_*$
  $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
else if $|C_\ell| \neq 0$ **then**
  $\Delta \leftarrow \Delta \oplus 3 \cdot L_*$
  $M_{\text{pad}} \leftarrow M_\ell || 10^{m-|M_\ell|-1}$
  $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_{\text{pad}})$
else
  $\mathsf{Tag}_e \leftarrow H$
$\mathsf{Tag}_a \leftarrow \text{HASH}_K(A)$
$\mathsf{Tag}' \leftarrow (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1 \cdots n-\tau]$
if $\mathsf{Tag}' = \mathsf{Tag}$ **then**
  return $M \leftarrow M_1 || M_2 || \cdots || M_\ell$
else
  return $\perp$

Fig. 2. Definition of OMD$[F, \tau]$. The function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ is a keyed compression function with $\mathcal{K} = \{0,1\}^k$ and $m \leq n$. The tag length is $\tau \in \{0, 1, \cdots, n\}$. Algorithms $\mathcal{E}$ and $\mathcal{D}$ can be called with arguments $K \in \mathcal{K}$, $N \in \{0,1\}^{\leq n-1}$, and $A, M, \mathbb{C} \in \{0,1\}^*$. ℓ_{max} is the bound on the maximum number of blocks in any input to the encryption or decryption algorithms.

Remark 1. Referring to Subsect. 3 for definitions of the resource parameters, it can be seen that: $\sigma_e = \lceil \sigma_M/m \rceil + \lceil \sigma_A/(n+m) \rceil + 2q_e$; $\sigma = \lceil (\sigma_M + \sigma_{\mathbb{C}'})/m \rceil + \lceil (\sigma_A + \sigma_{A'})/(n+m) \rceil + 2q$; *and* $\ell_{max} = \lceil L_{max}/m \rceil$.

5.1 Generalized OMD Using a Tweakable Random Function

Figure 3 shows the G-OMD$[\widetilde{R}, \tau]$ scheme which is a generalization of OMD$[F, \tau]$ using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$. The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks; namely, $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$, where $\mathcal{N} = \{0,1\}^{|N|}$ is the set of nonces and $\mathbb{N}$ is the set of positive integers.

Lemma 1. *Let G-OMD*$[\widetilde{R}, \tau]$ *be the scheme shown in Fig. 3. Then*

$$\mathbf{Adv}^{\mathrm{priv}}_{\text{G-OMD}[\widetilde{R},\tau]}(q_e, \sigma_e, \ell_{max}) = 0$$

$$\mathbf{Adv}^{\mathrm{auth}}_{\text{G-OMD}[\widetilde{R},\tau]}(q_e, q_v, \sigma, \ell_{max}) \leq \frac{q_v \ell_{max}}{2^n} + \frac{q_v}{2^\tau}$$

where q_e *and* q_v *are, respectively, the number of encryption and decryption queries,* ℓ_{max} *denotes the maximum number of* m*-bit blocks in an encryption or decryption query, and* σ_e *and* σ *are the total number of calls to the underlying tweakable random function* $\widetilde{R}$ *in all queries asked by the CPA and CCA adversaries against the privacy and authenticity of the scheme, respectively.*

The proof is provided in the full version of this paper [10].

5.2 Instantiating Tweakable RFs with PRFs

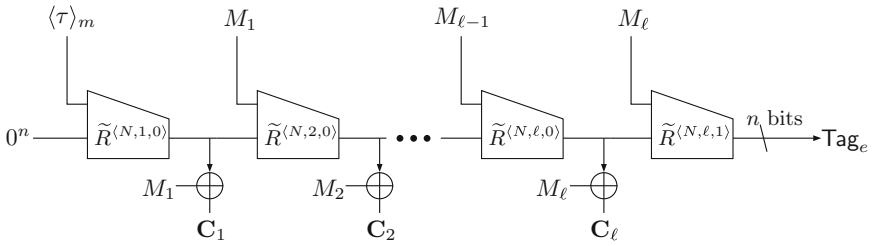
Step 1. Replace the tweakable RF $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ in G-OMD with a tweakable PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^k$. The following lemma states the classical bound on the security loss induced by this replacement step. The proof is a straightforward reduction and omitted here.

Lemma 2. *Let* $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ *be a tweakable RF and* $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ *be a tweakable PRF. Then*

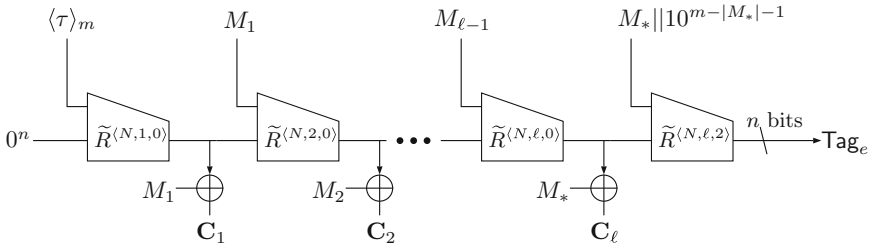
$$\mathbf{Adv}^{\mathrm{priv}}_{\text{G-OMD}[\widetilde{F},\tau]}(t, q_e, \sigma_e, \ell_{max}) \leq \mathbf{Adv}^{\mathrm{priv}}_{\text{G-OMD}[\widetilde{R},\tau]}(q_e, \sigma_e, \ell_{max}) + \mathbf{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t', \sigma_e)$$

$$\mathbf{Adv}^{\mathrm{auth}}_{\text{G-OMD}[\widetilde{F},\tau]}(t, q_e, q_v, \sigma, \ell_{max}) \leq \mathbf{Adv}^{\mathrm{auth}}_{\text{G-OMD}[\widetilde{R},\tau]}(q_e, q_v, \sigma, \ell_{max}) + \mathbf{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t'', \sigma)$$

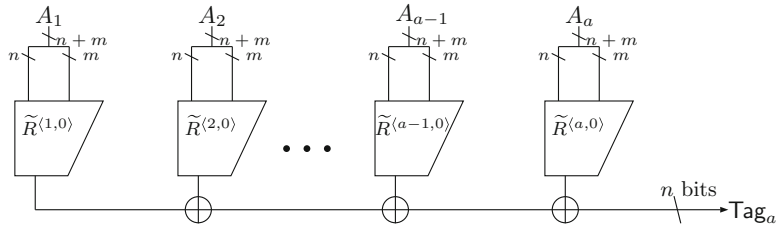
where q_e *and* q_v *are, respectively, the number of encryption and decryption queries,* $q = q_e + q_v$, ℓ_{max} *denotes the maximum number of* m*-bit blocks in an encryption or decryption query,* $t' = t + cn\sigma_e$ *and* $t'' = t + c'n\sigma$ *for some constants* c, c'*, and* σ_e *and* σ *are the total number of calls to the underlying compression function* F *in all queries asked by the CPA and CCA adversaries against the privacy and authenticity of the scheme, respectively.*



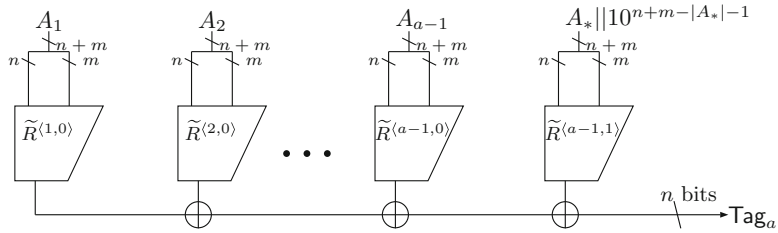
Encrypting a message whose length is a multiple of the block length. No padding is needed.



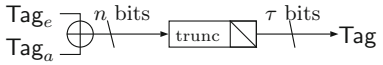
Encrypting a message whose length is not a multiple of the block length. The final message block is padded to make it a full block



Computing Tag_a for an associate data whose length is a multiple of the input length (i.e $|A_a| = n + m$).



Computing Tag_a for an associate data whose length is not a multiple of the input length. The final block is padded to make it a full block .



The Tag is computed as XOR of Tag_e and Tag_a truncated to τ bits.

Fig. 3. The G-OMD$[\widetilde{R}, \tau]$ scheme using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ (i.e. $\widetilde{R} \xleftarrow{\$} \mathrm{Func}^{\mathcal{T}}(n+m, n)$).

Step 2. We instantiate a tweakable PRF using a PRF by means of XORing (part of) the input by a mask generated as a function of the key and tweak as shown in Fig. 4. This method to tweak a PRF is (essentially) the XE method of [23]. In OMD the tweaks are of the form $T = (\alpha, i, j)$ where $\alpha \in \mathcal{N} \cup \{\varepsilon\}$, $1 \leq i \leq 2^{n-8}$ and $j \in \{0, 1, 2\}$. We note that not all combinations are used; for example, if $\alpha = \varepsilon$ (empty) which corresponds to processing of the associate data in Fig. 1 then $j \neq 2$. The masking function $\Delta_K(T) = \Delta_K(\alpha, i, j)$ outputs an n-bit mask such that the following two properties hold for any fixed string $H \in \{0,1\}^n$:

1. $\Pr[\Delta_K(\alpha, i, j) = H] \leq 2^{-n}$ for any (α, i, j)
2. $\Pr[\Delta_K(\alpha, i, j) \oplus \Delta_K(\alpha', i', j') = H] \leq 2^{-n}$ for $(\alpha, i, j) \neq (\alpha', i', j')$

where the probabilities are taken over random selection of the key.

It is easy to verify that these two properties are satisfied by the specific masking scheme of OMD as described in Sect. 4.

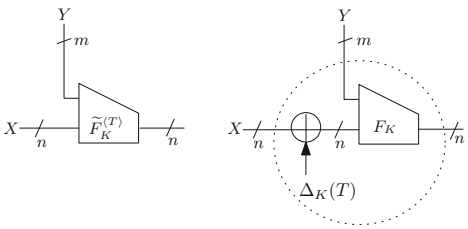


Fig. 4. Building a tweakable PRF $\widetilde{F}_K^{\langle T\rangle} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ using a PRF $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. There are several efficient ways to define the masking function $\Delta_K(T)$ [9,18,23]; we use the method of [18].

Lemma 3. *Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a function family with key space $\mathcal{K}$. Let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be defined by $\widetilde{F}_K^{\langle T\rangle}(X, Y) = F_K((X \oplus \Delta_K(T)), Y)$ for every $T \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$ and $\Delta_K(T)$ is the masking function of OMD as defined in Sect. 4. If F is PRF then $\widetilde{F}$ is tweakable PRF; more precisely*

$$\mathbf{Adv}^{\widetilde{\text{prf}}}_{\widetilde{F}}(t, q) \leq \mathbf{Adv}^{\text{prf}}_{F}(t', 2q) + \frac{3q^2}{2^n}\,.$$

The proof is a simple adaptation of a similar result on the security of the XE construction (to tweak a blockcipher) in [18]. As we use a PRF rather than PRP, our bound has two main terms. The first term is a single birthday bound loss of $\frac{0.5q^2}{2^n}$ to take care of the case that a collision might happen when computing the initial mask $\Delta_{N,0,0} = F_K(N||10^{n-1-|N|}, 0^m)$ using a PRF (F) rather than a PRP (as in [18]). The analysis of the remaining term (i.e. $\frac{2.5q^2}{2^n}$) is essentially

the same as the similar part in [18], but we note that in the context of our construction as we are directly dealing with PRFs unlike [18] in which PRPs are used, the bound obtained here does not have any loss terms caused by the switching (PRF–PRP) lemma. Therefore, instead of the $\frac{6q^2}{2^n}$ bound in [18] (from which $\frac{3.5q^2}{2^n}$ is due to using the switching lemma) our bound has only $\frac{2.5q^2}{2^n}$.

6 Instantiations

6.1 OMD-SHA256

Our primary recommendation to instantiate OMD, called OMD-SHA256, uses the underlying compression function of SHA-256 [1]. This is intended to be the appropriate choice for next-generation processors supporting Intel SHA Extensions.

The compression function of SHA-256 is a map SHA-256 : $\{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. On input a 256-bit chaining block X and a 512-bit message block Y, it outputs a 256-bit digest Z, i.e. let $Z = \text{SHA-256}(X, Y)$.

To use OMD with SHA-256, we use the first 256-bit argument X for chaining values as usual. We use the 512-bit argument Y (the message block in SHA-256) to input both a 256-bit message block and the key K which can be of any length $k \leq 256$ bits. If $k < 256$ then let the key be $K||0^{256-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{256} \times \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ needed in OMD as $F_K(H, M) = \text{SHA-256}(H, K||0^{256-k}||M)$.

The parameters of OMD-SHA256 are as follows:

- The message block length in bits is $m = 256$; i.e. $|M_i| = 256$. *If needed*, we pad the final block of the message with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 256 bits.
- The key length in bits can be $80 \leq k \leq 256$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{256-k} to make its length exactly 256 bits.
- The nonce (public message number) length in bits can be $96 \leq |N| \leq 255$. We *always* pad the nonce with $10^{255-|N|}$ to make its length exactly 256 bits.
- The secret message number length in bits is 0; that is, our scheme does not support secret message numbers.
- The associated data block length in bits is $2n = 512$; i.e. $|A_i| = 512$. *If needed*, we pad the final block of the associated data with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 512 bits.
- The tag length in bits can be $32 \leq \tau \leq 256$; but it must be noted that the selection of the tag length directly affects the security level.

6.2 OMD-SHA512

Our second recommendation to instantiate OMD, called OMD-SHA512, uses the underlying compression function of SHA-512 [1]. This is intended to be the appropriate choice for implementations on 64-bit machines.

The compression function of SHA-512 is a map SHA-512 : $\{0,1\}^{512} \times \{0,1\}^{1024} \rightarrow \{0,1\}^{512}$. On input a 512-bit chaining block X and a 1024-bit message block Y, it outputs a 512-bit digest Z, i.e. let $Z = \text{SHA-512}(X, Y)$.

To use OMD with SHA-512, we use the first 512-bit argument X for chaining values as usual. In our notation (see Fig. 1) this means that $n = 512$. We use the 1024-bit argument Y (the message block in SHA-512) to input both a 512-bit message block and the key K which can be of any length $k \leq 512$ bits. If $k < 512$ then let the key be $K||0^{512-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{512}$ needed in OMD as $F_K(H, M) = \text{SHA-512}(H, K||0^{512-k}||M)$.

The parameters of OMD-SHA512 are set as follows:

- The message block length in bits is $m = 512$; i.e. $|M_i| = 512$. *If needed*, we pad the final block of the message with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 512 bits.
- The key length in bits can be $80 \leq k \leq 512$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{512-k} to make its length exactly 512 bits.
- The nonce (public message number) length in bits can be $96 \leq |N| \leq 511$. We *always* pad the nonce with $10^{511-|N|}$ to make its length exactly 512 bits.
- The secret message number length in bits is 0; that is, our scheme does not support secret message numbers.
- The associated data block length in bits is $2n = 1024$; i.e. $|A_i| = 1024$. *If needed*, we pad the final block of the associated data with 10^* (i.e., a single 1 followed by the minimal number of 0's needed) to make its length exactly 1024 bits.
- The tag length in bits can be $32 \leq \tau \leq 512$.

6.3 Instantiating G-OMD with a Native Tweakable PRF

Considering the proof steps of Theorem 1, it can be seen that the security bound for the core of OMD, called G-OMD (Fig. 3), is free from the quadratic degradation (birthday term) in the total number of blocks in queries; namely, $\frac{3\sigma^2}{2^n}$. This term is introduced only in Lemma 3, when we instantiated a tweakable PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ from a PRF $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ using the XE masking method; i.e., $\widetilde{F}_K^{\langle T \rangle}(X, Y) = F_K(X \oplus \Delta_K(T), Y)$. This instantiation aimed to provide a large range for the allowed key length and the nonce length in instantiations of OMD using practical compression functions, regarding that most practical compression functions (e.g. those of the SHA family) do not have a dedicated key input and must be keyed by allocating some part of their input for the key.

To avoid such a degradation in the security bound, caused by the XE method, one can directly instantiate the tweakable PRF $\widetilde{F}$ from a PRF $F' : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^{m'}) \rightarrow \{0,1\}^n$, where $m' = m + |T|$, by defining $\widetilde{F}_K^{\langle T \rangle}(X, Y) = F'_K(X, T||Y)$. In practice, using an off-the-shelf (keyless) compression function with fixed input and output sizes, this will imply supporting a more restricted range of parameters' sizes (key length and nonce length) compared to OMD. Fortunately, the

allowed parameter sizes for would be still large enough for most of applications today, if one simply use the compression functions of SHA-256 or SHA-512. Let's call the G-OMD instantiated with these compression functions, G-OMD-SHA256 and G-OMD-SHA512, respectively.

For G-OMD-SHA256 and G-OMD-SHA512, we can define the tweakable function $\widetilde{F}_K^{\langle T \rangle}(X, Y)$ by $\widetilde{F}_K^{\langle T \rangle}(X, Y) = \text{SHA-256}(X, K||T||0^{256-k-|T|}||M)$ and $\widetilde{F}_K^{\langle T \rangle}(X, Y) = \text{SHA-512}(X, K||T||0^{512-k-|T|}||M)$, respectively. Considering that $T = (N, i, j)$, where N is the nonce, i is the block number, and $j \in \{0, 1, 2\}$, G-OMD-SHA256 and G-OMD-SHA512 will allow k-bit keys, $|N|$-bit nonce, ℓ-block messages and a-block associated data, subject to the constraints $k + |N| + max(\log \ell, \log a) + 2 \leq 256$ and $k + |N| + max(\log \ell, \log a) + 2 \leq 512$, respectively.

Acknowledgments. We would like to thank the anonymous reviewers of SAC 2014 for their constructive comments. The EPFL team was partially supported by Microsoft Research under MRL Contract No. 2014-006 (DP1061305).

References

1. Secure Hash Standard (SHS). NIST FIPS PUB 180–4, Mar 2012
2. Bellare, M.: New proofs for NMAC and HMAC: security without collision-resistance. IACR Cryptology ePrint Archive 2006, 43 (2006)
3. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: FOCS, pp. 394–403 (1997)
4. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
5. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. J. Cryptol. **21**(4), 469–491 (2008)
6. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Heidelberg (2000)
7. Bernstein, D.J.: Cryptographic competitions: CAESAR. http://competitions.cr.yp.to
8. Canvel, B., Hiltgen, A.P., Vaudenay, S., Vuagnoux, M.: Password interception in a SSL/TLS channel. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 583–599. Springer, Heidelberg (2003)
9. Chakraborty, D., Sarkar, P.: A general construction of tweakable block ciphers and different modes of operations. IEEE Trans. Inf. Theory **54**(5), 1991–2006 (2008)
10. Cogliani, S., Maimut, D., Naccache1, D., do Canto, R.P., Reyhanitabar, R., Vaudenay, S., Vizár, D.: Offset Merkle-Damgård (OMD) version 1.0: A CAESAR Proposal, Mar 2014. http://competitions.cr.yp.to/round1/omdv10.pdf
11. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof on-line authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)

12. Guilford, J., Cote, D., Gopal, V.: Fast SHA512 Implementations on Intel® Architecture Processors, Nov 2012. http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/fast-sha512-implementations-ia-processors-paper.html
13. Guilford, J., Yap, K., Gopal, V.: Fast SHA-256 Implementations on Intel® Architecture Processors, May 2012. http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/sha-256-implementations-paper.html
14. Gulley, S., Gopal, V., Yap, K., Feghali, W., Guilford, J., Wolrich, G.: Intel® SHA Extensions: New Instructions Supporting the Secure Hash Algorithm on Intel® Architecture Processors, Jul 2013. https://software.intel.com/sites/default/files/article/402097/intel-sha-extensions-white-paper.pdf
15. Iwata, T.: New blockcipher modes of operation with beyond the birthday bound security. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 310–327. Springer, Heidelberg (2006)
16. Iwata, T.: Authenticated encryption mode for beyond the birthday bound security. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 125–142. Springer, Heidelberg (2008)
17. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (2001)
18. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
19. Landecker, W., Shrimpton, T., Terashima, R.S.: Tweakable blockciphers with beyond birthday-bound security. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 14–30. Springer, Heidelberg (2012)
20. Lefranc, D., Painchault, P., Rouat, V., Mayer, E.: A generic method to design modes of operation beyond the birthday bound. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 328–343. Springer, Heidelberg (2007)
21. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 257–274. Springer, Heidelberg (2014)
22. Rogaway, P.: Authenticated-encryption with associated-data. In: ACM Conference on Computer and Communications Security, pp. 98–107 (2002)
23. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
24. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg (2004)
25. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: ACM Conference on Computer and Communications Security, pp. 196–205 (2001)
26. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
27. Vaudenay, S.: Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 534–546. Springer, Heidelberg (2002)