On Modes of Operations of a Block Cipher for Authentication and Authenticated Encryption

Debrup Chakraborty[1] and Palash Sarkar[2]

[1] Computer Science Department
CINVESTAV-IPN
Mexico, D.F., 07360, Mexico
email: debrup@cs.cinvestav.mx
[2] Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
email: palash@isical.ac.in

Abstract. This work deals with the various requirements of encryption and authentication in cryptographic applications. The approach is to construct suitable modes of operations of a block cipher to achieve the relevant goals. A variety of schemes suitable for specific applications are presented. While none of the schemes are built completely from scratch, there is a common unifying framework which connects them. All the schemes described have been implemented and the implementation details are publicly available. Performance figures are presented when the block cipher is the AES and the Intel AES-NI instructions are used. These figures suggest that the constructions presented here compare well with previous works such as the famous OCB mode of operation. In terms of features, the constructions provide several new offerings which are not present in earlier works. This work significantly widens the range of choices of an actual designer of cryptographic system.
Keywords: authentication, authenticated encryption, authenticated encryption with associated data, deterministic authenticated encryption with associated data, Galois field masking, block cipher.

1 Introduction

A block cipher maps a fixed length binary string to a string of the same length under the influence of a secret key. Formally, it is a map $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, where for each $K \in \mathcal{K}$, the function $E_K(\cdot) \overset{\Delta}{=} E(K,\cdot)$ is a bijection from $\{0,1\}^n$ to itself. A block cipher is a fundamental primitive in cryptography and is a major building block of several important cryptographic functionalities. There is a long history of research in the design and analysis of block ciphers. Currently, the most popular block cipher is the advanced encryption standard (AES) which has been standardised by the NIST of USA [18, 57].

Different cryptographic applications have varying security requirements. Further, the strings to be processed by such applications usually have varying lengths. Consequently, a block cipher has to be suitably tailored to handle such strings and also to attain the specific security goals. Methods for doing this are called modes of operations of a block cipher.

Below we describe several security goals that can arise in a cryptographic application.

Privacy: This is a basic goal whereby a secret transformation is applied to a given message so that the output of the transformation, called the ciphertext, does not reveal the message. The original message can be recovered by applying the inverse of the secret transformation to the ciphertext.

Authentication: For certain applications, the goal is different from that of achieving privacy of the message. Rather, it is to ensure that if some modification is made to the message during transmission, then it will be detected at the receiving end. As a result, the receiver will be able to ascertain the authenticity of the message along with the authenticity of the sender. One way of achieving this is to apply a secret transformation to the message to generate a tag and then transmit the tag along with the message. The secret transformation is available at the receiving end and a tag can be regenerated from the received message and compared to the received tag to determine the authenticity of the transmission.

Authenticated encryption (AE): In most cases, the requirement is to both protect the privacy of the message and to ensure authenticity. A method for simultaneously achieving both these goals is called authenticated encryption.

Authenticated encryption with associated data (AEAD): Often, along with the message, there is an additional information called the header (or associated data). The header needs to be authenticated, but, should not be encrypted as this may cause routing problems. The task of encrypting the message and authenticating both the header and the message is called authenticated encryption with associated data.

Deterministic authenticated encryption with associated data (DAEAD): AE(AD) constructions use a nonce, which is a quantity that is distinct for every message. If the nonce is reused, then security is lost. Deterministic authenticated encryption does away with the nonce. Only the message is processed using a secret (random) key to produce the ciphertext. An extension of this functionality allows the authentication of associated data and the message.

The on-going CAESAR project [14] has been launched with the aim of creating a portfolio of symmetric key algorithms for achieving a range of cryptographic functionalities including those mentioned above.

1.1 Contributions

This work provides a number of modes of operations of a block cipher for achieving the different functionalities mentioned above. The schemes are obtained by building on and refining existing schemes in the literature. None of the schemes reported here are built totally from scratch. There is, however, a common unifying theme which connects them.

The core of our approach to the different modes of operations is the notion of a pseudo-random function (PRF). Informally, a function f from a (finite) domain $\mathcal{D}$ to a co-domain $\mathcal{R}$ is said to be a PRF if it is "indistinguishable" from a function chosen uniformly at random from the set of all functions from $\mathcal{D}$ to $\mathcal{R}$. Formalisation of the notion of indistinguishability requires some restriction on a possible distinguishing "adversary". One approach is to require the adversary to be computationally bounded. The second approach is information theoretic where the restriction on the adversary is the number of times it can query f and the total number of bits it sends in all the queries. This work will follow the second approach. There are standard methods to convert a proof of PRF in the information theoretic setting to the computational setting.

A PRF can be used to provide authentication. This is the connection that we exploit in our constructions and analyses of the various modes of operations. A short summary of the different constructions in this paper is given below.

Vector-input pseudo-random function (PRF): A new generic method is provided which converts a single-input PRF to a vector-input PRF. The construction is simple, generic and efficient. It compares well to previously known constructions.

Authentication and AE(AD) schemes: These schemes are obtained by small modifications of similar previous schemes in [53]. The modification consists of eliminating an extra block cipher call for single-block messages. Instead, the mask generation process is applied in the backward direction to handle the complications which arise.

All previous works on (nonce-based) AEAD schemes considered the header to be a single string. We provide the first constructions of AEAD schemes which can handle a vector of strings as the header.

Deterministic authenticated encryption with associated data: New schemes for DAE and DAEAD are obtained by combining the generic vector-input PRF with a variant of the counter mode introduced in [59].

All the schemes presented in this work are efficient and fully parallelisable. They are accompanied by usual provable security treatment leading to concrete security bounds. A key feature of the designs is the use of masking techniques based on linear functions over binary characteristic fields. Several concrete instantations of the masking function are described. The masking functions are easily reconfigurable and as a result one specific masking function does not need to be hard coded into the specifications. Reconfigurability provides additional flexibility in the implementation of a particular functionality. One advantage is that depending on the application, it is possible to choose the masking function to be targeted either for high-end Intel processors; or, for 8-bit, 16-bit or 32-bit microcontrollers such as those manufactured by Atmel [17] and TI [29].

We present implementation details and performance results of all the schemes when the underlying block cipher is the AES. Two separate implementations were made – a simple reference implementation and a fast implementation on modern Intel processors using AES-NI and other SIMD instructions. Each of the schemes have been implemented using the various masking methods. Results for the fast implementations show minor variations in the efficiencies as the masking method varies. On the whole, the performance data shows that the schemes reported in this work compare favourably with existing works in the literature. Both the reference and the fast implementations are publicly available from [16]. We take this opportunity to mention that we have not filed any IP claims on any of the constructions in the paper.

Finally, we would like to mention that our work provides a unified and comprehensive treatment of the major functionalities that are realised using block cipher modes of operations. Working through the constructions and the proofs reveal the underlying and unifying thread of ideas which form the basis of the different constructions.

The recent CAESAR [14] competition has been launched with the goal of identifying a suite of schemes for the combined task of encryption and authentication. A set of security goals has been specified which include AEAD and DAEAD. Presently about 50 submissions are under consideration and these will go through a multi-year evaluation stage. The scope of CAESAR is broader than merely that of a mode of operation of a block cipher. Each submission has to be a complete cipher. Such a cipher can be a mode of operation along with a fully specified block cipher, or, it could be designed following other approaches.

Our work is along the more conventional lines of constructing modes of operations of a block cipher satisfying some of the established security goals in the literature. These constructions along with a fully specified block cipher (such as AES) could have formed possible submissions to CAESAR. Indeed, some of the CAESAR submissions are of these type. We, however, missed the CAESAR submission deadline by several months. Hopefully, due to its scope and completeness, the present work will be of scientific interest to the cryptographic community even though it could not be part of CAESAR. One category of users who may be interested in the modes of operations described in this work are various governmental agencies looking for a single platform for achieving different cryptographic functionalities using (a possibly proprietary) block cipher.

1.2 Previous and Related Works

Some of the basic works on authentication appear in [60, 25] and the works [56, 10, 9] develop a line of research on authentication schemes based on universal hash functions. The literature provides various modes of operations of a block cipher for achieving authentication. A long series of papers [6, 11, 41, 30, 31] has resulted in the CMAC [21] algorithm which has been standardised by NIST of USA. CMAC is based on the cipher-block chaining (CBC) mode of operation and is inherently sequential. Fully parallelisable modes of operations of a block cipher for authentication are known [12, 50, 15, 53]. We note that by no means the above-referenced papers are the only works on authentication. Fully referencing all such works is out of the scope of this paper. The previously reported schemes which are closest to the present work are those which appear in [50, 53]. Later we mention the exact relation of the scheme described here with that appearing in [53].

The first formal treatment of authenticated encryption was proposed in [7, 38]. Single-pass AE modes were proposed in [37, 26] and was quickly followed by the first version of the famous OCB mode [51]. Around the same time as these works, the notion of AEAD was first formally studied in [49] and generic construction methods were proposed. All later works on AE schemes also provided for handling of associated data and thus are in effect AEAD schemes. On the other hand, given an AE scheme and a collision-resistant hash function, it is possible to generically combine them to obtain an AEAD scheme [54].

Two later variants of the OCB mode have been proposed in [50, 40]. Currently, the version appearing in [40] is the "official" OCB mode of operation and is sometimes also denoted as OCB3. AE(AD) modes of operations having efficiency comparable to that of OCB were proposed in [15, 53]. Due to patent issues covering OCB and the modes in [37, 26, 50], the NIST of USA standardised less efficient AE schemes. One of these is GCM [44, 22] and the recent work [34] points out some security problems in GCM. The other NIST standardised scheme is CCM [61] which combines the counter mode of encryption with the sequential cipher block chaining (CBC) mode for authentication. Improved alternatives to CCM have been proposed as schemes EAX [8] and EAX-prime [58]. The scheme EAX-prime was, however, broken in [48] and improved authenticity bounds for EAX was given in [47].

Among the recent works on AEAD modes of operations are CLOC [32] and its variant SILC [33]. Both of these are based on the cipher-feedback mode (CFB) of operation and are essentially sequential algorithms. The claimed advantage is that these algorithms are suitable for lightweight devices such as those using 8-bit or 16-bit words. An interesting recent work is OTR [46] which describes a parallelisable AEAD scheme using only the encryption function of the block cipher.

DAE(AD) was first formally studied in [52]. The formal security definition was developed and constructions were provided. Improvements to the constructions were later provided in [36] and a scheme HBS was proposed. Unlike the work in [52] which uses only a block cipher, the work [36] also requires a finite field based polynomial hash. A later work which improves upon HBS is BTM [35]. A DAEAD mode of operation is secure against nonce-misuse. One approach to the construction of nonce-misuse resistant modes of operations is to start with an online cipher [5] and then modify it to obtain authentication. The constructions POET [43] and COPA [2] are of these type. Another interesting method used in [19] is to modify the encrypt-mix-encrypt [28] approach to construct a nonce-misuse resistant mode called ELmD.

A recent work [13] provides a comprehensive implementation of several modes of operations for both nonce-based AEAD and nonce-misuse resistant constructions. The implementation has been optimisied for new Haswell processor by Intel. Speed measurements are reported for both single messages as well as for multiple messages.

Most of the works in the literature on modes of operations for authenticated encryption use block ciphers as the building block. A systematic treatment of stream cipher modes of operations for combined encryption and authentication can be found in [55]. Another example of this type is HS1-SIV [39].

2 Notation and Preliminaries

The underlying primitive is a block cipher which is given by two functions

$$E, D : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n.$$

For a fixed key K, $E_K(\cdot) \stackrel{\Delta}{=} E(K, \cdot)$ and $D_K(\cdot) \stackrel{\Delta}{=} D(K, \cdot)$ are permutations of $\{0,1\}^n$ satisfying the following basic condition: for any $X \in \{0,1\}^n$, $D_K(E_K(X)) = X$. Some notation is given below.

1. In the descriptions of the modes of operations, we will use the notation π and π^{-1} to denote a secretly keyed permutation and its inverse. In practical terms, π can be instantiated as E_K and π^{-1} as D_K; or, π can be instantiated as D_K and π^{-1} as E_K. For the case when the underlying block cipher is AES, explicit recommendations will be given later.
2. Let $(X_1, \ldots, X_m)$ and $(Y_1, \ldots, Y_m)$ be two sequences of n-bits strings. Then $(X_1, \ldots, X_m) \oplus (Y_1, \ldots, Y_m)$ denotes the sequence $(X_1 \oplus Y_1, \ldots, X_m \oplus Y_m)$.
3. The notation $\mathsf{ecb}_\pi(X_1, \ldots, X_m)$ is defined as follows.

$$\mathsf{ecb}_\pi(X_1, \ldots, X_m) \stackrel{\Delta}{=} (\pi(X_1), \ldots, \pi(X_m)).$$

4. For an arbitrary binary string X, let $\mathsf{len}(X)$ denote the length of X.
5. If $\mathsf{len}(X) \geq r$, then the first r bits of X will be denoted by $\mathsf{First}_r(X)$.
6. For an integer ℓ such that $0 \leq \ell \leq 2^n - 1$, let $\mathsf{bin}_n(\ell)$ denote the n-bit binary representation of ℓ.

We give examples of $\mathsf{First}_r(X)$ and $\mathsf{bin}_n(\ell)$. We number the bits of X in increasing order from left to right. So, if $X = 11011100$, then we write $X = x_0x_1 \cdots x_7$ where, $x_0 = x_1 = x_3 = x_4 = x_5 = 1$ and $x_2 = x_6 = x_7 = 0$. $\mathsf{First}_3(X)$ is $x_0x_1x_2$ which is equal to 110. Let $n = 4$ and $\ell = 13$, then $\mathsf{bin}_4(13)$ is the 4-bit string $y_0y_1y_2y_3 = 1101$, i.e., $y_0 = y_1 = y_3 = 1$ and $y_2 = 0$.

Given a string X of arbitrary length, we define the function Format which describes how X is to be divided into n-bit blocks with possible padding at the end. This also defines the values of m and r from $\mathsf{len}(X)$ and n. Note that the map $X \mapsto \mathsf{Format}(X, n)$ for $n > 1$ is not an injective map. Non-injectivity arises due to strings of the following type: X is a string of length $i \times n$ (for some $i \geq 1$) ending with 10^j (for some $0 \leq j \leq n-2$) and X' is the prefix of X of length $i \times n - j - 1$. Then $\mathsf{Format}(X, n) = \mathsf{Format}(X', n)$. In fact, this is the only way in which X and X' can map to the same string under Format, a necessary condition for which is that n divides the length of one string but not the length of the other string. In our constructions, we tackle this condition using suitable masks.

For a non-empty set $\mathcal{X}$, define $\chi_q(\mathcal{X})$ to be

$$\chi_q(\mathcal{X}) = \{(x_1, \ldots, x_q) \in \mathcal{X}^q : x_i \neq x_j, 1 \leq i < j \leq q\}.$$

In other words, $\chi_q(\mathcal{X})$ consists of all $(x_1, \ldots, x_q)$ such that $x_1, \ldots, x_q$ are distinct elements of $\mathcal{X}$.

Table 1. Definition of $\mathsf{Format}(X, n)$ where X is an arbitrary length binary string and n is a positive integer. The values of the parameters r and m are defined from $\mathsf{len}(X)$ and n.

```
Format(X, n)
1.  if $\mathsf{len}(X) = 0$, then set $r = 0$, $m = 1$
2.  else write $\mathsf{len}(X) = (m-1)n + r$, where $1 \leq r \leq n$;
3.  if $r < n$, then set $\mathsf{pad}(X) = X||10^{n-r-1}$;
4.  else set $\mathsf{pad}(X) = X$;
5.  format $\mathsf{pad}(X)$ into $m$ blocks $X_1, \ldots, X_m$ each of length $n$;
return $(X_1, \ldots, X_m)$.
```

2.1 Pseudo-Random Functions

Formally, we will be studying functions from a finite non-empty set $\mathcal{X}$ to a finite non-empty set $\mathcal{Y}$. For example, $\mathcal{X}$ could be the set of all binary strings of lengths between 0 and 2^{64} and $\mathcal{Y}$ could be the set of all binary strings of length 128.

By a uniform random function ρ from $\mathcal{X}$ to $\mathcal{Y}$ we will mean a function chosen uniformly at random from the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$. A more convenient way to view ρ is the following. For distinct inputs $x_1, \ldots, x_q$, $q \geq 1$, the outputs $\rho(x_1), \ldots, \rho(x_q)$ are independent and uniformly distributed. If $\mathcal{X} = \mathcal{Y}$, then we can talk about a permutation π of $\mathcal{Y}$, which is a bijection $\pi : \mathcal{Y} \to \mathcal{Y}$. By a uniform random permutation, we will mean a permutation chosen uniformly at random from the set of all permutations of $\mathcal{Y}$.

The analysis of our constructions will follow the information theoretic approach. In the information theoretic approach, there is no bound on the computation time of an adversary. So, without loss of generality, we can consider the adversary to be a deterministic algorithm. An adversary interacts with an oracle and outputs a bit. The oracle takes as input an element of a set $\mathcal{X}$ and produces as output an element of a finite non-empty set $\mathcal{Y}$. The adversary $\mathcal{A}$ makes q queries to the oracle and then produces its output. Without loss of generality, we will make the assumption that the adversary never repeats a query.

The query complexity σ of an adversary is measured by the total number of bits that an adversary provides in all its queries. For n-bit block ciphers, it is more convenient to define the query complexity to be the total number of n-bit blocks that an adversary provides in all its queries.

Suppose that the oracle is instantiated twice by two random functions f and g both mapping $\mathcal{X}$ to $\mathcal{Y}$. Further, suppose that g is a uniform random function. Then the PRF-advantage of $\mathcal{A}$ in distinguishing f from a uniform random function is defined to be

$$\mathbf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) = \Pr[\mathcal{A}^f \to 1] - \Pr[\mathcal{A}^g \to 1]. \tag{1}$$

For positive integers q and $\sigma \geq q$, we define $\mathbf{Adv}_f^{\mathrm{prf}}(q, \sigma)$ to be the maximum advantage of any adversary which makes at most q distinct queries having query complexity σ. The quantity $\mathbf{Adv}_f^{\mathrm{prf}}(q, \sigma)$ is the PRF-advantage of f against any (q, σ)-bounded adversary.

2.2 Message Authentication Code

Let n be a positive integer and f be a random function from $\mathcal{X}$ to $\{0, 1\}^n$. Then the function f can be used to authenticate a message. The procedure is to apply f to the message to produce a tag and the message-tag pair is transmitted. At the receiving end, the same f is applied to the received message to recreate the tag. If the recreated tag is equal to the original tag, then the message-tag pair is accepted, otherwise it is rejected. The tag is called a message authentication code (MAC) and sometimes the function f is also called a MAC.

The authenticity of f is defined as follows. The adversary has access to f as an oracle and can submit queries in an adaptive manner. Finally, $\mathcal{A}$ outputs a "forged" pair (x, y) and is said to be successful if $f(x) = y$. The pair (x, y) must not be equal to any previous pair (x_i, y_i), where x_i was the i-th query and y_i was the corresponding response.

By $(x, y) \leftarrow \mathcal{A}^f$ we denote the event that $\mathcal{A}$ produces (x, y) as output after interacting with f. The advantage of $\mathcal{A}$ in breaking the authenticity of f is defined to be

$$\mathbf{Adv}_f^{\mathrm{auth}}(\mathcal{A}) = \Pr[f(x) = y]. \tag{2}$$

As in the case of PRF, we define $\mathbf{Adv}_f^{\mathrm{auth}}(q, \sigma)$ to be the maximum of $\mathbf{Adv}_f^{\mathrm{auth}}(\mathcal{A})$ taken over all adversaries making at most q queries and having query complexity at most σ. In this case, the query complexity also covers the forgery attempt.

Let f be a random function from $\mathcal{X}$ to $\{0,1\}^n$. The PRF-advantage of f and its security as an authentication function is related as follows.

$$\mathbf{Adv}_f^{\mathrm{auth}}(q, \sigma) \leq \frac{1}{2^n} + \mathbf{Adv}_f^{\mathrm{prf}}(q, \sigma). \tag{3}$$

Suppose that the output of f is truncated to t bits and denote the resulting function as t-f; further, suppose t-f is used for message authentication. Then we have

$$\mathbf{Adv}_{t\text{-}f}^{\mathrm{auth}}(q, \sigma) \leq 1/2^t + \mathbf{Adv}_f^{\mathrm{prf}}(q, \sigma). \tag{4}$$

Thus, to show the authentication property of f, it is sufficient to show that f is a good PRF. Equations (3) and (4) are known results and have been used in different ways in the literature. One way to prove these results can be found in [53].

2.3 Authenticated Encryption

Let $\mathcal{N}$ and $\mathcal{X}$ be finite non-empty sets of binary strings and let $\mathcal{F}_n[\mathcal{N}, \mathcal{X}]$ be the set of all functions $f : \mathcal{N} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$ such that if $f(N, X) = (Y, \mathsf{tag})$, then $\mathsf{len}(X) = \mathsf{len}(Y)$. Here $\mathcal{N}$ is called the set of nonces. Given an $f : \mathcal{N} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$, we define the following functions.

1. $f^{\mathrm{main}} : \mathcal{N} \times \mathcal{X} \to \mathcal{X}$ is defined to be $f^{\mathrm{main}}(N, X) = Y$ if $f(N, X) = (Y, \mathsf{tag})$ for some $\mathsf{tag} \in \{0,1\}^n$.
2. The function f is said to be an AE-function if for every $N \in \mathcal{N}$, $f_N^{\mathrm{main}}(\cdot) \stackrel{\Delta}{=} f^{\mathrm{main}}(N, \cdot)$ is a length preserving permutation. The invertibility of f_N^{main} ensures that decryption is possible, i.e., for a fixed N, it is possible to obtain X from Y.
3. For an AE-function f, $\widetilde{f} : \mathcal{N} \times \mathcal{X} \to \{0,1\}^n$ is defined to be $\widetilde{f}(N, Y) = \mathsf{tag}$ if $f(N, X) = (Y, \mathsf{tag})$ for some $X \in \mathcal{X}$. Due to the invertibility of f_N^{main}, it follows that $\widetilde{f}$ is well defined. The function $\widetilde{f}$ is said to be the authentication function associated with f.

An AE-function is required to satisfy two security properties – privacy and authenticity.

Let f be a random AE-function and f^* be a function distributed uniformly over $\mathcal{F}_n[\mathcal{N}, \mathcal{X}]$. Privacy is defined as indistinguishability from random strings. For defining privacy, an adversary $\mathcal{A}$ is assumed to have oracle access to f, i.e., for $1 \leq i \leq q$, $\mathcal{A}$ can adaptively query f on $(N^{(s)}, P^{(s)})$ and get back $(C^{(s)}, \mathsf{tag}^{(s)})$ in return. There is, however, a restriction on $\mathcal{A}$: the nonces of two different queries cannot be equal. Such an adversary is called nonce-respecting. Finally, $\mathcal{A}$ outputs a bit. As before, $\mathcal{A}^f \Rightarrow 1$ denotes the event that $\mathcal{A}$ produces 1 as output after interacting with the oracle f.

The advantage of $\mathcal{A}$ in breaking the privacy of f is defined to be

$$\mathbf{Adv}_f^{\text{priv}}(\mathcal{A}) = \Pr[\mathcal{A}^f \Rightarrow 1] - \Pr[\mathcal{A}^{f^*} \Rightarrow 1]. \tag{5}$$

By a (q, σ)-adversary we mean an adversary $\mathcal{A}$ which makes at most q queries and has query complexity at most σ. The resource bounded advantage $\mathbf{Adv}_f^{\text{priv}}(q, \sigma)$ is the maximum of $\mathbf{Adv}_f^{\text{priv}}(\mathcal{A})$ taken over all (q, σ)-adversaries $\mathcal{A}$.

We can think of privacy-advantage of f as the PRF-advantage of f with respect to nonce-respecting adversaries. We also define the privacy-advantage of f^{main} in a manner similar to that of (5).

The authenticity of an AE function is defined in the following manner. An adversary $\mathcal{A}$ has access to f; on a query (N, P), the corresponding output (C, tag) of f on (N, P) is returned to $\mathcal{A}$. The adversary adaptively makes $q-1$ queries $(N_1, P_1), \ldots, (N_{q-1}, P_{q-1})$ and obtains $(C_1, \mathsf{tag}_1), \ldots, (C_{q-1}, \mathsf{tag}_{q-1})$. It is assumed that the adversary does not repeat any query. At the end, $\mathcal{A}$ produces a forgery (N, C, tag) such that this triplet is not equal to any $(N_i, C_i, \mathsf{tag}_i)$ for $1 \leq i \leq q-1$. The adversary is deemed to be successful if there is an X such that $f(N, X) = (C, \mathsf{tag})$. Let succ denote this event. The advantage of $\mathcal{A}$, denoted as $\mathsf{Adv}_f^{\text{ae-auth}}(\mathcal{A})$ is defined to be $\Pr[\mathsf{succ}]$. The resource bounded advantage $\mathsf{Adv}_f^{\text{ae-auth}}(q, \sigma)$ is defined to be the maximum advantage of any adversary making q queries and having query complexity σ.

Consider a query (N, P) by the adversary. The response is (C, tag). The quantity C is the output of f^{main} on input (N, X) while tag is the output of $\widetilde{f}$ on (N, C). Suppose that f^{main} satisfies the privacy property. Then the output C appears to be random to the adversary. So, the sequence of queries provides the adversary with random strings $C_1, \ldots, C_{q-1}$ and their corresponding outputs $\mathsf{tag}_1, \ldots, \mathsf{tag}_{q-1}$ under $\widetilde{f}$. If the function $\widetilde{f}$ is a PRF, then the adversary gets no help from the queries in formulating its forgery. Consequently, a forgery will succeed with low probability. This intuition has been formalised and the the following result has been proved in [53].

Proposition 1. *Given an AE-function f, define another AE function h as follows: $h(N, X) = (Y, g(\mathsf{tag}))$, where $f(N, X) = (Y, \mathsf{tag})$ and g truncates tag to t bits. Then*

$$\mathbf{Adv}_h^{\text{ae-auth}}(q, \sigma) \leq \frac{1}{2^t} + \mathbf{Adv}_{f^{\text{main}}}^{\text{priv}}(q, \sigma) + \mathbf{Adv}_{\widetilde{f}}^{\text{prf}}(q, \sigma).$$

The above result shows that the authenticity of an AE function f follows from the privacy property of f^{main} and the PRF-property of $\widetilde{f}$. We note that the requirement of $\widetilde{f}$ to be a PRF is actually an overkill. Since, the adversary gets to see the outputs of $\widetilde{f}$ only on random inputs (under the assumption that f^{main} is a PRF), it is sufficient to have $\widetilde{f}$ to be a weak-PRF. On the other hand, in the context that we use Proposition 1, it does not appear that requiring $\widetilde{f}$ to be a weak-PRF will lead to more efficient constructions.

2.4 Authenticated Encryption with Associated Data

Let $\mathcal{H}$ be a finite non-empty set and let $\mathcal{N}$ and $\mathcal{X}$ be finite non-empty sets of binary strings. Let $\mathcal{F}_n[\mathcal{N}, \mathcal{H}, \mathcal{X}]$ be the set of all functions $f : \mathcal{N} \times \mathcal{H} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$ such that if $f(N, H, P) = (C, \mathsf{tag})$, then $\mathsf{len}(P) = \mathsf{len}(C)$. Here, $\mathcal{H}$ is the set of all possible headers and $\mathcal{N}$ is the set of all possible nonces. Note that $\mathcal{H}$ is simply defined to be a finite non-empty set with no particular structure. This allows a header to have a richer structure than a binary string. We will consider schemes where a header can be a vector of strings.

As in the case of authenticated encryption, given an $f : \mathcal{N} \times \mathcal{H} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$, we define the following notions.

1. For a function $f \in \mathcal{F}_n[\mathcal{N}, \mathcal{H}, \mathcal{X}]$ define $f^{\text{main}} : \mathcal{N} \times \mathcal{H} \times \mathcal{X} \to \mathcal{X}$ is defined to be $f^{\text{main}}(N, H, X) = Y$ if $f(N, H, X) = (Y, \mathsf{tag})$.
2. The function f is said to be an AEAD-function if for every $N \in \mathcal{N}$ and $H \in \mathcal{H}$, $f^{\text{main}}_{N,H}(\cdot) \overset{\Delta}{=} f^{\text{main}}(N, H, \cdot)$ is a length preserving permutation.

If we define $\mathcal{N}' = \mathcal{N} \times \mathcal{H}$ to be the set of nonces, then we go back to the formal framework for AE functions. In this case, we have the set of nonces $\mathcal{N}'$ to consist of possibly variable length strings. The security notions of privacy and authentication for $\mathcal{F}_n[\mathcal{N}', \mathcal{X}]$ are exactly the notions for $\mathcal{F}_n[\mathcal{N}, \mathcal{H}, \mathcal{X}]$ and coincide with the security notion of AEAD schemes introduced in [49]. We will use the notation $\mathbf{Adv}^{\text{aead-auth}}$ to denote the authentication advantage of an AEAD scheme.

In this case, the query complexity σ also takes into account the number of n-bit blocks formed from the headers provided as part of the different queries. We divide the query complexity into two parts σ_H and σ_P, where σ_H is the query complexity of the headers and σ_P is the query complexity of the nonces and the actual messages.

2.5 Deterministic Authenticated Encryption with Associated Data

Let $\mathcal{H}$ be a finite non-empty set and let $\mathcal{X}$ be a finite non-empty set of binary strings. Let $\mathcal{F}_n[\mathcal{H}, \mathcal{X}]$ be the set of all functions $f : \mathcal{H} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$ such that if $f(H, P) = (C, \mathsf{tag})$, then $\mathsf{len}(P) = \mathsf{len}(C)$. Here, $\mathcal{H}$ is the set of all possible headers. As before, given an $f : \mathcal{H} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$, we define the following notions.

1. $f^{\text{main}} : \mathcal{H} \times \mathcal{X} \to \mathcal{X}$ is defined to be $f^{\text{main}}(H, X) = Y$ if $f(H, X) = (Y, \mathsf{tag})$.
2. The function f is said to be a DAEAD-function if for every $H \in \mathcal{H}$, $f^{\text{main}}_H(\cdot) \overset{\Delta}{=} f^{\text{main}}(H, \cdot)$ is a length preserving permutation.

Security notions for a random DAEAD function are defined in a manner similar to that of AE(AD) schemes. The main difference is that there are no nonces in a DAEAD scheme. The only restriction is that an adversary for a DAEAD scheme is not allowed to repeat a query to the encryption oracle. Further, for authenticity, a forgery attempt is a triplet (H, C, tag) where (C, tag) is not equal to the output of any previous query $(H, \cdot)$ to the encryption oracle. The forgery (H, C, tag) is successful if there is some P such that the output of the DAEAD scheme on (H, P) is (C, tag).

Privacy of DAEAD schemes is defined as indistinguishability from random strings in exactly the same manner as that of AE(AD) schemes while authenticity is defined in terms of the probability of the adversary producing a successful forgery after interacting with the encryption oracle. The resource bounded advantages are defined in a manner similar to that of AEAD schemes. We will use the notation $\mathbf{Adv}^{\text{daead-auth}}$ to denote the authentication advantage of a DAEAD scheme.

3 Galois Field Masking

Let $\mathbb{F}_{2^n}$ be the finite field of 2^n elements. Suppose that n can be written as $n = n_1 \times n_2$. We will consider $\mathbb{F}_{2^n}$ to be an extension field, where the first extension is $\mathbb{F}_{2^{n_1}}$ which is of degree n_1 over $\mathbb{F}_2$ and the second extension is $\mathbb{F}_{2^n}$ which is of degree n_2 over $\mathbb{F}_{2^{n_1}}$.

Let $\rho(\alpha)$ be an *irreducible* polynomial of degree n_1 in the indeterminate α over $\mathbb{F}_2$, i.e., the coefficients of $\rho(\alpha)$ are bits. If $n_1 = n$, then we also require $\rho(\alpha)$ to be primitive. This polynomial is used to define $\mathbb{F}_{2^{n_1}}$.

Let $\mu(x)$ be a *primitive* polynomial of degree n_2 in the indeterminate x whose coefficients are from $\mathbb{F}_{2^{n_1}}$. This polynomial is used to define $\mathbb{F}_{2^n}$ over $\mathbb{F}_{2^{n_1}}$. An element of $\mathbb{F}_{2^n}$ can be written as a polynomial in x of degree less than n_2 where the coefficients of the polynomial are elements of $\mathbb{F}_{2^{n_1}}$. Alternatively, given an n-bit string β we can divide it into n_1-bit blocks, i.e., we can write $\beta = (b_0, b_1, \ldots, b_{n_2-1})$ where each b_i is an n_1-bit string. The string β will be taken to represent the following element of $\mathbb{F}_{2^n}$:

$$\beta = \beta(x) = b_0 \oplus b_1 x \oplus b_2 x^2 \oplus \cdots \oplus b_{n_2-1} x^{n_2-1}.$$

Here $b_0, \ldots, b_{n_2-1}$ are elements of $\mathbb{F}_{2^{n_1}}$ and each b_i is a polynomial $b_i(\alpha)$ of degree less than n_1 whose coefficients are bits. Addition of two elements $\beta(x)$ and $\gamma(x)$ is defined to be $\beta(x) \oplus \gamma(x)$. The product of $\beta(x)$ and $\gamma(x)$ is defined to be $\beta(x) \times \gamma(x) \bmod \mu(x)$ which is again a polynomial of degree less than n_2 with coefficients from $\mathbb{F}_{2^{n_1}}$.

Note.

1. For $n_1 < n$, we require $\mu(x)$ to be primitive, but, it is sufficient to have $\rho(\alpha)$ to be irreducible and for our application, we do not require the stronger condition of primitiveness for $\rho(\alpha)$.
2. When, however, $n_1 = n$, then $\mathbb{F}_{2^n}$ is essentially represented by the polynomial $\rho(\alpha)$ and we require this polynomial to be primitive.

Representation of $\mathbb{F}_{2^n}$ by $(\rho(\alpha), \mu(x))$: Let

$$\rho(\alpha) = \alpha^{n_1} \oplus s_{n_1-1}\alpha^{n_1-1} \oplus \cdots \oplus s_1\alpha \oplus s_0$$
$$\mu(x) = x^{n_2} \oplus t_{n_2-1}x^{n_2-1} \oplus \cdots \oplus t_1 x \oplus t_0.$$

Here $s_{n_1-1}, \ldots, s_1, s_0$ are elements of $\mathbb{F}_2$ while $t_{n_2-1}, \ldots, t_1, t_0$ are elements of $\mathbb{F}_{2^{n_1}}$. *The factorisation $n = n_1 \times n_2$ and the pair of polynomials $(\rho(\alpha), \mu(x))$ give a concrete representation of the field* $\mathbb{F}_{2^n}$. In the following discussion, we will be assuming such a representation. With this representation, we will also consider an n-bit string to be an element of $\mathbb{F}_{2^n}$ as discussed above. As a consequence, we will identify $\mathbb{F}_{2^n}$ with the set $\{0,1\}^n$. Examples of field defining polynomials for $n = 128$ are given in Table 2.

Table 2. Examples of suitable pairs of polynomials $(\rho(\alpha), \mu(x))$ for $n = 128$ are given below.

n_1	n_2	$n = n_1 \times n_2$	$\rho(\alpha)$	$\mu(x)$
128	1	128	$\alpha^{128} \oplus \alpha^7 \oplus \alpha^2 \oplus \alpha \oplus 1$	$x \oplus \alpha$
64	2	128	$\alpha^{64} \oplus \alpha^{63} \oplus \alpha^{29} \oplus \alpha^2 \oplus 1$	$x^2 \oplus x \oplus \alpha$
32	4	128	$\alpha^{32} \oplus \alpha^{27} \oplus \alpha^{25} \oplus \alpha^5 \oplus 1$	$x^4 \oplus x^3 \oplus x \oplus \alpha$
16	8	128	$\alpha^{16} \oplus \alpha^{10} \oplus \alpha^9 \oplus \alpha^6 \oplus 1$	$x^8 \oplus x^3 \oplus x \oplus \alpha$
8	16	128	$\alpha^8 \oplus \alpha^7 \oplus \alpha^3 \oplus \alpha^2 \oplus 1$	$x^{16} \oplus x^7 \oplus x \oplus \alpha$
1	128	128	$\alpha \oplus 1$	$x^{128} \oplus x^7 \oplus x^2 \oplus x \oplus 1$

3.1 (Word oriented) LFSR

We define a map

$$\psi : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n} \tag{6}$$

in the following manner. We will assume that $\mathbb{F}_{2^n}$ is represented by the pair of polynomials $(\rho(\alpha), \mu(x))$ and so we can think of any n-bit string as an element of $\mathbb{F}_{2^n}$.

Let γ be an n-bit string and suppose $\beta = \psi(\gamma)$ which is also an n-bit string. The strings γ and β can be respectively written as $(a_0, \ldots, a_{n_2-1})$ and $(b_0, \ldots, b_{n_2-1})$ where a_i and b_j are n_1-bit strings. We write

$$\psi(\gamma) = \psi(a_0, \ldots, a_{n_2-1}) = (b_0, \ldots, b_{n_2-1}) = \beta. \tag{7}$$

The definition of the b_j's from the a_i's is as follows.

$$\left.\begin{array}{ll} b_i = a_{i-1} & \text{if } 1 \leq i \leq n_2 - 1; \\ b_0 = t_0 a_{n_2-1} \oplus t_1 a_{n_2-2} \oplus \cdots \oplus t_{n_2-1} a_0. & \end{array}\right\} \tag{8}$$

Note that t_i's are the coefficients of $\mu(x)$ and are elements of $\mathbb{F}_{2^{n_1}}$. The map ψ is called a linear feedback shift register (LFSR) over $\mathbb{F}_{2^{n_1}}$. If $n_1 > 1$, then the elements of $\mathbb{F}_{2^{n_1}}$ can be considered to be words and the map is called a word-oriented LFSR. Using the theory of LFSRs [42], it can be proved that if $\mu(x)$ is a primitive polynomial, then for any non-zero γ, the sequence

$$\gamma = \psi^0(\gamma), \psi(\gamma), \psi^2(\gamma), \psi^3(\gamma), \ldots \tag{9}$$

has period $2^n - 1$, i.e., $\psi^{2^n-1}(\gamma) = \gamma$ and $\psi^i(\gamma)$ is not equal to γ for all $i < 2^n - 1$. A consequence of the period is the following relation.

$$\psi^{-i}(\gamma) = \psi^{2^n-1-i}(\gamma) \text{ for } 0 \leq i \leq 2^n - 2. \tag{10}$$

So, the elements $\psi^{-i}(\gamma)$ are elements in the sequence (9) and can be computed from γ by repeated application of ψ. When, i is small, however, it is much more efficient to compute $\psi^{-i}(\gamma)$ by the application of the map ψ^{-1} on γ.

Note.

1. For usual LFSR, we have $n_1 = 1$ and $n_2 = n$, $\rho(\alpha) = \alpha$ and $\mu(x)$ to be a primitive polynomial of degree n over $\mathbb{F}_2$.
2. Suppose $n_1 = n$, $n_2 = 1$, $\rho(\alpha)$ is a primitive polynomial over $\mathbb{F}_2$ and $\mu(x)$ is the constant polynomial α. Then any element of $\mathbb{F}_{2^n}$ can be written as a polynomial in α of degree at most $n - 1$. If $m(\alpha)$ is this polynomial, then $\psi(m(\alpha))$ is equal to $\alpha m(\alpha) \bmod \rho(\alpha)$. This has been called the powering-up map in [50] and can be seen as a special case of word-oriented LFSRs.
3. When $1 < n_2 < n$, for the map ψ to be efficient, we need to choose $\mu(x)$ carefully. We decided to fix the constant term of $\mu(x)$ to be α (i.e., the root of ρ) and the other coefficients are either 0 or 1. It turns out that there are many possible choices of $\mu(x)$ satisfying these conditions. Table 2 provides some examples.

The map ψ is invertible and the inverse can be computed as follows. Suppose as before that $\psi(a_0,\ldots,a_{n_2-1}) = (b_0,\ldots,b_{n_2-1})$ so that $(a_0,\ldots,a_{n_2-1}) = \psi^{-1}(b_0,\ldots,b_{n_2-1})$ and then the a_i's can be expressed in terms of the b_j's as follows.

$$\left.\begin{aligned} a_i &= b_{i+1} && \text{if } 0 \le i \le n_2 - 2; \\ a_{n_2-1} &= t_0^{-1}\left(b_0 \oplus t_1 a_{n_2-2} \oplus \cdots \oplus t_{n_2-1} a_0\right) \\ &= t_0^{-1}\left(b_0 \oplus t_1 b_{n_2-1} \oplus \cdots \oplus t_{n_2-1} b_1\right). \end{aligned}\right\} \tag{11}$$

The computation of this method depends on the efficiency of multiplying by t_0^{-1}. If $n_1 = 1$ and $n_2 = n$, then $t_0 = 1$ (since in this case $\mu(x)$ is a primtive polynomial over $\mathbb{F}_2$). In all other cases, by our choice of $\mu(x)$ we ensure that $t_0 = \alpha$. So, multiplying by t_0^{-1} corresponds to multiplying by α^{-1}.

We use ψ and ψ^{-1} to define certain masks. Since the period of ψ is $2^n - 1$, it is possible to interchange the roles of ψ and ψ^{-1} without affecting security.

3.2 The Masking Method

The modes of operations to be described subsequently will use masks. These masks are n-bit strings and are considered to be elements of $\mathbb{F}_{2^n}$ as represented by the pair of polynomials $(\rho(\alpha), \mu(x))$. Given an n-bit string γ and an integer i, we define $\Gamma_{\gamma,i}$ as follows.

$$\Gamma_{\gamma,i} \stackrel{\Delta}{=} \psi^i(\gamma) \tag{12}$$

where $\psi(\gamma)$ is given by (8). If $i = 0$, then $\psi^i(\gamma) = \gamma$; if $i > 0$, then $\psi^i(\gamma)$ can be computed iteratively as $\psi^i(\gamma) = \psi(\psi^{i-1}(\gamma))$; similarly, if $i < 0$, then $\psi^i(\gamma)$ can be computed iteratively as $\psi^i(\gamma) = \psi^{-1}(\psi^{i+1}(\gamma))$.

For the security proofs of the modes of operations to go through, certain properties of the masking function are required to hold. The following definition from [53] states these properties.

Definition 1. *Suppose $\psi : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is a linear function. We say that the function ψ is a proper masking function if it satisfies the following properties.*

1. *For any $\alpha \in \mathbb{F}_{2^n}$; any non-negative integer k with $0 \le k \le 2^n - 2$; and a uniform random $\beta \in \mathbb{F}_{2^n}$; $\Pr[\psi^k(\beta) = \alpha] = 1/2^n$.*
2. *For any $\alpha \in \mathbb{F}_{2^n}$; integers k_1, k_2 with $0 \le k_1 < k_2 \le 2^n - 2$; and a uniform random $\beta \in \mathbb{F}_{2^n}$; $\Pr[\psi^{k_1}(\beta) \oplus \psi^{k_2}(\beta) = \alpha] = 1/2^n$.*
3. *For any $\alpha \in \mathbb{F}_{2^n}$; integers k_1, k_2 with $0 \le k_1, k_2 \le 2^n - 2$; and uniform random $(\beta_1, \beta_2) \in \chi_2(\mathbb{F}_{2^n})$, $\Pr[\psi^{k_1}(\beta_1) \oplus \psi^{k_2}(\beta_2) = \alpha] = 1/(2^n - 1)$.*

The following proposition is based on results from [53].

Proposition 2. *The function ψ defined in (6) and (8) is a proper masking function, i.e., it satisfies Definition (1).*

3.3 Reconfigurability of the Masking Method

An important aspect of the above masking method is easy reconfigurability. To see this, suppose that $n_1 = 1$ and $n_2 = 128$. Then the only requirement on $\mu(x)$ is that it should be a primitive polynomial of degree 128 over $\mathbb{F}_2$. There are $\phi(2^{128} - 1)/n > 2^{119}$ such polynomials. Also, it is fairly easy to generate such a primitive polynomial using standard algorithms [45]. In terms of implementation, a polynomial of degree 128 over $\mathbb{F}_2$ can be represented by a 128-bit string. Call this string $\mathsf{PolyStr}$. For a specific implementation, $\mathsf{PolyStr}$ is fixed. Changing $\mathsf{PolyStr}$ to the string representation of another primitive polynomial is easy and gives rise to another specific instantiation. Similarly, when $n_1 = 128$ and $n_2 = 1$, we require $\rho(\alpha)$ to be a primitive polynomial of degree 128 and the 128-bit string $\mathsf{PolyStr}$ can be used to represent $\rho(\alpha)$.

For the more general case when $1 < n_1, n_2 < 128$, the polynomial $\rho(\alpha)$ can be represented using n_1 bits. By our choice of $\mu(x)$, the constant term is α and all other coefficients are either 0 or 1. So $\mu(x)$ is a monic polynomial of degree n_2 whose constant term is α and all other coefficients are either 0 or 1. As a result, $n_2 - 1$ bits are required to represent $\mu(x)$. The total number of bits required to represent both $\rho(\alpha)$ and $\mu(x)$ is $n_1 + n_2 - 1$. Changing these bits to represent another suitable pair of polynomials will give rise to a different instantiation.

We recommend that the values of the pair $(\rho(\alpha), \mu(x))$ should *not* be fixed as part of a standard. Instead a table of recommended values such as those in Table 2 should be provided. One justification for this would be the following. The efficiency of masking depends on the choice of the values of n_1 and n_2. For example, on processors having only 32-bit registers, it will be fastest to choose $n_1 = 32$ and $n_2 = 4$. Examples of processors with small word size are the Atmel AVR 8-bit and 32-bit microcontrollers [17] and the MSP430, MSP430X 16-bit microcontrollers [29]. The ability to suitably customise the masking method to extract the maximum speed from the target architecture will benefit designers.

4 Vector Input PRF

Let f be a PRF whose domain is the set of all binary strings of some maximum length. It has been pointed out in [52] that for certain applications, it is required to have a PRF which can take as input a tuple $(X_1, \ldots, X_k)$ where $k \geq 0$ and each X_i is a binary string. The S2V construction provided in [52] converts f to a vector-input PRF f^* as follows.

$$f^*(X_1, \ldots, X_k) = \begin{cases} f(1^n) & \text{if } k = 0; \\ f(\alpha^{k-1}Y_0 \oplus \alpha^{k-2}Y_1 \oplus \cdots \oplus Y_{k-1} \oplus_{\mathsf{end}} X_k) & \text{if } k > 0 \text{ and } |X_k| \geq n; \\ f(\alpha^k Y_0 \oplus \alpha^{k-1}Y_1 \oplus \cdots \oplus \alpha Y_{k-1} \oplus X_k || 10^{n-r-1}) & \text{if } k > 0 \text{ and } |X_k| = r < n. \end{cases} \quad (13)$$

where $Y_0 = f(\mathbf{0})$; $Y_i = f(X_i)$ for $i = 1, \ldots, k-1$; and $\oplus_{\mathsf{end}}$ is the operation of xoring to the last n bits of the second operand. The construction assumes that the output of f consists of binary strings of length n. The operation of multiplying by the powers of x is done over the finite field $\mathbb{F}_{2^n}$ which is represented using a primitive polynomial $\rho(\alpha)$ of degree n over $\mathbb{F}_2$. A restriction is that $k < n$. It has been shown in [52] that if f is a uniform random function, then $\mathsf{Adv}^{\text{prf}}_{f^*}(q, \sigma) \leq \sigma q / 2^n$.

The construction f^* is simple and efficient. The restriction $k < n$ is not an issue in practice. The construction is also generic though not completely so. The reason being that the output of f is an n-bit string. So, the construction would not work if the output of f could be strings of variable lengths. Again, for most practical applications this will not be an issue. The other minor drawback is the additional machinery of finite field arithmetic required over and above that of f. Though multiplication by x is very efficient, one may ask whether this can be done away with.

We provide a simple and generic description of a vector-input PRF from a single-input PRF. There is no restriction on f, no use of finite field arithmetic and the resulting information theoretic bound is better.

$$\overrightarrow{f}(X_1, \ldots, X_k) = f(\omega_0 || (f(\omega_1 || X_1) \oplus\!\!\!\rightarrow \cdots \oplus\!\!\!\rightarrow f(\omega_k || X_k))). \quad (14)$$

Here $\omega_0, \ldots, \omega_k$ are distinct, fixed w-bit words such that $2^w > k+1$. For binary strings of possibly different lengths, the binary operation $\oplus\!\!\!\rightarrow$ denotes the task of XORing the shorter string to the rightmost end of the longer string. When the outputs of f have a fixed length, we will use $\oplus$ instead of $\oplus\!\!\!\rightarrow$ to denote the task of XORing these outputs.

For practical purposes, it is sufficient to take $w = 8$, i.e., the ω_i's are distinct bytes. The value of k need not be fixed and for $k = 0$, the output is $f(\omega_0)$. Note that the case $k = 0$ is different from the case $k = 1$ and $|X_1| = 0$ where in the later case the output is $f(\omega_0||f(\omega_1))$.

The security of the construction can be argued in a simple manner and is given by the following result.

Theorem 1. *If f is a uniform random function, then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\overrightarrow{f}}(q, \sigma) \leq \frac{q(q-1)}{2^{\ell+1}}$$

where the adversary makes q queries $(X_1^{(1)}, \ldots, X_{k_1}^{(1)}), \ldots, (X_1^{(q)}, \ldots, X_{k_q}^{(q)})$ with
$\ell_i = \min\left(\mathsf{len}\left(f_1(X_1^{(i)})\right), \cdots, \mathsf{len}\left(f_{k_i}(X_{k_i}^{(i)})\right)\right)$; $f_i(X) \triangleq f(\omega_i||X)$; *and* $\ell = \min_{1 \leq i \leq q} \ell_i$.

Consequently, if the output of f consists only of strings of length n bits, then

$$\mathbf{Adv}^{\mathrm{prf}}_{\overrightarrow{f}}(q, \sigma) \leq \frac{q(q-1)}{2^{n+1}}.$$

Proof : Suppose the number of components in the ith query is k_i and the result of applying $\oplus\!\!\!\rightarrow$ to the outputs of $f_1, \ldots, f_{k_i}$ is the string Z_i. Since f is a uniform random function, then due to the input space separation, the f_i's are independent and uniform random functions. The length of Z_i is

$$\lambda_i = \max\left(\mathsf{len}\left(f_1(X_1^{(i)})\right), \cdots, \mathsf{len}\left(f_{k_i}(X_{k_i}^{(i)})\right)\right).$$

For $i \neq j$, we consider the probability that Z_i is equal to Z_j. If $\lambda_i \neq \lambda_j$, then this probability is clearly 0. So, suppose that $\lambda_i = \lambda_j$. Since the i-th and the j-th queries have to be distinct, the two queries must differ in some component, say $\imath$. The corresponding outputs of $f_\imath$ are then independent and uniformly distributed strings of appropriate lengths greater than or equal to ℓ. As a consequence, the rightmost ℓ bits of Z_i and Z_j are independent and uniformly distributed. So, the probability that the rightmost ℓ bits of Z_i and Z_j are equal is $1/2^\ell$ and the probability that Z_i equals Z_j is also at most $1/2^\ell$. Extending this argument, the probability that at least two of the Z's are equal is at most $q(q-1)/2^{\ell+1}$. Conditioned on the event that all the Z's are distinct, the outputs of f_0 are independent and uniformly distributed and hence provides no information to the adversary. Formalising this argument in a standard manner gives the desired result.

If the outputs of f all have the same length n, then $\ell = n$ and so the second part of the result follows. □

Note that the bound is better than the one for f^*. A downside of $\overrightarrow{f}$ with respect to f^* is that f is applied to one-byte longer strings. This will result in an (insignificant) loss of efficiency.

5 Overview of the Various Constructions

We provide constructions of several different kinds of primitives and for each kind, several constructions are described. The purpose of this section is to provide a high-level overview of the constructions which may help the reader in following the ensuing material.

The first construction is PAuth which is a PRF and hence can be used for authentication. Next, we tackle AE and AEAD schemes. The descriptions start with two algorithms $\mathsf{Forward1}$ and $\mathsf{Backward1}$; $\mathsf{Forward1}$ takes a nonce-message pair (N, P) to (C, tag) and $\mathsf{Backward1}$ takes (N, C) to (P, tag). The algorithm $\mathsf{Forward1}$ is used to define the encryption algorithm of an AE scheme $\mathsf{PAE1}$. The corresponding decryption algorithm algorithm is derived from $\mathsf{Backward1}$: given (N, C, tag), the decryption algorithm invokes $\mathsf{Backward1}$ on (N, C) to obtain (P, tag_1) and returns P if $\mathsf{tag} = \mathsf{tag}_1$, else returns $\perp$. The algorithm $\mathsf{Backward1}$ is also used to define a PRF $\mathsf{PAuth1}$ (and the associated authentication function $\widetilde{\mathsf{PAE}}$) which given $N||C$ applies $\mathsf{Backward1}$ to obtain (P, tag) and returns tag.

$\mathsf{PAE1}$ and $\mathsf{PAuth1}$ are combined to obtain an AEAD scheme $\mathsf{PAEAD1}$. Given nonce-header-message triplet (N, H, P), if H is non-empty $\mathsf{PAuth1}$ is applied to H to obtain tag_2 and $\mathsf{PAE1}$ is applied to (N, P) to obtain (C, tag_1) and $(C, \mathsf{tag}_1 \oplus \mathsf{tag}_2)$ is returned. (In case of empty header tag_2 is set to 0^n.) The scheme $\mathsf{PAEAD1}$ is naturally extended to $\overrightarrow{\mathsf{PAEAD1}}$ where the header can be a vector of strings. This variant arises by using $\overrightarrow{\mathsf{PAuth1}}$ to process the complex header.

In the schemes $\mathsf{PAE1}$, $\mathsf{PAEAD1}$ and $\overrightarrow{\mathsf{PAEAD1}}$, the encryption algorithm uses both π and π^{-1}, whereas the decryption algorithm uses only π^{-1}. We provide a "dual" of this strategy by defining schemes $\mathsf{PAE2}$, $\mathsf{PAEAD2}$ and $\overrightarrow{\mathsf{PAEAD2}}$ where the encryption algorithm uses only π and the decryption algorithm uses both π and π^{-1}.

The last primitive that we consider is deterministic authenticated encryption with associated data (DAE and DAEAD). The basic idea for these schemes is based on the SIV construction in [52] though the details are different. The scheme DAE takes as input a message P and applies PAuth to produce a tag. The tag is used as an IV in a counter type mode which is applied to P to obtain C. Finally (C, tag) is returned. The counter-type mode that we use is from [59]. Extension of DAE to DAEAD is done as follows. Given a vector of strings $(H_1, \ldots, H_k)$ as a header and a message P, $\overrightarrow{\mathsf{PAuth}}$ is applied to $(H_1, \ldots, H_k, P)$ to produce the tag. This tag is then used as in the case of DAE to produce C.

The following sections provide the details of the constructions mentioned above.

6 Authentication

Table 3 describes the construction of PAuth. The domain of PAuth consists of binary strings x, where $0 \leq \mathsf{len}(P) \leq 2^n - 8$. Then the maximum value of m returned by $\mathsf{Format}(x, n)$ is $\lceil (2^n - 8)/n \rceil$. The bound $2^n - 8$ on the length of any input to PAuth is due to the following reason. The masking function ψ has period $2^n - 1$ and it is used in both the forward and the backward directions. In the backward direction, only ψ^{-1} and ψ^{-2} are used. Having at most $\lceil (2^n - 8)/n \rceil$ blocks will ensure that there is no overlap between the forward and the backward applications of ψ. We note, however, that the security guarantee of PAuth does not hold for strings of lengths up to the above mentioned bound. This guarantee is given by Theorem 2 proved later. During actual use, it is essential to use PAuth in a manner such that the bound given by Theorem 2 is meaningful.

The output of PAuth is an n-bit string. This can be truncated to obtain a t-bit tag. We perform the security analysis of PAuth using n-bit tags. Using Proposition 4 it is possible to obtain the security of t-PAuth as an authentication function for any $t \leq n$.

The construction PAuth is very similar to the construction iPMAC given in [53]. The only difference is in the way the case of $m = 1$ is handled. In [53], for $m = 1$, sum is $\delta \oplus P_1$ or $\delta \oplus P_1 \oplus \Gamma_{\gamma,1}$ according as $r = n$ or $r < n$, where $\delta = \pi(\gamma)$. The construction PAuth avoids using δ which requires an additional application of π. Instead, the masks $\Gamma_{\gamma,-1}$ and $\Gamma_{\gamma,-2}$ are used. This eliminates an extra block cipher call when $m = 1$.

Table 3. Description of PAuth. The values of m and r are computed from $\mathsf{len}(P)$ and n as described in Format.

```
PAuth_{π,fStr}(P):
1.  $(P_1, \ldots, P_m) = \mathsf{Format}(P, n)$;
2.  $\gamma = \pi(\mathsf{fStr})$;
3.  if $(m = 1$ and $r < n)$ $\mathsf{sum} = P_1 \oplus \Gamma_{\gamma,-1}$;
4.  if $(m = 1$ and $r = n)$ $\mathsf{sum} = P_1 \oplus \Gamma_{\gamma,-2}$;
5.  if $(m > 1)$
6.      $(C_1, \ldots, C_{m-1})$
            $= \mathsf{ecb}_\pi(P_1 \oplus \Gamma_{\gamma,1}, \ldots, P_{m-1} \oplus \Gamma_{\gamma,m-1})$;
7.      $\mathsf{sum} = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m$;
8.      if $(r < n)$ then $\mathsf{sum} = \mathsf{sum} \oplus \Gamma_{\gamma,m}$;
9.  end if;
10. $\mathsf{tag} = \pi(\mathsf{sum})$;
return tag.
```

For an m-block message, PAuth makes one block cipher call on the string fStr and then makes a total of m calls to generate the output tag. The call on fStr can be done once per session and then a total of m calls are required for an m-block message.

The analysis of PAuth is very similar to that of iPMAC and yields the following result. We provide details of the analysis for PAuth in Section B. The bound for $\overrightarrow{\mathsf{PAuth}}$ follows from the bound of PAuth and Theorem 1.

Theorem 2. *Let q and $\sigma \geq q$ be positive integers. Then*

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{PAuth}}(q, \sigma) \leq \frac{\sigma(7q+2)}{2^n};$$
$$\mathsf{Adv}^{\mathrm{prf}}_{\overrightarrow{\mathsf{PAuth}}}(q, \sigma) \leq \frac{q(q-1)}{2^{n+1}} + \frac{\sigma(7q+2)}{2^n}.$$

7 Authenticated Encryption

Consider the algorithms Forward1 and Backward1 shown in Table 4. The input to Forward1 is a pair (N, P), where N is an n-bit string and P is a non-empty binary string having maximum length $2^n - 8$. The reason for the restriction on the length is similar to that for PAuth.

We do not define the encryption of the empty string for the following reason. The output of the encryption algorithm on input (N, P) is (C, tag) where C is of the same length as P. So, if P is the empty string, then C is also necessarily the empty string and hence, in this case, P is revealed by C. Such a scheme cannot satisfy the privacy property.

The output of Forward1 is (C, tag), where C is of the same length as P and tag is an n-bit string. The input of Backward1 is a pair (N, C), where N is an n-bit string and C is a binary string of maximum length $2^n - 8$. The output of Backward1 is (P, tag) where P is of the same length as that of C and tag is an n-bit string. In both Forward1 and Backward1, it is possible to truncate tag to a t-bit string. This is not shown in the description.

The authenticated encryption scheme PAE1 is defined from these and is given in Table 5. From Backward1, we also define the following function.

$$\mathsf{PAuth1}_{\pi,\mathsf{fStr}}(P): \ (C, \mathsf{tag}) = \mathsf{Backward1}_\pi(\mathsf{fStr}, P); \text{ return } \mathsf{tag}. \tag{15}$$

Note that PAuth1 is different from PAuth given in Table 3. The two functions are related though; and we later argue that the security analysis of PAuth1 follows easily from that of PAuth.

The call to Backward1 is made as part of PAuth1 which returns (C, tag). The output of PAuth1 consists only of tag. It can be verified from the description of Backward1 that the quantity tag does not depend on the last block of C. As a result, for an h-block message, the total number of block cipher calls made to generate the tag by PAuth1 is $h + 1$. One of these calls is made on fStr and can be made once per session. As a result, the number of block cipher calls made by PAuth1 is the same as that made by PAuth.

Table 6 describes the scheme PAEAD1. It makes use of PAuth1. The header H can be a binary string (possibly empty) of length at most $2^n - 8$. Table 7 provides an AEAD scheme when the header is a vector of strings. The number of components in the header vector can vary and can even be zero. Each component of the header vector is a binary string (possibly empty) of maximum length $2^n - 8$. We would like to highlight the following points.

1. The construction PAEAD1 with the empty header is same as the PAE1 scheme. So, in practice, there is no need to have separate implementations of PAE1 and PAEAD1.
2. The construction $\overrightarrow{\mathsf{PAEAD1}}$ where the header consists of a single string is *not* the same as the PAEAD1 construction. This is due to the fact that $\overrightarrow{f}$ on a single input does not coincide with f.

Table 4. Algorithms Forward1 and Backward1. The values of m and r are defined by the calls to Format.

```
Forward1_π(N, P):
4.  $(P_1, \ldots, P_m) = \mathsf{Format}(P, n)$;
5.  $\gamma = \pi^{-1}(N)$;
6.  if ($m = 1$ and $r < n$) then
7.    $\mathsf{tmp} = \pi^{-1}(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,-1})$;
8.    $C_1 = \mathsf{First}_r(P_1 \oplus \mathsf{tmp})$; $T_1 = C_1 || (10^{n-r-1})$;
9.    $\mathsf{sum} = T_1 \oplus \Gamma_{\gamma,-2}$;
10. end if;
11. if ($m = 1$ and $r = n$) then
12.   $C_1 = \pi(P_1 \oplus \Gamma_{\gamma,-1}) \oplus \Gamma_{\gamma,-1}$;
13.   $\mathsf{sum} = C_1 \oplus \Gamma_{\gamma,-3}$;
14. end if;
15. if $m > 1$ then
16.   $(C_1, \ldots, C_{m-1})$
          $= \mathsf{ecb}_\pi(P_1 \oplus \Gamma_{\gamma,1}, \ldots, P_{m-1} \oplus \Gamma_{\gamma,m-1})$
           $\oplus (\Gamma_{\gamma,1}, \ldots, \Gamma_{\gamma,m-1})$;
17.   if ($r = n$) then
18.     $C_m = \pi(P_m \oplus \Gamma_{\gamma,m}) \oplus \Gamma_{\gamma,m}$;
19.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus C_m$;
20.   else
21.     $\mathsf{tmp} = \pi^{-1}(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,m})$;
22.     $C_m = \mathsf{First}_r(P_m \oplus \mathsf{tmp})$; $T_m = C_m || (10^{n-r-1})$;
23.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus T_m \oplus \Gamma_{\gamma,m+1}$;
24.   end if;
25. end if;
26. $\mathsf{tag} = \pi^{-1}(\mathsf{sum})$;
return $(C_1 || \cdots || C_{m-1} || C_m, \mathsf{tag})$.
```

```
Backward1_π(N, C):
3.  $(C_1, \ldots, C_m) = \mathsf{Format}(C, n)$;
4.  $\gamma = \pi^{-1}(N)$;
5.  if ($m = 1$ and $r < n$) then
6.    $\mathsf{tmp} = \pi^{-1}(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,-1})$;
7.    $P_1 = \mathsf{First}_r(C_1 \oplus \mathsf{tmp})$;
8.    $\mathsf{sum} = C_1 \oplus \Gamma_{\gamma,-2}$;
9.  end if;
10. if ($m = 1$ and $r = n$) then
11.   $P_1 = \pi^{-1}(C_1 \oplus \Gamma_{\gamma,-1}) \oplus \Gamma_{\gamma,-1}$;
12.   $\mathsf{sum} = C_1 \oplus \Gamma_{\gamma,-3}$;
13. end if;
14. if ($m > 1$) then
15.   $(P_1, \ldots, P_{m-1})$
          $= \mathsf{ecb}_{\pi^{-1}}(C_1 \oplus \Gamma_{\gamma,1}, \ldots, C_{m-1} \oplus \Gamma_{\gamma,m-1})$
           $\oplus (\Gamma_{\gamma,1}, \ldots, \Gamma_{\gamma,m-1})$;
16.   if ($r = n$) then
17.     $P_m = \pi^{-1}(C_m \oplus \Gamma_{\gamma,m}) \oplus \Gamma_{\gamma,m}$;
18.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus C_m$;
19.   else
20.     $\mathsf{tmp} = \pi^{-1}(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,m})$;
21.     $P_m = \mathsf{First}_r(C_m \oplus \mathsf{tmp})$;
22.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus C_m \oplus \Gamma_{\gamma,m+1}$;
23.   end if;
24. end if;
25. $\mathsf{tag} = \pi^{-1}(\mathsf{sum})$;
return $(P_1 || \cdots || P_{m-1} || P_m, \mathsf{tag})$.
```

For a 4-block message, Figure 1 shows the working of the encryption algorithm of PAE1.

Table 5. Encryption and decryption algorithms for $\mathsf{PAE1}$.

$\mathsf{PAE1.Encrypt}_{\pi}(N, P)$:	$\mathsf{PAE1.Decrypt}_{\pi}(N, C, \mathsf{tag})$:
1. $(C, \mathsf{tag}) = \mathsf{Forward1}_{\pi}(N, P)$;	1. $(P, \mathsf{tag}') = \mathsf{Backward1}_{\pi}(N, C)$;
2. return (C, tag).	2. if $\mathsf{tag} = \mathsf{tag}'$ return P; else return $\bot$.

Table 6. Encryption and decryption algorithms for $\mathsf{PAEAD1}$.

$\mathsf{PAEAD1.Encrypt}_{\pi,\mathsf{fStr}}(N, H, P)$:	$\mathsf{PAEAD1.Decrypt}_{\pi,\mathsf{fStr}}(N, H, C, \mathsf{tag})$:
1. if $\lvert H\rvert = 0$, $\mathsf{tag}_2 = 0^n$;	1. if $\lvert H\rvert = 0$, $\mathsf{tag}_2 = 0^n$;
2. else	2. else
3. $\quad \upsilon = \pi^{-1}(\mathsf{fStr})$;	3. $\quad \upsilon = \pi^{-1}(\mathsf{fStr})$;
4. $\quad \mathsf{tag}_2 = \mathsf{PAuth1}_{\pi,\upsilon}(H)$;	4. $\quad \mathsf{tag}_2 = \mathsf{PAuth1}_{\pi,\upsilon}(H)$;
5. end if;	5. end if;
6. $(C, \mathsf{tag}_1) = \mathsf{Forward1}_{\pi}(N, P)$;	6. $(P, \mathsf{tag}_1) = \mathsf{Backward1}_{\pi}(N, C)$;
7. return $(C, \mathsf{tag}_1 \oplus \mathsf{tag}_2)$.	7. if $\mathsf{tag} = \mathsf{tag}_1 \oplus \mathsf{tag}_2$ return P; else return $\bot$.

Table 7. Encryption and decryption algorithms for $\overrightarrow{\mathsf{PAEAD1}}$.

$\overrightarrow{\mathsf{PAEAD1}}\mathsf{.Encrypt}_{\pi,\mathsf{fStr}}(N, H_1, \ldots, H_k, P)$:	$\overrightarrow{\mathsf{PAEAD1}}\mathsf{.Decrypt}_{\pi,\mathsf{fStr}}(N, H_1, \ldots, H_k, C, \mathsf{tag})$:
1. if $k = 0$, $\mathsf{tag}_2 = 0^n$;	1. if $k = 0$, $\mathsf{tag}_2 = 0^n$;
2. else	2. else
3. $\quad \upsilon = \pi^{-1}(\mathsf{fStr})$;	3. $\quad \upsilon = \pi^{-1}(\mathsf{fStr})$;
4. $\quad \mathsf{tag}_2 = \overrightarrow{\mathsf{PAuth1}}_{\pi,\upsilon}(H_1, \ldots, H_k)$;	4. $\quad \mathsf{tag}_2 = \overrightarrow{\mathsf{PAuth1}}_{\pi,\upsilon}(H_1, \ldots, H_k)$;
5. end if;	5. end if;
6. $(C, \mathsf{tag}_1) = \mathsf{Forward1}_{\pi}(N, P)$;	6. $(P, \mathsf{tag}_1) = \mathsf{Backward1}_{\pi}(N, C)$;
7. return $(C, \mathsf{tag}_1 \oplus \mathsf{tag}_2)$.	7. if $\mathsf{tag} = \mathsf{tag}_1 \oplus \mathsf{tag}_2$ return P; else return $\bot$.

7.1 PAEAD2: A Variant

In this section, we present a variant of PAEAD1. In the variant, the encryption algorithm uses only the permutation π (and not π^{-1}), while the decryption algorithm uses both π and π^{-1}. The core difference is in the modifications of the algorithms Forward1 and Backward1 to obtain Forward2 and Backward2 and the definition of PAuth2 from that of Backward2. The definitions of Forward2 and Backward2 are given in Table 8 and the definition of PAuth2 is as follows.

$$\mathsf{PAuth2}_{\pi,\mathsf{fStr}}(P) : \ (C, \mathsf{tag}) = \mathsf{Backward2}_{\pi}(\mathsf{fStr}, P); \ \text{return } \mathsf{tag}. \tag{16}$$

As discussed in the case of PAuth1, for an h-block message, the number of block cipher calls required to generate the tag produced by PAuth2 is $h + 1$. Again, one of these is on the string fStr and is to be computed once per session.

7.2 Recommendations for Use with AES

In PAE1, PAEAD1 and $\overrightarrow{\mathsf{PAEAD1}}$, the decryption algorithm uses only π^{-1} while the encryption algorithm uses both π and π^{-1}. In the case of AES, the encryption routine is simpler than the decryption routine. Hence, while using AES with PAE1, PAEAD1 and $\overrightarrow{\mathsf{PAEAD1}}$, it is advisable to instantiate π^{-1} using the encryption module of AES. While using PAE2, PAEAD2 and $\overrightarrow{\mathsf{PAEAD2}}$ with AES, it is advisable to instantiate π using the encryption module of AES.

7.3 Security

The following result states the security theorem for PAuth1 and PAuth2. The bounds for the vector versions follow from the single-input version and Theorem 1.

Table 8. The algorithms Forward2 and Backward2. Differences to Forward1 and Backward1 are highlighted using boxes.

```
$\mathsf{Forward2}_\pi(N, P)$:
4.  $(P_1, \ldots, P_m) = \mathsf{Format}(P, n)$;
5.  $\boxed{\gamma = \pi(N);}$
6.  if $(m = 1$ and $r < n)$ then
7.    $\boxed{\mathsf{tmp} = \pi(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,-1});}$
8.    $C_1 = \mathsf{First}_r(P_1 \oplus \mathsf{tmp})$; $T_1 = C_1||(10^{n-r-1})$;
9.    $\mathsf{sum} = T_1 \oplus \Gamma_{\gamma,-2}$;
10. end if;
11. if $(m = 1$ and $r = n)$ then
12.   $C_1 = \pi(P_1 \oplus \Gamma_{\gamma,-1}) \oplus \Gamma_{\gamma,-1}$;
13.   $\mathsf{sum} = C_1 \oplus \Gamma_{\gamma,-3}$;
14. end if;
15. if $m > 1$ then
16.   $(C_1, \ldots, C_{m-1})$
        $= \mathsf{ecb}_\pi(P_1 \oplus \Gamma_{\gamma,1}, \ldots, P_{m-1} \oplus \Gamma_{\gamma,m-1})$
          $\oplus(\Gamma_{\gamma,1}, \ldots, \Gamma_{\gamma,m-1})$;
17.   if $(r = n)$ then
18.     $C_m = \pi(P_m \oplus \Gamma_{\gamma,m}) \oplus \Gamma_{\gamma,m}$;
19.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus C_m$;
20.   else
21.     $\boxed{\mathsf{tmp} = \pi(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,m});}$
22.     $C_m = \mathsf{First}_r(P_m \oplus \mathsf{tmp})$; $T_m = C_m||(10^{n-r-1})$;
23.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus T_m \oplus \Gamma_{\gamma,m+1}$;
24.   end if;
25. end if;
26. $\boxed{\mathsf{tag} = \pi(\mathsf{sum});}$
return $(C_1|| \cdots ||C_{m-1}||C_m, \mathsf{tag})$.
```

```
$\mathsf{Backward2}_\pi(N, C)$:
3.  $(C_1, \ldots, C_m) = \mathsf{Format}(C, n)$;
4.  $\boxed{\gamma = \pi(N);}$
5.  if $(m = 1$ and $r < n)$ then
6.    $\boxed{\mathsf{tmp} = \pi(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,-1});}$
7.    $P_1 = \mathsf{First}_r(C_1 \oplus \mathsf{tmp})$;
8.    $\mathsf{sum} = C_1 \oplus \Gamma_{\gamma,-2}$;
9.  end if;
10. if $(m = 1$ and $r = n)$ then
11.   $P_1 = \pi^{-1}(C_1 \oplus \Gamma_{\gamma,-1}) \oplus \Gamma_{\gamma,-1}$;
12.   $\mathsf{sum} = C_1 \oplus \Gamma_{\gamma,-3}$;
13. end if;
14. if $(m > 1)$ then
15.   $(P_1, \ldots, P_{m-1})$
        $= \mathsf{ecb}_{\pi^{-1}}(C_1 \oplus \Gamma_{\gamma,1}, \ldots, C_{m-1} \oplus \Gamma_{\gamma,m-1})$
          $\oplus(\Gamma_{\gamma,1}, \ldots, \Gamma_{\gamma,m-1})$;
16.   if $(r = n)$ then
17.     $P_m = \pi^{-1}(C_m \oplus \Gamma_{\gamma,m}) \oplus \Gamma_{\gamma,m}$;
18.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus C_m$;
19.   else
20.     $\boxed{\mathsf{tmp} = \pi(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,m});}$
21.     $P_m = \mathsf{First}_r(C_m \oplus \mathsf{tmp})$;
22.     $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus C_m \oplus \Gamma_{\gamma,m+1}$;
23.   end if;
24. end if;
25. $\boxed{\mathsf{tag} = \pi(\mathsf{sum});}$
return $(P_1|| \cdots ||P_{m-1}||P_m, \mathsf{tag})$.
```

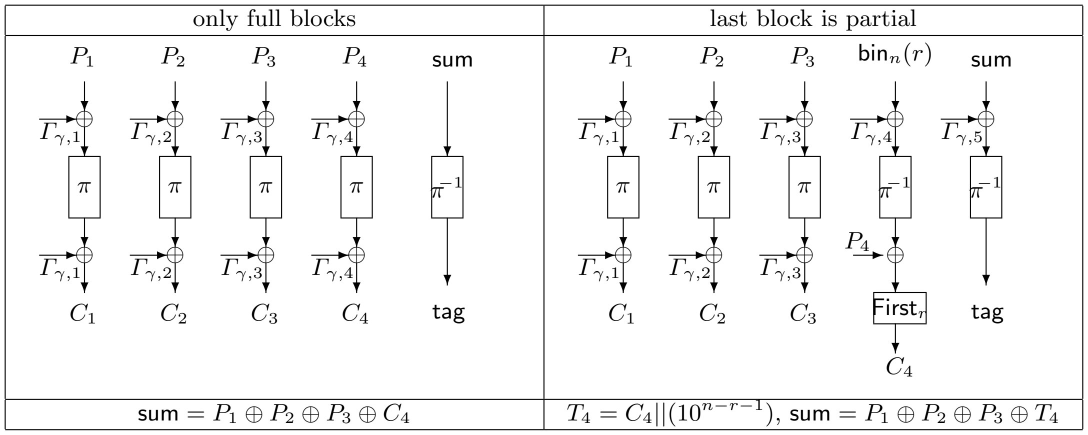

Fig. 1. Encryption using $\mathsf{PAE1}$: $\gamma = \pi^{-1}(N)$.

Theorem 3. *Let q and $\sigma \geq q$ be positive integers. Then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PAuth1}}(q,\sigma) \leq \frac{(7q+2)\sigma}{2^n};$$
$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PAuth2}}(q,\sigma) \leq \frac{(7q+2)\sigma}{2^n};$$
$$\mathbf{Adv}^{\mathrm{prf}}_{\overrightarrow{\mathsf{PAuth1}}}(q,\sigma) \leq \frac{q(q-1)}{2^{n+1}} + \frac{(7q+2)\sigma}{2^n};$$
$$\mathbf{Adv}^{\mathrm{prf}}_{\overrightarrow{\mathsf{PAuth2}}}(q,\sigma) \leq \frac{q(q-1)}{2^{n+1}} + \frac{(7q+2)\sigma}{2^n}.$$

The following provides the security statements for $\mathsf{PAE1}$ and $\mathsf{PAE2}$.

Theorem 4. *Let q and $\sigma \geq q$ be positive integers. Then*

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{PAE1}}(q,\sigma) \leq \frac{2(\sigma+2q)^2}{2^n};$$
$$\mathbf{Adv}^{\mathrm{ae\text{-}auth}}_{t\text{-}\mathsf{PAE1}} \leq \frac{1}{2^t} + \frac{2(\sigma+2q)^2}{2^n} + \frac{\sigma(7q+2)}{2^n};$$
$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{PAE2}}(q,\sigma) \leq \frac{2(\sigma+2q)^2}{2^n};$$
$$\mathbf{Adv}^{\mathrm{ae\text{-}auth}}_{t\text{-}\mathsf{PAE2}} \leq \frac{1}{2^t} + \frac{2(\sigma+2q)^2}{2^n} + \frac{\sigma(7q+2)}{2^n}.$$

The security statements for the AEAD schemes are given by the following theorem.

Theorem 5. *Let q and $\sigma \geq q$ be positive integers. Then*

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{PAEAD1}}(q,\sigma) \leq \frac{2(\sigma+2q)^2}{2^n};$$
$$\mathbf{Adv}^{\mathrm{aead\text{-}auth}}_{t\text{-}\mathsf{PAEAD1}}(q,\sigma) \leq \frac{1}{2^t} + \frac{1}{2^{n-1}} \times \left(\sigma(2\sigma+11q+2)+4q^2\right);$$
$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{PAEAD2}}(q,\sigma) \leq \frac{2(\sigma+2q)^2}{2^n};$$
$$\mathbf{Adv}^{\mathrm{aead\text{-}auth}}_{t\text{-}\mathsf{PAEAD2}}(q,\sigma) \leq \frac{1}{2^t} + \frac{1}{2^{n-1}} \times \left(\sigma(2\sigma+11q+2)+4q^2\right).$$

Table 9. Encryption and decryption algorithms for $\mathsf{PAE2}$.

$\mathsf{PAE2.Encrypt}_\pi(N,P)$:	$\mathsf{PAE2.Decrypt}_\pi(N,H_1,\ldots,H_k,C,\mathsf{tag})$:
1. $(C,\mathsf{tag}_1)=\mathsf{Forward2}_\pi(N,P)$;	1. $(P,\mathsf{tag}')=\mathsf{Backward2}_\pi(N,C)$;
2. return $(C,\mathsf{tag}_1\oplus\mathsf{tag}_2)$.	2. if $\mathsf{tag}=\mathsf{tag}'$ return P; else return $\bot$.

Table 10. Encryption and decryption algorithms for $\mathsf{PAEAD2}$.

$\mathsf{PAEAD2.Encrypt}_{\pi,\mathsf{fStr}}(N,H,P)$:	$\mathsf{PAEAD2.Decrypt}_{\pi,\mathsf{fStr}}(N,H_1,\ldots,H_k,C,\mathsf{tag})$:
1. if $\lvert H\rvert=0$, $\mathsf{tag}_2=0^n$;	1. if $\lvert H\rvert=0$, $\mathsf{tag}_2=0^n$;
2. else	2. else
3. $\quad\upsilon=\pi(\mathsf{fStr})$;	3. $\quad\upsilon=\pi(\mathsf{fStr})$;
4. $\quad\mathsf{tag}_2=\mathsf{PAuth2}_{\pi,\upsilon}(H)$;	4. $\quad\mathsf{tag}_2=\mathsf{PAuth2}_{\pi,\upsilon}(H)$;
5. end if;	5. end if;
6. $(C,\mathsf{tag}_1)=\mathsf{Forward2}_\pi(N,P)$;	6. $(P,\mathsf{tag}_1)=\mathsf{Backward2}_\pi(N,C)$;
7. return $(C,\mathsf{tag}_1\oplus\mathsf{tag}_2)$.	7. if $\mathsf{tag}=(\mathsf{tag}_1\oplus\mathsf{tag}_2)$ return P; else return $\bot$.

Table 11. Encryption and decryption algorithms for $\overrightarrow{\mathsf{PAEAD2}}$.

$\overrightarrow{\mathsf{PAEAD2}}.\mathsf{Encrypt}_{\pi,\mathsf{fStr}}(N,H_1,\ldots,H_k,P)$:	$\overrightarrow{\mathsf{PAEAD2}}.\mathsf{Decrypt}_{\pi,\mathsf{fStr}}(N,H_1,\ldots,H_k,C,\mathsf{tag})$:
1. if $k=0$, $\mathsf{tag}_2=0^n$;	1. if $k=0$, $\mathsf{tag}_2=0^n$;
2. else	2. else
3. $\quad\upsilon=\pi(\mathsf{fStr})$;	3. $\quad\upsilon=\pi(\mathsf{fStr})$;
4. $\quad\mathsf{tag}_2=\overrightarrow{\mathsf{PAuth2}}_{\pi,\upsilon}(H_1,\ldots,H_k)$;	4. $\quad\mathsf{tag}_2=\overrightarrow{\mathsf{PAuth2}}_{\pi,\upsilon}(H_1,\ldots,H_k)$;
5. end if;	5. end if;
6. $(C,\mathsf{tag}_1)=\mathsf{Forward2}_\pi(N,P)$;	6. $(P,\mathsf{tag}_1)=\mathsf{Backward2}_\pi(N,C)$;
7. return $(C,\mathsf{tag}_1\oplus\mathsf{tag}_2)$.	7. if $\mathsf{tag}=(\mathsf{tag}_1\oplus\mathsf{tag}_2)$ return P; else return $\bot$.

Security statements for the privacy of the variants of the AEAD schemes where the header can be a vector of strings remains unchanged. The security statements for authentication changes and is given by the following result. An additive degradation of $q(q-1)/2^{n+1}$ occurs due to the conversion of the single-input PRF to a vector-input PRF.

Theorem 6. *Let $\sigma\geq q\geq 1$. Then*

$$\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\overrightarrow{\mathsf{PAEAD1}}}(q,\sigma)\leq\frac{1}{2^t}+\frac{q(q-1)}{2^{n+1}}+\frac{1}{2^{n-1}}\times\Big(\sigma(2\sigma+11q+2)+4q^2\Big);$$
$$\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\overrightarrow{\mathsf{PAEAD2}}}(q,\sigma)\leq\frac{1}{2^t}+\frac{q(q-1)}{2^{n+1}}+\frac{1}{2^{n-1}}\times\Big(\sigma(2\sigma+11q+2)+4q^2\Big);$$

8 Deterministic Authenticated Encryption with Associated Data

The basic idea behind the DAE and the DAEAD schemes is based on the SIV construction in [52]. For encryption using the DAE scheme, a tag is generated by applying PAuth to the plaintext P. This tag is used as an IV to a variant of the counter mode (used in [59]) to obtain C from P. The final ciphertext is (C,tag). Decryption from (C,tag) is easy and done as follows. First apply the counter mode with tag to obtain P from C; apply PAuth to the obtained P to obtain tag_1; if $\mathsf{tag}=\mathsf{tag}_1$, then return P, else return $\bot$ indicating lack of authentication. The scheme is denoted as DAE and the encryption and decryption algorithms are given in Table 12.

Handling a header in this framework is also quite easy. The idea is to use $\overrightarrow{\mathsf{PAuth}}$ to authenticate the vector of strings $(H_1,\ldots,H_k,P)$ and obtain tag. The rest of the scheme remains unchanged. We denote by DAEAD the algorithm when a header is authenticated. The encryption and decryption algorithms of DAEAD are given in Table 13. Note that applying DAEAD to the case when the header is empty is not the same as the scheme DAE. This is due to the fact that $\mathsf{PAuth}(P)$ and $\overrightarrow{\mathsf{PAuth}}(P)$ do not provide the same output.

Table 12. Encryption and decryption algorithms for DAE.

$\mathsf{DAE.Encrypt}_{\pi,\mathsf{fStr}}(P)$:	$\mathsf{DAE.Decrypt}_{\pi,\mathsf{fStr}}(C,\mathsf{tag})$:
1. $\mathsf{tag} = \mathsf{PAuth}_{\pi,\mathsf{fStr}}(P)$;	2. $(C_1,\ldots,C_m) = \mathsf{Format}(C,n)$;
2. $(P_1,\ldots,P_m) = \mathsf{Format}(P,n)$;	3. $(P_1,\ldots,P_{m-1}) = (C_1,\ldots,C_{m-1})$
3. $(C_1,\ldots,C_{m-1}) = (P_1,\ldots,P_{m-1})$	$\oplus\ \mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;
$\oplus\ \mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;	4. $S_m = \mathsf{First}_r(C_m \oplus \pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;
4. $T_m = \mathsf{First}_r(P_m \oplus \pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;	5. $P = P_1\|\cdots\|P_{m-1}\|S_m$;
5. $C = C_1\|\cdots\|C_{m-1}\|T_m$;	2. $\mathsf{tag}_1 = \mathsf{PAuth}_{\pi,\mathsf{fStr}}(P)$;
5. return (C,tag).	3. if $\mathsf{tag} = \mathsf{tag}_1$ return P else return $\perp$.

Table 13. Encryption and decryption algorithms for DAEAD.

$\mathsf{DAEAD.Encrypt}_{\pi,\mathsf{fStr}}(H_1,\ldots,H_k,P)$:	$\mathsf{DAEAD.Decrypt}_{\pi,\mathsf{fStr}}(H_1,\ldots,H_k,C,\mathsf{tag})$:
1. $\mathsf{tag} = \overrightarrow{\mathsf{PAuth}}_{\pi,\mathsf{fStr}}(H_1,\ldots,H_k,P)$;	2. $(C_1,\ldots,C_m) = \mathsf{Format}(C,n)$;
2. $(P_1,\ldots,P_m) = \mathsf{Format}(P,n)$;	3. $(P_1,\ldots,P_{m-1}) = (C_1,\ldots,C_{m-1})$
3. $(C_1,\ldots,C_{m-1}) = (P_1,\ldots,P_{m-1})$	$\oplus\ \mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;
$\oplus\ \mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;	4. $S_m = \mathsf{First}_r(C_m \oplus \pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;
4. $T_m = \mathsf{First}_r(P_m \oplus \pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;	5. $P = P_1\|\cdots\|P_{m-1}\|S_m$;
5. $C = C_1\|\cdots\|C_{m-1}\|T_m$;	2. $\mathsf{tag}_1 = \overrightarrow{\mathsf{PAuth}}_{\pi,\mathsf{fStr}}(H_1,\ldots,H_k,P)$;
5. return (C,tag).	3. if $\mathsf{tag} = \mathsf{tag}_1$ return P else return $\perp$.

The security statements for DAE and DAEAD are given by the following result.

Theorem 7. *Let $\sigma \geq q \geq 1$. Then*

$$\begin{aligned}
\mathbf{Adv}^{\text{priv}}_{\mathsf{DAE}}(q,\sigma) &\leq \frac{2(\sigma+2q)^2}{2^n};\\
\mathbf{Adv}^{\text{daead-auth}}_{t\text{-}\mathsf{DAE}}(q,\sigma) &\leq \frac{1}{2^t} + \frac{1}{2^{n-1}} \times \left(\sigma(2\sigma+11q+2)+4q^2\right);\\
\mathbf{Adv}^{\text{priv}}_{\mathsf{DAEAD}}(q,\sigma) &\leq \frac{q(q-1)}{2^{n+1}} + \frac{2(\sigma+2q)^2}{2^n};\\
\mathbf{Adv}^{\text{daead-auth}}_{t\text{-}\mathsf{DAEAD}}(q,\sigma) &\leq \frac{1}{2^t} + \frac{q(q-1)}{2^{n+1}} + \frac{1}{2^{n-1}} \times \left(\sigma(2\sigma+11q+2)+4q^2\right).
\end{aligned}$$

9 Comparison to Some Existing Schemes

The schemes described here are modes of operations of a block cipher based. So, it makes sense to compare only with other modes of operations of a block cipher. We briefly discuss the relation of the schemes presented here to a selection of important works by other authors.

9.1 AEAD Schemes

The schemes proposed in this work make one block cipher call per message (or ciphertext) block and such schemes are called Rate-1 schemes. The NIST standardised GCM requires one block cipher call and one finite field multiplication per block of data, while the other NIST standard CCM requires two block cipher calls per data block. So, these are slower than the schemes considered here.

The early Rate-1 parallelisable schemes were due to Jutla [37] and Gligor-Donescu [26]. The sequence of designs by Rogaway called OCB1 [51], OCB2 [50] and OCB3 [40] are also Rate-1 parallelisable schemes. Of these, OCB3 (currently called OCB) is the most efficient.

Comparison to OCB. Below we highlight several aspects on which the constructions presented here differ from OCB.

Reconfigurable masking. The Galois field based masking strategy described here has the unique feature of easy reconfigurability. This feature is not present in OCB. The masking strategy of OCB is different from that used here. It is mentioned in [40] that if optimised with care, the masking strategy for OCB can be faster than that of Galois field based masking. The speed-up, though, is not much. In general, we expect the speed of the masking strategies used here to compare well with the OCB masking. This is also evident from the experimental results that we report later.

One aspect of having a reconfigurable masking strategy as part of the specification of a mode of operation is that it allows the optimisation of the code for a target processor. For example, for the current Intel processors one would choose $n_1 = 128$ and $n_2 = 1$ (or $n_1 = 1$ and $n_2 = 128$) and use the available vector instructions to write the code for the next and previous mask computations. On the other hand, for small processors such as the Atmel AVR 8-bit, 16-bit and 32-bit microcontrollers [17, 29] it would make more sense to choose $n_1 = 8$ (correspondingly $n_2 = 16$), $n_1 = 16$ (correspondingly $n_2 = 8$) and $n_1 = 32$ (correspondingly $n_2 = 4$) respectively. Fixing the values of n_1 and n_2 in the specification will bias the efficiency of masking towards one end of the processor architectures. In particular, we do not see any advantage of fixing one particular masking strategy as part of the specification.

Another aspect of having a reconfigurable masking strategy is that organisations will have the option of choosing their own secret values for $(\rho(\alpha), \mu(x))$. If either $n_1 = 1, n_2 = 128$ or $n_1 = 128, n_2 = 1$ hold, then as mentioned in Section 3.3, the number of possible choices of the pair $(\rho(\alpha), \mu(x))$ will be around 2^{119}. So, a particular organisation can randomly choose a mode of operation from this large family. This customisation facility will provide an additional layer of 'security by obscurity' over and above the provable security guarantee already enjoyed by the schemes.

Nonce length. Nonces in OCB have to be of lengths less than n. So, if an application generates n-bit nonces, then such an application will have to drop a bit while ensuring the uniqueness of nonces. In comparison, the nonce length for our schemes is n bits. If an application generates shorter length nonces, then this can simply be padded with zeros to obtain an n-bit nonce without loss of uniqueness.

Number of block cipher calls. For h header blocks and m message blocks, OCB uses a total of $m + h + 3$ block cipher calls. Out of these, one call is required once per session and the number of calls per message is $m+h+2$. In the case of non-empty header, the number of block cipher calls for the PAEAD schemes is $m + h + 4$ ($m + 2$ calls to process the message; $h + 2$ calls to process the header) where the two successive calls on fStr are made once per session. So, the number of calls per message is again $m + h + 2$. If the header is empty, then there are no block cipher calls required to process the header and so the total number of calls is $m + 2$. This is the same as that of the proposed PAE schemes.

In case the nonces are generated using a counter, OCB uses a clever strategy to ensure that the encryption of the nonce is to be done only once per 64 messages. This saves a block cipher call for 63 out of 64 messages. On the other hand, there is small additional processing that is required over and above the block cipher call. So, if the nonces do not occur as a counter, then OCB has to encrypt the nonce *and* perform a processing of it. In such a case, the strategy for avoiding the encryption of nonces is not useful but, the overhead remains.

Memory requirement. A feature required for efficient mask generation in OCB is that certain masks can be pre-computed and stored in memory. Storing $\ell + 2$ n-bit blocks allows the processing of 2^{ℓ} blocks. (A message consisting of 1024 bytes will have 64 blocks when $n = 128$; so, $\ell = 6$ and a total of 128 bytes of storage space for the key material will be required.) These blocks have to be securely stored as their leakage will result in the system becoming insecure. The schemes described here do not require such storage. For one thing this may be important in scenarios where storage is costly. Perhaps more importantly, by having a smaller storage requirement, the present schemes offer lesser targets for attack.

Standalone MAC algorithm. OCB processes the header to obtain an n-bit string and XORs it with the tag produced on the message. As a standalone algorithm, however, the header processing algorithm is not a MAC algorithm. If the requirement is to implement an authentication scheme *and* an AEAD scheme, then along with OCB one has to implement a separate MAC algorithm. In contrast, both $\mathsf{PAuth1}$ and $\mathsf{PAuth2}$ are standalone MAC algorithms and are respectively defined from $\mathsf{Backward1}$ and $\mathsf{Backward2}$. Providing the message to be the empty string returns the tag on the header. So, a single implementation of the AEAD algorithm can provide both the tasks of authentication and AEAD.

Handling of a vector of strings as the header. As argued in the context of DAEAD, applications sometimes require to authenticate a header of strings [52]. By extension, the same should also be true of nonce-based AEAD schemes. Consequently, we have described variants of the basic AEAD schemes which can handle a vector of strings as the header. Such an option is not part of OCB, though we note that it should be possible to modify the specification of OCB to handle vector headers.

Comparison to CLOC, SILC and OTR. Some of the recent works on AEAD schemes are the proposals CLOC [32], SILC [33] and OTR [46]. Another recent proposal is COBRA [3], but, a serious flaw has been pointed out in the construction by Nandi (see `http://competitions.cr.yp.to/round1/aescobra-withdraw.txt`).

The schemes CLOC and SILC are similar and are based on the CFB mode of operation. As a result, the constructions are inherently sequential in nature and cannot benefit from the pipelining structure of many block ciphers including the AES. The scheme CLOC was proposed as an improved alternative to CCM and EAX. The claimed advantage of CLOC is that of minimal overhead in terms of block cipher calls and memory requirement. If there is no header, then processing an m-block message by CLOC requires $2 + 2m$ calls and if there is an h-block header, then the total number of calls to process both the header and the message is $1 + h + 2m$. The state consists of two n-bit blocks, the memory required to store the two chaining blocks for encryption and authentication.

In the case of the schemes presented here, when $h = 0$, the number of block cipher calls is $m+2$ which is smaller than CLOC for all $m \geq 1$. If $h > 0$, then the number of required calls is $m+h+4$ which is smaller than or equal to that of CLOC for $m \geq 3$. Storing one n-bit quantity (the result of the double encryption of fStr) reduces the number of calls to $m + h + 2$ which is smaller than or equal to that of CLOC for all $m \geq 1$. In terms of memory requirement, the schemes in this work will require to store an n-bit mask along with output of the double encryption of fStr (when $h > 0$). So, CLOC would be preferred to the presented schemes when messages are very short and there is a very tight memory constraint. SILC is another scheme which has been proposed as an improvement over CLOC in requiring an even smaller hardware footprint. The number of block cipher calls required by SILC is one more than that of CLOC.

Both the schemes CLOC and SILC do not require the inverse of the block cipher. From a theoretical point of view, this is an advantage since the assumption on the block cipher is that of a pseudo-random permutation instead of a strong pseudo-random permutation. The other advantage is in hardware implementation, where the inverse of the block cipher is not required to be implemented leading to a smaller size hardware. The downside of CLOC and SILC however, is that these are sequential algorithms and cannot benefit from the pipelined implementation of the regular structure of block ciphers such as AES.

OTR is an interesting construction which also does not utilise the inverse of the block cipher and is parallelisable. It uses a Feistel structure two blocks at a time. The parallelism in OTR is of the following form. The odd numbered data blocks can be processed in parallel and then the even numbered data blocks can be processed in parallel. Further, the odd numbered data blocks have to be buffered to be XORed with the outputs of the encryptions of the even numbered data blocks. Using AES-NI instructions on Intel processors, it is possible to utilise this structure to get a fast implementation [13]. On the other hand, in hardware, exploiting the parallelism in OTR will certainly require buffering the segments of odd numbered data blocks which will substantially push up the area requirement. Also, even for software, it is not clear how to effectively utilise the parallelism in OTR on processors which do not support AES instructions. In contrast, the parallelism in all the proposed schemes in this paper is simple and regular and can be fully exploited in both hardware and software.

9.2 DAEAD Schemes

The first DAEAD scheme was proposed in [52] and was named the SIV construction. This construction was based on the S2V construction which builds a vector-input PRF from a single-input PRF. In SIV, the S2V method is used on the CMAC authentication algorithm to build a vector-input PRF to generate the tag on the header and the message. The CTR mode of encryption is used to process the message. The two components CMAC and CTR use independent keys. The CMAC algorithm is based on the CBC mode of operation and is inherently sequential. The CTR mode used in [52] uses $\mathrm{Ctr} + i$ as the i-th offset. This is different from the somewhat simpler CTR mode from [59] that we use which has $\mathrm{Ctr} \oplus \mathsf{bin}_n(i)$ as the i-th offset.

Improvements to the work in [52] were made in [36, 35] which proposed two constructions called HBS and BTM. These use a single key and a polynomial based hash function to process the vector of strings which comprise the header. The constructions of DAE and DAEAD schemes that we describe also use a single key and the processing of the header is done using a block cipher (and without any finite field multiplication) as in the SIV construction. Unlike SIV, however, we use the fully parallelisable PAuth algorithm and its vector version to process the header.

McOE-G [24], COPA [2] and POET [43] are three recent examples of nonce-misuse resistant AEAD modes of operations. Of these, McOE-G is a sequential algorithm which requires both block cipher calls and finite field multiplications. COPA and POET are both parallel modes which use only a block cipher as the building block. POET makes three calls per data block while COPA makes two calls per block. (For AES, there is a suggestion that POET may use reduced-round

AES, but, then the scheme no longer remains a mode of operation of AES.) So, COPA is faster than both McOE-G and POET and the experimental results in [13] confirms this. The parallelism in COPA, however, is not unrestricted. Similar to that of OTR, for COPA the inputs to the even numbered calls depend on the outputs of the odd numbered calls.

The construction ELmD [19] uses a different approach. It is an encrypt-mix-decrypt type of construction. Unlike COPA, the scheme is fully parallel and makes two block cipher calls per data block. The parallelism, however, alternates between calls to the encrypt module of the block cipher and the decrypt module of the block cipher. As a result, hardware implementations will require two separate pipelines of the encryption and the decryption modules of the block cipher.

All of the constructions McOE-G, COPA, POET and ELmD require both the encryption and the decryption modules of the underlying block cipher. In contrast, the DAEAD construction given here requires only the encryption module of the block cipher. It provides unrestricted parallelism and uses two calls per data block. As a result, it offers better features compared to the other schemes mentioned here.

10 Software Implementation

For the sake of completeness of the work and also for the sake of illustration of performance, we decided to implement all the constructions presented here. To obtain a complete implementation, we needed to decide on a block cipher. The natural choice is the AES since it is currently the most popular block cipher. We note though that the mode of operation is independent of the actual block cipher. Any other block cipher can also be easily plugged into the mode of operation.

Two implementations were made in the ‘C’ programming language and both the implementations are publicly available from [16]. The first implementation is a basic program where a binary string is represented as an array of zeros and ones. This is to be considered as the reference implementation of the schemes described in this work. Since the implementation itself is quite simple, we do not discuss it any further here.

The second implementation was targeted towards modern Intel processors. Whereever possible we tried to vectorize our code and utilize the 128-bit XMM registers through the SSE instruction set. The SSE instructions were accessed using Intel intrinsics. In all the schemes, the basic building blocks are the masking schemes and the block cipher calls. We present some specifics of the implementations of the masking schemes and the AES below.

Starting from the Westmere processor, Intel has produced a series of successive processors, namely Sandybridge, Ivybridge and most recently the Haswell processor. The latency and throughput figures of the various instructions slightly vary from processor to processor. As a result, a code which is optimised for one of these processors need not be the fastest on another processor. We have tried to optimise our code keeping the Ivybridge processor in mind and performance figures reported later are for this processor. At the same time, we do note that the basic principles of our code design will remain the same across all the processors. So, it should be possible to modify our code to get optimised implementations on the other processors.

AES supports 128-bit, 192-bit and 256-bit keys. Our reference implementation supports all of these key sizes. On the other hand, the fast implementation has been done only for 128-bit keys.

10.1 Implementing the Masks

The masking schemes take a 128-bit string as input and produces a 128-bit string as output. The details of the mapping depend on the type of mask and the chosen irreducible polynomial. We discuss these issues.

Implementing *xtimes*: First consider the implementation of the mask where $n_1 = 128$ and $n_2 = 1$. This operation is popularly known as *xtimes* or *doubling*. For this mask, if the input is $A \in \{0,1\}^{128}$, then the output B can be computed using the following steps. The polynomial representing $GF(2^{128})$ is given in Table 2.

```
1. $b \leftarrow \mathsf{msb}(A)$;
2. $B \leftarrow A \ll 1$;
3. if $b = 1$;
4.     $B \leftarrow B \oplus 0x87$;
5. return $B$.
```

The problem of implementing the above procedure using a 128-bit register in an Intel machine is that there is no instruction to left shift the content of an XMM register by one bit. Hence, to implement the above procedure within a 128-bit register using the available instructions requires some more work. In particular, the steps followed are shown in Figure 2, where the required instructions in each step are also mentioned. The piece of code using Intel intrinsics implementing the procedure is shown in Figure 3.

```
1. parse $A$ as $A_3||A_2||A_1||A_0$ where $A_3, A_2, A_1, A_0$ are 32-bit words, and
       $A_i = (a_{i,31}, a_{i,30}, \ldots, a_{i,1}, a_{i,0})$;
2. $R \leftarrow \underline{a_{3,31}}||\underline{a_{2,31}}||\underline{a_{1,31}}||\underline{a_{3,31}}$ where $\underline{a}$ means $a$ repeated 32 times;
       ($R$ can be obtained from $A$ by the instruction PSRAD)
3. $S \leftarrow \underline{a_{1,31}}||\underline{a_{1,31}}||\underline{a_{1,31}}||\underline{a_{3,31}}$;
       ($S$ can be obtained from $R$ by using PSHUFD instruction)
4. $S \leftarrow S \wedge (0x00||0x01||0x00||0x87)$;
5. $S \leftarrow S \oplus [(A_3||A_2) \ll 1||(A_1||A_0) \ll 1]$;
6. return $S$.
```

Fig. 2. Computing the next mask with the parameters $n_1 = 128, n_2 = 1$ using 128-bit registers.

```
 static inline void gfmulby2(__m128i a,__m128i* res){
*res = _mm_srai_epi32(a,31);
*res = _mm_shuffle_epi32(*res,0x57);
*res = _mm_and_si128(*res,_mm_set_epi32(0x00,0x01,0x00,0x87));
*res = _mm_xor_si128(*res,_mm_slli_epi64(a,0x01));
}
```

Fig. 3. The code corresponding to the procedure in Figure 2

There can be other strategies to implement *xtimes* using the 128-bit registers. We discuss two specific strategies adopted in [4] and [13].

1. The strategy adopted in [4] is depicted in Figure 4(a). The basic idea involves using a table `tab` with four 128-bit values. The instruction `MOVMSKPD` (equivalent intrinsic `_mm_movemask_pd(a)`), treats the input `a` as two 64-bit double precision values and outputs the their sign bits. Based on these values the appropriate mask from `tab` is chosen. This strategy uses a data dependent table look-up.

2. In [13] (shown in Figure 4(b)), the most significant bit of the input a is extracted by novel use of the instructions `PCMPGTB` and `PEXTRB`. Based on this extracted bit proper maskings are done[3]. This strategy requires a data dependent branching and hence is not a constant time code.

```
__m128i computeXtimes1(__m128i a)  {
    __m128i b; int r;
    __m128i tab[4];\
    tab[0]= _mm_set_epi32(0x00,0x00,0x00,0x00);
    tab[1]= _mm_set_epi32(0x00,0x01,0x00,0x00);
    tab[2]= _mm_set_epi32(0x00,0x00,0x00,0x87);
    tab[3]= _mm_set_epi32(0x00,0x01,0x00,0x87);

    r=_mm_movemask_pd((__m128d)a);
    b = _mm_xor_si128(tab[r],_mm_slli_epi64(a,0x01));
    return b;
  }
```

(a)

```
__m128i computeXtimes2(__m128i a)  {
   __m128i v1, v2, v3;
   v3 = _mm_set_epi32(0x0,0x0,0x0,0x87);
   v1 = _mm_slli_epi64(a,1);
   v2 = _mm_slli_si128(a,8);
   v2 = _mm_srli_epi64(v2,63);
   if(_mm_extract_epi8(_mm_cmpgt_epi8(_mm_set1_epi8(0x00),a),15)== 0xff)
        a = _mm_xor_si128(_mm_or_si128(v1,v2),v3);
   else a =_mm_or_si128(v1,v2);
   return a;
  }
```

(b)

Fig. 4. (a) The code for *xtimes* described in [4]; (b) The code for *xtimes* described in [13].

The various vector instructions along with their latencies and throughputs used in our implementation of *xtimes* and the implementations in [4] and [13] are shown in Table 14. The strategy in [4] uses the least number of instructions, and is also optimal in terms of the latencies, but it uses a table lookup and the latency associated with that is not shown in the Table. The strategy in [13] uses the maximum number of instructions. Later we present some experimental results to compare the three strategies. These show that our strategy is the most efficient followed by that of [13] and [4]. Moreover, both the strategies in [4] and [13] suffers from the fact that they use data dependent table lookups and branching, respectively. This feature can lead to avenues for software side channel attacks, whereas our strategy does not involve such issues.

Implementing the other masking functions: Implementation of the other kinds of masks using vector instructions have not been considered in the literature. For implementing them, we use the same strategy as in computing *xtimes*. In particular, we avoid data dependent branchings and/or table lookups.

Computing the other kind of masks requires more instructions. For example, consider mask generation where $GF(2^{128})$ is represented as a tower field with $n_1 = 32$ and $n_2 = 4$. Let $A =$

[3] The specific code discussed in [13] is incorrect. The instruction `PCMPGTB` does comparison of signed 8-bit integers, and this is not considered in the code described in [13].

	Instruction	Intrinsic	Latency	Throughput
This work	PSRAD	_mm_srai_epi32	2	2
	PSHUFD	_mm_shuffle_epi32	4	2
	PAND	_mm_and_si128	2	2
	PXOR	_mm_xor_si128	2	2
	PSLLQ	_mm_slli_epi64	2	1
Aoki et al. [4]	MOVMSKPD	_mm_movemask_pd	2	2
	PSLLQ	_mm_slli_epi64	2	1
	PXOR	_mm_xor_si128	2	2
Bogdanov et al. [13]	PSLLQ	_mm_slli_epi64	2	1
	PSLLDQ	_mm_slli_si128	4	2
	PSRLQ	_mm_srli_epi64	2	2
	PEXTRB	_mm_extract_epi8	3	1
	PCMPGTB	_mm_cmpgt_epi8	2	1
	PXOR	_mm_xor_si128	2	2
	POR	_mm_or_si128	2	2

Table 14. The vector instructions along with their latencies used in the various implementations of `xtimes`. The latency and throughput data are from [1]. Throughput here means the number of cycles of wait necessary for a instruction to be invoked after one invocation.

$A_0||A_1||A_2||A_3$, be a 128-bit string where each A_i is 32 bits long. Then to compute the mask based on the specific polynomials $\rho(\alpha)$ and $\mu(x)$ as in Table 2 the following procedure is required:

```
1. $C \leftarrow (A_3 \lll 1)$
2. **if** $\mathsf{msb}(A_3) = 1$
3.     $C \leftarrow C \oplus \mathsf{0x0a000021}$
4. $C \leftarrow C \oplus A_2 \oplus A_0$
5. **return** $A_2||A_1||A_0||C$
```

To implement this procedure within a 128-bit register we follow the procedure shown in Figure 5. The corresponding code using Intel intrinsics is shown in Figure 6.

```
1.  parse $A$ as $A_3||A_2||A_1||A_0$ where $A_3, A_2, A_1, A_0$ are 32-bit words, and
        $A_i = (a_{i,31}, a_{i,30}, \ldots, a_{i,1}, a_{i,0})$;
2.  $R \leftarrow \langle \underline{a_{3,31}}||\underline{a_{2,31}}||\underline{a_{1,31}}||\underline{a_{3,31}} \rangle$; (where $\underline{a}$ means $a$ repeated 32 times)
        ($R$ can be obtained from $A$ by the instruction PSRAD)
3.  $S \leftarrow R \wedge \langle \mathsf{0x0a000021}||\mathsf{0x00}||\mathsf{0x00}||\mathsf{0x00} \rangle$;
4.  $C \leftarrow (A_3 \lll 1)||(A_2 \lll 1)||(A_1 \lll 1)||(A_0 \lll 1)$;
5.  $C \leftarrow C \oplus S$;
6.  $C \leftarrow (C \gg 96)$;

7.  $T \leftarrow (A \gg 64)$;
    (now $T$ contains $0^{64}||A_3||A_2$)
8.  $T \leftarrow T \oplus A$;
9.  $T \leftarrow T \wedge (0^{96}||1^{32})$;
    (now $T$ contains $(0^{96}||A_0 \oplus A_2)$)
10. **return** $(A \lll 32) \oplus T \oplus C$.
```

Fig. 5. Computing the mask for $n_1 = 32, n_2 = 4$ with 128-bit registers.

```
__m128i mask32(__m128i a)  {
   __m128i t1, r;

   r = _mm_srai_epi32(a,31);
   r = _mm_and_si128(r, _mm_set_epi32(0x0a000021,0x00,0x00,0x00));
   r = _mm_xor_si128(_mm_slli_epi32(a,1),r);
   r = _mm_srli_si128(r,12);

   t1 = _mm_srli_si128(a,8);
   t1 = _mm_xor_si128(a,t1);
   t1 = _mm_and_si128(t1,_mm_set_epi32(0x00,0x00,0x00,0xffffffff));
   r = _mm_xor_si128(r,t1);
   r = _mm_xor_si128(r, _mm_slli_si128(a,4));
   return r;
}
```

Fig. 6. The code corresponding to the procedure in Figure 5.

Experimental Results: There are five options for generating the masks which correspond to the first five representations of $GF(2^{128})$ given in Table 2. We use the notation given in Table 15 to denote the different masking types.

$n_1 = 128$, $n_2 = 1$	$n_1 = 64$, $n_2 = 2$	$n_1 = 32$, $n_2 = 4$	$n_1 = 16$, $n_2 = 8$	$n_1 = 8$, $n_2 = 16$
Type-1	Type-2	Type-3	Type-4	Type-5

Table 15. Numbering the different types of maskings.

Experimental performance data for the various masking schemes are shown in Table 16. The values shown in the table correspond to the (average) number of cycles required for computing one mask. Two experiments were performed. In the first one, starting from a specific 128-bit value, 1024 masks were successively computed and the average time for computing each mask was found out. This experiment was repeated 1024 times and the row **1 mask** in Table 16 shows the median of of these 1024 values.

The other experiment was directed to judge the extent to which each of the masking strategies can utilize the instruction level pipelining. To do this we fixed eight different initial 128-bit values and successively computed 1024 masks for each of these values. Again we found the average time for computing each mask, and repeated this experiment 1024 times. The row **8 masks** reports the median of these 1024 values. Table 16 clearly shows that the for Type-1 maskings, our strategy is

Type	**1**			**5**	**4**	**3**	**2**
	[4]	[13]	**Ours**				
1 mask	15.27	11.58	4.21	7.24	7.51	5.93	6.30
8 masks	2.57	4.97	1.79	4.10	4.04	3.65	3.36

Table 16. Experimental comparison of the various masking strategies.

better than the ones used in [4] and [13]. The principal reason being that our strategy does not use any conditional branching. Such branchings may lead to higher latencies. The strategy depicted in [4] does not involve any branchings, but involves a data dependent table look up which has a performance penalty.

10.2 Implementing the AES

Modern Intel processors provide specialized instruction set called AES-NI for implementing the AES. Using AES-NI is the best option for implementing AES in processors where these instructions are available. AES-NI consists of the instructions AESENC, AESDEC, AESENCLAST, AESDECLAST. The last two instructions are to be used for the last round of AES encryption and decryption respectively and the first two are meant for the other rounds. In addition to these instructions there are instructions for key expansion.

To take advantage of the instruction level pipelining in the AES round instructions it is better if several calls to the same round of AES are clustered together [27]. In all the modes proposed in this paper the majority of the block cipher calls do not have any data dependency, hence it is possible to compute several AES calls at the same time. In our implementations, where ever possible we clustered eight different calls to a single round function.

10.3 Mask Generation on Small Processors

The Atmel AVR family of microcontrollers [17] support 8-bit and 32-bit words and the TI MSP430, MSP430X family of microcontrollers support 16-bit words [29]. On such processors, implementing 128-bit masking strategy with $n_1 = 128$ and $n_2 = 1$ is comparatively less efficient, the reason being the following. Consider the word size to be 32 bits. Then a 128-bit string has to be stored as four words. A left shift on the 128-bit string will requiring shifting out the msb of each word and placing it in the lsb of the next word. This typically requires quite a few instructions.

On the other hand, suppose for processors having 32-bit words, the masking strategy uses $n_1 = 32$ and $n_2 = 4$. Then generating the next mask can be performed using word level operations which leads to a significantly faster code. In fact, this was the motivation for introducing word-oriented LFSRs for designing stream ciphers such as SNOW [23] which are fast in software. Similarly, if the word size of the target processor is 8 bits, then the masking strategy with $n_1 = 8$ and $n_2 = 16$ will work best. This is one of the reasons that we do not fix the masking strategy.

11 Performance Results

Hardware and Software: Time was measured on a single core of a machine with the following configuration:

CPU:	Intel Core i7-3770K (IvyBridge) @ 3.50GHz (8 cores)
cache size:	8192 KB
Memory:	15.6 GB
OS:	Fedora release 18 (Spherical Cow)
Kernel:	Linux 3.11.10-100.fc18.x86_64
Compiler:	GCC version 4.7.2 20121109

Testing Methodology: For each of the schemes compilation was done with `-O3` optimization. The time was measured using the time stamp counter (TSC), which gets incremented with each CPU cycle. For each scheme X we did the following:

1. Ran X 1000 times on the same data, to ensure that the data and code gets into the cache.

2. Then we ran X 100 more times on the same data and measured the number of clock cycles taken by reading the TSC counter. We record the average time taken for each run.
3. Steps (1) and (2) are repeated 100 times and we report the median of these 100 runs.
4. The measurements are reported as cycles per byte, i.e., the total number of cycles divided by the total number of bytes. For authentication schemes, the total number of bytes consists of only the message which is to be authenticated; in case of authenticating a vector of strings, the total number of bytes consists of the total number of bytes in all the strings. For AE and DAE schemes, the total number of bytes consists of the number of bytes in the message to be encrypted. For AEAD and DAEAD schemes, the total number of bytes consists of the number of bytes in the message plus the total number of bytes in the header (which may be a vector of strings).

We have considered five types of maskings as indicated in Table 15. For each scheme and each masking type, we report the following three kinds of measurements.

Avg:	Average run time for messages of lengths 1 byte to 1024 bytes.
4KiB:	Run time for messages of length 4096 bytes.
ipi:	The internet performance index as introduced in [44] and used in [40].

We present performance figures of the different schemes in Tables 17 to 20. Analysis of performance figures for small messages with varying number of components for vector versions of the authentication and AEAD schemes are reported in Tables 21 to 23. Based on this data, we make the following observations.

1. Performance figures for Type-1 masking are consistently lower than the other types of maskings across all the tables, though the differences are quite small. This is due to the fact that the SSE2 code for Type-1 masking has the least number of instructions and also the least latency among all the five masking strategies.
2. From Table 17, the performance figures for PAuth are consistently lower than that for PAuth1 and PAuth2 and the same is true for the vector versions. The differences are again quite small. Explanation for this observation lies in the fact that both PAuth1 and PAuth2 use a small number of extra operations compared to PAuth.
3. For small messages, the overhead of the vector versions is substantial and this averages out for slightly longer messages. To see this, consider the figures for the vector versions of the authentication schemes in Table 17. The column 'Avg' reports the average over messages of lengths 1 to 1024 bytes and the performance figure is more than 9 cycles/byte which is substantially higher than the figure of around 2 cycles/byte for the single-input schemes. On the other hand, if we consider the columns '4KiB' and 'ipi', the performance figures for the vector versions go down quite sharply and the difference to the single-input version becomes considerably narrow. A similar effect in the difference of the performance figures between the single-input and the vector versions of the AEAD schemes can be observed from Tables 18 and 20.
4. The difference in the performance figures between authenticating a message and the authenticated encryption of a message of the same length is small. This can be observed from the figures for authentication given in Table 17 and that for AE given in Table 18.
5. Considering Tables 17 and 18, the figures for authentication or AE are higher than that for AEAD. Similarly, the performance figures for the vector versions of the authentication schemes are higher than that for the AEAD schemes. These may seem to be a counter-intuitive since for AEAD schemes more computation is required. To understand this, the first thing to note is that the tables provide cycles/byte figures. The explanation for the observation lies in the

fact that the figures for AEAD schemes correspond to a situation where the header size is 512 bytes. As a result, the total length of the message plus the header is no longer small and the cycles/byte computation averages out the overhead for the small length messages.

6. Considering Table 19, the performance figures for CTR is lower than the timings for Forward1 and Forward2. For small messages this gap is significant, but, becomes negligible as the length of the message grows. DAE (and also DAEAD) uses CTR whereas, the AE (and AEAD) schemes use either Forward1 or Forward2. Since DAE requires essentially two block cipher invocations per message block whereas the AE(AD) schemes require one such invocation, the speed of CTR being faster to a certain extent offsets this disadvantage.
7. From Table 19, the performance figures for Forward1 and PAE1 are the same and this is also true for Forward2 and PAE2. This is to be expected, since the AE algorithms essentially run the Forward algorithms for encryption.
8. From Table 19, for small messages, there is a significant difference in the performance figures between DAE and Ctr and the difference reduces for somewhat longer messages. A difference is to be expected since DAE does significantly more computation than CTR. The results show that for small messages, the overhead is substantial.
9. From Table 20, for vector versions of the AEAD schemes the performance figures increase with the increase in the number of components when the total length of all the components remain fixed. This is to be expected, since the fragmentation cannot benefit from the pipelining of the SSE2 instructions. Similarly, looking at Table 22, we find a similar effect of the number of components on the performance figures of the vector versions of the authentication schemes.
10. The effect of message lengths on the vector versions of the authentication schemes with a fixed number of components is shown in Table 21. These show that there is a substantial overhead for very small messages which drops off rather sharply as the message length becomes moderate.
11. Table 23 reports the average performance of the vector versions of AEAD schemes over small messages with a fixed number of components. These figures are to be compared with the performance figures in the column 'Avg' for the vector versions of the authentication schemes using Type-5 masking given in Table 17. These show that the additional cost of AEAD over authentication is small.
12. A general observation is that for small messages and also for messages with more number of components, the figures are higher. This is because for such messages, the pipeline for the AES-NI and the SSE2 instructions cannot be properly utilised.

Comparing performance to OCB: Performance results for OCB on modern Intel processors are given in [40] and [13]. Results in [40] are reported for the 'Clarkdale' processor while the results in [13] are reported for the latest 'Haswell' processor. On Clarkdale, OCB takes 1.48 cycles/byte (cpb) for 4K messages and 1.87 cpb for the ipi index [40]. For Haswell, [13] reports performance of 0.81 cpb for 2048 byte messages.

The Ivybridge processor on which we have performed our experiments is an intermediate design between Clarkdale and Haswell. Performance figures for PAE1 (and also PAE2) with Type-1 masking are 1.02 cpb and 1.51 cpb for 4096 byte messages and the ipi index respectively. For the other masking types, the maximum is for Type-5 masking where figures of 1.26 cpb and 1.75 cpb are respectively obtained for 4096 byte messages and the ipi index. The figures for PAE1 for all the different masking options are intermediate between what has been reported for OCB on Clarkdale and Haswell. This indicates that the performance of the PAE1 compares well with that of OCB. We mention, though, that on the same high-end Intel processor platform, we expect OCB to be faster than PAE1 by a fraction of a cpb. This small speed loss should be tolerable in view of the several

	Type-1			Type-2			Type-3			Type-4			Type-5		
	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi
PAuth	2.05	0.96	1.32	2.21	1.13	1.48	2.21	1.14	1.48	2.29	1.25	1.58	2.31	1.24	1.59
PAuth1	2.20	1.11	1.49	2.43	1.35	1.71	2.39	1.28	1.67	2.52	1.44	1.81	2.55	1.47	1.84
PAuth2	2.22	1.11	1.50	2.40	1.30	1.68	2.47	1.37	1.74	2.53	1.47	1.83	2.60	1.56	1.91
$\overrightarrow{\mathsf{PAuth}}$	9.10	3.06	3.76	9.28	3.26	4.11	9.28	3.27	3.95	9.37	3.38	4.11	9.35	3.40	4.11
$\overrightarrow{\mathsf{PAuth1}}$	9.20	3.26	3.93	9.37	3.42	4.16	9.40	3.48	4.14	9.43	3.62	4.26	9.56	3.64	4.28
$\overrightarrow{\mathsf{PAuth2}}$	9.26	3.27	3.94	9.36	3.42	4.17	9.51	3.58	4.21	9.51	3.62	4.26	9.58	3.64	4.29

Table 17. Performance figures in cycles/byte for the different authentication schemes and their vector versions. For the measurements of $\overrightarrow{\mathsf{PAuth}}$, $\overrightarrow{\mathsf{PAuth1}}$ and $\overrightarrow{\mathsf{PAuth2}}$, a message was divided into four almost equal length portions and the algorithms were invoked on the resulting 4-component vector.

	Type-1			Type-2			Type-3			Type-4			Type-5		
	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi
PAE1	2.79	1.02	1.51	2.94	1.17	1.66	2.95	1.19	1.68	2.99	1.21	1.72	3.01	1.26	1.75
PAE2	2.79	1.02	1.51	2.94	1.17	1.67	2.96	1.19	1.69	3.00	1.21	1.72	3.03	1.26	1.75
DAE	3.88	1.70	2.41	4.05	1.90	2.60	4.07	1.88	2.58	4.12	1.96	2.66	4.17	1.98	2.70
PAEAD1	1.49	1.07	1.31	1.63	1.22	1.46	1.67	1.27	1.51	1.74	1.27	1.55	1.77	1.31	1.59
PAEAD2	1.49	1.07	1.31	1.63	1.22	1.47	1.71	1.25	1.53	1.74	1.27	1.55	1.76	1.31	1.58
DAEAD	4.19	3.65	4.06	4.34	3.84	4.28	4.94	4.57	4.80	4.41	3.91	4.30	5.06	4.67	4.94

Table 18. Performance figures in cycles/byte for encryption using the AE and AEAD schemes. For the AEAD schemes, a single header of length 512 bytes was used.

	32	64	128	256	512	1024	2048	4096
PAE1	7.07	3.91	1.79	1.39	1.19	1.09	1.05	1.03
PAE2	7.08	3.91	1.78	1.38	1.19	1.09	1.05	1.03
DAE	11.95	6.99	3.68	2.71	2.17	1.91	1.77	1.70
Forward1	7.07	3.91	1.79	1.38	1.19	1.09	1.04	1.03
Forward2	7.09	3.91	1.78	1.39	1.19	1.09	1.04	1.03
Ctr	3.17	2.34	0.81	0.76	0.74	0.75	0.73	0.73

Table 19. Performance figures in cycles/byte for encryption using PAE1, PAE2 and DAE for different msg lengths with Type-1 masking. For comparison, we also report figures for Forward1, Forward2 and Ctr (as used in DAE).

	1 hdr (512)			2 hdrs (256+256)			3 hdrs $(256 + 2 \times 128)$			4 hdrs (4×128)			8 hdrs (8×64)		
	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi
$\overrightarrow{\mathsf{PAEAD1}}$	3.08	1.38	2.37	3.24	1.41	2.47	3.43	1.46	2.61	3.59	1.48	2.71	4.78	1.72	3.49
$\overrightarrow{\mathsf{PAEAD2}}$	3.10	1.38	2.43	3.26	1.41	2.50	3.44	1.46	2.63	3.61	1.49	2.73	4.80	1.73	3.51
DAEAD	4.09	3.63	3.99	4.29	3.68	4.11	4.48	3.72	4.21	4.61	3.76	4.30	6.04	4.04	5.24

Table 20. Performance figures in cycles/byte for encryption using $\overrightarrow{\mathsf{PAEAD1}}$, $\overrightarrow{\mathsf{PAEAD2}}$ and DAEAD where the number of headers vary. Type-1 masking (i.e. $n_1 = 128$ and $n_2 = 1$) has been used.

	1-64	65-128	129-256	257-512	513-1024
$\overrightarrow{\mathsf{PAuth}}$	47.15	17.05	9.96	6.73	4.88
$\overrightarrow{\mathsf{PAuth1}}$	47.80	17.11	9.99	6.71	5.15
$\overrightarrow{\mathsf{PAuth2}}$	47.84	17.29	10.08	6.78	5.15

Table 21. Average of cycles/byte figures for authentication of small messages using the vector versions of the authentication schemes. The column headings mention the lengths in bytes over which the average has been computed. Each message has been divided almost equally into four components and the algorithms were invoked on the resulting 4-component vector. Type-5 masking was used in each case.

	512	256+256	$256 + 2 \times 128$	4×128	8×64
$\overrightarrow{\mathsf{PAuth}}$	3.95	4.38	4.73	5.10	7.65
$\overrightarrow{\mathsf{PAuth1}}$	4.20	4.62	4.95	5.29	7.60
$\overrightarrow{\mathsf{PAuth2}}$	4.20	4.61	4.93	5.31	7.70

Table 22. Average of cycles/byte figures for authentication of small messages using vector versions of the authentication schemes with varying number of components. The column headings indicate the number and the lengths in bytes of the different components. Type-5 masking was used in each case.

$\overrightarrow{\mathsf{PAEAD1}}$	10.45
$\overrightarrow{\mathsf{PAEAD2}}$	10.27

Table 23. Average of cycles/byte figures for encryption using the vector versions of the AEAD schemes with lengths varying from 1 byte to 1024 bytes with a string of a particular length been divided almost equally into one message and three headers. Type-5 masking was used in each case.

features including reconfigurability mentioned in Section 9 and the fact that there are no IP claims on the schemes in this work.

12 Conclusion

In this work, we have presented a suite of schemes for a variety of tasks of encryption and authentication that have been defined in the literature. The constructions have a common unifying theme which make them suitable for a joint description. Implementation details and performance results are presented. These indicate that the schemes compare well with existing works and provide a designer with additional flexibility in choosing a particular scheme for implementation.

References

1. Intel 64 and IA-32 architectures optimization reference manual. available as `http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf`, 2014.
2. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and authenticated online ciphers. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2013.
3. Elena Andreeva, Atul Luykx, Bart Mennink, and Kan Yasuda. COBRA: A Parallelizable Authenticated Online Cipher Without Block Cipher Inverse. In *FSE*, 2014. to appear.
4. Kazumaro Aoki, Tetsu Iwata, and Kan Yasuda. How fast can a two-pass mode go? a parallel deterministic authenticated encryption mode for AES-NI. Directions in Authenticated Ciphers, workshop records, 2012.
5. Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online ciphers and the hash-cbc construction. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2001.

6. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.
7. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
8. Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.
9. 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.
10. 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.
11. John Black and Phillip Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2000.
12. John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 384–397. Springer, 2002.
13. Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. Aes-based authenticated encryption modes in parallel high-performance software. Cryptology ePrint Archive, Report 2014/186, 2014. `http://eprint.iacr.org/`.
14. CAESAR. Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`.
15. 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.
16. Debrup Chakraborty and Palash Sarkar. 'C' Code for Reference and Fast Implementations of Various Block Cipher Based Modes of Operations. `https://drive.google.com/file/d/0B7cNoZ_Dy-EhZFhCdFU2emNPQkU/edit?usp=sharing`, 2014.
17. Atmel Corporation. Atmel AVR 8-bit and 32-bit Microcontrollers. `http://www.atmel.in/products/microcontrollers/Avr/`, 2014. Accessed on 30th July, 2014.
18. Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES – The Advanced Encryption Standard (Information Security and Cryptography)*. Springer, Heidelberg, 2002.
19. Nilanjan Datta and Mridul Nandi. ELmD. submission to CAESAR `http://competitions.cr.yp.to/caesar-submissions.html`, 2014.
20. Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.
21. M. Dworkin. Recommendation for block cipher modes of operations: the CMAC mode for authentication, May 2005. National Institute of Standards and Technology, U.S. Department of Commerce. NIST Special Publication 800-38B.
22. Morris Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC, November 2011. NIST Special Publication 800-38D, `csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf`.
23. 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.
24. Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.
25. Edgar N. Gilbert, F. Jessie MacWilliams, and Neil J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53:405–424, 1974.
26. 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.
27. Shay Gueron. Intel's new aes instructions for enhanced performance and security. In Dunkelman [20], pages 51–66.
28. Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.
29. Texas Instruments. MSP 16-bit and 32-bit Microcontrollers. `http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/overview.page`, 2014. Accessed on 30th July, 2014.

30. Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.
31. Tetsu Iwata and Kaoru Kurosawa. Stronger security bounds for omac, tmac, and xcbc. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT*, volume 2904 of *Lecture Notes in Computer Science*, pages 402–415. Springer, 2003.
32. Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated Encryption for Short Input. In *FSE*, 2014. to appear.
33. Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. SILC:Simple Lightweight CFB. submission to CAESAR `http://competitions.cr.yp.to/caesar-submissions.html`, 2014.
34. Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012.
35. Tetsu Iwata and Kan Yasuda. Btm: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009.
36. Tetsu Iwata and Kan Yasuda. Hbs: A single-key mode of operation for deterministic authenticated encryption. In Dunkelman [20], pages 394–415.
37. 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.
38. Jonathan Katz and Moti Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC*, pages 245–254, 2000.
39. Ted Krovetz. HS1-SIV. submission to CAESAR `http://competitions.cr.yp.to/caesar-submissions.html`, 2014.
40. Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.
41. Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-Key CBC MAC. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2003.
42. R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications, revised edition.* Cambridge University Press, 1994.
43. David McGrew, Scott Fluhrer, Stefan Lucks, Christian Forler, Jakob Wenzel, Farzaneh Abed, and Eik List. Pipelineable on-line encryption. In *FSE*, 2014. to appear.
44. 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.
45. Alfred Menezes, Paul Van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
46. Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In *Crypto*, 2014. to appear.
47. Kazuhiko Minematsu, Stefan Lucks, and Tetsu Iwata. Improved Authenticity Bound of EAX, and Refinements. In Willy Susilo and Reza Reyhanitabar, editors, *ProvSec*, volume 8209 of *Lecture Notes in Computer Science*, pages 184–201. Springer, 2013.
48. Kazuhiko Minematsu, Hiraku Morita, and Tetsu Iwata. Cryptanalysis of EAXprime. *IACR Cryptology ePrint Archive*, 2012:18, 2012.
49. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.
50. 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.
51. 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.
52. 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.
53. Palash Sarkar. Pseudo-random functions and parallelizable modes of operations of a block cipher. *IEEE Transactions on Information Theory*, 56(8):4025–4037, 2010.
54. Palash Sarkar. A simple and generic construction of authenticated encryption with associated data. *ACM Trans. Inf. Syst. Secur.*, 13(4):33, 2010.
55. Palash Sarkar. Modes of operations for encryption and authentication using stream ciphers supporting an initialisation vector. *Cryptography and Communications - Discrete Structures, Boolean Functions and Sequences*, 6(3):189–231, September 2014.

56. 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.
57. Advanced Encryption Standard. Federal Information Processing Standard Publication 197, 2002. Available at `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
58. American National Standard Protocol Specification For Interfacing to Data Communication Networks. ANSI C12.22-2008, 2008.
59. Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.
60. 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.
61. Doug Whiting, Russ Housley, and Niels Ferguson. Counter with CBC-MAC (CCM). available as `http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf`, 2003.

A Notation and Terminology for the Security Proofs

Let f be a random function with domain $\mathcal{X}$. A collision for f consists of two distinct $x, x' \in \mathcal{X}$ such that $f(x) = f(x')$. The function f is said to be ε-collision resistant (ε-CR) if, for any two distinct x and x', $\Pr[f(x) = f(x')] \leq \varepsilon$. For certain types of functions, the upper bound may not be a constant and would depend on the length of the inputs. This is captured as follows. The function f is said to be ε-CR, if for any two distinct x and x', $\Pr[f(x) = f(x')] \leq \varepsilon \max(m, m')$, where m and m' are respectively obtained from calls to $\mathsf{Format}(x)$ and $\mathsf{Format}(x')$.

For the security analysis, we follow the approach used in [53]. The functions PAuth and the associated authentication functions of the AEAD modes of operations have the following general structure. Given an input x, a bijection π is applied to different strings; the final output of the function is also the output of the invocation of π on some n-bit string Z. Let f denote the function which maps x to $\pi(Z)$, i.e., $f(x) = \pi(Z)$ and let f_1 denote the function which takes as input x and produces Z, i.e., $f_1(x) = Z$. So, $f(x) = \pi(f_1(x))$. Following [53], this structure is formalised by the next definition.

Definition 2. *Let $\pi : \mathcal{Y} \to \mathcal{Y}$ be a uniform random bijection. A function $f : \mathcal{X} \to \mathcal{Y}$ is said to be a domain extender for π if $f = \pi \circ f_1^{(\pi)}$, where $f_1 : \mathcal{X} \to \mathcal{Y}$ and f_1 satisfies the following conditions.*

1. *On any input, f_1 invokes π a finite number of times.*
2. *The only randomness involved in computing f_1 comes from the invocations of π.*

When π is clear from the context, we will write f_1 instead of $f_1^{(\pi)}$.

Suppose $f = \pi \circ f_1^{(\pi)}$ is applied to q distinct inputs $x_1, \ldots, x_q$. This requires application of π to different substrings. Suppose $Z_i = f_1^{(\pi)}(x_i)$ and $f(x_i) = \pi(Z_i)$. The basic intuition to showing PRF property is that $Z_1, \ldots, Z_q$ are distinct and "fresh", i.e., they have not occurred as previous inputs to π during the computations of $f_1^{(\pi)}$ on the inputs $x_1, \ldots, x_q$. Then the outputs of π on $Z_1, \ldots, Z_q$ are uniformly distributed and independent of all previous values and are also independent of each other. As a result, these outputs are indistinguishable from the outputs of a uniform random function on inputs $x_1, \ldots, x_q$.

The actual analysis consists of determining the probability that $Z_1, \ldots, Z_q$ are distinct and "fresh". This amounts to considering collisions between these values and also between these values and the inputs to π during the computations of $f_1^{(\pi)}$ on $x_1, \ldots, x_q$. The collision analysis can be broken down into simpler terms. We follow the approach used in [53].

Definition 3. *Let $\pi : \mathcal{Y} \to \mathcal{Y}$ be a random bijection and $f = \pi \circ f_1^{(\pi)}$ be a map from $\mathcal{X}$ to $\mathcal{Y}$ satisfying Definition 2. For $x, x' \in \mathcal{X}$ with $x \neq x'$, let $Z = f_1(x)$, $Z' = f_1(x')$; and let $U_1, \ldots, U_m$ and $U'_1, \ldots, U'_{m'}$ be the inputs to the different invocations of π in the computation of $f_1(x)$ and $f_1(x')$ respectively.*

1. *Define $\mathsf{Self\text{-}Disjoint}(x)$ to be the event $\bigwedge_{i=1}^{m}(Z \neq U_i)$.*
2. *Define $\mathsf{Pairwise\text{-}Disjoint}(x, x')$ to be the event $\left(\bigwedge_{i=1}^{m}(Z' \neq U_i) \wedge \bigwedge_{j=1}^{m'}(Z \neq U'_i)\right)$.*

Definition 4. *Continuing with Definition 3, we say that f_1 is $(\varepsilon_1, \varepsilon_2)$-disjoint, if for all pairs of distinct $x, x' \in \mathcal{X}$,*

$$\Pr[\overline{\mathsf{Self\text{-}Disjoint}(x)}] \leq \varepsilon_1(m+1) \textit{ and } \Pr[\overline{\mathsf{Pairwise\text{-}Disjoint}(x, x')}] \leq \varepsilon_2(m + m' + 2).$$

The following result relates the collision probabilities to the PRF-advantage and follows directly from Theorem 2 in [53].

Theorem 8. *Let $\pi : \{0,1\}^n \to \{0,1\}^n$ be a uniform random permutation and $f = \pi \circ f_1^{(\pi)}$ be a map from $\mathcal{X}$ to $\mathcal{Y}$ satisfying Definition 2. Suppose that f_1 is ε-CR and also $(\varepsilon_1, \varepsilon_2)$-disjoint. Then for positive integers q and $\sigma \geq q$*

$$\mathbf{Adv}_f^{\mathrm{prf}}(q, \sigma) \leq \sigma(q\varepsilon + \varepsilon_1 + 2q\varepsilon_2) + \frac{q\sigma}{2^n}.$$

B Analysis of PAuth

The proof is based on Theorem 8 which in turn is based on bounding the probabilities of certain types of collisions.

For the analysis it is helpful to have a different description of PAuth. Let $\mathsf{PAuth}_{\pi,\mathsf{fStr}} : P \mapsto C_m$ where $\gamma = \pi(\mathsf{fStr})$, $C_i = \pi(D_i)$ for $1 \leq i \leq m$ and

$$D_i = \begin{cases} P_1 \oplus \Gamma_{\gamma,-1} & i = m = 1, r < n; \\ P_1 \oplus \Gamma_{\gamma,-2} & i = m = 1, r = n; \\ P_i \oplus \Gamma_{\gamma,i} & m > 1, 1 \leq i \leq m-1; \\ C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m & m > 1, i = m, r = n; \\ C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \Gamma_{\gamma,m} & m > 1, i = m, r < n. \end{cases} \tag{17}$$

Define the function PHash as $\mathsf{PHash}_{\pi,\mathsf{fStr}} : P \mapsto D_m$. Then $\mathsf{PAuth}_{\pi,\mathsf{fStr}}(P) = \pi(\mathsf{PHash}_{\pi,\mathsf{fStr}}(P))$. In computing D_m, $\mathsf{PHash}_{\pi,\mathsf{fStr}}$ invokes π a total of m times: once on fStr and $m-1$ times on $D_1, \ldots, D_{m-1}$. To apply Theorem 8, we need to determine ε, ε_1 and ε_2 such that f_1 is ε-CR and $(\varepsilon_1, \varepsilon_2)$-disjoint. The following lemmas perform this task.

Lemma 1. *Let x and x' be two distinct messages and m and m' are defined by* Format *respectively for x and x'. Suppose x and x' are mapped to D_m and $D'_{m'}$ under* PHash*. Assume that $m + m' - 3 \leq 2^{n-1}$. Then $\Pr[D_m = D'_{m'}] \leq (m + m')/2^n \leq 2\max(m, m')/2^n$.*

Proof : Assume without loss of generality that $m \geq m'$. We start by assuming that both m and m' are greater than one. The case when at least one of them is one is considered later. There are four cases depending on whether r and r' are less than n or equal to n.

Case $r = n$, $r' = n$: Since $x \neq x'$, let j be the first index such that either ($1 \leq j \leq m'$ and $P_j \neq P'_j$) or ($j = m' + 1$ and $P_i = P'_i$ for $1 \leq i \leq m'$).

If $j = m = m'$, then $P_i = P'_i$ for $1 \leq i \leq m' - 1$ and so $C_i = C'_i$ for $1 \leq i \leq m - 1$. So, $D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \neq C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_m = D'_{m'}$ and $\Pr[D_m = D'_{m'}] = 0$.

If $j = m = m' + 1$, then $P_i = P'_i$ for $1 \leq i \leq m' - 1$ and so $C_i = C'_i$ for $1 \leq i \leq m' - 1$. So,

$$\begin{aligned} D_m \oplus D'_{m'} &= C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'} \\ &= C_{m-1} \oplus P_m \oplus P'_{m'} \end{aligned}$$

Since C_{m-1} is the output of π, it is uniformly distributed over $\{0,1\}^n$ and hence, the last expression is zero with probability $1/2^n$.

So, we can assume that either ($m > m' + 1$, $j = m' + 1$) or ($1 \leq j \leq m'$ and $m > m'$). In either case, $D_j = \Gamma_{\gamma,j} + P_j$. We claim that with high probability D_j is different from

$D_1, \ldots, D_{j-1}, D_{j+1}, \ldots, D_{m-1}$ and $D'_1, \ldots, D'_{m'-1}$. To see this, first note that $D_i = P_i \oplus \varGamma_{\gamma,i}$, $1 \le i \le m-1$; and $D'_k = P'_k \oplus \varGamma_{\gamma,k}$, $1 \le k \le m'-1$. Let $\mathcal{E}$ be the event

$$\mathcal{E} : \left(\bigwedge_{\substack{i=1,\\ i \neq j}}^{m-1} (D_j \neq D_i) \right) \wedge \left(\bigwedge_{i=1}^{m'-1} (D_j \neq D'_i) \right).$$

In other words, the event $\mathcal{E}$ happens when D_j is distinct from all other D_i's and is also distinct from $D'_1, \ldots, D'_{m'-1}$. We first show that $\mathcal{E}$ occurs with high probability.

$$\begin{aligned}\Pr[\mathcal{E}] &= 1 - \Pr[\overline{\mathcal{E}}] \\ &\ge 1 - \sum_{\substack{i=1,\\ i \neq j}}^{m-1} \Pr[D_j = D_i] - \sum_{i=1}^{m'-1} \Pr[D_j = D'_i].\end{aligned}$$

If $j < m'$, then since $P_j \neq P'_j$, $D_j = P_j \oplus \varGamma_{\gamma,j} \neq P'_j \oplus \varGamma_{\gamma,j} = D'_j$ so that $\Pr[D_j = D'_j] = 0$. In all other cases, the individual probabilities of either $D_j = D'_i$ or $D_j = D_i$ for $i \neq j$ are $1/2^n$ by the properties of $\varGamma$ given in Definition 1. So,

$$\Pr[\mathcal{E}] \ge \left(1 - \frac{m + m' - 3}{2^n}\right).$$

We have

$$\begin{aligned}\Pr[D_m \neq D'_{m'}] &\ge \Pr[(D_m \neq D'_{m'}) \wedge \mathcal{E}] \\ &= \Pr[(D_m \neq D'_{m'}) | \mathcal{E}] \Pr[\mathcal{E}] \\ &\ge \Pr[(D_m \neq D'_{m'}) | \mathcal{E}] \times \left(1 - \frac{m + m' - 3}{2^n}\right). \qquad (18)\end{aligned}$$

Consider the event $D_m \neq D'_{m'}$ conditioned upon the event $\mathcal{E}$. Since π is a permutation and D_j is distinct from all other D_is and $D'_1, \ldots, D'_{m'-1}$, we have that C_j is distinct from all other C_is and $C'_1, \ldots, C'_{m'-1}$.

Since $r = r' = n$, we have

$$\begin{aligned}D_m &= C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \\ D'_{m'} &= C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'}.\end{aligned}$$

Consider the set of random variables.

$$\{C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_{m-1}, C'_1, \ldots, C'_{m'-1}\}.$$

Some of the random variables in this set can be equal. We are interested in a subset of random variables taking equal values only if the number of elements in this subset is odd. Let there be $t \ge 0$ such subsets and $Q_1, \ldots, Q_t$ be random variables where each Q_i is the XOR of the random variables in each subset. Note that $t \le m + m' - 3$. So, $D_m \oplus D'_{m'} = 0$ implies that $C_j \oplus Q_1 \oplus \ldots \oplus Q_t = P_m \oplus P'_{m'}$ for some $t \ge 0$ and $(C_j, Q_1, \ldots, Q_t)$ is distributed uniformly over $\chi_{t+1}(\mathcal{Y})$.

1. If $t = 0$, then $\Pr[D_m \neq D'_{m'} | \mathcal{E}] = \Pr[C_j \neq P_m \oplus P'_{m'} | \mathcal{E}] = (1 - 1/2^n)$.
2. If $t = 1$ and $P_m = P'_{m'}$, then $\Pr[D_m \neq D'_{m'} | \mathcal{E}] = \Pr[C_j \neq Q_t | \mathcal{E}] = 1$.
3. In all other cases, $\Pr[D_m \neq D'_{m'} | \mathcal{E}] = \Pr[C_j \oplus Q_1 \oplus \cdots \oplus Q_t \neq P_m \oplus P'_{m'} | \mathcal{E}] \ge 1 - 1/(2^n - t) \ge 1 - 1/(2^n - (m + m' - 3)) \ge 1 - 2/2^n$ (assuming $m + m' - 3 \le 2^{n-1}$).

Thus, the inequality, $\Pr[D_m \neq D'_{m'} | \mathcal{E}] \ge 1 - 2/2^n$ holds for all t.

From this and (18) we have $\Pr[D_m \neq D'_{m'}] \ge (1 - (m + m' - 1)/2^n)$ and so $\Pr[D_m = D'_{m'}] \le (m + m')/2^n$.

Case $r < n$, $r' < n$: In this case, we have

$$D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \mathit{\Gamma}_{\gamma,m}$$
$$D'_{m'} = C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'} \oplus \mathit{\Gamma}_{\gamma,m'}.$$

If $m = m'$, then the terms involving the $\mathit{\Gamma}$'s cancel out and the analysis is exactly the same as that for the case $r = r' = n$. (If $r \neq r'$, then Format ensures that the last blocks are distinct, i.e., $P_m \neq P'_{m'}$. If $P_m = P'_{m'}$ (and so necessarily $r = r'$), then there is an i with $1 \leq i \leq m' - 1$, such that $P_i = P'_i$.)

So suppose $m > m'$. Let $\mathcal{E}$ be the event that fStr is not equal to any of $D_1, \ldots, D_{m-1}$ or $D'_1, \ldots, D'_{m'-1}$. The probability of $\mathcal{E}$ is at least $1 - (m + m' - 2)/2^n$. In a manner similar to the previous case, it can be shown $\Pr[D_m \neq D'_{m'}|\mathcal{E}] \geq 1 - 2/2^n$ so that we again have $\Pr[D_m = D'_{m'}] \leq (m + m')/2^n$.

Cases ($r = n$, $r' < n$) and ($r < n$, $r' = n$): Both the cases are similar and we consider only $r = n$ and $r' < n$. In this case, we have

$$D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \mathit{\Gamma}_{\gamma,m}$$
$$D'_{m'} = C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'}.$$

It is possible that $m = m'$ and $P_i = P'_i$ for $1 \leq i \leq m$ even though $x \neq x'$. This happens when $x = \mathsf{pad}(x') \neq x'$. Then, $D_m \oplus D'_{m'} = \mathit{\Gamma}_{\gamma,m}$ which is equal to 0 with probability $1/2^n$. If $m > m'$ or $P_i \neq P'_i$ for some $1 \leq i \leq m'$, then an analysis similar to the previous case shows the desired result.

Now we consider the case where at least one of m or m' is equal to 1.

At least one of m or m' is equal to 1. If $m = m' = 1$ and $r = r' = n$, then $D_1 \oplus D'_1 = P_1 \oplus P'_1$. By the condition that the queries must be distinct, it follows that $P_1 \neq P'_1$ and so the probability that D_1 equals D'_1 is zero.

If $m = m' = 1$ and $r = n$, $r' < n$, then $D_1 \oplus D'_1 = P_1 \oplus P'_1 \oplus \mathit{\Gamma}_{\gamma,-2} \oplus \mathit{\Gamma}_{\gamma,-1}$. By Definition 1, it follows that the probability of this event is at most $1/2^n$.

If $m' = 1$ and $m > 1$, then $D'_1 = P'_1 \oplus \mathit{\Upsilon}$ where $\mathit{\Upsilon}$ is $\mathit{\Gamma}_{\gamma,-1}$ if $r' < n$ and $\mathit{\Upsilon}$ is $\mathit{\Gamma}_{\gamma,-2}$ if $r' = n$; and $D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \mathit{\Lambda}$, where $\mathit{\Lambda}$ is either 0^n or $\mathit{\Gamma}_m$ according as $r = n$ or $r < n$. These give rise to four cases and in all these four cases, the properties of the $\mathit{\Gamma}$'s guaranteed by Definition 1 ensure that $\Pr[D_m = D'_1] \leq 2m/2^n$. □

The disjointness probabilities can be bound in a similar manner and is given by the following result.

Lemma 2. *Let x and x' be two distinct messages having m and m' blocks respectively. Then*

1. $\Pr[D_m = D'_i] \leq 2/2^n$ *for* $1 \leq i \leq m' - 1$;
2. $\Pr[D_m = D_i] \leq 2/2^n$ *for* $1 \leq i \leq m - 1$;
3. $\Pr[D_m = \mathsf{fStr}] \leq 1/2^n$.

Proof : First suppose $m = 1$. Then $D_1 = P_1 \oplus \mathit{\Gamma}_{\gamma,-1}$ or $D_1 = P_1 \oplus \mathit{\Gamma}_{\gamma,-2}$ according as $r < n$ or $r = n$. Point 3 follows from this. For $1 \leq i \leq m' - 1$, $D'_i = P'_i \oplus \mathit{\Gamma}_{\gamma,i}$. So from Definition 1, $\Pr[D_1 = D'_i] \leq 2/2^n$ which proves Point 1. For $m = 1$, Point 2 is vacuous.

If $m > 1$, then $D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m$ or $D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \mathit{\Gamma}_{\gamma,m}$ according as $r = n$ or $r > n$. A straightforward analysis now shows the result. □

Consequently, $\Pr[\overline{\mathsf{Pairwise\text{-}Disjoint}(x, x')}] \leq (m+m')/2^n$ and $\Pr[\overline{\mathsf{Self\text{-}Disjoint}(x)}] \leq 2m/2^n$. Using Theorem 8 with $\varepsilon = \varepsilon_1 = \varepsilon_2 = 2/2^n$ gives Theorem 2.

C Security Arguments for the Authenticated Encryption Schemes

We first consider the analysis of PAE1. The analysis of PAE2 is similar and is briefly discussed later. The analysis of PAE1 consists of two parts – analysis of privacy and that of authenticity.

C.1 Privacy of PAE1

Let $\mathcal{A}$ be a (q, σ)-adversary, i.e., $\mathcal{A}$ makes a total of q queries and provides a total of σ n-bit blocks in all the queries. This also includes the n-bit blocks for the nonces. Recall that $\mathcal{A}$ is restricted to be nonce-respecting, i.e., $\mathcal{A}$ cannot repeat a nonce.

The s-th query is of the form $(N^{(s)}, P^{(s)})$ and gets back $(C^{(s)}, \mathsf{tag}^{(s)})$ where $\mathsf{len}(P^{(s)}) = \mathsf{len}(C^{(s)})$. Note that the output of $\mathsf{Format}(P^{(s)}, n)$ is $(P_1^{(s)}, \ldots, P_{m^{(s)}}^{(s)})$ and the output of $\mathsf{Format}(C^{(s)}, n)$ is $(C_1^{(s)}, \ldots, C_{m^{(s)}}^{(s)})$.

For $1 \leq s \leq q$ and $0 \leq i \leq m^{(s)} + 1$, define

$$A_i^{(s)} = \begin{cases} \gamma^{(s)} & \text{if } i = 0; \\ P_i^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},i} & \text{if } 1 \leq i \leq m^{(s)} - 1 \text{ and } m^{(s)} > 1; \\ P_{m^{(s)}}^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}} & \text{if } i = m^{(s)}, m^{(s)} > 1 \text{ and } r^{(s)} = n; \\ P_1^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},-1} & \text{if } i = m^{(s)} = 1 \text{ and } r^{(s)} = n; \\ \mathsf{tmp}^{(s)} & \text{if } i = m^{(s)} \text{ and } r^{(s)} < n; \\ \mathsf{tag}^{(s)} & \text{if } i = m^{(s)} + 1. \end{cases}$$

$$B_i^{(s)} = \begin{cases} N^{(s)} & \text{if } i = 0; \\ C_i^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},i} & \text{if } 1 \leq i \leq m^{(s)} - 1 \text{ and } m^{(s)} > 1; \\ C_{m^{(s)}}^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}} & \text{if } i = m^{(s)}, m^{(s)} > 1 \text{ and } r^{(s)} = n; \\ \mathsf{bin}_n(r^{(s)}) \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}} & \text{if } i = m^{(s)}, m^{(s)} > 1 \text{ and } r^{(s)} < n; \\ \left.\begin{array}{l} A_1^{(s)} \oplus \cdots \oplus A_{m^{(s)}-1}^{(s)} \\ \oplus \mathit{\Gamma}_{\gamma^{(s)},1} \oplus \cdots \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}-1} \oplus C_{m^{(s)}}^{(s)} \end{array}\right\} & \text{if } i = m^{(s)} + 1, m^{(s)} > 1 \text{ and } r^{(s)} = n; \\ \left.\begin{array}{l} A_1^{(s)} \oplus \cdots \oplus A_{m^{(s)}-1}^{(s)} \\ \oplus \mathit{\Gamma}_{\gamma^{(s)},1} \oplus \cdots \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}-1} \oplus C_{m^{(s)}}^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}+1} \end{array}\right\} & \text{if } i = m^{(s)} + 1, m^{(s)} > 1 \text{ and } r^{(s)} < n; \\ C_1^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},-1} & \text{if } i = 1, m^{(s)} = 1 \text{ and } r^{(s)} = n; \\ \mathsf{bin}_n(r^{(s)}) \oplus \mathit{\Gamma}_{\gamma^{(s)},-1} & \text{if } i = 1, m^{(s)} = 1 \text{ and } r^{(s)} < n. \\ C_1^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},-3} & \text{if } i = 2, m^{(s)} = 1 \text{ and } r^{(s)} = n; \\ C_1^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},-2} & \text{if } i = 2, m^{(s)} = 1 \text{ and } r^{(s)} < n. \end{cases}$$

We define the following sets of random variables.

$$\mathcal{D}^{(s)} = \left\{A_0^{(s)}, \ldots, A_{m^{(s)}+1}^{(s)}\right\}; \quad \mathcal{R}^{(s)} = \left\{B_0^{(s)}, \ldots, B_{m^{(s)}+1}^{(s)}\right\};$$

$$\mathcal{D} = \bigcup_{s=1}^{q} \mathcal{D}^{(s)}; \quad \mathcal{R} = \bigcup_{s=1}^{q} \mathcal{R}^{(s)}.$$

The number of elements in either of $\mathcal{D}$ or $\mathcal{R}$ equals $\sum_{s=1}^{q}(m^{(s)} + 2) \leq \sigma + 2q$. Note that σ is the query complexity which is the total number of n-bit blocks provided by the adversary in all its queries and so $\sigma = \sum_{s=1}^{q}(m^{(s)} + 1)$.

Assume that for any query, the quantities $C_1^{(s)}, \ldots, C_{m^{(s)}}^{(s)}, \mathsf{tag}^{(s)}$ are chosen as follows:

1. $C_1^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, \mathsf{tag}^{(s)}$ are chosen uniformly at random and independent of previous choices.
2. Further, if $r^{(s)} = n$, then $C_{m^{(s)}}^{(s)}$ is chosen uniformly and independently at random; if $r^{(s)} < n$, $\mathsf{tmp}^{(s)}$ is chosen independently and uniformly at random and $C_{m^{(s)}}^{(s)}$ is set to $\mathsf{First}_{r^{(s)}}(P_{m^{(s)}} \oplus \mathsf{tmp}^{(s)})$.

The quantities $C_1^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, C_{m^{(s)}}^{(s)}, \mathsf{tag}^{(s)}$ are returned to the adversary. Let $\mathsf{Coll}(\mathcal{D})$ be the event that two random variables in $\mathcal{D}$ take the same value and similarly define $\mathsf{Coll}(\mathcal{R})$. Further, let $\mathsf{Coll} = \mathsf{Coll}(\mathcal{D}) \vee \mathsf{Coll}(\mathcal{R})$. As is standard, it is possible to show that

$$\mathsf{Adv}(\mathcal{A}) \leq \Pr[\mathsf{Coll}].$$

The task now reduces to bounding the probability of Coll. Note that, $\gamma^{(s)} = \pi(N^{(s)})$. Since the adversary is nonce-respecting, the values $N^{(s)}$ are distinct so that applying the uniform random permutation π on these q values ensures that each $\gamma^{(s)}$ is uniformly distributed over $\mathbb{F}$ and the joint distribution of the $\gamma^{(s)}$s is uniform over $\chi_q(\mathbb{F})$. So, the probability that two of the Γ's are equal is at most $1/(2^n - 1) \leq 1/2^{n-1}$. Further, $\Gamma_{\gamma^{(s)},i} = \psi^i(\gamma^{(s)})$, i.e., $\Gamma_{\gamma^{(s)},i}$ depends on the actual value of the nonce $N^{(s)}$ provided in the s-th query. The properties of ψ from Definition 1 shows that for $1 \leq i < j \leq 2^n - 2$ and for any $\beta \in \mathbb{F}$, $\Pr[\Gamma_{\gamma^{(s)},i} \oplus \Gamma_{\gamma^{(s)},j} = \beta] = 1/2^n$.

Using the randomness of the γ's, the randomness of the C's and the randomness of the tag's, it is possible to show that for any two elements in $\mathcal{D}$, the probability that they are equal is at most $1/2^{n-1}$. This is a routine case analysis and is based on the properties of ψ given by Definition 1. Since the number of elements in $\mathcal{D}$ is $\sigma + 2q$, the probability of $\mathsf{Coll}(\mathcal{D})$ is at most $(\sigma + 2q)(\sigma + 2q - 1)/(2 \times 2^{n-1})$. In a similar manner, the same bound on the probability of $\mathsf{Coll}(\mathcal{R})$ can be obtained so that $\Pr[\mathsf{Coll}] \leq (\sigma + 2q)^2/2^{n-1}$. This shows the privacy statement of PAE1 in Theorem 4.

C.2 Analysis of PAuth1

Scheme PAuth1 is similar to PAuth. We highlight the similarities and also the differences of the two schemes in Table 24. The portions where PAuth1 differs from that of PAuth is marked by boxes.

Table 24. Descriptions of PAuth and PAuth1. The parameters m and r are defined by the call to $\mathsf{Format}(P, n)$.

$\mathsf{PAuth}_{\pi,\mathsf{fStr}}(P)$:	$\mathsf{PAuth1}_{\pi,\mathsf{fStr}}(P)$:
1. $(P_1, \ldots, P_m) = \mathsf{Format}(P, n)$;	1. $(P_1, \ldots, P_m) = \mathsf{Format}(P, n)$;
2. $\gamma = \pi(\mathsf{fStr})$;	2. $\gamma = \pi(\mathsf{fStr})$;
3. if $(m = 1$ and $r < n)$ $\mathsf{sum} = P_1 \oplus \Gamma_{\gamma,-1}$;	3. if $(m = 1$ and $r < n)$ $\boxed{\mathsf{sum} = P_1 \oplus \Gamma_{\gamma,-2};}$
4. if $(m = 1$ and $r = n)$ $\mathsf{sum} = P_1 \oplus \Gamma_{\gamma,-2}$;	4. if $(m = 1$ and $r = n)$ $\boxed{\mathsf{sum} = P_1 \oplus \Gamma_{\gamma,-3};}$
5. if $(m > 1)$	5. if $(m > 1)$
6. $(C_1, \ldots, C_{m-1}) = \mathsf{ecb}_\pi(P_1 \oplus \Gamma_{\gamma,1}, \ldots, P_{m-1} \oplus \Gamma_{\gamma,m-1})$;	6. $(C_1, \ldots, C_{m-1}) = \mathsf{ecb}_\pi(P_1 \oplus \Gamma_{\gamma,1}, \ldots, P_{m-1} \oplus \Gamma_{\gamma,m-1})$;
7. $\mathsf{sum} = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m$;	7. $\mathsf{sum} = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m$;
8. if $(r < n)$ then $\mathsf{sum} = \mathsf{sum} \oplus \Gamma_{\gamma,m}$;	8. if $(r < n)$ then $\boxed{\mathsf{sum} = \mathsf{sum} \oplus \Gamma_{\gamma,m+1};}$
9. end if;	9. end if;
10. $\mathsf{tag} = \pi(\mathsf{sum})$;	10. $\boxed{\mathsf{tag} = \pi(\mathsf{sum} \oplus \Gamma_{\gamma,1} \oplus \cdots \oplus \Gamma_{\gamma,m-1});}$
return tag.	return tag.

It is easy to argue that the changes do not affect security. The changes are in the masking of the last block. Let $\xi_m = \Gamma_{\gamma,1} \oplus \cdots \oplus \Gamma_{\gamma,m-1}$. In PAuth, padded last blocks are masked by $\Gamma_{\gamma,m+1} \oplus \xi_m$

instead of by $\mathit{\Gamma}_{\gamma,m}$ as in PAuth; full last blocks are masked only by ξ_m while in PAuth they are not masked at all. The following observations show that the collision analysis is not affected by these changes.

1. The masking of single block messages changes from $\mathit{\Gamma}_{\gamma,-1}$ and $\mathit{\Gamma}_{\gamma,-2}$ to $\mathit{\Gamma}_{\gamma,-2}$ and $\mathit{\Gamma}_{\gamma,-3}$ respectively. It is easy to argue that these changes do not affect the collision analysis with a single block message when when the number of blocks in the other message is at most 2^{n-1}.
2. Suppose the number of blocks in the messages are m and m' and assume that both are greater than 1.
   (a) Consider the collision analysis of the last blocks. The structures of the last blocks are as follows.

$$D_m = \begin{cases} C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \xi_m & \text{if } r = n; \\ C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \mathit{\Gamma}_{m+1} \oplus \xi_m & \text{if } r < n; \end{cases}$$

$$D'_{m'} = \begin{cases} C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'} \oplus \xi_{m'} & \text{if } r = n; \\ C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'} \oplus \mathit{\Gamma}_{m'+1} \oplus \xi_{m'} & \text{if } r' < n; \end{cases}$$

   If $m = m'$, then the mask ξ_m is used to mask the last block of both messages and has no effect on the collision analysis of the last block irrespective of whether they are padded or full. Suppose $m > m'$. There are, as before, four cases for the values of r and r'. The point here is that the probability of D_m being equal to $D'_{m'}$ can be shown to be small without involving the $\mathit{\Gamma}$s. We consider $r = r' = n$, the consideration for the other three cases being similar. In this case,

$$\begin{aligned} D_m \oplus D'_{m'} &= C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus C'_1 \oplus \cdots \oplus C'_{m'-1} \oplus P'_{m'} \oplus \Upsilon \\ &= Y \oplus \Upsilon \end{aligned}$$

   where Υ is the XOR of all the terms which depend on γ and Y is the other part. The analysis of the distribution of Y is exactly the same as the case $r = r' = n$ in the proof of Lemma 1. Further, Y does not depend on γ and hence Y and Υ are independent. This shows that the collision analysis of the last blocks remain unaffected by the additional masking done in PAuth.
   (b) Now consider the collision analysis of a last block (with $m > 1$) and an internal block. In this case also, it can be argued that the additional masking does not make any difference.

Note that it is possible that for some m, $\mathit{\Gamma}_{\gamma,m+1} \oplus \xi_m = \mathit{\Gamma}_{\gamma,m+1} \oplus \mathit{\Gamma}_{\gamma,1} \oplus \cdots \oplus \mathit{\Gamma}_{\gamma,m-1}$ is zero even for a uniform random γ. This happens if the minimal polynomial $\tau(x)$ of ψ over $\mathbb{F}_2$ divides $x^{m+1} \oplus x^{m-1} \oplus \cdots \oplus x_1$. But, this fact does not affect the collision analysis which remains unchanged from that of PAuth as argued above. In a nutshell, this happens because $\mathit{\Gamma}_{m+1}$ is used to rule out collisions only when the number of blocks in the two messages are equal and the last block of one is full while the last block of the other is partial. Since, the number of blocks in the two messages are equal, ξ_ms for both the messages are also equal and they cancel out leaving only $\mathit{\Gamma}_{\gamma,m+1}$. We are then back to the analysis of PAuth.

By the above argument, the PRF-bound for PAuth given by Theorem 2 also holds for PAuth1. This shows the statement about PAuth1 in Theorem 3. The case of PAuth2 is similar to above and the bound for PAuth2 in Theorem 3 is obtained in the same way from the bound of PAuth.

C.3 Authenticity of PAE1

The authentication function associated with $\mathsf{PAE1}$ is denoted by $\widetilde{\mathsf{PAE1}}$ which maps (N, Y) to tag if there is an X such that $\mathsf{PAE1}$ maps (N, X) to (Y, tag). The function $\mathsf{Backward1}$ maps (N, Y) to (X, tag) and the output of $\widetilde{\mathsf{PAE1}}$ is obtained by discarding the X.

The function $\mathsf{PAuth1}$ is also defined using $\mathsf{Backward1}$ as:

$$\mathsf{PAuth1}_{\pi,\mathsf{fStr}}(Y) : (C, \mathsf{tag}) = \mathsf{Backward1}_{\pi}(\mathsf{fStr}, P);\ \text{return } \mathsf{tag}.$$

This suggests that $\mathsf{PAuth1}$ and $\widetilde{\mathsf{PAE1}}$ are very similar. They are not identical though. The difference arises in the manner the first n-bit block is treated. In $\mathsf{PAuth1}$, this is fixed to the string fStr, whereas, in $\widetilde{\mathsf{PAE1}}$ this is part of the input. A consequence of this difference is that the domain of $\mathsf{PAuth1}$ consists of strings of lengths greater than or equal to 0, whereas, the domain of $\widetilde{\mathsf{PAE1}}$ consists of strings of lengths greater than or equal to n.

Nevertheless, it is easy to argue that $\widetilde{\mathsf{PAE1}}$ is also a PRF. The basic idea for the collision analysis is the following. The masks are generated from γ which is obtained by applying π^{-1} to the first n-bit block. If the first blocks of two different queries are equal, then $\gamma = \gamma'$ and the collision analysis is exactly the same as that for $\mathsf{PAuth1}$. On the other hand, if the first blocks are unequal, then (γ, γ') is uniformly distributed over $\chi_2(\{0,1\}^n)$. As a result, the collision analysis between the internal variables corresponding to the two queries becomes easier: two such variables are equal with probability $1/2^n$. Due to this, the PRF-bound for $\mathsf{PAuth1}$ given by Theorem 3 also holds for $\widetilde{\mathsf{PAE1}}$.

Theorem 9. *Let q and $\sigma \geq q$ be positive integers. Then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\widetilde{\mathsf{PAE1}}}(q, \sigma) \leq \frac{(7q+2)\sigma}{2^n}. \tag{19}$$

For $1 \leq t \leq n$, let t-$\mathsf{PAE1}$ denote the AE function obtained from $\mathsf{PAE1}$ by truncating the tag to (the first) t bits. So, n-$\mathsf{PAE1}$ is in fact $\mathsf{PAE1}$. The privacy bounds for t-$\mathsf{PAE1}$ and t-$\mathsf{PAE1}^{\mathrm{main}}$ are the same as that of $\mathsf{PAE1}$. Using Proposition 1, we have

$$\mathbf{Adv}^{\text{ae-auth}}_{t\text{-}\mathsf{PAE1}}(q, \sigma) \leq \frac{1}{2^t} + \mathbf{Adv}^{\mathrm{prf}}_{t\text{-}\mathsf{PAE1}^{\mathrm{main}}}(q, \sigma) + \mathbf{Adv}^{\mathrm{prf}}_{\widetilde{\mathsf{PAE1}}}(q, \sigma).$$

The statement about authenticity of $\mathsf{PAE1}$ in Theorem 4 follows this and Theorem 9.

C.4 Analysis of PAE2

The privacy of $\mathsf{PAE2}$ follows in a manner similar to that of $\mathsf{PAE1}$ and the same bound holds. Following our approach of authentication analysis, we need to study the PRF-property of $\widetilde{\mathsf{PAE2}}$. Note that $\widetilde{\mathsf{PAE2}}$ uses both π and π^{-1} which is unlike $\widetilde{\mathsf{PAE1}}$ which uses only π^{-1}.

The differences between $\widetilde{\mathsf{PAE1}}$ and $\widetilde{\mathsf{PAE2}}$ are in the use π instead of π^{-1} at certain steps. For producing the masks γ it does not matter whether π or π^{-1} is applied. For the two functions, tag is produced by applying π^{-1} or π. This also does not cause any additional difficulty. In each case, the argument boils down to showing that the different values of $\mathsf{sum} \oplus \Gamma_{\gamma,1} \oplus \cdots \oplus \Gamma_{\gamma,m-1}$ are distinct and are also different from the different values of $C_i \oplus \Gamma_{\gamma,i}$. This analysis remains the same for both algorithms and so we omit the details. The bounds for $\mathsf{PAE2}$ are the same as that of $\mathsf{PAE1}$ as stated in Theorem 4.

D Security Arguments for the AEAD Schemes

Privacy of PAEAD1 as stated in Theorem 5 is easy to obtain and the analysis is similar to that of PAE1. Similarly, the privacy of PAEAD2 stated in Theorem 5 follows easily from the privacy of PAE2.

Before getting into the analysis of authenticity of the AEAD schemes, we make a brief digression to take a closer look at the security requirements of the associated authentication function of an AE scheme.

D.1 PRF Against Almost Nonce-Respecting Adversaries

Let f be an AE function and consider $\widetilde{f}$. An adversary attacking the PRF property of $\widetilde{f}$ has only one restriction on the queries, namely, two queries $(N^{(s)}, Y^{(s)})$ and $(N^{(t)}, Y^{(t)})$ cannot be the same. Now suppose, that the following additional restriction is made: for $1 \leq s < t \leq q-1$, $N^{(s)} \neq N^{(t)}$. Note that there is no restriction on $N^{(q)}$ which may or may not be equal to one of $N^{(s)}$ for $1 \leq s \leq q-1$. Adversaries of the above type will be called almost nonce-respecting (ANR). (If the restriction of distinctness is also imposed on $N^{(q)}$, then the adversary is nonce-respecting.) The ANR-PRF-advantage of f with respect to an almost nonce-respecting adversary $\mathcal{A}$ is defined to be

$$\mathbf{Adv}_{f}^{\text{anr-prf}}(\mathcal{A}) = \Pr[\mathcal{A}^{f} \Rightarrow 1] - \Pr[\mathcal{A}^{f^*} \Rightarrow 1]. \tag{20}$$

The resource bounded advantage is defined as usual to be $\mathbf{Adv}_{f}^{\text{anr-prf}}(q, \sigma)$. In the definition of authentication security of an AE function, an adversary is actually restricted to be ANR. In view of this, the following weaker version of Proposition 1 can be obtained.

Proposition 3. *Given an AE-function f, define another AE function h as follows: $h(N, X) = (Y, g(\mathsf{tag}))$, where $f(N, X) = (Y, \mathsf{tag})$ and $g : \{0,1\}^n \to \{0,1\}^t$ is a regular function. Then*

$$\mathbf{Adv}_{h}^{\text{ae-auth}}(q, \sigma) \leq \frac{1}{2^t} + \mathbf{Adv}_{f^{\text{main}}}^{\text{priv}}(q, \sigma) + \mathbf{Adv}_{\widetilde{f}}^{\text{anr-prf}}(q, \sigma).$$

Suppose $f_1 : \mathcal{N} \times \mathcal{X} \to \{0,1\}^n$ and $f_2 : \mathcal{H} \to \{0,1\}^n$ are independent random functions. Consider the function $f_3 : \mathcal{N} \times \mathcal{H} \times \mathcal{X} \to \{0,1\}^n$ defined as $f_3(N, H, X) \stackrel{\Delta}{=} f_1(N, X) \oplus f_2(H)$. Since f_1 and f_2 are independent functions, it is easy to show that

$$\mathbf{Adv}_{f_3}^{\text{prf}}(q, \sigma) \leq \mathbf{Adv}_{f_1}^{\text{prf}}(q, \sigma) + \mathbf{Adv}_{f_2}^{\text{prf}}(q, \sigma). \tag{21}$$

Now consider the following more complicated scenario. Let

$$\mathcal{S} = (\mathcal{N} \times \mathcal{X}) \bigcup (\mathcal{N} \times \mathcal{H} \times \mathcal{X})$$

and consider a function $f_4 : \mathcal{S} \to \{0,1\}^n$ defined as follows:

$$f_4(N, X) = f_1(N, X);$$
$$f_4(N, H, X) = f_3(N, H, X) = f_1(N, X) \oplus f_2(H).$$

The f_4 so defined is not a PRF and is easily demonstrated by four queries:

1. $(N^{(1)}, X^{(1)})$ returning $Y^{(1)} = f_1(N^{(1)}, X^{(1)})$;
2. $(N^{(1)}, H^{(1)}, X^{(1)})$ returning $Y^{(2)} = f_1(N^{(1)}, X^{(1)}) \oplus f_2(H^{(1)})$;

3. $(N^{(2)}, X^{(2)})$ with $(N^{(2)}, X^{(2)}) \neq (N^{(1)}, X^{(1)})$, returning $Y^{(3)} = f_1(N^{(2)}, X^{(2)})$;
4. $(N^{(2)}, H^{(1)}, X^{(2)})$ returning $Y^{(4)} = f_1(N^{(2)}, X^{(2)}) \oplus f_2(H^{(1)})$.

Then $f_2(H^{(1)}) = Y^{(1)} \oplus Y^{(2)} = Y^{(3)} \oplus Y^{(4)}$ showing that f_4 is not a PRF. The problem arises due to the fact that it is allowed to query f_4 on $(N^{(1)}, X^{(1)})$, $(N^{(1)}, H^{(1)}, X^{(1)})$ and $(N^{(2)}, X^{(2)})$, $(N^{(2)}, H^{(1)}, X^{(2)})$. Such an adversary is certainly not nonce-respecting and since two nonces have been repeated, it is also not almost nonce-respecting. The following, however, can be proved.

Proposition 4. *Let $\sigma \geq q \geq 1$. Then*

$$\mathbf{Adv}^{\text{anr-prf}}_{f_4}(q, \sigma) \leq \mathbf{Adv}^{\text{prf}}_{f_1}(q, \sigma) + \mathbf{Adv}^{\text{prf}}_{f_2}(q, \sigma). \tag{22}$$

Proof : We provide the main idea of the proof. If the adversary never repeats a nonce, then the analysis is the same as that for two independent PRFs with different input spaces. So, suppose that the adversary repeats a nonce. Under the almost nonce-respecting restriction, only one nonce can be repeated. Let this nonce be N and the corresponding queries be (N, M, H) and (N, M_1, H_1) with the condition that $(N, M, H) \neq (N, M_1, H_1)$. If $M \neq M_1$, then the pair $(N, M) \neq (N, M_1)$ and since the nonce is different from all the other nonces, the nonce-message pairs are all different. From the PRF-property of f_1, it follows that the outputs of f_4 will appear to be independent and uniformly distributed.

So, suppose that $M = M_1$ and we consider the two queries (N, M, H) and (N, M, H_1). The nonces of all other queries will be distinct from N and so the outputs of f_1 on these queries will be independent of the output of f_1 on (N, M). As a result, the outputs of f_4 on all queries other than (N, M, H) and (N, M, H_1) will be independent of the output of f_4 on these two queries. So, it is sufficient to argue that the outputs of f_4 on these two queries will be independent. Since queries cannot be repeated, we have $(N, M, H) \neq (N, M, H_1)$ which implies $H \neq H_1$. As a result, the outputs of f_2 on H and H_1 are independent which shows that the outputs of f_4 on (N, M, H) and (N, M, H_1) are also independent.

The above argument can be formalised in a standard manner. $\square$

In other words, if we restrict to almost nonce-respecting adversaries, then the ANR-PRF-bound for f_4 is upper bounded by the sum of the PRF-bounds for f_1 and f_2. From Proposition 3, this is sufficient to reason about the authentication security of an AE function.

D.2 Analysis of the AEAD Schemes

For authentication, we need to consider the function $\widetilde{\mathsf{PAEAD1}}$. Let $\mathsf{PAEAD1}_{\pi,\mathsf{fStr}}(N, H, P) = (C, \mathsf{tag})$ and $\upsilon = \pi^{-1}(\mathsf{fStr})$. Then from the definition of $\mathsf{PAEAD1}$, the following holds.

- If H is null, then

$$\widetilde{\mathsf{PAEAD1}}_{\pi,\mathsf{fStr}}(N, H, P) = \widetilde{\mathsf{PAE1}}_{\pi}(N, P). \tag{23}$$

- If H is not null, then

$$\widetilde{\mathsf{PAEAD1}}_{\pi,\mathsf{fStr}}(N, H, P) = \widetilde{\mathsf{PAE1}}_{\pi}(N, C) \oplus \mathsf{PAuth1}_{\pi^{-1},\upsilon}(H). \tag{24}$$

Similar equations can be written for $\mathsf{PAEAD2}$.

The functions $\mathsf{PAuth1}$ and $\widetilde{\mathsf{PAE1}}$ are both PRFs. However, they are not independent functions since the same π^{-1} is used for both of them. We will see how to tackle this difficulty a bit later and for the moment suppose that these are independent. Then using (22), we get a upper bound on the

ANR-PRF-advantage of $\widetilde{\mathsf{PAEAD1}}$. Using Proposition 3 this is sufficient to show the authentication security bound of t-PAEAD.

Now we turn to the issue of how to tackle the non-independence of $\mathsf{PAuth1}$ and $\widetilde{\mathsf{PAE1}}$. Let $\mathcal{E}$ be the event that the set of inputs to π^{-1} in $\mathsf{PAuth1}$ is disjoint from the set of inputs to π^{-1} in $\widetilde{\mathsf{PAE1}}$. (Consequently, the set of inputs to π in $\mathsf{PAuth1}$ will also be disjoint from the set of inputs to π in $\widetilde{\mathsf{PAE1}}$.) Then the PRF bounds for the individual functions would hold and using standard arguments we obtain

$$\mathbf{Adv}^{\text{anr-prf}}_{\widetilde{\mathsf{PAEAD1}}}(q,\sigma) \le \mathbf{Adv}^{\text{prf}}_{\widetilde{\mathsf{PAE1}}}(q,\sigma) + \mathbf{Adv}^{\text{prf}}_{\mathsf{PAuth1}}(q,\sigma) + \Pr[\overline{\mathcal{E}}]. \tag{25}$$

Proposition 3 gives

$$\begin{aligned}\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\mathsf{PAEAD1}}(q,\sigma) &\le \frac{1}{2^t} + \mathbf{Adv}^{\text{priv}}_{\mathsf{PAEAD1}^{\text{main}}}(q,\sigma) + \mathbf{Adv}^{\text{anr-prf}}_{\widetilde{\mathsf{PAEAD1}}}(q,\sigma)\\ &\le \frac{1}{2^t} + \mathbf{Adv}^{\text{priv}}_{\mathsf{PAEAD1}^{\text{main}}}(q,\sigma)\\ &\quad + \mathbf{Adv}^{\text{prf}}_{\widetilde{\mathsf{PAE1}}}(q,\sigma) + \mathbf{Adv}^{\text{prf}}_{\mathsf{PAuth1}}(q,\sigma) + \Pr[\overline{\mathcal{E}}]. \end{aligned} \tag{26}$$

The task, thus, reduces to bounding $\Pr[\overline{\mathcal{E}}]$. The event $\mathcal{E}$ represents the separation of the inputs for π^{-1} in the message and header part. In the $\widetilde{\mathsf{PAE1}}$ part, the masks are obtained from γ which is obtained as $\pi^{-1}(N)$. On the other hand, in the $\mathsf{PAuth1}$ part the masks are obtained from $\pi^{-1}(\upsilon) = \pi^{-1}(\pi^{-1}(\mathsf{fStr}))$. Since, the probability that N is equal to $\pi^{-1}(\mathsf{fStr})$ is $1/2^n$, we obtain an effective separation of the masks. We consider this in more details.

First consider the inputs and outputs to π^{-1} determined by $\widetilde{\mathsf{PAEAD}}_{\pi^{-1}}(N^{(s)}, C^{(s)})$. For the s-th query, let $A_i^{(s)}$ and $B_i^{(s)}$ $(1 \le i \le m^{(s)})$ be the different inputs and outputs to π, so that $\pi^{-1}(B_i^{(s)}) = A_i^{(s)}$. The expressions for $A_i^{(s)}$ and $B_i^{(s)}$ are given in Section C.1. Note that each $B_i^{(s)}$ (other than the nonce) is masked with the XOR of one or more of the $\Gamma_{\gamma^{(s)},i}$s.

Next consider the inputs and outputs to π^{-1} determined by $\mathsf{PAuth1}_{\pi^{-1}}(\upsilon, H^{(s)})$. Such calls are made only if $H^{(s)}$ is non-null. Let the number of n-bit blocks in $H^{(s)}$ be $k^{(s)}$ and let the length of the last block before padding be $p^{(s)}$. Denote the blocks as $H_1^{(s)}, \ldots, H_{k^{(s)}}^{(s)}$. These blocks are the output of $\mathsf{Format}(H^{(s)}, n)$ which also defines the values of $k^{(s)}$ and $p^{(s)}$. Let $T_i^{(s)} = \pi^{-1}(H_i^{(s)})$ for $1 \le i \le k^{(s)} - 1$; and let $\mathsf{PAuth1}_{\pi^{-1}}(\upsilon, H^{(s)})$ be denoted by $\mathsf{htag}^{(s)}$.

Let $\omega = \pi^{-1}(\upsilon)$ and $\Omega_i = \psi^i(\omega)$. Since fStr does not depend on the queries, neither do the Ω_i's or υ. For $1 \le s \le q$, if $H^{(s)}$ is non-null, then let $E_i^{(s)}$ for $1 \le i \le k^{(s)}$ be the different inputs to π and $F_i^{(s)}$ be the different outputs of π (and so are inputs to π^{-1}), i.e., $\pi(E_i^{(s)}) = F_i^{(s)}$. The different $F_i^{(s)}$s are as follows.

$$\begin{array}{ll} H_1^{(s)} \oplus \Gamma_{\gamma^{(s)},-1} & \text{if } (k^{(s)} = 1 \text{ and } p^{(s)} < n);\\ H_1^{(s)} \oplus \Gamma_{\gamma^{(s)},-2} & \text{if } (k^{(s)} = 1 \text{ and } p^{(s)} = n);\\ \left.\begin{array}{l} H_1^{(s)} \oplus \Gamma_{\gamma^{(s)},1}, \ldots, H_{k^s-1}^{(s)} \oplus \Gamma_{\gamma^{(s)},k^{(s)}-1},\\ E_1^{(s)} \oplus \cdots \oplus E_{k^s-1}^{(s)} \oplus H_{m^{(s)}}^{(s)} \end{array}\right\} & \text{if } (k^{(s)} > 1 \text{ and } p^{(s)} = n);\\ \left.\begin{array}{l} H_1^{(s)} \oplus \Gamma_{\gamma^{(s)},1}, \ldots, H_{k^s-1}^{(s)} \oplus \Gamma_{\gamma^{(s)},k^{(s)}-1},\\ E_1^{(s)} \oplus \cdots \oplus E_{k^s-1}^{(s)} \oplus H_{k^{(s)}}^{(s)} \oplus \Gamma_{\gamma^{(s)},k^{(s)}} \end{array}\right\} & \text{if } (k^{(s)} > 1 \text{ and } p^{(s)} < n). \end{array}$$

We are interested in the event $\overline{\mathcal{E}}$ which holds if one of the following occur.

1. Some $B_j^{(t)}$ is equal to some $F_{k^{(s)}}^{(s)}$, where $k^{(s)} > 1$ and $p^{(n)} = n$.

2. Some $B^{(t)}_j$ is equal to some $F^{(s)}_i$,
3. Some $B^{(t)}_j$ is equal to either υ or ω.

Each $B^{(t)}_j$ (other than the nonce) has a component which is either $\delta^{(t)}$, or $\Gamma_{\gamma^{(t)},j}$ or a XOR of some of the Γ's. For a fixed j, $\Gamma_{\gamma^{(t)},j}$ uniquely determines $\gamma^{(t)}$. Similarly, for a fixed i, Ω_i uniquely determines ω. Here both i and j are greater than 1.

Consider the event $\Gamma_{\gamma^{(t)},j} = \upsilon$, i.e., $\psi^j(\gamma) = \upsilon$. If $N^{(t)} = \mathsf{fStr}$, then $\gamma^{(t)} = \upsilon$; since $j \geq 1$, from Definition 1, we have $\Pr[\psi^j(\gamma^{(t)}) = \upsilon] = 1/2^n$. If $N^{(t)} \neq \mathsf{fStr}$, then since $\gamma^{(t)} = \pi^{-1}(N^{(t)})$ and $\upsilon = \pi^{-1}(\mathsf{fStr})$, the pair $(\gamma^{(t)}, \upsilon)$ is uniformly distributed over $\chi_2(\mathbb{F})$. Again from Definition 1, it follows that $\Pr[\psi^j(\gamma^{(t)}) = \upsilon] = 1/(2^n - 1)$. So, in both cases, $\Pr[\Gamma_{\gamma^{(t)},j} = \upsilon] \leq 1/(2^n - 1)$.

Now consider the event $\Gamma_{\gamma^{(t)},j} = \omega$, i.e., $\psi^j(\gamma^{(t)}) = \omega$. The analysis is similar, the difference being that in this case $\omega = \pi^{-1}(\upsilon)$ and so the event $N^{(t)} = \upsilon$ holds with probability $1/2^n$. Using this it is possible to show that $\Pr[\Gamma_{\gamma^{(t)},j} = \upsilon] \leq 1/2^{n-1}$. A similar analysis holds for the event $\Gamma_{\gamma^{(t)},j} = \Omega_i$.

These show that the events in Points 2 and 3 above hold with probability at most $1/2^{n-1}$. For the event in Point 1, $F^{(s)}_{k^{(s)}} = E^{(s)}_1 \oplus \cdots \oplus E^{(s)}_{k^s-1} \oplus H^{(s)}_{m^{(s)}}$. Since $k^{(s)} > 1$, there is at least one $E^{(s)}_i$ in the expression for $F^{(s)}_{k^{(s)}}$. The probability that this is equal to $\Gamma_{\gamma^{(t)},j}$ can again be shown to be bounded above by $1/2^{n-1}$. So, the event in Point 1 also holds with probability at most $1/2^{n-1}$.

As a result of this analysis, we obtain $\Pr[\overline{\mathcal{E}}] \leq \sigma_H \sigma_P / 2^{n-1} \leq \sigma^2/2^{n-1}$. This leads to the authenticity bounds for $\mathsf{PAEAD1}$ in Theorem 5. The authenticity bounds for $\mathsf{PAEAD2}$ in Theorem 5 are obtained in a similar manner.

The security bounds for the privacy and authenticity of $\overrightarrow{\mathsf{PAEAD1}}$ and $\overrightarrow{\mathsf{PAEAD2}}$ are the same as that for $\mathsf{PAEAD1}$ and $\mathsf{PAEAD2}$ with an additional degradation of $q(q-1)/2$ which arises due to the use of the vector input versions of $\mathsf{PAuth1}$ and $\mathsf{PAuth2}$.

E Analysis of Deterministic Authenticated Encryption with Associated Data

The analysis of privacy of DAE is similar to that of the PAEAD schemes. There are some differences in the descriptions of the internal variables arising due to the difference in the core encryption modes of the schemes. This, however, does not cause any problem and the privacy bounds for DAE and DAEAD given in Theorem 7 are the same as that of $\mathsf{PAEAD1}$.

The analysis of authenticity proceeds along the lines similar to the analysis of authenticity of the PAEAD schemes. We describe this below for DAEAD the case of DAE being similar and simpler.

For notational convenience, let us denote $\mathsf{DAEAD}_{\pi,\mathsf{fStr}}$ as $\mathbf{E}$, where the randomness of $\mathbf{E}$ arises from that of π. The adversary makes a total of $(q-1)$ encryption queries $(\overrightarrow{H}^{(1)}, P^{(1)}), \ldots, (\overrightarrow{H}^{(q-1)}, P^{(q-1)})$ obtaining in response $(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q-1)}, \mathsf{tag}^{(q-1)})$ respectively and finally outputs a forgery $(\overrightarrow{H}, C, \mathsf{tag})$. By definition, the triplet $(\overrightarrow{H}, C, \mathsf{tag})$ is not equal to $(\overrightarrow{H}^{(s)}, C^{(s)}, \mathsf{tag}^{(s)})$ for $s = 1, \ldots, q-1$.

The decryption algorithm implicitly defines a P from the forgery triplet $(\overrightarrow{H}, C, \mathsf{tag})$. We claim that the pair $(\overrightarrow{H}, P)$ is not equal to $(\overrightarrow{H}^{(s)}, P^{(s)})$ for $s = 1, \ldots, q$. This can be seen as follows. Suppose $(C, \mathsf{tag}) = (C^{(s)}, \mathsf{tag}^{(s)})$, then the condition $(\overrightarrow{H}, C, \mathsf{tag}) \neq (\overrightarrow{H}^{(s)}, C^{(s)}, \mathsf{tag}^{(s)})$ forces $\overrightarrow{H} \neq \overrightarrow{H}^{(s)}$ and so $(\overrightarrow{H}, P) \neq (\overrightarrow{H}^{(s)}, P^{(s)})$. So suppose $(C, \mathsf{tag}) \neq (C^{(s)}, \mathsf{tag}^{(s)})$: if $\overrightarrow{H} \neq \overrightarrow{H}^{(s)}$, then again $(\overrightarrow{H}, P) \neq (\overrightarrow{H}^{(s)}, P^{(s)})$; so, further suppose that $\overrightarrow{H} = \overrightarrow{H}^{(s)}$. If $P = P^{(s)}$, then it necessarily follows that $(C, \mathsf{tag}) = (C^{(s)}, \mathsf{tag}^{(s)})$ which contradicts the hypothesis. So, $P \neq P^{(s)}$ implying $(\overrightarrow{H}, P) \neq (\overrightarrow{H}^{(s)}, P^{(s)})$.

The decryption algorithm of DAEAD produces tag_1 from $(\overrightarrow{H}, P)$ and this is compared to tag provided as part of the forgery attempt. In light of the above discussion, the tags $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q-1)}, \mathsf{tag}_1$ are produced as the output of $\overrightarrow{\mathsf{PAuth}}$ on the distinct inputs $(\overrightarrow{H}^{(1)}, P^{(1)}), \ldots, (\overrightarrow{H}^{(q-1)}, P^{(q-1)}), (\overrightarrow{H}, P)$. Assuming that $\overrightarrow{\mathsf{PAuth}}$ is a PRF, $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q-1)}, \mathsf{tag}_1$ are independently and uniformly distributed and so the probability that tag_1 is equal to tag provided in the forgery attempt is $1/2^n$. Formalising this argument in a standard manner provides the authenticity bound for DAEAD in Theorem 7.