GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

CSE 470 – CRYPTOGRAPHY AND COMPUTER SECURITY

ELEPHANT - ROMULUS

PROJECT REPORT

Can Duyar
171044075

<u>RESEARCH PART:</u>

ELEPHANT:

Introduction:

The Elephant authenticated encryption technique is shown. Elephant uses a nonce-based encrypt-then-MAC design, using counter mode encryption and a variant of the Wegman-Carter-Shoup MAC function for message authentication.

The mode is based on permutations and only assesses them in the forward direction. Unlike OCB-based authenticated encryption techniques, there is no requirement to implement multiple primitives or the inverse of the primitive. Additionally, we may depend on and expand on the rich literature of permutations utilized in sponge-based lightweight hashing. Elephant, on the other hand, is not sponge-based; rather, it deviates from the traditional technique of serial permutation-based authenticated encryption. Elephant is parallelizable by design, simple to construct thanks to the use of LFSRs for masking, and efficient thanks to clever considerations on how the masking should be executed exactly. The Elephant mode is structurally sound, according to a security analysis using the ideal permutation model.

Elephant's parallelizability eliminates the need to instantiate it with a huge permutation.we can use as few as 160-bit permutations while still meeting the NIST lightweight call's security goals. The Elephant scheme is made up of three parts:

<u>1. Dumbo: Elephant-Spongent-π:</u> This instance meets the security analysis' minimum permutation size requirement. it delivers 112-bit security if the online complexity is no more than 246 blocks. This instance, like Spongent, is well-suited to hardware.

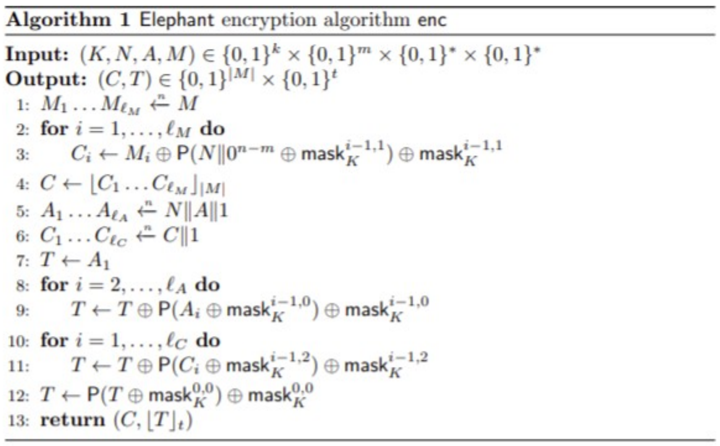
2- <u>Jumbo: Elephant-Spongent-π:</u> This is a more conservative version of Elephant. it uses the same permutation family but provides 127-bit security under the same online complexity conditions. Spongent- is ISO/IEC standardized, as an example.

3- <u>Delirium: Elephant-Keccak-f:</u> This model is geared primarily toward software applications, yet it still works admirably in hardware. Elephant instantiated with Keccak-f also achieves 127-bit security, with a higher bound on the online complexity of roughly 270 blocks. The permutation is the smallest NIST SHA-3 instance that meets our needs.

Elephant's three cryptographic permutations can also be used for cryptographic hashing – Spongent and Keccak are both sponges – but due to our quest for small permutations, these cryptographic hash functions cannot meet the 112-, or 127-bit security levels guaranteed by our authenticated encryption schemes. To accomplish sponge-based hashing with at least 112-bit security, however, a cryptographic permutation of at least 225 bits is required.

Encryption:

Encryption enc takes a key K ∈ {0, 1}^k, a nonce N ∈ {0, 1}^m, associated data A ∈ {0, 1}^ ∗ and a message M ∈ {0, 1}^ ∗ as input and produces a ciphertext C ∈ {0, 1}^|M| and a tag T ∈ {0,1}^t.

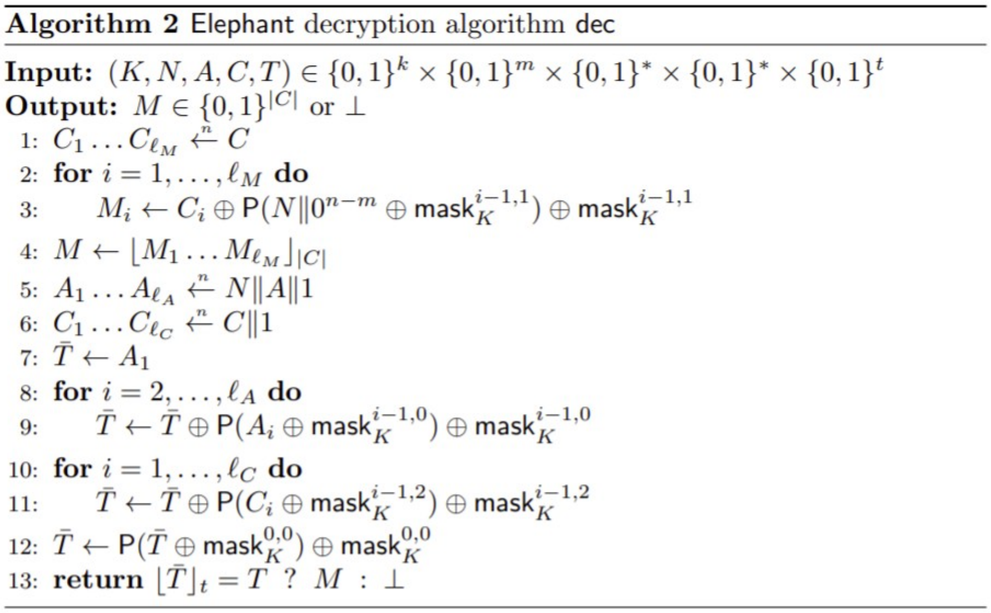


Algorithm 1 Elephant encryption algorithm enc

```
Input: $(K, N, A, M) \in \{0,1\}^k \times \{0,1\}^m \times \{0,1\}^* \times \{0,1\}^*$
Output: $(C, T) \in \{0,1\}^{|M|} \times \{0,1\}^t$
 1: $M_1 \dots M_{\ell_M} \xleftarrow{n} M$
 2: for $i = 1, \dots, \ell_M$ do
 3:     $C_i \leftarrow M_i \oplus \mathsf{P}(N \| 0^{n-m} \oplus \mathsf{mask}_K^{i-1,1}) \oplus \mathsf{mask}_K^{i-1,1}$
 4: $C \leftarrow \lfloor C_1 \dots C_{\ell_M} \rfloor_{|M|}$
 5: $A_1 \dots A_{\ell_A} \xleftarrow{n} N \| A \| 1$
 6: $C_1 \dots C_{\ell_C} \xleftarrow{n} C \| 1$
 7: $T \leftarrow A_1$
 8: for $i = 2, \dots, \ell_A$ do
 9:     $T \leftarrow T \oplus \mathsf{P}(A_i \oplus \mathsf{mask}_K^{i-1,0}) \oplus \mathsf{mask}_K^{i-1,0}$
10: for $i = 1, \dots, \ell_C$ do
11:     $T \leftarrow T \oplus \mathsf{P}(C_i \oplus \mathsf{mask}_K^{i-1,2}) \oplus \mathsf{mask}_K^{i-1,2}$
12: $T \leftarrow \mathsf{P}(T \oplus \mathsf{mask}_K^{0,0}) \oplus \mathsf{mask}_K^{0,0}$
13: return $(C, \lfloor T \rfloor_t)$
```

Decryption:

Decryption dec accepts a key K ∈ {0, 1}^k, a nonce N ∈ {0, 1}^m, associated data A ∈ {0, 1}^ ∗ , a ciphertext C ∈ {0, 1}^ ∗ and a tag T ∈ {0, 1}^t as input and returns a message M ∈ {0, 1}^|M| if the tag is correct.



Algorithm 2 Elephant decryption algorithm dec

```
Input: $(K, N, A, C, T) \in \{0,1\}^k \times \{0,1\}^m \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t$
Output: $M \in \{0,1\}^{|C|}$ or $\perp$
 1: $C_1 \dots C_{\ell_M} \xleftarrow{n} C$
 2: for $i = 1, \dots, \ell_M$ do
 3:     $M_i \leftarrow C_i \oplus \mathsf{P}(N \| 0^{n-m} \oplus \mathsf{mask}_K^{i-1,1}) \oplus \mathsf{mask}_K^{i-1,1}$
 4: $M \leftarrow \lfloor M_1 \dots M_{\ell_M} \rfloor_{|C|}$
 5: $A_1 \dots A_{\ell_A} \xleftarrow{n} N \| A \| 1$
 6: $C_1 \dots C_{\ell_C} \xleftarrow{n} C \| 1$
 7: $\bar{T} \leftarrow A_1$
 8: for $i = 2, \dots, \ell_A$ do
 9:     $\bar{T} \leftarrow \bar{T} \oplus \mathsf{P}(A_i \oplus \mathsf{mask}_K^{i-1,0}) \oplus \mathsf{mask}_K^{i-1,0}$
10: for $i = 1, \dots, \ell_C$ do
11:     $\bar{T} \leftarrow \bar{T} \oplus \mathsf{P}(C_i \oplus \mathsf{mask}_K^{i-1,2}) \oplus \mathsf{mask}_K^{i-1,2}$
12: $\bar{T} \leftarrow \mathsf{P}(\bar{T} \oplus \mathsf{mask}_K^{0,0}) \oplus \mathsf{mask}_K^{0,0}$
13: return $\lfloor \bar{T} \rfloor_t = T \ ? \ M \ : \ \perp$
```

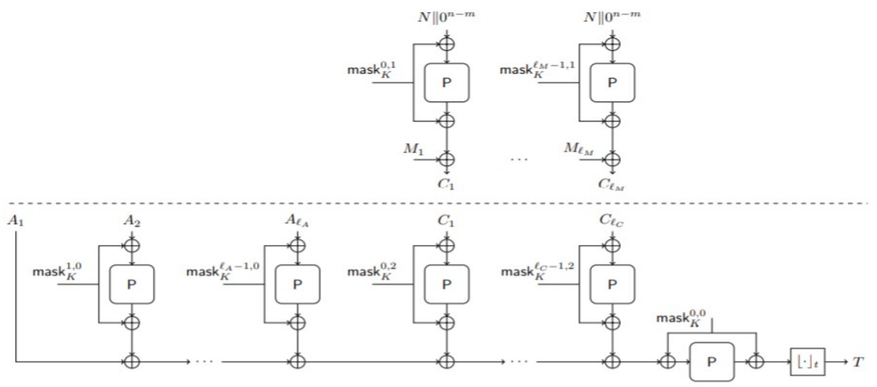


Figure 1: Depiction of Elephant. For the encryption part (top): message is padded as $M_1 \dots M_{\ell_M} \xleftarrow{n} M$, and ciphertext equals $C = \lfloor C_1 \dots C_{\ell_M} \rfloor_{|M|}$. For the authentication part (bottom): nonce and associated data are padded as $A_1 \dots A_{\ell_A} \xleftarrow{n} N\|A\|1$, and ciphertext is padded as $C_1 \dots C_{\ell_C} \xleftarrow{n} C\|1$.

160-Bit Permutation and LFSR:

The Spongent-[160] permutation is defined by this. The LFSR with 160-bit masking is defined in the table below. Dumbo makes use of these parts.

$$(x_0, \dots, x_{19}) \mapsto (x_1, \dots, x_{19}, x_0 \lll 3 \oplus x_3 \ll 7 \oplus x_{13} \gg 7)\,.$$

Spongent Permutation:

Spongent-π[160] denotes: {0, 1}^160 → The 80-round Spongent permutation is {0, 1}^160. It works as follows with a 160-bit input X:

```
for $i = 1, \dots, 80$ do
    $X \leftarrow X \oplus 0^{153}\|\text{lCounter}_{160}(i) \oplus \text{rev}(0^{153}\|\text{lCounter}_{160}(i))$
    $X \leftarrow \text{sBoxLayer}_{160}(X)$
    $X \leftarrow \text{pLayer}_{160}(X)$
```

where rev reverses the order of the bits in its input and lCounter160, sBoxLayer160, and pLayer160 are defined as follows:

- lCounter160: This is a 7-bit LFSR initialized with "1110101" and defined by the primitive polynomial p(x) = x7 + x6 + 1;

- sBoxLayer160: This function is made up of 40 S-box S : {0, 1}^4 → {0, 1}^4 s applied in a row. This S-box is written in hexadecimal as:

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(X)$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

- pLayer160: This function shifts the j-th bit of its input to P160(j), where

$$P_{160}(j) = \begin{cases} 40 \cdot j \bmod 159\,, & \text{if } j \in \{0,\dots,158\}\,, \\ 159\,, & \text{if } j = 159\,. \end{cases}$$

176-Bit Permutation and LFSR:

The Spongent-[176] permutation is defined by it. The LFSR with 176-bit masking is defined below. Jumbo makes use of these components.

$$(x_0,\dots,x_{21}) \mapsto (x_1,\dots,x_{21}, x_0 \lll 1 \oplus x_3 \ll 7 \oplus x_{19} \gg 7)\,.$$

Spongent Permutation:

Spongent-[176]: {0, 1}^176 → {0, 1}^176 denotes the 90-round Spongent permutation. It works with a 176-bit input X in the following way:

```
for $i = 1,\dots,90$ do
    $X \leftarrow X \oplus 0^{169} \| \mathsf{lCounter}_{176}(i) \oplus \mathsf{rev}(0^{169} \| \mathsf{lCounter}_{176}(i))$
    $X \leftarrow \mathsf{sBoxLayer}_{176}(X)$
    $X \leftarrow \mathsf{pLayer}_{176}(X)$
```

where the function rev, like previously, reverses the order of the bits in its argument. The function lCounter176 is the same as lCounter160 but is initialized with "1000101," the function sBoxLayer176 is the function S from the previous section applied 44 times in parallel, and the function pLayer176 is now defined as the function that moves the j-th bit of its input to bit position P176(j).

$$P_{176}(j) = \begin{cases} 44 \cdot j \bmod 175\,, & \text{if } j \in \{0,\dots,174\}\,, \\ 175\,, & \text{if } j = 175\,. \end{cases}$$

200-Bit Permutation and LFSR:

The LFSR with 200-bit masking is defined in the table below. Delirium makes use of these components.

$$(x_0,\dots,x_{24}) \mapsto (x_1,\dots,x_{24}, x_0 \lll 1 \oplus x_2 \lll 1 \oplus x_{13} \ll 1)\,.$$

Keccak Permutation:

On a 200-bit input X, Keccak-f[200] works as follows:

```
for $i = 1,\dots,18$ do
    $X \leftarrow \iota \circ \chi \circ \pi \circ \rho \circ \theta(X)$
```

where the functions θ, ρ, π, χ, and ι are defined as follows:

$$\theta\colon a[x,y,z] \leftarrow a[x,y,z] \oplus \bigoplus_{y'=0}^{4} a[x-1,y',z] \oplus \bigoplus_{y'=0}^{4} a[x+1,y',z-1]\,,$$

$$\rho\colon a[x,y,z] \leftarrow a[x,y,z+t[x,y]]\,,$$

$$\pi\colon a[x,y,z] \leftarrow a[x+3y,x,z]\,,$$

$$\chi\colon a[x,y,z] \leftarrow a[x,y,z] \oplus (a[x+1,y,z] \oplus 1) a[x+2,y,z]\,,$$

$$\iota\colon a[x,y,z] \leftarrow a[x,y,z] \oplus RC[i,x,y,z]\,.$$

For ρ, the function t[x, y] is defined as:

t	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	153	231	3	10	171
$y=1$	55	276	36	300	6
$y=0$	28	91	0	1	190
$y=4$	120	78	210	66	253
$y=3$	21	136	105	45	15

and for i, the round constants are given by:

$$RC[i,x,y,z] = \begin{cases} rc[j+7i]\,, & \text{if } (x,y,z) = (0,0,2^j-1)\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$$

where rc is computed from a binary LFSR defined by the primitive polynomial p(x) = x^8 + x^6 + x^5 + x^4 + 1.

Parameterization of Elephant:

Elephant is made up of three instances, the first two of which are created by utilizing the permutation and LFSR to instantiate the mode. We narrow our emphasis to n ∈ {160, 176, 200}. We also set m = 96, which means we only allow nonces with a length of 96 bits. The parameters k, t, and N remain adjustable. We offer the following three Elephants (with Dumbo as the primary member):

instance	k	m	n	t	P	φ_1	expected security strength	limit on online complexity
Dumbo	128	96	160	64	Spongent-π[160]	(3)	2^{112}	$2^{50}/(n/8)$
Jumbo	128	96	176	64	Spongent-π[176]	(4)	2^{127}	$2^{50}/(n/8)$
Delirium	128	96	200	128	Keccak-f[200]	(5)	2^{127}	$2^{74}/(n/8)$

Conclusion:

Elephant is a family of authenticated encryption algorithms based around the Spongent-pi and Keccak permutations.

- •Dumbo has a 128-bit key, a 96-bit nonce, and a 64-bit authentication tag. It is based around the Spongent-π[160] permutation.
- •Jumbo has a 128-bit key, a 96-bit nonce, and a 64-bit authentication tag. It is based around the Spongent-π[176] permutation.
- •Delirium has a 128-bit key, a 96-bit nonce, and a 128-bit authentication tag. It is based around the Keccak[200] permutation.

ROMULUS:

Introduction:

Romulus is an authenticated encryption with associated data (AEAD) technique based on a tweakable block cipher (TBC) Skinny, as described in this introduction. Romulus is divided into two families: Romulus-N, a nonce-based AE (NAE), and Romulus-M, a nonce-misuse-resistant AE (MRAE).

TBCs have been recognized as a useful primitive because they may be used to build simple and secure NAE/MRAE systems, such as ΘCB3 and SCT. While these techniques are computationally efficient (in terms of the amount of primitive calls) and secure, lightweight applications are not their primary use cases, and they are not well-suited to small devices. With this in mind, Romulus aspires towards TBC-based NAE and MRAE schemes that are lightweight, efficient, and highly secure.

Although Romulus-overall N's structure is comparable to a (TBC-based variant of) block cipher mode COFB, we make significant changes to reach our design aim. Because of the faster MAC computation for associated data, Romulus-N requires fewer TBC calls than ΘCB3, and the hardware implementation is much smaller than ΘCB3 due to the lower state size and inverse-freeness (i.e., TBC inverse is not needed). In reality, the state size of Romulus-N is exactly what is required for computing TBC. It also encrypts an n-bit plaintext block with only one call to the n-bit block TBC, resulting in no efficiency loss. For small messages, Romulus-N is extremely efficient, which is especially important in many lightweight applications, requiring only two TBC calls to handle one associated data block and one message block (in comparison, other designs such as ΘCB3 OCB3, TAE, and CCM require from three to five TBC calls in the same situation). Romulus-N achieves these benefits without sacrificing security, i.e., Romulus-N has complete n-bit security, which is comparable to ΘCB3.

The state size of Romulus-N is comparable to other size-oriented and n-bit secure AE systems, such as typical permutation-based AEs using 3n-bit permutation with n-bit rate (3n to 3.5n bits). Because of the decreased output size, the underlying cryptographic primitive is predicted to be substantially more lightweight and/or faster (3n vs n bits). Furthermore, the n-bit security of Romulus-N is demonstrated using the standard model, which guarantees security not only mathematically but qualitatively. To elaborate, a security proof in the standard model allows one to precisely tie the primitive's security status to the overall security of the mode that employs it. In our situation, the best attack on each member of Romulus is a chosen-plaintext attack (CPA) in the single-key configuration against Skinny, i.e., Romulus retains the stated n-bit security unless Skinny is cracked by CPA adversaries in the single-key setting. With non-standard models, such a guarantee is impossible, and it's sometimes difficult to determine the influence of a discovered "flaw" in the primitive on the mode's security. In a broader sense, "uninstantiable" Random Oracle-Model schemes best exemplify the gap between the proof and the actual security.

Parameters:

Romulus has the following parameters:

- Nonce length nl $\in$ {96, 128}.
- Key length k = 128.
- Message block length n = 128.
- Counter bit length d $\in$ {24, 56, 48}.

- AD block length n + t, where t ∈ {96, 128}.
- Tag length τ = 128.
- A TBC E’ : K × T’ × M → M, where K = {0, 1}^k , M = {0, 1}^n , and T’ = T × B × D. Here, T = {0, 1}^t , D = [[2^d − 1]]$_0$, and B = [[256]]$_0$ for parameters t and d, and B is also represented as a byte. For tweak T’ = (T, B, D) ∈ T’ , T is always assumed to be a byte string including ε, and t is a multiple of 8. E’ is either Skinny-128-384 or Skinny-128-256 with appropriate tweakey encoding functions.

Members of Romulus:

Families of NAE and MRAE. Romulus is divided into two families: Romulus-N and Romulus-M, each with multiple members (the sets of parameters). The former uses nonce-based AE (NAE) to protect against Nonce-respecting opponents, whereas the latter uses Rogaway and Shrimpton's nonce Misuseresistant AE (MRAE). The name Romulus refers to a pair of families.

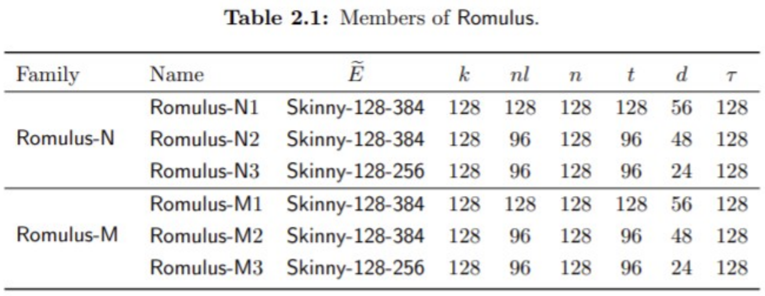
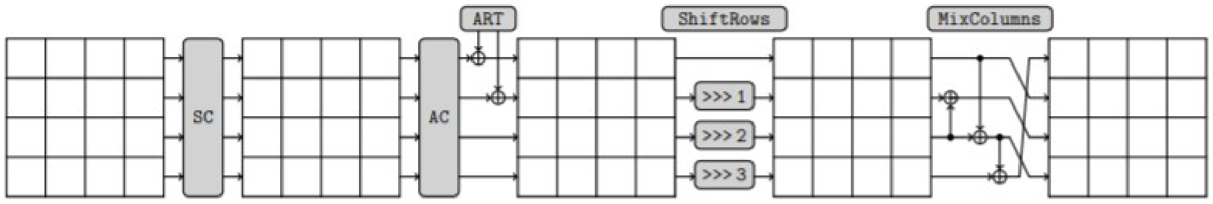


Table 2.1: Members of Romulus.

Family	Name	$\widetilde{E}$	k	nl	n	t	d	τ
Romulus-N	Romulus-N1	Skinny-128-384	128	128	128	128	56	128
	Romulus-N2	Skinny-128-384	128	96	128	96	48	128
	Romulus-N3	Skinny-128-256	128	96	128	96	24	128
Romulus-M	Romulus-M1	Skinny-128-384	128	128	128	128	56	128
	Romulus-M2	Skinny-128-384	128	96	128	96	48	128
	Romulus-M3	Skinny-128-256	128	96	128	96	24	128

The Round Function:

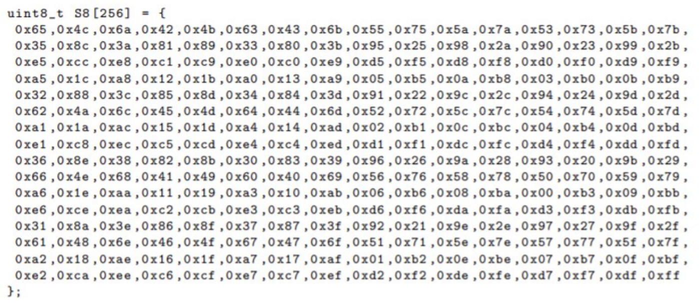
Skinny-128-256 and Skinny-128-384 have 48 and 56 rounds, respectively. SubCells(SC), AddConstants(AC), AddRoundTweakey(ART), ShiftRows(SR), and MixColumns(MC) are the five operations that make up an encryption round.



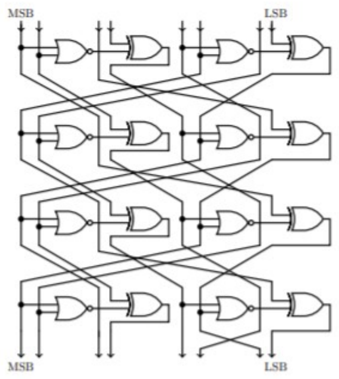
SubCells(SC):

Every cell of the cipher internal state receives an 8-bit Sbox. This Sbox's behavior is described in hexadecimal notation below.

8-bit Sbox S8 used in Skinny when s = 8



```
uint8_t  S8[256] = {
 0x65 ,0x4c ,0x6a ,0x42 ,0x4b ,0x63 ,0x43 ,0x6b ,0x55 ,0x75 ,0x5a ,0x7a ,0x53 ,0x73 ,0x5b ,0x7b ,
 0x35 ,0x8c ,0x3a ,0x81 ,0x89 ,0x33 ,0x80 ,0x3b ,0x95 ,0x25 ,0x98 ,0x2a ,0x90 ,0x23 ,0x99 ,0x2b ,
 0xe5 ,0xcc ,0xe8 ,0xc1 ,0xc9 ,0xe0 ,0xc0 ,0xe9 ,0xd5 ,0xf5 ,0xd8 ,0xf8 ,0xd0 ,0xf0 ,0xd9 ,0xf9 ,
 0xa5 ,0x1c ,0xa8 ,0x12 ,0x1b ,0xa0 ,0x13 ,0xa9 ,0x05 ,0xb5 ,0x0a ,0xb8 ,0x03 ,0xb0 ,0x0b ,0xb9 ,
 0x32 ,0x88 ,0x3c ,0x85 ,0x8d ,0x34 ,0x84 ,0x3d ,0x91 ,0x22 ,0x9c ,0x2c ,0x94 ,0x24 ,0x9d ,0x2d ,
 0x62 ,0x4a ,0x6c ,0x45 ,0x4d ,0x64 ,0x44 ,0x6d ,0x52 ,0x72 ,0x5c ,0x7c ,0x54 ,0x74 ,0x5d ,0x7d ,
 0xa1 ,0x1a ,0xac ,0x15 ,0x1d ,0xa4 ,0x14 ,0xad ,0x02 ,0xb1 ,0x0c ,0xbc ,0x04 ,0xb4 ,0x0d ,0xbd ,
 0xe1 ,0xc8 ,0xec ,0xc5 ,0xcd ,0xe4 ,0xc4 ,0xed ,0xd1 ,0xf1 ,0xdc ,0xfc ,0xd4 ,0xf4 ,0xdd ,0xfd ,
 0x36 ,0x8e ,0x38 ,0x82 ,0x8b ,0x30 ,0x83 ,0x39 ,0x96 ,0x26 ,0x9a ,0x28 ,0x93 ,0x20 ,0x9b ,0x29 ,
 0x66 ,0x4e ,0x68 ,0x41 ,0x49 ,0x60 ,0x40 ,0x69 ,0x56 ,0x76 ,0x58 ,0x78 ,0x50 ,0x70 ,0x59 ,0x79 ,
 0xa6 ,0x1e ,0xaa ,0x11 ,0x19 ,0xa3 ,0x10 ,0xab ,0x06 ,0xb6 ,0x08 ,0xba ,0x00 ,0xb3 ,0x09 ,0xbb ,
 0xe6 ,0xce ,0xea ,0xc2 ,0xcb ,0xe3 ,0xc3 ,0xeb ,0xd6 ,0xf6 ,0xda ,0xfa ,0xd3 ,0xf3 ,0xdb ,0xfb ,
 0x31 ,0x8a ,0x3e ,0x86 ,0x8f ,0x37 ,0x87 ,0x3f ,0x92 ,0x21 ,0x9e ,0x2e ,0x97 ,0x27 ,0x9f ,0x2f ,
 0x61 ,0x48 ,0x6e ,0x46 ,0x4f ,0x67 ,0x47 ,0x6f ,0x51 ,0x71 ,0x5e ,0x7e ,0x57 ,0x77 ,0x5f ,0x7f ,
 0xa2 ,0x18 ,0xae ,0x16 ,0x1f ,0xa7 ,0x17 ,0xaf ,0x01 ,0xb2 ,0x0e ,0xbe ,0x07 ,0xb7 ,0x0f ,0xbf ,
 0xe2 ,0xca ,0xee ,0xc6 ,0xcf ,0xe7 ,0xc7 ,0xef ,0xd2 ,0xf2 ,0xde ,0xfe ,0xd7 ,0xf7 ,0xdf ,0xff
};
```



AddConstants(AC):

To generate round constants, a 6-bit affine LFSR with the state (rc5, rc4, rc3, rc2, rc1, rc0) (with rc0 being the least significant bit) is utilized. It has the following update function:

$$(rc5||rc4||rc3||rc2||rc1||rc0) \rightarrow (rc4||rc3||rc2||rc1||rc0||rc5 \oplus rc4 \oplus 1).$$

The six bits are set to zero and then updated before being used in a round. Depending on the size of the internal state, the bits from the LFSR are ordered into a 4 x 4 array (only the first column of the state is influenced by the LFSR bits).

$$\begin{bmatrix} c_0 & 0 & 0 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

,with c2 = 0x2 and (c0, c1) = (rc3||rc2||rc1||rc0, 0||0||rc5||rc4) when s = 4
(c0, c1) = (0||0||0||0||rc3||rc2||rc1||rc0, 0||0||0||0||0||0||rc5||rc4) when s = 8.

Using bitwise exclusive-or, the round constants are joined with the state while keeping array positioning in mind. The values of the (rc5,rc4,rc3,rc2,rc1,rc0) constants are encoded to byte values for each round in the table below, with rc0 being the least significant bit.

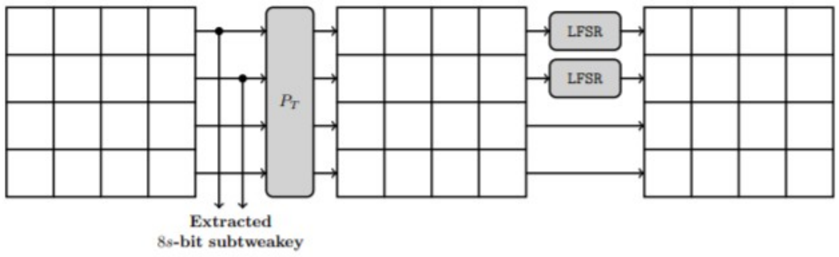
Rounds	Constants
1 - 16	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E
17 - 32	1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38
33 - 48	31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04
49 - 62	09,13,26,0C,19,32,25,0A,15,2A,14,28,10,20

AddRoundTweakey(ART):

All tweakey arrays' first and second rows are taken and bitwise exclusive-ored to the cipher internal state while maintaining array placement. For i = {0, 1} and j = {0, 1, 2, 3}, the formal expression is:

- $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j}$ when z = 2,
- $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j} \oplus TK3_{i,j}$ when z = 3.

The tweakey schedule in Skinny. Each tweakey word TK1, TK2 and TK3 (if any) follows a similar transformation update, except that no LFSR is applied to TK1.



ShiftRows(SR):

The rows of the cipher state cell array in this layer are rotated to the right, much like in AES. The second, third, and fourth cell rows are rotated to the right by 1, 2, and 3 positions, respectively. In other words, the cells positions of the cipher internal state cell array are permuted P: for every $0 \le i \le 15$, we set $IS_i \leftarrow IS_{P[i]}$ with,

$$P = [0, 1, 2, 3, 7, 4, 5, 6, 10, 11, 8, 9, 13, 14, 15, 12].$$

MixColumns(MC):
The following binary matrix M is multiplied by each column of the cipher internal state array:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

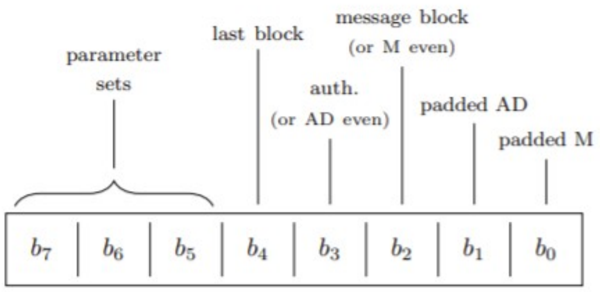
The Authenticated Encryption Romulus:

-The Tweakey Encoding:

Domain separation:
Block cipher calls and the various versions of Romulus, we'll employ a domain separation byte B. The bitwise representation of this byte is B = (b7||b6k||b5||b4||b3||b2||b1||b0), where b7 is the MSB and b0 is the LSB.

- b7 b6 b5 will specify the parameter sets. They are fixed to:
• 000 for Romulus-N1
• 001 for Romulus-M1
• 010 for Romulus-N2
• 011 for Romulus-M2
• 100 for Romulus-N3
• 101 for Romulus-M3

Domain separation when using the tweakable block cipher:



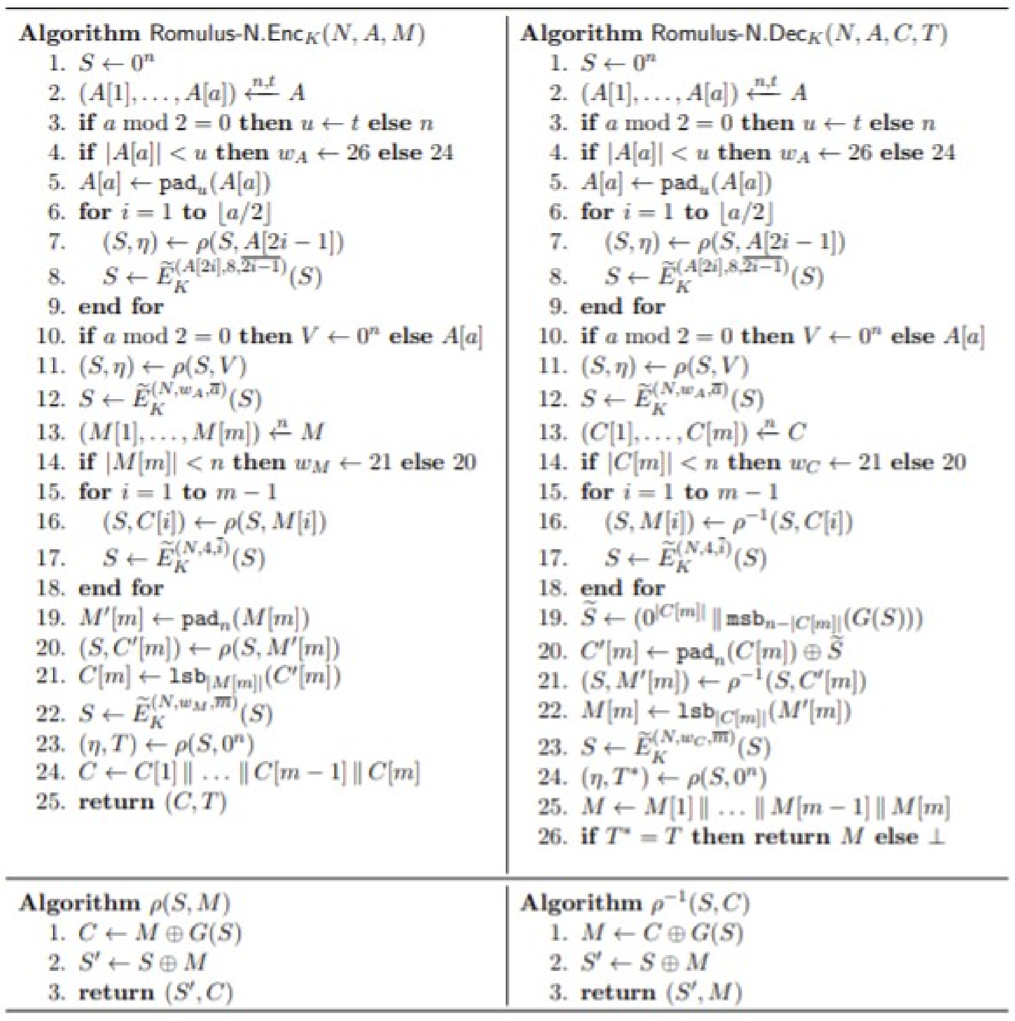
LFSR:

For counters, we utilize LFSRs. lfsrc is a one-to-one mapping for positive integer c lfsrc: $[[2\ c - 1]]_0 \rightarrow$ {0, 1}^c \ {0^c} defined as follows. For positive integer c, let $F_c(x)$ be the lexicographically-first polynomial among the the irreducible degree c polynomials of a minimum number of coefficients. Specifically $F_c(x)$ for c $\in$ {56, 24} are

$F_{56}(x)$ = x^56 + x^7 + x^4 + x^2 + 1,
$F_{24}(x)$ = x^24 + x^4 + x^3 + x + 1,

$\mathrm{lfsr}_c(D)$ = (2^D) mod $F_c(x)$

Encryption and Decryption of Romulus-N:



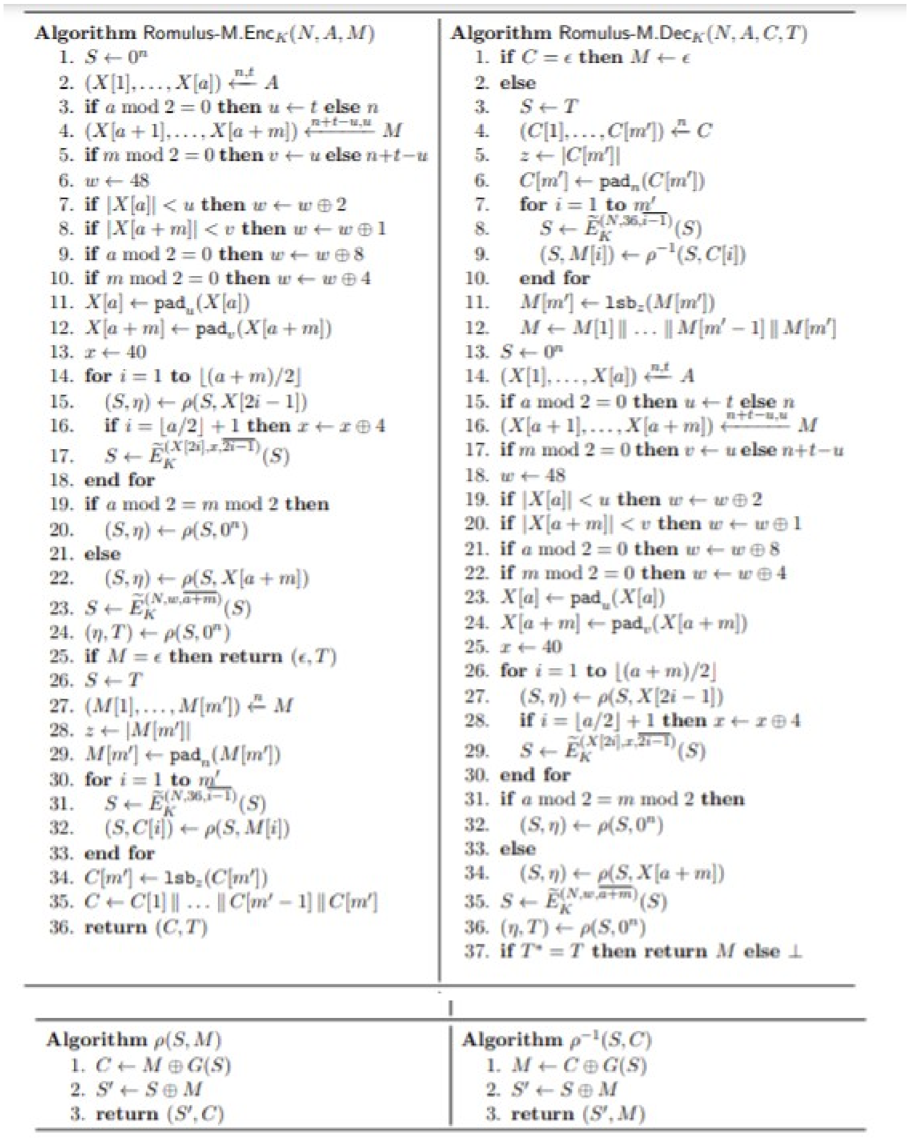
```
Algorithm Romulus-N.Enc_K(N, A, M)
 1. $S \leftarrow 0^n$
 2. $(A[1], \dots, A[a]) \xleftarrow{n,t} A$
 3. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
 4. if $|A[a]| < u$ then $w_A \leftarrow 26$ else 24
 5. $A[a] \leftarrow \mathsf{pad}_u(A[a])$
 6. for $i = 1$ to $\lfloor a/2 \rfloor$
 7.     $(S, \eta) \leftarrow \rho(S, A[2i-1])$
 8.     $S \leftarrow \widetilde{E}_K^{(A[2i], 8, \overline{2i-1})}(S)$
 9. end for
10. if $a \bmod 2 = 0$ then $V \leftarrow 0^n$ else $A[a]$
11. $(S, \eta) \leftarrow \rho(S, V)$
12. $S \leftarrow \widetilde{E}_K^{(N, w_A, \overline{a})}(S)$
13. $(M[1], \dots, M[m]) \xleftarrow{n} M$
14. if $|M[m]| < n$ then $w_M \leftarrow 21$ else 20
15. for $i = 1$ to $m - 1$
16.     $(S, C[i]) \leftarrow \rho(S, M[i])$
17.     $S \leftarrow \widetilde{E}_K^{(N, 4, \overline{i})}(S)$
18. end for
19. $M'[m] \leftarrow \mathsf{pad}_n(M[m])$
20. $(S, C'[m]) \leftarrow \rho(S, M'[m])$
21. $C[m] \leftarrow \mathtt{lsb}_{|M[m]|}(C'[m])$
22. $S \leftarrow \widetilde{E}_K^{(N, w_M, \overline{m})}(S)$
23. $(\eta, T) \leftarrow \rho(S, 0^n)$
24. $C \leftarrow C[1] \parallel \dots \parallel C[m-1] \parallel C[m]$
25. return $(C, T)$
```

```
Algorithm Romulus-N.Dec_K(N, A, C, T)
 1. $S \leftarrow 0^n$
 2. $(A[1], \dots, A[a]) \xleftarrow{n,t} A$
 3. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
 4. if $|A[a]| < u$ then $w_A \leftarrow 26$ else 24
 5. $A[a] \leftarrow \mathsf{pad}_u(A[a])$
 6. for $i = 1$ to $\lfloor a/2 \rfloor$
 7.     $(S, \eta) \leftarrow \rho(S, A[2i-1])$
 8.     $S \leftarrow \widetilde{E}_K^{(A[2i], 8, \overline{2i-1})}(S)$
 9. end for
10. if $a \bmod 2 = 0$ then $V \leftarrow 0^n$ else $A[a]$
11. $(S, \eta) \leftarrow \rho(S, V)$
12. $S \leftarrow \widetilde{E}_K^{(N, w_A, \overline{a})}(S)$
13. $(C[1], \dots, C[m]) \xleftarrow{n} C$
14. if $|C[m]| < n$ then $w_C \leftarrow 21$ else 20
15. for $i = 1$ to $m - 1$
16.     $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
17.     $S \leftarrow \widetilde{E}_K^{(N, 4, \overline{i})}(S)$
18. end for
19. $\widetilde{S} \leftarrow (0^{|C[m]|} \parallel \mathtt{msb}_{n-|C[m]|}(G(S)))$
20. $C'[m] \leftarrow \mathsf{pad}_n(C[m]) \oplus \widetilde{S}$
21. $(S, M'[m]) \leftarrow \rho^{-1}(S, C'[m])$
22. $M[m] \leftarrow \mathtt{lsb}_{|C[m]|}(M'[m])$
23. $S \leftarrow \widetilde{E}_K^{(N, w_C, \overline{m})}(S)$
24. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
25. $M \leftarrow M[1] \parallel \dots \parallel M[m-1] \parallel M[m]$
26. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm $\rho(S, M)$
 1. $C \leftarrow M \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', C)$
```

```
Algorithm $\rho^{-1}(S, C)$
 1. $M \leftarrow C \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', M)$
```

Encryption and Decryption of Romulus-M:



```
**Algorithm** Romulus-M.Enc$_K(N, A, M)$
 1. $S \leftarrow 0^n$
 2. $(X[1], \ldots, X[a]) \xleftarrow{n,t} A$
 3. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
 4. $(X[a+1], \ldots, X[a+m]) \xleftarrow{n+t-u,u} M$
 5. if $m \bmod 2 = 0$ then $v \leftarrow u$ else $n+t-u$
 6. $w \leftarrow 48$
 7. if $|X[a]| < u$ then $w \leftarrow w \oplus 2$
 8. if $|X[a+m]| < v$ then $w \leftarrow w \oplus 1$
 9. if $a \bmod 2 = 0$ then $w \leftarrow w \oplus 8$
10. if $m \bmod 2 = 0$ then $w \leftarrow w \oplus 4$
11. $X[a] \leftarrow \mathsf{pad}_u(X[a])$
12. $X[a+m] \leftarrow \mathsf{pad}_v(X[a+m])$
13. $x \leftarrow 40$
14. for $i = 1$ to $\lfloor (a+m)/2 \rfloor$
15.    $(S, \eta) \leftarrow \rho(S, X[2i-1])$
16.    if $i = \lfloor a/2 \rfloor + 1$ then $x \leftarrow x \oplus 4$
17.    $S \leftarrow \widetilde{E}_K^{(X[2i], x, \overline{2i-1})}(S)$
18. end for
19. if $a \bmod 2 = m \bmod 2$ then
20.    $(S, \eta) \leftarrow \rho(S, 0^n)$
21. else
22.    $(S, \eta) \leftarrow \rho(S, X[a+m])$
23. $S \leftarrow \widetilde{E}_K^{(N, w, \overline{a+m})}(S)$
24. $(\eta, T) \leftarrow \rho(S, 0^n)$
25. if $M = \epsilon$ then return $(\epsilon, T)$
26. $S \leftarrow T$
27. $(M[1], \ldots, M[m']) \xleftarrow{n} M$
28. $z \leftarrow |M[m']|$
29. $M[m'] \leftarrow \mathsf{pad}_n(M[m'])$
30. for $i = 1$ to $m'$
31.    $S \leftarrow \widetilde{E}_K^{(N, 36, \overline{i-1})}(S)$
32.    $(S, C[i]) \leftarrow \rho(S, M[i])$
33. end for
34. $C[m'] \leftarrow \mathtt{lsb}_z(C[m'])$
35. $C \leftarrow C[1] \parallel \ldots \parallel C[m'-1] \parallel C[m']$
36. return $(C, T)$
```

```
**Algorithm** Romulus-M.Dec$_K(N, A, C, T)$
 1. if $C = \epsilon$ then $M \leftarrow \epsilon$
 2. else
 3.    $S \leftarrow T$
 4.    $(C[1], \ldots, C[m']) \xleftarrow{n} C$
 5.    $z \leftarrow |C[m']|$
 6.    $C[m'] \leftarrow \mathsf{pad}_n(C[m'])$
 7.    for $i = 1$ to $m'$
 8.       $S \leftarrow \widetilde{E}_K^{(N, 36, \overline{i-1})}(S)$
 9.       $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
10.    end for
11.    $M[m'] \leftarrow \mathtt{lsb}_z(M[m'])$
12.    $M \leftarrow M[1] \parallel \ldots \parallel M[m'-1] \parallel M[m']$
13. $S \leftarrow 0^n$
14. $(X[1], \ldots, X[a]) \xleftarrow{n,t} A$
15. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
16. $(X[a+1], \ldots, X[a+m]) \xleftarrow{n+t-u,u} M$
17. if $m \bmod 2 = 0$ then $v \leftarrow u$ else $n+t-u$
18. $w \leftarrow 48$
19. if $|X[a]| < u$ then $w \leftarrow w \oplus 2$
20. if $|X[a+m]| < v$ then $w \leftarrow w \oplus 1$
21. if $a \bmod 2 = 0$ then $w \leftarrow w \oplus 8$
22. if $m \bmod 2 = 0$ then $w \leftarrow w \oplus 4$
23. $X[a] \leftarrow \mathsf{pad}_u(X[a])$
24. $X[a+m] \leftarrow \mathsf{pad}_v(X[a+m])$
25. $x \leftarrow 40$
26. for $i = 1$ to $\lfloor (a+m)/2 \rfloor$
27.    $(S, \eta) \leftarrow \rho(S, X[2i-1])$
28.    if $i = \lfloor a/2 \rfloor + 1$ then $x \leftarrow x \oplus 4$
29.    $S \leftarrow \widetilde{E}_K^{(X[2i], x, \overline{2i-1})}(S)$
30. end for
31. if $a \bmod 2 = m \bmod 2$ then
32.    $(S, \eta) \leftarrow \rho(S, 0^n)$
33. else
34.    $(S, \eta) \leftarrow \rho(S, X[a+m])$
35. $S \leftarrow \widetilde{E}_K^{(N, w, \overline{a+m})}(S)$
36. $(\eta, T) \leftarrow \rho(S, 0^n)$
37. if $T^* = T$ then return $M$ else $\perp$
```

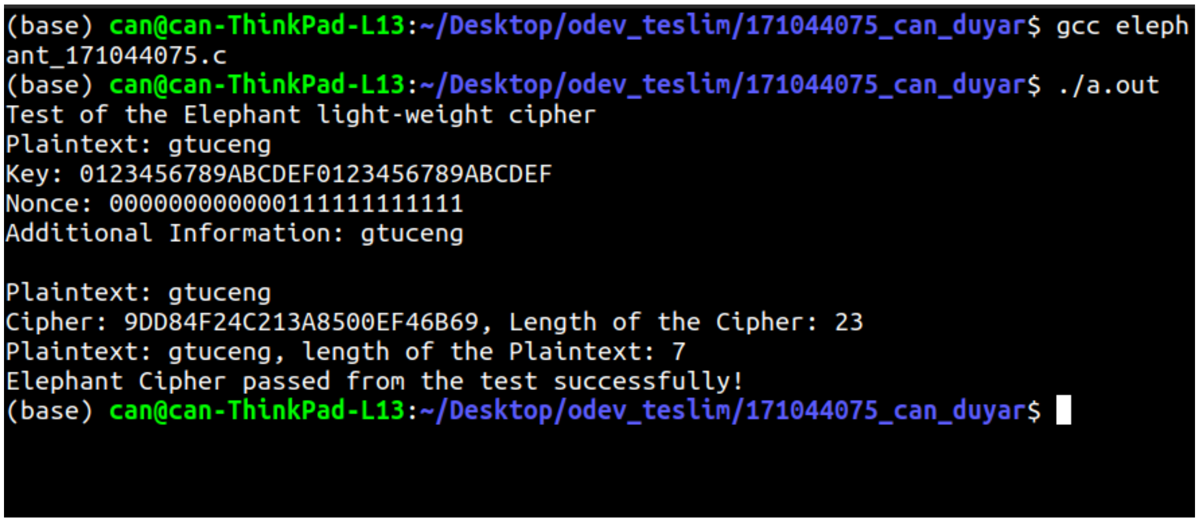
```
**Algorithm** $\rho(S, M)$
 1. $C \leftarrow M \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', C)$
```

```
**Algorithm** $\rho^{-1}(S, C)$
 1. $M \leftarrow C \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', M)$
```

Conclusion:

The NIST lightweight cryptography competition has reached the final round, with a shortlist of ten contenders. Each differs in their approach, but they aim to create a cryptography method that is secure, has a low footprint, and is robust against attacks. So while many of the contenders, such as ASCON, GIFT and Isap, use the sponge method derived from the SHA-3 standard (Keccak), Romulus takes a more traditional approach and looks towards a more traditional light-weight crypto approach. Overall it is defined as a tweakable block cipher (TBC) and which supports authenticated encryption with associated data (AEAD). For its more traditional approach, it uses the Skinny lightweight tweakable block cipher.

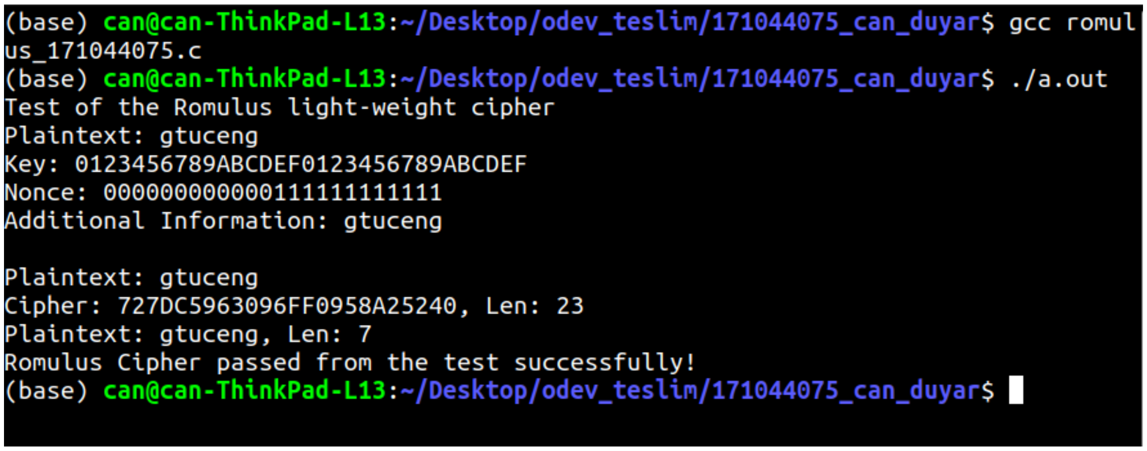
Output of the Elephant light-weight cipher:



```
(base) can@can-ThinkPad-L13:~/Desktop/odev_teslim/171044075_can_duyar$ gcc eleph
ant_171044075.c
(base) can@can-ThinkPad-L13:~/Desktop/odev_teslim/171044075_can_duyar$ ./a.out
Test of the Elephant light-weight cipher
Plaintext: gtuceng
Key: 0123456789ABCDEF0123456789ABCDEF
Nonce: 000000000000111111111111
Additional Information: gtuceng

Plaintext: gtuceng
Cipher: 9DD84F24C213A8500EF46B69, Length of the Cipher: 23
Plaintext: gtuceng, length of the Plaintext: 7
Elephant Cipher passed from the test successfully!
(base) can@can-ThinkPad-L13:~/Desktop/odev_teslim/171044075_can_duyar$
```

Output of the Romulus light-weight cipher:



```
(base) can@can-ThinkPad-L13:~/Desktop/odev_teslim/171044075_can_duyar$ gcc romul
us_171044075.c
(base) can@can-ThinkPad-L13:~/Desktop/odev_teslim/171044075_can_duyar$ ./a.out
Test of the Romulus light-weight cipher
Plaintext: gtuceng
Key: 0123456789ABCDEF0123456789ABCDEF
Nonce: 000000000000111111111111
Additional Information: gtuceng

Plaintext: gtuceng
Cipher: 727DC5963096FF0958A25240, Len: 23
Plaintext: gtuceng, Len: 7
Romulus Cipher passed from the test successfully!
(base) can@can-ThinkPad-L13:~/Desktop/odev_teslim/171044075_can_duyar$
```