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 $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, where for each $K \in \mathcal{K}$, the function $E_K(\cdot) \stackrel{\Delta}{=} E(K,\cdot)$ 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 the USA [18, 58].

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 these 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. 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 [13] 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. 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: The authentication scheme is obtained by a small modification of a similar previous scheme in [54]. 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.

AE(AD) schemes: The AE schemes described in this work are single pass (or Rate-1) schemes. The basic idea of these schemes is similar to the ECB like schemes in the early works in [37, 26]. Structurally, the handling of the last block and tag generation has similarities to OCB2 [51]. AEAD schemes are obtained by combining the AE schemes with the authentication schemes. A carefully placed block cipher call is used to ensure that the combination is secure. 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 [60].

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.

Galois field masking: The various modes of operations require an appropriate masking strategy. Use of Galois field masking based on word-oriented LFSRs have been suggested in the past [14]. Details, especially those pertaining to implementations were missing. In this work, we fully develop the previous suggestions. The multiply-by-x (or `xtimes` or "doubling") operation is described in the more general setting where a tower field representation is used. Several different tower field representations are considered. Constant time implementations without any table look-up are described for the different masking options. Code examples using Intel intrinsics using SSE instructions are provided. Further, detailed timing comparisons are provided both for the generation of a single mask as also for the different schemes using various masking strategies using SSE instructions on modern Intel processors.

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 SSE 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 [15].

The CAESAR [13] 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, are far too late for the CAESAR submission. Hopefully, due to its scope and completeness including implementation and performance details, 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 [61, 25] and the works [57, 9, 8] 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 [5, 10, 41, 30, 31] has resulted in the CMAC [21] algorithm which has been standardised by the NIST of the 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 [11, 51, 14, 54]. 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 [51, 54]. Later we mention the exact relation of the scheme described here with that appearing in [54].

The first formal treatment of authenticated encryption was proposed in [6, 38]. Single-pass AE modes were proposed in [37, 26] and was quickly followed by the first version of the famous OCB mode [52]. Around the same time as these works, the notion of AEAD was first formally studied in [50] 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 [55].

Two later variants of the OCB mode have been proposed in [51, 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 close to that of OCB were proposed in [14]. The work [54] also proposed AE modes of operations which, however, have flaws and attacks on these modes are known. Due to the patent issues covering OCB and the modes in [37, 26, 51], the NIST of the 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 [62] 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 [7] and EAX-prime [59]. The scheme EAX-prime was, however, broken in [48] and improved authenticity bounds for EAX were 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 [53]. 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 [53] 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 [4] and then modify it to obtain authentication. The constructions POET [43] and COPA [1] 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.

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 [56]. 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

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

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 $X = x_0 x_1 \cdots x_7$ 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 $x_0 x_1 x_2$ which is equal to 110. Let $n = 4$ and $\ell = 13$, then $\mathsf{bin}_4(13)$ is the 4-bit string $y_0 y_1 y_2 y_3 = 1101$, 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-

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.

$\mathsf{Format}(X, n)$
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 $\mathsf{pad}(X) = X \vert\vert 10^{n-r-1}$;
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 $(X_1, \ldots, X_m)$.

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 $\mathcal{X}$.

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

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

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

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

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.

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

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

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

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 [54].

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. Let $f \in \mathcal{F}_n[\mathcal{N},\mathcal{X}]$, i.e., $f : \mathcal{N} \times \mathcal{X} \to \mathcal{X} \times \{0,1\}^n$. We say that f is an AE-function if for every N, the functions $f_N(\cdot) \stackrel{\Delta}{=} f(N,\cdot)$ is an injection. Injectivity ensures that given $N \in \mathcal{N}$ and (Y,tag), there is a unique X such that $f(N,X) = (Y,\mathsf{tag})$.

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 $1 \le i \le q$, $\mathcal{A}$ 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.

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,tag_i) for $1 \le i \le q-1$. 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.

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 say that f is an AEAD-function if for every $N \in \mathcal{N}$ and $H \in \mathcal{H}$, the function $f_{N,H}(\cdot) \stackrel{\Delta}{=} f(N, H, \cdot)$ is an injection.

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 [50]. 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 say that f is a DAEAD-function if for every $H \in \mathcal{H}$, the function $f_H(\cdot) \stackrel{\Delta}{=} f(H, \cdot)$ is an injection.

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. This polynomial is used to define $\mathbb{F}_{2^{n_1}}$. Let $\mu(x)$ be a monic *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.

Note. 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 $\rho(\alpha)$.

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	$n = n_1 \times n_2$	$\rho(\alpha)$	$\mu(x)$
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 Multiply-by-x Map

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 $\mathbb{F}_{2^n}$. We define a map

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

in the following manner. Let $\gamma \in \mathbb{F}_n$ be written as $\gamma(x) = a_0 \oplus a_1 x \oplus \cdots \oplus a_{n_2-1}x^{n_2-1}$ where a_i are elements of $\mathbb{F}_{2^{n_1}}$. The map ψ acts on γ in the following manner.

$$\begin{aligned}\beta &= \psi(\gamma)\\ &= x\gamma(x) = x(a_0 \oplus a_1 x \oplus \cdots \oplus a_{n_2-1}x^{n_2-1}) \bmod \mu(x)\\ &= a_0 x \oplus a_1 x^2 \oplus \cdots \oplus a_{n_2-2}x^{n_2-1} \oplus a_{n_2-1}x^{n_2} \bmod \mu(x)\\ &= a_{n_2-1}t_0 \oplus (a_{n_2-1}t_1 \oplus a_0)x \oplus \cdots \oplus (a_{n_2-1}t_{n_2-1} \oplus a_{n_2-2})x^{n_2-1}.\end{aligned} \tag{7}$$

Writing $\beta(x) = b_0 \oplus b_1 x \oplus \cdots \oplus b_{n_2-1}x^{n_2-1}$, the coefficients b_j's are given by (7). The map ψ is invertible and the inverse map ψ^{-1} is given as follows:

$$\begin{aligned}\gamma &= \psi^{-1}(\beta)\\ &= x^{-1}\beta(x) = x^{-1}(b_0 \oplus b_1 x \oplus \cdots \oplus b_{n_2-1}x^{n_2-1}) \bmod \mu(x)\\ &= t_0^{-1}b_0 x^{n_2-1} \oplus (t_0^{-1}b_0 t_{n_2-1} \oplus b_{n_2-1})x^{n_2-2} \oplus \cdots \oplus (t_0^{-1}b_0 t_2 \oplus b_2)x \oplus (t_0^{-1}b_0 t_1 \oplus b_1).\end{aligned} \tag{8}$$

When $n_1 = 1$ and $n_2 = 128$, the corresponding ψ has been called the powering-up map in [51]. The description given in (7) can be considered to be the general powering up map. 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. Such a choice of $\mu(x)$ requires a multiplication by α for the forward map ψ and a multiplication by α^{-1} for the backward map ψ^{-1}.

Since $\mu(x)$ is a primitive polynomial, it can be proved [42] that 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.

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

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. 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.

$$\Gamma_{\gamma,i} \stackrel{\Delta}{=} \psi^i(\gamma) \tag{11}$$

where $\psi(\gamma)$ is given by (7). 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 [54] 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 [54].

Proposition 1. *The function ψ defined in (6) and (7) 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.

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 specification. 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 [53] 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 [53] converts f to a vector-input PRF f^* as follows:

$$f^*(X_1, \ldots, X_k) = \begin{cases} f(1^n) & \text{if } k = 0; \\ f(\alpha^{k-1}Y_0 \oplus \alpha^{k-2}Y_1 \oplus \cdots \oplus Y_{k-1} \oplus_{\mathsf{end}} X_k) & \text{if } k > 0 \text{ and } |X_k| \geq n; \\ f(\alpha^k Y_0 \oplus \alpha^{k-1}Y_1 \oplus \cdots \oplus \alpha Y_{k-1} \oplus X_k || 10^{n-r-1}) & \text{if } k > 0 \text{ and } |X_k| = r < n, \end{cases} \quad (12)$$

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 α 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 [53] that if f is a uniform random function, then $\mathsf{Adv}^{\mathrm{prf}}_{f^*}(q, \sigma) \leq \sigma q / 2^n$.

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.

$$\overrightarrow{f}(X_1, \ldots, X_k) = f(\omega_0 || (f(\omega_1 || X_1) \oplus\!\!\!\rightarrow \cdots \oplus\!\!\!\rightarrow f(\omega_k || X_k))). \quad (13)$$

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 Forward1 and Backward1; Forward1 takes a nonce-message pair (N, P) to (C, tag) and Backward1 takes (N, C) to (P, tag). The

algorithm Forward1 is used to define the encryption algorithm of an AE scheme PAE1. The corresponding decryption algorithm is derived from Backward1: given (N, C, tag), the decryption algorithm invokes Backward1 on (N, C) to obtain (P, tag_1) and returns P if $\mathsf{tag} = \mathsf{tag}_1$, else returns $\perp$.

The encryption algorithm of PAE1 uses only π whereas the decryption algorithm uses both π and π^{-1}. We define PAE2 to be the 'dual' of this strategy, i.e., the encryption algorithm of PAE2 uses both π and π^{-1} while the decryption algorithm uses only π. PAE1 (resp. PAE2) and PAuth are combined to obtain an AEAD scheme PAEAD1 (resp. PAEAD2). The scheme PAEAD1 (resp. PAEAD2) is naturally extended to $\overrightarrow{\mathsf{PAEAD1}}$ (resp. $\overrightarrow{\mathsf{PAEAD2}}$) where the header can be a vector of strings. This variant arises by using $\overrightarrow{\mathsf{PAuth}}$ to process the complex header.

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 [53] 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 [60]. 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 P, where $\mathsf{len}(P) \geq 0$. 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 (4) it is possible to obtain the security of t-PAuth as an authentication function for any $t \leq n$.

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,δ}(P):
1.  (P_1, ..., P_m) = Format(P, n);
2.  κ = π(δ);
3.  if (m = 1 and r < n) sum = P_1 ⊕ Γ_{κ,−1};
4.  if (m = 1 and r = n) sum = P_1 ⊕ Γ_{κ,−2};
5.  if (m > 1)
6.      (C_1, ..., C_{m−1})
            = ecb_π(P_1 ⊕ Γ_{κ,1}, ..., P_{m−1} ⊕ Γ_{κ,m−1});
7.      sum = C_1 ⊕ ··· ⊕ C_{m−1} ⊕ P_m;
8.      if (r < n) then sum = sum ⊕ Γ_{κ,m};
9.  end if;
10. tag = π(sum);
return tag.
```

The construction PAuth is very similar to the construction iPMAC given in [54]. The only difference is in the way the case of $m = 1$ is handled. In [54], for $m = 1$, sum is $\delta \oplus P_1$ or $\delta \oplus P_1 \oplus \Gamma_{\kappa,1}$ according as $r = n$ or $r < n$, where $\delta = \pi(\kappa)$. The construction PAuth avoids using δ which requires an additional application of π. Instead, the masks $\Gamma_{\kappa,-1}$ and $\Gamma_{\kappa,-2}$ are used. This eliminates an extra block cipher call when $m = 1$.

For an m-block message, PAuth makes one block cipher call on the string δ and then makes a total of m calls to generate the output tag. The call on δ 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 $\mathsf{Forward1}$ and $\mathsf{Backward1}$ shown in Table 4. The input to $\mathsf{Forward1}$ is a pair (N, P), where N is an n-bit string and P is a non-empty binary string.

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

The authenticated encryption scheme $\mathsf{PAE1}$ is defined from these and is given in Table 5. Table 6 describes the scheme $\mathsf{PAEAD1}$. It makes use of PAuth. The header H can be a (possibly empty) binary string. 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 (possibly empty) binary string. We would like to highlight the following points.

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

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

The structure of $\mathsf{PAE1}$ is similar to that of OCB2 [51]. In particular, the method of handling the last block and generating the tag are essentially the same. The difference to OCB2 arises in the way the masking is done. OCB2 uses the powering up method to mask the inputs corresponding to the messages. The mask for the tag generation is done by multiplying the current mask with $(x + 1)$. If the discrete log of $(x + 1)$ is large, then this ensures that the masks for the messages are distinct from the masks for the tags. In [14], another method for masking was proposed which avoided the multiplication by $(x + 1)$ using a so-called interleaving strategy. $\mathsf{PAE1}$ uses a different strategy to separate the masks for the message blocks from the masks for the tag generation. XORing δ_1 ensures (with high probability) that the masks for the tag generation will be distinct from the masks for the message blocks.

7.1 A Variant

The encryption algorithm of the AE and AEAD algorithms uses only π whereas the decryption algorithm uses both π and π^{-1}. It is possible to define a 'dual' AE(AD) algorithm where the decryption

Table 4. The algorithms $\mathsf{Forward1}$ and $\mathsf{Backward1}$. The values of m and r are defined by the calls to Format.

$\mathsf{Forward1}_{\pi,\delta}(N, P)$:	$\mathsf{Backward1}_{\pi,\delta}(N, C)$:
1. $(P_1, \ldots, P_m) = \mathsf{Format}(P, n)$;	1. $(C_1, \ldots, C_m) = \mathsf{Format}(C, n)$;
2. $\gamma = \pi(N \oplus \delta)$;	2. $\gamma = \pi(N \oplus \delta)$;
3. $C_i = \pi(P_i \oplus \Gamma_{\gamma,i}) \oplus \Gamma_{\gamma,i}$; $i = 1, \ldots, m-1$;	3. $P_i = \pi^{-1}(C_i \oplus \Gamma_{\gamma,i}) \oplus \Gamma_{\gamma,i}$; $i = 1, \ldots, m-1$;
4. $\mathsf{pad} = \pi(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,m})$;	4. $\mathsf{pad} = \pi(\mathsf{bin}_n(r) \oplus \Gamma_{\gamma,m})$;
5. $C_m = \mathsf{First}_r(P_m \oplus \mathsf{pad})$; $T_m = C_m \Vert 0^{n-r}$;	5. $P_m = \mathsf{First}_r(C_m \oplus \mathsf{pad})$; $T_m = \mathsf{First}_r(C_m) \Vert 0^{n-r}$;
6. $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus T_m \oplus \Gamma_{\gamma,m+1} \oplus \delta \oplus \mathsf{pad}$;	6. $\mathsf{sum} = P_1 \oplus \cdots \oplus P_{m-1} \oplus T_m \oplus \Gamma_{\gamma,m+1} \oplus \delta \oplus \mathsf{pad}$;
7. $\mathsf{tag} = \pi(\mathsf{sum})$;	7. $\mathsf{tag} = \pi(\mathsf{sum})$;
return $(C_1 \Vert \cdots \Vert C_{m-1} \Vert C_m, \mathsf{tag})$.	return $(P_1 \Vert \cdots \Vert P_{m-1} \Vert P_m, \mathsf{tag})$.

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

$\mathsf{PAE1.Encrypt}_{\pi,\mathsf{fStr}}(N, P)$:	$\mathsf{PAE1.Decrypt}_{\pi,\mathsf{fStr}}(N, C, \mathsf{tag})$:
1. $\delta_0 = \pi(\mathsf{fStr})$; $\delta_1 = \Gamma_{\delta_0,1}$;	1. $\delta_0 = \pi(\mathsf{fStr})$; $\delta_1 = \Gamma_{\delta_0,1}$;
2. $(C, \mathsf{tag}) = \mathsf{Forward1}_{\pi,\delta_1}(N, P)$;	2. $(P, \mathsf{tag}') = \mathsf{Backward1}_{\pi,\delta_1}(N, C)$;
3. return (C, tag).	3. if $\mathsf{tag} = \mathsf{tag}'$ return P; else return $\perp$.

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. $\delta_0 = \pi(\mathsf{fStr})$; $\delta_1 = \Gamma_{\delta_0,1}$;	1. $\delta_0 = \pi(\mathsf{fStr})$; $\delta_1 = \Gamma_{\delta_0,1}$;
2. if $\lvert H\rvert = 0$, $\mathsf{tag}_2 = 0^n$;	2. if $\lvert H\rvert = 0$, $\mathsf{tag}_2 = 0^n$;
3. else $\mathsf{tag}_2 = \mathsf{PAuth}_{\pi,\delta_0}(H)$;	3. else $\mathsf{tag}_2 = \mathsf{PAuth}_{\pi,\delta_0}(H)$;
4. $(C, \mathsf{tag}_1) = \mathsf{Forward1}_{\pi,\delta_1}(N, P)$;	4. $(P, \mathsf{tag}_1) = \mathsf{Backward1}_{\pi,\delta_1}(N, C)$;
5. return $(C, \mathsf{tag}_1 \oplus \mathsf{tag}_2)$.	5. if $\mathsf{tag} = (\mathsf{tag}_1 \oplus \mathsf{tag}_2)$ return P; else return $\perp$.

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. $\delta_0 = \pi(\mathsf{fStr})$; $\delta_1 = \Gamma_{\delta_0,1}$;	1. $\delta_0 = \pi(\mathsf{fStr})$; $\delta_1 = \Gamma_{\delta_0,1}$;
2. if $k = 0$, $\mathsf{tag}_2 = 0^n$;	2. if $k = 0$, $\mathsf{tag}_2 = 0^n$;
3. else $\mathsf{tag}_2 = \overrightarrow{\mathsf{PAuth}}_{\pi,\delta_0}(H_1, \ldots, H_k)$;	3. else $\mathsf{tag}_2 = \overrightarrow{\mathsf{PAuth}}_{\pi,\delta_0}(H_1, \ldots, H_k)$;
4. $(C, \mathsf{tag}_1) = \mathsf{Forward1}_{\pi,\delta_1}(N, P)$;	4. $(P, \mathsf{tag}_1) = \mathsf{Backward1}_{\pi,\delta_1}(N, C)$;
5. return $(C, \mathsf{tag}_1 \oplus \mathsf{tag}_2)$.	5. if $\mathsf{tag} = (\mathsf{tag}_1 \oplus \mathsf{tag}_2)$ return P; else return $\perp$.

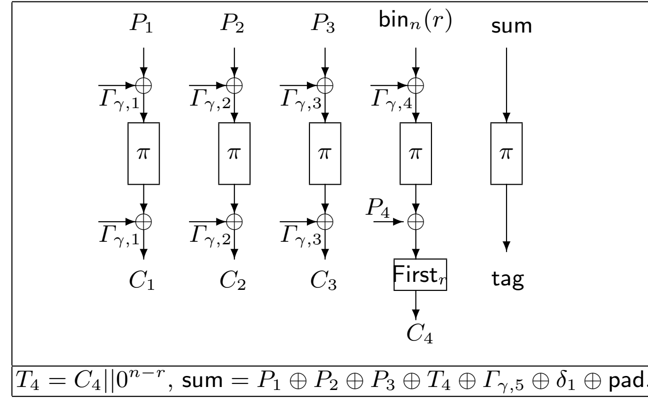

Fig. 1. Encryption using PAE1.

algorithm uses only π while the encryption algorithm uses both π and π^{-1}. The corresponding algorithms are denoted as Forward2, Backward2, PAE2, PAEAD2 and $\overrightarrow{\mathsf{PAEAD2}}$. These algorithms are shown in Tables 8, 9, 10 and 11.

It is to be noted that for a single block message, the output of PAE1 is the same as that of PAE2. If the number of blocks is more than one, then the last ciphertext block and the tag is same for both PAE1 and PAE2. This is due to the fact that the changes made to Forward1 to obtain Forward2 do not affect the last ciphertext block and the tag computation.

7.2 Security

The following provides the security statements for PAE and PAE2.

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

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

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

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

$$\begin{aligned}
\mathbf{Adv}^{\text{priv}}_{\mathsf{PAEAD1}}(q,\sigma) &\leq \frac{(\sigma+q+2)^2}{2^n};\\
\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\mathsf{PAEAD1}}(q,\sigma) &\leq \frac{2^{n-t}}{2^n-(\sigma+q+3)} + \frac{(\sigma+q+2)^2}{2^n};\\
\mathbf{Adv}^{\text{priv}}_{\mathsf{PAEAD2}}(q,\sigma) &\leq \frac{(\sigma+q+2)^2}{2^n};\\
\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\mathsf{PAEAD2}}(q,\sigma) &\leq \frac{2^{n-t}}{2^n-(\sigma+q+3)} + \frac{(\sigma+q+2)^2}{2^n}.
\end{aligned}$$

Table 8. The algorithms Forward2 and Backward2. The values of m and r are defined by the calls to Format. Differences to Forward1 and Backward1 are shown using boxes.

$\mathsf{Forward2}_{\pi,\delta}(N,P)$:	$\mathsf{Backward2}_{\pi,\delta}(N,C)$:
1. $(P_1,\dots,P_m)=\mathsf{Format}(P,n)$;	1. $(C_1,\dots,C_m)=\mathsf{Format}(C,n)$;
2. $\gamma=\pi(N\oplus\delta)$;	2. $\gamma=\pi(N\oplus\delta)$;
3. $\boxed{C_i=\pi^{-1}(P_i\oplus\Gamma_{\gamma,i})\oplus\Gamma_{\gamma,i};}$ $i=1,\dots,m-1$;	3. $\boxed{P_i=\pi(C_i\oplus\Gamma_{\gamma,i})\oplus\Gamma_{\gamma,i};}$ $i=1,\dots,m-1$;
4. $\mathsf{pad}=\pi(\mathsf{bin}_n(r)\oplus\Gamma_{\gamma,m})$;	4. $\mathsf{pad}=\pi(\mathsf{bin}_n(r)\oplus\Gamma_{\gamma,m})$;
5. $C_m=\mathsf{First}_r(P_m\oplus\mathsf{pad})$; $T_m=C_m\|\|0^{n-r}$;	5. $P_m=\mathsf{First}_r(C_m\oplus\mathsf{pad})$; $T_m=\mathsf{First}_r(C_m)\|\|0^{n-r}$;
6. $\mathsf{sum}=P_1\oplus\cdots\oplus P_{m-1}\oplus T_m\oplus\Gamma_{\gamma,m+1}\oplus\delta\oplus\mathsf{pad}$;	6. $\mathsf{sum}=P_1\oplus\cdots\oplus P_{m-1}\oplus T_m\oplus\Gamma_{\gamma,m+1}\oplus\delta\oplus\mathsf{pad}$;
7. $\mathsf{tag}=\pi(\mathsf{sum})$;	7. $\mathsf{tag}=\pi(\mathsf{sum})$;
return $(C_1\|\|\cdots\|\|C_{m-1}\|\|C_m,\mathsf{tag})$.	return $(P_1\|\|\cdots\|\|P_{m-1}\|\|P_m,\mathsf{tag})$.

Table 9. Encryption and decryption algorithms for PAE2.

$\mathsf{PAE2.Encrypt}_{\pi,\mathsf{fStr}}(N,P)$:	$\mathsf{PAE2.Decrypt}_{\pi,\mathsf{fStr}}(N,C,\mathsf{tag})$:
1. $\delta_0=\pi(\mathsf{fStr})$; $\delta_1=\Gamma_{\delta_0,1}$;	1. $\delta_0=\pi(\mathsf{fStr})$; $\delta_1=\Gamma_{\delta_0,1}$;
2. $(C,\mathsf{tag})=\mathsf{Forward2}_{\pi,\delta_1}(N,P)$;	2. $(P,\mathsf{tag}')=\mathsf{Backward2}_{\pi,\delta_1}(N,C)$;
3. return (C,tag).	3. if $\mathsf{tag}=\mathsf{tag}'$ return P; else return $\perp$.

Table 10. Encryption and decryption algorithms for PAEAD2.

$\mathsf{PAEAD2.Encrypt}_{\pi,\mathsf{fStr}}(N,H,P)$:	$\mathsf{PAEAD2.Decrypt}_{\pi,\mathsf{fStr}}(N,H,C,\mathsf{tag})$:
1. $\delta_0=\pi(\mathsf{fStr})$; $\delta_1=\Gamma_{\delta_0,1}$;	1. $\delta_0=\pi(\mathsf{fStr})$; $\delta_1=\Gamma_{\delta_0,1}$;
2. if $\|H\|=0$, $\mathsf{tag}_2=0^n$;	2. if $\|H\|=0$, $\mathsf{tag}_2=0^n$;
3. else $\mathsf{tag}_2=\mathsf{PAuth}_{\pi,\delta_0}(H)$;	3. else $\mathsf{tag}_2=\mathsf{PAuth}_{\pi,\delta_0}(H)$;
4. $(C,\mathsf{tag}_1)=\mathsf{Forward2}_{\pi,\delta_1}(N,P)$;	4. $(P,\mathsf{tag}_1)=\mathsf{Backward2}_{\pi,\delta_1}(N,C)$;
5. return $(C,\mathsf{tag}_1\oplus\mathsf{tag}_2)$.	5. if $\mathsf{tag}=(\mathsf{tag}_1\oplus\mathsf{tag}_2)$ return P; else return $\perp$.

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

$\overrightarrow{\mathsf{PAEAD2}}.\mathsf{Encrypt}_{\pi,\mathsf{fStr}}(N,H_1,\dots,H_k,P)$:	$\overrightarrow{\mathsf{PAEAD2}}.\mathsf{Decrypt}_{\pi,\mathsf{fStr}}(N,H_1,\dots,H_k,C,\mathsf{tag})$:
1. $\delta_0=\pi(\mathsf{fStr})$; $\delta_1=\Gamma_{\delta_0,1}$;	1. $\delta_0=\pi(\mathsf{fStr})$; $\delta_1=\Gamma_{\delta_0,1}$;
2. if $k=0$, $\mathsf{tag}_2=0^n$;	2. if $k=0$, $\mathsf{tag}_2=0^n$;
3. else $\mathsf{tag}_2=\overrightarrow{\mathsf{PAuth}}_{\pi,\delta_0}(H_1,\dots,H_k)$;	3. else $\mathsf{tag}_2=\overrightarrow{\mathsf{PAuth}}_{\pi,\delta_0}(H_1,\dots,H_k)$;
4. $(C,\mathsf{tag}_1)=\mathsf{Forward2}_{\pi,\delta_1}(N,P)$;	4. $(P,\mathsf{tag}_1)=\mathsf{Backward2}_{\pi,\delta_1}(N,C)$;
5. return $(C,\mathsf{tag}_1\oplus\mathsf{tag}_2)$.	5. if $\mathsf{tag}=(\mathsf{tag}_1\oplus\mathsf{tag}_2)$ return P; else return $\perp$.

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 5. *Let $\sigma \geq q \geq 1$. Then*

$$\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\overrightarrow{\mathsf{PAEAD1}}}(q,\sigma) \leq \frac{2^{n-t}}{2^n-(\sigma+q+3)} + \frac{(\sigma+q+2)^2}{2^n} + \frac{q(q-1)}{2^{n+1}};$$
$$\mathbf{Adv}^{\text{aead-auth}}_{t\text{-}\overrightarrow{\mathsf{PAEAD2}}}(q,\sigma) \leq \frac{2^{n-t}}{2^n-(\sigma+q+3)} + \frac{(\sigma+q+2)^2}{2^n} + \frac{q(q-1)}{2^{n+1}}.$$

8 Deterministic Authenticated Encryption with Associated Data

The basic idea behind the DAE and the DAEAD schemes is based on the SIV construction in [53]. 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 [60]) 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 $\perp$ 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. $\mathsf{tag}=\mathsf{PAuth}_{\pi,\mathsf{fStr}}(P)$;	1. $(C_1,\ldots,C_m)=\mathsf{Format}(C,n)$;
2. $(P_1,\ldots,P_m)=\mathsf{Format}(P,n)$;	2. $(P_1,\ldots,P_{m-1})=(C_1,\ldots,C_{m-1})$ $\oplus\,\mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;
3. $(C_1,\ldots,C_{m-1})=(P_1,\ldots,P_{m-1})$ $\oplus\,\mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;	3. $P_m=\mathsf{First}_r(C_m\oplus\pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;
4. $C_m=\mathsf{First}_r(P_m\oplus\pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;	4. $P=P_1\|\cdots\|P_{m-1}\|P_m$;
5. $C=C_1\|\cdots\|C_{m-1}\|C_m$;	5. $\mathsf{tag}_1=\mathsf{PAuth}_{\pi,\mathsf{fStr}}(P)$;
6. return (C,tag).	6. 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. $\mathsf{tag}=\overrightarrow{\mathsf{PAuth}}_{\pi,\mathsf{fStr}}(H_1,\ldots,H_k,P)$;	1. $(C_1,\ldots,C_m)=\mathsf{Format}(C,n)$;
2. $(P_1,\ldots,P_m)=\mathsf{Format}(P,n)$;	2. $(P_1,\ldots,P_{m-1})=(C_1,\ldots,C_{m-1})$ $\oplus\,\mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;
3. $(C_1,\ldots,C_{m-1})=(P_1,\ldots,P_{m-1})$ $\oplus\,\mathsf{ecb}_\pi(\mathsf{tag}\oplus\mathsf{bin}_n(1),\ldots,\mathsf{tag}\oplus\mathsf{bin}_n(m-1))$;	3. $P_m=\mathsf{First}_r(C_m\oplus\pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;
4. $C_m=\mathsf{First}_r(P_m\oplus\pi(\mathsf{tag}\oplus\mathsf{bin}_n(m)))$;	4. $P=P_1\|\cdots\|P_{m-1}\|P_m$;
5. $C=C_1\|\cdots\|C_{m-1}\|C_m$;	5. $\mathsf{tag}_1=\overrightarrow{\mathsf{PAuth}}_{\pi,\mathsf{fStr}}(H_1,\ldots,H_k,P)$;
6. return (C,tag).	6. 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 6. *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. 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 [52], OCB2 [51] and OCB3 [40] are also Rate-1 parallelisable schemes. Of these, OCB3 is the latest version and is currently called OCB.

Comparison to OCB [40]. 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 $n_1 = 1$ and $n_2 = 128$ and use the available 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.

TWEAKING THE AE(AD) SCHEMES. The n-bit string fStr which parameterises the constructions is assumed to be publicly known. The security analysis argues over the randomness in π. The string fStr can be considered to be a tweak to the schemes, i.e., by varying fStr it is possible to obtain different schemes. It is to be noted that the mask γ is obtained by applying π to $N \oplus \delta_1$ where $\delta_1 = \psi(\pi(\mathsf{fStr}))$. Since γ depends on fStr, all the ciphertext blocks and the tag also depends on fStr. It is possible to use fStr as a secret key. For one thing, this increases the resistance of the schemes to certain attacks in the multi-user setting [16]. Additionally, this provides another way of achieving 'security by obscurity' over and above the usual provable security guarantee.

It is to be noted that we do not make any provable security claims when we point out the possibility of 'security by obscurity' obtained by secretly choosing fStr and/or (ρ, μ).

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 $m + h + 4$ ($m + 3$ calls to process the message; $h + 1$ calls to process the header) where there are two successive calls on fStr. These two calls 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 the total number of calls is $m + 3$, out of which one call is required per session and $m + 2$ calls are required per message.

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.

SET-UP TIME. OCB requires an initial time to prepare the table storing the masks to be used during actual encryption. This is a one-time task and the stored array is used for successive messages in a session. This strategy is useful if there are many messages in one session. For short sessions and also for short messages, the overhead of computing the stored table will become significant. The AEAD algorithm considered here have a minimal set-up time. The AE algorithms require just one application of π on fStr and if there is a header then another application of π is required. The smaller set-up time makes the new schemes attractive across a wider range of session lengths and message lengths.

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 [53]. 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 [2], but, a serious flaw has been pointed out in the construction by Nandi [49].

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 $m + 3$ which is smaller than CLOC for all $m \geq 2$. If $h > 0$, then the number of required calls is $m + h + 4$ which is smaller than or equal to that of CLOC for $m \geq 4$. Storing two n-bit quantities (the result of the successive encryptions of fStr) reduces the number of calls to $m + h + 2$ which is smaller than that of CLOC for all $m \geq 2$. 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 on 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 [12]. 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 [53] 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 [53] uses $\mathrm{Ctr} + i$ as the i-th offset. This is different from the somewhat simpler CTR mode from [60] that we use which has $\mathrm{Ctr} \oplus \mathsf{bin}_n(i)$ as the i-th offset.

Improvements to the work in [53] 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. As a result, during decryption, the processing of the ciphertext and that of the header can be carried out in parallel. This is an advantage in hardware.

McOE-G [24], COPA [1] and POET [43] are three recent examples of nonce-misuse resistant AEAD modes of operations. (Attacks on POET have been described in [49].) 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 [12] 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 [15]. 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 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.

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 produce 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.

Some code fragments are provided using Intel intrinsics instructions. Explanations of the instructions used are provided in Appendix A. Intel provides a wide variety of instructions and the real challenge is to choose the proper instructions which lead to efficient implementations.

Implementing `xtimes`: First consider the implementation of the mask where $n_1 = 1$ and $n_2 = 128$. This operation is popularly known as `xtimes` or *doubling*. A basic description of this operation is the following: Given an input γ, the output $\beta = (\gamma \ll 1) \oplus (\mathsf{msb}(\gamma) \cdot \texttt{const})$ where $\mathsf{msb}(\gamma)$ is the most significant bit of γ and `const` is a 128-bit string representing $x^{128} \bmod \mu(x)$. For the $\mu(x)$ given in Table 2 corresponding to $n_1 = 1$ and $n_2 = 128$, `const` = 0x87.

The problem of implementing the above procedure using an XMM register is that there is no instruction to left shift the content of an XMM register by one bit. There is, however, an instruction which does the following: consider an 128-bit register as two 64-bit words and perform a single bit left shift on both the words. To utilise this instruction requires some prepatory work. The piece of code using Intel intrinsics implementing the procedure is shown in Figure 2.

One aspect of this code is that it requires only the 32 least significant bits of `const` to have ones and the 96 other bits must all be zero. For `const` = 0x87 this property holds. On the other hand, this limits the reconfigurability of the code to a certain extent. The resulting code is only partially configurable in the following sense. Suppose $\mu(x)$ is changed to another primitive polynomial and `const` is changed to the new value of $x^{128} \bmod \mu(x)$. The existing code will run only if it still holds that only the last 32 bits of the new `const` has ones.

For a fully reconfigurable code, it should be possible to change $\mu(x)$ to *any* other primitive polynomial and simply changing `const` to the corresponding $x^{128} \bmod \mu(x)$ will allow the code to run correctly. It turns out that such a reconfigurable code requires a few more instructions. The Intel intrinsics code for reconfigurably computing `xtimes` is shown in Figure 3.

There are other strategies to implement `xtimes` using the 128-bit registers. We discuss two specific strategies adopted in [3] and [12].

1. The strategy adopted in [3] 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 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 [12] (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.

The latencies and throughput of the instructions for the Haswell processor are given in Table 23 in Appendix A. From this it can be seen that the strategy in [3] uses the least number of instructions, and is

[3] The specific code discussed in [12] is incorrect. The instruction `PCMPGTB` does comparison of signed 8-bit integers, and this is not considered in the code described in [12].

```
computeMaskXtimes(a) { \
__m128i b; \
        b = _mm_srai_epi32(a,31); \
        b = _mm_shuffle_epi32(b,0x57); \
        b = _mm_and_si128(b,_mm_set_epi32(0x00,0x01,0x00,0x87)); \
        b = _mm_xor_si128(b,_mm_slli_epi64(a,0x01)); \
        a = b; \
}
```

Fig. 2. The code for computing `xtimes` for $n_1 = 1$ and $n_2 = 128$ and the polynomial given in Table 2.

```
computeMaskXtimesR(a) { \
__m128i res, t1; \
        t1 = _mm_srai_epi32(a,31); \
        res = _mm_shuffle_epi32(t1,0xff); \
        res = _mm_and_si128(res,_mm_set_epi32(0x00,0x00,0x00,0x87)); \
        t1 = _mm_shuffle_epi32(t1,0x55); \
        t1 = _mm_and_si128(t1,_mm_set_epi32(0x00,0x01,0x00,0x00)); \
        t1 = _mm_xor_si128(t1,_mm_slli_epi64(a,0x01)); \
        t1 = _mm_xor_si128(t1,res); \
        a = t1; \
}
```

Fig. 3. The code for fully reconfigurable computation of `xtimes` for $n_1 = 1$ and $n_2 = 128$.

```
__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 [3]; (b) The code for `xtimes` described in [12].

also optimal in terms of the latencies, but it uses a table lookup and it is difficult to estimate the latency associated with a table look-up. The strategy in [12] 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 [12] and [3]. Moreover, both the strategies in [3] and [12] suffer 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 SSE instructions has not been considered in the literature. For implementing them, we use a similar strategy as in computing `xtimes`. In particular, we avoid data dependent branchings and/or table lookups.

The generic strategy for implementing these masks is the following. As discussed in Section 3, consider the implementation of $\mathbb{F}_{2^n}$ by a pair of polynomials $(\rho(\alpha), \mu(x))$ where $\mu(x) = x^{n_2} \oplus t_{n_2-1}x^{n_2-1} \oplus \cdots \oplus t_1 x \oplus \alpha$ with t_i's to be either 0 or 1. Suppose $\gamma = (a_{n_2-1}, \ldots, a_0)$ and $\psi(\gamma) = \beta = (b_{n_2-1}, \ldots, b_0)$ with $a_i, b_j \in \mathbb{F}_{2^{n_1}}$. Then γ and β can be obtained from each other using the following relations.

$$\beta = (a_{n_2-2}, \ldots, a_0, 0^{n_1}) \oplus (a_{n_2-1}t_{n_2-1}, \ldots, a_{n_2-1}t_1, \alpha a_{n_2-1}); \tag{14}$$

$$\gamma = (\alpha^{-1}b_0, b_{n_2-1}, \ldots, b_1) \oplus (0^{n_1}, t_{n_2-1}\alpha^{-1}b_0, \ldots, t_1\alpha^{-1}b_0). \tag{15}$$

We discuss the computation of β from γ. Similar issues apply to the inverse computation. If n_1 is a multiple of 8, then computing $(a_{n_2-2}, \ldots, a_0, 0^{n_1})$ from $(a_{n_2-1}, \ldots, a_0)$ can be done using a single operation. Depending on the value of n_1, the availability of instructions (and hence the total number of instructions) required to compute $(0^{n_1}, t_{n_2-1}\alpha^{-1}b_0, \ldots, t_1\alpha^{-1}b_0)$ varies. The cases $n_1 = 16$ and $n_1 = 32$ require the same number of instructions which is lower than the number of instructions required for the cases $n_1 = 8$ and $n_1 = 64$. The case $n_1 = 64$ requires the maximum number of instructions. We provide the code for the case $n = 32$ in Figure 5.

In all cases, the required number of instructions does not depend on the weight of the polynomial $\mu(x)$, i.e., the number of instructions is independent of the number of t_i's which are 1. Further, the implementations support full reconfigurability. Changing the pair $(\rho(\alpha), \mu(x))$ only requires changing some constants in the code.

```
#define computeMask32(a) { \
       __m128i t1, res; \
 \
       res = _mm_srai_epi32(a,31); \
       res = _mm_and_si128(res, _mm_set_epi32(0x0a000021,0x00,0x00,0x00)); \
       res = _mm_xor_si128(_mm_slli_epi32(a,1),res); \
       res = _mm_srli_si128(res,12); \
\
        t1= _mm_shuffle_epi8(a, _mm_set_epi32(0x0f0e0d0c,0xffffffff,0x0f0e0d0c,0xffffffff));\
       res = _mm_xor_si128(res,t1); \
       res = _mm_xor_si128(res, _mm_slli_si128(a,4)); \
       a = res; \
}
```

Fig. 5. The code for the case $n_1 = 32$ and $n_2 = 4$.

Experimental Results: There are six options for generating the masks which correspond to the five representations of $GF(2^{128})$ given in Table 2. Type-0 and Type-0R correspond to maskings strategies

$n_1 = 1$, $n_2 = 128$	$n_1 = 1$, $n_2 = 128$	$n_1 = 8$, $n_2 = 16$	$n_1 = 16$, $n_2 = 8$	$n_1 = 32$, $n_2 = 4$	$n_1 = 64$, $n_2 = 2$
Type-0	Type-0R	Type-1	Type-2	Type-3	Type-4

Table 14. Numbering the different types of maskings. Type-0 and Type-0R compute the same function. The difference is that Type-0R is fully reconfigurable whereas Type-0 is only partially reconfigurable.

given in Figures 2 and 3 respectively. We use the notation given in Table 14 to denote the different masking types.

Experimental performance data for the various masking schemes are shown in Table 15. 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. This experiment was repeated 1024 times and the row **1 mask** in Table 15 shows the median 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.

Type	`xtimes`		**0**	**0R**	**1**	**2**	**3**	**4**
	[3]	[12]						
1 mask	16.30	11.42	4.10	6.17	5.91	5.54	5.54	6.21
8 masks	2.41	4.25	1.77	2.54	3.06	2.87	2.85	3.21

Table 15. Experimental comparison of the number of cycles required for the various masking strategies.

Table 15 clearly shows that for implementing `xtimes`, Type-0 masking is better than the ones used in [3] and [12]. The principal reason being that our strategy does not use any conditional branching. Such branchings may lead to higher latencies. The strategy depicted in [3] does not involve any branchings, but involves a data dependent table look up which has a performance penalty. Type-0R masking provides a constant-time implementation of `xtimes` (without conditional branchings or table look-ups) in a fully reconfigurable manner. The time taken by this masking is more than Type-0 masking as is to be expected. It is to be noted that times for Type-0R are better than the times for `xtimes` reported in [3, 12]. For generating a single mask, the time for Type-0R masking is more than those of Type-1 to Type-4 maskings, but, for batch computation of eight masks, the time taken by Type-0R masking becomes lower than the types for these other masking strategies. Lastly, we note that the entries are the number of cycles for computing a 16-byte mask. So, in terms of cycles/byte parameter, there is not much difference between the different masking strategies.

Note that when we use maskings for implementing the various schemes, we compute the masks in batches whenever possible. So, the figures provided for batch computation of the masks are really relevant when compared against the timing figures of the different schemes.

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(R) Core(TM) i5-4200U (Haswell) CPU @1.60GHz GHz.
cache size:	3072 KB.
Memory:	7.7 GiB.
OS:	Ubuntu 14.04 LTS.
Kernel:	3.13.0-32-generic.
Compiler:	GCC version 4.8.2.
Compile flags:	`-finline-functions -maes -msse4.2 -m64 -O3`.

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. Following the framework for timing measurement used in `http://web.cs.ucdavis.edu/~rogaway/ocb/news/code/timing_x86.c` (accessed on 24th December, 2014), for each scheme X we did the following:

1. Ran X 10000 times on the same data, to ensure that the data and code gets into the cache.
2. Then we ran X 1024 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 1024 times and we report the median of these 1024 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 14. 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 16 to 19. For the AE schemes, the quantity $\delta_0 = \pi(\mathsf{fStr})$ was pre-computed and for the AEAD schemes both $\delta_0 = \pi(\mathsf{fStr})$ and $\pi(\delta_0)$ were pre-computed.

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 20 to 22. Based on this data, we make the following observations.

1. Performance figures for Type-0 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 SSE code for Type-0 masking has the least number of instructions and also the least latency among all the five masking strategies.
2. 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 16. The column 'Avg' reports the average over messages of lengths 1 to 1024 bytes and the performance figure for $\overrightarrow{\mathsf{PAuth}}$ is more than 7 cycles/byte which is substantially higher than the figure of around 2 cycles/byte for the single-input scheme PAuth. On the other hand, if we consider the columns '4KiB' and 'ipi', the performance figures for the vector version go down quite sharply but still the difference to the single-input version is significant.
3. 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 16 and that for AE given in Table 17.
4. Considering Tables 16 and 17, 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 point 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.
5. Considering Table 18, 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.
6. From Table 18, 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.

7. From Table 18, 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.
8. From Table 19, for the 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 SSE instructions. Similarly, looking at Table 21, we find a similar effect of the number of components on the performance figures of the vector versions of the authentication schemes.
9. The effect of message lengths on the vector versions of the authentication schemes with a fixed number of components is shown in Table 20. These show that there is a substantial overhead for very small messages which drops off rather sharply as the message length becomes moderate.
10. Table 22 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-0 masking given in Table 16. These show that the additional cost of AEAD over authentication is small.
11. 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 SSE instructions cannot be properly utilised.

	Type-0			Type-0R			Type-1			Type-2			Type-3			Type-4		
	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi
PAuth	1.72	0.85	1.13	1.80	0.91	1.20	1.87	0.98	1.27	1.84	0.97	1.27	1.85	0.97	1.25	1.89	1.01	1.32
$\overrightarrow{\mathsf{PAuth}}$	7.61	2.91	4.30	7.70	2.99	4.38	7.74	3.09	4.45	7.71	3.07	4.42	7.71	3.07	4.42	7.92	3.08	4.48

Table 16. Performance figures in cycles/byte for PAuth and its vector version. For the measurements of $\overrightarrow{\mathsf{PAuth}}$, a message was divided into four almost equal length portions and the algorithms were invoked on the resulting 4-component vector.

Comparing performance to OCB: Performance results for OCB on modern Intel processors are given in [40] and [12]. Results in [40] are reported for the 'Clarkdale' processor while the results in [12] 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, [12] reports performance of 0.81 cpb for 2048 byte messages.

Performance figures for PAE1 (resp. PAE2) with Type-0 masking on Haswell processor are 0.95 cpb (resp. 0.96) for 2048-byte messages. For the other masking types, for 4096-byte message, the maximum is for Type-4 masking where the figure of 1.08 cpb is obtained. 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 or PAE2 by a fraction of a cpb. This small speed loss should be tolerable in view of the several features including reconfigurability mentioned in Section 9.

Possible improvements to the code:

1. For utilising the pipelining of the AES instructions, we have organised the codes so as to process eight blocks at a time. This requires unrolling the loop for eight blocks. For one thing, it is possible

	Type-0			Type-0R			Type-1			Type-2			Type-3			Type-4		
	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi	Avg	4KiB	ipi
PAE1	2.40	0.90	1.29	2.49	0.98	1.37	2.60	1.14	1.54	2.54	1.04	1.43	2.55	1.04	1.43	2.61	1.08	1.71
PAE2	2.40	0.90	1.30	2.49	0.99	1.38	2.58	1.10	1.47	2.53	1.04	1.43	2.56	1.04	1.43	2.60	1.09	1.50
DAE	3.28	1.43	2.03	3.38	1.54	2.10	3.43	1.61	2.18	3.41	1.59	2.17	3.41	1.60	2.17	3.55	1.66	2.28
PAEAD1	1.19	0.92	1.06	1.31	1.01	1.15	1.36	1.14	1.25	1.34	1.06	1.23	1.33	1.06	1.20	1.38	1.11	1.25
PAEAD2	1.20	0.92	1.07	1.31	1.00	1.16	1.35	1.12	1.24	1.32	1.06	1.20	1.32	1.05	1.20	1.38	1.11	1.25
DAEAD	3.68	3.38	3.54	3.79	3.51	3.71	3.81	3.54	3.73	3.79	3.51	3.69	3.79	3.51	3.72	3.82	3.54	3.72

Table 17. 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	5.04	3.33	2.39	1.70	1.27	1.05	0.95	0.90
PAE2	5.04	3.36	2.45	1.73	1.28	1.06	0.96	0.90
DAE	9.83	5.89	2.91	2.16	1.79	1.58	1.51	1.47
Forward1	5.03	3.32	2.38	1.68	1.26	1.05	0.95	0.90
Forward2	5.03	3.33	2.43	1.72	1.28	1.06	0.96	0.90
Ctr	2.67	1.76	0.67	0.65	0.63	0.62	0.61	0.60

Table 18. 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
$\overrightarrow{\mathsf{PAEAD1}}$	2.36	1.16	1.84	2.51	1.19	1.94	2.64	1.21	2.02	2.77	1.24	2.11	4.06	1.50	2.97
$\overrightarrow{\mathsf{PAEAD2}}$	2.37	1.16	1.85	2.51	1.19	1.94	2.64	1.21	2.02	2.77	1.24	2.11	4.06	1.50	2.98
DAEAD	3.66	3.37	3.55	3.77	3.39	3.66	3.91	3.42	3.72	4.08	3.45	3.86	5.44	3.71	4.75

Table 19. Performance figures in cycles/byte for encryption using $\overrightarrow{\mathsf{PAEAD1}}$, $\overrightarrow{\mathsf{PAEAD2}}$ and DAEAD where the number of headers vary. Type-0 masking has been used. For computing average, the header lengths were kept fixed and only the message lengths were varied from 1 byte to 1024 bytes.

	1-64	65-128	129-256	257-512	513-1024
$\overrightarrow{\mathsf{PAuth}}$	36.81	14.84	8.30	5.40	4.07

Table 20. Average of cycles/byte figures for authentication of small messages using $\overrightarrow{\mathsf{PAuth}}$. 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 algorithm was invoked on the resulting 4-component vector. Type-0 masking was used in each case.

	512	256+256	$256 + 2 \times 128$	4×128	8×64
$\overrightarrow{\mathsf{PAuth}}$	3.30	3.55	3.79	4.01	6.36

Table 21. Average of cycles/byte figures for authentication of small messages using $\overrightarrow{\mathsf{PAuth}}$ with varying number of components. The column headings indicate the number and the lengths in bytes of the different components. Type-0 masking was used in each case.

$\overrightarrow{\mathsf{PAEAD1}}$	7.67
$\overrightarrow{\mathsf{PAEAD2}}$	7.72

Table 22. 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-0 masking was used in each case.

to unroll loop for a larger number of blocks. More importantly and especially for smaller messages, it should be advantageous to unroll loop for x blocks for each value of x from 1 to 8.

2. We have used only SSE instructions in developing our code. Carefully, using the later AVX instructions can possibly lead to improvements in the speed.

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. 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.
2. Elena Andreeva, Atul Luykx, Bart Mennink, and Kan Yasuda. COBRA: A Parallelizable Authenticated Online Cipher Without Block Cipher Inverse. In *FSE*, 2014. to appear.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.

8. Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.
9. Daniel J. Bernstein. Stronger security bounds for Wegman-Carter-Shoup authenticators. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2005.
10. John Black 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.
11. 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.
12. 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/`.
13. CAESAR. Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`.
14. 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.
15. Debrup Chakraborty and Palash Sarkar. 'C' Code for Reference and Fast Implementations of Various Block Cipher Based Modes of Operations. `https://docs.google.com/file/d/0B7cNoZ_Dy-EhbUEtOE1xLWQzNDQ/`, 2015.
16. Sanjit Chatterjee, Alfred Menezes, and Palash Sarkar. Another look at tightness. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 293–319. Springer, 2011.
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 Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2014.
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. Mridul Nandi. Forging attacks on two authenticated encryption schemes COBRA and POET. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2014.
50. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.
51. 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.
52. 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.
53. 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.
54. Palash Sarkar. Pseudo-random functions and parallelizable modes of operations of a block cipher. *IEEE Transactions on Information Theory*, 56(8):4025–4037, 2010.
55. Palash Sarkar. A simple and generic construction of authenticated encryption with associated data. *ACM Trans. Inf. Syst. Secur.*, 13(4):33, 2010.
56. 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.
57. 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.
58. Advanced Encryption Standard. Federal Information Processing Standard Publication 197, 2002. Available at `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
59. American National Standard Protocol Specification For Interfacing to Data Communication Networks. ANSI C12.22-2008, 2008.

60. 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.
61. 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.
62. 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 Some Intel Intrinsics Instructions

In Table 23 we provide descriptions of some of the Intel intrinsics instructions that have been used to implement the finite field arithmetic for mask generation. Also included are the instructions used in [3, 12] for implementing `xtimes`. The descriptions of the instructions are taken from

`https://software.intel.com/sites/landingpage/IntrinsicsGuide/`.

Each of these instructions return a 128-bit value which is assumed to be stored in `dst`. With the help of the provided descriptions the reader will be able to understand the code fragments provided in the main text. The latency and throughput figures are for the Haswell processor.

The descriptions of the instructions in Table 23 are self-explanatory. Only the instruction `_mm_shuffle_epi32(a, imm)` requires further explanation. It does the following: Let $f : \{0,1\}^2 \rightarrow \{0,1\}^2$ be any function; `imm` encodes $f(11)||f(10)||f(01)||f(00)$ as an 8-bit integer; write the 128-bit operand $\mathtt{a}$ as $\mathtt{a}_3\mathtt{a}_2\mathtt{a}_1\mathtt{a}_0$; then the output is $\mathtt{a}_{f(3)}\mathtt{a}_{f(2)}\mathtt{a}_{f(1)}\mathtt{a}_{f(0)}$, where $\{0,1\}^2$ is identified with $\{0,1,2,3\}$ in the usual manner.

B Analysis of PAuth

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 [54]. 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 [54], this structure is formalised by the next definition.

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

Table 23. Explanations of some Intel intrinsics instructions.

instruction	explanation	latency	throughput
`__m128i _mm_srai_epi32 (__m128i a, int imm)`	Shift packed 32-bit integers in `a` right by `imm` while shifting in sign bits and store the results in `dst`.	1	–
`__m128i _mm_shuffle_epi32 (__m128i a, int imm)`	Shuffle 32-bit integers in `a` using the control in `imm` and store the results in `dst`.	1	1
`__m128i _mm_set_epi32 (__m128i a, int imm)`	Set packed 32-bit integers in `dst` with the supplied values.		
`__m128i _mm_set1_epi8 (__m128i a, int imm)`	Broadcast 8-bit integer `a` to all elements of `dst`.		
`__m128i _mm_slli_epi64 (__m128i a, int imm)`	Shift packed 64-bit integers in `a` left by `imm` while shifting in zeros, and store the results in `dst`.	1	1
`__m128i _mm_srli_epi64 (__m128i a, int imm)`	Shift packed 64-bit integers in `a` right by `imm` while shifting in zeros, and store the results in `dst`.	1	1
`__m128i _mm_slli_si128 (__m128i a, int imm)`	Shift `a` left by `imm` bytes while shifting in zeros, and store the results in `dst`.	1	0.5
`__m128i _mm_or_si128 (__m128i a, int imm)`	Compute the bitwise OR of 128 bits (representing integer data) in `a` and `b`, and store the result in `dst`.	1	0.33
`__m128i _mm_and_si128 (__m128i a, int imm)`	Compute the bitwise AND of 128 bits (representing integer data) in `a` and `b`, and store the result in `dst`.	1	0.33
`__m128i _mm_xor_si128 (__m128i a, int imm)`	Compute the bitwise XOR of 128 bits (representing integer data) in `a` and `b`, and store the result in `dst`.	1	0.33
`__m128i _mm_movemask_pd (__m128i a)`	Set each bit of mask `dst` based on the most significant bit of the corresponding packed double-precision (64-bit) floating-point element in `a`.		
`__m128i _mm_extract_epi8 (__m128i a)`	Extract an 8-bit integer from `a`, selected with `imm`, and store the result in the lower element of `dst`.	2	–
`__m128i _mm_cmpgt_epi8 (__m128i a)`	Compare packed 8-bit integers in `a` and `b` for greater-than, and store the results in `dst`.	1	0.5

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 [54].

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 $x, x' \in \mathcal{X}$,*

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

The following result relates the collision probabilities to the PRF-advantage and follows directly from Theorem 2 in [54].

Theorem 7. *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 $\sigma \geq q$*

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

The proof is based on Theorem 7 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 $\kappa = \pi(\mathsf{fStr})$, $C_i = \pi(D_i)$ for $1 \leq i \leq m$ and

$$D_i = \begin{cases} P_1 \oplus \mathit{\Gamma}_{\kappa,-1} & i = m = 1, r < n; \\ P_1 \oplus \mathit{\Gamma}_{\kappa,-2} & i = m = 1, r = n; \\ P_i \oplus \mathit{\Gamma}_{\kappa,i} & m > 1, 1 \leq i \leq m-1; \\ C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m & m > 1, i = m, r = n; \\ C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \mathit{\Gamma}_{\kappa,m} & m > 1, i = m, r < n. \end{cases} \quad (16)$$

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 7, 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, $D_j = \Gamma_{\kappa,j} + P_j$. 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 $D_i = P_i \oplus \Gamma_{\kappa,i}$, $1 \leq i \leq m - 1$; and $D'_k = P'_k \oplus \Gamma_{\kappa,k}$, $1 \leq k \leq m' - 1$. 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.

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

If $j < m'$, then since $P_j \neq P'_j$, $D_j = P_j \oplus \Gamma_{\kappa,j} \neq P'_j \oplus \Gamma_{\kappa,j} = D'_j$ 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 Γ given in Definition 1. So,

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

We have

$$\begin{aligned} \Pr[D_m \neq D'_{m'}] &\geq \Pr[(D_m \neq D'_{m'}) \wedge \mathcal{E}] \\ &= \Pr[(D_m \neq D'_{m'}) | \mathcal{E}] \Pr[\mathcal{E}] \\ &\geq \Pr[(D_m \neq D'_{m'}) | \mathcal{E}] \times \left(1 - \frac{m + m' - 3}{2^n} \right). \end{aligned} \tag{17}$$

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 $t \geq 0$ 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 $t \leq m + m' - 3$. 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 $t \geq 0$ 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, $\Pr[D_m \neq D'_{m'}|\mathcal{E}] = \Pr[C_j \oplus Q_1 \oplus \cdots \oplus Q_t \neq P_m \oplus P'_{m'}|\mathcal{E}] \geq 1 - 1/(2^n - t) \geq 1 - 1/(2^n - (m + m' - 3)) \geq 1 - 2/2^n$ (assuming $m + m' - 3 \leq 2^{n-1}$).

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

From this and (17) we have $\Pr[D_m \neq D'_{m'}] \geq (1 - (m + m' - 1)/2^n)$ and so $\Pr[D_m = D'_{m'}] \leq (m + m')/2^n$.

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

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

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

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

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, $D_m \oplus D'_{m'} = \mathit{\Gamma}_{\kappa,m}$ 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 $D_1 \oplus D'_1 = P_1 \oplus P'_1 \oplus \mathit{\Gamma}_{\kappa,-2} \oplus \mathit{\Gamma}_{\kappa,-1}$. 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 $\mathit{\Gamma}_{\kappa,-1}$ if $r' < n$ and $\mathit{\Upsilon}$ is $\mathit{\Gamma}_{\kappa,-2}$ 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. $\Pr[D_m = D'_i] \le 2/2^n$ *for* $1 \le i \le m'-1$*;*
2. $\Pr[D_m = D_i] \le 2/2^n$ *for* $1 \le i \le m-1$*;*
3. $\Pr[D_m = \mathsf{fStr}] \le 1/2^n$*.*

Proof : First suppose $m = 1$. Then $D_1 = P_1 \oplus \Gamma_{\kappa,-1}$ or $D_1 = P_1 \oplus \Gamma_{\kappa,-2}$ according as $r < n$ or $r = n$. Point 3 follows from this. For $1 \le i \le m'-1$, $D'_i = P'_i \oplus \Gamma_{\kappa,i}$. So from Definition 1, $\Pr[D_1 = D'_i] \le 2/2^n$ 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 $D_m = C_1 \oplus \cdots \oplus C_{m-1} \oplus P_m \oplus \Gamma_{\kappa,m}$ according as $r = n$ or $r > n$. A straightforward analysis now shows the result. □

Consequently, $\Pr[\overline{\mathsf{Pairwise\text{-}Disjoint}(x,x')}] \le (m+m')/2^n$ and $\Pr[\overline{\mathsf{Self\text{-}Disjoint}(x)}] \le 2m/2^n$. Using Theorem 7 with $\varepsilon = \varepsilon_1 = \varepsilon_2 = 2/2^n$ gives Theorem 2.

C Security Argument for the AE(AD) Schemes

The security analysis that we perform is quite standard. In particular, we do not introduce any new proof technique in the analysis. We first consider PAE1 and PAEAD1. The modifications required to obtain the bounds for PAE2 and PAEAD2 are mentioned later.

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 $(P^{(s)}_1, \ldots, P^{(s)}_{m^{(s)}})$.

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

$$A^{(s)}_i = \begin{cases} N^{(s)} \oplus \delta_1 & \text{if } i = 0; \\ P^{(s)}_i \oplus \Gamma_{\gamma^{(s)},i} & \text{for } 1 \le i \le m^{(s)}-1; \\ \mathsf{bin}_n\left(r^{(s)}\right) \oplus \Gamma_{\gamma^{(s)},m^{(s)}} & \text{if } i = m^{(s)}; \\ \left.\begin{array}{l} \mathsf{sum}^{(s)} \\ \quad = P^{(s)}_1 \oplus \cdots \oplus P^{(s)}_{m^{(s)}-1} \oplus \left(\mathsf{First}_{r^{(s)}}\left(C^{(s)}_{m^{(s)}}\right) || 0^{n-r^{(s)}}\right) \\ \qquad \oplus \Gamma_{\gamma^{(s)},m^{(s)}+1} \oplus \delta_1 \oplus \mathsf{pad}^{(s)} \end{array}\right\} & \text{if } i = m^{(s)}+1. \end{cases}$$

$$B^{(s)}_i = \begin{cases} \gamma^{(s)} & \text{if } i = 0; \\ C^{(s)}_i \oplus \Gamma_{\gamma^{(s)},i} & \text{for } 1 \le i \le m^{(s)}-1; \\ \mathsf{pad}^{(s)} & \text{if } i = m^{(s)}; \\ \mathsf{tag}^{(s)} & \text{if } i = m^{(s)}+1 \end{cases}$$

Next define the following sets of random variables.

$$\mathcal{D}^{(s)} = \left\{A^{(s)}_0, \ldots, A^{(s)}_{m^{(s)}+1}\right\}; \quad \mathcal{R}^{(s)} = \left\{B^{(s)}_0, \ldots, B^{(s)}_{m^{(s)}+1}\right\};$$

$$\mathcal{D} = \{\mathsf{fStr}\} \cup \left(\bigcup_{s=1}^{q} \mathcal{D}^{(s)}\right); \quad \mathcal{R} = \{\delta_0\} \cup \left(\bigcup_{s=1}^{q} \mathcal{R}^{(s)}\right).$$

The number of elements in either of $\mathcal{D}$ or $\mathcal{R}$ equals $1+\sum_{s=1}^{q}(m^{(s)}+2) = 1+\sigma+q$. 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 δ_0 and $\gamma^{(s)}, C_1^{(s)}, \ldots, C_{m^{(s)}}^{(s)}, \mathsf{tag}^{(s)}$ for any s are chosen independently in the following manner:

1. δ_0 is chosen uniformly at random from $\{0,1\}^n$.
2. $\gamma^{(s)}, C_1^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, \mathsf{tag}^{(s)}$ are chosen uniformly and independently at random from $\{0,1\}^n$.
3. The quantity $\mathsf{pad}^{(s)}$ is chosen uniformly at random from $\{0,1\}^n$ and $C_{m^{(s)}}^{(s)}$ is set to $\mathsf{First}_{r^{(s)}}(P_{m^{(s)}} \oplus \mathsf{pad}^{(s)})$.

The quantities $C_1^{(s)}, \ldots, C_{m^{(s)}-1}^{(s)}, C_{m^{(s)}}^{(s)}, \mathsf{tag}^{(s)}$ are returned to the adversary. So, for any query, the adversary obtains independent and uniform random strings.

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, $\gamma^{(s)} = \pi(N^{(s)} \oplus \delta_1)$. 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 $\beta \in \mathbb{F}$, $\Pr[\Gamma_{\gamma^{(s)},i} \oplus \Gamma_{\gamma^{(s)},j} = \beta] = 1/2^n$.

Consider the event $\mathsf{Coll}(\mathcal{D})$. We argue that the probability that two elements of $\mathcal{D}$ are equal is at most $1/2^n$. There are several cases to consider:

Case 1. fStr equal to some other element of $\mathcal{D}$: The string fStr is a fixed n-bit string, whereas every other element of $\mathcal{D}$ is uniformly distributed over $\{0,1\}^n$. Hence, the probability of fStr being equal to any other element in $\mathcal{D}$ is $1/2^n$.

In view of Case 1, we need to only consider possible collisions between pairs of elements in $\mathcal{D}_1 = \mathcal{D} \setminus \{\mathsf{fStr}\}$.

Case 2. $N^{(s)} \oplus \delta_1$ is equal to some other element of $\mathcal{D}_1$: Let X be an element in $\mathcal{D}$ other than $N^{(s)} \oplus \delta_1$. If $X = N^{(t)} \oplus \delta_1$ for $t \neq s$, then $N^{(s)} \oplus \delta_1$ is necessarily distinct from X since $N^{(s)} \neq N^{(t)}$ (as the adversary is nonce-respecting). If X is an element in $\mathcal{D}_1$ other than an element of the form $N^{(t)} \oplus \delta_1$, then it is of the form $Y \oplus \Gamma_{\gamma^{(t)},j}$. Since $N^{(s)} \oplus \delta_1$ is independent of $\Gamma_{\gamma^{(t)},j}$, the probability that $N^{(s)} \oplus \delta_1$ equals X is $1/2^n$.

In view of Cases 1 and 2, we need to only consider possible collisions between pairs of elements in $\mathcal{D}_2 = \mathcal{D}_1 \setminus \cup_{s=1}^{q}\{N^{(s)} \oplus \delta_1\}$.

Case 3. Two elements X_1 and X_2 in $\mathcal{D}_2$ correspond to different queries: Then X_1 depends on $\gamma^{(s)}$ and X_2 depends on $\gamma^{(t)}$. So the probability that $\gamma^{(s)}$ and $\gamma^{(t)}$ are equal is $1/2^n$.

Case 4. Two elements X_1 and X_2 in $\mathcal{D}_2$ correspond to the same query: Then X_1 can be written as $Y_1 \oplus \Gamma_{\gamma^{(s)},i}$ and X_2 can be written as $Y_2 \oplus \Gamma_{\gamma^{(s)},j}$ where Y_1 and Y_2 are independent of $\gamma^{(s)}$ and $i \neq j$. In this case, the event that X_1 equals X_2 is the event that $\Gamma_{\gamma^{(s)},i} \oplus \Gamma_{\gamma^{(s)},j} = Z$ where Z is independent

of $\gamma^{(s)}$. From the properties of the masking function given in Definition 1 the probability of the last event is $1/2^n$.

Since the number of elements in $\mathcal{D}$ is $1+\sigma+q$, the probability of $\mathsf{Coll}(\mathcal{D})$ is at most $(1+\sigma+q)(\sigma+q)/(2\times 2^n)$.

All elements in $\mathcal{R}$ are chosen independently and uniformly at random from $\{0,1\}^n$ and so the probability that two of them are equal is $1/2^n$ so that $\Pr[\mathsf{Coll}] \leq (\sigma+q+1)^2/(2\times 2^n)$. Putting together the bounds for $\mathsf{Coll}(\mathcal{D})$ and $\mathsf{Coll}(\mathcal{R})$ we obtain the bound for Coll to be $(1+\sigma+q)(\sigma+q)/2^n$. This shows the privacy statement of PAE1 in Theorem 3.

C.2 Authenticity of PAE1

The authenticity analysis builds on the privacy analysis. In addition to encryption queries, the adversary $\mathcal{A}$ makes an implicit decryption query through its forgery attempt. Let there be a total of $q-1$ encryption queries and the one decryption query for a total of q queries. Let the forgery attempt be $(N^{(q)}, C^{(q)}, \mathsf{tag}^{(q)})$ which implicitly defines the decryption query. Denote by $\mathsf{sum}^{(q)}$ the value of sum corresponding to this decryption query. The adversary wins if $\pi(\mathsf{sum}^{(q)}) = \mathsf{tag}^{(q)}$. Here we are assuming that the length of the output tag is n bits. Later we mention the more general case when tag is truncated to $t \leq n$ bits.

The quantity δ_0 is chosen as in the analysis of the privacy. Further, for the s-th encryption query, the quantities $\gamma^{(s)}, C_1^{(s)}, \ldots, C_{m^{(s)}}^{(s)}, \mathsf{tag}^{(s)}$ are also chosen as in the privacy analysis. The sets $\mathcal{D}$ and $\mathcal{R}$ and the quantities in them corresponding to the encryption queries are defined as in the privacy analysis.

For the decryption query $(N^{(q)}, C^{(q)}, \mathsf{tag}^{(q)})$, the various ciphertext blocks are adversarially chosen and so cannot be assumed to follow the uniform distribution. Suppose that the output of $\mathsf{Format}(C^{(q)}, n)$ is $(C_1^{(q)}, \ldots, C_{m^{(q)}}^{(q)})$. In this case, the following choices are used.

1. Case $N^{(q)}$ is not equal to $N^{(s)}$ for any $1 \leq s < 1$: In this case, $\gamma^{(q)}, P_1^{(q)}, \ldots, P_{m^{(q)}-1}^{(q)}, \mathsf{pad}^{(q)}$ are chosen independently and uniformly at random and $P_{m^{(q)}}^{(q)}$ is set to be equal to $\mathsf{First}_{r^{(q)}}(C_{m^{(r)}} \oplus \mathsf{pad}^{(q)})$. The quantities

$$N^{(q)} \oplus \delta_1, P_1^{(q)} \oplus \mathit{\Gamma}_{\gamma^{(q)},1}, \ldots, P_{m^{(q)}-1}^{(q)} \oplus \mathit{\Gamma}_{\gamma^{(q)},m^{(q)}-1}, \mathsf{bin}_n(r^{(q)}) \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(s)}}$$

   are added to $\mathcal{D}$ and the quantities

$$\gamma^{(q)}, C_1^{(q)} \oplus \mathit{\Gamma}_{\gamma^{(q)},1}, \ldots, C_{m^{(q)}-1}^{(q)} \oplus \mathit{\Gamma}_{\gamma^{(q)},m^{(q)}-1}, \mathsf{pad}^{(q)}$$

   are added to $\mathcal{R}$.
2. Case $N^{(q)} = N^{(s)}$ for some $1 \leq s < q$: Then $\gamma^{(q)}$ is set to be equal to $\gamma^{(s)}$.
   - For each $1 \leq i \leq m^{(q)} - 1$; if $C_i^{(q)} = C_i^{(s)}$, then $P_i^{(q)}$ is set to be equal to $P_i^{(s)}$, otherwise $P_i^{(q)}$ is chosen independently and uniformly at random and $P_i^{(q)} \oplus \mathit{\Gamma}_{\gamma^{(q)},i}$ is added to $\mathcal{D}$ and $C_i^{(q)} \oplus \mathit{\Gamma}_{\gamma^{(q)},i}$ is added to $\mathcal{R}$.
   - If $(m^{(q)} = m^{(s)}$ and $r^{(q)} = r^{(s)})$, then $\mathsf{pad}^{(q)}$ is set to be equal to $\mathsf{pad}^{(s)}$; if $(m^{(s)} > m^{(q)}$ and $\mathsf{bin}_n(r^{(q)}) = P_{m^{(q)}}^{(s)})$ then $\mathsf{pad}^{(q)}$ is set to be equal to $C_{m^{(q)}}^{(s)} \oplus \mathit{\Gamma}_{\gamma^{(s)},m^{(q)}}$ (which is already in $\mathcal{R}$); otherwise $\mathsf{pad}^{(q)}$ is chosen independently and uniformly at random, $\mathsf{pad}^{(q)}$ is added to $\mathcal{R}$ and $\mathsf{bin}_n(r^{(q)}) \oplus \mathit{\Gamma}_{\gamma^{(q)},m^{(q)}}$ is added to $\mathcal{D}$.
   - Finally $P_{m^{(q)}}^{(q)}$ is set to be equal to $\mathsf{First}_{r^{(q)}}(C_{m^{(r)}} \oplus \mathsf{pad}^{(q)})$.

The goal of the authenticity analysis is to show that with high probability $\mathsf{sum}^{(q)}$ is distinct from all other elements in $\mathcal{D}$. Then the probability that $\pi(\mathsf{sum}^{(q)})$ equals $\mathsf{tag}^{(q)}$ (i.e., the adversary is successful) is $1/(2^n - \sigma - q - 1)$ if $\mathsf{tag}^{(q)}$ does not occur in $\mathcal{R}$; and the probability is 0 if $\mathsf{tag}^{(q)}$ indeed occurs in $\mathcal{R}$. So, under the condition that $\mathsf{sum}^{(q)}$ is distinct from all other elements in $\mathcal{D}$, the probability that the adversary is successful is at most $1/(2^n - \sigma - q - 1)$.

The condition that $\mathsf{sum}^{(q)}$ is distinct from all other elements in $\mathcal{D}$ requires arguing over the randomness of δ_0 and $\gamma^{(s)}, C_1^{(s)}, \ldots, C_{m^{(s)}}^{(s)}, \mathsf{tag}^{(s)}$ which in turn requires arguing over the distinctness of the elements in $\mathcal{D}$ which implies the (distinct) randomness of the elements in $\mathcal{R}$ (since π is a permutation).

Let Coll be the event that there is a collision among the elements of $\mathcal{D}$ or a collision among the elements of $\mathcal{R}$. It is standard to argue that

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}) &= \Pr[\pi(\mathsf{sum}^{(q)}) = \mathsf{tag}^{(q)}] \\ &\leq \Pr[\pi(\mathsf{sum}^{(q)}) = \mathsf{tag}^{(q)} | \overline{\mathsf{Coll}}] + \Pr[\mathsf{Coll}] \\ &\leq \frac{1}{2^n - \sigma - q - 1} + \Pr[\mathsf{Coll}].\end{aligned}$$

If PAE is modified to truncate the n-bit tag to $t \leq n$ bits, then the above bound gets modified to the following:

$$\mathsf{Adv}(\mathcal{A}) = \frac{2^{n-t}}{2^n - \sigma - q - 1} + \Pr[\mathsf{Coll}].$$

So, the analysis boils down to that of upper bounding the probability of Coll. The analysis of collision probability among the elements corresponding only to encryption queries is the same as that for case of privacy. So, we need to only consider collision between two elements where one element is defined from an encryption query and the other is defined from a decryption query. Suppose we are considering the collision of X and X' in $\mathcal{D}$ where X is defined from an encryption query and X' is defined from a decryption query. If $N^{(q)}$ is not equal to any previous $N^{(s)}$, then $\gamma^{(q)}$ is independent of all other quantities and this is sufficient to show that the probability of $X = X'$ is at most $1/2^n$.

Consider the element $\mathsf{sum}^{(q)}$. Due to the occurrence of a $\gamma^{(q)}$-term in $\mathsf{sum}^{(q)}$, the probability of $\mathsf{sum}^{(q)}$ being equal to $N^{(s)} \oplus \delta_1$ is $1/2^n$; due to the occurrence of δ_1 in $\mathsf{sum}^{(q)}$, the probability of $\mathsf{sum}^{(q)}$ being equal to any element in $\mathcal{D}$ other than those of the type $\mathsf{sum}^{(s)}$ or $N^{(s)} \oplus \delta_1$ is at most $1/2^n$. The main crux of the argument is to show that when $N^{(q)}$ is equal to $N^{(s)}$ for some previous s, then $\mathsf{sum}^{(q)}$ is distinct from $\mathsf{sum}^{(s)}$. The forms of $\mathsf{sum}^{(s)}$ and $\mathsf{sum}^{(q)}$ are as follows:

$$\begin{aligned}\mathsf{sum}^{(s)} &= P_1^{(s)} \oplus \cdots \oplus P_{m^{(s)}-1}^{(s)} \oplus \left(\mathsf{First}_{r^{(s)}}\left(C_{m^{(s)}}\right) || 0^{n-r^{(s)}}\right) \oplus \mathit{\Gamma}_{\gamma^{(s)}, m^{(s)}+1} \oplus \delta_1 \oplus \mathsf{pad}^{(s)}, \\ \mathsf{sum}^{(q)} &= P_1^{(q)} \oplus \cdots \oplus P_{m^{(q)}-1}^{(q)} \oplus \left(\mathsf{First}_{r^{(q)}}\left(C_{m^{(q)}}\right) || 0^{n-r^{(q)}}\right) \oplus \mathit{\Gamma}_{\gamma^{(q)}, m^{(q)}+1} \oplus \delta_1 \oplus \mathsf{pad}^{(q)}.\end{aligned}$$

Since $N^{(q)} = N^{(s)}$, it follows that $\gamma^{(q)} = \gamma^{(s)}$. There are several cases to consider:

Case 1. $m^{(q)} \neq m^{(s)}$**:** In this case, $\mathsf{sum}^{(s)} \oplus \mathsf{sum}^{(q)} = \mathit{\Gamma}_{\gamma^{(q)}, m^{(s)}+1} \oplus \mathit{\Gamma}_{\gamma^{(q)}, m^{(q)}+1} \oplus \mathsf{str}$ where str is independent of $\gamma^{(q)}$. Using the properties of ψ from Definition 1 it follows that $\Pr[\mathsf{sum}^{(s)} = \mathsf{sum}^{(q)}] \leq 1/2^n$.

Case 2. $m^{(q)} = m^{(s)}$**:** In this case, the term $\mathit{\Gamma}_{\gamma^{(s)}, m^{(s)}+1}$ cancels out with the term $\mathit{\Gamma}_{\gamma^{(q)}, m^{(q)}+1}$ in $\mathsf{sum}^{(s)} \oplus \mathsf{sum}^{(q)}$. There are two subcases to consider.

Subcase 2a. $r^{(s)} \neq r^{(q)}$**:** The quantity $\mathsf{pad}^{(s)}$ is a uniformly distributed n-bit string. If ($m^{(s)} > m^{(q)}$ and $P^{(s)}_{m^{(q)}} = \mathsf{bin}_n(r^{(q)})$), then $\mathsf{pad}^{(q)} = C^{(s)}_{m^{(q)}} \oplus \Gamma_{\gamma^{(q)},m^{(q)}}$ where $C^{(s)}_{m^{(q)}}$ is independent and uniformly distributed; otherwise $\mathsf{pad}^{(q)}$ is itself independent and uniformly distributed. In both cases, $\mathsf{pad}^{(q)}$ is independent of $\mathsf{pad}^{(s)}$. We can write $\mathsf{sum}^{(s)} \oplus \mathsf{sum}^{(q)} = \mathsf{pad}^{(s)} \oplus \mathsf{pad}^{(q)} \oplus \mathsf{str}$ where str is independent of both $\mathsf{pad}^{(s)}$ and $\mathsf{pad}^{(q)}$. This shows that the probability of the event $\mathsf{sum}^{(s)} = \mathsf{sum}^{(q)}$ is $1/2^n$.

Subcase 2b. $r^{(s)} = r^{(q)}$**:** In this case, $\mathsf{pad}^{(q)}$ is equal to $\mathsf{pad}^{(s)}$ and so neither γ nor pad has any effect. So, we are left with considering the probability that

$$P^{(s)}_1 \oplus \cdots \oplus P^{(s)}_{m^{(q)}-1} \oplus \left(\mathsf{First}_{r^{(q)}}\left(C^{(s)}_{m^{(q)}}\right)||0^{n-r^{(q)}}\right) = P^{(q)}_1 \oplus \cdots \oplus P^{(q)}_{m^{(q)}-1} \oplus \left(\mathsf{First}_{r^{(q)}}\left(C^{(q)}_{m^{(q)}}\right)||0^{n-r^{(q)}}\right).$$

If there is some $1 \leq i \leq m^{(q)} - 1$ such that $C^{(q)}_i$ is not equal to $C^{(s)}_i$, then $P^{(q)}_i$ is chosen uniformly and independent of all other quantities. This is sufficient to show that the desired probability is at most $1/2^n$. On the other hand, if $C^{(q)}_i = C^{(s)}_i$ for all $1 \leq i \leq m^{(q)} - 1$, then it necessarily follows that $C^{(q)}_{m^{(q)}} \neq C^{(s)}_{m^{(s)}}$ and in this case $\mathsf{sum}^{(q)}$ is distinct from $\mathsf{sum}^{(s)}$. So, in all cases, it follows that the probability of $\mathsf{sum}^{(q)}$ being equal to $\mathsf{sum}^{(s)}$ is at most $1/2^n$.

A simpler analysis shows that the probability of two elements in $\mathcal{R}$ being equal is also $1/2^n$. So, as in the privacy analysis we have $\Pr[\mathsf{Coll}] \leq (\sigma + q + 1)^2/2^n$.

C.3 Security Argument for PAEAD1

The analysis of the AEAD scheme follows along the same lines as that of the AE scheme. In this case, we also have to account for the header part. As for the AE scheme, we first consider the privacy of PAEAD1 and then take up the analysis of its authenticity.

The main intuition behind the security is the following. There is a call to PAuth and inside the PAuth algorithm the masks are generated from $\pi(\delta_0)$. Whereas the masks in the calls to Forward are generated from $\pi(N \oplus \delta_1)$. With high probability, δ_0 is distinct from $N \oplus \delta_1$ and so the masks used in PAuth are (almost) independent of the masks used in the calls to Forward. This separation of the masks in effect results in a separation of the input spaces of π in PAuth and Forward. So, the calls to π in PAuth and the calls to π in Forward are made on disjoint input sets. As a result, the two algorithms PAuth and Forward behave like independent random functions. It is from this input space separation that the security of PAEAD follows.

C.4 Privacy of PAEAD1

Queries by the adversary are of the form $(N^{(s)}, H^{(s)}, P^{(s)})$ for $s = 1, \ldots, q$, where $N^{(s)}$'s are the nonces, $H^{(s)}$'s are the headers and $P^{(s)}$'s are the messages. The output of the s-th query is $(C^{(s)}, \mathsf{tag}^{(s)})$ where $C^{(s)}$ is of the same length as $P^{(s)}$.

We need to consider the sets $\mathcal{D}$ and $\mathcal{R}$ as in the AE schemes. Due to the presence of the header, in the case of AEAD schemes, we define two sets $\mathcal{D}_H$ and $\mathcal{D}_P$ consisting of elements in the domain of π arising in the header and the message part respectively. Define $\mathcal{D} = \mathcal{D}_H \cup \mathcal{D}_P$. Similarly, we define the sets $\mathcal{R}_H$ and $\mathcal{R}_P$ and $\mathcal{R}$. The sets $\mathcal{D}_P$ and $\mathcal{R}_P$ are defined as in Section C.1.

The total number of n-bit blocks in all the messages will be denoted by σ_P and so the number of elements in $\mathcal{D}_P$ (and also $\mathcal{R}_P$) is $\sigma_P + q + 1$. The total number of n-bit blocks in all the headers will be denoted by σ_H. The total number of n-bit blocks provided by the adversary in all the headers and the messages will be denoted by σ so that $\sigma = \sigma_P + \sigma_H$.

Recall that for PAEAD, the algorithm PAuth is called with π and so in this case, PAuth only makes calls to π. The first call is on δ_0 and does not depend on the actual header. The following notation

will be used to denote the internal quantities computed by PAuth to distinguish these from the similar named quantities computed by the AE algorithm.

- The header blocks (if present) of the s-th query will be denoted as $H_1^{(s)}, \ldots, H_{h^{(s)}}^{(s)}$ where $h^{(s)}$ is the number of header blocks in the s-th query. Note that $H_1^{(s)}, \ldots, H_{h^{(s)}}^{(s)}$ is the output of $\mathsf{Format}(H^{(s)}, n)$ and so if the length of $H^{(s)}$ is not a multiple of n, then $H_{h^{(s)}}^{(s)}$ is an n-bit string obtained by 10^* padding.
- We will denote the outputs of π on the first $h^{(s)} - 1$ header blocks by $J_1^{(s)}, \ldots, J_{h^{(s)}-1}^{(s)}$.
- The sum in the PAuth computed in the s-th query will be denoted by $\mathsf{hsum}^{(s)}$.

Note that $\mathsf{tag}^{(s)} = \mathsf{tag}_1^{(s)} \oplus \mathsf{tag}_2^{(s)}$ where $\mathsf{tag}_1^{(s)}$ is the tag generated by the $\mathsf{PAE1}$ algorithm and $\mathsf{tag}_2^{(s)}$ is the tag generated by PAuth.

The contents of the sets $\mathcal{D}_H$ and $\mathcal{R}_H$ are defined as follows. First δ_0 is put into $\mathcal{D}_H$ and κ is chosen independently and uniformly at random and put into $\mathcal{R}_H$. If $H^{(s)}$ is distinct from $H^{(s')}$ for $1 \le s' < s$, then the following are performed for the s-th query. Note that as the set $\mathcal{D}_H$ grows, so does the set $\mathcal{D}$ and similarly for $\mathcal{R}_H$ and $\mathcal{R}$.

1. Case $h^{(s)} > 1$:
   - For $1 \le i \le h^{(s)} - 1$, if $H_i^{(s)}$ is distinct from $H_i^{(s')}$ for all $s' < s$, then add $H_i^{(s)} \oplus \Gamma_{\kappa,i}$ to $\mathcal{D}_H$; choose $J_i^{(s)}$ independently and uniformly at random from the set of all n-bit strings. Add $J_i^{(s)}$ to $\mathcal{R}_H$.
   - If $r^{(s)} = n$, define $\mathsf{hsum}^{(s)} = J_1^{(s)} \oplus \cdots \oplus J_{h^{(s)}-1}^{(s)} \oplus H_{h^{(s)}}^{(s)}$; otherwise, define $\mathsf{hsum}^{(s)} = J_1^{(s)} \oplus \cdots \oplus J_{h^{(s)}-1}^{(s)} \oplus H_{h^{(s)}}^{(s)} \oplus \Gamma_{\kappa,h^{(s)}}$.
2. Case $h^{(s)} = 1$:
   - If $r^{(s)} < n$, define $\mathsf{hsum}^{(s)} = H_1^{(s)} \oplus \Gamma_{\kappa,-1}$.
   - If $r^{(s)} = n$, define $\mathsf{hsum}^{(s)} = H_1^{(s)} \oplus \Gamma_{\kappa,-2}$.
3. In both cases (i.e., irrespective of whether $h^{(s)}$ equals 1 or not), $\mathsf{hsum}^{(s)}$ is added to $\mathcal{D}_H$; $\mathsf{tag}_2^{(s)}$ is chosen independently and uniformly at random from the set of all n-bit strings and it is added to $\mathcal{R}_H$.

Note that the number of elements in $\mathcal{D}_H$ (and also $\mathcal{R}_H$) is at most $\sigma_H + 1$. Then the number of elements in $\mathcal{D}$ (and also $\mathcal{R}$) is at most $\sigma_P + q + 1 + \sigma_H + 1 = \sigma_P + \sigma_H + q + 2 = \sigma + q + 2$.

As a response to the s-th query, the adversary is returned $(C^{(s)}, \mathsf{tag}_1^{(s)} \oplus \mathsf{tag}_2^{(s)})$ where $C^{(s)}$ is chosen to be an independent and uniform random string of length equal to $P^{(s)}$ and $\mathsf{tag}_1^{(s)}$ is chosen to be an independent and uniform random n-bit string.

As in the case of the privacy analysis of $\mathsf{PAE1}$, let $\mathsf{Coll}(\mathcal{D})$ (resp. $\mathsf{Coll}(\mathcal{R})$) be the event that there is a collision in $\mathcal{D}$ (resp. $\mathcal{R}$) and let Coll be the event $\mathsf{Coll}(\mathcal{D}) \vee \mathsf{Coll}(\mathcal{R})$. As is standard, the privacy bound of any adversary reduces to showing an upper bound on the probability of $\Pr[\mathsf{Coll}]$.

Consider $\mathsf{Coll}(\mathcal{D})$. The collision analysis between any two elements in $\mathcal{D}_P$ remains unchanged from the privacy of PAE. Further, using the properties of ψ from Definition 1, it can be shown that the probability that any two elements in $\mathcal{D}_H$ are equal with probability at most $1/2^{n-1}$. So, it only remains to consider collisions between an element of $\mathcal{D}_H$ and an element of $\mathcal{D}_P$. The element δ_0 in $\mathcal{D}_H$ is an independent and uniform random n-bit string and is equal to any element of $\mathcal{D}_P$ with probability at most $1/2^n$. Similarly, the element κ is also an independent and uniform random n-bit string. Any element in $\mathcal{D}_H$ which depends on κ is equal to an element of $\mathcal{D}_P$ with probability at most $1/2^n$. Apart from δ_0, the only element in $\mathcal{D}_H$ which does not depend on κ is $\mathsf{hsum}^{(s)}$ if $h^{(s)} > 1$ and $r^{(s)} = n$. In this

case $\mathsf{hsum}^{(s)} = J_1^{(s)} \oplus \cdots \oplus J_{h^{(s)}-1}^{(s)} \oplus H_{h^{(s)}}^{(s)}$ and since the J's are chosen independently and uniformly at random the probability that such an $\mathsf{hsum}^{(s)}$ is equal to an element of $\mathcal{D}_P$ is also at most $1/2^n$.

The above shows that the probability of $\mathsf{Coll}(\mathcal{D})$ is at most $(\sigma+q+2)(\sigma+q+1)/(2\times 2^{n-1}) \leq (\sigma+q+2)^2/2^n$. Similarly, the probability of $\mathsf{Coll}(\mathcal{R})$ can also be shown to be at most $(\sigma+q+2)^2/2^n$ so that $\Pr[\mathsf{Coll}] \leq (\sigma+q+2)^2/2^{n-1}$. This establishes the privacy bound for PAEAD.

C.5 Authenticity of PAEAD1

First consider that n-bit tags are produced. The adversary $\mathcal{A}$ makes $q-1$ encryption queries $(N^{(1)}, H^{(1)}, P^{(1)}), \ldots, (N^{(q-1)}, H^{(q-1)}, P^{(q-1)})$ and the forgery attempt (N, H, C, tag) which as before is an implicit decryption query. We need to define the sets $\mathcal{D}$ and $\mathcal{H}$. As earlier $\mathcal{D} = \mathcal{D}_H \cup \mathcal{D}_P$ and $\mathcal{R} = \mathcal{R}_H \cup \mathcal{R}_P$. The sets $\mathcal{D}_P$ and $\mathcal{R}_P$ are defined from the encryption queries and the decryption query as in Section C.2 for the analysis of the authenticity of PAE1. The elements in the sets $\mathcal{D}_H$ and $\mathcal{R}_H$ corresponding to the encryption queries and the decryption query is obtained as in Section C.4, i.e., for the sets $\mathcal{D}_H$ and $\mathcal{R}_H$ there is no difference in the way encryption and decryption queries are handled.

Denote by hsum and sum the sums corresponding to the header and the plaintext respectively in the decryption query. We need to upper bound the probability $\Pr[\pi(\mathsf{hsum}) \oplus \pi(\mathsf{sum}) = \mathsf{tag}]$. Suppose that hsum and sum are distinct from all other elements in the range and also from each other. To ensure this we need to ensure that the event Coll does not occur. In particular, we can write.

$$\begin{aligned}\Pr[\pi(\mathsf{hsum}) \oplus \pi(\mathsf{sum}) = \mathsf{tag}] &= \Pr[\pi(\mathsf{hsum}) \oplus \pi(\mathsf{sum}) = \mathsf{tag}|\overline{\mathsf{Coll}}] + \Pr[\mathsf{Coll}] \\ &\leq \frac{1}{2^n - \sigma - q - 3} + \Pr[\mathsf{Coll}].\end{aligned}$$

The number of elements in $\mathcal{D}$ is at most $\sigma+q+2$. Conditioned on the event $\overline{\mathsf{Coll}}$, the quantities are distinct and are distributed uniformly over a set containing $2^n - (\sigma+q+2)$ elements. So, if tag is 0^n, then the probability $\Pr[\pi(\mathsf{hsum}) \oplus \pi(\mathsf{sum}) = \mathsf{tag}]$ is 0 and otherwise it is at most $1/(2^n - \sigma - q - 3)$. If tag is truncated to t bits, then

$$\Pr[\pi(\mathsf{hsum}) \oplus \pi(\mathsf{sum}) = \mathsf{tag}] \leq \frac{2^{n-t}}{2^n - \sigma - q - 3} + \Pr[\mathsf{Coll}].$$

So, again the analysis boils down to upper bounding Coll. For the encryption queries, this analysis is the same as the analysis for the privacy of PAEAD. For the decryption query, the entries to $\mathcal{D}_H$ and $\mathcal{R}_H$ remain unchanged from that for the encryption queries and so the collision analysis for these elements also remains unchanged. Further, the collision analysis of the elements in $\mathcal{D}_P$ and $\mathcal{R}_P$ for encryption and decryption queries is the same as that for the authenticity of PAE. So, it only remains to consider possible collisions between the elements in $\mathcal{D}_P$ (resp. $\mathcal{R}_P$) corresponding to the decryption query and the elements of $\mathcal{D}_H$ (resp. $\mathcal{R}_H$). This analysis is again made easy by the fact that the masks for the header portion are generated from κ which with high probability is distinct from the γ corresponding to the decryption query. So, to summarise, the analysis of the collisions for the privacy and authenticity of PAE and the privacy of PAEAD lead to the collision analysis for the authenticity of PAEAD. This leads to the bound $\Pr[\mathsf{Coll}] \leq (\sigma+q+2)^2/2^{n-1}$.

C.6 Security Arguments for PAE2 and PAEAD2

The overall structure of the argument is the same as that for PAE1 and PAEAD2. The difference arises due to the difference in the definitions of Forward1 and Forward2 (and also Backward1 and Backward2).

This affects the definition of the elements in the sets $\mathcal{D}$ and $\mathcal{R}$ that is defined in the security analysis. In the case of PAE2, the definition of the elements A_i's and B_j's are given below.

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

$$A_i^{(s)} = \begin{cases} N^{(s)} \oplus \delta_1 & \text{if } i = 0; \\ \boxed{C_i^{(s)} \oplus \Gamma_{\gamma^{(s)},i}} & \text{for } 1 \leq i \leq m^{(s)} - 1; \\ \mathsf{bin}_n\left(r^{(s)}\right) \oplus \Gamma_{\gamma^{(s)},m^{(s)}} & \text{if } i = m^{(s)}; \\ \left.\begin{array}{l} \mathsf{sum}^{(s)} \\ = P_1^{(s)} \oplus \cdots \oplus P_{m^{(s)}-1}^{(s)} \oplus \left(\mathsf{First}_{r^{(s)}}\left(C_{m^{(s)}}^{(s)}\right) || 0^{n-r^{(s)}}\right) \\ \quad \oplus \Gamma_{\gamma^{(s)},m^{(s)}+1} \oplus \delta_1 \oplus \mathsf{pad}^{(s)} \end{array}\right\} & \text{if } i = m^{(s)} + 1. \end{cases}$$

$$B_i^{(s)} = \begin{cases} \gamma^{(s)} & \text{if } i = 0; \\ \boxed{P_i^{(s)} \oplus \Gamma_{\gamma^{(s)},i}} & \text{for } 1 \leq i \leq m^{(s)} - 1; \\ \mathsf{pad}^{(s)} & \text{if } i = m^{(s)}; \\ \mathsf{tag}^{(s)} & \text{if } i = m^{(s)} + 1 \end{cases}$$

The differences to the case of PAE1 is highlighted using boxes. Using this alternative definition, it is routine to carry out the rest of the analysis.

D 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 6 are the same as that of 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 6.