On Authenticated Encryption Using Stream Ciphers Supporting an Initialisation Vector

Palash Sarkar

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

Abstract. We describe a systematic framework for using a stream cipher supporting an initialisation vector (IV) to perform various tasks of authentication and authenticated encryption. These include message authentication code (MAC), authenticated encryption (AE), authenticated encryption with associated data (AEAD) and deterministic authenticated encryption (DAE) with associated data. Several schemes are presented and rigourously analysed. A major component of the constructions is a keyed hash function having provably low collision and differential probabilities. Methods are described to efficiently extend such hash functions to take double inputs and more generally multiple inputs. In particular, double-input hash functions are required for the construction of AEAD schemes. An important practical aspect of our work is that a designer can combine off-the-shelf stream ciphers with off-the-shelf hash functions to obtain secure primitives for MAC, AE, AEAD and DAE(AD).
Keywords: stream cipher with IV, universal hash function, authentication, authenticated encryption with associated data, deterministic authenticated encryption, modes of operations.

1 Introduction

Stream ciphers are one of the basic primitives for performing cryptographic operations. In the conventional view, a stream cipher is considered to be a pseudo-random generator. It takes as input a short secret random seed (called the secret key) and produces as output a long random looking keystream. Encryption is performed by bitwise XORing the keystream to the message. Decryption is performed by regenerating the keystream (from the shared secret seed) and XORing it to the ciphertext.

Stream ciphers defined in the above manner are not very convenient for applications. A single secret key cannot be securely used to encrypt two different messages. So, for each message, it is required to have a separate secret key. As a result, the key management issue becomes complicated.

In the modern view, a stream cipher SC takes as input a secret key K and an initialisation vector (IV). One of the early works which advocated this approach was [26] and all proposals in the estream [2] portfolio support an IV. The IV is not required to be secret. A message M is encrypted as $M \oplus \mathsf{SC}_K(\mathsf{IV})$. The IV is communicated to the receiver publicly, or could be generated at both ends as a counter value. Flexibility in usage arises from the fact that the same key can now be used with different messages; the IV only needs to be changed. Since there is no secrecy requirement on the IV, this is a much more easier task to manage.

This view of stream ciphers has, however, changed the theoretical model. The keystream that is used to encrypt a particular message, is still required to appear random to a computationally bounded adversary. So, the requirement is that for distinct values of IV, the outputs $\mathsf{SC}_K(\mathsf{IV})$ should appear independent and uniform random to a computationally bounded adversary. This is

theoretically captured by the notion of a pseudo-random function (PRF), and in the modern view, a stream cipher with IV is modelled as a PRF. See [6] for further justification of this view.

The starting point of our work is the model of a stream cipher with IV as a PRF. This is a PRF which takes as input a short fixed-length string and produces as output a long string. We consider the problem of constructing higher-level primitives using such a PRF. The primitives that we consider are message authentication code (MAC), authenticated encryption (AE), AE with associated data (AEAD) and deterministic authenticated encryption (DAE) with associated data. For each of these primitives, we describe efficient constructions and provide formal security analyses.

All of the primitives that we consider have important practical applications. Efficient methods to construct them are required to actually deploy cryptographic mechanisms. Previous work has mostly considered methods to construct these primitives based on block ciphers. Our work is the first systematic work showing how to obtain these primitives from stream ciphers supporting an IV.

A core requirement in all the constructions is a keyed hash function. Such a hash function is selected from a family $\{\mathsf{Hash}_\tau\}$ by choosing a uniform random τ. For all distinct x and x' and any y, we require the probability that $\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x')$ equals y to be low. (We are assuming that Hash_τ produces a fixed-length bit string as output). Such hash functions are called almost XOR universal hash functions. Starting from [11, 34, 15], constructions and efficient implementations of such functions have been studied extensively in the literature (see for example [33, 17, 10, 8]).

For the construction of AEAD schemes, we need an extended notion. The above kind of hash functions take a single input. To handle associated data, the hash function needs to take two inputs. A double-input hash function cannot be naively constructed from a single-input hash function by merely concatenating the two inputs. This is because of the fact that even if (X, Y) is not equal to (X', Y'), it may still be the case that $X||Y = X'||Y'$. Applying a single-input hash function to the concatenated value will lead to the same output even though the pairs (X, Y) and (X', Y') are distinct.

To tackle this problem, we need a suitable encoding of the pair and then apply the single-input hash function. Two generic encodings are described. Further, we describe modifications of some well-known hash functions such as Poly1305 [8] and NH and NH^T [10] to obtain double-input hash functions. A generic method for modifying a hash function to tackle variable length inputs is described in [10]. This method is not suitable for use with a stream cipher with IV. Similarly, a generic method described in [27] to extend a single-input PRF to a multi-input PRF is not suited for use with a stream cipher with IV. Detailed discussion on these issues are given later.

As mentioned earlier, the constructions that we describe are important from an application point of view. There are several well known stream ciphers and hash functions. SNOW [13] is a well-known example. The eStream project [2] has resulted in stream ciphers geared either towards fast software implementations or for small hardware implementations. Combining a fast software oriented stream cipher with a fast software oriented hash function using one of the methods proposed in this work will result in a secure primitive with a fast software implementation. Similarly, one can combine a hardware oriented stream cipher with a hardware oriented hash function using our methods to get a secure primitive with small hardware footprint. So, in a sense, our constructions are general templates for obtaining stream cipher based primitives for MAC, AE, AEAD and DAE(AD).

Prior and related work. The literature contains several scattered works on using a stream cipher for MAC and AE. For MAC, the basic idea goes back to [34].

The combined primitive of authenticated encryption was formalised in [19, 3]. Separate definition of privacy and authenticity of an encryption scheme was given in [5]. Modes of operations of block

ciphers to achieve AE have been studied in the literature and several schemes are known [18, 16, 25, 24, 12, 30]. NIST of USA has standardised one such scheme, called GCM [20]. The issue of authenticating an associated data was first formally tackled in [23]. Later works such as [24, 20, 31] incorporated methods to perform AE and at the same time authenticate an associated data. Deterministic authenticated encryption (with associated data) was formalised in [27] to model the so-called key-wrap problem. Existing works on provable security treatment of AE, AEAD and DAE(AD) schemes have almost exclusively focussed on modes of operations of block ciphers to achieve these functionalities.

A method for achieving AE using a stream cipher is given in [7]. This is one of the four constructions that we define. As far as we are aware of, there has been no previous work on constructing AEAD schemes from a stream cipher with IV. Similarly, there has been no previous work on constructing DAE(AD) schemes from a stream cipher with IV. Stream cipher based constructions for AE have been proposed as a combined primitive (see for example HELIX [14] and its later version PHELIX) but have not been successful. HELIX has been cryptanalysed in [21].

We would like to point out the differences of our AE constructions with the generic constructions for AE in [4]. The schemes in [4] provide generic methods to combine a symmetric encryption scheme (typically a block cipher mode of operation) and a MAC scheme to obtain AE schemes. (The work [4] pre-dates the notions of AEAD and DAE(AD) and these as well as MAC schemes are not considered in [4].) In contrast, we provide methods for combining a stream cipher with IV with suitable hash functions with provable differential properties to obtain AE (and other) schemes. If one takes any of the generic schemes proved secure in [4] and instantiates the symmetric encryption scheme by a stream cipher with IV and the MAC scheme by an AXU hash function, then the resulting particular scheme is not guaranteed to be secure. A main reason for this is that AXU hash functions by themselves are significantly weaker than MAC schemes. In contrast, all schemes that we propose combine stream ciphers with IV and hash functions and are accompanied by provable security guarantees (under the assumption that the underlying stream cipher is secure as a PRF). Apart from this basic difference in security guarantees, there are other differences in the overall approach of our work and that of [4]. We carefully consider important practical issues such as obtaining the possibly long keys of the hash function and reducing the number of initialisations of the stream cipher. Such issues are absent in [4] which is primarily a theoretical work.

2 Preliminaries

In this section, we introduce a few basic notions.

2.1 Stream Cipher With IV

Let $\mathsf{SC}_K : \{0,1\}^n \to \{0,1\}^L$ be a stream cipher with IV, i.e., for every choice of K, SC_K maps an IV of length n bits to a string of length L bits. The length L is assumed to be long enough for practical sized messages to be encrypted. Actual encryption of a message M of ℓ bits using an initialisation vector IV is done by XORing the first ℓ bits of $\mathsf{SC}_K(\mathsf{IV})$ to the message. By a slight abuse of notation, we will write this as $M \oplus \mathsf{SC}_K(\mathsf{IV})$. The IV is from an IV-space which is defined to be $\{0,1\}^n$. In other words, initialisation vectors are n-bit strings. The key K is from a suitable (finite) key space. For all practical designs, keys are at least as long as the IVs [2].

In the above, we have taken the output to be a (long) fixed length string. A suitable length prefix of the output is used to encrypt a message. This allows a clean mathematical description of the functionality of a stream cipher with IV. An alternative would be to consider the outputs to

be of variable lengths. This causes difficulties in the formulation. Given a particular value of the IV, the length of the output would not be known a priori. It would depend on the length of the message to be encrypted. Our formulation avoids such difficulties without any loss of generality.

Consider an algorithm to generate the output of a stream cipher from the inputs K and IV. For most practical stream ciphers [2], such an algorithm goes through two phases. First, there is an initialisation phase, in which no output is produced. Then, the output bits begin to be produced and this phase is called the keystream generation phase. The speed at which the output bits are produced in the keystream generation phase can be very fast. In comparison, the latency of the initialisation phase can be significant. As a consequence of the latency in the initialisation phase, for most practical stream ciphers, it is advisable to avoid repeated initialisations. We take this aspect into account in our constructions of higher level primitives.

2.2 Pseudo-Random Function (PRF)

Let $\mathcal{D}$ and $\mathcal{R}$ be finite non-empty sets and f is a random (but, not necessarily, uniform random) function from $\mathcal{D}$ to $\mathcal{R}$. In other words, f is drawn from the set of all functions from $\mathcal{D}$ to $\mathcal{R}$ according to some distribution.

For cryptographic applications, a random function f arises from a function family $\{F_K\}_{K\in\mathcal{K}}$, $F_K : \mathcal{D} \to \mathcal{R}$ where a key K is drawn uniformly at random from the finite set $\mathcal{K}$ and f is set to be equal to F_K. The randomness in f arises from the uniform random choice of K.

A possible distinguisher (also called an adversary) $\mathcal{A}$ is a probabilistic algorithm which is given oracle access to f. It adaptively queries the oracle with inputs $x^{(1)}, \ldots, x^{(q)}$, receiving in return the responses $f(x^{(1)}), \ldots, f(x^{(q)})$. At the end of the interaction, $\mathcal{A}$ outputs a bit. Denote by $\mathcal{A}^f \Rightarrow 1$, the event that this output is 1.

Let f^* be a function chosen uniformly at random from the set of all functions mapping $\mathcal{D}$ to $\mathcal{R}$. Adversary $\mathcal{A}$'s interaction with f^* is defined in a manner similar to the above and denote by $\mathcal{A}^{f^*} \Rightarrow 1$, the event that the output of $\mathcal{A}$ after interaction with f^* is 1. The PRF-advantage of $\mathcal{A}$ in distinguishing f from f^* is defined to be the following.

$$\mathsf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) = \Pr[\mathcal{A}^f \Rightarrow 1] - \Pr[\mathcal{A}^{f^*} \Rightarrow 1]. \tag{1}$$

Informally, the above definition captures the idea that the function f is indistinguishable from a uniform random function f^*. In the context of function families, the above can be viewed in the following manner. A uniform random K is chosen and kept secret from the adversary $\mathcal{A}$. The adversary adaptively queries the function F_K on distinct inputs $x^{(1)}, \ldots, x^{(q)}$ receiving in return the values $F_K(x^{(1)}), \ldots, F_K(x^{(q)})$. These values should "look random" to $\mathcal{A}$.

A measure of the amount of interaction of the adversary with the oracle f is the number of queries q. This measure, however, may not capture the entire picture. The length of the queries could vary. A better measure is the so-called query complexity. The total query complexity σ of $\mathcal{A}$ is defined to be the total number of bits it provides in all the queries. In other words, this is the sum of the lengths of all the individual queries, i.e., $\sigma = \sum_{i=1}^{q} |x^{(i)}|$. The maximum (more precisely, the supremum) of such advantages over all adversaries which run in time t, make q queries and have total query complexity σ is denoted by $\mathsf{Adv}_f^{\mathrm{prf}}(t, q, \sigma)$.

The definition of advantage in (1) does not use the absolute value. Note that for any adversary $\mathcal{A}$ there is another adversary $\mathcal{A}'$ which is the same as $\mathcal{A}$ except that it flips the output of $\mathcal{A}$ to provide its output. So, if the advantage of $\mathcal{A}$ is negative, then the advantage of $\mathcal{A}'$ is positive. Since the resource bounded advantage is defined to be the maximum over all adversaries, this value will

always be positive. On the other hand, avoiding absolute values makes it somewhat simpler to work with the inequalities. This approach, in fact, has been used earlier by other authors.

Stream Cipher with IV and PRF. The security assumption on the stream cipher with IV is that of a pseudo-random function. See [6] for a discussion on the justification of this assumption.

A stream cipher with IV can be considered to be a function family $\{\mathsf{SC}_K\}_{K\in\mathcal{K}}$ where $\mathcal{D} = \{0,1\}^n$ and $\mathcal{R} = \{0,1\}^L$. A random function arises from this family through the uniform random choice of a key K. For the PRF-assumption, the requirement is that SC_K is computationally indistinguishable from a function which is chosen uniformly at random from the set of all functions which map $\{0,1\}^n$ to $\{0,1\}^L$. By $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t,q,\sigma)$ we will denote the resource-bounded advantage of an adversary in breaking the PRF-property of SC. In this context, all queries are n-bit strings and so $\sigma = qn$.

2.3 Hash Functions

Let $\{\mathsf{Hash}_\tau\}$ be a keyed family of hash functions on a common domain and a common range. The key τ is chosen from an appropriate finite set. Two kinds of probabilities [9] are defined for the hash function family.

Collision probability. For all distinct x and x', the collision probability of Hash_τ corresponding to the pair (x, x') is $\Pr[\mathsf{Hash}_\tau(x) = \mathsf{Hash}_\tau(x')]$, where the probability is taken over the uniform random choice of τ.

Differential probability. For all distinct x and x' and any y, the differential probability of Hash_τ corresponding to the triplet (x, x', y) is $\Pr[\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x') = y]$, where the probability is taken over the uniform random choice of τ.

If all the collision probabilities are bounded above by ε, then Hash_τ is said to be almost universal (ε-AU); if all the differential probabilities are bounded above by ε, then Hash_τ is said to be almost XOR universal (ε-AXU).

In this work, we will be taking the range of a hash function to be the set of all strings of some fixed length. This set forms an additive group under the operation $\oplus$. More generally, one can consider the range to be an arbitrary finite group and define the corresponding differential probability with $\oplus$ replaced by the group operation.

There are well known examples of both AU and AXU hash families. Univariate polynomials over finite fields give rise to such families. For such hash function families, a key τ usually consists of a fixed number of elements of the underlying finite field and can be represented by a fixed-length bit-string. The basic idea of polynomial-based hashing yields function families which are AU/AXU only for fixed-length inputs. These can be modified to handle variable-length inputs [8, 10, 32]. The output (called a digest) is a short fixed-length string.

Another class of hash families are known. These can also handle variable length inputs and produces short fixed-length digests. But, the key τ for such hash functions can be quite long. A prefix of the key having length equal to the length of the message is actually used. In certain cases, the length of the key-prefix is actually slightly greater than the length of the message to be hashed. Examples of such hash families are known [15, 10, 29]. In particular, UMAC [10] is a well-known example which can be used to perform very fast hashing in software.

One issue about using such hash functions is the key. Since the key can be quite long, it is not practical to store the key. Instead, a stream cipher is used to generate the key. In this work, we will consider in details how such hash functions can be securely and efficienctly combined with a stream cipher supporting an IV.

For the sake of convenience, we will distinguish two types of hash functions.

Type-I. For such hash functions, the key length is independent of the message length. Typically, a key will be a short fixed-length bit string.

Type-II. For such hash functions, the number of bits of the hash key required to produce the digest is as long as the message length. In some cases, it is actually slightly longer than the message length. Suppose τ is the hash key required to hash a message M and τ' is any string such that τ is a prefix of τ'. Then, $\mathsf{Hash}_\tau(M) = \mathsf{Hash}_{\tau'}(M)$, i.e., using a key longer than that necessary to hash a message M does not change the digest.

Distribution of $\mathsf{Hash}_\tau(x)$. Given x and y, the probability $\Pr[\mathsf{Hash}_\tau(x) = y]$ is well defined. This probability is taken over the uniform random choice of τ. In fact, $\mathsf{Hash}_\tau(x)$ is a random variable which is distributed over the range of the hash function. As an example, consider $\mathsf{Hash}_\tau(x)$ to be the usual polynomial based hash function over the field $GF(2^n)$. In this case, x is a tuple $(x_0, \ldots, x_t)$, where x_is are n-bit strings which encode elements of $GF(2^n)$. The definition of polynomial-based Hash_τ is as follows.

$$\mathsf{Hash}_\tau(x_0, \ldots, x_t) = x_0 \oplus \tau x_1 \oplus \cdots \oplus x_t \tau^t.$$

For any fixed y in $GF(2^n)$, the probability that $\mathsf{Hash}_\tau(x) = y$ is at most $t/2^n$ (which follows from the fact that a univariate polynomial of degree t over a field has at most t roots).

For other examples of Hash_τ (such as the multilinear map from [15]), the random variable $\mathsf{Hash}_\tau(x)$ is uniformly distributed over the range. One should note that this does not imply that $\mathsf{Hash}_\tau(x)$ is a "random oracle" or even a PRF which is a much stronger notion: this would require that for distinct $x_1, \ldots, x_k$, the values $\mathsf{Hash}_\tau(x_1), \ldots, \mathsf{Hash}_\tau(x_k)$ "look" independent and uniformly distributed. The almost uniform distribution of $\mathsf{Hash}_\tau(x)$ does not imply the PRF condition.

For the analysis of most of the schemes, we will not require the almost uniform distribution of $\mathsf{Hash}_\tau(x)$. This will be required at one or two places and will be mentioned explicitly.

2.4 Nonce-Based Message Authentication Code

Here we will consider deterministic MAC algorithms which utilise a nonce. Such a MAC algorithm takes as input a nonce N and a message M and produces a tag as $\mathsf{tag} = \mathsf{MAC}_K(N, M)$. The key K is secret and is shared between a sender and a receiver. The sender generates tag and sends (N, M, tag) to the receiver; the receiver regenerates the tag from the received nonce and the message and compares to the received tag. If they are equal, the receiver accepts, otherwise, the receiver rejects. Nonce-based MACs have been proposed and Poly1305 [8] is a good example.

MAC algorithms are generally defined without nonces and later we will briefly consider this. Even more generally, a MAC scheme consists of two probabilistic algorithms – a tag generation algorithm and a tag verification algorithm. Deterministic MAC schemes are a special case of general MAC schemes. We will not have occasion to consider probabilistic schemes in this work.

Security of nonce-based deterministic MAC schemes is defined using a game in the following manner. An adversary $\mathcal{A}$ is a probabilistic algorithm which has oracle access to the tag generation algorithm $\mathsf{MAC}_K(\cdot, \cdot)$, which is instantiated with a uniform random key K. $\mathcal{A}$ adaptively queries the oracle with inputs $(N^{(1)}, M^{(1)}), \ldots, (N^{(q)}, M^{(q)})$ and receives in response the corresponding tags $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q)}$. The restriction on $\mathcal{A}$ is that the nonces $N^{(1)}, \ldots, N^{(q)}$ have to be distinct.

After the interaction, $\mathcal{A}$ produces a forgery (N, M, tag) such that (N, M) is not equal to $(N^{(i)}, M^{(i)})$ for any $1 \leq i \leq q$. Note that N may be equal to one of the earlier $N^{(i)}$. The advantage of $\mathcal{A}$ in breaking the MAC algorithm is defined to be the following probability:

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr\left[(N, M, \mathsf{tag}) \leftarrow \mathcal{A}^{\mathsf{MAC}_K(\cdot,\cdot)} : \mathsf{MAC}_K(N, M) = \mathsf{tag}\right].$$

This is to be considered as the probability of the event $\mathsf{MAC}_K(N, M) = \mathsf{tag}$ conditioned on the following event: $\mathcal{A}$ provides $(N^{(1)}, M^{(1)})$ and receives $\mathsf{tag}^{(1)}$; then provides $(N^{(2)}, M^{(2)})$ and receives $\mathsf{tag}^{(2)}$; ...; and finally produces (N, M, tag).

By $\mathsf{Adv}_{\mathsf{MAC}}(t, q, \sigma)$, we denote the maximum advantage of any adversary running in time t, making q queries (including the forgery attempt) and having query complexity σ. The query complexity also counts the number of bits in the forgery.

2.5 Authenticated Encryption (With Associated Data)

The goal of such a scheme is to perform both encryption and authentication of a message and also possibly authenticate an additional data or header. There are two deterministic algorithms – the encryption and the decryption algorithms.

The encryption algorithm $\mathsf{AEAD.Encrypt}_K(N, H, M)$ takes as input a nonce N, a (possibly empty) header H and a message M; and produces as output a ciphertext Γ. In many AEAD schemes (including the ones we construct), Γ consists of a pair of strings (C, tag) where the length of C is equal to the length of M and tag is considered to be the autheticator. The decryption algorithm $\mathsf{AEAD.Decrypt}_K(N, H, \Gamma)$ takes as input a nonce N, a header H and Γ and either outputs a message M or outputs $\perp$ signifying that the input is rejected. The usual correctness requirement is to be satisfied, namely,

$$\mathsf{AEAD.Decrypt}_K(N, H, \mathsf{AEAD.Encrypt}_K(N, H, M)) = M.$$

There are two security requirements on an AE scheme – confidentiality (or privacy) and authentication. Below we separately define confidentiality and authentication.

Privacy of an AEAD scheme. As usual, let an adversary $\mathcal{A}$ be a probabilistic algorithm. $\mathcal{A}$ has oracle access to $\mathsf{AEAD.Encrypt}_K(\cdot, \cdot, \cdot)$ where K is chosen uniformly at random and can adaptively query the oracle with inputs $(N^{(1)}, H^{(1)}, M^{(1)}), \ldots, (N^{q)}, H^{(q)}, M^{(q)})$ receiving in response the corresponding outputs $(\Gamma^{(1)}, \ldots, \Gamma^{(q)})$. The restriction on the queries is that the nonces $N^{(1)}, \ldots, N^{(q)}$ must be distinct. Finally, $\mathcal{A}$ outputs a bit. Let $\mathcal{A}^{\text{real}} \Rightarrow 1$ be the event that $\mathcal{A}$ outputs the bit 1 after the interaction.

Now consider the interaction of $\mathcal{A}$ with an oracle which behaves as follows. On input a triple (N, H, M), it returns a uniform random string Γ. The length of Γ is equal to the length of $\mathsf{AEAD.Encrypt}_K(N, H, M)$. This string is independent of all other inputs and responses. Denote by $\mathcal{A}^{\text{rnd}} \Rightarrow 1$ the event that $\mathcal{A}$ outputs the bit 1 after such interaction. The advantage of an adversary $\mathcal{A}$ in breaking the privacy of the AEAD scheme is defined as follows.

$$\mathsf{Adv}^{\text{aead-priv}}_{\mathsf{AEAD}}(\mathcal{A}) = \Pr[\mathcal{A}^{\text{real}} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{rnd}} \Rightarrow 1].$$

The maximum of this advantage over all adversaries running in time t, making q queries and having query complexity σ will be denoted by $\mathsf{Adv}^{\text{priv}}_{\mathsf{AEAD}}(t, q, \sigma)$.

Authentication of an AEAD scheme. As before, the adversary $\mathcal{A}$ is a probabilistic algorithm having oracle access to $\mathsf{AEAD.Encrypt}_K(\cdot, \cdot, \cdot)$ where K is chosen uniformly at random. The adversary also has an implicit access to the decryption oracle as we point out later. The goal of the adversary is to produce a forgery.

$\mathcal{A}$ adaptively queries the oracle with inputs $(N^{(1)}, H^{(1)}, M^{(1)}), \ldots, (N^{q)}, H^{(q)}, M^{(q)})$ and receives in response the corresponding outputs $(\Gamma^{(1)}, \ldots, \Gamma^{(q)})$. The restriction on the queries is that the

nonces $N^{(1)}, \ldots, N^{(q)}$ are distinct. At the end of the interaction, $\mathcal{A}$ outputs a tuple (N, H, Γ), where (N, H, Γ) is not equal to $(N^{(i)}, H^{(i)}, \Gamma^{(i)})$ for any $1 \leq i \leq q$. The nonce N in the forgery attempt may be equal to one of the earlier $N^{(i)}$.

The advantage of an adversary $\mathcal{A}$ in breaking the authentication of the AEAD scheme is defined as follows.

$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD}}(\mathcal{A}) = \Pr\left[(N, H, \Gamma) \leftarrow \mathcal{A}^{\mathsf{AEAD.Encrypt}_K(\cdot,\cdot,\cdot)} : \mathsf{AEAD.Decrypt}_K(N, H, \Gamma) \neq \perp\right].$$

The maximum of this advantage over all adversaries running in time t, making q queries and having query complexity σ will be denoted by $\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD}}(t, q, \sigma)$. Note that the definition of advantage requires an implicit access to the decryption oracle of the AEAD scheme.

AE Schemes. These are defined in a manner similar to that of AEAD schemes. The only difference is that there is no header or associated data in an AE scheme. Definitions of privacy and authentication of AE schemes are obtained from that of AEAD schemes by dropping the header. The superscripts ae-priv and ae-auth will be used to denote privacy and authentication for AE schemes.

3 Double-Input Hash Functions

Rogaway and Shrimpton [27] consider the problem of constructing a PRF which can handle a vector of strings as input. They propose an iterative method which repeatedly applies a single-input PRF successively to the different components of the vector. It is mentioned in [27] that this procedure is more efficient than using a padding rule to first convert the vector of strings to a single string and then applying a single-input PRF to this single string. In the context that [27] considers, this is perhaps correct. The PRFs considered in [27] is basically a mode of operation of a block cipher. As such, the parameter of interest is the number of block cipher invocations to process the vector.

In the context of stream cipher with IV, the parameter changes. Considered as a PRF, the input is the IV and the output is the (suitably truncated) keystream. Thus, each application of the PRF involves an initialisation of the stream cipher. As discussed earlier, for most practical stream ciphers [2] the latency of the initialisation phase is usually significant compared to the time per byte of the key generation phase. So, for using a stream cipher with IV, it is of interest to reduce the number of times the stream cipher is applied to different IVs. Using the approach of [27] to handle a vector of strings will mean that the stream cipher is invoked with different IVs and hence will be repeatedly initialised. Instead, it is better to encode the vector of strings into a single string and apply a hash function to the single string. The output of the hash function is then used as the IV to the stream cipher. In this approach, the stream cipher is initialised only once.

For this approach to work, we need efficient methods for encoding a vector of strings into a single string. To a certain extent, the method depends on the nature of the single-input hash function. We present two generic methods. For both methods, the underlying single-input hash function is defined for strings of certain lengths and has low differential and collision probabilities only for equal length strings.

We start by describing the extension of a single-input hash function to a double-input hash function. Full details of these extensions are provided. Later we mention how the same idea easily extends to construct multi-input hash functions.

Generic Construction-I. Let h_τ be a hash function which produces an n-bit digest and takes as inputs strings whose lengths are multiples of some block length β. The block length need not

be equal to the digest length n. To ensure that the domain is finite, we put a restriction on the maximum possible length of an input string. The differential probabilities of h_τ are small on *equal* length strings. We further assume that for any string x, $h_\tau(A) = h_\tau(A||0^{\beta i})$ for $i \geq 1$. This null-extension property is true for many hash functions such as those based on polynomials or linear functions.

For any two strings A and B, define the embedding $\mathcal{E}$ as follows.

$$\mathcal{E}(A, B) = A||B||\mathsf{bin}_w(\mathsf{len}(A))||10^k.$$

Here k is the minimum non-negative integer such that the total length of $A||B||\mathsf{bin}_w(\mathsf{len}(A))||10^k$ is a multiple of the block length β. We are assuming that the length of A is at most $2^w - 1$ bits.

Proposition 1. *Suppose that* $(A, B) \neq (A', B')$*;* $C = \mathcal{E}(A, B)$*,* $C' = \mathcal{E}(A', B')$*; and* $\ell = \mathsf{len}(C) \geq \ell' = \mathsf{len}(C')$*. Then the followings hold.*

1. $C \neq C'||0^{\ell-\ell'}$,
2. $\Pr[h_\tau(C) \oplus h_\tau(C') = \alpha] \leq \varepsilon$ *for every* $\alpha \in \{0,1\}^n$*. Here* ε *is an upper bound on the differential probabilities of* h_τ*.*

Proof. Let $D = A||B||\mathsf{bin}_w(\mathsf{len}(A))$, $D' = A'||B'||\mathsf{bin}_w(\mathsf{len}(A'))$ and $m = \mathsf{len}(D)$, $m' = \mathsf{len}(D')$.

If $\ell > \ell'$ then certainly $m > m'$ and so the $(m+1)$st bit of C is 1 while the $(m+1)$st bit of $C||0^{\ell-\ell'}$ is 0. So, let $\ell = \ell'$. Without loss of generality, assume that $m \geq m'$. If $m > m'$, then as before the $(m+1)$st bit of C is 1 while the $(m+1)$st bit of $C||0^{\ell-\ell'}$ is 0. So, suppose that $m = m'$. If $\mathsf{len}(A) \neq \mathsf{len}(A')$, then the last w bits of D and D' are different and so $C \neq C'$. On the other hand, if $\mathsf{len}(A) = \mathsf{len}(A')$, then this fact together with $m = m'$ implies that $\mathsf{len}(B) = \mathsf{len}(B')$. Since $(A, B) \neq (A', B')$, we have either $A \neq A'$ or $B \neq B'$. Since the lengths of A and A' are equal and so are the lengths of B and B', it follows that $A||B \neq A'||B'$. So, in all cases we see that $C \neq C'||0^{\ell-\ell'}$. This proves the first statement.

For the second statement, note that by the null-extension property of the hash function, $h_\tau(C') = h_\tau(C'||0^{\ell-\ell'})$. So,

$$\Pr[h_\tau(C) \oplus h_\tau(C') = \alpha] = \Pr[h_\tau(C) \oplus h_\tau(C'||0^{\ell-\ell'}) = \alpha] \leq \varepsilon.$$

The last inequality follows from the differential property of h. □

Define $\mathsf{HashDbl}_\tau(A, B)$ as

$$\mathsf{HashDbl}_\tau(A, B) = h_\tau(\mathcal{E}(A, B)).$$

The inputs A and B need not have lengths which are multiples of β. The only constraint on the pair (A, B) is that the length of $\mathcal{E}(A, B)$ is at most the length of an input to h_τ. It is easy to see that the function $\mathsf{HashDbl}_\tau(A, B)$ is a double argument hash function with low differential probabilities.

Polynomial hashing. Let $\mathbb{F} = GF(2^n)$ and $h_\tau : \cup_{i=1}^{L/n} \mathbb{F}^i \to \mathbb{F}$ defined as

$$h_\tau(M_0, M_1, \ldots, M_{t-1}) = \tau M_0 \oplus \tau^2 M_1 \oplus \cdots \oplus \tau^t M_{t-1}.$$

Here the block length β equals n. It is well-known that h_τ has differential probability of at most $t/2^n$ for two messages with t blocks. Further, appending a message with all-zero blocks does not change the value of the digest under h_τ, i.e., the null-extension property holds. Applying Generic Construction-I provides a desired double-input hash function.

Note. GCM [20] uses a hash function called GHASH which uses polynomial based hashing. The construction is essentially a double-input hash function, though, this is not explicitly mentioned. To handle variable length inputs, GHASH concatenates the lengths of both the inputs. As the above construction shows, this is not required. So, following the above construction, one can obtain a simplification of GHASH.

Modification of Poly-1305. The description of Poly-1305 is at byte level. A message M consists of a sequence of bytes. These are formatted into segments where each segment consists of 16 bytes and the last segment has possibly less than 16 bytes. A one is appended to each segment; for the last segment, after appending the one, further zeros are appended (if required) so that the segment length comes to 129 bits. As a result of this padding, each segment length becomes 129 bits. Denote by $\mathcal{E}\text{-Poly1305}(M)$ the output of this padding scheme. It has been shown in [8], that for any integer u, if $\mathcal{E}\text{-Poly1305}(M) - \mathcal{E}\text{-Poly1305}(M') - u \equiv 0 \bmod 2^{130} - 5$, then $M = M'$.

The digest is obtained by evaluating a polynomial at a point which is equal to the key of the hash function. This polynomial is defined by the segments which form the coefficients of the polynomial. Denote the digest as $\mathsf{poly}_\tau(\mathcal{E}\text{-Poly1305}(M))$, where τ is the key.

As before, let $\mathsf{bin}_{32}(\mathsf{len}(A))$ be the 32-bit binary encoding of the length of the bit string A. This 32-bit encoding is considered to be a sequence of 4 bytes. Define the encoding $\mathcal{E}(A, B)$ as follows:

$$\mathcal{E}(A, B) \mapsto \mathcal{E}\text{-Poly1305}(A||B||\mathsf{bin}_{32}(\mathsf{len}(A))).$$

Then the map $\mathsf{HashDbl}_\tau(A, B) = \mathsf{poly}_\tau(\mathcal{E}(A, B))$ is a double argument hash function with low differential probabilities.

Generic Construction-II. A generic method is described in [10] for the following: Given a single-input hash function which is defined for strings whose lengths are multiples of the block length β and which has low collision (resp. differential) probabilities for *equal* length strings, construct a single-input hash function which can handle arbitrary length inputs and has low collision (resp. differential) probabilities for two strings of possibly unequal lengths. The construction is the following.

Let $\{h_\tau\}$ be a family of functions where $h_\tau : \mathcal{D} \to \mathcal{R}$, with $\mathcal{D} = \cup_{i \in I}\{0,1\}^i$ for some finite non-empty index set I of non-negative integers and $\mathcal{R} = \{0,1\}^n$. In other words, the range consists of fixed length strings whereas the domain consists of strings of some particular lengths. Let a be the length of the longest string in $\mathcal{D}$ and $\alpha \geq \lceil \log_2 a \rceil$ be a positive integer. Given a message M, it is formatted into $M_1, \ldots, M_t$ such that $M_1, \ldots, M_{t-1}$ are all exactly a bits long and M_t is at most a bits.

A family $\{h'_\tau\}$ is defined as follows: On input M, if M is the empty string, then return 0^α; else, let $\mathsf{Len} = \mathsf{bin}_\alpha(\mathsf{len}(M) \bmod a)$ and i be the least non-negtive integer such that $M_t||0^i$ is in A and return

$$h_\tau(M_1)|| \cdots ||h_\tau(M_{t-1})||h_\tau(M_t||0^i)||\mathsf{Len}.$$

Note that the length is appended *after* the hashing. This approach cannot be directly used with a stream cipher with IV. The output of such a single-input hash function is not a fixed-length binary string and hence cannot be directly used as an IV of a stream cipher.

We describe a generic construction which ensures that the output is always a fixed-length string. The first part shows how to construct a single-input hash function which can handle variable length strings and the second part shows how to construct a double-input hash function. In contrast to the first generic construction that we have described, in this case, it is not required for the underlying single-input hash function to possess the null-extension property.

First part. Let M be a message and let $\mathcal{E}(M)$ be $M||10^k$ where k is the minimum positive integer such that the length of $\mathcal{E}(M)$ is multiple of the block length β and $\mathcal{E}(M)$ can be hashed using h. Suppose there are r blocks in $\mathcal{E}(M)$. We assume that r is at most $2^n - 1$, where n is the size of the digest of the hash function. For usual practical applications, n is at least 32 and so this assumption will be easily satisfied in practice. Define a new hash function as follows.

$$h^*_{\tau,\rho}(M) = h_\tau(\mathcal{E}(M)) \oplus \psi_\rho(r). \tag{2}$$

Here ψ_ρ maps n bits to n bits and satisfies the following differential property: For $r \neq r'$ and for any n-bit string α, $\Pr[\psi_\rho(r) \oplus \psi_\rho(r') = \alpha] = 1/2^n$. Such functions ψ can be constructed using the so-called powering-up method [24] and from linear feedback shift registers [28].

Proposition 2. *If the collision (resp. differential) probabilities of h are bounded above by ε, then the collision (resp. differential) probabilities of h^* given by (2) are also bounded above by ε.*

Proof. We show the result for differential probabilities. Let M and M' be two distinct strings and let α be any n-bit string. Let the number of blocks in $\mathcal{E}(M)$ and $\mathcal{E}(M')$ be r and r' respectively. There are two cases to consider.
Case $r \neq r'$. Then the following computation shows the result.

$$\begin{aligned}
&\Pr[h^*_{\tau,\rho}(M) \oplus h_{\tau,\rho}(M') = \alpha] \\
&= \Pr[h_\tau(\mathcal{E}(M)) \oplus \psi_\rho(r) \oplus h_\tau(\mathcal{E}(M')) \oplus \psi_\rho(r') = \alpha] \\
&= \Pr[\psi_\rho(r) \oplus \psi_\rho(r') = \alpha \oplus h_\tau(\mathcal{E}(M)) \oplus h_\tau(\mathcal{E}(M'))] \\
&= \frac{1}{2^n} \\
&\leq \varepsilon.
\end{aligned}$$

Case $r = r'$. This means that the lengths of $\mathcal{E}(M)$ and $\mathcal{E}(M')$ are equal. For this case, we will utilise the fact that h has low differential probabilities for equal length strings.

If the lengths of M and M' are not equal, then the 1 padded at the end of the longer string is different from the 0 present in the same position of the shorter string. If the lengths of M and M' are equal, then since they are distinct, there is a position where the two strings differ. In either case, $B = \mathcal{E}(M) \neq \mathcal{E}(M') = B'$. Further, the lengths of B and B' are equal. The following computation now completes the proof.

$$\begin{aligned}
&\Pr[h^*_{\tau,\rho}(M) \oplus h_{\tau,\rho}(M') = \alpha] \\
&= \Pr[h_\tau(\mathcal{E}(M)) \oplus \psi_\rho(r) \oplus h_\tau(\mathcal{E}(M')) \oplus \psi_\rho(r') = \alpha] \\
&= \Pr[h_\tau(B) \oplus h_\tau(B') = \alpha] \\
&\leq \varepsilon.
\end{aligned}$$

The last inequality follows from the differential property of h for equal length strings. □

Second part. Modification of h^*_τ to handle two arguments can be done in the following manner. Given two strings H and M of arbitrary lengths, define $\mathcal{E}(H, M)$ to be

$$\mathcal{E}(H, M) = H||M||\mathsf{bin}_w(\mathsf{len}(H))||10^k$$

where, H is of size at most $2^w - 1$ and k is the minimum non-negative integer such that the entire length of the string is a multiple of the block length β. Again note that the block length need not

necessarily be equal to n, which is the size of the digest. Let r be the number of blocks in $\mathcal{E}(H, M)$ and define $h^*_{\tau,\rho}(H, M)$ as follows.

$$h^*_{\tau,\rho}(H, M) = h_\tau(\mathcal{E}(H, M)) \oplus \psi_\rho(r). \tag{3}$$

Proposition 3. *Suppose that the collision (resp. differential) probabilities of h are bounded above by ε. Then the collision (resp. differential) probabilities of h^* given by (3) are also bounded above by ε.*

Proof. We prove the statement on differential probabilities. Let (H, M) and (H', M') be two distinct header-message pairs and let α be any n-bit string. Let r and r' be the number of blocks in $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ respectively.

As in the proof of Proposition 2, it can be shown that if $r \neq r'$, then

$$\Pr[h^*(H, M) \oplus h^*(H', M') = \alpha] = 1/2^n \leq \varepsilon.$$

Suppose that $r = r'$, i.e., the number of blocks in $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ are equal. Then the lengths of $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ are also equal.

If the lengths of $H||M$ and $H'||M'$ are not equal, then the lengths of $H||M||\mathsf{bin}_w(\mathsf{len}(H))$ and $H'||M'||\mathsf{bin}_w(\mathsf{len}(H'))$ are also unequal. The 1 padded to the end of the longer of these two strings is different from the 0 at the same position in the shorter string. So, $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ are distinct. Suppose now that the lengths of $H||M$ and $H'||M'$ are equal. Then the lengths of $H||M||\mathsf{bin}_w(\mathsf{len}(H))$ and $H'||M'||\mathsf{bin}_w(\mathsf{len}(H'))$ are also equal. If the length of H is different from the length of H', then $\mathsf{bin}_w(\mathsf{len}(H))$ is different from $\mathsf{bin}_w(\mathsf{len}(H'))$ and so, once more, $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ are distinct.

If the lengths of H and H' are equal, then since the lengths of $H||M$ and $H'||M'$ are also equal, it necessarily follows that the lengths of M and M' are equal. Then, by the fact that $(H, M) \neq (H', M')$, it follows that $H||M \neq H'||M'$ and so $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ are distinct. Thus, in all cases, the equal length strings $\mathcal{E}(H, M)$ and $\mathcal{E}(H', M')$ are seen to be distinct. Now applying the differential property of h for equal length strings, we get the result. □

NH. The core of the UMAC construction [10] is a non-linear hash function NH and its so-called Toeplitz version NH^T. Both these functions take as input, strings whose lengths are even multiples of a word length w and produce as outputs strings of length $2w$ bits. The collision and differential properties of these functions for equal length strings are analysed in [10].

A major advantage of this hash is its speed in software. Using the approach outlined above, both NH and NH^T can be tailored for use in the AE schemes to be proposed later. It should be noted that the generic method for securely handling variable length strings given in [10] does not provide a hash function suitable for use with a stream cipher. Further, as described above, both NH and NH^T can be converted into double-input hash functions which can handle arbitrary length strings and preserve the collision and differential properties. This makes them suitable for use in the AEAD schemes to be proposed later.

3.1 Multi-Input Hash Function

A related question is that of obtaining a hash function which takes $\kappa \geq 2$ inputs and has low differential probabilities. It is easy to extend the constructions of double-input hash functions to cover this case. We provide the details for the two generic constructions given above.

Extension of Generic Construction-I. As before, let $h_\tau : \cup_{i=0}^{L/\beta}\{0,1\}^{i\beta} \to \{0,1\}^n$ be such that the differential probabilities of h_τ are small on *equal* length strings, and h_τ satisfies the null-extension property.

Fix $\kappa \geq 2$. For strings $A_1, \ldots, A_\kappa$, define the embedding $\mathcal{E}$ as follows.

$$\mathcal{E}(A_1, \ldots, A_\kappa) = A_1||\cdots||A_\kappa||\mathsf{bin}_w(\mathsf{len}(A_1))||\cdots||\mathsf{bin}_w(\mathsf{len}(A_{\kappa-1}))||10^k.$$

Here k is the minimum non-negative integer such that the total length of the right hand side is a multiple of the block length β. Observe that the lengths of $\kappa - 1$ of the input strings are concatenated.

Define a hash function h^* on κ strings as follows.

$$h^*_\tau(A_1, \ldots, A_\kappa) = h_\tau(\mathcal{E}(A_1, \ldots, A_\kappa)).$$

As in Proposition 1, it can be shown that the differential probabilities of h^* are low if the differential probabilities of h are low.

Extension of Generic Construction-II. In this case, we do not have the null extension property of the underlying hash function.

Fix $\kappa \geq 2$. Given κ input strings $A_1, \ldots, A_\kappa$ define $\mathcal{E}(A_1, \ldots, A_\kappa)$ as follows.

$$\mathcal{E}(A_1, \ldots, A_\kappa) = A_1||\cdots||A_\kappa||\mathsf{bin}_w(\mathsf{len}(A_1))||\cdots||\mathsf{bin}_w(\mathsf{len}(A_{\kappa-1}))||10^k.$$

As before, k is the minimum non-negative integer such that the entire length of the string on the right hand side is a multiple of the block length β. Let r be the number of blocks in $\mathcal{E}(A_1, \ldots, A_\kappa)$ and define $h^*_{\tau,\rho}(H, M)$ as follows.

$$h^*_{\tau,\rho}(A_1, \ldots, A_\kappa) = h_\tau(\mathcal{E}(A_1, \ldots, A_\kappa)) \oplus \psi_\rho(r). \tag{4}$$

As in Proposition 3, it can be shown that the differential probabilities of h^* are small if the differential probabilities of h are small.

Other encodings. We describe two possible modifications.

1. Instead of appending the lengths at the end of the message, it is possible to append the length of an input component immediately after the component appears. For example, Generic Construction-I can be modified as follows.

   $$\mathcal{E}(A_1, \ldots, A_\kappa) = A_1||\mathsf{bin}_w(\mathsf{len}(A_1))||\cdots||A_{\kappa-1}||\mathsf{bin}_w(\mathsf{len}(A_{\kappa-1}))||A_\kappa||\mathsf{bin}_w(\mathsf{len}(A_\kappa))||10^k.$$

   Similarly for Generic Construction-II. The advantage of doing this is that one does not need to carry forward the encodings of the different lengths to be appended after all the inputs have been read. On the other hand, this requires the length of A_κ also to be appended and makes the resulting string longer. Note that the simpler encoding scheme

   $$A_1||\mathsf{bin}_w(\mathsf{len}(A_1))||\cdots||A_{\kappa-1}||\mathsf{bin}_w(\mathsf{len}(A_{\kappa-1}))||A_\kappa||10^k$$

   is not correct. To see this suppose $\beta = 4$ and $w = 2$, i.e., the block length is 4 and strings are of maximum length 3. Let $(A_1, A_2) = (1, 01)$ and $(A'_1, A'_2) = (10, 1)$. Then the encoding of (A_1, A_2) is $1||01||01||100$ and the encoding of (A'_1, A'_2) is $10||10||1||100$ and so the two encodings are equal.

   So, if all the lengths are padded at the end, then padding $(\kappa - 1)$ lengths is sufficient. But, if lengths are to be padded immediately following the individual strings, then all the κ lengths have to be padded.

2. The lengths of all the arguments are encoded as w-bit binary numbers. It is possible to have different length encodings of the arguments. For example, if it is known that the first argument is never more than $2^{16}-1$ bits long, then one can use a 16-bit encoding for the length of the first argument and a 32-bit encoding of the lengths of the other arguments. Whatever encoding is decided must be used consistently for all inputs. This strategy will result in reducing the length of the final encoded string.

4 MAC Construction

A well-known method [34] for constructing a MAC is to combine a PRF with a hash function in the following manner: given a nonce N and a message M, the digest is defined to be $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus \mathsf{PRF}_K(N)$. The secret key of the MAC algorithm is (K, τ), i.e., it consists of the secret key of the PRF and the secret key of the hash function. When using a stream cipher with IV, the PRF is replaced by SC, i.e., $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus \mathsf{SC}_K(N)$. Here N plays the role of the IV.

As long as τ is short, i.e., Hash is a Type-I hash function, the above construction is efficient and practical. For a Type-II hash function, the hash key τ is very long. Generating such a long uniform random bit string and storing it securely is very difficult. This reduces the practicability of the scheme.

A MAC scheme is shown in Table 1. This can be instantiated using either Type-I or Type-II hash function. For Type-II hash functions, only the prefix of τ required for hashing M is to be generated. The integrity algorithm [1] of 3GPP is a variant of the scheme given in Table 1. The only difference is that the values of R and τ are produced as $(\tau, R) = \mathsf{SC}_K(N)$. Compared to the scheme in Table 1, this change does not improve the efficiency or the practicability. It, however, makes the formal analysis more complicated. So, we have chosen to ignore this variant.

Table 1. A MAC scheme.

$\mathsf{MAC}_K(N, M)$
$(R, \tau) = \mathsf{SC}_K(N);$
$\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R.$

The analysis of the security of the MAC construction (as well as the later constructions) will involve several stages of analysis of conditional probabilities. A useful fact to keep in mind while following this analysis is the following simple result.

Proposition 4. *Let E and F be events and B is the disjoint union of two other events F_1 and F_2 none of which are null. Also, suppose that $\Pr[E|F_1]$ and $\Pr[E|F_2]$ are both at most ε. Then $\Pr[E|F]$ is also at most ε.*

Proof. From the given conditions, we have $\Pr[E \wedge F_1] \leq \varepsilon \times \Pr[F_1]$ and $\Pr[E \wedge F_2] \leq \varepsilon \times \Pr[F_2]$. Also, $\Pr[E] = \Pr[F_1] + \Pr[F_2]$. So,

$$\begin{aligned}
\Pr[E \wedge F] &= \Pr[E \wedge (F_1 \vee F_2)] \\
&= \Pr[E \wedge F_1] + \Pr[E \wedge F_2] \\
&\leq \varepsilon \times \Pr[F_1] + \varepsilon \times \Pr[F_2] \\
&= \varepsilon(\Pr[F_1] + \Pr[F_2]) \\
&= \varepsilon \Pr[F].
\end{aligned}$$

From this it immediately follows that $\Pr[E|F] \leq \varepsilon$. □

Theorem 1. *Suppose the differential probabilities of* Hash *are bounded above by* ε*, where* $\varepsilon \geq 1/2^n$*. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}(t,q,\sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t,q,\sigma) + \varepsilon.$$

Proof. In the first standard reduction, SC_K is replaced by a random oracle. On distinct inputs, this oracle returns independent and uniform random strings of appropriate lengths. The standard argument is the following. Let $\mathcal{A}$ be an adversary attacking the MAC scheme. Using $\mathcal{A}$, an adversary $\mathcal{B}$ attacking the PRF-property of SC is constructed as follows. $\mathcal{B}$ has access to an oracle which is either $\mathsf{SC}_K()$ (the real oracle) or a random oracle; $\mathcal{B}$ runs $\mathcal{A}$, if $\mathcal{A}$ makes a tag-generation query on $(N^{(i)}, M^{(i)})$, $\mathcal{B}$ queries its oracle with $N^{(i)}$ to obtain $(R^{(i)}, \tau^{(i)})$ and uses the response to generate $\mathsf{tag}^{(i)}$ which it provides to $\mathcal{A}$; at the end $\mathcal{A}$ outputs (N, M, tag); $\mathcal{B}$ makes one more query to its oracle on N and uses the output (R, τ) to regenerate the tag and compare to the tag provided by $\mathcal{A}$; if these two are equal, then $\mathcal{B}$ outputs 1, else it outputs 0. Then $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ produces a successful forgery.

The difference in the probabilities of $\mathcal{B}$ outputting 1 corresponding to the real and random oracles is upper bounded the PRF-advantage of SC. Let $\mathsf{succ}(\mathsf{real})$ (resp. $\mathsf{succ}(\mathsf{rnd})$) be the event that $\mathcal{A}$ is successful when $\mathcal{B}$'s oracle is SC_K (resp. the random oracle). Note that if $\mathcal{B}$'s oracle is real, then $\mathcal{A}$ is attacking the MAC-property. Then, it is standard to show the following.

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}(t,q,\sigma) = \Pr[\mathsf{succ}(\mathsf{real})] \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t,q,\sigma) + \Pr[\mathsf{succ}(\mathsf{rnd})].$$

The rest of the analysis is to show that $\Pr[\mathsf{succ}(\mathsf{rnd})]$ is bounded above by ε. As above, let $(R^{(i)}, \tau^{(i)})$ be the response of the random oracle on input $N^{(i)}$ and let (R, τ) be the response of the random oracle on input N, where (N, M, tag) is the forgery attempt. Recall that the constraint on nonces is that they should be distinct. Combined with the fact that the oracle is now a random one, it follows that the $(R^{(i)}, \tau^{(i)})$ are independent and uniformly distributed. The forgery attempt is (N, M, tag); if N is distinct from the previous nonces, then the (R, τ) is independent of the previous $(R^{(i)}, \tau^{(i)})$; on the other hand, if N is equal to one of the previous nonces, then this independence condition does not hold.

The probability $\Pr[\mathsf{succ}(\mathsf{rnd})]$ is the probability that $\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag}$ given the event

$$\bigwedge_{i=1}^{q} \left(\mathsf{Hash}_\tau(M^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)} \right).$$

It is not too difficult to see (by applying Proposition 4) that upper bounding $\Pr[\mathsf{succ}(\mathsf{rnd})]$ by ε can be done by fixing $M^{(1)}, \ldots, M^{(q)}$ and $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q)}$ to arbitrary values and showing that the following conditional probability is upper bounded by ε.

$$\Pr\left[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag} \,\middle|\, \bigwedge_{i=1}^{q} \left(\mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)} \right) \right]. \tag{5}$$

In the following, we will assume that $M^{(1)}, \ldots, M^{(q)}$ and $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q)}$ are arbitrary fixed (non-random) values.

There are two cases to consider. If N is different from each $N^{(i)}$, then by the random oracle assumption, (R, τ) is distributed uniformly and is independent of all previous $(R^{(i)}, \tau^{(i)})$. Due to the independence, the conditional probability in (5) is equal to the unconditional probability $\Pr[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag}]$ which in turn, is equal to $1/2^n \leq \varepsilon$.

So, suppose that $N = N^{(i)}$ for some i. Since (N, M) is not equal to $(N^{(i)}, M^{(i)})$, it necessarily follows that $M \neq M^{(i)}$. Further, (R, τ) is independent of $(R^{(j)}, \tau^{(j)})$ for $j \neq i$ and the probability in (5) is equal to

$$\Pr\left[\mathsf{Hash}_{\tau}(M) \oplus R = \mathsf{tag} | \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)}\right]. \tag{6}$$

Note that $(R, \tau) = \$(N)$ and $(R^{(i)}, \tau^{(i)}) = \$(N^{(i)})$, where $\$$ denotes the random oracle. By assumption $N = N^{(i)}$ and so $R = R^{(i)}$. (If R and τ were generated as $(\tau, R) = \mathsf{SC}_K(N)$, then this would not have held.) Since R is uniformly distributed and is independent of $\tau^{(i)}$, it follows that

$$\Pr[R \oplus \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)}] = 1/2^n. \tag{7}$$

Also, one would expect $\tau = \tau^{(i)}$. If a Type-I hash function is used, then the hash key is a fixed length string and so $\tau = \tau^{(i)}$ holds. But, for Type-II hash functions, we are working with an overloaded notation where τ (resp. $\tau^{(i)}$) is the appropriate length prefix of the hash key required for hashing the message M (resp. $M^{(i)}$). Since the lengths of M and $M^{(i)}$ can be different, we do not necessarily have $\tau = \tau^{(i)}$.

We now consider the following joint probability.

$$\Pr\left[\mathsf{Hash}_{\tau}(M) = \mathsf{tag} \oplus R, \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus R\right].$$

Assume that $M^{(i)}$ is not shorter than M. (The other case is similar.) Then the length of τ is at most the length of $\tau^{(i)}$. By the condition on Type-II hash function, $\mathsf{Hash}_{\tau}(M) = \mathsf{Hash}_{\tau^{(i)}}(M)$, i.e., hashing using a longer key does not change the digest.

$$\begin{aligned}
&\Pr\left[\mathsf{Hash}_{\tau^{(i)}}(M) = \mathsf{tag} \oplus R, \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus R\right] \\
&= \sum_{a,b} \Pr\left[\mathsf{Hash}_{a}(M) = \mathsf{tag} \oplus b, \mathsf{Hash}_{a}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b\right] \times \Pr[\tau^{(i)} = a] \times \Pr[R = b] \\
&= \frac{1}{2^n} \times \sum_{a,b} \Pr\left[\mathsf{Hash}_{a}(M) = \mathsf{tag} \oplus b, \mathsf{Hash}_{a}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b\right] \times \Pr[\tau^{(i)} = a].
\end{aligned}$$

There is no randomness in the condition $\mathsf{Hash}_a(M) = \mathsf{tag} \oplus b, \mathsf{Hash}_a(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b$, i.e., it either holds with probability 0 or with probability 1. For a fixed value of a, the condition holds with probability 1, if $b = \mathsf{Hash}_a(M) \oplus \mathsf{tag} = \mathsf{Hash}_a(M^{(i)}) \oplus \mathsf{tag}^{(i)}$, otherwise it holds with probability 0. So, for every a, there is at most one b for which the probability is non-zero. As a result,

$$\begin{aligned}
&\Pr\left[\mathsf{Hash}_{\tau^{(i)}}(M) = \mathsf{tag} \oplus R, \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus R\right] \\
&= \frac{1}{2^n} \times \sum_{a,b} \Pr\left[\mathsf{Hash}_{a}(M) = \mathsf{tag} \oplus b, \mathsf{Hash}_{a}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b\right] \times \Pr[\tau^{(i)} = a] \\
&\leq \frac{1}{2^n} \times \sum_{a} \Pr\left[\mathsf{Hash}_{a}(M) \oplus \mathsf{Hash}_{a}(M^{(i)}) = \mathsf{tag} \oplus \mathsf{tag}^{(i)}\right] \times \Pr[\tau^{(i)} = a] \\
&\leq \frac{1}{2^n} \times \Pr\left[\mathsf{Hash}_{\tau^{(i)}}(M) \oplus \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag} \oplus \mathsf{tag}^{(i)}\right] \\
&\leq \frac{\varepsilon}{2^n}.
\end{aligned}$$

Combined with (7), this shows that the probability in (6) and hence, the probability in (5) is bounded above by ε. Combined with the other inequalities, we get the desired result. □

Note. If a vector of strings is required to be authenticated, then one can use a multi-input hash function. The ensuing analysis remains unchanged.

4.1 PRF from PRF

SC maps *fixed-length* bit strings to *long* (fixed-length) strings and the security assumption on SC is that of a PRF. We briefly consider how SC can be used to construct a PRF which maps *long and variable-length strings* to *short fixed-length* bit strings. Such a PRF immediately gives a MAC algorithm which does not use a nonce.

The conversion uses a hash function which maps the long message into a short digest. SC is then applied on the short digest and a fixed length prefix of the keystream is produced as the output. From this discussion, the PRF constructed from SC and hash is the following. Given secret key (K, τ) and input M, the output tag is produced as follows: $N = \mathsf{Hash}_\tau(M)$ and $\mathsf{tag} = \mathsf{SC}_K(N)$.

The idea behind the security of this construction is the following. If $M^{(1)}, \ldots, M^{(q)}$ are distinct inputs, and Hash has low collision probabilities, then with high probability $N^{(1)}, \ldots, N^{(q)}$ are also distinct, where $N^{(i)} = \mathsf{Hash}_\tau(M^{(i)})$. Conditioned under this event and assuming SC_K to be a PRF, the outputs $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q)}$ appear to be independent and uniformly distributed to a computationally bounded adversary. We make a more detailed analysis in the context of DAE schemes.

As in the case of authentication, if τ is long, then it is advantageous to generate τ using the stream cipher itself. This variant is as follows. Here fStr is some fixed string (such as 0^n). Given a secret key K and a message M the output tag is produced as follows: $\tau = \mathsf{SC}_K(\mathsf{fStr})$; $N = \mathsf{Hash}_\tau(M)$ and $\mathsf{tag} = \mathsf{SC}_K(N)$. The idea behind the security of this scheme is almost the same as the previous one. Only difference is that we need to argue that the probability that fStr is equal to any $N^{(i)}$ is small, which follows from the fact that $\mathsf{Hash}_\tau(x)$ is almost uniformly distributed for any fixed x.

5 Constructions of AE Schemes

The proposed schemes for performing AE are given in Table 2. Only the encryption algorithms are described. Corresponding decryption algorithms can be easily constructed.

Table 2. AE schemes.

AE-1.$\mathsf{Encrypt}_{K,\tau}(N, M)$ $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$.	AE-2.$\mathsf{Encrypt}_{K,K'}(N, M)$ $\tau = \mathsf{SC}_K(K')$; $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$.
AE-3.$\mathsf{Encrypt}_{K,\tau}(N, M)$ $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$.	AE-4.$\mathsf{Encrypt}_{K,K'}(N, M)$ $\tau = \mathsf{SC}_K(K')$; $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$.

AE-1 has been suggested as a standard method of performing authenticated encryption from a stream cipher [7]. For AE-1, a Type-I hash function is suitable. This is because, for this scheme, the hash key is provided as part of the secret key of the scheme. If a Type-II hash function is desired

to be used, then the hash key will be long and it will not be practical to specify this as part of the secret key. In this case, AE-2 should be used. In this scheme, the hash key is generated from K' which itself is a fixed length string.

On the face of it, AE-2 may appear to be slower compared to AE-1, since it requires two invocations of SC. There are two points to note.

First, the two invocations of SC can be done in parallel. So, in hardware, if a Type-II hash function requires smaller space than a Type-I hash function, then it may be advantageous to use AE-2 in comparison to AE-1.

Second, the hash key τ in AE-2 does not depend on the message. So, subject to the availability of secure storage a sufficiently long hash key may be generated and stored once per session. Then the time for generating the hash key is amortised over all the messages hashed in one session. This approach can be used when messages are comparatively short which is true, for example, in internet communications. There are, of course, many related implementation issues such as cache misses which would also determine the actual speed.

Schemes AE-3 and AE-4 hash the message M instead of the ciphertext C. One possible advantage of hashing the message M instead of the ciphertext C is that the task of hashing and ciphertext generation can proceed in parallel. For hardware implementations, this can be an advantage.

In AE-2, there are two keys K and K'. If a single key system is desired, then one may generate K' itself by applying SC_K on a fixed n-bit string fStr as $K' = \mathsf{SC}(\mathsf{fStr})$. Again, this will have a minimal effect on the security bound. But, on the other hand, it will require a separate application of SC. It should be noted, though, that K' can be generated once per session.

Privacy of the schemes. We consider only AE-1 and the rest are similar. Under the assumption that SC is a PRF, the privacy of the scheme is easy to argue. Suppose the q queries are $(N^{(i)}, M^{(i)})$ and the corresponding responses are $(C^{(i)}, \mathsf{tag}^{(i)})$. The nonce restriction implies that $N^{(1)}, \ldots, N^{(q)}$ are distinct. Since $N^{(1)}, \ldots, N^{(q)}$ are distinct, by the PRF assumption on SC, a computationally bounded adversary will not be able to distinguish between $\mathsf{SC}(N^{(1)}), \ldots, \mathsf{SC}(N^{(q)})$ and independent and uniform random strings of corresponding lengths.

The strings $(R^{(1)}, Z^{(1)}), \ldots, (R^{(q)}, Z^{(q)})$ are prefixes of appropriate lengths of these strings. Since $C^{(i)} = Z^{(i)} \oplus M^{(i)}$ and $\mathsf{tag}^{(i)} = \mathsf{Hash}_\tau(C^{(i)}) \oplus R^{(i)}$, the random variables

$$(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q)}, \mathsf{tag}^{(q)})$$

also appear independent and uniformly distributed to a computationally bounded adversary. This shows the privacy of AE-1 and the following bound can be derived.

$$\mathsf{Adv}^{\text{ae-priv}}_{\text{AE-1}}(t, q, \sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t, q, \sigma). \tag{8}$$

The privacy bounds of the other schemes are the same.

Authentication of the schemes. In this case, the arguments for the first two schemes are slightly different with the argument for AE-2 being a little more complicated. We first present this argument below. Later, we give the argument for AE-3 from which the argument for AE-4 follows after some modifications based on the argument for AE-2.

Theorem 2. *Suppose the differential probabilities of* Hash *are bounded above by ε. Then*

$$\mathsf{Adv}^{\text{ae-auth}}_{\text{AE-2}}(t, q, \sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t, q, \sigma) + \frac{q+1}{2^n} + \varepsilon.$$

$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE}\text{-}4}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + \frac{q+1}{2^n} + \varepsilon.$$

Proof. We prove the result for AE-2. Suppose the queries are $(N^{(1)}, M^{(1)}), \ldots, (N^{(q)}, M^{(q)})$ and the corresponding responses are $(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q)}, \mathsf{tag}^{(q)})$. Further, suppose that the forgery attempt is (N, C, tag) with the restriction that this is not equal to $(N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$ for any $1 \le i \le q$. The forgery attempt implicitly defines $(R, Z) = \mathsf{SC}_K(N)$.

Let $\mathsf{succ}(\mathsf{real})$ be the event that the forgery is accepted. Let Repeat be the event that K' is equal to one of $N^{(1)}, \ldots, N^{(q)}$ or N. Since K' is chosen uniformly at random, $\Pr[\mathsf{Repeat}] \le (q+1)/2^n$. Then

$$\begin{aligned}\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE}\text{-}2}(t,q,\sigma) &= \Pr[\mathsf{succ}(\mathsf{real})] \\ &= \Pr[\mathsf{succ}(\mathsf{real})|\mathsf{Repeat}] \times \Pr[\mathsf{Repeat}] + \Pr[\mathsf{succ}(\mathsf{real})|\overline{\mathsf{Repeat}}] \times \Pr[\overline{\mathsf{Repeat}}] \\ &\le \Pr[\mathsf{Repeat}] + \Pr[\mathsf{succ}(\mathsf{real})|\overline{\mathsf{Repeat}}].\end{aligned}$$

(The event Repeat is the essential difference between the analysis of AE-1 and AE-2. In AE-1, this event is not defined and τ is a uniform random string which is independent of all other random variables. In AE-2, under the condition $\overline{\mathsf{Repeat}}$, and assuming SC to be a PRF, we can assume that τ is a uniform random string which is independent of all other random variables. The rest of the proof is the same for both schemes.)

Next, SC_K is replaced by a random oracle. On distinct inputs, this oracle returns independent and uniform random strings of appropriate lengths. Let $\mathsf{succ}(\mathsf{rnd})$ be the event that the adversary is successful against the random oracle. Then, it is standard to show the following.

$$\Pr[\mathsf{succ}(\mathsf{real})|\overline{\mathsf{Repeat}}] \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + \Pr[\mathsf{succ}(\mathsf{rnd})|\overline{\mathsf{Repeat}}].$$

The analysis of $\Pr[\mathsf{succ}(\mathsf{rnd})|\overline{\mathsf{Repeat}}]$ is to show the following (with $\mathsf{SC}()$ replaced with a random oracle $\$()$).

$$\begin{aligned}&\Pr[\mathsf{succ}(\mathsf{rnd})|\overline{\mathsf{Repeat}}] \\ &= \Pr\left[\mathsf{Hash}_\tau(C) \oplus R = \mathsf{tag} \,\middle|\, \overline{\mathsf{Repeat}} \wedge \bigwedge_{i=1}^{q} (\mathsf{Hash}_\tau(C^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)})\right] \qquad (9) \\ &\le \varepsilon.\end{aligned}$$

Strictly speaking, we should also be conditioning on the event $\bigwedge_{i=1}^{q} \left(Z^{(i)} = M^{(i)} \oplus C^{(i)}\right)$. But, conditioned on $\overline{\mathsf{Repeat}}$ and replacing $\mathsf{SC}()$ by $\$()$ results in both R and τ being independent of $Z^{(i)}$ for all i. So conditioning on this event will not change the probability.

There are two cases to consider. If N is different from each $N^{(i)}$, then (R, Z) is distributed uniformly and is independent of all previous $(R^{(i)}, Z^{(i)})$. Due to the independence, the conditional probability in (9) is equal to the unconditional probability $\Pr[\mathsf{Hash}_\tau(C) \oplus R = \mathsf{tag}]$ which due to the uniform distribution of R and its independence from τ, is equal to $1/2^n \le \varepsilon$.

So, suppose that $N = N^{(i)}$ for some i. Then (R, Z) is independent of $(R^{(j)}, Z^{(j)})$ for $j \ne i$ and $R = R^{(i)}$. The probability in (9) is equal to

$$\Pr\left[\mathsf{Hash}_\tau(C) \oplus R = \mathsf{tag} | \mathsf{Hash}_\tau(C^{(i)}) \oplus R = \mathsf{tag}^{(i)}\right]. \qquad (10)$$

By the restriction on the adversary, we have $(N, C, \mathsf{tag}) \ne (N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$. Since we are assuming $N = N^{(i)}$, it necessarily follows that $(C, \mathsf{tag}) \ne (C^{(i)}, \mathsf{tag}^{(i)})$. If $C = C^{(i)}$, then certainly $\mathsf{tag} \ne \mathsf{tag}^{(i)}$ and so the above probability is zero. So, let $C \ne C^{(i)}$. In this case, as in the proof of Theorem 1, the probability in (10) can be shown to be bounded above by ε. From this the result follows. □

Theorem 3. *Suppose the differential probabilities of* Hash *are bounded above by* ε*. Then*

$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}1}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + \varepsilon.$$

$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}3}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + \varepsilon.$$

Proof. We give the proof for AE-3.

Suppose as before that the queries are $(N^{(1)}, M^{(1)}), \ldots, (N^{(q)}, M^{(q)})$ and the corresponding responses are $(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q)}, \mathsf{tag}^{(q)})$. Further, suppose that the forgery attempt is (N, C, tag) with the restriction that this is not equal to $(N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$ for any $1 \leq i \leq q$.

Let $\mathsf{succ}(\mathsf{real})$ be the event that the forgery is accepted. Let SC_K be replaced by a random oracle. On distinct inputs, this oracle returns independent and uniform random strings of appropriate lengths. Let $\mathsf{succ}(\mathsf{rnd})$ be the event that the adversary is successful against the random oracle. Then, as before, it is standard to show the following.

$$\Pr[\mathsf{succ}(\mathsf{real})] \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + \Pr[\mathsf{succ}(\mathsf{rnd})].$$

We show

$$\begin{aligned}
&\Pr[\mathsf{succ}(\mathsf{rnd})] \\
&= \Pr\left[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag} \,\middle|\, \bigwedge_{i=1}^{q} \left(Z^{(i)} = M^{(i)} \oplus C^{(i)}, \mathsf{Hash}_\tau(M^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)}\right)\right] \qquad (11) \\
&\leq \varepsilon.
\end{aligned}$$

In this case, we cannot ignore the event $\bigwedge_{i=1}^{q} \left(Z^{(i)} = M^{(i)} \oplus C^{(i)}\right)$. This is because the message, rather than the ciphertext is hashed. If the N in the forgery attempt is equal to some $N^{(i)}$, then Z and $Z^{(i)}$ will have an overlap. Since the adversary provides C in the forgery attempt and M equals $C \oplus Z$, this M is not independent of $Z^{(i)}$.

There are two cases to consider. If N is different from each $N^{(i)}$, then (R, Z) is distributed uniformly and is independent of all previous $(R^{(i)}, Z^{(i)})$. Due to the independence, the conditional probability in (11) is equal to the unconditional probability $\Pr[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag}]$ which in turn, is equal to $1/2^n \leq \varepsilon$.

So, suppose that $N = N^{(i)}$ for some i. Then (R, Z) is independent of $(R^{(j)}, Z^{(j)})$ for $j \neq i$ and $R = R^{(i)}$. As a result, (11) reduces to the following.

$$\Pr\left[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag} \,\middle|\, \left(Z^{(i)} = M^{(i)} \oplus C^{(i)}, \mathsf{Hash}_\tau(M^{(i)}) \oplus R = \mathsf{tag}^{(i)}\right)\right]. \qquad (12)$$

Suppose that the length of M is at most the length of $M^{(i)}$, so that, Z is a prefix of $Z^{(i)}$. The other condition where $Z^{(i)}$ is a proper prefix of Z can be dealt with in a similar manner. Conditioning on the event $Z^{(i)} = M^{(i)} \oplus C^{(i)}$, fixes the value of Z. Let this value be z. Then $M = C \oplus z$. If Z is a proper prefix of $Z^{(i)}$, then certainly $M \neq M^{(i)}$; if $Z = Z^{(i)}$, then $z = M^{(i)} \oplus C^{(i)}$ and $M = C \oplus z = C \oplus C^{(i)} \oplus M^{(i)}$.

Since $N = N^{(i)}$ and for an allowed forgery, $(N, C, \mathsf{tag}) \neq (N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$, it necessarily follows that $(C, \mathsf{tag}) \neq (C^{(i)}, \mathsf{tag}^{(i)})$. If $C = C^{(i)}$, then $\mathsf{tag} \neq \mathsf{tag}^{(i)}$. From $M = C \oplus C^{(i)} \oplus M^{(i)}$ and $C = C^{(i)}$, it follows that $M = M^{(i)}$. Then, the probability in (12) is 0.

So, suppose that $C \neq C^{(i)}$ and then $C \oplus z = M \neq M^{(i)}$ (under the condition $Z^{(i)} = M^{(i)} \oplus C^{(i)}$). The probability in (12) is equal to

$$\Pr\left[\mathsf{Hash}_\tau(C \oplus z) \oplus R = \mathsf{tag} \,\middle|\, \mathsf{Hash}_\tau(M^{(i)}) \oplus R = \mathsf{tag}^{(i)}\right].$$

As in the proof of Theorem 1, this probability can be shown to be bounded above by ε. □

6 Handling Associated Data

None of the schemes given in the previous section can directly handle associated data. It is tempting to concatenate the header and then apply the hash function. But, this does not work. To see this, first consider AE-1 and AE-2 and suppose that $\mathsf{Hash}_\tau(C)$ is replaced by $\mathsf{Hash}_\tau(H||C)$. The authentication property of this scheme is easily attacked. The attacker $\mathcal{A}$ queries the oracle on the input $(N, H = h_1, M = m_1m_2)$ and receives $(C = c_1c_2, \mathsf{tag})$; here h_1, m_1, m_2, c_1 and c_2 are bits and tag is computed as $\mathsf{tag} = \mathsf{Hash}_\tau(h_1c_1c_2) \oplus R$, where $(R, Z) = \mathsf{SC}_K(N)$.

The forgery is $(N, H' = h_1c_1, C'_2 = c_2, \mathsf{tag})$. First note that this forgery is allowed, since $H' \neq H$ (and $C' \neq C$). We have $H'||C' = H||C$, but, this is allowed. Validity of the forgery is seen as follows. Since N is not changed, R remains unchanged and so $\mathsf{Hash}_\tau(H'||C') \oplus R = \mathsf{Hash}(H||C) \oplus R = \mathsf{tag}$.

AE-3 and AE-4 also do not support associated data. Again, the simple possibility of hashing the concatenation of the header and the message (i.e., $\mathsf{Hash}_\tau(H||M)$) instead of only the message (i.e., $\mathsf{Hash}_\tau(M)$) gives rise to an insecure scheme. The adversary $\mathcal{A}$ makes a query $(N, H' = 0, M' = 00)$ and obtains in response $(C' = c'_1c'_2, \mathsf{tag}')$. Here $\mathsf{tag}' = \mathsf{Hash}_\tau(H'||M') \oplus R = \mathsf{Hash}_\tau(000) \oplus R$. Let $(R, Z = z_1z_2)$ be the output of $\mathsf{SC}_K(N)$ and then $c_1 = z_1$; $c_2 = z_2$. The adversary outputs $(N, H = 00, c_1, \mathsf{tag}')$ as the forgery. Since the nonce is not changed, the pair (R, Z) remains the same, and so, the single bit message $M = m_1$ corresponding to the forgery is $m_1 = c_1 \oplus z_1 = c_1 \oplus c_1 = 0$. The digest computation for the forgery is then $\mathsf{Hash}_\tau(H||M) \oplus R = \mathsf{Hash}_\tau(000) \oplus R = \mathsf{tag}'$. So, the concatenate-and-hash strategy does not work.

The requirement is a hash function which accepts two inputs and has low collision and differential probabilities. More formally, we need a family $\{\mathsf{Hash}_\tau\}$ where for each τ, the function $\mathsf{Hash}_\tau(A, B)$ is an n-bit string. For such hash functions, we will overload notation and write $\mathsf{Hash}_\tau(A)$ as a shorthand for $\mathsf{Hash}_\tau(A, \lambda)$, where λ is the empty string. Existing hash functions can be modified to obtain such double-input hash functions as has been described in Section 3.

In Table 3, we present several ways to incorporate additional data into an AE scheme. Only the encryption algorithms are shown and the corresponding decryption algorithms can be easily constructed. Schemes described along the first column can be efficiently instantiated by Type-I hash functions, i.e., hash functions for which keys are short fixed-length bit strings. Schemes described along the second column can be efficiently instantiated by Type-II hash functions, i.e., hash functions for which keys are long. The first four schemes incorporate the header while hashing the ciphertext or the message. The later four schemes hash the header along with the nonce. Schemes 1, 2, 5 and 6 hash the ciphertext, whereas schemes 3, 4, 7 and 8 hash the message.

We provide explanations for Schemes 5 to 8. Since N is a nonce, the pair (H, N) is also a nonce, i.e., the freshness of N ensures the freshness of (H, N). Assuming that Hash has low collision probabilities, the outputs V are distinct and hence play the role of nonces. Conditioned on the event that the Vs are distinct, Scheme AEAD-5 reduces to Scheme AE-1; Scheme AEAD-6 reduces to Scheme AE-2; Scheme AEAD-7 reduces to Scheme AE-3; and Scheme AEAD-8 reduces to Scheme AE-4. The privacy and authentication bounds for these AEAD schemes are the same as that for the corresponding AE schemes plus an additive term of $\varepsilon \times \binom{q}{2}/2^n \leq \varepsilon \times q^2/2^{n+1}$ which accounts for the probability that there is a collision among the Vs.

The AEAD Schemes 1 to 4 are almost the same as AE schemes 1 to 4, except for the incorporation of the header information H as the first argument of the hash function. Under the assumption that the double-input hash function has low collision and differential probabilities, the analysis of these AEAD schemes is exactly the same as that of the corresponding AE schemes and hence, have the same security bounds.

Table 3. AEAD schemes.

AEAD-1.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,C) \oplus R$.	AEAD-2.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $\tau = \mathsf{SC}_K(K')$; $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,C) \oplus R$.
AEAD-3.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,M) \oplus R$.	AEAD-4.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $\tau = \mathsf{SC}_K(K')$; $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,M) \oplus R$.
AEAD-5.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $V = \mathsf{Hash}_\tau(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$.	AEAD-6.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $(\tau_1,\tau_2) = \mathsf{SC}_K(K')$; $V = \mathsf{Hash}_{\tau_1}(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_{\tau_2}(C) \oplus R$.
AEAD-7.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $V = \mathsf{Hash}_\tau(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$.	AEAD-8.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $(\tau_1,\tau_2) = \mathsf{SC}_K(K')$; $V = \mathsf{Hash}_{\tau_1}(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_{\tau_2}(M) \oplus R$.

GCM. The NIST standard GCM [20] can be seen as a construction following the scheme AEAD-1. A top-level view of GCM computation is as follows.

$\mathsf{GCM}_{K,\tau}(N,H,M)$
$V = f_\tau(N)$;
$(R,Z) = \mathsf{SC}_K(V)$;
$C = M \oplus Z$;
$\mathsf{tag} = \mathsf{GHASH}_\tau(H,C) \oplus R$.

The function $\mathsf{len}(A)$ applied to a string A returns an $n/2$-bit string containing the length of A. There are several points to be noted about the top-level specification. It uses an underlying n-bit block cipher E_K; the key K in GCM is the secret key of this block cipher.

The hash key τ is obtained as $\tau = E_K(0^n)$. The nonce N in GCM can be of arbitrary length. An n-bit value V is produced by applying f_τ to N and the function f_τ is defined as follows: if $\mathsf{len}(N) = n - 32$, then return $N||0^{31}||1$; otherwise, return $\mathsf{GHASH}_\tau(N)$.

The function SC_K is defined in the following manner.

$$\mathsf{SC}_K(V) = (E_K(V_0), E_K(V_1), \ldots).$$

$V_0 = V$ and for $i \geq 1$, V_i is defined from V_{i-1} by incrementing the last 32 bits of V_{i-1}. The block cipher E_K is used in a counter mode of operation and can be seen as a stream cipher supporting an IV.

Defined in this manner, SC, however, is not a PRF. To see this let V be an n-bit string and V' be obtained from V by incrementing the last 32-bits of V. Then the first block of $\mathsf{SC}_K(V')$ equals the second block of $\mathsf{SC}_K(V)$ and so the outputs can be easily distinguished from random strings.

Even though SC is not a PRF, the GCM construction is secure. This is because the input V to SC is essentially obtained as the output of a hash function with low differential probabilities.

As a result, for two inputs $V^{(s)}$ and $V^{(t)}$ to SC, the blocks $V_i^{(s)}$ and $V_j^{(t)}$ are equal with very low probability. This probability is upper bounded by the maximum differential probability of the hash function. Conditioned on the event that all the input blocks are distinct and under the assumption that E_K is a pseudo-random permutation, the outputs of E_K on these blocks will appear to be random to a computationally bounded adversary.

GHASH_τ is a double-input hash function and is defined in the following manner. Given two strings A and B, write A as $A_1, \ldots, A_{r_1-1}, A_{r_1}$ and B as $B_1, \ldots, B_{r_2-1}, B_{r_2}$ where the blocks are n-bit blocks except for the last blocks A_{r_1} and B_{r_2} which are possibly partial blocks. Let k_1 and k_2 be the minimum non-negative integers such that $A_{r_1}||0^{k_1}$ and $A_{r_2}||0^{k_2}$ are n-bit blocks.

Then $\mathsf{GHASH}_\tau(A, B)$ is defined as follows.

$$\mathsf{GHASH}_\tau(A, B) = \mathsf{Poly}_\tau(A_1, \ldots, A_{r_1-1}, A_{r_1}||0^{k_1}, B_1, \ldots, B_{r_2-1}, B_{r_2}||0^{k_2}, \mathsf{len}(A)||\mathsf{len}(B)).$$

Here Poly_τ is the usual polynomial hash function over $GF(2^n)$, i.e.,

$$\mathsf{Poly}_\tau(X_1, \ldots, X_k) = X_1\tau^k \oplus X_2\tau^{k-1} \oplus \cdots \oplus X_k\tau.$$

The definition of the double-input hash function $\mathsf{GHASH}_\tau(A, B)$ is different from the polynomial-based double-input hash function definition that we described in Section 3. In our definition, only the length of the first input is required to be concatenated.

7 Deterministic Authenticated Encryption

This is an authenticated encryption (with associated data) scheme which does not use a nonce. Syntactically, such a scheme is almost the same as that of an AEAD scheme.

It consists of two algorithms $\mathsf{DAEAD.Encrypt}_K(H, M)$ and $\mathsf{DAEAD.Decrypt}_K(H, C, \mathsf{tag})$, where K is from a suitable key space. The encryption algorithm produces (C, tag) as output and the decryption algorithm either rejects its input (by producing the symbol $\perp$) or produces a message. Soundness is ensured by the following condition.

$$\mathsf{DAE.Decrypt}_K(H, \mathsf{DAE.Encrypt}_K(H, M)) = M.$$

This is clearly a deterministic algorithm and invoking the encryption algorithm twice on the same message will return the same output.

Security of DAEAD schemes are defined in much the same way as that for AE schemes. There are two parts – privacy and authentication. (See [27] for an approach where the two parts can be merged into an equivalent single security model.) Since the scheme is deterministic, the adversary is not allowed to repeat a query to the encryption oracle. For authentication, a forgery attempt is a triplet (H, C, tag) where (C, tag) is not equal to the output of any possible previous query (H, M) to the encryption oracle. As in the case of AEAD, we denote by $\mathsf{Adv}_{\mathsf{D}}^{\text{daead-priv}}(t, q, \sigma)$ (resp. $\mathsf{Adv}_{\mathsf{D}}^{\text{daead-auth}}(t, q, \sigma)$) the maximum advantage of any adversary in breaking the privacy (resp. the authenticity) of a DAEAD scheme D, where the maximum is over all adversaries running in time t, making q queries and having query complexity σ (i.e., sending a total of σ bits in all its queries).

A DAE scheme is a special case of a DAEAD scheme where the header is the empty string for all messages. Conceptually, a DAE scheme is not much different from an AE scheme. (Similarly, for DAEAD and AEAD schemes.) This can be seen by considering the nonce-message pair of an AE scheme to be a message of a DAE scheme. In the security model of an AE scheme, the nonce

cannot be repeated, while in a DAE scheme, the message cannot be repeated. So, considering the nonce to be a part of the message ensures that messages are not repeated.

This, however, causes a difference in how the two primitives are constructed. While it is true that messages in a DAE scheme are distinct, it is not necessarily true that there is a specific location of n bits which have distinct values for different messages. Rather the entire message needs to be considered as a nonce. This makes the construction of DAE schemes significantly more inefficient compared to an AE scheme.

In broad terms, a DAE scheme can always be used as an AE scheme: simply consider the pair (N, M) as the message in the DAE scheme. But, the converse is not true, i.e., an AE scheme cannot be used as a DAE scheme.

Table 4. DAEAD scheme.

$\mathsf{DAEAD}\text{-}1_{K,\tau}.\mathsf{Encrypt}(H, M)$	$\mathsf{DAEAD}\text{-}1_{K,\tau}.\mathsf{Decrypt}(H, C, \mathsf{tag})$
$V = \mathsf{Hash}_\tau(H, M);$	$Z = \mathsf{SC}_K(\mathsf{tag});$
$\mathsf{tag} = \mathsf{SC}_K(V);$	$M = C \oplus Z;$
$Z = \mathsf{SC}_K(\mathsf{tag});$	$V = \mathsf{Hash}_\tau(H, M);$
$C = M \oplus Z;$	$\mathsf{ttag} = \mathsf{SC}_K(V);$
return (C, tag).	if (ttag=tag) then return M;
	else return $\perp$.

In Table 4, we present a DAEAD scheme. The header is assumed to be a bit string and this requires a double-input hash function Hash. (In general if the header is an m-tuple of strings, then the requirement is a hash function which takes $(m + 1)$ inputs. Such hash functions can be constructed as discussed in Section 3.1.) The scheme in Table 4 is suitable for Type-I hash functions, i.e., hash functions where the key τ is a short fixed length string. Extension of the scheme to Type-II hash functions is straightforward. Let K' be another independent and uniform random n-bit string and produce τ as $\tau = \mathsf{SC}_K(K')$.

The basic idea of the construction is simple and is based on the SIV construction in [27]. Intuitively, messages are considered to be nonces. If the pairs (H, M) are distinct, then the outputs V of the hash function are also distinct (under the assumption that the hash function has low collision probabilities). Assuming SC to be a PRF and applying it on distinct values of V ensures that the different values of tag are independent and uniformly distributed. In particular, with high probability, these values are also distinct. Again applying SC on the different values of tag ensures that the values of Z are independent and uniformly distributed. Since C is obtained by XORing Z and M, the values of C are also independent and uniformly distributed. So, the different (C, tag) pairs are independent and uniformly distributed strings. From this the privacy of the scheme follows. Formalising this argument is routine and gives the following result.

Theorem 4.

$$\mathsf{Adv}^{\text{daead-priv}}_{\mathsf{DAEAD}}(t, q, \sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t, q, \sigma) + \frac{q^2\varepsilon}{2} + \frac{q^2}{2^{n+1}}.$$

We next consider the authentication property. Write $\mathbf{E}_{K,\tau}$ as a shorthand for the encryption function. Note that ttag computed in the decryption module is a function of (H, C, tag) and write $\mathsf{RGen}(H, C, \mathsf{tag})$ to denote this function, i.e., $\mathsf{ttag} = \mathsf{RGen}(H, C, \mathsf{tag})$.

Theorem 5.

$$\mathsf{Adv}^{\text{daead-auth}}_{\mathsf{DAEAD}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + q\varepsilon + \frac{1}{2^n}.$$

Proof. Suppose the queries to the encryption oracle are $(H^{(1)}, M^{(1)}), \ldots, (H^{(q)}, M^{(q)})$ and the corresponding outputs are $(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q)}, \mathsf{tag}^{(q)})$. Let the forgery attempt be (H, C, tag).

Before getting into the detailed analysis, we make a few simple but important observations.

1. For fixed values of K and τ, the pair (C, tag) is functionally determined from (H, M). If (C', tag') is different from (C, tag), then the corresponding (H', M') is necessarily different from (H, M).
2. The forgery attempt (H, C, tag) implicitly defines a message M as $M = C \oplus \mathsf{SC}_K(\mathsf{tag})$.
3. By the restriction on the adversary, (H, C, tag) cannot be equal to $(H^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$ for any i.

If (C, tag) equals $(C^{(i)}, \mathsf{tag}^{(i)})$ for some i, then certainly $H \neq H^{(i)}$ and so, $(H, M) \neq (H^{(i)}, M^{(i)})$. If on the other hand, (C, tag) is not equal to any $(C^{(i)}, \mathsf{tag}^{(i)})$, then (H, M) is different from $(H^{(i)}, M^{(i)})$. So, in both cases, (H, M) is different from all previous $(H^{(i)}, M^{(i)})$ that have been queried to the encryption oracle.

Let Seen be the following event.

$$\mathbf{E}_{K,\tau}(H^{(1)}, M^{(1)}) = (C^{(1)}, \mathsf{tag}^{(1)}), \ldots, \mathbf{E}_{K,\tau}(H^{(1)}, M^{(1)}) = (C^{(q)}, \mathsf{tag}^{(q)}).$$

Since the sources of randomness are K and τ, Seen is equivalent to the event

$$\bigwedge_{i=1}^{q} \left(Z^{(i)} = M^{(i)} \oplus C^{(i)}\right) \wedge \bigwedge_{i=1}^{q} \left(\mathsf{tag}^{(i)} = \mathsf{SC}_K(\mathsf{Hash}_\tau(H^{(i)}, M^{(i)}))\right).$$

Note that $V^{(i)} = \mathsf{Hash}_\tau(H^{(i)}, M^{(i)})$. We show that $\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}]$ is small, which proves the authenticity of the scheme.

Since (H, M) is distinct from all previous $(H^{(i)}, M^{(i)})$, the probability that V is equal to one of the $V^{(i)}$s is at most $q\varepsilon$. Let $\mathsf{DistinctV}$ be the event that V is distinct from all the $V^{(i)}$s.

$$\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}] \leq q\varepsilon + \Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}].$$

In the next step, replace SC_K by a random oracle, i.e., the $\mathsf{tag}^{(i)}$s and the $Z^{(i)}$s are strings of appropriate lengths drawn independently and uniformly at random. Let this be change be denoted by $\mathsf{SC2rnd}$ and for notational simplicity, we will write the corresponding probability as $\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}, \mathsf{SC2rnd}]$. As is standard, it is possible to show the following.

$$\begin{aligned}&\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}] \\ &\leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma) + \Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}, \mathsf{SC2rnd}].\end{aligned}$$

Under the condition $\mathsf{DistinctV}$ and $\mathsf{SC2rnd}$, ttag is uniformly distributed and independent of all other random variables. As a result, the probability that this value is equal to tag is $1/2^n$.

Putting the inequalities together gives the result. □

8 Conclusion

In this paper, we considered the problem of utilising a stream cipher with IV to construct several important cryptographic primitives. Several constructions are given for MAC, AE, AEAD and DAE(AD) schemes. These constructions can be instantiated using known stream ciphers. Additionally, certain types of hash functions are required. We define suitable variants of well-known hash functions which are tailored for our applications. As a result of our work, a designer of practical cryptographic systems gets the option of using a wider variety of secure and efficient constructions for important cryptographic tasks than was previously known.

References

1. Document 1: Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3: 128-EEA3 & 128-EIA3 specification. `http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm`, 28 May, 2011.
2. eSTREAM, the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`.
3. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Okamoto [22], pages 531–545.
4. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Okamoto [22], pages 531–545.
5. Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Okamoto [22], pages 317–330.
6. Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.
7. Daniel J. Bernstein. Cycle counts for authenticated encryption. Workshop Record of SASC 2007: The State of the Art of Stream Ciphers, dated November 1, 2007, `http://cr.yp.to/papers.html#aecycles`.
8. Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.
9. Daniel J. Bernstein. Stronger security bounds for Wegman-Carter-Shoup authenticators. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2005.
10. John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.
11. Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
12. 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.
13. Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2002.
14. Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer, 2003.
15. Edgar N. Gilbert, F. Jessie MacWilliams, and Neil J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53:405–424, 1974.
16. 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.
17. Shai Halevi and Hugo Krawczyk. MMH: Software message authentication in the Gbit/second rates. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 1997.
18. 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.
19. Jonathan Katz and Moti Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC*, pages 245–254, 2000.

20. David A. McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.
21. Frédéric Muller. Differential attacks against the helix stream cipher. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2004.
22. Tatsuaki Okamoto, editor. *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*. Springer, 2000.
23. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.
24. 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.
25. 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.
26. Phillip Rogaway and Don Coppersmith. A software-optimised encryption algorithm. In Ross J. Anderson, editor, *FSE*, volume 809 of *Lecture Notes in Computer Science*, pages 56–63. Springer, 1993.
27. Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.
28. Palash Sarkar. A general mixing strategy for the ECB-Mix-ECB mode of operation. *Inf. Process. Lett.*, 109(2):121–123, 2008.
29. Palash Sarkar. A new multi-linear hash family. Cryptology ePrint Archive, Report 2008/216, 2008. `http://eprint.iacr.org/`.
30. Palash Sarkar. Pseudo-random functions and parallelizable modes of operations of a block cipher. *IEEE Transactions on Information Theory*, 56(8):4025–4037, 2010.
31. Palash Sarkar. A simple and generic construction of authenticated encryption with associated data. *ACM Trans. Inf. Syst. Secur.*, 13(4):33, 2010.
32. Palash Sarkar. A trade-off between collision probability and key size in universal hashing using polynomials. *Des. Codes Cryptography*, 58(3):271–278, 2011.
33. Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.
34. Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.