ORANGE

Designers/Submitters:
Bishwajit Chakraborty - Indian Statistical Institute,Kolkata
Mridul Nandi - Indian Statistical Institute, Kolkata, India

bishu.math.ynwa@gmail.com
mridul.nandi@gmail.com

March 22, 2019

1 Introduction

This work proposes ORANGE, a variant of sponge authenticated encryption and sponge hash which can absorb data in the optimum rate. In other words, it is an Optimum RAte spoNGE construction. In this submission, we propose an authenticated encryption, named as ORANGE-Zest and a hash function, named as ORANGISH based on a 256-bit permutation.

Underlying Permutation. Both construction use PHOTON_{256} as the underlying permutation. Among the existing 256-bit permutations, PHOTON_{256} [5] is one of the lightest designs in the literature. It has been well studied and well analysized. Moreover, PHOTON_{256} is also a part of ISO-IEC: 29192-5 standard, which deal specifically with light-weight cryptography.

Hash Mode. The mode of hash function ORANGISH is very close to the JH hash function [10] which is one of the finalists of SHA3-competition. JH mode allows us to absorb 128 bit data for each permutation call. Thus, it has higher throughput compared with classical sponge hash function [1, 2]. The design of ORANGISH is expected to provide collision and preimage security against all adversaries running in time 2^{112} (i.e. making 2^{112} permutation calls).

Authenticated Encryption Mode. The mode for ORANGE-Zest is a close variant of sponge with full state absorption. The full state absorption is possible as we hold another state of size 128-bits, a part of the output of previous execution of the underlying permutation. We use this dynamic secret state to mask a part of the ciphertext. This mode can be easily generalized to a design based on a permutation with $2n$ bit state. In our case, $n = 128$. To summarize the performance of our AE mode, it has $3n$ bit state with $2n$ bit rate. To process $2n$ bit blocks, we apply $4n$-bit XOR, in addition to one permutation call. The design of ORANGE is expected to provide privacy and confidentiality against all adversaries running in time 2^{128} (i.e. making 2^{128} permutation calls) having at most 2^{64} data.

2 Notations and Conventions

We use $\{0,1\}^+$ and $\{0,1\}^n$ to denote the set of all non-empty (binary) strings, and n-bit strings, respectively. λ denotes the empty string and $\{0,1\}^* = \{0,1\}^+ \cup \{\lambda\}$. For all practical purposes: we use little-endian format of indexing, and assume all binary strings are *byte-oriented*, i.e. belong in $(\{0,1\}^8)^*$. For any string $B \in \{0,1\}^+$, $|B|$ denotes the number of bits in B, and for $0 \le i \le |B|-1$, b_i denotes the i-th bit of B, i.e. $B = b_{|B|-1} \cdots b_0$. where b_0 is the least significant bit (LSB) and $b_{|B|-1}$ is the most significant bit (MSB). Given a nonempty bit string B of size $x < n$, we denote $\mathsf{pad}(B)$ as $0^{n-x-1}1B$. Thus we always pad the extra bits from MSB side. When $x = n$, we define $\mathsf{pad}(B)$ as B itself. The chop function chops either the most significant or least significant bits. For $k \le n$, and $B \in \{0,1\}^n$, $\lfloor B \rfloor_k := B_{k-1} \dots B_0$ and $\lceil B \rceil_k := B_{n-1} \dots B_{n-k}$.

For $B \in \{0,1\}^+$, $(B_{\ell-1}, \dots, B_0) \xleftarrow{n} B$, denotes the n-bit *block parsing* of B into $(B_{\ell-1}, \dots, B_0)$, where $|B_i| = n$ for $0 \le i \le \ell-2$, and $1 \le |B_{\ell-1}| \le n$. For $A, B \in \{0,1\}^+$, and $|A| = |B|$, $A \oplus B$ denotes the "bitwise XOR" operation on A and B. For $A, B \in \{0,1\}^+$, $A \| B$ denotes the "string concatenation" operation on A and B. For any $B \in \{0,1\}^+$ and a non-negative integer s, $B \ll s$ and $B \lll s$ denote the "left shift by s" and "circular left shift by s" operations on B, respectively. The notations for right shift and circular right shift are analogously defined using $\gg$ and $\ggg$, respectively. Given two matrices $M_{m \times l}$ and $N_{l \times n}$, $M \cdot N$ denotes the matrix multiplication of M and N.

We will use a compact representation of if-else statement by the following expression $P \ ? \ b \ : \ c$ where P is some mathematical statement. This evaluates to b if P is true and c otherwise. $P_1 \ \& \ P_2 \ ? \ b_1 \ : \ b_2 \ : \ b_3 \ : \ b_4$ evaluates to b_1 if both P_1 and P_2 are true, to b_2 if only P_1 is true, to b_3 if only P_2 is true and to b_4 if none of P_1 , P_2 are true.

Field Multiplication . It is well known that $x^{128} + x^7 + x^2 + x + 1$ is a primitive polynomial over the finite field of order 2. We define a constant $a := 0^{120}10000111$. Given $B \in \{0,1\}^{128}$, the α-multiplication on an 128 bit string $B := b_{127} \cdots b_1 b_0$, denoted by $\alpha \cdot B$, is defined as $(B \ll 1) \oplus a$ if $b_{127} = 1$, $B \ll 1$, otherwise. For a $c \in \mathbb{Z}_{\ge 0}$, $\alpha^c \cdot B$ denotes c times repeated α-multiplication of B.

2.1 Our Recommendation

ORANGE is primarily parameterized by its underlying Permutation P. We choose P to be PHOTON_{256} as described in Algorithm 2. We propose a hash function, called ORANGISH, and authenticated encryption ORANGE-Zest. Description of both are given in 1. Our proposal of ORANGE-Zest uses a nonce-size of 128−bits and a key-size of 128−bits to produce a 128-bit tag. It is clear from the description that the hash function ORANGISH is very close to the process of associated data in ORANGE-Zest. So a combined implementation of both ORANGISH and ORANGE-Zest would be optimized.

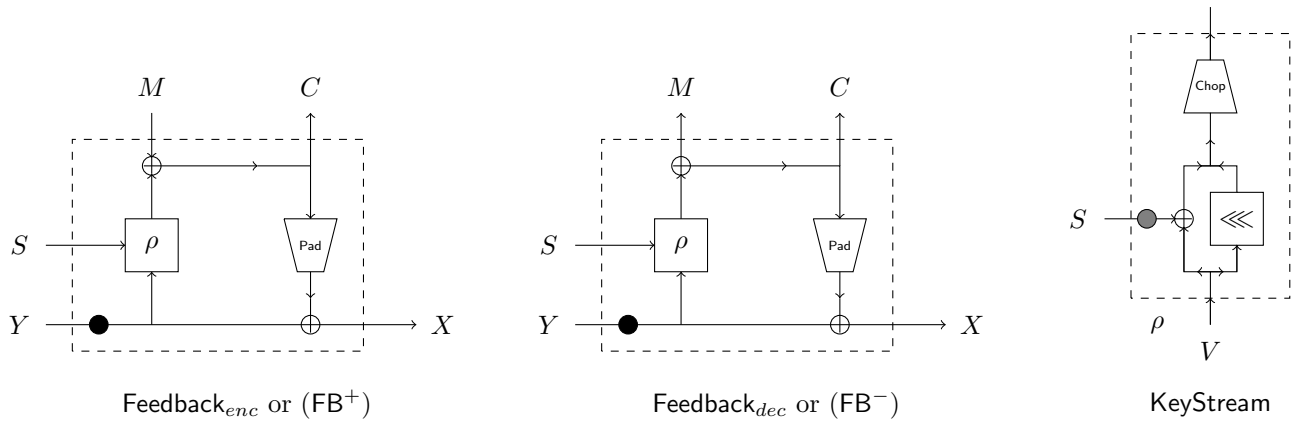

Figure 1: Feedback process for ORANGE-Zest: KeyStream module or the function ρ describes how the key-stream is defined. Feedback functions describe to define the next input X for the block cipher and the ciphertext (for encryption feedback) and message (for decryption feedback). The black circular dot represents the mult operation which is nothing but the α^{δ_M}-multiplication to the most significant half of Y (the previous block cipher output). Note that $\delta_M = 0, 1, 2$ for imtermediate block, complete last block, partial last block respectively .The gray circular dot represents the mult operation which is nothing but the α-multiplication to S. Here, Pad and Chop, pads and chops appropriate amounts of bits from MSB or LSB sides. The exact definitions of these process can be found in Algorithm 1

3 PHOTON$_{256}$ Permutation

We use PHOTON$_{256}$ [5] as our underlying 256-bit permutation in our mode. We use exactly same permutation without changing any part of the definition as it has been well studied. However, for the sake of completeness we provide a brief description of the permutation in this section (see Algorithm 2). It is applied on a state of 64 elements of 4 bits each, which is represented as a (8×8) matrix X. Let $X[i, j]$ denote the element at i-th row and j-th column of X.

PHOTON$_{256}$ is composed of 12 rounds. Each round applies four layers of functions AddConstant, SubCells, ShiftRows and MixColumnSerial on the state in a sequence. The description of these functions are given in Algorithm2. Informally, AddConstant adds fixed constants to the cells of the internal state. SubCells applies an 4-bit S-Box (see Table. 1) to each of the 64 4-bit cells. ShiftRows rotates the position of the cells in each of the rows and MixColumnSerial linearly mixes all the columns independently using a serial matrix multiplication. The multiplication with the coefficients in the matrix is in $GF(2^4)$ with $x^4 + x + 1$ being the irreducible polynomial.

We represent a serial matrix Serial$[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$ by

$$\text{Serial}[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7] := \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \end{pmatrix}.$$

Table 1: The PHOTON S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S-box	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Algorithm 1 ORANGE-Zest and ORANGISH and their main modules. Here, $\perp$ and $\top$ denote the abort and accept symbols respectively.

function ORANGE-ZEST$_{[\mathsf{P}]}$.enc(K, N, A, M)
  $(A_{a-1}, \ldots, A_0) \xleftarrow{n} A$
  $(M_{m-1}, \ldots, M_0) \xleftarrow{n} M$
  if $a = 0, m = 0$ **then**
    $(T, *) \leftarrow \mathsf{P}((K \oplus 2) \| N)$
    return (λ, T)
  if $a = 0$ **then**
    $(C, U) \leftarrow \mathsf{proc\_txt}(K, (K \oplus 1) \| N, M, +)$
    return $\mathsf{proc\_tg}(U)$
  $C \leftarrow \lambda$
  if $a \neq 0$ **then** $U \leftarrow \mathsf{proc\_hash}(K \| N, A, 1, 2)$
  if $m \neq 0$ **then** $(C, U) \leftarrow \mathsf{proc\_txt}(K, U, M, +)$
  return $(C, \mathsf{proc\_tg}(U))$

function ORANGISH(D)
  $(D_{d-1}, \ldots, D_0) \xleftarrow{n} D$
  $D_d \leftarrow (n \upharpoonright |D_{d-1}|)?\ 0^{n-2}10 : 0^{n-1}1$
  $D_{d-1} \leftarrow \mathsf{pad}(D_{d-1})$
  $X \leftarrow (0^n || D_0)$
  for $i = 0$ to $d - 1$ **do**
    $A_i \leftarrow (D_i \| D_{i+1})$
  $Z \leftarrow \mathsf{proc\_hash}(X, (A_{d-1} \| \ldots \|, A_0), 1, 1)$
  $Z_1 \leftarrow \mathsf{P}(Z)$
  $Z_2 \leftarrow \mathsf{P}(Z_1)$
  return $\lfloor Z_2 \rfloor_n \| \lfloor Z_1 \rfloor_n$

function $\mathsf{proc\_txt}(S_0, U_0, D, \mathsf{dir})$
  $(D_{d-1}, \ldots, D_0) \xleftarrow{2n} D$
  for $i = 0$ **to** $d - 1$ **do**
    $V_i \leftarrow \mathsf{P}(U_i)$
    if $i = d - 1$ **then**
      $c \leftarrow (2n \mid |D_{d-1}|)?1 : 2$
      $V_i \leftarrow \mathsf{mult}(c, V_i)$
    $KS_i \leftarrow \rho(S_i, V_i)$
    $D'_i \leftarrow D_i \oplus \lfloor KS_i \rfloor_{|D_i|}$
    if $\mathsf{dir} = \text{"}+\text{"}$ **then** $D_i \leftarrow D'_i$
    $S_{i+1} \leftarrow \lceil V_i \rceil_n$
    $U_{i+1} \leftarrow V_i \oplus \mathsf{pad}(D_i)$
  return (D', U_a)

function ORANGE-ZEST$_{[\mathsf{P}]}$.dec(K, N, A, C, T)
  $(A_{a-1}, \ldots, A_0) \xleftarrow{n} A$
  $(C_{m-1}, \ldots, C_0) \xleftarrow{n} C,\ M \leftarrow \lambda$
  if $a = 0, m = 0$ **then** $(T', *) \leftarrow \mathsf{P}((K \oplus 2) \| N)$
  if $a = 0$ **then**
    $(M, U) \leftarrow \mathsf{proc\_txt}(K, (K \oplus 1) \| N, C, -)$
    $T' \leftarrow \mathsf{proc\_tg}(U)$
  if $a \neq 0$ **then** $U \leftarrow \mathsf{proc\_hash}(N \| K, A, 1, 2)$
  if $m \neq 0$ **then** $(M, U) \leftarrow \mathsf{proc\_txt}(K, U, C, -)$
  $T' \leftarrow \mathsf{proc\_tg}(U)$
  if $T \neq T'$ **then**
    return $\perp$
  else
    return $(M, \top)$

function $\mathsf{proc\_hash}(X, D, c_0, c_1)$
  $(D_{d-1}, \ldots, D_0) \xleftarrow{2n} D$
  $X_0 \leftarrow X$
  for $i = 0$ **to** $d - 2$ **do**
    $Y_i \leftarrow \mathsf{P}(X_i)$
    $X_{i+1} \leftarrow Y_i \oplus D_i$
  $c \leftarrow (2n \mid |D_{d-1}|)?c_0 : c_1$
  $Y_{d-1} \leftarrow \mathsf{P}(X_{d-1})$
  $Y_{d-1} \leftarrow \mathsf{mult}(c, Y_{d-1})$
  $X_d \leftarrow Y_{d-1} \oplus \mathsf{pad}(D_{d-1})$
  return X_d

function $\rho(S, Y)$
  $(Y^b, Y^t) \xleftarrow{n} Y$
  $Z \leftarrow (Y^b \oplus \alpha S) \| (Y^t \lll 1)$
  return Z

function $\mathsf{mult}(c, V)$
  $(V^b, V^t) \xleftarrow{n} V$
  return $V^t \parallel \alpha^c \cdot V^b$

function $\mathsf{proc\_tg}(U)$
  $(U^b, U^t) \xleftarrow{n} U$
  return $\mathsf{P}(U^t \| U^b)$

4 Security of ORANGE

Here we describe some possible strategies to attack the ORANGE mode, and give a rough estimate on the amount of data and time required to mount those attacks (see Table 2). In the following discussion:

- D denotes the data complexity of the attack. This parameter quantifies the online resource requirements, and includes the total number of blocks (among all messages and associated data) processed through the underlying permutation for a fixed master key. Note that for simplicity we also use D to denote the data complexity of forging attempts.
- T denotes the time complexity of the attack. This parameter quantifies the offline resource requirements, and includes the total time required to process the off line evaluations of the underlying permutation. Since one call of the permutation can be assumed to take a constant amount of time, we generally take T as the total number of off line calls to the permutation.

Security Model	Data complexity ($\log_2 D$)	Time complexity ($\log_2 T$)
IND-CPA	64	128
INT-CTXT	64	128

Table 2: Security Claims of ORANGE-Zest. We remark that the given values indicate the amount of data and time required to make the attack advantage close to 1.

Algorithm 2 PHOTON_{256} Modules. Note that we view the state X as a matrix and $M^8 \cdot X$ in MixColumnSerial represents the matrix multiplication in the underlying field $GF(2^4)$ defined over the irreducible polynomial $x^4 + x + 1$.

function AddConstant(X, K)
  RC[12] $\leftarrow \{1, 3, 7, 14, 13, 11, 6, 12, 9, 2, 5, 10\}$
  IC[8] $\leftarrow \{0, 1, 3, 7, 15, 14, 12, 8\}$
  for $i = 0$ to 7 **do**
    $X[i, 0] \leftarrow X[i, 0] \oplus \text{RC}[k] \oplus \text{IC}[i]$
  return X

function SubCells(X)
  for i = 0 to 7 j=0 to 7 **do**
    $X[i, j] \leftarrow \text{S-box}(X[i, j])$
  return X

function ShiftRows(X)
  for $i = 0$ to 7 $j = 0$ to 7 **do**
    $X'[i, j] \leftarrow X[i, (j + i)\%4]$
  return X'

function MixColumnSerial(X)
  $M \leftarrow \text{Serial}[2, 4, 2, 11, 2, 8, 5, 6]$
  $M^8 \cdot X$
  return X

function PHOTON_{256}(X)
  for $i = 0$ to 11 **do**
    $X \leftarrow \mathsf{AddConstant}(X)$
    $X \leftarrow \mathsf{SubCells}(X)$
    $X \leftarrow \mathsf{ShiftRows}(X)$
    $X \leftarrow \mathsf{MixColumnSerial}(X)$
  return X

Table 3: Security of Hash Function Family ORANGISH.

Mode	Security	Time complexity security (in bits)
ORANGISH	Collision	112
ORANGISH	Pre-image	128

4.1 IND-CPA and INT-CTXT Security of ORANGE-Zest

The privacy security of a permutation based construction relies on no collision of the inputs among online and offline permutation calls. Note that both top and bottom part of the input of a permutation call during the computation of an encryption query has full entropy due to two previous outputs. Hence a collision would happen with probability at most $1/2^{-256}$. The privacy claim of our design follows from this observation.

Note that the tag verification algorithm is almost same as that of Beetle [3]. Hence, a similar argument follows.

4.2 Collision Security of ORANGISH

To mount a collision attack on ORANGISH, suppose an adversary can make q many permutation calls. Suppose all the states reachable from the initial state (we define the initial state as 0^{256}) using the permutation calls are called *reachable states*. The adversary can set up the queries in an adaptive way to make all the query inputs (and hence query outputs) *reachable states*. We claim that the number of reachable state can be at most nq (by using multi-collision argument, details will be provided later). Hence, finding a collision pair has probability at most $n^2q^2/2^{256}$. This leads to our claim on the collision security.

5 Preimage Security of ORANGISH

In ORANGISH we set the tag size as 256 bits and the tag squeeze rate as 128 bits. So given a preimage target $T_2\|T_1$, an adversary needs to find a Z such that $\mathsf{PHOTON}_{256}(Z\|T_1) = \star\|T_2$ or $\mathsf{PHOTON}^{-1}_{256}(Z\|T_2) = \star\|T_1$. It is easy to see that the probability of this event can be bounded by $\frac{q}{2^{128}}$ where q is the number of P and P^{-1} call.

6 Existing Analysis of PHOTON_{256}

Basic security analysis for PHOTON_{256} has been provided explicitly in the original paper [5]. PHOTON is an ISO standard with a comfortable security margin. As we have used PHOTON_{256} we only report briefly the known analysis of it.

A rebound-like attack [5] allows us to distinguish 8 rounds of PHOTON_{256} from an ideal permutation of the same size with time complexity 2^{16} (and later reduced to $2^{10.8}$[7]) and memory complexity of 2^8. In [6] Jean et al. presented a distinguisher for 9 round PHOTON_{256} with time complexity of 2^{184} and memory complexity of 2^{32}. Some other attacks are improved Indifferentiable [8] and statistical Integral distinguisher [4]. Recently, Wang et al. [9] presented the first full round distinguishers on PHOTON_{256} based on zero-sum partitions of size 2^{184}.

We believe that all these distinguishers have no impact on the security of our construction as these attacks are much more costlier than the security target we are aiming.

7 Design Rational

7.1 Choice of the Mode

Our primary goal is to design a lightweight cipher that has optimum throughput. No such sponge variant is known so far which can absorb message at the rate of the state of the permutation. Our design achieves this at the cost of an additional state. So it is optimum in rate. We also use JH variant of hash which also absorbs much higher data compared with classical sponge hash.

7.2 Need of an additional state

A b-bit permutation with r bit rate leaks r bit information about the permutation outputs. So when $r = b$, all the state value would be leaked and the key can be computed easily. Thus we need additional state to keep some amount of secret. We find that 128 bit additional state (chosen dynamically) provides the desired security.

7.3 Choice of the Permutation

PHOTON is an ISO-standard lightweight permutation which also provides sufficient amount of security level.

References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007.

[2] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 181–197. Springer, 2008.

[3] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Cryptology ePrint Archive*, 2018:805, 2018.

[4] Tingting Cui, Ling Sun, Huaifeng Chen, and Meiqin Wang. Statistical integral distinguisher with multi-structure and its application on AES. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part I*, pages 402–420, 2017.

[5] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[6] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved rebound attack on the finalist grøstl. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *LNCS*, pages 110–126. Springer, 2012.

[7] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Multiple limited-birthday distinguishers and applications. In *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, pages 533–550, 2013.

[8] Yusuke Naito and Kazuo Ohta. Improved indifferentiable security analysis of PHOTON. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 340–357, 2014.

[9] Qingju Wang, Lorenzo Grassi, and Christian Rechberger. Zero-sum partitions of PHOTON permutations. In *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, pages 279–299, 2018.

[10] Hongjun Wu. The hash function jh. *Submission to NIST (round 3)*, 6, 2011.

Appendix A

Figures of ORANGE for Different Cases

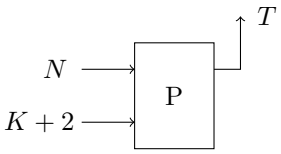

Figure 2: ORANGE-Zest encryption ($|A| = 0, |M| = 0$).

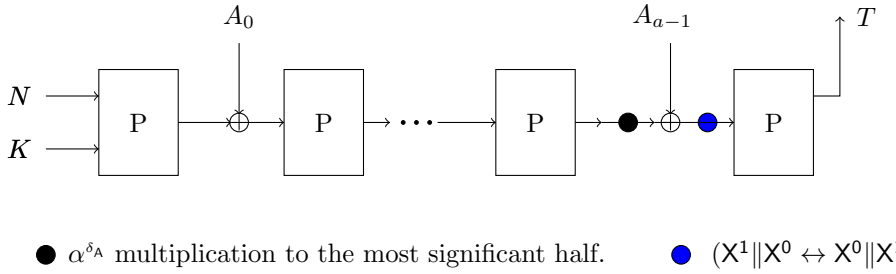

Figure 3: ORANGE-Zest encryption ($|M| = 0, |A| \neq 0, \delta_A = 1/2$ for complete-last/ partial block).

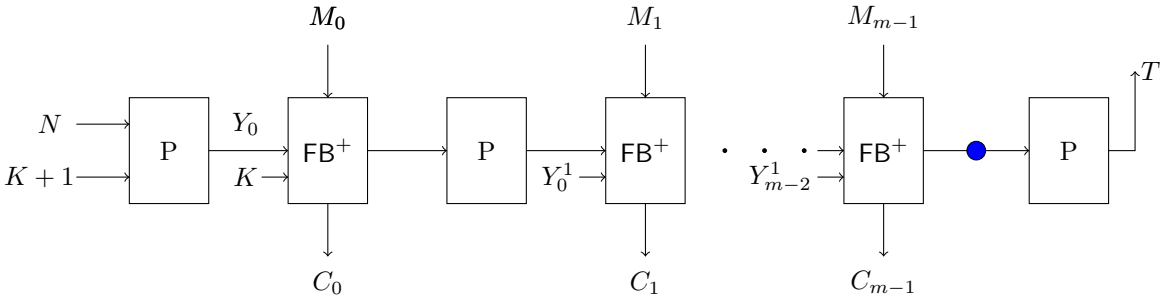

Figure 4: ORANGE-Zest encryption ($|A| = 0, |M| \neq 0$).

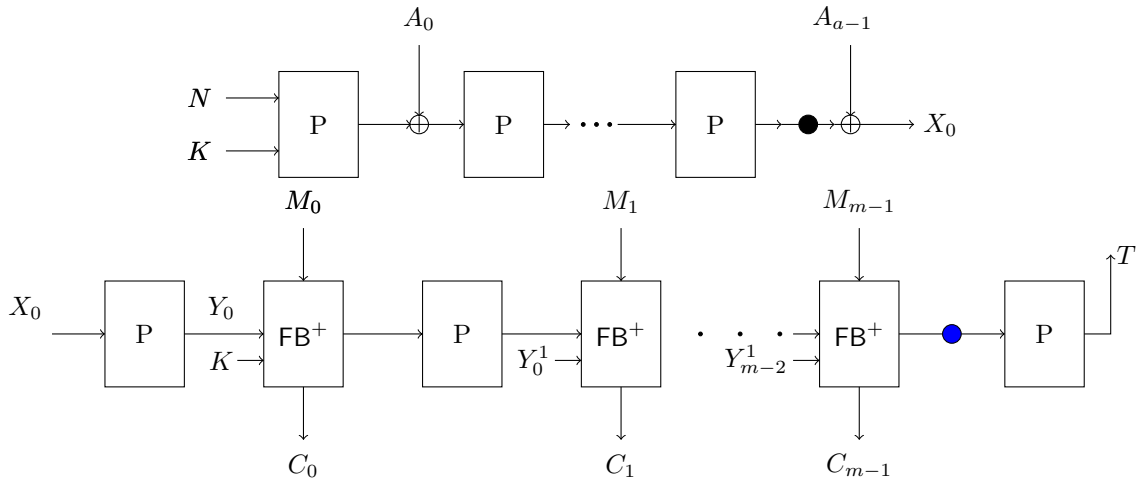

Figure 5: ORANGE-Zest encryption ($|A| \neq 0, |M| \neq 0$).

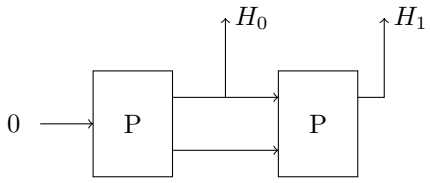

Figure 6: ORANGISH output ($|M| = 0$). The final hash output is defined as $H_1 \| H_0$.

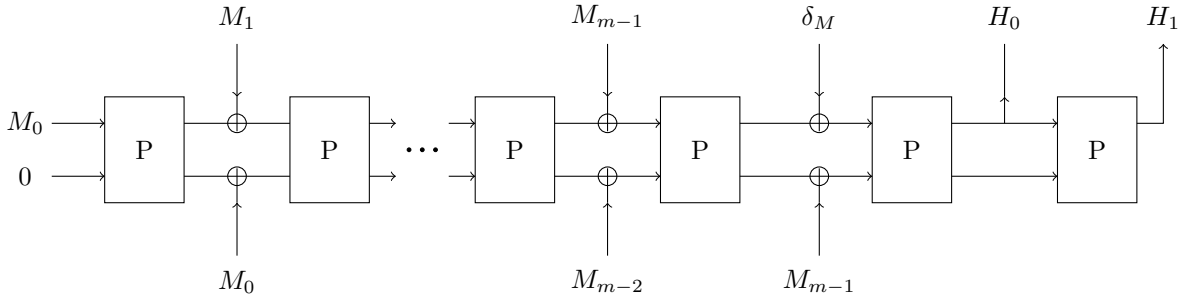

Figure 7: ORANGISH output ($|M| \neq 0$, $\delta_M = 1/2$ for complete/ partial input). The final hash output is defined as $H_1 \| H_0$.

Appendix B

Test vectors for ORANGE-Zest

Test vector 1:
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E0F$
$PT =$
$AD = 00010203$
$CT = F41612F0FD6758018FDC1377675401DF$

Test vector 2:
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E0F$
$PT =$
$AD =$
$CT = 5A65624E01D1349D2211EFBD52217976$

Test vector 3:

$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E0F$
$PT = 000102030405060708090A0B0C0D0E0F101112131415161718191A$
$AD = 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D$
$CT =$
$D75622A343E2459DEAA1B9A1784C5B84DC3ED112E895154CBDC0261C367EBBF849231F4C0$
$79B16DCEA57DC$

Test vectors for ORANGISH

Test vector 1:
$Msg = 00010203$
$MD = 51390073EFBB1DEF2CEAD9688CC2C9D907F2EF6AC8C8D7E73317EB2C28155226$

Test vector 2:
$Msg = 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223242526$
$2728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F404142434445464748494A4B4C4D4E4$
$F505152535455565758595A5B5C5D5E5F606162$
$MD = 7B1A8606FF708377BB612E0712C7E824921A8D78B9AD3258A7B400E96AA349C3$

Test vector 3:
$Msg = 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223242526$
$2728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F$
$MD = 85739793F2A59EC254488C3931447E86E0F3C0C919899DDA1BF34B1639DFDCD8$