π–*Cipher v2.0*[1,2]

Designers: Danilo Gligoroski[3] and Hristina Mihajloska[4] and Simona Samardjiska[3,4] and Håkon Jacobsen[3] and Mohamed El-Hadedy[5] and Rune Erlend Jensen[6] and Daniel Otte [7]

Submitter: Hristina Mihajloska

hristina.mihajloska@finki.ukim.mk

12.10.2015

[1]Since, the name of the cipher contains the Greek letter π, in the software implementations we will use the name `PiCipher`. More precisely in this document we propose the following four variants of the cipher: `Pi16Cipher096v2`, `Pi32Cipher128v2`, `Pi64Cipher128v2`, `Pi64Cipher256v2`

[2]This is a compilation of the documentation performed on 12.10.2015. The cipher is not changed, and the submitted Reference C code on 15.09.2015 corresponds to the textual description of the cipher. However, this compilation corrects the mistakes in the pseudo-code given in the initial submission.

[3]ITEM, Norwegian University of Science and Technology, Trondheim, Norway

[4]FCSE, "Ss Cyril and Methodius" University, Skopje, Republic of Macedonia

[5]Department of Computer Science, University of Virginia, Charlottesville, Virginia

[6]IDI, Norwegian University of Science and Technology, Trondheim, Norway

[7]Ruhr Universität Bochum, Germany

Changes

Changes in the cipher

π-Cipher v2.0 has one essential tweak and one reduction of the number of rounds in comparison with π-Cipher v1.0.

1. **Padding Rule (essential tweak).** In π-Cipher v1.0, the padding rule for the last block of the AD is the following:

$$AD_a \leftarrow \begin{cases} AD_a & \text{if } |AD_a| = bitrate, \\ AD_a||10^* & \text{if } |AD_a| < bitrate, \end{cases}$$

and for the last block of the message M is the following:

$$M_m \leftarrow \begin{cases} M_m & \text{if } |M_m| = bitrate, \\ M_m||10^* & \text{if } |M_m| < bitrate, \end{cases}$$

where 1 represents the byte `0x01`, and 0 represents the byte `0x00`.

We modify the padding rule as following: "Append 1 in any case, and fill the rest of the block with 0s". Thus, the changes will be:

The padding rule for the associated data AD is the following:

$$Pad(AD) = AD_1||AD_2||\ldots||AD_a||10^*$$

and for the message M is the following:

$$Pad(M) = M_1||M_2||\ldots||M_m||10^*$$

where 1 represents the byte `0x01`, and 0 represents the byte `0x00`.

Note that if the associated data AD (the message M) has length that is a multiple of the *rate*, then the number of processed blocks of AD (M) is increased by one, and thus $a \leftarrow a + 1$ ($m \leftarrow m + 1$).

2. **Reduction of the number of rounds.** We decided to reduce the number of rounds from 4 to 3.

Changes in explanation

We made some improvements in the documentation of the π-Cipher v1.0. This is the list of changes:

1. We reduce the number of variants of the cipher. Instead of the six variants in v1.0, now we have just four: π16-Cipher096, π32-Cipher128, π64-Cipher128 and π64-Cipher256.

2. We add an algorithmic description of the encryption/authentication and decryption/verification procedures in Section 1.2.

3. In Section 4.2.1 we give clarification about the feature Tag second preimage resistance - resistance against finding second preimage for an authentication tag when the key is known (insider attack) for short messages.

4. From v2.0 π-Cipher supports the concept of "open authorship" and that is explained in the two new paragraphs of the Chapter 6.

5. We add new parts in the documentation of π-Cipher, as follows:

   - Section 3.1: The security proof of π-Cipher.
   - Section 4.2.2: Explanation of how to use tweakable parameter N for wide blocks.
   - Section 4.2.3: Explanation of how to securely use incremental property of π-Cipher.
   - Section 4.2.4: Rational why we consider π-Cipher to be STREAM OAE2$^+$ design.

Table of Contents

Chapter 1

Specification

1.1 Parameters, variables and constants

The following parameters and variables are used in the specification of π-Cipher:

$\pi\omega$-Cipher$n$	AEAD cipher defined with ω-bit words and n-bit security.
$\omega = 16, 32, 64$	Size of binary words in bits that are used in π-Cipher.
π function	Main permutation function of the cipher.
M	Message or plaintext. Thus, $M = M_1 \| \| M_2 \| \| \ldots \| \| M_m$.
$mlen$	Length of a message less than $2^{64} - 1$ bytes.
m	Number of message blocks.
K	Secret key.
$klen$	Length of a key K in bytes. It can be 12 bytes (96 bits), 16 bytes (128 bits) or 32 bytes (256 bits).
AD	Associated data. This data can not be encrypted or decrypted.
$adlen$	Length of an associated data less than $2^{64} - 1$ bytes.

a	Number of associated data blocks. Thus, $AD = AD_1 \| \| AD_2 \| \| \ldots \| \| AD_a$.
IS	Internal state, bijectively transformed by the π function. Throughout this document when IS is used as a common internal state for many parallel computations, we will use the abbreviation CIS (*Common Internal State*).
I_i	Chunk from the internal state IS. It is a 4-tuple of ω–bit words. $I_i = (I_{i1}, I_{i2}, I_{i3}, I_{i4})$.
N	The internal state IS is divided into N chunks of length $4 \times \omega$. N is even number and $N \geqslant 4$. Thus, $IS = (I_1, I_2, \ldots, I_N)$.
b	Size of IS in bits. It is constrained by the following relation: $b = N \times 4 \times \omega$.
$rate$	Rate of the IS from the sponge construction paradigm point of view. $IS_{rate} = I_1 \ \| \ I_3 \ \| \ \ldots \ \| \ I_{N-1}$.
r	Size of IS_{rate} in bits.
$capacity$	Capacity of the IS from the sponge construction paradigm point of view. $IS_{capaity} = I_2 \ \| \ I_4 \ \| \ \ldots \ \| \ I_N$.
c	Size of $IS_{capacity}$ in bits.
$\|\|\|\|$	Operator of interleaved concatenation in order to correctly denote a concatenation of IS_{rate} and $IS_{capaity}$ that restores IS i.e., $IS = IS_{rate} \ \|\|\|\| \ IS_{capaity}$.
PMN	Public message number. The size $\|PMN\|$ in bits is constrained by the following relation: $8 \times klen + \|PMN\| + 8 \leqslant b$.
SMN	Secret message number. The size $\|SMN\|$ in bits is constrained by the following relations: $\|SMN\| = 0$ or $\|SMN\| = r$.
$NONCE$	$NONCE = (PMN, SMN)$.

C	Ciphertext.
$clen$	Length of the ciphertext in bytes, where $clen = mlen + \|SMN\| + tlen$ (here $\|SMN\|$ is in bytes).
T	Authentication tag for the message, $NONCE$ and associated data.
$tlen$	Length of the authentication tag in bytes. It is constrained by the following relation: $tlen \leqslant \frac{r}{8}$.
R	Tweakable parameter that represents the number of rounds in π function.
ctr	64-bit counter used in the cipher. It is initialized from the first 64 bits of the $IS_{capacity}$.
$\boxplus_d$	Operation of componentwise addition of two d-dimensional vectors of ω-bit words in $(\mathbb{Z}_{2^\omega})^d$.

π-Cipher is designed for different word sizes and different security levels. The recommended variants are presented in Table 4.1.

Table 1.1: Basic characteristics of all variants of the π-Cipher

	Word size ω (in bits)	$klen$ (in bits)	PMN (in bits)	SMN (in bits)	b (in bits)	N	$rate$ (in bits)	Tag T (in bits)	R
π16-Cipher096	16	96	32	0 or 128	256	4	128	$\leqslant 128$	3
π32-Cipher128	32	128	128	0 or 256	512	4	256	$\leqslant 256$	3
π64-Cipher128	64	128	128	0 or 512	1024	4	512	$\leqslant 512$	3
π64-Cipher256	64	256	128	0 or 512	1024	4	512	$\leqslant 512$	3

1.2 General design properties

π-Cipher is parallel, incremental, provably secure, nonce based authenticated encryption cipher with associated data that offers some level of tag second-preimage resistance. It involves several solid cryptographic concepts such as:

1. The design belongs to the category of Encrypt-then-MAC authenticated ciphers.

2. Its parallel and incremental design is similar to the design of the counter based XOR MAC scheme of [2], but in order to achieve the second-preimage resistance for the MAC tags, instead of XOR operations for the intermediate tag components we use componentwise additions in $(\mathbb{Z}_{2^\omega})^d$.

3. In [5, Sec. 3.3] the authors mention that it is possible to construct an authenticated encryption using two pass sponge construction [4] that is proven to be secure as long as the underlying sponge permutation has no structural distinguishers. In the same paper [5] the duplex sponge construction is introduced. Although these constructions have the property to be tag second-preimage resistant, neither of them is incremental. An incremental and parallel two pass scheme is proposed in [13]. However, the design goal for that scheme was not to be tag second-preimage resistant. Combining all these ideas, in our design we use a two pass counter based sponge component that we call *triplex component*. The used π permutation is based on ARX (Addition, Rotation and XOR) operations.

CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) [3] has the term *Robustness* as one of its main goals. π-Cipher offers the following features that can be connected with the robustness:

- **Tag second-preimage resistance**. π-Cipher offers *some level* of tag second-preimage resistance. We say some level, since the computational efforts for finding a second-preimage for a given pair *(message, tag)* are not the same as for finding second-preimages for hash functions (see Section 4.2.1).

- **An intermediate level of nonce-misuse resistance**. In this case the robustness is manifested when legitimate key holder uses the same key K, the same associated data AD, the same public message number PMN but different secret message numbers SMN_1 and SMN_2.

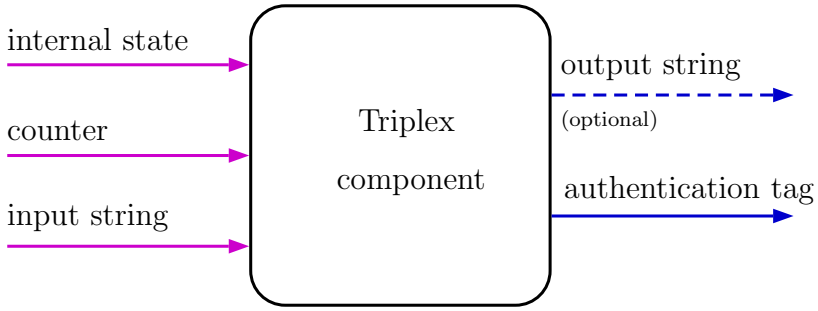


Figure 1.1: A general scheme of the triplex component

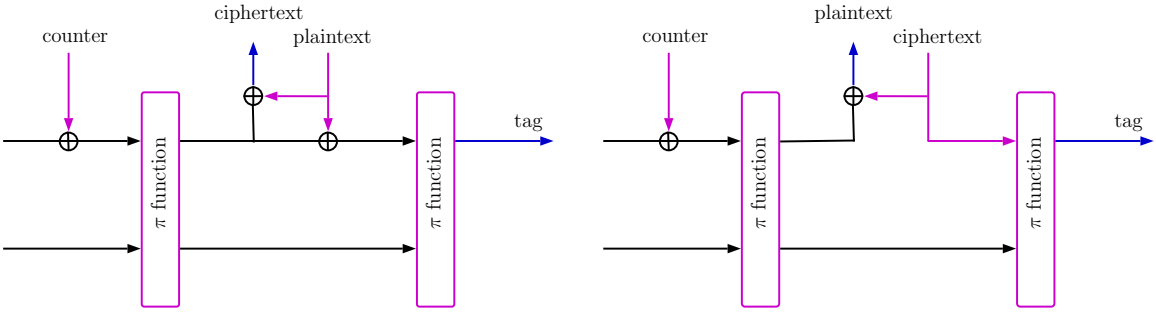
- **Flexible block size**. The flexibility of block sizes allows secure authenticated encryption of data in rest (see Section 4.2.2) and secure incremental mode of operation (see Section 4.2.3).

1.2.1 Authenticated encryption

The encryption/authentication procedure of π-Cipher accepts key K with fixed-length of *klen* bytes, message M with *mlen* bytes and associated data AD with *adlen* bytes. The cipher uses a fixed-length public message number PMN and secret message number SMN. The output of the encryption/authentication procedure is a ciphertext C with *clen* bytes and a tag T with fixed-length of *tlen* bytes. The length *clen* of the ciphertext C is a sum of the byte length of the message, the authentication tag and the encrypted secret message number. The decryption/verification procedure accepts key K, associated data AD, ciphertext C, public message number PMN and tag T, and returns the decrypted pair (SMN, M) if the tag has been verified or $\perp$ otherwise.

The main building element in the operations of encryption/authentication and decryption/verification is our new construction related to the duplex sponge, called triplex component. It uses the permutation function π twice, it injects a counter into the internal state and digests an input string. The triplex component always outputs a tag. Optionally after the first call of the permutation function it can output a string (that can be a ciphertext block or a message block). The general scheme of the triplex component is presented in Figure 1.1.

Because of the differences in the encryption/authentication and decryption/verification procedures, there are two different variants of the triplex component. We call them *e-*



(a) e-triplex (b) d-triplex

Figure 1.2: Triplex component

triplex (for the phase of encryption) and *d-triplex* (for the phase of decryption). The only difference in these two components is how the input string is treated after the first call of the permutation function. In the first one, the input string (plaintext) is XORed with the current internal state and the result proceeds to the second invocation of the permutation function π. In the d-triplex component, the input string (ciphertext) is directly injected as a part of the internal state before the second invocation of the permutation function π. The graphical representation of the e-triplex and d-triplex components is given in Figure 1.2.

The encryption/authentication operation of π-Cipher can be described in four phases:

1. **Initialization**. In this phase we append a single "1" (i.e., `0x01` in hexadecimal byte representation) and the smallest number of 0's (i.e., `0x00` in hexadecimal byte representation) to the concatenated value of the key and the public message number. The length of the result should be less than or equal to the length of the internal state of the permutation function π. In other words the internal state is initialized with $K||PMN||10^*$, where $|K||PMN||10^*| = b$. Then the internal state is updated by applying the permutation function π. Because, π-Cipher works in parallel mode, it has an initial value for the state for all of the parallel parts. We call it *Common Internal State (CIS)*. The CIS is initialized as:

$$CIS \leftarrow \pi(K||PMN||10^*).$$

The next part of this phase is initializing the counter ctr.

Since $CIS = CIS_{rate}||||CIS_{capacity}$ we initialize the *ctr* as the first 64 bits (little endian representation) of the $CIS_{capacity}$.

The graphical representation of the Initialization phase is given in Figure 1.3.

2. **Processing the associated data**.
The associated data $AD = AD_1||\ldots||AD_i||\ldots||AD_a$ is processed block by block in parallel using e–triplex components. The padding rule for the associated data AD is the following:

$$Pad(AD) = AD_1||AD_2||\ldots||AD_a||10^*$$

where 1 represents the byte `0x01`, and 0 represents the byte `0x00`. Note that if the associated data AD has length that is a multiple of the *rate*, then the number of processed blocks of AD is increased by one, and thus $a \leftarrow a + 1$.

To every block AD_i we associate a unique counter calculated as a sum of the initial counter *ctr* and the ordinal number of the processed block i. The input to every e–triplex component is CIS, $ctr + i$ and AD_i, and the output is an intermediate tag t'_i.

The tag T' for the associated data is computed as a component-wise addition $\boxplus_d$ of a vectors $t'_i \in (\mathbb{Z}_{2^\omega})^d$ of dimension d, where d is the number of ω-bit words in the *rate* part. In other words,

$$T' = \boxplus_{d,\,i=1}^{a} t'_i = t'_1 \boxplus_d t'_2 \boxplus_d \ldots \boxplus_d t'_a$$

where $t'_i = (t'_{i1}, t'_{i2}, \ldots, t'_{id})$ is a d-dimensional vector of ω-bit words. In the case where $N = 4$, the dimension of these vectors is 8 ($d = 8$).

The final part of this phase is to update the value of the *Common Internal State* CIS. Updating the CIS is done by xoring it with the tag T' and applying the π permutation function i.e. by the following expression:

$$CIS \leftarrow \pi(CIS_{rate} \bigoplus T' \;\; |||| \;\; CIS_{capacity})$$

This step is described graphically in Figure 1.4.

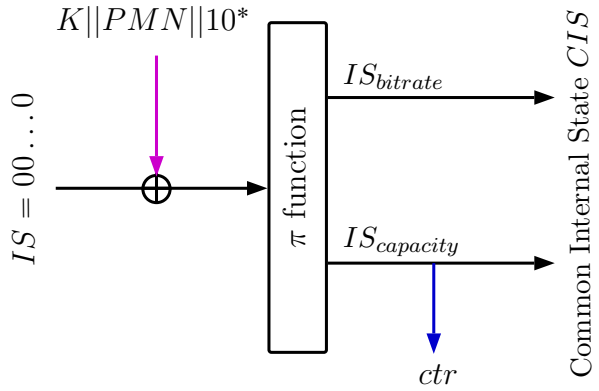


Figure 1.3: Initialization step

3. **Processing the secret message number**. This phase is omitted if the length of the secret message number SMN is 0 (it is the empty string). If SMN is not the empty string, then the first step in this phase is a call to the e-triplex component. The input is the following triplet: $(CIS, ctr + a + 1, SMN)$, and the output is the following pair: (C_0, t_0). The second step of this phase is updating the CIS (for free) which becomes the value of the current internal state after the processing of SMN. Formally, the updating part can be described by the following two expressions:

$$IS \leftarrow \pi(CIS_{rate} \bigoplus (ctr + a + 1) \ \ |||| \ \ CIS_{capacity}),$$
$$CIS \leftarrow \pi(IS_{rate} \bigoplus SMN \ \ |||| \ \ IS_{capacity})$$

The tag produced from this phase is

$$T'' = T' \boxplus_d t_0.$$

This phase is described graphically in Figure 1.5.

4. **Processing the message**. The message $M = M_1||\ldots||M_j||\ldots||M_m$ is processed block by block in parallel by e–triplex components. The padding rule for the message M is the following:

$$Pad(M) = M_1||M_2||\ldots||M_m||10^*$$

where 1 represents the byte `0x01`, and 0 represents the byte `0x00`. Note that if the

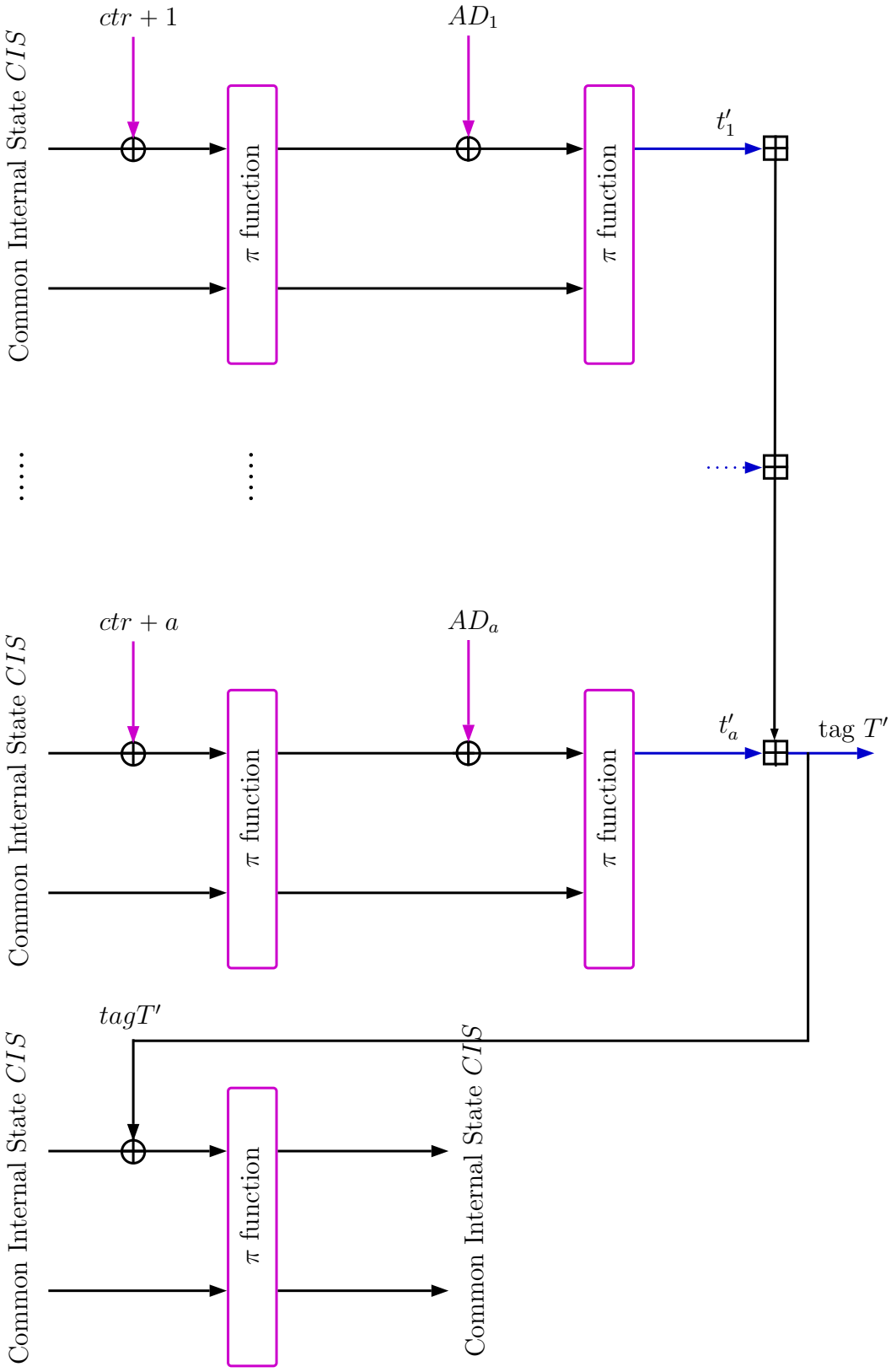


Figure 1.4: Processing the associated data AD with a blocks in parallel

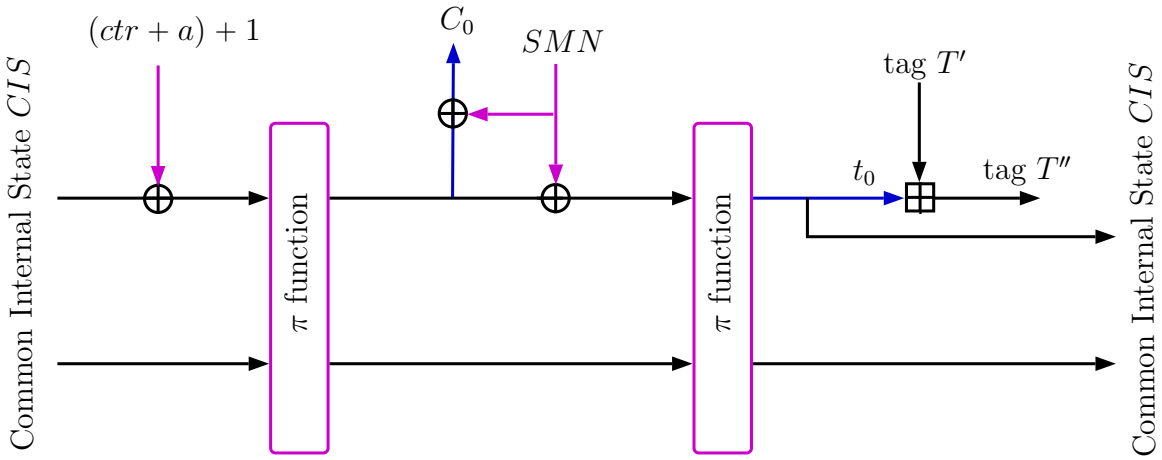


Figure 1.5: Processing the secret message number SMN

message M has length that is a multiple of the *rate*, then the number of processed blocks of M is increased by one, and thus $m \leftarrow m + 1$.

To every block M_j we associate a unique block counter. It can be calculated as:

$$ctr \leftarrow \begin{cases} ctr + a + j & \text{if } |SMN| = 0, \\ ctr + a + 1 + j & \text{if } |SMN| = r, \end{cases}$$

where j is the ordinal number of the processed block in the message, and $0 < j \leqslant m$. The input to every e–triplex component is CIS, block ctr and M_j, and the output is a pair (C_j, t_j). By definition we put that the length of the final ciphertext block C_m is the same as the length of the un-padded last plaintext block M_m i.e., $|C_m| = |M_m|$.

The final tag T is obtained as a $\boxplus_d$ sum of all block tags t_j and the previously obtained tag T''.

$$T = T'' \boxplus_d t_1 \boxplus_d \ldots \boxplus_d t_j \boxplus_d \ldots \boxplus_d t_m.$$

where $t_j = (t_{j1}, t_{j2}, \ldots, t_{jd})$ is a d-dimensional vector of ω-bit words. In this case where $N = 4$, the dimension $d = 8$.

This phase is described graphically in Figure 1.6.

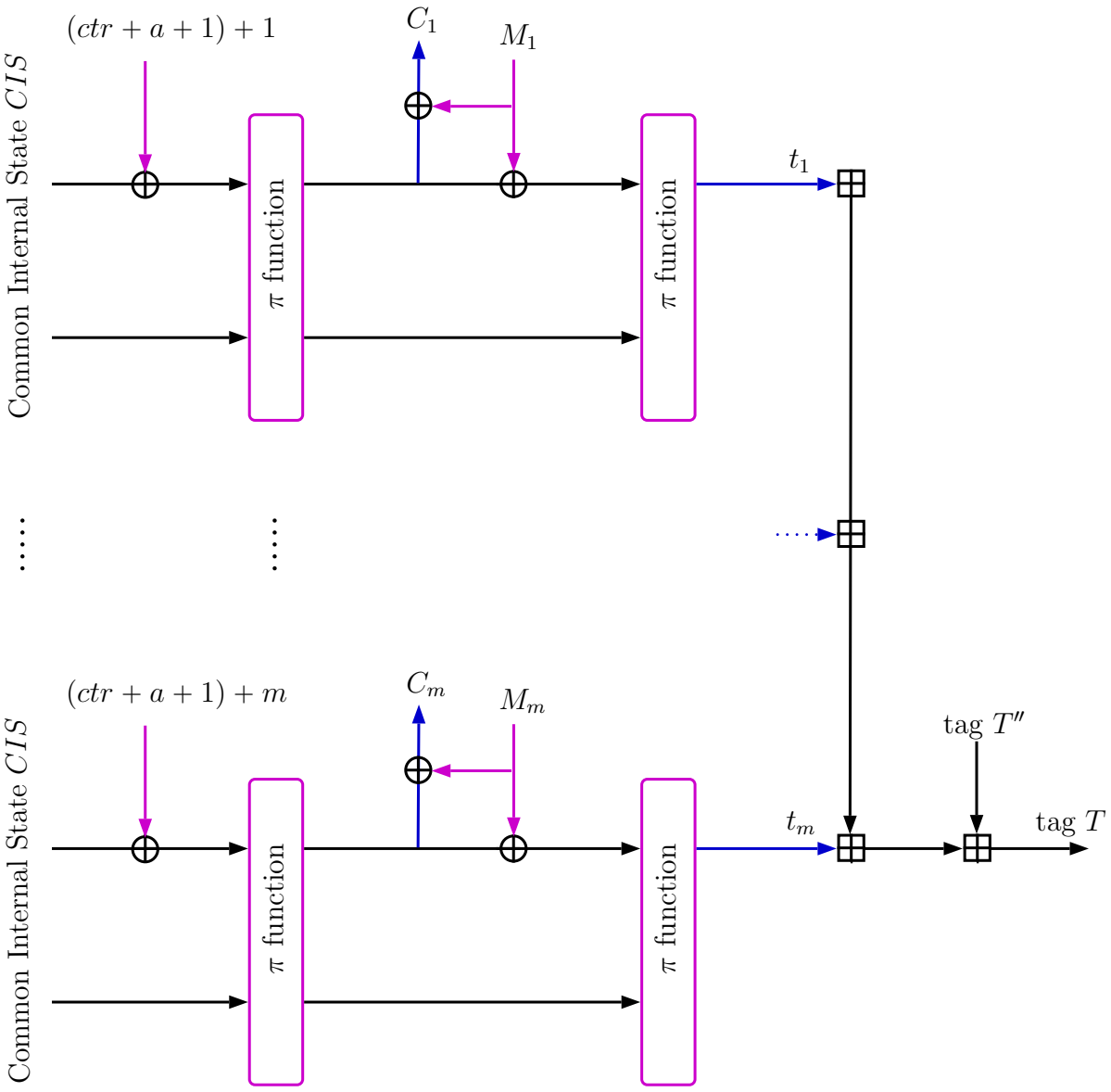


Figure 1.6: Processing the message M with m blocks in parallel

The output of the encryption/authentication procedure is the ciphertext

$$C = C_0||C_1||\ldots||C_m||T.$$

The encryption/authentication part is defined in Algorithm 1.

1.2.2 Decryption and verification

The decryption/verification procedure is defined correspondingly. There are four phases and the only difference is in the last two (so the Initialization phase and Processing the associated data phase are completely the same as in the encryption/authentication procedure).

The decryption of SMN is performed in the phase of Processing the secret message number. Thus, instead of using an e-triplex component, we use a d-triplex component. The input parameters are: CIS, incremented counter $ctr + a + 1$ and the ciphertext block C_0. The output is a pair (SMN, t_0). The tag is processed in the same way as in the encryption/authentication procedure. In this phase decrypted value of C_0 belongs to SMN.

For the decryption of the rest of the ciphertext we continue to use a d-triplex component (instead of e-triplex). The output is now a decrypted message block and a tag value.

At the end, the supplied tag value T is compared to the one computed by the algorithm. Only if the tag is correct, the decrypted message is returned.

The decryption/verification part is defined in Algorithm 2.

1.3 The π-function

The core part of every sponge construction is the permutation function, and the whole security of the primitive relies on it. The design goal for our sponge construction was to obtain a strong permutation, which for different values of the parameter ω (the bit size of the words) provides different features, i.e. to be very efficient when $\omega = 64$ and lightweight when $\omega = 16$.

π-Cipher has ARX based permutation function which we denote as π function. It uses similar operations as the operations used in the hash function Edon-R [7] but instead

Algorithm 1: Encryption $\mathcal{E}_K(PMN, AD, SMN, M)$

1 Initialization: $K||PMN||10^*$ where $|K||PMN||10^*| = b$
2 $CIS \longleftarrow \pi(K||PMN||10^*)$
3 $ctr \longleftarrow [CIS_{capacity}]^{64}$
4 $AD = AD_1||AD_2||\ldots||AD_{a-1}||AD_a$
5 $AD \longleftarrow Pad(AD)$
6 $M = M_1||M_2||\ldots||M_{m-1}||M_m$, and $m_{old} \longleftarrow m$, $lastBlockLength \longleftarrow |M_m|$
7 $M \longleftarrow Pad(M)$
8 **for** $i = 1$ **to** a **do**
8.1 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + i)||||CIS_{capacity})$
8.2 $IS \longleftarrow \pi(IS_{rate} \oplus AD_i||||IS_{capacity})$
8.3 $t'_i \longleftarrow IS_{rate}$
8.4 $T' \longleftarrow T' \boxplus_d t'_i$
9 $ctr \longleftarrow ctr + a$
10 $CIS \longleftarrow \pi(CIS_{rate} \oplus T'||||CIS_{capacity})$
11 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + 1)||||CIS_{capacity})$
12 $C_0 \longleftarrow IS_{rate} \oplus SMN$
13 $IS \longleftarrow \pi(IS_{rate} \oplus SMN||||IS_{capacity})$
14 $t_0 \longleftarrow IS_{rate}$, $T'' \longleftarrow T' \boxplus_d t_0$, $T \longleftarrow T''$
15 $CIS \longleftarrow IS$
16 $ctr \longleftarrow ctr + 1$
17 **for** $i = 1$ **to** m **do**
17.1 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + i)||||CIS_{capacity})$
17.2 $C_i \longleftarrow IS_{rate} \oplus M_i$
17.3 $IS \longleftarrow \pi(IS_{rate} \oplus M_i||||IS_{capacity})$
17.4 $t_i \longleftarrow IS_{rate}$
17.5 $T \longleftarrow T \boxplus_d t_i$
18 $C \longleftarrow C_0||C_1||\ldots||C_{m_{old}}||T$, where $|C_{m_{old}}| = lastBlockLength$
19 **return** C

Algorithm for padding: $Pad(S)$

1 $S = S_1||S_2||\ldots||S_l$
2 **if** $|S_l| < r$ **then**
3 $S_l \longleftarrow S_l||10^*$, where $|S_l| = r$
4 **else**
5 $S_{l+1} \longleftarrow 10^*$, where $|S_{l+1}| = r$, and $l \longleftarrow l + 1$
6 **return** S

Algorithm 2: Decryption $\mathcal{D}_K(PMN, AD, C, T)$

1 Initialization: $K||PMN||10^*$ where $|K||PMN||10^*| = b$
2 $CIS \longleftarrow \pi(K||PMN||10^*)$
3 $ctr = [CIS_{capacity}]^{64}$
4 $AD = AD_1||AD_2||\ldots||AD_{a-1}||AD_a$
5 $AD = Pad(AD)$
7 $C = C_0||C_1||\ldots||C_m$, where $|C_i| = r$ for $0 \leqslant i < m$ and $lastBlockLength \longleftarrow |C_m|$
8 **for** $i = 1$ **to** a **do**
8.1 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + i)||||CIS_{capacity})$
8.2 $IS \longleftarrow \pi(IS_{rate} \oplus AD_i||||IS_{capacity})$
8.3 $t'_i \longleftarrow IS_{rate}$
8.4 $T' \longleftarrow T' \boxplus_d t'_i$
9 $ctr \longleftarrow ctr + a$
10 $CIS \longleftarrow \pi(CIS_{rate} \oplus T'||||CIS_{capacity})$
11 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + 1)||||CIS_{capacity})$
12 $SMN \longleftarrow IS_{rate} \oplus C_0$
13 $IS \longleftarrow \pi(C_0||||IS_{capacity})$
14 $t_0 \longleftarrow IS_{rate}$, $T'' \longleftarrow T' \boxplus_d t_0$, $\mathtt{T} \longleftarrow T''$
15 $CIS \longleftarrow IS$
16 $ctr \longleftarrow ctr + 1$
17 **for** $i = 1$ **to** $m - 1$ **do**
17.1 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + i)||||CIS_{capacity})$
17.2 $M_i \longleftarrow IS_{rate} \oplus C_i$
17.3 $IS \longleftarrow \pi(C_i||||IS_{capacity})$
17.4 $t_i \longleftarrow IS_{rate}$
17.5 $\mathtt{T} \longleftarrow \mathtt{T} \boxplus_d t_i$
18 **if** $(|C_m| < r)$ **then**
19 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + m)||||CIS_{capacity})$
20 $M_m \longleftarrow IS_{rate} \oplus C_m$
21 $IS_{rate} \longleftarrow IS_{rate} \oplus 0^{lastBlockLength}10^*$
22 $IS \longleftarrow \pi(C_m||[IS_{rate}]_{(r-m)}||||IS_{capacity})$
23 $t_m \longleftarrow IS_{rate}$
24 $\mathtt{T} \longleftarrow \mathtt{T} \boxplus_d t_m$
25 **else**
26 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + m)||||CIS_{capacity})$
27 $M_m \longleftarrow IS_{rate} \oplus C_m$
28 $IS \longleftarrow \pi(C_m||||IS_{capacity})$
29 $t_m \longleftarrow IS_{rate}$
30 $\mathtt{T} \longleftarrow \mathtt{T} \boxplus_d t_m$
31 $IS \longleftarrow \pi(CIS_{rate} \oplus (ctr + m + 1)||||CIS_{capacity})$
32 $IS \longleftarrow \pi(IS_{rate} \oplus 10^*||||IS_{capacity})$
33 $t_{m+1} \longleftarrow IS_{rate}$
34 $\mathtt{T} \longleftarrow \mathtt{T} \boxplus_d t_{m+1}$
35 $M \longleftarrow M_1||M_2||\ldots||M_m$
31 **if** $T \neq \mathtt{T}$ **then**
32 **return** $\perp$
33 **else return** (SMN, M)

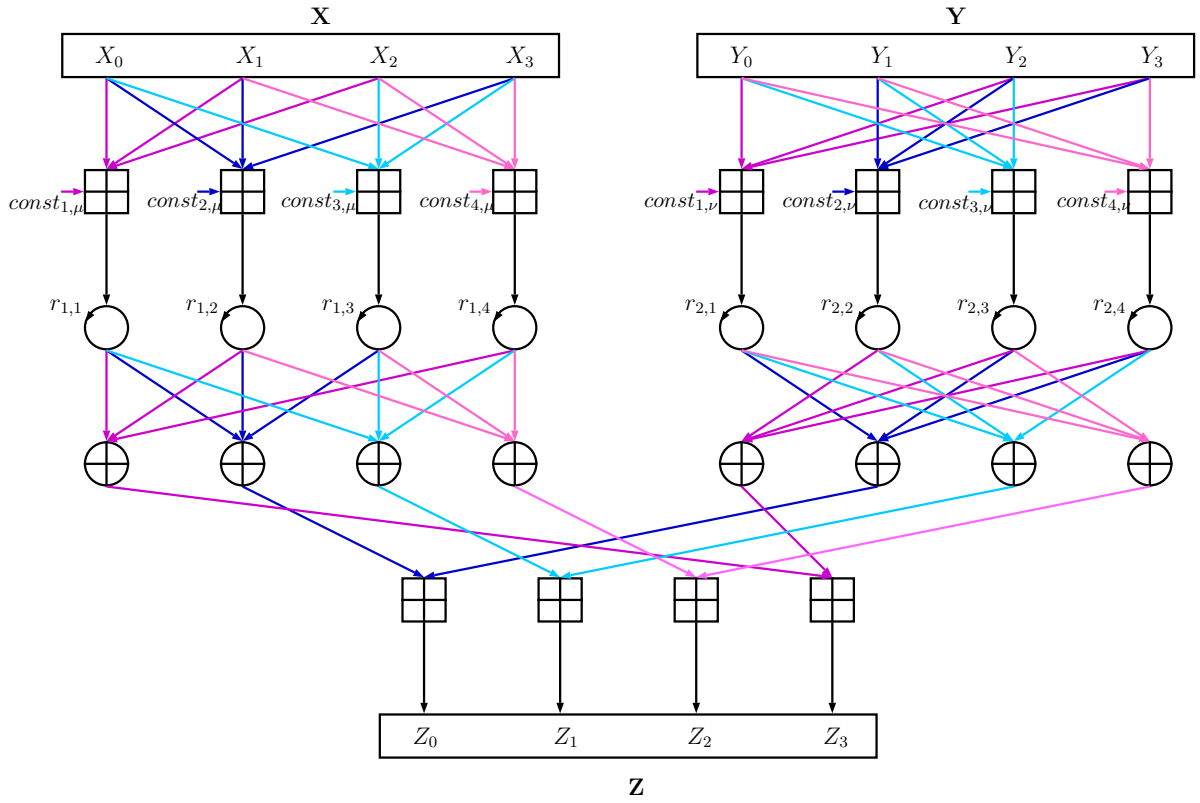


Figure 1.7: Graphical representation of the ARX operation $*$.

of using 8–tuples here we use 4–tuples. The permutation operates on a b bits state and updates the internal state through a sequence of R successive transformations - rounds. The state IS can be represented as a list of N 4-tuples, each of length ω-bits, where $b = N \times 4 \times \omega$, i.e.,

$$IS = (\underbrace{(IS_{11}, IS_{12}, IS_{13}, IS_{14})}_{I_1}, \underbrace{(IS_{21}, IS_{22}, IS_{23}, IS_{24})}_{I_2}, \ldots, \underbrace{(IS_{N1}, IS_{N2}, IS_{N3}, IS_{N4})}_{I_N}). \tag{1.1}$$

The general permutation function π consists of three main transformations $\mu, \nu, \sigma : \mathbb{Z}_{2^\omega}^4 \rightarrow \mathbb{Z}_{2^\omega}^4$, where $\mathbb{Z}_{2^\omega}$ is the set of all integers between 0 and $2^\omega - 1$. These transformations do the work of diffusion and nonlinear mixing of the input.

The following operations are applied:

- Addition + modulo 2^ω;

Table 1.2: The rotation vectors used in μ and ν.

ω	$\rho_{1,\omega}$	$\rho_{2,\omega}$
16	(1, 4, 9, 11)	(2, 5, 7, 13)
32	(5, 11, 17, 23)	(3, 10, 19, 29)
64	(7, 19, 31, 53)	(11, 23, 37, 59)

- Rotate left (circular left shift) operation, $ROTL^{\rho}(X)$, where X is a ω-bit word and ρ is an integer with $0 \leqslant \rho < \omega$;
- Bitwise XOR operation $\oplus$ on ω–bit words.

Let $\mathbf{X} = (X_0, X_1, X_2, X_3)$, $\mathbf{Y} = (Y_0, Y_1, Y_2, Y_3)$ and $\mathbf{Z} = (Z_0, Z_1, Z_2, Z_3)$ be three 4-tuples of ω–bit words.

Further, let us denote by $*$ the following operation:

$$\mathbf{Z} = \mathbf{X} * \mathbf{Y} \equiv \sigma(\mu(\mathbf{X}) \boxplus_4 \nu(\mathbf{Y})) \tag{1.2}$$

where $\boxplus_4$ is the component-wise addition of two 4-dimensional vectors in $\left(\mathbb{Z}_{2^\omega}\right)^4$.

An algorithmic definition of the $*$ operation over two 4–dimensional vectors $\mathbf{X}$ and $\mathbf{Y}$ for different word sizes is given in Table 1.9, Table 1.10 and Table 1.11.

Also a graphical representation of the $*$ operation is given in Figure 1.7.

The following is a formalization of the $*$ operation (adopted from [7]).

The left rotation of a ω–bit word X by ρ positions denoted by $ROTL^{\rho}(X)$, can be expressed as a linear matrix–vector multiplication over the ring $(\mathbb{Z}_2, +, \cdot)$ i.e. $ROTL^{\rho}(X) = \mathbf{E}^{\rho} \cdot X$ where $\mathbf{E}^{\rho} \in \mathbb{Z}_2^{\omega} \times \mathbb{Z}_2^{\omega}$ is a matrix obtained from the identity matrix by rotating its columns by r positions in the direction top to bottom. Further on, if we have a vector $X \in \left(\mathbb{Z}_{2^\omega}\right)^4$ represented as $\mathbf{X} = (X_0, X_1, X_2, X_3)$ and we want to rotate all X_i by ρ_i $(0 \leqslant i < 4)$ positions to the left, then we denote the operation by $ROTL^{\rho}(\mathbf{X})$, where $\rho = (\rho_0, \ldots, \rho_3) \in \{0, 1, \ldots, \omega - 1\}^4$ is a rotation vector. The operation $ROTL^{\rho}(\mathbf{X})$ can also be represented as a linear matrix–vector multiplication over the ring $(\mathbb{Z}_2, +, \cdot)$ i.e.

Table 1.3: The matrices $\widehat{\mathbb{A}}_1$, $\mathbb{A}_2$, $\widehat{\mathbb{A}}_3$ and $\mathbb{A}_4$.

$\widehat{\mathbb{A}}_1$	$\mathbb{A}_2$	$\widehat{\mathbb{A}}_3$	$\mathbb{A}_4$
$\begin{pmatrix} const_{1,\mu\omega} \\ const_{2,\mu\omega} \\ const_{3,\mu\omega} \\ const_{4,\mu\omega} \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix}$	$\begin{pmatrix} const_{1,\nu\omega} \\ const_{2,\nu\omega} \\ const_{3,\nu\omega} \\ const_{4,\nu\omega} \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{pmatrix}$

$ROTL^{\rho}(\mathbf{X}) = \mathbf{D}^{\rho} \cdot \mathbf{X}$ where $\mathbf{D}^{\rho} \in \mathbb{Z}_2^{4\omega} \times \mathbb{Z}_2^{4\omega}$,

$$\mathbf{D}^{\mathbf{r}} = \begin{pmatrix} \mathbf{E}^{\rho_0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}^{\rho_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E}^{\rho_2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}^{\rho_3} \end{pmatrix},$$

and the submatrices $\mathbf{E}^{\rho_i} \in \mathbb{Z}_2^{\omega} \times \mathbb{Z}_2^{\omega}$, $0 \leqslant i < 4$ are obtained from the identity matrix by rotating its columns by ρ_i positions in the direction top to bottom, and the submatrices $\mathbf{0} \in \mathbb{Z}_2^{\omega} \times \mathbb{Z}_2^{\omega}$ are the zero matrices.

Furthermore, we use the following notations:

- $\widehat{\mathbb{A}}_1, \widehat{\mathbb{A}}_3 : \mathbb{Z}_{2^\omega}^4 \to \mathbb{Z}_{2^\omega}^4$ are two bijective transformations in $\mathbb{Z}_{2^\omega}^4$ over the ring $(\mathbb{Z}_{2^\omega}, +, \cdot)$ where $\omega = 8, 16, 32, 64$. The mappings $\widehat{\mathbb{A}}_i, i = 1, 3$ can be described as:

  $$\widehat{\mathbb{A}}_i(\mathbf{X}) = \mathbf{C}_i + \mathbb{A}_i \cdot \mathbf{X},$$

  where $\mathbf{C}_i \in \mathbb{Z}_{2^\omega}^4, i = 1, 2$ are two constant vectors and $\mathbb{A}_1$ and $\mathbb{A}_3$ are two 4×4 invertible matrices over the ring $(\mathbb{Z}_{2^\omega}, +, \cdot)$. Also these matrices are MDS (maximum distance separable) matrices which redound to have maximal diffusion of the bits. All elements in these two matrices are either 0 or 1, since we want to avoid the operations of multiplication (as more costly microprocessor operations) in the ring $(\mathbb{Z}_{2^\omega}, +, \cdot)$, and stay only with operations of addition.

- $\mathbb{A}_2, \mathbb{A}_4 : \mathbb{Z}_{2^\omega}^4 \to \mathbb{Z}_{2^\omega}^4$ are two linear bijective transformations that are described by two invertible matrices (we use the same notation: $\mathbb{A}_2, \mathbb{A}_4$) of order $q \times q$ over the ring $(\mathbb{Z}_2, +, \cdot)$ $(q = 4\omega)$. Since we want to apply XOR operations on ω–bit registers,

the matrices $\mathbb{A}_2$ and $\mathbb{A}_4$ will be of the form

$$\begin{pmatrix} \mathbb{B}_{1,1} & \mathbb{B}_{1,2} & \mathbb{B}_{1,3} & \mathbb{B}_{1,4} \\ \mathbb{B}_{2,1} & \mathbb{B}_{2,2} & \mathbb{B}_{2,3} & \mathbb{B}_{2,4} \\ \mathbb{B}_{3,1} & \mathbb{B}_{3,2} & \mathbb{B}_{3,3} & \mathbb{B}_{3,4} \\ \mathbb{B}_{4,1} & \mathbb{B}_{4,2} & \mathbb{B}_{4,3} & \mathbb{B}_{4,4} \end{pmatrix},$$

where $\mathbb{B}_{i,j} \in \mathbb{Z}_2^\omega \times \mathbb{Z}_2^\omega$, $1 \leqslant i, j \leqslant 4$ are either the identity matrix or the zero matrix i.e. $\mathbb{B}_{i,j} \in \{\mathbf{0}, \mathbf{1}\}$.

Now we give the formal definitions for the permutations: σ, μ and ν.

Definition 1. *The transformation* $\sigma : \mathbb{Z}_{2^\omega}^4 \rightarrow \mathbb{Z}_{2^\omega}^4$ *is defined as:*

$$\sigma(X_0, X_1, X_2, X_3) = (X_3, X_0, X_1, X_2)$$

Lemma 1. *The transformation* σ *is a permutation.* □

Definition 2. *The transformations* $\mu : \mathbb{Z}_{2^\omega}^4 \rightarrow \mathbb{Z}_{2^\omega}^4$ *and* $\nu : \mathbb{Z}_{2^\omega}^4 \rightarrow \mathbb{Z}_{2^\omega}^4$ *are defined as:*

$$\mu \equiv \widehat{\mathbb{A}}_1 \circ ROTL^{\rho_{1,\omega}} \circ \mathbb{A}_2$$
$$\nu \equiv \widehat{\mathbb{A}}_3 \circ ROTL^{\rho_{2,\omega}} \circ \mathbb{A}_4$$

where the rotation vectors $\rho_{i,\omega}$, $i = 1, 2$, $\omega = 16, 32, 64$ *are given in Table 1.2, and the matrices* $\widehat{\mathbb{A}}_1$, $\mathbb{A}_2$, $\widehat{\mathbb{A}}_3$ *and* $\mathbb{A}_4$ *are given in Table 1.3. In Table 1.3, the symbols* $\mathbf{1}, \mathbf{0} \in \mathbb{Z}_2^\omega \times \mathbb{Z}_2^\omega$ *denote the identity matrix and the zero matrix, and the constants* $const_{i,\omega}$, $i = 1, 2$, $\omega = 16, 32, 64$ *are given (in hexadecimal notation) in Table 1.4 and Table 1.5. The rationale for choosing these constants is given in Section 5.2.*

Table 1.4: List of constants used in μ transformation.

$const_{i,\mu\omega}$	$\mu 16$	$\mu 32$	$\mu 64$
1	`0xF0E8`	`0xF0E8E4E2`	`0xF0E8E4E2E1D8D4D2`
2	`0xE4E2`	`0xE1D8D4D2`	`0xD1CCCAC9C6C5C3B8`
3	`0xE1D8`	`0xD1CCCAC9`	`0xB4B2B1ACAAA9A6A5`
4	`0xD4D2`	`0xC6C5C3B8`	`0xA39C9A999695938E`

Table 1.5: List of constants used in ν transformation.

$const_{i,\nu\omega}$	$\nu 16$	$\nu 32$	$\nu 64$
1	0xD1CC	0xB4B2B1AC	0x8D8B87787472716C
2	0xCAC9	0xAAA9A6A5	0x6A696665635C5A59
3	0xC6C5	0xA39C9A99	0x5655534E4D4B473C
4	0xC3B8	0x9695938E	0x3A393635332E2D2B

Lemma 2. *The transformations μ and ν are permutations of $\mathbb{Z}_{2^\omega}, \omega = 16, 32, 64$.*

Proof. The proof follows immediately from the fact that all transformations $\mathbb{A}_i$, $i = 1, 2, 3, 4$ and $ROTL^{\rho_{i,\omega}}$, $i = 1, 2, \omega = 16, 32, 64$ are expressed by invertible matrices over the rings $(\mathbb{Z}_{2^\omega}, +, \dots), \omega = 16, 32, 64$ or over the ring $(\mathbb{Z}_2, +, \dots)$. □

Theorem 1. *The operation $* : (\mathbb{Z}_{2^\omega}^4)^2 \to \mathbb{Z}_{2^\omega}^4$ defined as:*

$$\mathbf{X} * \mathbf{Y} = \sigma(\mu(\mathbf{X}) \ \boxplus_4 \ \nu(\mathbf{Y}))$$

is a permutation. □

Let us recall equation (1.1) where the internal state is presented as $IS = (I_1, I_2, \dots, I_N)$. One round of the π function consists of two consecutive transformations E_1 and E_2 defined as follows.

Definition 3. *The function $E_1 : (\mathbb{Z}_{2^\omega}^4)^{N+1} \to (\mathbb{Z}_{2^\omega}^4)^N$ used in the π function is defined as:*

$$\begin{aligned} E_1(C, I_1, \dots, I_N) &= (J_1, \dots, J_N), \quad \textit{where} \\ J_1 &= C * I_1, \\ J_i &= J_{i-1} * I_i, \ i = 2, \dots, N \end{aligned} \tag{1.3}$$

where C is a 4-tuple of ω-bit constant values.

Definition 4. *The function $E_2 : (\mathbb{Z}_{2^\omega}^4)^{N+1} \to (\mathbb{Z}_{2^\omega}^4)^N$ used in π function is defined as:*

$$\begin{aligned} E_2(C, I_1, \dots, I_N) &= (J_1, \dots, J_N), \quad \textit{where} \\ J_N &= I_N * C, \\ J_{N-i} &= I_{N-i} * J_{N-i+1}, \ i = 1, \dots, N-1 \end{aligned} \tag{1.4}$$

where C is a 4-tuple of ω-bit constant values.

Finally, one round of the π function is defined as:

$$\pi(I_1, \ldots, I_N) = E_2(C2, E_1(C1, I_1, \ldots, I_N)) \tag{1.5}$$

One round of the cipher is graphically described in Figure 1.8. In the figure, the diagonal arrows can be interpreted as $*$ operations between the source and destination, and the vertical or horizontal arrows as equality signs " = ".

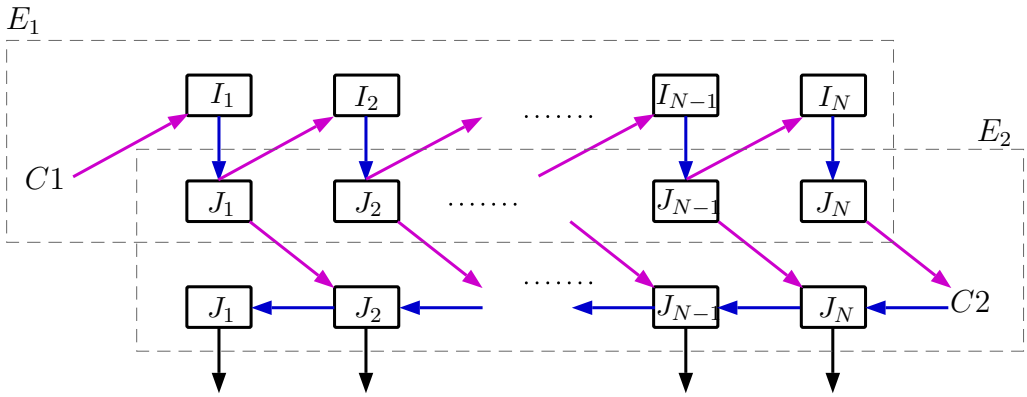


Figure 1.8: One round of π-Cipher

The number of rounds R is a tweakable parameter. For π-Cipher v2.0 we recommend $R = 3$. The complete formula for the π function with $R = 3$ is the following:

$$\pi(I_1, \ldots, I_N) = E_2(C6, E_1(C5, E_2(C4, E_1(C3, E_2(C2, E_1(C1, I_1, \ldots, I_N))))))$$

The constants $C1, C2, \ldots, C6$ are generated in the same way as the constants of the $*$ operation and their values (in hexadecimal notation) for different word sizes are given in Table 1.6, Table 1.7 and Table 1.8.

Table 1.6: Round constants for π16-Cipher

$C1 = \{$0xB4B2, 0xB1AC, 0xAAA9, 0xA6A5$\}$
$C2 = \{$0xA39C, 0x9A99, 0x9695, 0x938E$\}$
$C3 = \{$0x8D8B, 0x8778, 0x7472, 0x716C$\}$
$C4 = \{$0x6A69, 0x6665, 0x635C, 0x5A59$\}$
$C5 = \{$0x5655, 0x534E, 0x4D4B, 0x473C$\}$
$C6 = \{$0x3A39, 0x3635, 0x332E, 0x2D2B$\}$

Table 1.7: Round constants for π32-Cipher

$C1 = \{$0x8D8B8778, 0x7472716C, 0x6A696665, 0x635C5A59$\}$
$C2 = \{$0x5655534E, 0x4D4B473C, 0x3A393635, 0x332E2D2B$\}$
$C3 = \{$0x271E1D1B, 0x170FF0E8, 0xE4E2E1D8, 0xD4D2D1CC$\}$
$C4 = \{$0xCAC9C6C5, 0xC3B8B4B2, 0xB1ACAAA9, 0xA6A5A39C$\}$
$C5 = \{$0x9A999695, 0x938E8D8B, 0x87787472, 0x716C6A69$\}$
$C6 = \{$0x6665635C, 0x5A595655, 0x534E4D4B, 0x473C3A39$\}$

Table 1.8: Round constants for π64-Cipher

$C1 = \{$0x271E1D1B170FF0E8, 0xE4E2E1D8D4D2D1CC,
0xCAC9C6C5C3B8B4B2, 0xB1ACAAA9A6A5A39C$\}$
$C2 = \{$0x9A999695938E8D8B, 0x87787472716C6A69,
0x6665635C5A595655, 0x534E4D4B473C3A39$\}$
$C3 = \{$0x3635332E2D2B271E, 0x1D1B170FF0E8E4E2,
0xE1D8D4D2D1CCCAC9, 0xC6C5C3B8B4B2B1AC$\}$
$C4 = \{$0xAAA9A6A5A39C9A99, 0x9695938E8D8B8778,
0x7472716C6A696665, 0x635C5A595655534E$\}$
$C5 = \{$0x4D4B473C3A393635, 0x332E2D2B271E1D1B,
0x170FF0E8E4E2E1D8, 0xD4D2D1CCCAC9C6C5$\}$
$C6 = \{$0xC3B8B4B2B1ACAAA9, 0xA6A5A39C9A999695,
0x938E8D8B87787472, 0x716C6A696665635C$\}$

Table 1.9: An algorithmic description of the ARX operation $*$ for 16–bit words.

$*$ operation for 16–bit words

Input: $\mathbf{X} = (X_0, X_1, X_2, X_3)$ and $\mathbf{Y} = (Y_0, Y_1, Y_2, Y_3)$
where X_i and Y_i are 16–bit variables.
Output: $\mathbf{Z} = (Z_0, Z_1, Z_2, Z_3)$ where Z_i are 16–bit variables.
Temporary 16–bit variables: $T_0, \ldots, T_{11}$.

μ–transformation for X:

1.
$$\begin{array}{lcl} T_0 & \leftarrow & ROTL^{1}(\texttt{0xF0E8} + X_0 + X_1 + X_2); \\ T_1 & \leftarrow & ROTL^{4}(\texttt{0xE4E2} + X_0 + X_1 + X_3); \\ T_2 & \leftarrow & ROTL^{9}(\texttt{0xE1D8} + X_0 + X_2 + X_3); \\ T_3 & \leftarrow & ROTL^{11}(\texttt{0xD4D2} + X_1 + X_2 + X_3); \end{array}$$

2.
$$\begin{array}{lcl} T_4 & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_5 & \leftarrow & T_0 \oplus T_1 \oplus T_2; \\ T_6 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_7 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \end{array}$$

ν–transformation for Y:

1.
$$\begin{array}{lcl} T_0 & \leftarrow & ROTL^{2}(\texttt{0xD1CC} + Y_0 + Y_2 + Y_3); \\ T_1 & \leftarrow & ROTL^{5}(\texttt{0xCAC9} + Y_1 + Y_2 + Y_3); \\ T_2 & \leftarrow & ROTL^{7}(\texttt{0xC6C5} + Y_0 + Y_1 + Y_2); \\ T_3 & \leftarrow & ROTL^{13}(\texttt{0xC3B8} + Y_0 + Y_1 + Y_3); \end{array}$$

2.
$$\begin{array}{lcl} T_8 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_9 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \\ T_{10} & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_{11} & \leftarrow & T_0 \oplus T_1 \oplus T_2; \end{array}$$

σ–transformation for both $\mu(X)$ and $\nu(Y)$:

1.
$$\begin{array}{lcl} Z_3 & \leftarrow & T_4 + T_8; \\ Z_0 & \leftarrow & T_5 + T_9; \\ Z_1 & \leftarrow & T_6 + T_{10}; \\ Z_2 & \leftarrow & T_7 + T_{11}; \end{array}$$

Table 1.10: An algorithmic description of the ARX operation $*$ for 32–bit words.

$*$ operation for 32–bit words
Input: $\mathbf{X} = (X_0, X_1, X_2, X_3)$ and $\mathbf{Y} = (Y_0, Y_1, Y_2, Y_3)$ where X_i and Y_i are 32–bit variables. **Output:** $\mathbf{Z} = (Z_0, Z_1, Z_2, Z_3)$ where Z_i are 32–bit variables. **Temporary 32–bit variables:** $T_0, \ldots, T_{11}$.
μ–transformation for X: 1. $\begin{array}{lcl} T_0 & \leftarrow & ROTL^5(\texttt{0xF0E8E4E2} + X_0 + X_1 + X_2); \\ T_1 & \leftarrow & ROTL^{11}(\texttt{0xE1D8D4D2} + X_0 + X_1 + X_3); \\ T_2 & \leftarrow & ROTL^{17}(\texttt{0xD1CCCAC9} + X_0 + X_2 + X_3); \\ T_3 & \leftarrow & ROTL^{23}(\texttt{0xC6C5C3B8} + X_1 + X_2 + X_3); \end{array}$ 2. $\begin{array}{lcl} T_4 & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_5 & \leftarrow & T_0 \oplus T_1 \oplus T_2; \\ T_6 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_7 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \end{array}$ ν–transformation for Y: 1. $\begin{array}{lcl} T_0 & \leftarrow & ROTL^3(\texttt{0xB4B2B1AC} + Y_0 + Y_2 + Y_3); \\ T_1 & \leftarrow & ROTL^{10}(\texttt{0xAAA9A6A5} + Y_1 + Y_2 + Y_3); \\ T_2 & \leftarrow & ROTL^{19}(\texttt{0xA39C9A99} + Y_0 + Y_1 + Y_2); \\ T_3 & \leftarrow & ROTL^{29}(\texttt{0x9695938E} + Y_0 + Y_1 + Y_3); \end{array}$ 2. $\begin{array}{lcl} T_8 & \leftarrow & T_1 \oplus T_2 \oplus T_3; \\ T_9 & \leftarrow & T_0 \oplus T_2 \oplus T_3; \\ T_{10} & \leftarrow & T_0 \oplus T_1 \oplus T_3; \\ T_{11} & \leftarrow & T_0 \oplus T_1 \oplus T_2; \end{array}$ σ–transformation for both $\mu(X)$ and $\nu(Y)$: 1. $\begin{array}{lcl} Z_3 & \leftarrow & T_4 + T_8; \\ Z_0 & \leftarrow & T_5 + T_9; \\ Z_1 & \leftarrow & T_6 + T_{10}; \\ Z_2 & \leftarrow & T_7 + T_{11}; \end{array}$

Table 1.11: An algorithmic description of the ARX operation $*$ for 64–bit words.

$*$ **operation for 64–bit words**
Input: $\mathbf{X} = (X_0, X_1, X_2, X_3)$ and $\mathbf{Y} = (Y_0, Y_1, Y_2, Y_3)$ where X_i and Y_i are 64–bit variables. **Output:** $\mathbf{Z} = (Z_0, Z_1, Z_2, Z_3)$ where Z_i are 64–bit variables. **Temporary 64–bit variables:** $T_0, \ldots, T_{11}$.
μ–transformation for X: 1. $T_0 \leftarrow ROTL^{7}(\texttt{0xF0E8E4E2E1D8D4D2} + X_0 + X_1 + X_2)$; $T_1 \leftarrow ROTL^{19}(\texttt{0xD1CCCAC9C6C5C3B8} + X_0 + X_1 + X_3)$; $T_2 \leftarrow ROTL^{31}(\texttt{0xB4B2B1ACAAA9A6A5} + X_0 + X_2 + X_3)$; $T_3 \leftarrow ROTL^{53}(\texttt{0xA39C9A999695938E} + X_1 + X_2 + X_3)$; 2. $T_4 \leftarrow T_0 \oplus T_1 \oplus T_3$; $T_5 \leftarrow T_0 \oplus T_1 \oplus T_2$; $T_6 \leftarrow T_1 \oplus T_2 \oplus T_3$; $T_7 \leftarrow T_0 \oplus T_2 \oplus T_3$;
ν–transformation for Y: 1. $T_0 \leftarrow ROTL^{11}(\texttt{0x8D8B87787472716C} + Y_0 + Y_2 + Y_3)$; $T_1 \leftarrow ROTL^{23}(\texttt{0x6A696665635C5A59} + Y_1 + Y_2 + Y_3)$; $T_2 \leftarrow ROTL^{37}(\texttt{0x5655534E4D4B473C} + Y_0 + Y_1 + Y_2)$; $T_3 \leftarrow ROTL^{59}(\texttt{0x3A393635332E2D2B} + Y_0 + Y_1 + Y_3)$; 2. $T_8 \leftarrow T_1 \oplus T_2 \oplus T_3$; $T_9 \leftarrow T_0 \oplus T_2 \oplus T_3$; $T_{10} \leftarrow T_0 \oplus T_1 \oplus T_3$; $T_{11} \leftarrow T_0 \oplus T_1 \oplus T_2$;
σ–transformation for both $\mu(X)$ and $\nu(Y)$: 1. $Z_3 \leftarrow T_4 + T_8$; $Z_0 \leftarrow T_5 + T_9$; $Z_1 \leftarrow T_6 + T_{10}$; $Z_2 \leftarrow T_7 + T_{11}$;

Chapter 2

Security goals

Table 2.1: A list of security goals for the π-Cipher.

Goal	Pi16Cipher096v2 Bits of security	Pi32Cipher128v2 Bits of security	Pi64Cipher128v2 Bits of security	Pi64Cipher256v2 Bits of security
Confidentiality for the plaintext	96	128	128	256
Confidentiality for the secret message number SMN	96	128	128	256
Integrity for the plaintext	96	128	128	256
Integrity for the associated data	96	128	128	256
Integrity for the public message number PMN	96	128	128	256
Integrity for the secret message number SMN	96	128	128	256
Confidentiality for the plaintext M, when $(K, AD, (PMN, SMN_1))$ and $(K, AD, (PMN, SMN_2))$	96	128	128	256
Integrity for the plaintext M, when $(K, AD, (PMN, SMN_1))$ and $(K, AD, (PMN, SMN_2))$	96	128	128	256

We want to emphasize our position that the security level of 80 bits should be considered as insecure and should be abandoned in future cryptographic designs. This is

due to the recent reported practical speed of many different computing systems (GPUs, supercomputers, FPGAs). That is why our lowest level for security is 96 bits of security.

Users have two options for nonces in π-Cipher. A nonce can be $NONCE = PMN$ or $NONCE = (PMN, SMN)$.

If the legitimate key holder uses the same $NONCE$ to encrypt two different pairs of (plaintext, associated data) (M_1, AD) and (M_2, AD) with the same secret key K then the confidentiality and the integrity of the plaintexts are not preserved in π–Cipher. **Thus, the first six goals in Table 2 are achieved under the assumption that PMN is always different for any two different pairs of (plaintext, associated data) (M_1, AD) and (M_2, AD) with the same secret key K.**

Additionally, π-Cipher offers **an intermediate level of robustness** when a legitimate key holder uses the same secret key K, the same associated data AD, the same public message number PMN but different secret message numbers SMN_1 and SMN_2 for encrypting two different plaintexts M_1 and M_2. In that case confidentiality and integrity of the plaintexts are preserved. **However, in that case the confidentiality of SMN_1 and SMN_2 is not preserved.**

Chapter 3

Security analysis

3.1 Security proof of π-Cipher

The security proof is based on the detailed proof for the sponge based authenticated ciphers given by Bart Mennink at al. in the ASIACRYPT 2014 paper [11].

Unlike other sponge-based schemes, π-Cipher offers parallelism in the encryption part and processing the associated data part. The number of parallel chains depends on the number of data blocks that every part has (a chains in the phase of the associated data and m chains in the phase of the encryption). However π-Cipher starts with the initialization phase, where by a padding function (K, N) is mapped to $(K||N||10^*)$ and processed by the π-function. After the initialization part, the state is branched into a new states and associated data is processed. These a states are merged into one new state and the newly obtained state is branched into m new states where message blocks are encrypted. Finally the tag T is computed. For the privacy proof, we consider an adversary that makes q_p permutation queries and q_ε encryption queries of total length λ_ε. Let ε_K be an encryption query, consisting of a associated data blocks, and m message blocks. So the corresponding state values are given as follows:

$$\left(s^{init}; \quad \begin{bmatrix} s_{1,0}^{AD} & s_{1,1}^{AD} \\ \vdots & \vdots \\ s_{a,0}^{AD} & s_{a,1}^{AD} \end{bmatrix}; \quad s^{CIS}; \quad s_0^{SMN}; \quad s_1^{SMN}; \quad \begin{bmatrix} s_{1,0}^{M} & s_{1,1}^{M} \\ \vdots & \vdots \\ s_{m,0}^{M} & s_{m,1}^{M} \end{bmatrix} \right)$$

Here, if the j-th encryption query is of length $a+m$, then the number of state values $\sigma_{\epsilon,j}$ (calls to the π-function) is $2a + 2m + 4$ (2 of them is for the processing of SMN).

So the total number of π-function evaluations via the encryption queries is:

$$\sigma_\varepsilon := \sum_{j=1}^{q_\varepsilon} \sigma_{\varepsilon,j} \leqslant q_\varepsilon(2a + 2m + 4) = 2\lambda_\varepsilon + 4q_\varepsilon. \tag{3.1}$$

Also for decryption queries the number is the same and is denoted as $\sigma_{\mathcal{D}}$ and $\sigma_{\mathcal{D},|}$ analogously.

3.1.1 Privacy of π-Cipher

Theorem 2. *Let $\Pi = (\mathcal{E}, \mathcal{D})$ be the proposed authenticated encryption scheme with an ideal permutation π which operates on b bits. Then,*

$$\boldsymbol{Adv}_{\Pi}^{priv}(q_p, q_\varepsilon, \lambda_\varepsilon) \leqslant \frac{(q_p + \sigma_\varepsilon)^2}{2^b} + \frac{q_p + \sigma_\varepsilon}{2^k} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a}{2^r} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}},$$

where σ_ε is defined in (3.1).

For the privacy proof we need to obtain an upper bound for the advantage of an adversary who can distinguish the output of the proposed scheme with a random oracle in the ideal permutation model.

$$\mathbf{Adv}_{\Pi}^{priv}(\mathscr{A}) = \Delta_{\mathscr{A}}(\pi^{\pm}, \mathcal{E}_K; \pi^{\pm}, \$). \tag{3.2}$$

With replacing the permutation function π with some random function f from $0,1^b$ to $0,1^b$ and using the PRP/PRF Switching Lemma, we have:

$$\begin{aligned} \Delta_{\mathscr{A}}(\pi^{\pm}, \mathcal{E}_K; \pi^{\pm}, \$) - \Delta_{\mathscr{A}}(f^{\pm}, \mathcal{E}_K; f^{\pm}, \$) &\leqslant \frac{(q_p + \sigma_\varepsilon)(q_p + \sigma_\varepsilon - 1)}{2^{b+1}} \\ &\leqslant \frac{(q_p + \sigma_\varepsilon)^2}{2^{b+1}} \end{aligned} \tag{3.3}$$

Now we will restrict our attention to an adversary with oracle access to $(f^{\pm}, \{\mathcal{E}_K, \$\})$. Also we assume that the adversary only queries full blocks and that no padding rules are involved.

To aim the goal we define two collision events, guess and hit. Event guess corresponds to a primitive call in an encryption query hitting a direct primitive query or the opposite,

while hit corresponds to pairs of primitive calls that collide in encryption queries. So event is set to bad, if some of the events guess or hit occur.

$$\Delta_{\mathscr{A}}(f^{\pm}, \mathcal{E}_K; f^{\pm}, \$) \leqslant \mathbf{Pr}(\mathscr{A}^{f^{\pm}, \mathcal{E}_K} \text{sets event}). \tag{3.4}$$

We define the probability of *event is set* as follows:

$$\mathbf{Pr}(\mathbf{guess} \vee \mathbf{hit}) = \mathbf{Pr}(\mathbf{guess} \vee \mathbf{hit} | \neg(\mathbf{key} \vee \mathbf{multi}) + \mathbf{Pr}(\mathbf{key} \vee \mathbf{multi})) \tag{3.5}$$

Event guess. This event may occurs in the i-th permutation query (for $i = 1, \cdots, q_p$) or in any state evaluation of the j-th encryption query (for $j = 1, \cdots, q_\epsilon$). Denote that the state values of the j-th encryption query are as follows:

$$\left(s_j^{init}; \quad \begin{bmatrix} s_{j,1,0}^{AD} & s_{j,1,1}^{AD} \\ \vdots & \vdots \\ s_{j,a,0}^{AD} & s_{j,a,1}^{AD} \end{bmatrix}; \quad s_j^{CIS}; \quad s_{j,0}^{SMN}; \quad s_{j,1}^{SMN}; \quad \begin{bmatrix} s_{j,1,0}^{M} & s_{j,1,1}^{M} \\ \vdots & \vdots \\ s_{j,m,0}^{M} & s_{j,m,1}^{M} \end{bmatrix} \right) \tag{3.6}$$

We assume that event (**guess** $\vee$ **hit** $\vee$ **key** $\vee$ **multi**) has not been set before and also event (**key** $\vee$ **multi**) has not been set by this query before. From the further analysis we will exclude s_j^{init} from **guess** event because that case belongs to the **key** event. For $i = 1, \cdots, q_p$ let $j_i \in \{1, \cdots, q_\varepsilon\}$ be the number of encryption queries made before the i-th permutation query. And for $j = 1, \cdots, q_\varepsilon$ let $i_j \in \{1, \cdots, q_p\}$ be the number of permutation queries made before the j-th encryption query. Here we have two games, the first one is when we play with the permutation queries, and the other with encryption queries.

- In the first one the bad event occurs when input to the $f^{\pm}$ (x_i, y_i) collide with the elements in set $\mathscr{F}$. So for the forward query x_i there are at most ρ state values that have the same rate part, thus the capacity is unknown to the adversary and the bad event occurs with probability at most $\rho/2^c$. For the inverse query the situation is more complicated. We take y_i from the set of all encryption queries made before the i-th permutation query. Therefore the probability that guess is set via a direct query to the primitive is at most $\frac{q_p \rho}{2^c} + \sum_{i=1}^{q_p} \sum_{j=1}^{j_i} \frac{\sigma_{\epsilon,j}}{2^b}$.

- In the second one, bad event is set in the j-th encryption query for $j \in 1, \cdots, q_\varepsilon$. In this case we consider the probability that some of the states from (2) sets guess,

assuming that it has not been set before. Thus we have 3 subcases here.

1. The state value $s^{AD}_{j,k,0}$ for $k = 1, \cdots, a$, equals $f(s^{init}_j) \oplus ctr + k$ where ctr is some secret value not determined by the adversarial input. By assumption that $f(s^{init}_j)$ is randomly drawn from $\{0,1\}^b$, the probability to guess the state is at most $\frac{ai_j}{2^b}$ (where i_j is the number of permutation queries made before the j-th encryption query). The state value $s^{AD}_{j,k,1}$ for $k = 1, \cdots, a$, equals $f(s^{AD}_{j,k,0}) \oplus AD_k$ where AD_k is a value determined by the adversarial input. The probability to guess the state is at most $\frac{ai_j}{2^b}$.
2. $s^{CIS}_j = f(s^{init}_j) \oplus T'$ and is guessed with probability at most $i_j/2^b$
3. The same is situation in $s^{SMN}_{j,0}$ and $s^{SMN}_{j,1}$, probability is bound to $i_j/2^b$.
4. For the last branching (processing the message) we have the same situation as in the associated data part.

Concluding, the j-th encryption query sets **guess** with probability at most $\frac{ai_j}{2^b} + \frac{ai_j}{2^b} + 3\frac{i_j}{2^b} + \frac{mi_j}{2^b} + \frac{mi_j}{2^b}$. Or summing over all q_ε encryption queries, we get $\sum_{j=1}^{q_\varepsilon} \frac{ai_j}{2^b} + \frac{ai_j}{2^b} + 3\frac{i_j}{2^b} + \frac{mi_j}{2^b} + \frac{mi_j}{2^b} = \sum_{j=1}^{q_\varepsilon} \frac{i_j(2a+2m+3)}{2^b} \leqslant \sum_{j=1}^{q_\varepsilon} \frac{i_j\sigma_{\varepsilon,j}}{2^b}$.

Here we use that $\sum_{i=1}^{q_p}\sum_{j=1}^{j_i} \sigma_{\epsilon,j} \leqslant q_p\sigma_\epsilon$ with assumption that j_i always has its maximum value q_ϵ and $\sum_{j=1}^{q_\epsilon} i_j\sigma_{\epsilon,j} = \sum_{j=1}^{q_\epsilon}\sum_{k=1}^{\sigma_{\epsilon,j}} i_j \leqslant q_p\sigma_\epsilon$ with assumption that i_j always has its maximum value q_p.

Concluding,

$$\begin{aligned}\mathbf{Pr}(\mathbf{guess}|\neg(\mathbf{key} \vee \mathbf{multi})) &\leqslant \frac{q_p\rho}{2^c} + \sum_{i=1}^{q_p}\sum_{j=1}^{j_i}\frac{\sigma_{\varepsilon,j}}{2^b} + \sum_{j=1}^{q_\varepsilon}\frac{i_j\sigma_{\varepsilon,j}}{2^b} \\ &\leqslant \frac{q_p\rho}{2^c} + \frac{2q_p\sigma_\varepsilon}{2^b}. \end{aligned} \tag{3.7}$$

Event hit. We need to find whether the event hit is set, or whether two independent states (with different parents) collide in the encryption queries. It is clear that for the initialization state s^{init} stands: $s^{init}_j \neq s^{init}_{j'}$ because of the uniqueness of the nonce. So here any state value $s_{j,k}$ hits an initial state value $s_{j',1}$ only if $[s_{j,k}]^k = K$ which happens with probability $\sigma_\varepsilon/2^k$. In the other states $(\sigma_\varepsilon - q_\epsilon)$ for any two states $s_{j,k}, s_{j',k'}$ we have the following:

- $s^{AD}_{j,i,0} = f(s^{init}_j) \oplus (ctr + i), 1 \leqslant i \leqslant a$. Because of the fact that s^{init}_j is always fresh, collision between $s^{AD}_{j,i,0}$ and some older state will happen with probability no more

than $a/2^b$ for all i's. Note that $s^{AD}_{j,i,0}$ can never collide with a state from the same query because of the incremental counter's value. If $s^{AD}_{j,i,0}$ is a new state, then also it is a new input to f and $s^{AD}_{j,i,1}$ is a new one too. It hits a certain older state with probability $1/2^b$. If $s^{AD}_{j,i,1}$ is new for all i's, then the output of the function f is random, and the tag T' generated from the associated data phase is random too.

- $s^{CIS}_j = f(s^{init}_j) \oplus T'$ hits a state from an older query with probability at most $1/2^b$. Here we have a special case. The state s^{CIS}_j can collide with some of the states $s^{AD}_{j,i,0}$ from this query if and only if $[(ctr + i)||0^*]^r$ collide with T' (they have the same parent state s^{init}_j)). Probability for this is no more than $a/2^r$.

- $s^{SMN}_{j,0} = f(s^{CIS}_j) \oplus (ctr + a + 1)$. This state is new, so it can hit some older state with probability $1/2^b$, and the same is for $s^{SMN}_{j,1}$.

- $s^{M}_{j,i,0} = f(s^{SMN}_{j,1}) \oplus (ctr + a + 1 + i)$, so it is a new state for all parallel instances i and can collide with some other state from an older query with probability at most $m/2^b$. However, if $s^{M}_{j,i,0}$ is a new state, also new states are $s^{M}_{j,i,1}$ for all i's and the output of the function f is random. The tag that is generated at the end is a random value.

In total, the j-th encryption query sets event *hit* with probability at most: $\frac{(2a+3+2m)}{2^b}(\sigma_{\varepsilon,1} + \sigma_{\varepsilon,2} + \ldots + \sigma_{\varepsilon,j-1}) + \frac{a}{2^r}$

Summing over all queries,

$$\begin{aligned}
\mathbf{Pr}(\mathbf{hit}|\neg(\mathbf{key} \vee \mathbf{multi})) &\leqslant \sigma_\varepsilon/2^k + \sum_{j=1}^{q_\varepsilon} \frac{(2a+3+2m)}{2^b}(\sigma_{\varepsilon,1} + \ldots + \sigma_{\varepsilon,j-1}) + \frac{a}{2^r} \\
&\leqslant \sigma_\varepsilon/2^k + \sigma_\varepsilon a/2^r + \frac{\binom{\sigma_\varepsilon - q_\varepsilon}{2}}{2^b} \\
&\leqslant \frac{\sigma_\varepsilon}{2^k} + \frac{q_\varepsilon a}{2^r} + \frac{(\sigma_\varepsilon - q_\varepsilon)^2}{2^{b+1}} \qquad (3.8)
\end{aligned}$$

Event key. $\mathbf{Pr}(\mathbf{key}) \leqslant q_p/2^k$.

Event multi. Event **multi(j,k)** is used to bound the number of states that collide in the rate part. So, consider a new state value $s_{j,k-1}$; then for a fixed state value $x \in \{0,1\}^b$ it satisfies $f(s_{j,k-1}) = x$ or $s_{j,k} = f(s_{j,k-1}) \oplus w = x$ for some predetermined value w with probability $2/2^b$. Now, let $\alpha \in \{0,1\}^r$, more than ρ state values hit α with

probability at most $\binom{\sigma_\epsilon}{\rho}(2/2^r)^\rho$. According to the Stirling's approximation for factorials ($n! \sim \sqrt{2\pi n}(\frac{n}{e})^n \geqslant (\frac{n}{e})^n$), we have $\binom{\sigma_\epsilon}{\rho}(2/2^r)^\rho \leqslant (\frac{2e\sigma_\epsilon}{\rho 2^r})^\rho$. Considering any possible choice of α we obtain that

$$\mathbf{Pr}(\mathbf{multi}) = 2^r(\frac{2e\sigma_\epsilon}{\rho 2^r})^\rho. \tag{3.9}$$

So at the end, to complete the proof we need to make addition of the four bounds as:

$$\begin{aligned}\mathbf{Pr}(guess \vee hit) &= Pr(guess \vee hit|\neg(key \vee multi) + Pr(key \vee multi)) \\ &\leqslant \frac{q_p + \sigma_\varepsilon}{2^k} + \frac{(\sigma_\varepsilon - q_\varepsilon)^2}{2^{b+1}} + \frac{q_p\rho}{2^c} + \frac{q_\varepsilon a}{2^r} + \frac{2q_p\sigma_\varepsilon}{2^b} + 2^r(\frac{2e\sigma_\epsilon}{\rho 2^r})^\rho\end{aligned} \tag{3.10}$$

To simplify previous evaluation and also substitute ρ with some known values, we put $\rho = max\{r, \sqrt{\frac{2e\sigma_\varepsilon 2^c}{q_p 2^r}}\}$ and get the following:

$$\mathbf{Pr}(guess \vee hit) \leqslant \frac{q_p + \sigma_\varepsilon}{2^k} + \frac{(\sigma_\varepsilon - q_\varepsilon)^2}{2^{b+1}} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a}{2^r} + \frac{2q_p\sigma_\varepsilon}{2^b} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} \tag{3.11}$$

At the end, we have the final bounds for privacy of π-Cipher:

$$\mathbf{Adv}^{priv}_{\Pi}(\mathscr{A}) \leqslant \frac{(q_p + \sigma_\varepsilon)^2}{2^{b+1}} + \frac{q_p + \sigma_\varepsilon}{2^k} + \frac{q_p\sigma_\varepsilon + (\sigma_\varepsilon - q_\varepsilon)^2}{2^{b+1}} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a}{2^r} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} \tag{3.12}$$

We assume that $\frac{q_p\sigma_\varepsilon + (\sigma_\varepsilon - q_\varepsilon)^2}{2^{b+1}} \leqslant \frac{(q_p+\sigma_\varepsilon)^2}{2^{b+1}}$ and get the following:

$$\mathbf{Adv}^{priv}_{\Pi}(\mathscr{A}) \leqslant \frac{(q_p + \sigma_\varepsilon)^2}{2^b} + \frac{q_p + \sigma_\varepsilon}{2^k} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a}{2^r} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} \tag{3.13}$$

3.1.2 Authenticity of π-Cipher

Theorem 3. *Let $\Pi = (\mathcal{E}, \mathcal{D})$ be the proposed authenticated encryption scheme with an ideal permutation π which operates on b bits. Then,*

$$\boldsymbol{Adv}_{\Pi}^{auth}(q_p, q_\varepsilon, \lambda_\varepsilon, q_\mathcal{D}, \lambda_\mathcal{D}) \leqslant \frac{(q_p + \sigma_\varepsilon + \sigma_\mathcal{D})^2}{2^b} + \frac{q_\mathcal{D}}{2^\tau} + \frac{q_p + \sigma_\varepsilon + \sigma_\mathcal{D}}{2^k} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a + q_\mathcal{D} a}{2^r} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} + \frac{\sigma_\mathcal{D}(q_p + \sigma_\varepsilon + \sigma_\mathcal{D}/2)}{2^c},$$

where σ_ε and $\sigma_\mathcal{D}$ are defined in (3.1).

A forgery of an AE scheme is defined as the ability of an adversary $\mathscr{A}$ to generate a valid (N, AD, C, T) tuple, without directly querying it to the encryption oracle. Also an adversary attempts to make a decryption queries that does not result in $\perp$.

Let $\Pi = \{\mathcal{E}, \mathcal{D}\}$ be π-Cipher authenticated scheme with ideal permutation (π-function) which operates on $b(r+c)$ bits. The adversary $\mathscr{A}$ is given access to the Encryption oracle $\mathcal{E}_K$, Decryption oracle $\mathcal{D}_K$, permutation function p and its inverse p^{-1} oracle. Our goal is to bound the adversary's advantage to forge the scheme $\Pi = \{\mathcal{E}, \mathcal{D}\}$:

$$Adv_{\Pi}^{Auth}(\mathscr{A}) = \mathbf{Pr}(\mathscr{A}^{p^{\pm}, \mathcal{E}_K, \mathcal{D}_K} \text{forges}) \tag{3.14}$$

Using the PRP/PRF Switching Lemma and knowing that adversary can ask no more than $q_p + \sigma_\varepsilon + \sigma_\mathcal{D}$ evaluations to the function f, we have:

$$\mathbf{Pr}(\mathscr{A}^{p^{\pm}, \mathcal{E}_K, \mathcal{D}_K} \text{forges}) - \mathbf{Pr}(\mathscr{A}^{f^{\pm}, \mathcal{E}_K, \mathcal{D}_K} \text{forges}) \leqslant \frac{(q_p + \sigma_\varepsilon + \sigma_\mathcal{D})(q_p + \sigma_\varepsilon + \sigma_\mathcal{D} - 1)}{2^{b+1}} \leqslant \frac{(q_p + \sigma_\varepsilon + \sigma_\mathcal{D})^2}{2^{b+1}}$$

We will focus on adversary $\mathscr{A}$ to have an oracle access to $(f^{\pm}, \mathcal{E}_K, \mathcal{D}_K)$ and also that only makes full-block queries.

For this proof we need the same setting as in the privacy proof, about the guess and hit events, but here extended with new $\mathcal{D}$-related collision events $\mathcal{D}_{\mathbf{guess}}$ and $\mathcal{D}_{\mathbf{hit}}$. The state values are the same as in 3.6 with a δ appended to the subscript, where $\delta \in \{\mathcal{E}, \mathcal{D}\}$.

We observe that:

$$\mathbf{Pr}(\mathscr{A}^{f^{\pm},\mathcal{E}_K,\mathcal{D}_K}\text{forges}) \leqslant \mathbf{Pr}(\mathscr{A}^{f^{\pm},\mathcal{E}_K,\mathcal{D}_K}\text{forges}|\neg\text{event}) + \mathbf{Pr}(\mathscr{A}^{f^{\pm},\mathcal{E}_K,\mathcal{D}_K}\text{sets event}) \tag{3.15}$$

A bound on the probability that $\mathscr{A}$ forges when event does not happen is the same with the case where $\mathscr{A}$ can guess the tag T_j for some decryption query j. The final tag i s calculated as $f(s^{AD}_{\mathcal{D},j,1,1}) \boxplus \ldots f(s^{AD}_{\mathcal{D},j,a,1} \boxplus f(s^{SMN}_{\mathcal{D},j,1})) \boxplus f(s^{C}_{\mathcal{D},j,1,1}) \boxplus f(s^{C}_{\mathcal{D},j,m,1})$ Because the query is new, and event guess or hit doesn't happen before, at least one of these states is a new too, so f's output is uniformly distributed, and the final tag gets new value. So, the $j-th$ forgery attempt is successful with probability at most $1/2^{\tau}$. Summing over all decryption queries $q_{\mathcal{D}}$ we get:

$$\mathbf{Pr}(\mathscr{A}^{f^{\pm},\mathcal{E}_K,\mathcal{D}_K}\text{forges}|\neg\text{event}) \leqslant \frac{q_p}{2^{\tau}}$$

Next we need to explain what is the bound on the probability when adversary sets event. Here, $event = guess \vee hit \vee \mathcal{D}guess \vee \mathcal{D}hit$ and

$$\mathbf{Pr}(\mathscr{A}^{f^{\pm},\mathcal{E}_K,\mathcal{D}_K}\text{sets event}) \leqslant \mathbf{Pr}(guess \vee hit \vee \mathcal{D}guess \vee \mathcal{D}hit)$$
$$\leqslant \mathbf{Pr}(guess \vee hit \vee \mathcal{D}guess \vee \mathcal{D}hit|\neg(key \vee multi) + \mathbf{Pr}(key \vee multi) \tag{3.16}$$

Event $\mathcal{D}$guess. Note that the adversary may freely choose the rate part in decryption queries and primitive queries (the ciphertext represents the rate part). **$\mathcal{D}$guess** sets bad as soon as there is a primitive state and a decryption state whose capacity parts are equal.

$\mathcal{D}\mathbf{guess}(i;j,k) \equiv x_i = s_{\delta,j,k}$, where $\delta = \mathcal{D}$ which means that an adversary $\mathscr{A}$ is not able to ask a query $\mathcal{E}$. This happens with probability at most $q_p\sigma_{\mathcal{D}}/2^c$.

$$\mathbf{Pr}(\mathcal{D}\mathbf{guess}|\neg(\mathbf{key} \vee \mathbf{multi})) \leqslant q_p\sigma_{\mathcal{D}}/2^c.$$

Event $\mathcal{D}$hit. In this case an adversary has an ability to reuse nonces in the decryption queries. Any decryption state can hit the initial one (where just the key is unknown) with probability at most $\sigma_{\mathcal{D}}/2^k$. We have several subcases:

1. $(N;AD,C) = (N_{\delta,j};AD_{\delta,j},C_{\delta,j})$ but $T \neq T_{\delta,j}$. This case is with probability 0.

2. $(N; AD) = (N_{\delta,j}; AD_{\delta,j})$ but $C \neq C_{\delta,j}$. Let we say that $C_{\delta,j}$ shares the longest common prefix l with C and $l < m$. In this case $s^C_{j,l,0} = s^C_{\delta,j,l,0}$ and $s^C_{j,l,1} = C_l||[s^C_{\delta,j,l,1}]_c \neq s^C_{\delta,j,l,1}$, thus $s^C_{j,l,1}$ is a new state and new input to f. It can hit a certain older state with probability $1/2^c$. In total, the j-th decryption query sets event $\mathcal{D}hit$ with probability at most: $\frac{\sigma_{\mathcal{D},j}\sigma_\varepsilon + \binom{\sigma_{\mathcal{D},1}+\ldots+\sigma_{\mathcal{D},j-1}}{2}}{2^c}$

   If ciphertexts $C \neq C_{\delta,j}$ are different in all their blocks, then $s^{SMN}_{j,0} = s^{SMN}_{\delta,j,0}$ and $s^{SMN}_{j,1} = C_0||[s^{SMN}_{\delta,j,1}]_c \neq s^{SMN}_{\delta,j,1}$. This means that the state is fresh and can be hit with some older state with probability $1/2^c$.

   Here we have one subcase, or if SMN is the same. This means that $C_0 = C_{\delta,j,0}$ and CIS for the message phase is the same $f(s^{SMN}_{j,1}) = f(s^{SMN}_{\delta,j,1})$. So the reasoning carries over the case where the rest of the ciphertexts is different or they have longest common prefix.

3. $N = N_{\delta,j}$ but $AD \neq AD_{\delta,j}$. The analysis is the same as in the ciphertext case, except the case where inner collision can be done between the state where CIS is updated and states from the associated data ($a/2^r$). In the rest the reasoning carries over for all new future states.

4. $N \neq N_{\delta,j}$. The nonce is new so all future states in this query are new. Simplification can be applied here.

Summing over all queries we get:

$$\begin{aligned} Pr(\mathcal{D}\mathbf{hit}|\neg(\mathbf{key} \vee \mathbf{multi})) &\leqslant \sum_{j=1}^{q_\mathcal{D}} \frac{\sigma_\varepsilon \sigma_{\mathcal{D},j}}{2^c} + \binom{\sum_{j=1}^{q_\mathcal{D}} \sigma_{\mathcal{D},j}}{2}/2^c + \frac{q_\mathcal{D} a}{2^r} + \frac{\sigma_\mathcal{D}}{2^k} \\ &\leqslant \frac{\sigma_\mathcal{D}\sigma_\varepsilon + \sigma^2_\mathcal{D}/2}{2^c} + \frac{q_\mathcal{D} a}{2^r} + \frac{\sigma_\mathcal{D}}{2^k} \end{aligned}$$

Together with all the bounds from the privacy proof via 3.16 we get:

$$\begin{aligned} \mathbf{Pr}(event) \leqslant & \frac{q_p + \sigma_\varepsilon + \sigma_\mathcal{D}}{2^k} + \frac{(\sigma_\varepsilon - q_\varepsilon)^2}{2^{b+1}} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a + q_\mathcal{D} a}{2^r} + \frac{2q_p\sigma_\varepsilon}{2^b} + \\ & \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} + \frac{q_p \sigma_\mathcal{D}}{2^c} + \frac{\sigma_\mathcal{D}\sigma_\varepsilon + \sigma^2_\mathcal{D}/2}{2^c} \end{aligned} \tag{3.17}$$

At the end, we have the final bounds for integrity of π-Cipher:

$$Adv_{\Pi}^{Auth}(\mathscr{A}) \leqslant \frac{(q_p+\sigma_\varepsilon+\sigma_{\mathcal{D}})^2}{2^{b+1}} + \frac{q_{\mathcal{D}}}{2^\tau} + \frac{q_p+\sigma_\varepsilon+\sigma_{\mathcal{D}}}{2^k} + \frac{(\sigma_\varepsilon-q_\varepsilon)^2}{2^{b+1}} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a + q_{\mathcal{D}} a}{2^r} + \frac{q_p\sigma_\varepsilon}{2^{b+1}} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} + \frac{q_p\sigma_{\mathcal{D}}}{2^c} + \frac{\sigma_{\mathcal{D}}\sigma_\varepsilon + \sigma_{\mathcal{D}}^2/2}{2^c}$$

We assume that $\frac{q_p\sigma_\varepsilon+(\sigma_\varepsilon-q_\varepsilon)^2}{2^{b+1}} \leqslant \frac{(q_p+\sigma_\varepsilon+\sigma_{\mathcal{D}})^2}{2^{b+1}}$ and get the following:

$$Adv_{\Pi}^{Auth}(\mathscr{A}) \leqslant \frac{(q_p+\sigma_\varepsilon+\sigma_{\mathcal{D}})^2}{2^{b}} + \frac{q_{\mathcal{D}}}{2^\tau} + \frac{q_p+\sigma_\varepsilon+\sigma_{\mathcal{D}}}{2^k} + \frac{q_p r}{2^c} + \frac{q_\varepsilon a + q_{\mathcal{D}} a}{2^r} + \sqrt{\frac{8e\sigma_\varepsilon q_p}{2^b}} + \frac{\sigma_{\mathcal{D}}(q_p+\sigma_\varepsilon+\sigma_{\mathcal{D}}/2)}{2^c} \tag{3.18}$$

3.2 Bit diffusion analysis

We give a bit diffusion analysis for the $*$ operation and for one round of the π function of π16-Cipher, π32-Cipher and π64-Cipher.

In our analysis, we have used two experimental settings:

1. Examining the propagation of a one bit difference in a 10000 randomly generated $\mathbf{X}$ and $\mathbf{Y}$ (inputs) of the $*$ operation;

2. Examining the propagation of a one bit difference in a 1000 randomly generated Internal states IS for one round of the π function.

We performed several experiments for different word sizes ($\omega = 16, 32, 64$).

In the experimental setting under (1) we generated 10000 random values for $\mathbf{X}$ and $\mathbf{Y}$. First, we measured what is the Hamming distance between outputs $\mathbf{Z}$ and $\mathbf{Z}'$ ($\mathbf{Z} = \mathbf{X} * \mathbf{Y}$ and $\mathbf{Z}' = \mathbf{X}' * \mathbf{Y}$), where 1 bit is changed in the input $\mathbf{X}$ ($HammingDist(\mathbf{X}, \mathbf{X}') = 1$). After that we measured the Hamming distance between outputs $\mathbf{Z}$ and $\mathbf{Z}'$ of the $*$ operation ($\mathbf{Z} = \mathbf{X} * \mathbf{Y}$ and $\mathbf{Z}' = \mathbf{X} * \mathbf{Y}'$) where 1 bit is changed in $\mathbf{Y}$ ($HammingDist(\mathbf{Y}, \mathbf{Y}') = 1$). The results for $\omega = 16, 32, 64$ are shown in Figure 3.1, Figure 3.2 and Figure 3.3.

In the experimental setting under (2) we generated 1000 random values for the Internal State IS and used just one round of the π function. Here we measure the Hamming distance between the outputs ($\pi(IS)$ and $\pi(IS')$), where one bit is changed in the IS

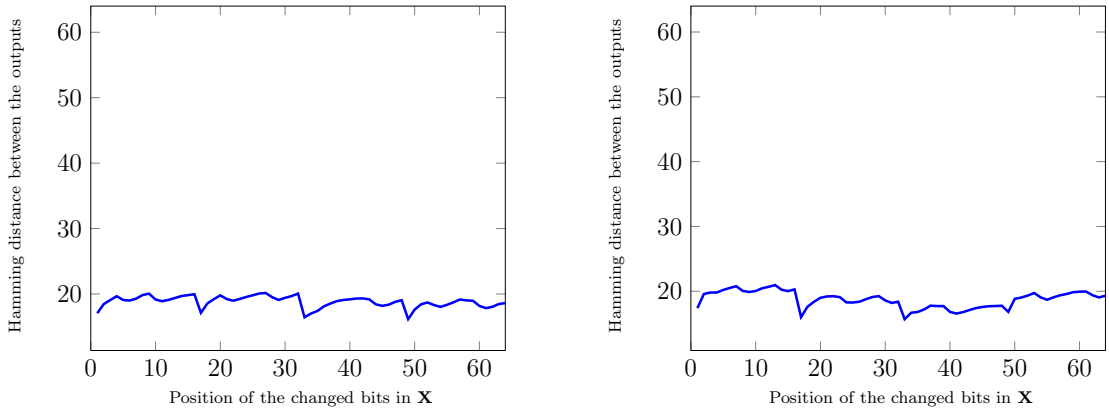


Figure 3.1: Avalanche effect of the $*$ operation for $\omega = 16$

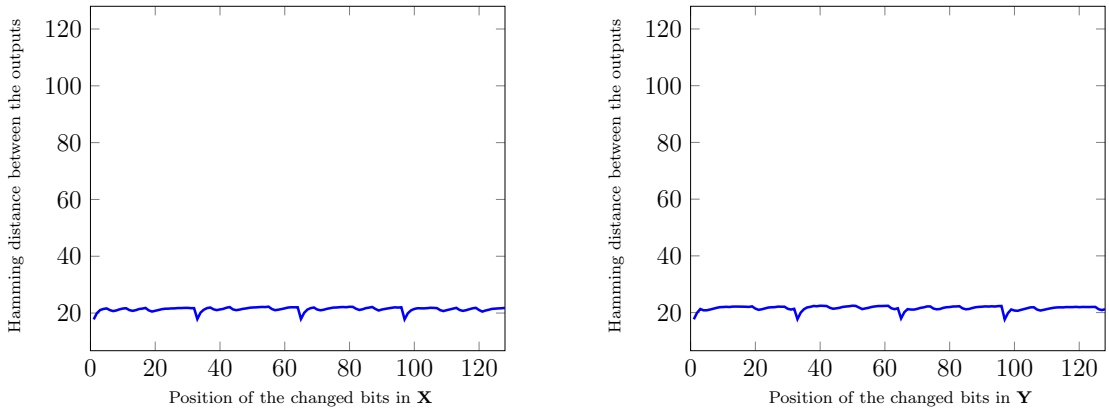


Figure 3.2: Avalanche effect of the $*$ operation for $\omega = 32$

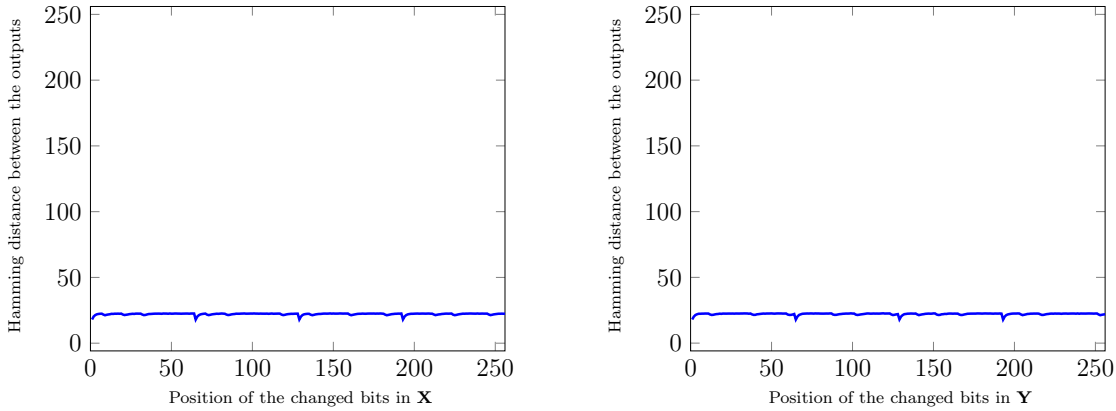


Figure 3.3: Avalanche effect of the $*$ operation for $\omega = 64$

($HammingDistance(IS, IS') = 1$). The results for $\omega = 16, 32, 64$ are shown in Figure 3.4, Figure 3.5 and Figure 3.6.

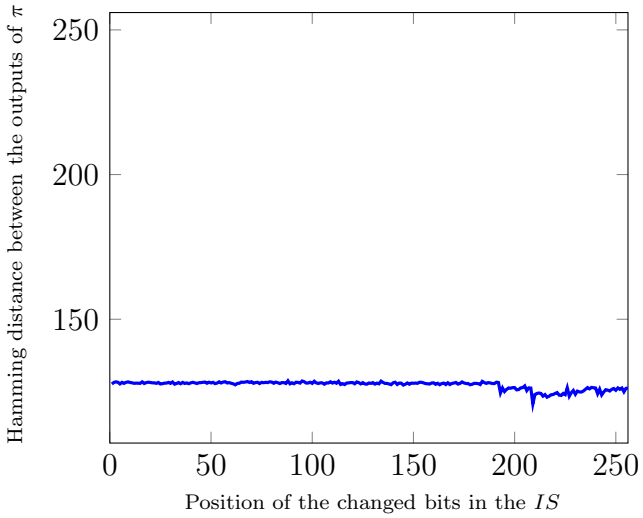


Figure 3.4: Avalanche effect of one round of the π function where $\omega = 16$ ($Min = 120.732$, $\mathbf{Avg = 127.255}$, $Max = 128.731$)

3.3 Distinguisher for one round of π16-Cipher096

We describe a distinguisher attack for one round of π16-Cipher096.

For $\omega = 16$ the size of the part J_2 in Figure 3.7 is 64 bits. Thus, by trying all 2^{64} values for J_2 (the graphical representation is in Figure 3.8) we can obtain in a unique way a list $\mathcal{L}_1 = \{(I_{1i}, I_{2i}) \mid 0 \leqslant i < 2^{64}\}$ of 2^{64} different values for the pairs (I_{1i}, I_{2i}) (the graphical representation is in Figure 3.9). We build a similar list $\mathcal{L}_2 = \{(I'_{1i}, I'_{2i}) \mid 0 \leqslant i < 2^{64}\}$ for the next encrypted block. Knowing that the encryption for the next block is by injecting a counter that was incremented by 1 from the previous counter value, and that it was injected in the position of I_1, while the value of I_2 was the same, we build a matching list $\mathcal{L}_3 = \{(I_{1i}, I'_{1i}, I_{2i}, I'_{2i}) \mid 0 \leqslant i < 2^{64}\}$ where $I'_{1i} - I_{1i} = 1$. With probability 1 there will be a pair of values where $I_{2i} = I'_{2i}$, while if the values of J_1 and J_3 were from an ideal random source, the probability that there will be a pair where $I_{2i} = I'_{2i}$ is very low (much less than 2^{-32}). In total, the complexity of the attack is 2^{65} computations of the

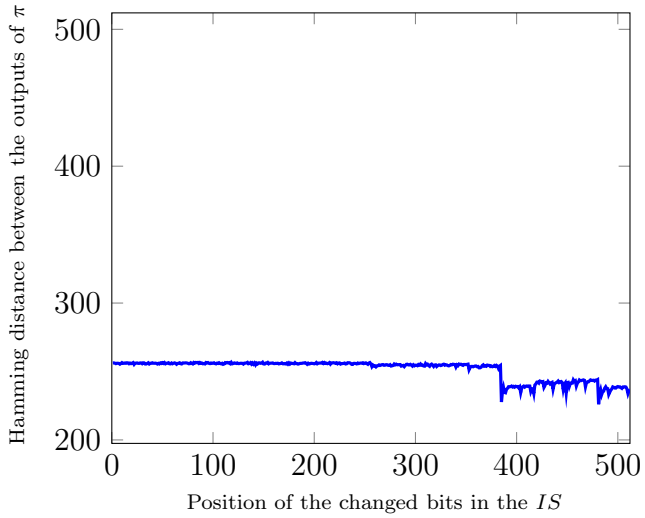


Figure 3.5: Avalanche effect of one round of the π function where $\omega = 32$ ($Min = 226.063$, $\mathbf{Avg = 256.765}$, $Max = 251.472$)

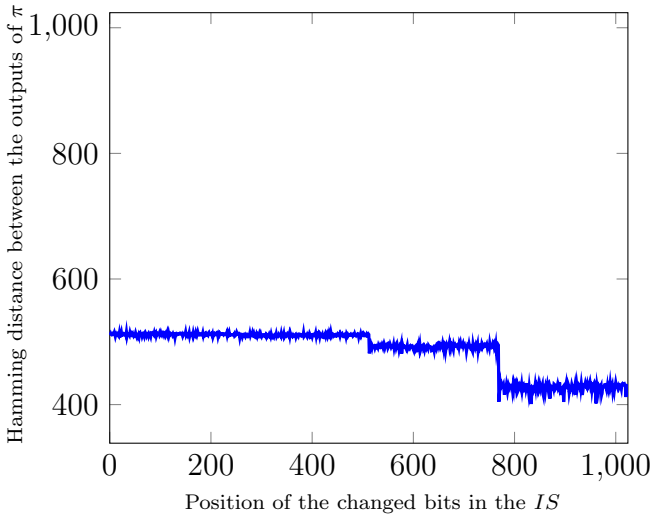


Figure 3.6: Avalanche effect of one round of the π function where $\omega = 64$ ($Min = 400.88$, $\mathbf{Avg = 485.646}$, $Max = 515.76$)

operation $*$, and the space is $2^{65} \times 16 = 2^{69}$ bytes.

For more than one round, or for bigger values of ω, this distinguishing attack is more complex than the claimed security of the cipher.

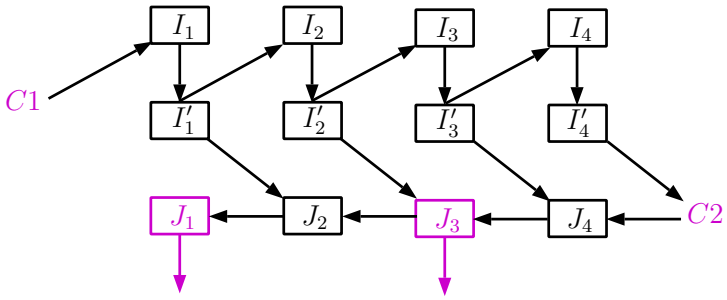


Figure 3.7: One round of π16-Cipher

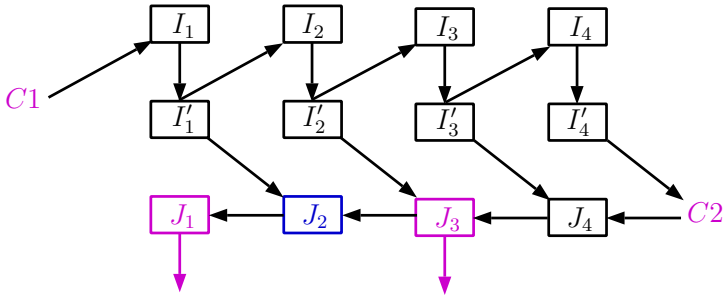


Figure 3.8: For J_2 we try all possible 2^{64} values.

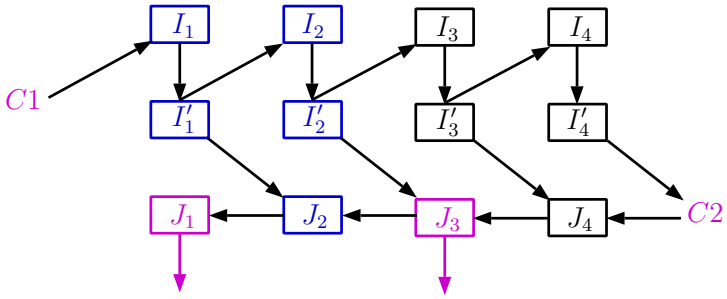


Figure 3.9: The blue squares are either guessed ones (that is J_2) or obtained in a unique way from the definition of the operation $*$.

Chapter 4

Features

4.1 Main Features

In this section we discuss the features of π-Cipher loosely following the structure of the features section `http://competitions.cr.yp.to/features.html` on the CAESAR web site.

Cipher-structure (Encrypt than MAC). First we want to point out that it is relatively straightforward to show that the π-Cipher is an Encrypt-then-MAC authenticated cipher. Let us recall the definition for the Encrypt-then-MAC authenticated cipher: We say that the authenticated cipher is Encrypt-then-MAC if a message M is encrypted under a secret key K_1 and then the tag T is calculated with another secret key K_2 as $MAC(K_2, C)$. The pair (C, T) is the output of the authenticated encryption procedure.

If we describe the e-triplex component used in π-Cipher in a mathematical form we have the following. First the message M is encrypted producing the ciphertext C as

$$\begin{aligned} IS &\leftarrow \pi(CIS_{rate} \bigoplus counter \;\; |||| \;\; CIS_{capacity}), \\ C &\leftarrow M \bigoplus IS_{rate}. \end{aligned}$$

Then, the tag T is calculated as

$$t \leftarrow \pi(C \;\; |||| \;\; IS_{capacity})_{rate}.$$

Here, the value of $CIS_{rate} \bigoplus counter \;\; |||| \;\; CIS_{capacity}$ has the role of K_1 in the definition of Encrypt-then-MAC, and the value of $C \;\; |||| \;\; IS_{capacity}$ has the role of the pair (K_2, C)

in the $MAC(K_2, C)$ part of the definition of Encrypt-then-MAC.

Associated Data and NONCE reuse. If we encrypt two different plaintexts M_1 and M_2 with the same secret key K, associated data AD and nonce $NONCE = (PMN, SMN)$, then neither the confidentiality nor the integrity of the plaintexts are preserved in the π–Cipher. However, as one measure to reduce the risks of a complete reuse of the $NONCE$ we have adopted the strategy of a composite $NONCE = (PMN, SMN)$. If either PMN or SMN are different, then both the confidentiality and integrity of plaintexts are preserved.

Plaintext corruption, associated-data corruption, message-number corruption, ciphertext corruption. We posit that the π-Cipher can straightforwardly be proven INT-CTXT secure under the assumption that the permutation π is an ideal random permutation without any structural distinguishers, based on the security proof in Section 3.1.

Ciphertext prediction. The best distinguishing attack that we know for the π-Cipher is for the version π16-Cipher096 with just one round and is described in Section 3.3. The complexity of the attack is 2^{65} computations of the operation $*$, and the space is $2^{65} \times 16 = 2^{69}$ bytes.

Replay and reordering. For the π-Cipher, the standard defense against both replay and reordering is for the sender to use strictly increasing public message numbers PMNs, and for the receiver to refuse any message whose message number is no larger than the largest number of any verified message. This requires both the sender and receiver to keep state.

Sabotage. The π-Cipher puts the encryption of the SMN value as the first block of the ciphertext C. Thus, in protocols that use the π-Cipher, the receiver can make an early reject of invalid messages by decrypting the first block (containing the SMN) and comparing it to its expected value. Only if this check passes the receiver continues with the rest of the decryption and tag computation. Note however, that this requires the protocol to not return error messages to the sender, in order to avoid timing attacks. AES-GCM does not have this property.

Plaintext espionage. Since the attacker's goal here is to figure out the user's secret message, the only feasible attack can happen when the size of the secret message is small by building a table of encrypted secret messages. To defend against this attack the π-Cipher requires the nonce pair $NONCE = (PMN, SMN)$ to have a unique value for every encryption.

Message-number espionage. In the π-Cipher there is a dedicated phase for encrypting the secret message number SMN, and figuring out the value of SMN is equivalent to breaking the whole cipher which is infeasible under the assumptions that the permutation $\pi()$ is random.

General input scheduling. The π-Cipher can offer two ways for reducing the latency: **(1)** If the key K and the public message number PMN are known in advance and used repeatedly, then it is possible to precompute phase 1. and store the resulting Common Internal State (CIS) for subsequent applications of the cipher. **(2)** If the key K, the public message number PMN and the associated data AD are known in advance and used repeatedly, then it is possible to precompute both phase 1. and phase 2. for subsequent uses. In both cases, in order to preserve the confidentiality and the integrity of the plaintext, for every encryption the secret message numbers SMN's must be unique.

Software performance. For efficient software implementations, we propose to use the π64-Cipher. On modern Intel CPUs (Sandy Bridge and Haswell) the initial and slightly optimized implementation (but non-SIMD) achieves the speeds from 6 cpb up to 12 cpb.

Lightweight hardware performance. For a lightweight hardware implementation we propose to use π16-Cipher096. Our implementation of the π-function on FPGA Xilinx Virtex7 needs 266 slices achieving a throughput of 4.34 Gpbs at 347 MHz. Also the message processor is built and it costs 1114 slices working at 250 MHz.

4.2 Extra Features

4.2.1 Tag second preimage resistance - resistance against finding second preimage for an authentication tag when the key is known (insider attack)

π-Cipher offers *some level* of tag second-preimage resistance. We say some level, since the computational efforts for finding a second-preimage for a given pair *(message, tag)* are not the same as for finding second-preimages for hash functions. And if the π-Cipher is used for authenticated encryption of messages with arbitrary length (in the framework of the maximally allowed size of the messages which is up to 2^{64} bytes), as it was shown by Leurent in [12] the tag second-preimages can be computed with complexities from 2^{22} using messages that are long 2^{11} blocks up to complexity 2^{45} using messages that are long 2^{22} blocks. Laurent's analysis on π-Cipher v1 was based on Wagner's generalized birthday attack [14] which complexity can be described as follows. The complexity of finding a preimage message $M = M_1||M_2||\ldots||M_K$ of K blocks, where K is smaller than some predefined big number, is:

$$\min_K O(K \cdot 2^{\frac{k}{1+\lg[K]}}) \tag{4.1}$$

If the length of the messages is not restricted, then the minimum in equation (4.1) is achieved for messages of $K = 2^{\sqrt{k}-1}$ blocks.

However, there are situations when we need to encrypt relatively short messages. For example, if we use the π-Cipher in IPSec or TLS with the most common Internet packet size of 1500 bytes, then for π64-Cipher128 or π64-Cipher256 the encrypted messages will have "just" 24 packets. In that case the Wagner's generalized birthday attack in order to find a second-preimage for a given tag will need 2^{108} e-triplex invocations and a space of 2^{113} bytes. We note that by choosing different values for the tweakable parameter N, we can achieve different levels of tag second-preimage resistance.

4.2.2 A wide block tweakable feature of π-Cipher

π-Cipher is designed to be tweakable for different word sizes, different security levels and different block sizes. The goal of π-Cipher is to achieve both high performance and strong security.

Authenticated encryption of stored data has its own specifics and differs from authenticated encryption of data in transit. The most notable difference is that an authenticated encryption algorithm used for data storage has to support independent encryption and decryption of portions of data. Those portions of data are typically represented as disk sectors. Old hard disk drives (HDDs) had sectors of 512 bytes but the hard disk producers moved to a 4096 (4K) byte sector size with the Advanced Format [10]. The new AF HDD storage devices read data from or write data to 4K-byte (4,096 bytes) physical sectors on the HDD media.

In contrast to the hard disk, a Solid State Drive (SSD) consists of semiconductor memory building blocks, and it contains no mechanical parts. The smallest unit of an SSD is a page, and it can be of size as small as 2KB, 4KB, 8KB, 16KB etc. It is not possible to read less than one page at once [8]. Several pages on the SSD are summarized to a block. For example, the Samsung SSD 840 EVO has blocks of size 2048KB, and each block contains 256 pages of 8KB each.

Table 4.1: Wide block characteristics of π64-Cipher256

	klen (in bits)	PMN (in bits)	SMN (in bits)	Rate in Bytes	N	Tag T (in bits)	R
wide block of 512B	256	512	0	512	32	256	2
wide block of 2KB	256	512	0	2048	128	256	2
wide block of 4KB	256	512	0	4096	256	256	2
wide block of 8KB	256	512	0	8192	512	256	2
wide block of 16KB	256	512	0	16384	1024	256	2

Using the encryption algorithm that can encrypt one sector (one page) as one block of message will be an advantage. From this perspective, we can use very efficiently π64-Cipher256 by tuning the parameter N to define arbitrary big block sizes that match the disk sector (page) sizes. For example, if $N = 256$ (instead of the recommended value $N = 4$) we can encrypt a whole sector of 4KB as one block. The sector number can be used as a part of the counter that is used in the π-Cipher when processing each block, the authenticated tag for that block can be stored in the data structure associated with that disk sector, and all those disk sectors can be processed independently and in parallel.

For encrypting standard old hard disk drives (HDDs) where the sector size is 512

bytes the rate of the cipher should be 512B and parameter should be $N = 32$ to process one sector at once.

Having all this in mind, we recommend 5 instances of π64-Cipher256 with parameters N that fits the sizes of older and modern disk sectors. The recommended variants are presented in Table 4.1. Note that the number of rounds here is 2.

4.2.3 How to securely do incremental encryption (even with nonce misuse)

According to the paper *"General Overview of the First-Round CAESAR Candidates for Authenticated Ecryption"* [1] by Abed, Forler and Lucks, for the *"Incremental Authenticated Encryption"* they took the following (in our opinion correct) criteria:

> "Note that some schemes may provide this property under the requirement of reusing the nonce. We consider nonce misuse to be an *erroneous* usage which should not be encouraged to obtain a nice feature. Hence, we denote scheme to provide incremental authenticated encryption only if the nonce is used only once and never is repeated."

By this criteria, they correctly categorized π-Cipher as *non-incremental* cipher.

In this note we explain how π-Cipher can be used as incremental authenticated cipher, that complies with the criteria of Abed, Forler and Lucks for incremental authenticated encryption. It comes with a costs of an additional data overhead of 64 bits per encrypted block. That data overhead serves as an update counter $UpdCtr$, that records the history of updates for every data block.

The default block sizes for π-Cipher are 128, 256 or 512 bits. Adding a data overhead of 64 bits for every data block would be an enormous and unacceptable overhead of 50%, 25% or 12.5%. Luckily, π-Cipher is designed to be tweakable for different block sizes. We have already described variants of π64-Cipher256 with block sizes of 512B, 2KB, 4KB, 8KB and 16KB. For those sizes the corresponding percentage of the overhead of 64 bits is: 1.5%, 0.39%, 0.19%, 0.09% and 0.04%. For even bigger block sizes, this overhead can become negligible.

Note that in case that the user do not want to keep in memory the precomputed values for CIS and $ctr + a + 1$ then he/she has to execute again the Initialization phase

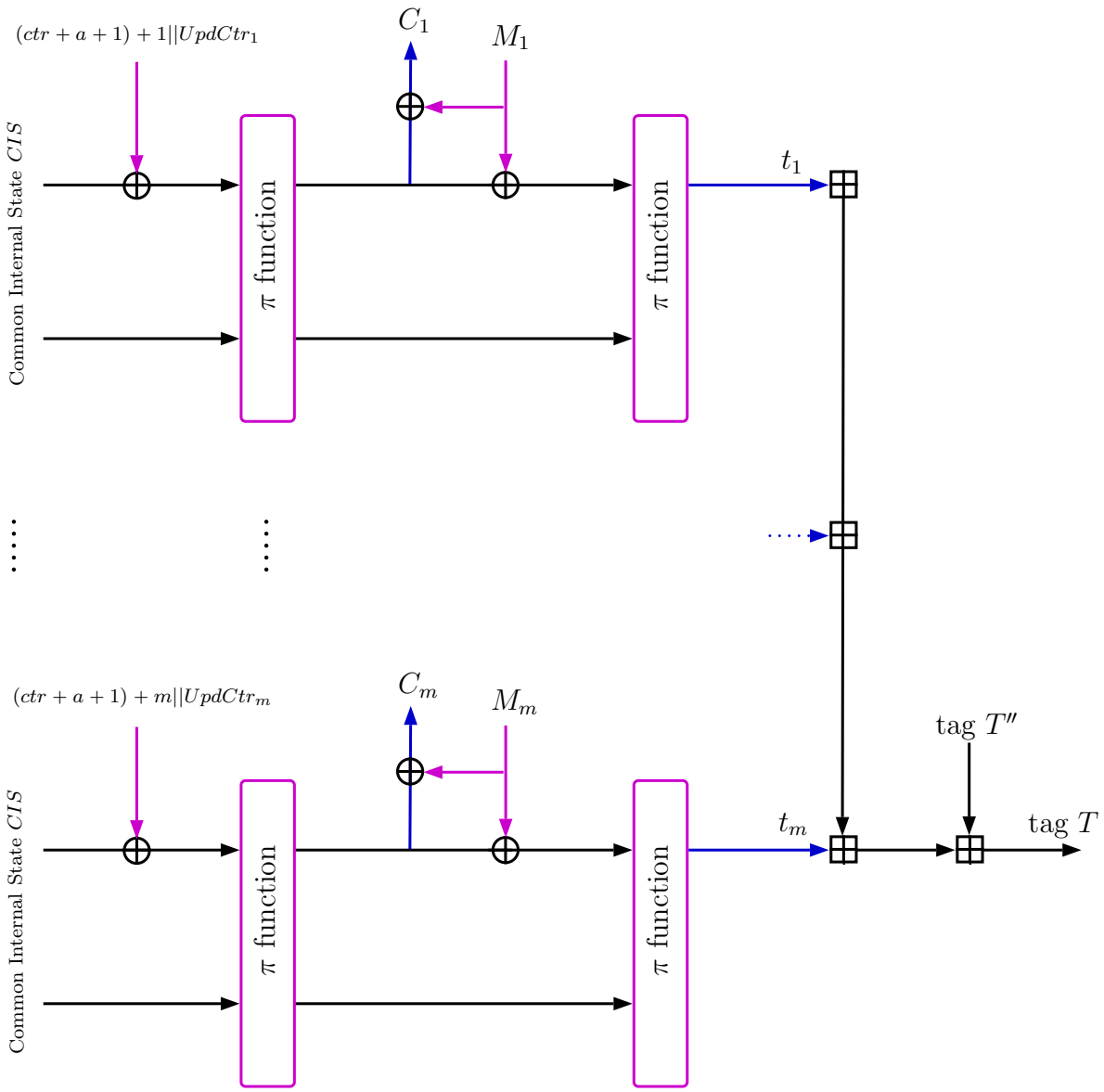


Figure 4.1: Processing the message M with m blocks in parallel. Here the 64-bit update counters $UpdCtr_i$ are injected into the sponge state.

Algorithm 1 - Incremental tag update operation
Input. The common internal state CIS, $ctr + a + 1$, the block index i, the old block M_i, the update counter for that block $UpdCtr_i$, the new block M_i' and the old tag value T.
Output. A new ciphertext block C_i and a new tag T.
1. For the old block M_i, calculate: $IS \leftarrow \pi(CIS_{rate} \oplus ((ctr + a + 1) + i \parallel UpdCtr_i) \,\|\|\| \, CIS_{capacity})$; $C_i \leftarrow M_i \bigoplus IS_{rate}$; $t_i \leftarrow \pi(C_i \,\|\|\| \, IS_{capacity})_{rate}$;
2. Increase the value of the update counter $UpdCtr_i \leftarrow UpdCtr_i + 1$
3. For the new block M_i', calculate: $IS \leftarrow \pi(CIS_{rate} \oplus ((ctr + a + 1) + i \parallel UpdCtr_i) \,\|\|\| \, CIS_{capacity})$; $C_i' \leftarrow M_i' \bigoplus IS_{rate}$; $t_i' \leftarrow \pi(C_i' \,\|\|\| \, IS_{capacity})_{rate}$;
4. Combine T, t_i and t_i' via a combining group operation $\boxplus_d$ to get the final authentication tag value $T' = T \boxminus_d t_i \boxplus_d t_i'$;
5. Replace: $C_i \leftarrow C_i'$ and $T \leftarrow T'$;
6. Output C_i and T.

Table 4.2: Incremental authentication encryption update with π-Cipher.

with the secret key K and the nonce part PMN, then will have to process the associated data and finally to encrypt again SMN.

As it is shown in Step 1 and Step 3, the incremental authentication encryption mode works in such a way that the 64-bit update counters $UpdCtr_i$ are injected in the permutation state concatenated to the block counters. That means that the vectors $(PMN, SMN, ctr, UpdCtr_i)$ form a unique nonce, for every block M_i, thus there is no nonce repetition and misuse. A graphical presentation of the encryption phase in this incremental authenticated encryption mode of π-Cipher is given in Figure 4.1.

4.2.4 π-Cipher is STREAM OAE2$^+$

Recently, Rogaway et al., [9] have introduced the definition of Online Authenticated-Encryption 2 and its STREAM OAE2 sub-variant. Their definition as they say *effectively does capture best-possible security for a users choice of plaintext segmentation and ciphertext expansion.* They also mention that the sponge duplex construction in [5] resembles OAE2.

If we have in mind that π-Cipher is based on the duplex construction of [5], with the additional cryptographic mechanisms that strengthen its robustness such as the features:

- tag second preimage resistance
- wide block tweakability
- incrementability
- use of SMN that guarantees confidentiality and integrity even when the K, AD and PMN are reused

it is clear why we consider π-Cipher has STREAM OAE2$^+$ design.

Chapter 5

Design rationale

Disclaimer: "The designer/designers have not hidden any weaknesses in this cipher."

5.1 Why parallelism, incrementality and tag second-preimage resistance?

While AES-GCM can be parallelized it is not a tag second-preimage resistant. In our presentation at DIAC 2013 [6] we located several reasons why MACs should retain some hash properties when the key is known. We argued that since the *Robustness* is one of the main goals of the future AEAD standard, tag second-preimage resistance should be one of the features that an AEAD cipher should posses. We also gave two realistic scenarios ("Secure audit logs" and Multi-cast authentication) when the lack of tag second-preimage resistance in authenticated encryption can be exploited.

While the proposed sponge constructions of authenticated encryption offer tag second-preimage resistance, they lack some of the properties that are also useful and desired (such as parallelizability and incrementality). As a result of this line of reasoning, we designed a cipher for authenticated encryption that is parallel, incremental and offers a certain level of tag second-preimage resistance.

5.2 Why constants in π–Cipher and how to choose them

In order to avoid the existence of some trivial fixed points in the permutation function π, we decided to use constants in the affine bijective transformations $\widehat{\mathbb{A}_1}$ and $\widehat{\mathbb{A}_3}$ from μ and ν permutations. The reason why we choose these constants is that they are represented as a sequence of equal distribution of 0s and 1s. Having these constants in $\widehat{\mathbb{A}_1}$ and $\widehat{\mathbb{A}_3}$ we are not aware of any point X such that

$$\pi(\mathbf{X}) = \mathbf{X}.$$

The size and value of the constants depend on the length of the words in the π–Cipher. First we generate the set *Constants* of all possible 8-bit (1 byte) candidates with equal distribution of 0s and 1s in their binary representation. The total number of elements in this set is 70, and is given by:

$$\begin{aligned} \mathit{Constants} = \{&\texttt{0xF0}, \texttt{0xE8}, \texttt{0xE4}, \texttt{0xE2}, \texttt{0xE1}, \texttt{0xD8}, \texttt{0xD4}, \texttt{0xD2}, \texttt{0xD1}, \texttt{0xCC},\\ &\texttt{0xCA}, \texttt{0xC9}, \texttt{0xC6}, \texttt{0xC5}, \texttt{0xC3}, \texttt{0xB8}, \texttt{0xB4}, \texttt{0xB2}, \texttt{0xB1}, \texttt{0xAC},\\ &\texttt{0xAA}, \texttt{0xA9}, \texttt{0xA6}, \texttt{0xA5}, \texttt{0xA3}, \texttt{0x9C}, \texttt{0x9A}, \texttt{0x99}, \texttt{0x96}, \texttt{0x95},\\ &\texttt{0x93}, \texttt{0x8E}, \texttt{0x8D}, \texttt{0x8B}, \texttt{0x87}, \texttt{0x78}, \texttt{0x74}, \texttt{0x72}, \texttt{0x71}, \texttt{0x6C},\\ &\texttt{0x6A}, \texttt{0x69}, \texttt{0x66}, \texttt{0x65}, \texttt{0x63}, \texttt{0x5C}, \texttt{0x5A}, \texttt{0x59}, \texttt{0x56}, \texttt{0x55},\\ &\texttt{0x53}, \texttt{0x4E}, \texttt{0x4D}, \texttt{0x4B}, \texttt{0x47}, \texttt{0x3C}, \texttt{0x3A}, \texttt{0x39}, \texttt{0x36}, \texttt{0x35},\\ &\texttt{0x33}, \texttt{0x2E}, \texttt{0x2D}, \texttt{0x2B}, \texttt{0x27}, \texttt{0x1E}, \texttt{0x1D}, \texttt{0x1B}, \texttt{0x17}, \texttt{0x0F}\} \end{aligned}$$

The constants used in the $\pi\omega$–Ciphers with different word sizes consist of a concatenation of a consecutive 8-bit elements from the set *Constants*.

π16–Cipher uses eight 16-bit constants for the $*$ operation and eight 4-tuples of 16-bit constants for the rounds. We start from the first element of the set `0xF0` and take 16 successive byte values up to the value `0xB8` and form the constants for the $*$ operation. After that, we take every eight successive byte values to form one round constant. We repeat this procedure 6 times and generate the constants $C_1, C_2, \ldots, C_6$ for the rounds.

Since π32–Cipher uses eight 32-bit constants for the $*$ operation and eight 4-tuples of 32-bit constants for the rounds, we take 32 successive byte values starting from the

first one `0xF0` and form the constants for the $*$ operation. The next 128 consecutive byte values are used for generating constants for the rounds. Because we need 128 bytes for the round constants of π32–Cipher and have 70 elements in the set, for the 71st byte we take the value of the first element in the set *Constants* again.

π64–Cipher uses eight 64-bit constants for the $*$ operation. They are taken as 64 successive byte values from the set, starting from the first one `0xF0`. Since π64–Cipher uses 4–tuples of 8 bytes round constants, we take them successively starting from the value `0x27` in the set *Constants*.

The values of both constants (for the $*$ operation and for rounds) for different π–Ciphers are given in Section 1.3.

Chapter 6

Intellectual property

We, the designers of the π-Cipher hereby declare that, to the best of our knowledge, the design of the algorithm that we have submitted for the CAESAR competition, is not covered by any patents. We also hereby declare that we intend never to cover the design of the π-Cipher by any patent.

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.

π-Cipher stands for the concept of "open authorship". It is very similar concept to the concept of open source projects, but we introduce it in the area of the intellectual efforts of crypto designs. In essence, it gives opportunity to all people that contribute anyhow in the development of π-Cipher (for example if a tweak is introduced due to an analysis of the cipher, or if a new mode of operation is proposed, or new implementations are produced), then all those contributors have an opportunity (if they want) to ask to be added to the list of designers for new versions or variants of π-Cipher.

The members of the π-Cipher v2.0 and further versions will be listed in chronological order as they joined the list of designers.

Chapter 7

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.

Acknowledgments

We would like to thank Gaëtan Leurent and Thomas Fuhr for their detailed observation on the π-Cipher, pointing out the problem with the padding function in v1.0, and giving us a note for removing a bug in the reference C code. Also we would like to thank Bart Mennink for his valuable and excellent advices in the process of proving the security of π-Cipher.

Much of the work on this cipher was done while Hristina Mihajloska was visiting Danilo Gligoroski's group at Department of Telematics, NTNU. This visit was supported from ICT COST Action - IC1306 *Cryptography for Secure Digital Interaction* as a Short Term Scientific Mission (STSM).

References

[1] Farzaneh Abed, Christian Forler, and Stefan Lucks. General Overview of the First-Round CAESAR Candidates for Authenticated Ecryption. Cryptology ePrint Archive, Report 2014/792, 2014. `http://eprint.iacr.org/`.

[2] Mihir Bellare, Roch Guérin, and Phillip Rogaway. Xor macs: New methods for message authentication using finite pseudorandom functions. In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 1995.

[3] D. J. Bernstein. Cryptographic competitions: Caesar call for submissions, draft 5 (2013.12.01). Cryptographic competitions: CAESAR, 2013. Available at `http://competitions.cr.yp.to/caesar-call-5.html`.

[4] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Proceedings of the 18th International Conference on Selected Areas in Cryptography*, SAC'11, pages 320–337, 2012.

[6] Danilo Gligoroski, Hristina Mihajloska, and Håkon Jacobsen. Should MAC's retain hash properties when the key is known in the next AEAD? Presentation at DIAC 2013, 2013. `http://2013.diac.cr.yp.to/slides/gligoroski.pdf`.

[7] Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, and Vlastimil Klima. Cryptographic hash function EDON-$\mathcal{R}'$. In *1st International Workshop on Security and Communication Networks*, pages 85–95, Trondheim, Norway, May 2009. IEEE.

[8] Emmanuel Goossaert. Coding for ssds part 3: Pages, blocks, and the flash translation layer, February, 2014. `http://codecapsule.com/2014/02/12/coding-for-ssds-part-3-pages-blocks-and-the-`

[9] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizr. Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. Cryptology ePrint Archive, Report 2015/189, 2015. `http://eprint.iacr.org/`.

[10] IDEMA. The Advent of Advanced Format. *idema.org*, 2013. `http://www.idema.org/?page_id=2369`.

[11] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond 2 c/2 security in sponge-based authenticated encryption modes. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 85–104. Springer Berlin Heidelberg, 2014.

[12] Gaëtan Leurent. Tag Second-preimage Attack against π-cipher, March 2014. `https://hal.inria.fr/hal-00966794`.

[13] Pawel Morawiecki and Josef Pieprzyk. Parallel authenticated encryption with the duplex construction. Cryptology ePrint Archive, Report 2013/658, 2013. `http://eprint.iacr.org/`.

[14] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.