PRØST v1.1

Designers/Submitters

Elif Bilge Kavun[1]
Martin M. Lauridsen[2]
Gregor Leander[1]
Christian Rechberger[2]
Peter Schwabe[3]
Tolga Yalçın[4]

Affiliations

[1] Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany
[2] DTU Compute, Technical University of Denmark, Denmark
[3] Digital Security Group, Radboud University Nijmegen, The Netherlands
[4] University of Information Science and Technology, Ohrid, Republic of Macedonia

Contact

proest@compute.dtu.dk

January 14, 2015

1 Summary

The main contribution of this document is the presentation of PRØST, a new highly secure and efficient permutation with strong security bounds, suitable for a wide range of platforms and modes.

The CAESAR candidates described herein use the excellent existing modes of operation COPA [3], OTR [13] and APE [2], and instantiate them using the PRØST permutation, either directly (in the case of APE) or through the single-key Even-Mansour construction [8] (in the case of COPA and OTR). These candidates are called PRØST-COPA-n, PRØST-OTR-n and PRØST-APE-n.

2 Notation

Throughout this document, in the context of the PRØST permutation, we use n to denote the security level of the resulting block cipher when used in a single-key Even-Mansour construction (see Section 4.2). For PRØST, this happens when the state size is $2n$, so in our specification we use PRØST-n to denote the version of PRØST which has a state size of $2n$ bits, and which fulfills this condition.

We use $\mathbb{F}_2$ to denote the finite field of size 2, and we interchangeably switch between $\mathbb{F}_2^n$ and $\mathbb{F}_{2^n}$. For example, we can write the element $x^2 + x + 1 \in \mathbb{F}_{2^n}$ as $(0, \ldots, 0, 1, 1, 1) \in \mathbb{F}_2^n$ or as 7 in hexadecimal notation (typewriter font) for short. We use $\mathbb{Z}_n$ to denote both the additive group of order n and the set $\{0, \ldots, n-1\}$.

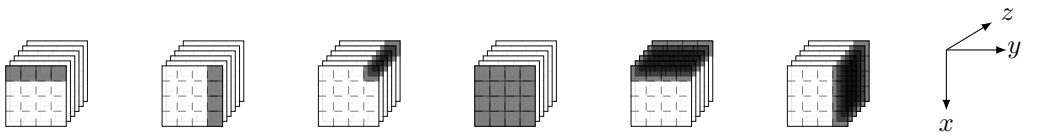
In the following, let X be a binary string. We use the convention that the rightmost bit in any representation of X is the least significant bit. The bit length of X is denoted $|X|$. We use ε to denote the empty string, so $|\varepsilon| = 0$. When writing 0^k or 1^k we mean k repetitions of that particular bit. We use $\mathrm{msb}_\ell(X)$ and $\mathrm{lsb}_\ell(X)$ to denote the ℓ most significant bits, respectively least significant bits, of X. We use $\|$ to denote concatenation of strings, $\oplus$ to denote XOR and $\odot$ to denote bitwise AND. Left and right *shifts* by k positions of X are denoted $X \ll k$ and $X \gg k$, respectively, while left and right *rotations* by k positions of X are denoted $X \lll k$ and $X \ggg k$.

An asterisk in superscript denotes the Kleene star, e.g. $\mathbb{F}_2^*$ are finite binary strings of non-negative length, including ε. A plus in superscript denotes "at least one", so e.g. $(\mathbb{F}_2^b)^+$ is the set of finite binary strings with length equal to a positive multiple of b.

3 The PRØST Permutation

This section specifies the PRØST permutation. Section 4 presents the use of PRØST in existing AEAD (authenticated encryption with associated data) modes, which comprise the CAESAR competition proposals in this document.

For the PRØST permutation, we consider a $2n$-bit state s as a three-dimensional block of height h, width w and depth d along the x, y and z axes respectively (see Figure 1g). We re-use the nomenclature of Keccak [5] when referring to the terms *row*, *column*, *lane* (which we also call *register*), *slice*, *plane* and *sheet* as indicated in Figure 1.



(a) Row (b) Column (c) Lane (d) Slice (e) Plane (f) Sheet (g) Axes

Figure 1: Nomenclature for state parts

With the orientation of the (x, y, z) axes, we use $s_{x,y}$ to refer to the register in row x and column y of a state s, and we use $s_{x,y,z}$ to denote the zth bit of $s_{x,y}$. For PRØST, we always have $h = w = 4$, so $2n = 16d$. As such, the embedding of the state s into the (x, y)-plane results in a 4×4 matrix

$$s = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix},$$

where each $s_{x,y}$ is a d-bit register. We specify PRØST for $d \in \{16, 32\}$, so effectively we specify two permutations (see also Section 3.6). The PRØST permutation consists of compositions of permutations which we refer to as rounds, borrowing from the design of iterated block cipher constructions. We denote the total number of rounds by T. We use $R_i : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ to refer to round i, $0 \leq i < T$, which is defined as

$$R_i(s) = (\texttt{AddConstants}_i \circ \texttt{ShiftPlanes}_i \circ \texttt{MixSlices} \circ \texttt{SubRows})\,(s),$$

where $s \in \mathbb{F}_2^{2n}$ (we describe mapping state representations $\mathbb{F}_2^{4\times4\times d} \leftrightarrow \mathbb{F}_2^{2n}$ in Section 3.1), so PRØST-$n(s) = (R_{T-1} \circ \cdots \circ R_0)(s)$. We give the definition of each of the round operations below, but first we describe how one maps a stream of bytes to a PRØST state.

3.1 Mapping a Byte Array to a PRØST State

The PRØST state is a 4×4 matrix of d-bit registers. Inputs to an authenticated cipher as specified by the CAESAR rules are byte arrays. This subsection describes how to map a byte array of length $2d$ to a PRØST state.

On a high level, we map words of length $\ell = d/8$ bytes to matrix coefficients. This is done row first, column second, i.e., the first ℓ bytes go into $s_{0,0}$, the next ℓ bytes go to $s_{0,1}$, and so on.

On a low level we map arrays of ℓ bytes to ℓ-byte words. For this mapping we use little-endian representation because it is natively supported by the large majority of modern computer architectures. For example, when we have $d = 32$, a register is $\ell = 4$ bytes long and the first four bytes of the byte array $\texttt{bs}$ are mapped to register $s_{0,0}$ as

$$s_{0,0} = \texttt{bs}[3] \parallel \texttt{bs}[2] \parallel \texttt{bs}[1] \parallel \texttt{bs}[0],$$

or, considering $s_{0,0}$ as an integer, as

$$s_{0,0} = \sum_{i=0}^{3} \texttt{bs}[i]2^{8i}.$$

When putting this together we obtain that register $s_{x,y}$ of the PRØST state s with $0 \leq x, y \leq 3$ is loaded from a byte array $\texttt{bs}$ as

$$s_{x,y} = \texttt{bs}[4\ell x + \ell y + \ell - 1] \parallel \texttt{bs}[4\ell x + \ell y + \ell - 2] \parallel \cdots \parallel \texttt{bs}[4\ell x + \ell y],$$

or, again considering $s_{x,y}$ as an integer, as

$$s_{x,y} = \sum_{i=0}^{\ell} \texttt{bs}[4\ell x + \ell y + i]2^{8i}.$$

The mapping of a state to a byte stream is done by the obvious inversion of this mapping.

3.1.1 Mapping a State to $\mathbb{F}_2^{2n}$

Let $F : \mathbb{F}_2^{4\times 4\times d} \to \mathbb{F}_2^{2n}$ be defined for a state s as

$$F(s) = (s_{0,0} \| s_{0,1} \| \cdots \| s_{3,2} \| s_{3,3}).$$

When a PRØST state is XORed with a $2n$-bit value X, we implicitly mean that X is XORed to the state mapped to $\mathbb{F}_2^{2n}$ by F, i.e. $X \oplus F(s)$. Note that for all $s_{x,y}$, we have that $s_{x,y,0}$ corresponds to the most significant (leftmost) bit, and $s_{x,y,d-1}$ corresponds to the least significant (rightmost) bit.

3.2 SubRows

PRØST uses a single 4-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ which is given in Table 1. The SubRows operation substitutes each row of the state according to this S-box. More precisely, a state s will be mapped to a state s' where

$$\forall x \in \{0,\ldots,3\}, \forall z \in \{0,\ldots,d-1\} \quad : \quad \left(s'_{x,0,z} \| s'_{x,1,z} \| s'_{x,2,z} \| s'_{x,3,z}\right) = S\left(s_{x,0,z} \| s_{x,1,z} \| s_{x,2,z} \| s_{x,3,z}\right).$$

Table 1: Action of the PRØST S-box in hexadecimal notation

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	0	4	8	f	1	5	e	9	2	7	a	c	b	d	6	3

3.3 MixSlices

For a state s, the MixSlices operation updates each of the d slices of s by multiplying the bits of each one, arranged in a 16-bit column vector, from the left by a 16×16 matrix M over $\mathbb{F}_2$, i.e. we obtain $s' = \texttt{MixSlices}(s)$ as

$$\forall z \in \{0,\ldots,d-1\} \quad : \quad \begin{pmatrix} s'_{0,0,z} \\ s'_{0,1,z} \\ s'_{0,2,z} \\ s'_{0,3,z} \\ s'_{1,0,z} \\ \vdots \\ s'_{3,2,z} \\ s'_{3,3,z} \end{pmatrix} = M \begin{pmatrix} s_{0,0,z} \\ s_{0,1,z} \\ s_{0,2,z} \\ s_{0,3,z} \\ s_{1,0,z} \\ \vdots \\ s_{3,2,z} \\ s_{3,3,z} \end{pmatrix}.$$

The matrix M used for MixSlices is given by (1). It is involutive, i.e. self-inverse, with Hamming weight 88. The matrix is MDS, so the branch number is 5. This means that from a differential and linear cryptanalysis perspective, for each slice, if $k > 0$ is the number of active bits in a column,

then the corresponding column of s' has at least $5-k$ active bits.

$$M = \begin{pmatrix} \texttt{1000100100101011} \\ \texttt{0100100000011001} \\ \texttt{0010010011001000} \\ \texttt{0001001001100100} \\ \texttt{1001100010110010} \\ \texttt{1000010010010001} \\ \texttt{0100001010001100} \\ \texttt{0010000101000110} \\ \texttt{0010101110001001} \\ \texttt{0001100101001000} \\ \texttt{1100100000100100} \\ \texttt{0110010000010010} \\ \texttt{1011001010011000} \\ \texttt{1001000110000100} \\ \texttt{1000110001000010} \\ \texttt{0100011000100001} \end{pmatrix} \tag{1}$$

3.4 $\texttt{ShiftPlanes}_i$

The $\texttt{ShiftPlanes}_i$ operation rotates the registers of each plane of the state towards right (positive z direction) by a particular amount. In contrast to many existing designs, PRØST uses two different shift operations, depending on the parity of the round number i; one for even numbered rounds and one for odd numbered rounds.

For round i with $0 \leq i < T$, the rotation amount for plane j with $0 \leq j \leq 3$ is given by the jth element of row $i \bmod 2$, denoted $\pi_{i \bmod 2,j}$, of a 2×4 matrix π over $\mathbb{Z}_d$. We use π_0 and π_1 to denote the first, respectively second row of π. Thus, $\texttt{ShiftPlanes}_i$ applied to a state s in round i will result in a state s' where

$$\forall x \in \{0,\ldots,3\}, \forall y \in \{0,\ldots,3\} \quad : \quad s'_{x,y} = \begin{cases} s_{x,y} \ggg \pi_{0,x} & , i \text{ even} \\ s_{x,y} \ggg \pi_{1,x} & , i \text{ odd} \end{cases}.$$

Table 2: Shift vectors π_1 and π_2 for the $\texttt{ShiftPlanes}_i$ operation for $d = 16$ and $d = 32$

d	π_0	π_1
16	$(0, 2, 4, 6)$	$(0, 1, 8, 9)$
32	$(0, 4, 12, 26)$	$(1, 24, 26, 31)$

3.5 $\texttt{AddConstants}_i$

In the $\texttt{AddConstants}_i$ operation, different round/lane-dependent constants are XORed to each lane. Let $c_0 = \text{msb}_d(\texttt{0x75817b9d})$ and $c_1 = \text{msb}_d(\texttt{0xb2c5fef0})$. In the following, let $j = 4x + y$, with $x, y \in \{0,\ldots,3\}$, so j is the lane index with $0 \leq j \leq 15$. In round i, $0 \leq i < T$, we have $s' = \texttt{AddConstants}_i(s)$ with

$$\forall x \in \{0,\ldots,3\}, \forall y \in \{0,\ldots,3\} \quad : \quad s'_{x,y} = \begin{cases} s_{x,y} \oplus (c_0 \lll (i+j)) & , j \text{ even} \\ s_{x,y} \oplus (c_1 \lll (i+j)) & , j \text{ odd} \end{cases}.$$

3.6 PRØST-128 and PRØST-256 permutations

We specify two permutations to be used: PRØST-256 and PRØST-128. These are summarized in Table 3.

Table 3: PRØST permutations and their parameters.

Name	State size $2n$	Register length d	Rounds T
PRØST-128	256 bits	16 bits	16
PRØST-256	512 bits	32 bits	18

3.7 Absence of Trap-doors

We faithfully declare that we have not inserted any hidden weaknesses in the PRØST permutation.

4 Authenticated Encryption with PRØST

Authenticated encryption (AE) with associated data (AD) is a symmetric-key cryptographic primitive which combines encryption and authentication in a single algorithm, with focus on both aspects from the design phase. This is contrary to the earlier paradigm of obtaining secrecy and authentication by applying some combination of encryption scheme with a MAC.

AE schemes are most commonly based on block ciphers as the underlying primitive. Examples of such include OCB[1-3] [16, 15, 11], GCM [12], CCM [20], AEGIS [21], COPA [3], COBRA [4], FIDES [7], OTR [13], the McOE family [9] and POET [1].

With the advent of permutation-based designs, notably in the Sponge constructions, the use of permutations as the underlying primitive in AE schemes is becoming increasingly frequent. So far, permutation-based AE schemes include SpongeWrap [6], APE [2] and PPAE [10].

In this section we describe our proposals for the CAESAR competition, which are all specific instantiations of AEAD schemes using the PRØST permutation as the underlying primitive. Each of the instantiations are based on existing AE schemes. The proposals are summarized in Table 4, which lists the specific instantiations, while the specifications in the following sections use general n for their descriptions.

Table 4: Our proposals, underlying PRØST permutation and their rankings. Other columns indicate onlineness (in encryption and decryption), parallelizability (P), nonce misuse-resistance (NMR), easy constant-time implementation (CT) and cheap power analysis countermeasures (CM).

			Online					
Rank	Proposal	Permutation	Enc	Dec	P	NMR	CT	CM
1	PRØST-COPA-128	PRØST-128	✓	✓	✓	✓	✓	✓
2	PRØST-COPA-256	PRØST-256	✓	✓	✓	✓	✓	✓
3	PRØST-OTR-128	PRØST-128	✓	✓	✓		✓	✓
4	PRØST-OTR-256	PRØST-256	✓	✓	✓		✓	✓
5	PRØST-APE-256[256, 256]	PRØST-256	✓			✓	✓	✓
6	PRØST-APE-128[128, 128]	PRØST-128	✓			✓	✓	✓

4.1 Notation

Throughout our description of the proposals, we use P_n as a shorthand for the PRØST-n permutation, i.e. PRØST with n-bit security level and internal state size of $2n$ bits.

For each of our proposals, we define the following parameters: $\mathcal{M}$ denotes the space of allowed messages; $\mathcal{C}$ denotes the space of allowed ciphertexts; $\mathcal{A}$ denotes the space of allowed associated data; $\mathcal{N}$ denotes the space of allowed nonces (public message numbers). None of our proposals use secret message numbers. We use τ to denote the tag length. We define a *block* as a string of b bits, hence b is the *block size* of a proposal. For a binary string X, we denote by $X\|10^*$ the string X which is padded with a 1 followed by zeroes, such that $|X\|10^*|$ becomes the smallest multiple of b *larger than* $|X|$. This means that if $|X| = b$, then we have $|X\|10^*| = |X| + b$, so the string is expanded by another full block. We use the $\perp$ symbol to denote the output of the decryption algorithm when tag verification fails.

In our proposals, we *always* pad the message $M \in \mathcal{M}$ using the 10^* padding (see also Section 6.5). Thus, the first thing to do when encrypting is setting $M = M\|10^*$. This also implies that our proposals do ciphertext expansions, in the sense that the length of ciphertext will never equal the length of the message, but will always have length equal to a positive multiple of b. The largest expansion possible is b bits, which occurs when $|M| \equiv 0 \mod b$. Note that this padding is not, in general, applied to the associated data.

We use $a \stackrel{def}{=} \lceil |A|/b \rceil$ to denote the number of blocks of associated data and $w \stackrel{def}{=} |M\|10^*|/b$ to denote the number of message blocks (after doing the 10^* padding) throughout. Table 5 gives the mentioned parameters for our proposals.

Table 5: Proposal parameters

Rank	Proposal	Block size b	$\mathcal{M}$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{N}$	Tag length τ
1	PRØST-COPA-128	256	$\mathbb{F}_2^*$	$(\mathbb{F}_2^{256})^+$	$\mathbb{F}_2^*$	$\mathbb{F}_2^{256}$	256
2	PRØST-COPA-256	512	$\mathbb{F}_2^*$	$(\mathbb{F}_2^{512})^+$	$\mathbb{F}_2^*$	$\mathbb{F}_2^{512}$	256
3	PRØST-OTR-128	256	$\mathbb{F}_2^*$	$(\mathbb{F}_2^{256})^+$	$\mathbb{F}_2^*$	$\mathbb{F}_2^{128}$	128
4	PRØST-OTR-256	512	$\mathbb{F}_2^*$	$(\mathbb{F}_2^{512})^+$	$\mathbb{F}_2^*$	$\mathbb{F}_2^{256}$	256
5	PRØST-APE-256[256, 256]	256	$\mathbb{F}_2^*$	$(\mathbb{F}_2^{256})^+$	$\mathbb{F}_2^*$	$\mathbb{F}_2^{256}$	256
6	PRØST-APE-128[128, 128]	128	$\mathbb{F}_2^*$	$(\mathbb{F}_2^{128})^+$	$\mathbb{F}_2^*$	$\mathbb{F}_2^{128}$	128

For a message $M \in \mathcal{M}$, we write $M[i]$, $1 \leq i \leq w$, to denote the ith block of b consecutive bits of M, starting from the most significant end. We use an equivalent indexing for a ciphertext $C \in \mathcal{C}$.

When we write multiplications, we mean multiplication in a finite field $\mathbb{F}_{2^{2n}}$ defined modulo an irreducible polynomial $f_{2n}(x)$ over $\mathbb{F}_2$ of degree $2n$. The irreducible polynomials used for our proposals are given in Table 6. Constants in hexadecimal notation occurring in the proposal descriptions are considered members of the respective field, where the polynomial coefficients are given by the radix 2 representation of said constant. The least significant bit is the least significant coefficient. For example, 2 represents $x \in \mathbb{F}_{2^{2n}}$ and 7 represents $x^2 + x + 1 \in \mathbb{F}_{2^{2n}}$. Pseudocode showing multiplication by 2 is given in Algorithm 1.

4.2 Block Cipher-based AE to Permutation-based AE

It is a well-known result that any permutation can be turned into a block cipher by plugging it into a single-key Even-Mansour construction [8]. As such, any block cipher-based AE scheme can be turned into a permutation-based one by this construction.

Table 6: Defining polynomials for finite fields used in our proposals

Field	Defining polynomial $f_{2n}(x)$	Value added when reducing in hex
$\mathbb{F}_{2^{256}}$	$x^{256} + x^{10} + x^5 + x^2 + 1$	0x425
$\mathbb{F}_{2^{512}}$	$x^{512} + x^8 + x^5 + x^2 + 1$	0x125

Algorithm 1: xtime(v)

Data: $x \in \mathbb{F}_2^{2n}$
$f_{256} \leftarrow$ 0x423
$f_{512} \leftarrow$ 0x123
if $msb_1(v) = 1$ **then**
  return $(v \ll 1) \oplus f_{2n}$
else
  return $v \ll 1$
end

For the purpose of using PRØST-n in block cipher-based AE schemes, we define the single-key Even-Mansour block cipher, which takes a single $2n$-bit key K, as $\tilde{P}_{n,K} : \mathbb{F}_2^{2n} \times \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$, defined as $\tilde{P}_{n,K}(x) \stackrel{\text{def}}{=} K \oplus P_n(x \oplus K)$.

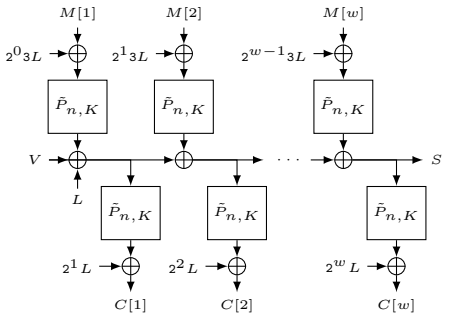
With PRØST, we have designed a permutation which has a simple design, is efficient, has strong security bounds and is inherently timing-attack protected. With this, we next define specific instantiations of authenticated encryption schemes based on PRØST, both when the underlying scheme is block cipher or permutation based, either by using it directly or through $\tilde{P}_{n,K}$.

4.3 PRØST-COPA-n

COPA is a design by Andreeva et al. from Asiacrypt 2013 [3]. In this part, we give our specification of PRØST-COPA-n; the instantiation of the COPA AE scheme using $\tilde{P}_{n,K}$ as the underlying block cipher. The resulting proposal PRØST-COPA-n enjoys the security proofs of COPA, and in particular, COPA is resistant to nonce-misuse, leaking only the length of the common message prefix (which is optimal for single-pass schemes) [3]. PRØST-COPA-n is a fully parallelizable, online AE scheme, and uses the very efficient doubling of Algorithm 1 as opposed to general multiplication in $\mathbb{F}_{2^{2n}}$ for the tweak values. The scheme uses two calls to PRØST-n and two doublings in $\mathbb{F}_{2^{2n}}$ message block.

The encryption of w message blocks is depicted in Figure 2 (processing associated data is not shown). The complete description of encryption with authentication, decryption with verification and processing of associated data for PRØST-COPA-n is given by Algorithms 2 through 5.

Figure 2: Encryption of w message blocks with PRØST-COPA-n



Algorithm 2: PRØST-COPA-n-$\mathcal{E}(M, A, N)$

Data: $M \in \mathcal{M}, A \in \mathcal{A}, N \in \mathcal{N}$

$V \leftarrow$ PRØST-COPA-n-ProcessAD(A, N) /* First process A and N */

$M \leftarrow M \| 10^*$ /* Pad M to positive multiple of b bits */

$L \leftarrow \tilde{P}_{n,K}(0^b)$ /* Set initial values */

$V \leftarrow V \oplus L$

$(\Delta_0, \Delta_1) \leftarrow (3L, 2L)$

$\Sigma \leftarrow 0$

for $i = 1, \ldots, w$ **do**

  $V \leftarrow \tilde{P}_{n,K}(M[i] \oplus \Delta_0) \oplus V$ /* Process message blocks */

  $C[i] \leftarrow \tilde{P}_{n,K}(V) \oplus \Delta_1$

  $(\Delta_0, \Delta_1) \leftarrow (2\Delta_0, 2\Delta_1)$

  $\Sigma \leftarrow \Sigma \oplus M[i]$

end

$T \leftarrow \mathrm{lsb}_\tau(\tilde{P}_{n,K}(\tilde{P}_{n,K}(\Sigma \oplus 2^{w-1}3^2 L) \oplus V) \oplus 2^{w-1}7L)$ /* Compute tag */

return (C, T)

Algorithm 3: PRØST-COPA-n-$\mathcal{D}(A, C, T)$

Data: $C \in \mathcal{C}, A \in \mathcal{A}, N \in \mathcal{N}, T \in \mathbb{F}_2^\tau$

$V \leftarrow$ PRØST-COPA-n-ProcessAD(A, N) /* First process A and N */
$V \leftarrow V \oplus L$
$(\Delta_0, \Delta_1) \leftarrow (3L, 2L)$
$\Sigma \leftarrow 0$
for $i = 1, \ldots, w$ **do**
  $V' \leftarrow \tilde{P}_{n,K}^{-1}(C[i] \oplus \Delta_1)$ /* Process ciphertext blocks */
  $M[i] \leftarrow \tilde{P}_{n,K}^{-1}(V' \oplus V) \oplus \Delta_0$
  $V \leftarrow V'$
  $(\Delta_0, \Delta_1) \leftarrow (2\Delta_0, 2\Delta_1)$
  $\Sigma \leftarrow \Sigma \oplus M[i]$
end
$T' \leftarrow \mathrm{lsb}_\tau(\tilde{P}_{n,K}(\tilde{P}_{n,K}(\Sigma \oplus 2^{w-1}3^2L) \oplus V) \oplus 2^{w-1}7L)$ /* Compute verification tag */
if $T' = T$ **then**
  return StripPadding(M)
else
  return $\perp$
end

Algorithm 4: PRØST-COPA-n-ProcessAD(A, N)

Data: $A \in \mathcal{A}, N \in \mathcal{N}$

$X \leftarrow A \| N$ /* Append N to A */
$L \leftarrow \tilde{P}_{n,K}(0^b)$ /* Set initial values */
$\Delta \leftarrow 3^3 L$
$V \leftarrow 0$
$\ell \leftarrow \lceil |X|/b \rceil$
for $i = 1, \ldots, \ell - 1$ **do**
  $V \leftarrow V \oplus \tilde{P}_{n,K}(X[i] \oplus \Delta)$ /* Process all blocks of X except the last */
  $\Delta \leftarrow 2\Delta$
end
if $|X[\ell]| = b$ **then**
  $V \leftarrow \tilde{P}_{n,K}(V \oplus X[\ell] \oplus 3\Delta)$ /* Process last block of X if full */
else
  $V \leftarrow \tilde{P}_{n,K}(V \oplus X[\ell] \| 10^* \oplus 3^2\Delta)$ /* Process last block of X if partial */
end
return V

Algorithm 5: StripPadding(X)

Data: $X \in \mathbb{F}_2^*$

while $X \neq \varepsilon$ **and** $lsb_1(X) = 0$ **do**

  $X = X \gg 1$ `/* Peel off zeroes in the end */`

end

if $X \neq \varepsilon$ **then**

  $X \leftarrow X \gg 1$ `/* If` $X \neq \varepsilon$ `remove the one-bit */`

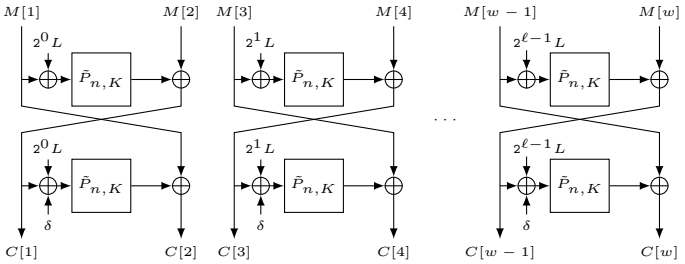
end

return X

4.4 PRØST-OTR-n

OTR is an AE scheme designed by Minematsu from Eurocrypt 2014 [13]. Here, we specify PRØST-OTR-n, which is the instantiation of OTR using $\tilde{P}_{n,K}$ as the underlying block cipher. OTR uses 2-round Feistel ciphers to encrypt two consecutive message blocks. It uses one call to PRØST-n and half a doubling in $\mathbb{F}_{2^{2n}}$ per message block.

Figure 3: Encryption of w message blocks with PRØST-OTR-n, when w is even



PRØST-OTR-n inherits the features of OTR. In particular, it is nonce-based, but not nonce-misuse resistant. This means that security is lost if nonces are re-used. Comparing to schemes which offer nonce-misuse resistance, the gain is performance. The proposal is online and completely parallelizable. It does not require the inverse of the underlying primitive, due to the employment of the Feistel cipher structure.

The encryption of w message blocks, for even w, with PRØST-OTR-n is depicted in Figure 3. In the figure, $\ell = w/2$. The complete description of encryption with authentication, decryption with verification and processing of associated data for PRØST-OTR-n is given by Algorithms 6 through 10.

Algorithm 6: PRØST-OTR-n-$\mathcal{E}(M, A, N)$

Data: $M \in \mathcal{M}, A \in \mathcal{A}, N \in \mathcal{N}$
$(C, TE) \leftarrow$ PRØST-OTR-n-Enc(M, N)
if $A \neq \epsilon$ **then** $TA \leftarrow$ PRØST-OTR-n-ProcessAD(A)
else $TA \leftarrow 0^b$
$T \leftarrow \text{msb}_\tau(TE \oplus TA)$
return (C, T)

Algorithm 7: PRØST-OTR-n-$\mathcal{D}(C, A, N, T)$

Data: $C \in \mathcal{C}, A \in \mathcal{A}, N \in \mathcal{N}, T \in \mathbb{F}_2^\tau$
$(M, TE) \leftarrow$ PRØST-OTR-n-Dec(C, N)
if $A \neq \epsilon$ **then** $TA \leftarrow$ PRØST-OTR-n-ProcessAD(A)
else $TA \leftarrow 0$
$T' \leftarrow \text{msb}_\tau(TE \oplus TA)$
if $T' = T$ **then**
  return StripPadding(M)
else
  return $\perp$
end

Algorithm 8: PRØST-OTR-n-ProcessAD(A)

Data: $A \in \mathcal{A}$
$\Xi \leftarrow 0^b$ `/* Set initial values */`
$\gamma \leftarrow \tilde{P}_{n,K}(0^b)$
$Q \leftarrow 4\gamma$
for $i = 1, \ldots, a-1$ **do**
  $\Xi \leftarrow \Xi \oplus \tilde{P}_{n,K}(Q \oplus A[i])$ `/* Process first` $a-1$ `blocks of` A `*/`
  $Q \leftarrow 2Q$
end
if $|A[a]| = b$ **then**
  $\Xi \leftarrow \Xi \oplus A[a]$
  $TA \leftarrow \tilde{P}_{n,K}(Q \oplus 2\gamma \oplus \Xi)$
else
  $\Xi \leftarrow \Xi \oplus A[a] \| 10^*$
  $TA \leftarrow \tilde{P}_{n,K}(Q \oplus \gamma \oplus \Xi)$
end
return TA

Algorithm 9: PRØST-OTR-n-Enc(M, N)

Data: $M \in \mathcal{M}, N \in \mathcal{N}$

$M \leftarrow M \| 10^*$ /* Pad M to positive multiple of b bits */
$\Sigma \leftarrow 0^b$ /* Set initial values */
$\delta \leftarrow \tilde{P}_{n,K}(N \| 10^*)$ /* Set δ to the encryption of the padded nonce */
$L \leftarrow 4\delta$
for $i = 1, \ldots, \lfloor w/2 \rfloor$ **do**
  $C[2i-1] \leftarrow \tilde{P}_{n,K}(L \oplus M[2i-1]) \oplus M[2i]$ /* Process M blocks in pairs */
  $C[2i] \leftarrow \tilde{P}_{n,K}(L \oplus \delta \oplus C[2i-1]) \oplus M[2i-1]$
  $\Sigma \leftarrow \Sigma \oplus M[2i]$
  $L \leftarrow 2L$
end
if w is even **then**
  $L' \leftarrow L \oplus \delta$
else
  $L' \leftarrow L$ /* Handle last block if w is odd */
  $C[w] \leftarrow \tilde{P}_{n,K}(L') \oplus M[w]$
  $\Sigma \leftarrow \Sigma \oplus M[w]$
end
$TE \leftarrow \tilde{P}_{n,K}(3L' \oplus \delta \oplus \Sigma)$ /* Compute encryption part of tag */
return (C, TE)

Algorithm 10: PRØST-OTR-n-Dec(C, N)

Data: $C \in \mathcal{C}, N \in \mathcal{N}$

$\Sigma \leftarrow 0$ /* Set initial values */
$\delta \leftarrow \tilde{P}_{n,K}(N \| 10^*)$ /* Set δ to the encryption of the padded nonce */
$L \leftarrow 4\delta$
for $i = 1, \ldots, \lfloor w/2 \rfloor$ **do**
  $M[2i-1] \leftarrow \tilde{P}_{n,K}(L \oplus \delta \oplus C[2i-1]) \oplus C[2i]$ /* Process C blocks in pairs */
  $M[2i] \leftarrow \tilde{P}_{n,K}(L \oplus M[2i-1]) \oplus C[2i-1]$
  $\Sigma \leftarrow \Sigma \oplus M[2i]$
  $L \leftarrow 2L$
end
if w is even **then**
  $L' \leftarrow L \oplus \delta$
else
  $L' \leftarrow L$ /* Handle last block if w is odd */
  $M[w] \leftarrow \tilde{P}_{n,K}(L') \oplus C[w]$
  $\Sigma \leftarrow \Sigma \oplus M[w]$
end
$TE \leftarrow \tilde{P}_{n,K}(3L' \oplus \delta \oplus \Sigma)$ /* Compute encryption part of tag */
return (M, TE)

4.5 PRØST-APE-$n[r, c]$

APE is a permutation-based AE scheme by Andreeva et al. from FSE 2014 [2]. It does not require a block cipher, but rather a permutation as the underlying primitive. Here, we give the specification of our instantiation of APE using PRØST-n as the underlying permutation. The $[r, c]$ part of the proposal name indicates the sizes of *rate* and *capacity*, two parameters which are described in the following.

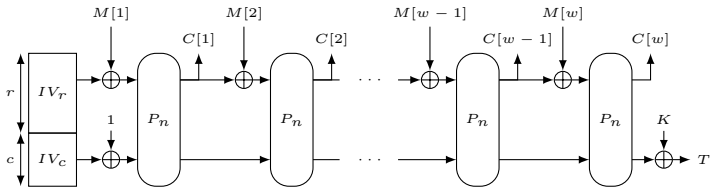


Figure 4: Encrypting and authenticating m message blocks of r bits each using PRØST-APE-$n[r, c]$

PRØST-APE-$n[r, c]$ is online in encryption, i.e. one does not need to know all message blocks, and in particular the number of message blocks, before one can start encrypting. Indeed, the ciphertext block $C[i]$ depends only on message blocks $M[1], \ldots, M[i]$. For decryption, however, one starts decrypting the last ciphertext block first, and hence decryption is *not* online. The fact that one decrypts in reverse implies the need for the inverse the underlying permutation as well.

PRØST-APE-$n[r, c]$ has two important parameters: the *rate* denoted r and the *capacity* denoted c. For PRØST-APE-$n[r, c]$, the block size b equals the rate r, and we use r rather than b in our description here. As the underlying permutation PRØST-n has a state size of $2n$ bits, we have $r + c = 2n$. It uses a single c-bit key $K \in \mathbb{F}_2^c$.

In our specification, subscript r or c denotes the rate, respectively capacity part of the $(r+c)$-bit operand, with the convention that the rate part holds the most significant bits of the state, i.e. $X_r = \mathrm{msb}_r(X)$ and $X_c = \mathrm{lsb}_c(X)$. Figure 4 illustrates the encryption and tag generation for w message blocks of r bits each. When there is no associated data, i.e. $|A| = 0$, the IV is set to $IV = (IV_r \| IV_c) = (0^r \| K)$. Nonce N is effectively considered part of the associated data, and is prepended to this. The complete description of encryption with authentication, decryption with verification and processing of associated data for PRØST-APE-$n[r, c]$ is given by Algorithms 11 and 12.

APE achieves privacy and integrity up to the bound $2^{c/2}$, both in the ideal permutation model and the standard model. In other words, the choice of rate and capacity influence performance and security, in the sense that increasing capacity and decreasing rate will increase security and decrease performance, and vice versa. We refer to Table 4 for the specific parameters of our proposals. APE is the first permutation-based AE scheme to obtain nonce-misuse resistance. In particular, when nonces are repeated, it leaks only the XOR of the common prefix of the message.

Algorithm 11: PRØST-APE-$n[r,c]$-$\mathcal{E}(M,A,N)$

Data: $M \in \mathcal{M}, A \in \mathcal{A}, N \in \mathcal{N}$

$M \leftarrow M\|10^*$ /* Pad M to positive multiple of r bits */
$V \leftarrow 0^r\|K$
$X \leftarrow N\|A\|10^*$ /* Prepend N to A and pad to positive multiple of r bits */
for $i = 1, \ldots, |X|/r$ **do**
  $V \leftarrow P_n((X[i] \oplus V_r)\|V_c)$ /* Process nonce and AD */
end
$V_c \leftarrow V_c \oplus (0^{c-1}\|1)$
for $i = 1, \ldots, w$ **do**
  $V \leftarrow P_n((M[i] \oplus V_r)\|V_c)$ /* Process message */
  $C[i] \leftarrow V_r$
end
$T \leftarrow V_c \oplus K$ /* Compute tag */
return (C, T)

Algorithm 12: PRØST-APE-$n[r,c]$-$\mathcal{D}(C,A,N,T)$

Data: $C \in \mathcal{C}, A \in \mathcal{A}, N \in \mathcal{N}, T \in \mathbb{F}_2^\tau$

$IV \leftarrow 0^r\|K$
$X \leftarrow N\|A\|10^*$ /* Prepend N to A and pad to positive multiple of r bits */
for $i = 1, \ldots, |X|/r$ **do**
  $IV \leftarrow P_n((X[i] \oplus V_r)\|V_c)$ /* Process nonce and AD */
end
$IV_c \leftarrow IV_c \oplus (0^{c-1}\|1)$
$C[0] \leftarrow IV_r$ /* Set dummy $C[0] = IV_r$ for a smoother loop */
$V \leftarrow (0^r\|(K \oplus T))$
for $i = w, \ldots, 1$ **do**
  $V \leftarrow P_n^{-1}(C[i]\|V_c)$ /* Process ciphertext */
  $M[i] \leftarrow V_r \oplus C[i-1]$
end
if $IV_c = V_c$ **then**
  return StripPadding(M)
else
  return $\perp$
end

5 Security Goals

Table 7 summarizes our security claims for our six proposals with respect to confidentiality of plaintext, integrity of plaintext and integrity of associated data. Note that in all cases, the claimed security level is $n/2$ for a proposal using PRØST-n. This is because the security proofs for the modes in which we use PRØST all give this security bound, when using a primitive which is indistinguishable from a random permutation/block cipher up to the n-bit bound, as is the case for PRØST-n.

Table 7: Proposal ranks and security claims (in bits) for our proposals for plaintext confidentiality (PT_{CONF}), plaintext integrity (PT_{INT}) and associated data integrity (AD_{INT})

Rank	Proposal	PT_{CONF}	PT_{INT}	AD_{INT}
1	PRØST-COPA-128	64	64	64
2	PRØST-COPA-256	128	128	128
3	PRØST-OTR-128	64	64	64
4	PRØST-OTR-256	128	128	128
5	PRØST-APE-256[256, 256]	128	128	128
6	PRØST-APE-128[128, 128]	64	64	64

5.1 Remarks regarding additional security

The COPA and APE proposals offer a level of resistance against nonce-misuse. For the details of the nonce-misuse resistance of APE and COPA parameter sets, we refer to their respective mode papers [2, 3]. Additionally, all proposals have the following additional security goals:

- Any straight-forward implementation will run in constant time, and
- Due to the choice of a lightweight 4-bit S-box as the only non-linear element, countermeasures against power/EM side-channel attacks are much cheaper.

6 Design Rationale

In this section we discuss the design rationale for both the PRØST permutation but also for choices made when using PRØST in the existing AEAD modes COPA, OTR and APE.

The main design rationale for the PRØST permutation was efficiency and strong, easily verifiable security arguments:

- **Strong arguments** are possible as we follow the wide-trail strategy. Here we modified the usual AES-like structure by interleaving two different `ShiftRows`-like operations in the $\texttt{ShiftPlanes}_i$ operation. This results in significantly improved bounds on the best linear and differential characteristics.
- **Efficiency** is a result of mainly two efforts. First, we optimized every single component with respect to implementation cost. Second, the strong arguments mentioned above allows to keep the number of rounds low.

Below, we describe the design rationale behind the separate components within the PRØST permutation.

6.1 SubRows

For the SubRows operation, we use a very simple (in terms of hardware/software-efficiency – the formulation is given in Appendix B), 10-instruction, 4-bit S-box $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$. The concrete S-box was the result of a hardware assisted search though a significant subset of all possible S-boxes. Besides being very efficient in terms of cycle count, this S-box is also optimal with respect to linear and differential attacks.

We chose S among all S-boxes fulfilling the following criteria.

1. S is an involution, which prevents the encryption/decryption overhead,
2. The maximal probability of a differential is $1/4$,
3. There are exactly 24 differentials with probability $1/4$,
4. The maximal absolute bias of a linear approximation is $1/4$,
5. There are exactly 36 linear approximations with absolute bias $1/4$, and
6. Output bits have algebraic degrees of 2, 2, 3 and 3, respectively.

Having only one single S-box within one plane allows to implement the S-box application using bit-slicing. On top, keeping the S-boxes identical for all planes and all rounds reduced the code space and avoids additional overhead. The increased danger of symmetries throughout the cipher is countered by relatively heavy round constants.

6.2 MixSlices

We had three major requirements for the MixSlices operation.

- Linear and differential branch number 5,
- Low density, and
- As a heuristic to minimize implementation characteristics, for both encryption and decryption, MixSlices is its own inverse.

We elaborate on the first two requirements below.

6.2.1 High Branch Number

The main design goal of the MixSlices transformation is to follow the wide trail strategy. Hence, the MixSlices transformation is related to an $\mathbb{F}_2$-linear error-correcting code over $\mathbb{F}_2^4$ with minimum distance 5. Note that in our setting, linear and differential branch numbers are identical.

In other words, a difference in $k > 0$ rows of a slice will result in a difference of at least $5 - k$ rows after one MixSlices application. While this is a good bound for 2 rounds, only the interaction with the $\texttt{ShiftPlanes}_i$ operation guarantees an overall secure design.

6.2.2 Low density

The density roughly corresponds to the number of XOR operations that have to be performed when implementing the matrix. It is therefore a suitable metric when optimizing performance – both in software and hardware.

Among all 16×16 binary matrices, we search through the involutive ones with branch number 5 and with a particular low Hamming weight. The optimal solution, i.e. the lowest Hamming weight of such a matrix, we were able to find with our hardware-assisted search had Hamming weight 88. Note that we cannot guarantee that our matrix is actually optimal (as the minimal number of ones in such a matrix is generally unknown).

6.3 $\texttt{ShiftPlanes}_i$

We had two design criteria for this transformation. Firstly, the shift values should result in full diffusion after as few rounds as possible. Secondly, the differential/linear trails with fewest active S-boxes over a given number of rounds, should have as many active S-boxes as possible.

The (π_0, π_1) pairs for PRØST (see Table 2) were found to be optimal in the sense that they give the best diffusion, number of active S-boxes (see also Sections 7.1 and 7.2) and implementation cost, for the specified register lengths d.

For register length $d = 16$ we obtain full diffusion after 3 rounds and for $d = 32$ after 4 rounds. For lower bounds on the number of active S-boxes over various number of rounds, we refer to Section 7.2.

With respect to implementation cost, we have optimized to have as many multiples of 8 as possible, as these are free on 8-bit platforms and cheap on larger platforms. For the constants that are not multiples of 8, we minimize their sum, as the implementation cost is proportional to the constant. This sum, not counting multiples of 8, is 22 for the $d = 16$ case and 88 for the $d = 32$ case, which roughly translates to the implementation cost in cycles.

6.4 $\texttt{AddConstants}_i$

The purpose of adding round constants is to make each round different. If the rounds are all the same, then fixed points x such that $R(x) = x$ for the round function R extend to the entire permutation. For example, if $P = R^{10}$, then fixed points for R^2 and R^5 would also extend to P. Therefore, one can expect several fixed points for P, whereas for an ideal permutation only a single fixed point is expected. By choosing round-dependent constants for $\texttt{AddConstants}_i$, we expect the number of fixed points to be close to 1.

We use the most significant d bits of two 32-bit constants c_0 and c_1 from which all round constants are derived through rotations. Round i uses $c_0 \lll (i + j)$ for even j and $c_1 \lll (i + j)$ for odd j, where $j = 4x + y$ is the lane number of $s_{x,y}$. The two constants $c_0 =$ `0x75817b9d` and $c_1 =$ `0xb2c5fef0` are generated from the first 64 digits after the decimal point of π as illustrated by the code in Listing 1.

Listing 1: Code for generating c_0, c_1

```
#include <stdint.h>
#include <stdio.h>

const uint32_t pi[64] = {1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,
                         3,8,4,6,2,6,4,3,3,8,3,2,7,9,5,0,
                         2,8,8,4,1,9,7,1,6,9,3,9,9,3,7,5,
                         1,0,5,8,2,0,9,7,4,9,4,4,5,9,2,3};
int main(void)
{
        uint32_t c1=0;
```

```
        uint32_t c2=0;
        int i;
        for(i=0;i<32;i++) {
                c1 |= (pi[i]&1) << i;
                c2 |= (pi[i+32]&1) << i;
        }
        printf("c1 = 0x%08x\n", c1);
        printf("c2 = 0x%08x\n", c2);
        return 0;
}
```

6.5 Ciphertext Expansion in AEAD Candidates

Motivated by simplicity of implementation aspects we choose, for all proposals in this document, to *always* do ciphertext expansion by applying 10^* padding to the message as the first thing before encryption. The reasoning behind this choice is that:

- This approach is less prone to implementation errors, which are frequently a source of real-world security break-downs. By effectively eliminating the special cases an implementation has to be able to handle, chances of introducing implementation errors are diminished,
- The choice to always do ciphertext expansion is easier to optimize in both software and hardware, which in turn aids comparisons, and
- The impact on size of implementation can be significant. Be it code size in software or control logic in hardware, lower complexity of padding rules leads to smaller implementations.

By allowing a minor ciphertext expansion, in comparison to average packet sizes in some corner use-cases, we achieve the advantages mentioned above. We believe this is the right trade-off for the vast majority of use-cases in practice, and indeed will facilitate greater ease of application.

7 Security Analysis

In this section we summarize some of our findings in the security analysis of PRØST. A more extensive treatment will be available from [19].

7.1 Diffusion and Strict Avalanche Criterion

The well-known concept of diffusion was first defined by Shannon in his 1949 seminal work [18]. In its original meaning, the goal of diffusion is to lessen the short-term statistical properties from the plaintext in the resulting ciphertext, such that an attacker would need to obtain a lot of ciphertext to infer knowledge of the statistical properties of the plaintext. The modern definition of diffusion is slightly different, and we take it to mean the extent to which output bits depend on input bits. We say we have *full diffusion* when each bit of the output depends on each bit of the input.

A condition somewhat parallel to diffusion is the *Strict Avalanche Criterion* (SAC). A cipher or permutation is said to satisfy the SAC when flipping any bit of the input results in flipping each bit of the output with probability 1/2 (for a block cipher, one averages over all keys). In this section we present our findings on diffusion and the SAC for the PRØST permutation.

Lastly, we consider the *avalanche effect*, which is the property that on average, half the bits of the output are flipped when a single input bit is flipped.

7.1.1 Diffusion

For our analysis, we assume that for the 4-bit S-box used, each of the 4 output bits depend on each of the 4 input bits. Also, we assume that the `MixSlices` operation mixes the bits in a column, so each of the 4 output bits depend on each of the 4 input bits.

After applying `SubRows` of the first round, all bits within a row are interdependent by assumption. Applying `MixSlices` of the first round implies that we get bit interdependency between all bits in the same slice. The $\texttt{ShiftPlanes}_i$ operation cyclically shifts the planes by a round-dependent amount, essentially mixing the slices.

As `SubRows` and `MixSlices` take care of mixing bits in slices in each round, the question becomes how many rounds are required to make the plane shifting of $\texttt{ShiftPlanes}_i$ create bit-interdependency between all planes.

When determining the choice of (π_0, π_1) vectors, we conducted an experiment trying out all combinations of $\pi_0, \pi_1 \in \mathbb{Z}_d^4$, and investigate the required number of rounds to obtain full diffusion. For $d = 16$ we found that the best pairs (π_0, π_1) require a minimum of 3 rounds for full diffusion, and there are 4096 such pairs. For $d = 32$ a minimum of 4 rounds are required for full diffusion, and there are 729088 pairs (π_0, π_1) obtaining this minimum. Unfortunately, there is no pair (π_0, π_1) which obtains the best diffusion for both $d = 16$ and $d = 32$, hence the two parameter sets in Table 2.

7.1.2 SAC and Avalanche Effect

For the purpose of investigating the SAC and avalanche effect, we conduct randomized experiments to measure the degree to which these two properties are obtained. For the $2n$-bit permutation PRØST-n, the statements of the avalanche effect and SAC are given by (2) and (3), respectively, where $\mathbf{e}_i$ denotes $2n$-bit string with a 1 on position i and zeroes elsewhere, and subscript j denotes the jth bit:

$$\forall i \in \{1, \ldots, 2n\} \quad : \quad 2^{-2n} \sum_{x \in \mathbb{F}_2^{2n}} \mathrm{wt}\left(\text{PRØST-}n(x) \oplus \text{PRØST-}n(x \oplus \mathbf{e}_i)\right) = n \tag{2}$$

$$\forall i \in \{1, \ldots, 2n\}, \forall j \in \{1, \ldots, 2n\} \quad : \quad \mathbf{Pr}[\text{PRØST-}n(x)_j \neq \text{PRØST-}n(x \oplus \mathbf{e}_i)_j] = \frac{1}{2} \tag{3}$$

To experimentally measure the extent to which PRØST meets these two criteria, we sample a random $X \subset \mathbb{F}_2^{2n}$ and define the degree of avalanche effect $\deg_{ava}$ and degree of SAC $\deg_{SAC}$ as defined by Serf in his investigation of the AES finalists [17]. Our results show that we obtain SAC and avalanche effect degrees of 1.0 (which is ideal) after 2-3 rounds for $d = 16$ and 3-4 rounds for $d = 32$.

7.2 Bounds on Active S-boxes

Here we give lower bounds on the number of active S-boxes for various PRØST parameters. These bounds are of paramount importance as, combined with the differential- and linear properties of the S-box (see Section 6.1), provide upper bounds on the differential- and linear trail probabilities for the full PRØST permutation.

To lower bound the number of active S-boxes for $d = 16$ and $d = 32$, we model the propagation of active S-boxes over a particular number of rounds as an integer programming problem (see Appendix A). The particular choice of (π_0, π_1) used in PRØST arise from solving this problem for randomly chosen subsets of the (π_0, π_1) giving optimal diffusion (see Section 7.1.1), for $d = 16$ and $d = 32$, and choosing those giving the best bounds.

The findings for the number of active S-boxes for the (π_0, π_1) from Table 2, for various number of rounds, are given in Table 8. The integer programming problem is modeled in Appendix A.

Table 8: Lower bounds on the number of active S-boxes for $d \in \{16, 32\}$ for various number of rounds. In the $d = 32, T = 8$ case, the number in parenthesis was the best obtained when the solver stopped due to memory limitations.

	Rounds T				
d	4	5	6	7	8
16	25	41	85	96	105
32	25	41	105	169	(210)

8 Features

All the proposed parameter sets have the following features of the permutation in common.

- Designed for straight-forward bit-sliced implementations,
- Even the most straight-forward implementation of the permutation leads to constant execution times,
- Due to the choice of a lightweight 4-bit S-box as the only non-linear element, countermeasures against power/EM side-channel attacks are much cheaper than ARX or AES-based designs,
- Fast and compact in software on a wide range of platforms, and
- Fast, compact, energy-efficient, and allows for low-latency implementations in hardware implementations.

We will back up these claim in upcoming supplementary documents on implementation aspects of Prøst.
All the proposed modes have the following features in common:

- They are based on a large cryptographically secure permutation. This is arguably one of the simplest, if not the simplest way, to build primitives for symmetric key cryptography, including authenticated encryption primitives. In particular, this avoids complicated and often very inelegant considerations related to key schedules for traditional block cipher based constructions, and
- They are based on a *single* permutation. Having a single permutation further avoids considerations related to the independence of permutations.

All of the above described features are advantages over AES and AES-GCM. In addition, we to point out the following features which is shared with AES-GCM, but other proposals are often lacking:

- A simple and clean design, and
- Strong bounds against large classes of cryptanalytic attacks, like linear and differential cryptanalysis which are among the most powerful attack vectors known. This has two advantages. Firstly it gives assurance that these attack vectors in their basic form do not lead to attacks

on the proposal. On top of that we chose a number of rounds that gives a large security margin. Secondly, it saves valuable time with respect to cryptanalysis. Finding and improving upon known statistical properties for linear and differential attacks is a very time consuming task for cryptanalysts. Knowledge of bounds saves a lot of this time, which can instead be spent on other attack vectors or more advanced attacks. Most of our bounds are tight, i.e. we are able to give matching characteristics/trails [19]. This will be of independent interest for external cryptanalysts.

Now we discuss more specific features of the parameter sets. We start with PRØST-COPA-n, our main proposal, that comes with two security levels. On top of the advantages mentioned above, it offers a level of nonce-misuse resistance. We feel this is an important property as in many environments this seemingly simple task of keeping track of nonces and making sure they are unique, e.g. by implementing a counter, can actually be very difficult. Resets in virtualized environments, or very resource constrained environments, are examples of such situations.

For environments where respecting the requirements for unique nonces is easy to achieve, we propose PRØST-OTR-n, again with two security levels. This comes with improved implementation characteristics.

Both classes of parameter sets allow for parallelization, which, together with the bit-sliced nature of our permutation, allows for very efficient implementations on modern SIMD architectures.

As a complementary set of parameters, we propose PRØST-APE-n. Its advantages are the possibility of a very fine-grained adjustment of the security against generic attacks, by allowing different performance/security trade-offs through rate/capacity adjustments. To focus cryptanalytic attention, we do, however, limit ourselves to the two main security levels. Also, in contrast to the PRØST-COPA-n and PRØST-OTR-n proposals, we do not need twice the key length for the claimed security level.

For the details of the nonce-misuse resistance of APE and COPA parameter sets, we refer to their respective mode papers [2, 3].

9 Intellectual Property

The design team behind the PRØST submission for the CAESAR competition are not aware of any known patents, patent applications, planned patent applications or other intellectual-property constraints pertaining to the use of the cipher. If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

10 Consent

The submitter/submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter/submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter/submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter/submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are

not subject to appeal. The submitter/submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

References

[1] F. Abed, S. Fluhrer, C. Forler, E. List, S. Lucks, D. McGrew, and J. Wenzel. Pipelineable On-Line Encryption. In C. Cid and C. Rechberger, editors, *Fast Software Encryption*, LNCS. Springer, 2014.

[2] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In C. Cid and C. Rechberger, editors, *Fast Software Encryption*, LNCS. Springer, 2014. http://eprint.iacr.org/2013/791/.

[3] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser, and K. Yasuda. Parallelizable and Authenticated Online Ciphers. In K. Sako and P. Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013*, volume 8269 of *LNCS*, pages 424–443. Springer, 2013. eprint.iacr.org/2013/790/.

[4] E. Andreeva, A. Luykx, B. Mennink, and K. Yasuda. COBRA: A Parallelizable Authenticated Online Cipher Without Block Cipher Inverse. In C. Cid and C. Rechberger, editors, *Fast Software Encryption*, LNCS. Springer, 2014.

[5] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak specifications. Submission to NIST, 2008. http://keccak.noekeon.org/Keccak-specifications.pdf.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *LNCS*, pages 320–337. Springer, 2012. http://eprint.iacr.org/2011/499.pdf.

[7] B. Bilgin, A. Bogdanov, M. Knezevic, F. Mendel, and Q. Wang. Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In G. Bertoni and J.-S. Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 142–158. Springer-Verlag Berlin Heidelberg, 2013.

[8] O. Dunkelman, N. Keller, and A. Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 336–354, 2012. http://eprint.iacr.org/2011/541/.

[9] E. Fleischmann, C. Forler, S. Lucks, and J. Wenzel. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. Cryptology ePrint Archive, Report 2011/644, 2011. http://eprint.iacr.org/2011/644/.

[10] D. Khovratovich. PPAE: parallelizable permutation-based authenticated encryption. Presentation at Directions in Authenticated Ciphers – DIAC 2013, 2013. http://2013.diac.cr.yp.to/slides/khovratovich.pdf.

[11] T. Krovetz and P. Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.

[12] D. A. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM), 2004. http://www.cryptobarn.com/papers/gcm-spec.pdf.

[13] K. Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In *EUROCRYPT*, pages 275–292, 2014.

[14] N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In *Inscrypt*, pages 57–76, 2011.

[15] P. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In P. J. Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, 2004. http://www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf.

[16] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.

[17] P. Serf. The degrees of completeness, of avalanche effect, and of strict avalanche criterion for MARS, RC6, Rijndael, Serpent, and Twofish with reduced number of rounds. 2000.

[18] C. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28:656–715, 1949. http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf.

[19] The Prøst team. Security Analysis of Prøst, 2014 (to appear).

[20] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610 (Proposed Standard), 2008. http://www.ietf.org/rfc/rfc3610.txt.

[21] H. Wu and B. Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. Cryptology ePrint Archive, Report 2013/695, 2013. http://eprint.iacr.org/2013/695/.

A Bounds on Active S-boxes

To determine lower bounds on the number of active S-boxes for a given number of rounds T, we employ the modeling of the permutation and its propagation of active S-boxes as a mixed integer programming problem. This is similar to the approach by Mouha et al. [14].

Consider the $2n$-bit state as viewed from the (y, z)-plane, which yields a two-dimensional structure of h rows and d columns (we use here the more general size of h rows in each slice). We now let $x_{i,j}^t$ be a binary variable with $0 \leq i < h$, $0 \leq j < d$ and $0 \leq t \leq T$. We let $x_{i,j}^t = 1$ if and only if row i in slice j is active when going into round t, starting from $t = 0$.

We model the propagation of active S-boxes as the propagation of ones in x over time t. First, to require at least one active S-box in the input, we use the constraint given by (4):

$$\sum_{i=0}^{h-1} \sum_{j=0}^{d-1} x_{i,j}^0 \geq 1. \tag{4}$$

To model a round of PRØST, we need to make sure that

- Firstly, when a slice has $k > 0$ active S-boxes, then after applying `MixSlices`, the corresponding output slice has at least $5 - k$ active S-boxes, and
- Secondly, the plane rotations given by the amounts indicated by π_0 and π_1 in the $\texttt{ShiftPlanes}_i$ operation are correctly modeled in the same round, i.e. using the same t value.

These two requirements are modeled by the two constraints given as (5) and (6), where a_j^t defined for $t \in \{0, \ldots, T-1\}, j \in \{0, \ldots, d-1\}$ is an auxiliary binary variable equals 1 if and only if $\sum_{i=0}^{h-1} x_{i,j}^t \geq 1$. Here, $\mathcal{B}$ is the branch number, so for PRØST we have $\mathcal{B} = 5$.

$$\forall t \in \{0, \ldots, T-1\}, \forall j \in \{0, \ldots, d-1\} \quad : \quad \sum_{i=0}^{h-1} x_{i,j}^t + \sum_{i=0}^{h-1} x_{i,(j+\pi_{t \bmod 2,i} \bmod d)}^{t+1} \geq B \cdot a_j^t \tag{5}$$

$$\forall t \in \{0, \ldots, T-1\}, \forall j \in \{0, \ldots, d-1\}, \forall i \in \{0, \ldots, h-1\} \quad : \quad a_j^t \geq x_{i,j}^t \tag{6}$$

Finally, (7) gives the objective variable z which we want to minimize to find the T-round trail with fewest active S-boxes:

$$z \stackrel{\text{def}}{=} \sum_{t=0}^{T-1} \sum_{i=0}^{h-1} \sum_{j=0}^{d-1} x_{i,j}^t. \tag{7}$$

B S-box Formulation

For an input $(a\|b\|c\|d)$ with $a, b, c, d \in \mathbb{F}_2$, the following is a fast implementation of $(a\|b\|c\|d) = S(a\|b\|c\|d)$.

$$\begin{aligned} p &= a \\ q &= b \\ a &= c \oplus (p \odot q) \\ b &= d \oplus (q \odot c) \\ c &= p \oplus (a \odot b) \\ d &= q \oplus (b \odot c) \end{aligned}$$

C Changes from PRØST v1.0 to PRØST v1.1

- Added Section 2 which introduces document-wide notation
- In the PRØST permutation specification:
  - Changed state name to s.
  - Changed description to work on $2n$ bits rather than n bits, to avoid confusion with the description of Section 4.
  - Changed the name of `AddConstant` to $\texttt{AddConstants}_i$ to stress round number dependency.
  - Changed the name of `ShiftPlanes` to $\texttt{ShiftPlanes}_i$ to stress round number dependency.
  - Corrected $S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ to $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ in the `SubRows` description.
  - Cleaned up the description of $\texttt{AddConstants}_i$.
  - Added Section 3.1 which describes the mapping of a byte stream to PRØST state and vice versa, and also describes the bit ordering when XORing with $2n$-bit values.
- In the CAESAR candidate specification section:
  - Introduced and clarified notation.
  - Clarified the parameter sets for each of the six candidates, i.e. defined suggested tag lengths, nonce lengths and specifications of finite fields $\mathbb{F}_{2^{256}}$ and $\mathbb{F}_{2^{512}}$.
  - Clarified how to handle fractional cases. Motivated by simplicity of implementation aspects, we choose to *always* do ciphertext expansion by applying 10^* padding to the message stream as the first thing before encryption. For more on this choice, see Section 6.5.
  - Rewrote pseudo-code to reflect updated notation and clarifications.
  - Fixed m which should say w in Figures 2, 3 and 4.
- Made consistent the use of $\mathbb{F}$ for finite fields rather than GF notation.
- Corrected the rotation direction in the description of design rationale for the $\texttt{AddConstants}_i$ operation.
- Minor corrections to the model describing the search for lowest weight differential/linear trails of Appendix A.
- Added Section 6.5 which motivates the choice to always do ciphertext expansion through 10^* message padding.
- Made the correction that PRØST achieves full diffusion after 3 and 4 rounds, rather than 2 and 3 rounds, for $d = 16$ and $d = 32$, respectively.