A Simple and Generic Construction of Authenticated Encryption With Associated Data

Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
email: palash@isical.ac.in

Abstract. We revisit the problem of constructing a protocol for performing authenticated encryption with associated data (AEAD). A technique is described which combines a collision resistant hash function with a protocol for authenticated encryption (AE). The technique is both simple and generic and does not require any additional key material beyond that of the AE protocol. Concrete instantiations are shown where a 256-bit hash function is combined with some known single-pass AE protocols employing either 128-bit or 256-bit block ciphers. This results in possible efficiency improvement in the processing of the header.
Keywords: authenticated encryption with associated data, collision resistant hash function, generic construction.

1 Introduction

An authenticated encryption (AE) protocol combines privacy and authentication. The classical method of achieving this with a block cipher is to use two passes over the message – the first pass encrypts the data while the second pass generates the authentication tag. The cost per message block is approximately two block cipher calls.

The formal definition of AE as an integrated cryptographic primitive was independently introduced in [7] and [1]. Around the same time, there was a search for a *single* pass AE protocol, i.e., an algorithm which makes a single pass over the data and simultaneously achieves both privacy and authentication. The first construction of single-pass AE protocols was due to Jutla [6]. Independent work proposed by Gligor and Donescu [5] also described single-pass AE protocols. Jutla's work was refined and polished by Rogaway [13] to obtain OCB. Later, based on the work of Liskov, Rivest and Wagner [8], Rogaway [11] introduced an efficient family of tweakable block ciphers (TBC) and several modes of operations based on this family. In particular, the OCB protocol of [13] was refined to obtain OCB1 protocol in [11]. The OCB1 protocol is currently called the OCB protocol [9] and a patent on this algorithm is held by Rogaway [9]. Rogaway's approach to modes of operations via tweakable block ciphers was extended and generalized by Chakraborty and Sarkar in [3]. In particular, it was shown that OCB (as given in [11]) is a special case of a family of single-pass AE protocols all of which have similar efficiencies.

From the point of applications, AE is not really sufficient, as has been argued in [10]. Very often, along with the message, there is an associated data (or header) which is required to be authenticated but not encrypted. An example is IP packets. Apart from the actual message, such a packet has a header which, among other things, provide information regarding the source and destination of the packet. The header cannot be encrypted as otherwise it will

not be possible for a router to read it. On the other hand, the header certainly needs to be authenticated to prevent fraudulent packets. The solution is to include authentication information for the header, but not actually encrypt it.

As mentioned in [10], the usual practitioners solution is to combine an encryption scheme along with a message authentication code (MAC) algorithm – encrypt the message M, prepend the header H and then obtain a MAC for the entire string. This approach necessarily requires two passes over the data.

Rogaway [10] was the first to treat the problem of associated data as a cryptographic primitive in its own right. He introduced the notion of authenticated encryption with associated data (AEAD). In [10], it was eloquently argued that naive solutions to combine a single pass AE protocol to achieve AEAD is inherently problematic. The task of designing an efficient AEAD protocol has subtleties and should be tackled as an independent problem.

Two methods for generically achieving AEAD is described in [10]. The first method builds on a suggestion called nonce stealing (attributed to Cam-Winget and Walker in [10]) and has limited applicability since the associated data in this case can only be a few bytes. The second method, called ciphertext translation, works with arbitrary length associated data. The basic idea is to use an AE protocol with a key K to generate an intermediate CT; use a pseudo-random function (PRF) with another key K' to process the associated data H to obtain Δ; and the final ciphertext is obtained by XORing Δ into the last $|\Delta|$ bits of CT.

This generic approach of ciphertext translation requires the use of two independent keys K and K'. However, for the special case of OCB (as given in [13]) and the parallel message authentication algorithm PMAC [2], it was shown in [10], that a single key is sufficient. The proof of this result is quite long and complex. The later work [11], shows how the notion of tweakable block cipher can be used to simplify the proof of the ciphertext translation construction as applied to the versions of OCB and PMAC given in [11]. This construction also holds for the generalization given in [3].

Our Contributions. In this work, we revisit the problem of constructing an AEAD protocol. We describe a simple and generic construction of an AEAD protocol from an AE protocol and a collision resistant hash function. This is done in two simple steps.

The first step extends an AE protocol which can handle fixed length nonces to AE protocol which can handle variable length nonces. We denote by AE^+, a protocol which can handle variable length nonces. The idea of this extension is simple. Given a nonce N for AE^+, we use a collision resistant hash function h to obtain a nonce $L = h(N)$ for the AE protocol. Then invoke the encryption or decryption algorithm of the AE protocol with L and the message or ciphertext as required. The idea of the proof is also simple. Assuming $h()$ to be collision resistant ensures that if the Ns are distinct, then so are the Ls. Consequently, AE^+ is secure if AE is secure.

The second step is even more simple and constructs an AEAD protocol from AE^+. Basically, given nonce N and header H for the AEAD protocol, concatenate the two to obtain $H||N$ which is used as the nonce for AE^+. The freshness of N ensures the freshness of $H||N$. Consequently, if AE^+ is secure, then so is the constructed AEAD.

For the first step to work, we need the size of the digest of $h()$ to be equal to the size of the nonces used in the AE protocol. Our proposal for concrete instantiation of $h()$ is to use a hash function with a 256-bit digest such as SHA-256 [14]. This works fine when we

combine with an AE protocol which handles 256-bit nonces. Examples are protocols given in [3] which use 256-bit block ciphers.

There is a problem, however, when we wish to use AES-128. OCB [11] and the AE protocols in [3] have the limitation that the nonce size is equal to the block length of the underlying block cipher. So, if we use AES-128, then the size of nonces in these protocols will be 128 bits. Consequently, we cannot directly combine these AE protocols with a 256-bit hash function to obtain an AEAD protocol.

The way around this problem is to define a variant of the protocols in [3] (which includes OCB) so that they handle $2n$-bit nonces even when used with n-bit block ciphers. We show that starting from a tweakable block cipher, such a variant is rather easy to define. As a result, we obtain AE protocols which use AES-128 and can handle 256-bit nonces. Such AE protocols can be combined with a 256-bit hash function using our technique to obtain an AEAD protocol.

As mentioned earlier, the technique of combining a hash function and an AE protocol to obtain an AEAD protocol is both simple and generic. There are several further advantages.

1. A collision resistant hash function does not require a secret key. Hence, the key for the AEAD protocol consists only of the key for the AE protocol. In contrast, [10] required a complicated proof to ensure that a single key can be used to combine OCB and PMAC to obtain an AEAD protocol.
2. In our proposal, the header is hashed, while in the technique of ciphertext translation used in [10, 11], the header is processed using a PRF which is in turn built using a block cipher. Consequently, whenever the hashing speed is faster than the block cipher speed, the processing of the header in our construction will be faster than the processing in ciphertext translation.

To summarize, we believe that our work provides a simple, generic and efficient solution to an important cryptographic problem.

2 Preliminaries

The definition of AE and AEAD protocols and their security definitions are based on [10, 11].

A block cipher is a map $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, where $\mathcal{K}$ is a finite non-empty set called the key space and for all $K \in \mathcal{K}$, $E(K,\cdot) = E_K(\cdot)$ is a permutation of $\{0,1\}^n$. $\mathrm{Perm}(n)$ denotes the set of all permutations of $\{0,1\}^n$. The notation $\pi \xleftarrow{\$} \mathrm{Perm}(n)$ denotes the choice of a random permutation on n bits.

An adversary is a probabilistic algorithm with possible access to encryption and/or decryption oracles. The notation $A^{O_1,O_2} \Rightarrow 1$ denotes the event that an adversary A outputs 1 after interacting with the oracles O_1 and O_2. We will assume that an adversary does not ask a query for which it can easily obtain the answer. Thus, it never repeats a query; does not ask for the decryption of a ciphertext which it has previously received as an output of an encryption query; and neither does it ask for the encryption of a plaintext which it has previously received as output of a decryption query. The notation $\mathbf{Adv}(A)$ denotes the advantage of an adversary A. The definitions of various advantages are as follows.

Definition 1. *Let $E_K(\cdot)$ be a block cipher and let A be an adversary. We define the following advantages.*

$$\mathbf{Adv}^{\mathrm{prp}}_{E}(A) = \mathsf{Prob}[K \xleftarrow{\$} \mathcal{K} : A^{E_K(\cdot)} \Rightarrow 1] - \mathsf{Prob}[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\pi(\cdot)} \Rightarrow 1].$$
$$\mathbf{Adv}^{\pm\mathrm{prp}}_{E}(A) = \mathsf{Prob}[K \xleftarrow{\$} \mathcal{K} : A^{E_K(\cdot),D_K(\cdot)} \Rightarrow 1] - \mathsf{Prob}[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1].$$

Here D denotes the inverse of E and the probability is over random choices of K as well as random bits of the adversary A.

The extension of these advantages to resource bounded advantages are done in the usual manner: $\mathbf{Adv}^{\mathrm{xxx}}_{\Pi}(\mathcal{R}) = \sup_A\{\mathbf{Adv}^{\mathrm{xxx}}_{\Pi}(A)\}$ over all adversaries A that use resources at most $\mathcal{R}$. The resources of interest are the number of queries q made by the adversary, the total number σ_n of n-bit blocks provided by the adversary in all its queries and the running time t.

2.1 Authenticated Encryption

An AE protocol consists of two deterministic algorithms – an encryption and a decryption algorithm. The encryption algorithm $\mathsf{AE.Enc}$ takes as input a triple (K, N, M) where $K \in \mathcal{K}$ is a secret key, $N \in \mathcal{N}$ is a nonce and $M \in \mathcal{M}$ is a message to be encrypted. It returns a pair (C, tag) as output, where $C \in \mathcal{C}$ and tag is a string of fixed length τ. We will have $\mathcal{C}$ to be equal to the message space $\mathcal{M}$. Further, efficient AE algorithms have $|M| = |C|$.

The nonce is not considered to be part of the ciphertext, though it is required for decryption. The only requirement on the nonce is that it should not be repeated. The decryption algorithm $\mathsf{AE.Dec}$ takes as input a tuple (K, N, C, tag) and either returns bad or returns M, where M is such that $(C, \mathsf{tag}) = \mathsf{AE.Enc}(K, N, M)$. We will write $\mathsf{AE.Enc}_K(N, M)$ instead of $\mathsf{AE.Enc}(K, N, M)$ and $\mathsf{AE.Dec}_K(N, C, \mathsf{tag})$ instead of $\mathsf{AE.Dec}(K, N, C, \mathsf{tag})$.

The security of an authenticated encryption protocol consists of two parts – privacy and authenticity. The adversary is given access to the encryption oracle and is assumed to be nonce respecting, i.e., it does not repeat a nonce in its queries to the oracle. We assume that the output of the encryption oracle $\mathcal{E}_K(\cdot,\cdot)$ consists of the concatenation of the ciphertext C and tag.

Following Rogaway [11], the privacy of a encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ against a nonce respecting adversary A is defined in the sense of "indistinguishability from random strings" in the following manner:

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

where $\$(\cdot,\cdot)$ is an oracle that takes (N, M) as input and returns $|M| + \tau$ random bits as output.

For defining authenticity, the adversary is said to successfully *forge* if it outputs a pair (N, C, tag) which is valid and (C, tag) was not the result of any prior (N, M) query. Formally,

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

Fixed and variable length nonces. Based on the length of the nonces, we distinguish between two kinds of AE protocols.

1. AE with fixed length nonces, i.e., $\mathcal{N} = \{0,1\}^m$ where m is a fixed integer.
2. AE with variable length nonces, i.e., the length of the nonces can vary.

In both cases, the restriction that nonces cannot be repeated still has to hold. We will use AE to denote the first kind of protocol, while we will use AE$^+$ to denote the second kind of protocol.

2.2 Authenticated Encryption With Associated Data

An AEAD protocol consists of two deterministic algorithms. The encryption algorithm is $\mathsf{AEAD.Enc}_K(N, H, M)$, where K is the secret key, N is a nonce, H is a header and M is a message to be encrypted. The algorithm returns (C, tag). The decryption algorithm $\mathsf{AEAD.Dec}_K(N, H, C, \mathsf{tag})$ returns either bad or the proper message M. The goal of an AEAD protocol is to encrypt the message M but to authenticate both the message and the header. In particular, the header is authenticated but not encrypted. Thus, the header can be publicly read but any alteration to it will be detected during decryption.

The security notion of an AEAD protocol is an extension of the security notion of an AE protocol. As before this consists of privacy and authenticity. The adversary is given access to the encryption oracle and is assumed to be nonce respecting, i.e., it does not repeat a nonce in its queries to the oracle. Following Rogaway [11], the privacy of a encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ against a nonce respecting adversary A is defined in the sense of "indistinguishability from random strings" in the following manner:

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

where $\$(\cdot,\cdot,\cdot)$ is an oracle that takes (N, H, M) as input and returns $|M| + \tau$ random bits as output. For defining authenticity, the adversary is said to successfully *forge* if it outputs a pair (N, H, C, tag) which is valid and (C, tag) was not the result of any prior (N, H, M) query. Formally,

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

2.3 Parameters of an Adversary

An adversary for either an AE or an AEAD protocol makes a few oracle queries, runs for a certain amount of time and has a definite advantage. By an (ϵ, q, t)-adversary A attacking the privacy of an AE protocol, we mean that A makes at most q oracle queries, runs for time t and that $\mathbf{Adv}^{\mathrm{priv}}_{\Pi}(A) \leq \epsilon$. Similar notation will be used for an adversary attacking the authenticity of an AE protocol as also for adversaries attacking the privacy and authenticity of an AEAD protocol.

Quantification over adversaries is done in the usual manner to obtain resource bounded definitions of the different advantages. For example, if Π is an AE protocol, then $\mathbf{Adv}^{\mathrm{priv}}_{\Pi}(q, t)$ denotes the supremum of advantages over all adversaries A attacking the privacy of the AE protocol and running in time at most t and making at most q oracle queries.

2.4 Collision Resistant Hash Function

A collision resistant hash function $h : \{0,1\}^* \to \{0,1\}^m$ is a deterministic algorithm that takes as input an arbitrary length binary string and returns as output a binary string of length m.

Intuitively, the security requirement is that it should be infeasible for an adversary to find two strings $x \neq x'$ such that $h(x) = h(x')$. Following Stinson [15] and Rogaway [12], we formalize an adversary A's advantage in finding collision in the following manner.

$$\mathbf{Adv}_h(A) = \mathsf{Prob}[(x, x') \Leftarrow A : x \neq x' \text{ and } h(x) = h(x')].$$

The adversary is allowed to be probabilistic and the probability in the above advantage is computed over the random choices made by the adversary. By an (ϵ, t)-adversary A attacking the collision resistance of a hash function h we will mean that the runtime of A is at most t and that $\mathbf{Adv}_h(A) \leq \epsilon$.

3 Constructions

3.1 From AE to AE$^+$

We construct an AE$^+$ protocol from an AE protocol and a collision resistant hash function $h : \{0,1\}^* \to \{0,1\}^m$, where m is the size of nonces in the AE protocol. Figure 1 shows the encryption and decryption algorithms.

Fig. 1. Encryption and Decryption using AE$^+$.

$\mathsf{AE}^+.\mathsf{Enc}_K(N, M)$	$\mathsf{AE}^+.\mathsf{Dec}_K(N, C, \mathsf{tag})$
1. $L = h(N)$;	1. $L = h(N)$;
2. return $\mathsf{AE}.\mathsf{Enc}_K(L, M)$.	2. return $\mathsf{AE}.\mathsf{Dec}_K(L, C, \mathsf{tag})$.

The security reduction for the above protocol is easy. The basic idea is to build an adversary B for AE from an adversary A for AE^+. Intuitively, adversary A can choose to attack AE^+ in two ways – either by finding a collision for $h()$ or by attacking AE. Since we assume that it is difficult to find a collision for $h()$, any successful attack on AE^+ with high probability leads to an attack on AE. Below we formalize this argument separately for privacy and authentication.

Our formalization is based on Rogaway's approach in [12] (the unkeyed, concrete C2-form). The crux of this formalization is to parameterize the theorem statement with explicit mention of adversaries. Usual theorem statements of security results are stated in terms of resource bounded notions of advantages, i.e., relations are obtained between the supremum (over all adversaries with specific resource bounds) of the advantages of the different components. Such an approach creates formalistic problems when talking about unkeyed hash functions. The simplifying idea, then, is to explicitly relate advantages of adversaries for different components. See [12] for further justification and details of this approach.

Theorem 1. *The privacy and the authenticity of the AE$^+$ shown in Figure 1 are stated as follows.*

Privacy. *Let A be an (ϵ, q, t)-adversary attacking the privacy of the AE^+ protocol. Then it is possible to construct an (ϵ_1, q, t_1)-adversary B attacking the privacy of the AE protocol and an (ϵ_2, t_1)-adversary C attacking the collision resistance of h, such that*

$$\epsilon = \frac{\epsilon_1}{1-\epsilon_2} \approx (1+\epsilon_2)\epsilon_1.$$

Authenticity. *Let A be an (ϵ, q, t)-adversary attacking the authenticity of the AE^+ protocol. Then it is possible to construct an (ϵ_1, q, t_1)-adversary B attacking the authenticity of the AE protocol and an (ϵ_2, t_1)-adversary C attacking the collision resistance of h, such that*

$$\epsilon = \frac{1}{1-\epsilon_2}\left(\epsilon_1 - \frac{\epsilon_2}{2^\tau}\right) \approx (1+\epsilon_2)\left(\epsilon_1 - \frac{\epsilon}{2^\tau}\right).$$

In the above $t_1 = t + t'$, where t' is the sum of the following times.

- *Time for evaluating h on the variable length nonces of the q queries made by A.*
- *Time for oracle calls to the AE protocol.*
- *Time for insertion and search in a list (maintained as a height balanced binary tree) to determine a possible collision.*
- *The overhead time for bookkeeping. This time is proportional to the total length (in bits) of all the queries.*

Proof : The two proofs are described separately.

Privacy: A and B's advantages are the probabilities of distinguishing the outputs of the corresponding protocol from random strings. Adversary A has access to an oracle $\mathcal{O}_1$. This oracle is either the real oracle, i.e., the output of the AE^+ protocol or an oracle which simply returns random strings. A makes queries to the oracle in an adaptive manner and finally outputs a bit. We have

$$\mathbf{Adv}^{\text{priv}}_{\mathsf{AE}^+}(A) = \mathsf{Prob}[A^{\mathcal{O}_1} \Rightarrow 1 | \mathcal{O}_1 \text{ is real }] - \mathsf{Prob}[A^{\mathcal{O}_1} \Rightarrow 1 | \mathcal{O}_1 \text{ is random }]$$

The construction of B from A is described as follows. B has access to its own oracle $\mathcal{O}_2$ which is either real (i.e., returns the output of AE protocol) or random (i.e., returns random strings). B operates by simulating A's queries. The queries submitted by A are of the form (N_i, M_i) for $i = 1, \ldots, q$. If A ever submits a query (N_i, M_i) such that there is a $j < i$, with $N_i \neq N_j$ and $h(N_i) = h(N_j)$, then B outputs a random bit and aborts. This means that A has found a collision for h and hence B cannot use A to attack the AE protocol. In this case, the best strategy for B is to output a random bit and abort. Denote by Coll the event that there is a collision for $h()$. If the i-th query does not give rise to a collision, then B computes $L_i = h(N_i)$ and submits (L_i, M_i) to its oracle $\mathcal{O}_2$. Whatever it gets, it returns to A. Finally, B outputs whatever A outputs. Note that $\mathcal{O}_1$ is real if and only if $\mathcal{O}_2$ is real.

The above also describes the construction of C from A. C is almost the same as B, except that if a collision is found, then C reports it; and if at the end, no collision has been found, then C reports failure. Thus, the advantage $\mathbf{Adv}(C)$ of C is equal to $\mathsf{Prob}[\mathsf{Coll}]$.

Assuming that the oracles are real we compute.

$$\begin{aligned}\mathsf{Prob}[B^{\mathcal{O}_2} \Rightarrow 1] &= \mathsf{Prob}[(B^{\mathcal{O}_2} \Rightarrow 1) \wedge (\mathsf{Coll} \vee \overline{\mathsf{Coll}})] \\ &= \mathsf{Prob}[(B^{\mathcal{O}_2} \Rightarrow 1) \wedge (\mathsf{Coll})] + \mathsf{Prob}[(B^{\mathcal{O}_2} \Rightarrow 1) \wedge (\overline{\mathsf{Coll}})] \\ &= \mathsf{Prob}[(B^{\mathcal{O}_2} \Rightarrow 1)|\mathsf{Coll}]\mathsf{Prob}[\mathsf{Coll}] + \mathsf{Prob}[(B^{\mathcal{O}_2} \Rightarrow 1)|\overline{\mathsf{Coll}}]\mathsf{Prob}[\overline{\mathsf{Coll}}] \\ &= \frac{1}{2}\mathsf{Prob}[\mathsf{Coll}] + \mathsf{Prob}[A^{\mathcal{O}_1} \Rightarrow 1]\mathsf{Prob}[\overline{\mathsf{Coll}}].\end{aligned}$$

Exactly the same computation will hold assuming both the oracles to be random. $\mathbf{Adv}(B)$ is the difference $\mathsf{Prob}[B^{\mathcal{O}_2} \Rightarrow 1|\mathcal{O}_2 \text{ is real}] - \mathsf{Prob}[B^{\mathcal{O}_2} \Rightarrow 1|\mathcal{O}_2 \text{ is random}]$ and similarly $\mathbf{Adv}(A)$ is the difference $\mathsf{Prob}[A^{\mathcal{O}_1} \Rightarrow 1|\mathcal{O}_1 \text{ is real}] - \mathsf{Prob}[A^{\mathcal{O}_1} \Rightarrow 1|\mathcal{O}_1 \text{ is random}]$. Thus, we have

$$\mathbf{Adv}(B) = (1 - \mathsf{Prob}[\mathsf{Coll}])\mathbf{Adv}(A) = (1 - \mathbf{Adv}(C))\mathbf{Adv}(A).$$

Rearranging we obtain the desired relation between the different advantages. The runtime of B (and C) can be easily verified.

Authenticity: The idea of the proof is similar to the above. The technical difference is in the behaviour of the adversaries A and B. In this case, the adversaries are given an encryption oracle instantiated by a secret key. They can submit queries to the oracle and receive answers. The queries of A are of the form (N_i, M_i), for $i = 1, \ldots, q$ and the answers it receives are of the form (C_i, tag_i). The queries of B and the answers it receives are similar; the only difference being that in case of B nonces are strings of length m, whereas in case of A nonces are strings of variable lengths.

The simulation of the queries of A by B is similar to the case of privacy. Suppose that the encryption queries of A are $(N_i, M_i, \mathsf{tag}_i)$, $1 \leq i \leq q$, and the forgery attempt is (N, C, tag). In this case, we say that the event Coll happens if either for distinct i, j, $h(N_i) = h(N_j)$; or, there is some i such that $h(N_i) = h(N)$. Note that here Coll also covers the forgery attempt. In case of Coll, B chooses a "new" (L, C) pair and a random tag and outputs (L, C, tag) as its own forgery. If Coll does not happen, and A outputs (N, C, tag), then B outputs $(h(N), C, \mathsf{tag})$. The collision finding adversary C is easily seen to be similar to B; reporting a collision if one is found; or reporting failure at the end. Then, as in the case of privacy, the advantage $\mathbf{Adv}(C)$ of C is equal to $\mathsf{Prob}[\mathsf{Coll}]$.

Let Succ_A (resp. Succ_B) be the event that A (resp. B) is successful in its forgery attempt. Then, $\mathbf{Adv}(A) = \mathsf{Prob}[\mathsf{Succ}_A]$ and $\mathbf{Adv}(B) = \mathsf{Prob}[\mathsf{Succ}_B]$. We have $\mathsf{Prob}[\mathsf{Succ}_B|\mathsf{Coll}] = 1/2^\tau$. This is because, in the event of Coll, the tag is chosen randomly and the probability that it equals the correct tag is $1/2^\tau$. Now proceeding as in the case of privacy, we obtain

$$\mathsf{Prob}[\mathsf{Succ}_B] = \frac{\mathsf{Prob}[\mathsf{Coll}]}{2^\tau} + \mathsf{Prob}[\mathsf{Succ}_A]\mathsf{Prob}[\overline{\mathsf{Coll}}].$$

The rest is similar to that in the case of privacy. □

3.2 From AE$^+$ to AEAD

The conversion from AE$^+$ to AEAD is also very simple. The encryption and decryption algorithms are shown in Figure 2. Note that though we concatenate in the order $H||N$, the

Fig. 2. Encryption and Decryption using AEAD.

$\mathsf{AEAD.Enc}_K(N, H, M)$	$\mathsf{AEAD.Dec}_K(N, H, C, \mathsf{tag})$
return $\mathsf{AE}^+.\mathsf{Enc}_K(H\|\|N, M)$.	return $\mathsf{AE}^+.\mathsf{Dec}_K(H\|\|N, C, \mathsf{tag})$.

concatenation $N||H$ will have the same security. Our reason for choosing the first option is that this may provide some additional efficiency benefits as described later.

The security of AEAD from that of AE^+ is immediate. In the AEAD protocol, N does not repeat. Consequently, the string $H||N$ in the AE^+ protocol also does not repeat. Hence, if the AE^+ protocol is secure, then so is the AEAD protocol. The idea is to show that if there is an adversary A for AEAD, then one can construct an adversary B for AE^+. The details are easy to check. We briefly mention the case of forging query. A behaves in the following manner – it adaptively makes some encryption queries and then finally makes one forging query. The i-th encryption query is of the form (N_i, H_i, M_i) and in return it expects a proper (C_i, tag_i). Adversary B obtains (C_i, tag_i) by querying its own oracle with $(H_i||N_i, M_i)$. It then returns (C_i, tag_i) to A. Consider the tuple $\Gamma_i = (N_i, H_i, M_i, \mathsf{tag}_i)$ associated with the i-th encryption query.

The forging query by A is of the form (N, H, M, tag) and it must not be equal to Γ_i for any i. Clearly, then $(N||H, M, \mathsf{tag})$ is not equal to $(N_i||H_i, M_i, \mathsf{tag})$ for any i and B outputs $(N||H, M, \mathsf{tag})$ as its own forgery attempt. By the construction of the protocol it is clear that B is successful if A is successful. Based on this discussion, we state the following result.

Theorem 2.

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{AEAD}}(q, t) = \mathbf{Adv}^{\mathrm{priv}}_{\mathsf{AE}^+}(q, t_1) \text{ and } \mathbf{Adv}^{\mathrm{auth}}_{\mathsf{AEAD}}(q, t) = \mathbf{Adv}^{\mathrm{auth}}_{\mathsf{AE}^+}(q, t_).$$

Here $t_1 = t + t'$, where t' is the bookkeeping time which is proportional to the total length (in bits) of all the queries made by the adversary for the AEAD protocol.

3.3 AEAD From AE

Combining the two constructions given above one obtains an AEAD protocol from an AE protocol and a collision resistant hash function. Basically, for each encryption request consisting of (N, H, M), where N is a nonce of length n, H is an arbitrary length header and M is a message, first concatenate N and H, then hash $N||H$ to obtain L and invoke the AE protocol on (L, M). The obtained output (C, tag) is defined to be the output of the AEAD protocol on input (N, H, M).

In short,

$$\left.\begin{aligned} \mathsf{AEAD.Enc}_K(N, H, M) &= \mathsf{AE.Enc}_K(h(H||N), M) \text{ and} \\ \mathsf{AEAD.Dec}_K(N, H, C, \mathsf{tag}) &= \mathsf{AE.Dec}_K(h(H||N), C, \mathsf{tag}). \end{aligned}\right\} \quad (1)$$

4 Issues

4.1 Generic Construction and Single Key

The construction of AEAD described in this paper is simple and generic. In particular, it can be instantiated with any collision resistant hash function and an AE protocol. The only

restriction is that the output of the hash function must have the same length as that of nonces used in the underlying the AE protocol.

The AEAD protocol uses a single key which is the same as the key of the AE protocol. Many AE protocols are known which use a single key that is same as the secret key of the underlying protocol. Thus, for all such protocols, the secret key of the resulting AEAD protocol is the same as the secret key of the underlying block cipher.

The previous generic constructions due to Rogaway [10] require two keys. This is due to the fact that encryption is done using one key and the header is processed to obtain a MAC using another key. For the security proof, these keys are required to be independent. Rogaway [10] shows that it is possible to build single key AEAD from the AE protocol OCB and the MAC algorithm PMAC. In this situation, the entire construction is treated as a single algorithm and the security proof is long and complicated.

In contrast, our construction avoids the use of the second key by hashing the nonce-header combination to obtain a nonce for the AE protocol. This makes it possible to use our algorithm with any AE protocol and not just OCB. From a practical point of view this increases a designers flexibility in choice of algorithms.

4.2 Efficiency and Online Computation

The described AEAD protocol is quite efficient in that it adds minimal overhead to the underlying AE protocol. The only extra computation is that of hashing the header-nonce combination. This requires a single pass over the data stream consisting of header-nonce. Since the header is to be authenticated, a one time processing of the header is the minimum computation that one would expect.

The other issue is that of online processing. The header can be processed "on-the-fly", i.e., successive header blocks are hashed as they become available and once processed are not required to be stored. The entire AEAD algorithm can be processed on-the-fly if the underlying AE algorithm supports this behaviour.

4.3 Pre-Computation

For some applications, it may happen that the header remains unchanged for a particular session. For each message in the session a new nonce is generated and the message is encrypted with the session header and the nonce. For such applications it will be advantageous to process the header once and use the result for all the messages. In our AEAD protocol, this can be done by hashing (a suitable initial part of) the header H to obtain an intermediate hash δ. Suppose the nonces used in the session are $N_1, N_2, \ldots$ The nonces for the AE protocol are obtained by hashing $\delta||N_1, \delta||N_2, \ldots$. The results are the same as hashing $H||N_1, H||N_2, \ldots$. Since δ is reused the repetitive hashing of δ can be avoided.

4.4 Performance

In our approach, the header is hashed while in ciphertext translation [10, 11], the header is processed using a block cipher based PRF. Usually, hashing a string is faster than applying a PRF to it. So, our proposal is expected to be faster than the suggestion in [10, 11].

There is another possible advantage. Suppose the header is some public information such as an IP address. Several users may share the same IP address. However, the secret keys of the different users will not be the same. In our proposal, the common header will be hashed once and the same hash value will be used by the different users. On the other hand, if a PRF is used to process the header, then each user will necessarily have to perform separate processing of the header.

5 Instantiating the AEAD Protocol

We propose that the function $h()$ be instantiated by an "off-the-shelf" hash function like SHA-256 [14]. The size of the digest is then 256 bits. Consequently, to build the AEAD protocol, we need an AE protocol whose nonces are 256 bits.

The AE protocols in [11, 3] can handle n-bit nonces when used with n-bit block ciphers. So, if used with AES-128, these AE protocols cannot be directly combined with SHA-256. Next, we describe a variant of these protocols so that they can handle $2n$-bit nonces when used with an n-bit block cipher.

5.1 Tweakable Block Cipher and Authenticated Encryption

Rogaway [11] introduced the XE and XEX construction of tweakable block ciphers. This was generalized in [3] to a ring $\mathbf{R}$, where $\mathbf{R}$ can be instantiated by either $\mathbb{Z}_{2^n}$ or $GF(2^n)$. These constructions extend a block cipher $E_K : \{0,1\}^n \rightarrow \{0,1\}^n$ to a tweakable block cipher whose tweak space is $\{0,1\}^n \times \{1, 2, \ldots, 2^n - 2\}$.

Here we show that it is easy to modify the construction to obtain a tweakable block cipher whose tweak space is $\{0,1\}^{2n} \times \{1, 2, \ldots, 2^n - 2\}$. In other words, a tweak consists of 2 n-bit blocks and a positive integer less than $2^n - 1$. (We note that the construction easily extends to handle t n-bit blocks, where $t \geq 2$ is a fixed integer.) Let $T = (N, l)$ be a tweak, where $N = (N_1, N_2)$ and N_1, N_2 are n-bit blocks. We define a TBC $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \rightarrow \{0,1\}^n$ where $\mathcal{T} = \{0,1\}^{2n} \times \{1, 2, \ldots, 2^n - 2\}$. As is usual, we write $\widetilde{E}_K^{N,l}$ to denote $\widetilde{E}(K, (N, l), M)$.

XE Construction: $\widetilde{E}_K^{N,l} = E_K(M + \Delta)$.
XEX Construction. $\widetilde{E}_K^{N,l} = E_K(M + \Delta) - \Delta$.

In the above, $\Delta = f_l(\mathcal{N})$ and $\mathcal{N} = E_K(E_K(N_1) + N_2)$. The operations $+$ and $-$ are over the ring $\mathbf{R}$. The only difference from the construction in [3] is in the definition of $\mathcal{N}$. In [3], N is an n-bit block and $\mathcal{N}$ is defined to be $E_K(N)$; here N is a $2n$-bit block (N_1, N_2) and $\mathcal{N}$ is defined to be $E_K(E_K(N_1) + N_2)$. This difference, however, does not cause any problem in the security analysis of the XE and the XEX construction given in Theorem 1 of [3].

Rogaway [11] showed how to construct an AE protocol from the TBC he defined in [11]. In [3], it was shown that essentially the same construction also holds for the general class of TBC defined in [3]. The TBC is in turn instantiated using a block cipher using the XEX construction used in [3], i.e., with N to be an n-bit string. The same construction holds when N is a $2n$-bit string as in the XEX construction mentioned above.

We briefly describe encryption/decryption methods of the AE protocol and identify two variants depending on whether the nonce is an n-bit string or a $2n$-bit string. This description is based on [3] which is a generalization of [11].

Suppose the input is a nonce N and message $M_1, \ldots, M_{m-1}, M_m$, where M_m is a possible partial block and the other M_is are n-bit blocks. The output is $C_1, \ldots, C_{m-1}, C_m$ and a τ-bit tag tag, where $|C_m| = |M_m|$ and the other C_is are n-bit blocks. Then

$$\left.\begin{array}{ll} C_i = E_K(M_i + \Delta_{i,0}) - \Delta_{i,0} & 1 \le i \le m-1; \\ C_m = M_m + \mathsf{Last}_l(\mathsf{Pad}); & \\ \mathsf{tag} = \mathsf{Last}_\tau(E_K(\mathsf{sum} + \Delta_{m,1}) - \Delta_{m,1}). & \end{array}\right\} \quad (2)$$

Here, as before the operations $+$ and $-$ are over $\mathbf{R}$ and the definitions of the various terms are as follows.

1. $l = |M_m| = |C_m|$,
2. $\mathsf{Pad} = E_K(\mathsf{bin}_n(l) + \Delta_{m,0}) - \Delta_{m,0}$,
3. $\mathsf{sum} = (M_1 + \cdots + M_{m-1}) + ((0^{n-l}||C_m) - \mathsf{Pad})$,
4. $\Delta_{i,b} = f_{\phi(i,b)}(\mathcal{N})$,
5. $\phi(i,b)$ is an injective map from $\{1, 2, \ldots, 2^{n/2}\} \times \{0,1\} \to \{1, 2, \ldots, 2^n - 2\}$.,
6. $\mathsf{Last}_l(x)$ returns the l least significant bits of a binary string x of length $\ge l$ and
7. $\mathsf{bin}_n(l)$ is the n-bit binary representation of the integer l with $0 \le l \le 2^n - 1$.

Equation (2) describes the encryption algorithm. The decryption algorithm is easily obtained from this description – given $(C_1, \ldots, C_{m-1}, C_m)$ and tag, obtain $M_1, \ldots, M_{m-1}, M_m$, regenerate the tag and compare with the given tag.

The only thing we have not specified so far is how to obtain $\mathcal{N}$ from N. Doing this gives rise to the two variants, which we call AE_1 and AE_2.

1. If the nonce N is an n-bit, then $\mathcal{N} = E_K(N)$ and the resulting algorithm is called AE_1.
2. On the other hand, if the nonce $N = (N_1, N_2)$ is a $2n$-bit string, then $\mathcal{N} = E_K(N_2 + E_K(N_1))$ and the resulting algorithm is called AE_2.

AE_1 is the algorithm given in [3], while AE_2 is the extension to $2n$-bit nonces that we have introduced here.

OCB. Rogaway [11] defined OCB which can be seen as a special case of AE_1 by the following instantiation. (The manner in which sum is generated is slightly different from that of [11].) $\mathbf{R}$ is taken to be $GF(2^n)$ and $\Delta_{i,b} = (1+x)^b x^i \mathcal{N}$, where the multiplication is done modulo a primitive polynomial $\tau(x)$ of degree n over $GF(2)$. In fact, the field $GF(2^n)$ is also realised using this $\tau(x)$. The definition of $\Delta_{i,b}$ implicitly defines $\phi(i,b)$ to be the map $i + Lb$, where L is the discrete log of $(1+x)$ to base x in $GF(2^n)$ realised using $\tau(x)$. This has been called the technique of linear separation in [3]. The value of L should be sufficiently "large" and for $n = 128$, Rogaway [11] computes this value using MAPLE for the specific $\tau(x)$ given in the paper. However, for $n = 256$, the value of L is not given for any $\tau(x)$. In fact, computing discrete logarithm over $GF(2^{256})$ is not easy.

Other AE protocols. Other AE_1 protocols having efficiency similar to that of OCB were described in [3]. It was shown that $\mathbf{R}$ can be instantiated as both $\mathbb{Z}_{2^n}$ and as $GF(2^n)$. We mention one instantiation; for the other instantiations, refer to [3]. Let $\mathbf{R}$ be $GF(2^n)$ and $\Delta_{i,b} = x^{2i+b}\mathcal{N}$. This technique has been called interleaved separation and unlike linear separation does not require the computation of discrete logarithm over $GF(2^n)$. So, for example, this method easily specifies an AE protocol for 256-bit block ciphers.

5.2 Building AEAD Protocols

We consider combination with the AE protocols described in Section 5.1. Two variants are described in 5.1 – AE_1 and AE_2. Both use an n-bit block cipher, but AE_1 uses n-bit nonces while AE_2 uses $2n$-bit nonces.

Combining with a 128-bit block cipher. Suppose that we wish to use AES-128 [4] as the block cipher. Then, we combine $h()$ with AE_2, where AE_2 is instantiated with AES-128.

Combining with a 256-bit block cipher. If we wish to use a 256-bit block cipher (possibly Rijndael-256), then we combine $h()$ with AE_1, where AE_1 is instantiated with the desired block cipher.

In both cases, we obtain a secure AEAD protocol where the security is based on both the underlying block cipher and the hash function.

6 Conclusion

In this work, we have described a generic method of combining a collision resistant hash function with an AE protocol to obtain an AEAD protocol. We showed how this can be used with single-pass AE protocols such as OCB [11] and its generalizations [3] both for 128-bit and 256-bit block ciphers. Our technique is simple and does not require any key material beyond that of the AE protocol. Further, processing of the header is expected to be faster in the new proposal compared to that in the technique of ciphertext translation.

References

1. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
2. John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 384–397. Springer, 2002.
3. Debrup Chakraborty and Palash Sarkar. A general construction of tweakable block ciphers and different modes of operations. *IEEE Transactions on Information Theory*, 54(5):1991–2006, 2008.
4. Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES – The Advanced Encryption Standard (Information Security and Cryptography)*. Springer, Heidelberg, 2002.
5. Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2001.
6. Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.
7. Jonathan Katz and Moti Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC*, pages 245–254, 2000.
8. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
9. Phillip Rogaway. OCB Mode. http://www.cs.ucdavis.edu/ rogaway/ocb/.
10. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.
11. Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

12. Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.
13. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
14. Secure Hash Standard. *Federal Information Processing Standard Publication 180-2.* U.S. Department of Commerce, National Institute of Standards and Technology(NIST), 2002. Available at http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf.
15. Douglas R. Stinson. Some observations on the theory of cryptographic hash functions. *Des. Codes Cryptography*, 38(2):259–277, 2006.