CLX: A Family of Lightweight Authenticated Encryption Algorithms

Designers and Submitters: Hongjun Wu and Tao Huang

Division of Mathematical Sciences
Nanyang Technological University
wuhongjun@gmail.com

29 March 2019

Contents

1 Specifications **3**
- 1.1 Recommended parameter sets . . . 3
- 1.2 Operations, Variables and Functions . . . 4
  - 1.2.1 Operations . . . 4
  - 1.2.2 Variables and Constants . . . 4
  - 1.2.3 The Permutation . . . 5
- 1.3 CLX-128 . . . 6
  - 1.3.1 The initialization . . . 6
  - 1.3.2 Processing the associated data . . . 7
  - 1.3.3 The encryption . . . 7
  - 1.3.4 The finalization . . . 8
  - 1.3.5 The decryption . . . 8
  - 1.3.6 The verification . . . 9
- 1.4 CLX Authenticated Encryption Algorithms . . . 9
  - 1.4.1 The initialization . . . 10
  - 1.4.2 Processing the associated data . . . 10
  - 1.4.3 The encryption . . . 11
  - 1.4.4 The finalization . . . 11
  - 1.4.5 The decryption . . . 11
  - 1.4.6 The verification . . . 12
- 1.5 CLX-Hash . . . 12
  - 1.5.1 The permutation $P'_{288,n}$ . . . 12
  - 1.5.2 The initialization . . . 13
  - 1.5.3 Processing the message . . . 13
  - 1.5.4 The finalization . . . 13

2 Security Goals **15**
- 2.1 Security goals with unique nonce . . . 15
- 2.2 Security goals with repeated nonce . . . 15
- 2.3 Security goals of hash function . . . 16

3 Security Analysis **17**
3.1 Security of the Duplex Mode 17
3.2 Properties of the Permutation $P_{160+x,n}$ 17
3.2.1 Differential properties of the permutation 17
3.2.2 Linear properties of the permutation $P_{160+x,n}$ 19
3.2.3 Algebraic properties of the permutation $P_{160+x,n}$ 20
3.3 Forgery Attacks 21
3.3.1 Forgery attacks on nonce and associated data 21
3.3.2 Forgery attacks on plaintext/ciphertext 21
3.4 State Recovery Based on State Collision 22
3.5 Key Recovery Attacks for Repeated Nonce 22
3.5.1 Differential cryptanalysis 23
3.5.2 Linear cryptanalysis 23
3.5.3 Algebraic attacks 23
3.6 Related-key Attacks 23
3.7 Slide attack 24
3.8 Security Analysis of CLX-Hash 24

4 The Performance of CLX **25**
4.1 Hardware Performance 25
4.2 Software Performance 25

5 Features **27**

6 Design Rationale **28**

Chapter 1

Specifications

CLX authenticated encryption algorithms and CLX hash function are specified in this chapter. CLX authenticated encryption algorithms are based on the Duplex mode [3], while CLX hash function is based on the sponge mode [2].

1.1 Recommended parameter sets

CLX Authenticated Encryption Algorithms. CLX supports three key sizes: 128 bits, 192 bits and 256 bits. In the name of an CLX algorithm, letter 'Q' indicates the fast variant, letter 'H' indicates the high security variant which protects the secret key when nonce is reused.

- Primary member: CLX-128
  128-bit key, 96-bit nonce, 64-bit tag, 160-bit state
  (for small state)
- CLX-128Q
  128-bit key, 96-bit nonce, 64-bit tag, 192-bit state
- CLX-128H
  128-bit key, 96-bit nonce, 64-bit tag, 192-bit state
- CLX-192Q
  192-bit key, 96-bit nonce, 64-bit tag, 256-bit state
- CLX-192H
  192-bit key, 96-bit nonce, 64-bit tag, 256-bit state
- CLX-256Q
  256-bit key, 96-bit nonce, 64-bit tag, 320-bit state
- CLX-256H
  256-bit key, 96-bit nonce, 64-bit tag, 320-bit state

CLX Hash Function. CLX provides a hash function with 256-bit message digest.

- Primary member: CLX-Hash
  256-bit message digest, 288-bit state

1.2 Operations, Variables and Functions

The operations, variables and functions used in CLX are defined below.

1.2.1 Operations

The following operations are used in the description of CLX:

$\oplus$	:	bit-wise exclusive OR
$\&$	:	bit-wise AND
$\sim$	:	bit-wise NOT
$\|\|$	:	concatenation
$\lfloor a \rfloor$	:	floor operator, gives the integer part of a

1.2.2 Variables and Constants

The following variables and constants are used in CLX:

$a_{\{i \cdots j\}}$	:	the word consists of $a_i \|\| a_{i+1} \|\| \cdots \| a_j$, where a_i is the ith bit of a
AD	:	associated data, a sequence of bytes
ad_i	:	one bit of associated data
$adlen$	:	the length of associated data in bits
C	:	ciphertext, a sequence of bytes
c_i	:	the ith ciphertext bit
$FrameBits$	:	Three-bit FrameBits
		FrameBits = 1 for nonce
		FrameBits = 3 for associated data
		FrameBits = 5 for plaintext and ciphertext
		FrameBits = 7 for finalization
$FrameBits_i$	:	The ith bit of FrameBits
K	:	the key
k_i	:	the ith bit of K

$klen$	:	the key length in bits
M	:	the plaintext, a sequence of bytes
m_i	:	the ith bit of the plaintext
MD	:	the message digest
md_i	:	the ith bit of the message digest
$mlen$	:	the length of the plaintext in bits
NONCE	:	the 96-bit nonce
nonce_i	:	the ith bit of the 96-bit nonce
$P_{w,n}$	:	the w-bit permutation with n rounds
S	:	the state of the permutation
s_i	:	the ith bit of the state of the permutation
T	:	the 64-bit authentication tag
t_i	:	the ith bit of the authentication tag
x	:	the state size is $160 + x$ bits

1.2.3 The Permutation

In CLX, permutation $P_{160+x,n}$ is used. The state of the permutation is (160+x)-bit, and the permutation consists of n rounds. In the ith round of the permutation, a (160+x)-bit nonlinear feedback shift register is used to update the state.

The state update function for 160-bit permutation (Fig. 1.1)

StateUpdate(S) for $x = 0$:
    feedback $= s_0 \oplus s_{35} \oplus (\sim (s_{93} \& s_{106})) \oplus s_{127}$
    for i from 0 to 158: $s_i = s_{i+1}$
    s_{159} = feedback
end

The state update function for (160+x)-bit permutation ($x > 0$) (Fig. 1.2)

StateUpdate(S) for $x > 0$:
    feedback $= s_0 \oplus s_x \oplus s_{35+x} \oplus (\sim (s_{93+x} \& s_{106+x})) \oplus s_{127+x}$
    for i from 0 to $(160 + x) - 2$: $s_i = s_{i+1}$
    $s_{(160+x)-1}$ = feedback
end

For example, $P_{160,320}$ means that the 160-bit state of the permutation is updated with 320 rounds. 32 rounds of the permutation can be computed in parallel on 32-bit CPU.

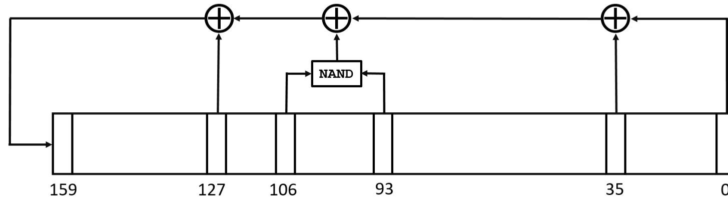

Figure 1.1: The 160-bit Nonlinear Feedback Shift Register in CLX

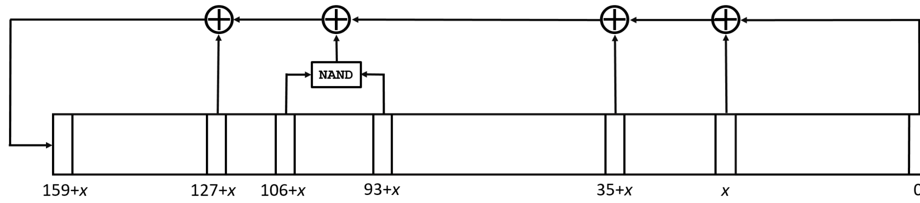

Figure 1.2: The $(160 + x)$-bit Nonlinear Feedback Shift Register in CLX

1.3 CLX-128

CLX-128 uses a 128-bit key and a 96-bit nonce. The state size is 160-bit. The associated data length and the plaintext length are less than 2^{50} bytes. The authentication tag is 64-bit.

1.3.1 The initialization

The initialization of CLX-128 consists of two stages: key setup and nonce setup.

Key Setup. The key setup is to randomize the state using the key.

1. Set the 160-bit state S as 0.
2. Set $s_{31} = 1$.
3. Set $s_{\{32,159\}} = k_{\{0,127\}}$.
4. Update the state using $P_{160,1152}$.

Nonce Setup. The nonce setup consists of three steps. In each step, the Framebits (of nonce) are XORed with the state, then we update the state using the permutation $P_{160,480}$, then 32 bits of the nonce are XORed with the state.

for i from 0 to 2:
  $s_{\{36\cdots 38\}} = s_{\{36\cdots 38\}} \oplus FrameBits_{\{0\cdots 2\}}$
  Update the state using $P_{160,384}$
  $s_{\{128\cdots 159\}} = s_{\{128\cdots 159\}} \oplus nonce_{\{32i\cdots 32i+31\}}$
end for

1.3.2 Processing the associated data

After the initialization, we process the associated data AD. In each step, the Framebits (of associated data) are XORed with the state, then we update the state using the permutation $P_{160,384}$, then 32 bits of the associated data are XORed with the state. If the last block is not a full block (or we call it as a partial block), the last block is padded with zeros, and the number of bytes of associated data in the partial block is XORed with the state.

/* processing the full blocks of associated data, each block 32 bits*/
for i from 0 to $\lfloor adlen/32 \rfloor$:
  $s_{\{68 \cdots 70\}} = s_{\{68 \cdots 70\}} \oplus FrameBits_{\{0 \cdots 2\}}$
  Update the state using $P_{160,384}$
  $s_{\{128 \cdots 159\}} = s_{\{128 \cdots 159\}} \oplus ad_{\{32i \cdots 32i+31\}}$
end for

/* processing the partial block (if exists) of associated data, less than 32 bits*/
if ($adlen$ mod 32) > 0:
  $s_{\{68 \cdots 70\}} = s_{\{68 \cdots 70\}} \oplus FrameBits_{\{0 \cdots 2\}}$
  Update the state using $P_{160,384}$
  $lenp = adlen$ mod 32 /* number of bits in partial block */
  $startp = adlen - lenp$ /* starting position of partial block */
  $s_{\{128 \cdots 128+lenp-1\}} = s_{\{128 \cdots 128+lenp-1\}} \oplus ad_{\{startp \cdots adlen-1\}}$
  /* the length (bytes) of the last partial block is XORed with the state*/
  $s_{\{64 \cdots 65\}} = s_{\{64 \cdots 65\}} \oplus (lenp/8)$
end if

1.3.3 The encryption

After processing the associated data, we encrypt the plaintext M. In each step, the Framebits (of plaintext) are XORed with the state. Then we update the state using the permutation $P_{160,960}$, then 32 bits of the plaintext are XORed with the state and we obtain 32 bits of ciphertext from the XORed state. If the last block is not a full block, the last block is padded with zeros, and the number of bytes of plaintext in the partial block is XORed with the state.

/* processing the full blocks of plaintext, each block 32 bits*/
for i from 0 to $\lfloor mlen/32 \rfloor$:
  $s_{\{68 \cdots 70\}} = s_{\{68 \cdots 70\}} \oplus FrameBits_{\{0 \cdots 2\}}$
  Update the state using $P_{160,1152}$
  $s_{\{128 \cdots 159\}} = s_{\{128 \cdots 159\}} \oplus m_{\{32i \cdots 32i+31\}}$
  $c_{\{32i \cdots 32i+31\}} = s_{\{128 \cdots 159\}}$
end for

/* processing the partial block (if exists) of plaintext, less than 32 bits*/

if ($mlen$ mod 32) > 0:

$s_{\{68 \cdots 70\}} = s_{\{68 \cdots 70\}} \oplus FrameBits_{\{0 \cdots 2\}}$

Update the state using $P_{160,1152}$

$lenp = mlen$ mod 32 /* number of bits in partial block */

$startp = mlen - lenp$ /* starting position of partial block */

$s_{\{128 \cdots 128+lenp-1\}} = s_{\{128 \cdots 128+lenp-1\}} \oplus m_{\{startp \cdots mlen-1\}}$

$c_{\{startp \cdots mlen-1\}} = s_{\{128 \cdots 128+lenp-1\}}$

/* the length (bytes) of the last partial block is XORed to the state*/

$s_{\{64 \cdots 65\}} = s_{\{64 \cdots 65\}} \oplus (lenp/8)$

end if

1.3.4 The finalization

After encrypting the plaintext, we generate the authentication tag T in two steps. In each step, the Framebits (of finalization) are XORed with the state, then we update the state using the permutation $P_{160,960}$, then obtain 32 bits of tag as part of the state.

for i from 0 to 1:

$s_{\{68 \cdots 70\}} = s_{\{68 \cdots 70\}} \oplus FrameBits_{\{0 \cdots 2\}}$

Update the state using $P_{160,1152}$

$t_{\{32i \cdots 32i+31\}} = s_{\{128 \cdots 159\}}$

end for

1.3.5 The decryption

In a decryption process, the initialization and processing the associate data are the same as the encryption process. After processing the associated data, we decrypt the ciphertext C. In each step, the Framebits of ciphertext (the value is 5) are XORed with the state, then we update the state using the permutation $P_{160,1152}$. We obtain 32 bits of plaintext by XORing the ciphertext with 32 state bits $s_{\{128 \cdots 159\}}$. If the last block is not a full block, the last block is padded with zeros, and the number of bytes of ciphertext in the partial block is XORed with the state.

/* processing the full blocks of ciphertext, each block 32 bits*/

for i from 0 to $\lfloor mlen/32 \rfloor$:

$s_{\{68 \cdots 70\}} = s_{\{68 \cdots 70\}} \oplus FrameBits_{\{0 \cdots 2\}}$

Update the state using $P_{160,1152}$

$m_{\{32i \cdots 32i+31\}} = s_{\{128 \cdots 159\}} \oplus c_{\{32i \cdots 32i+31\}}$

$s_{\{128 \cdots 159\}} = c_{\{32i \cdots 32i+31\}}$

end for

/* processing the partial block of ciphertext, less than 32 bits*/
if ($mlen$ mod 32) > 0:

$s_{\{68\cdots70\}} = s_{\{68\cdots70\}} \oplus FrameBits_{\{0\cdots2\}}$
Update the state using $P_{160,1152}$
$lenp = mlen$ mod 32 /* number of bits in partial block */
$startp = mlen - lenp$ /* starting position of partial block */
$m_{\{startp\cdots mlen-1\}} = s_{\{128\cdots128+lenp-1\}} \oplus c_{\{startp\cdots mlen-1\}}$
$s_{\{128\cdots128+lenp-1\}} = c_{\{startp\cdots mlen-1\}}$
/* the length (bytes) of the last partial block is XORed to the state*/
$s_{\{64\cdots65\}} = s_{\{64\cdots65\}} \oplus (lenp/8)$

end if

1.3.6 The verification

After decrypting the ciphertext, we generate a 64-bit authentication tag T', then compare T' with the received tag T. The value of the finalization Framebits is 7.

for i from 0 to 1:

$s_{\{68\cdots70\}} = s_{\{68\cdots70\}} \oplus FrameBits_{\{0\cdots2\}}$
Update the state using $P_{160,1152}$
$t_{\{32i\cdots32i+31\}} = s_{\{128\cdots159\}}$

end for

$T' = t'_{\{0\cdots63\}}$. Accept the message if $T' = T$; otherwise, reject.

1.4 CLX Authenticated Encryption Algorithms

The specifications of all the CLX authenticated encryption algorithms are similar to that of CLX-128. In a CLX authenticated encryption algorithm, the state size is $160 + x$ bits. The associated data length and the plaintext length are less than 2^{50} bytes. The authentication tag is 64-bit. The same secret key should not be used in different CLX algorithms. The parameters used in the CLX authenticated encryption algorithms are given in Table 1.2.

Table 1.2: The Parameters of CLX Authenticated Encryption Algorithms

	State	x	Permu1	Permu2	Permu3	Permu4
CLX-128Q	192 bits	32	$P_{192,384}$	$P_{192,640}$	$P_{192,1280}$	$P_{192,640}$
CLX-128H	192 bits	32	$P_{192,384}$	$P_{192,1280}$	$P_{192,1280}$	–
CLX-192Q	256 bits	96	$P_{256,384}$	$P_{256,768}$	$P_{256,1408}$	$P_{256,768}$
CLX-192H	256 bits	96	$P_{256,384}$	$P_{256,1408}$	$P_{256,1408}$	–
CLX-256Q	320 bits	160	$P_{320,384}$	$P_{320,896}$	$P_{320,1536}$	$P_{320,896}$
CLX-256H	320 bits	160	$P_{320,384}$	$P_{320,1536}$	$P_{320,1536}$	–

1.4.1 The initialization

The initialization consists of key setup and nonce setup.

Key Setup.

1. Set the (160+x)-bit state S as 0.
2. Set $s_{31+x} = 1$.
3. Set $s_{\{32+x,127+2x\}} = k_{\{0,95+x\}}$.
4. Update the state using Permu3.

Nonce Setup.

for i from 0 to 2:
  $s_{\{36+x\cdots 38+x\}} = s_{\{36+x\cdots 38+x\}} \oplus FrameBits_{\{0\cdots 2\}}$
  Update the state using Permu1
  $s_{\{128+x\cdots 159+x\}} = s_{\{128+x\cdots 159+x\}} \oplus nonce_{\{32i\cdots 32i+31\}}$
end for

1.4.2 Processing the associated data

The associated data is divided into 32-bit blocks, then a 32-bit associated data block is processed in each step.

/* processing the full blocks of associated data, each block 32 bits*/
for i from 0 to $\lfloor adlen/32 \rfloor$:
  $s_{\{68+x\cdots 70+x\}} = s_{\{68+x\cdots 70+x\}} \oplus FrameBits_{\{0\cdots 2\}}$
  Update the state using Permu1
  $s_{\{128+x\cdots 159+x}\} = s_{\{128+x\cdots 159+x\}} \oplus ad_{\{32i\cdots 32i+31\}}$
end for

/* processing the partial block of the associated data*/
if ($adlen$ mod 32) > 0:
  $s_{\{68+x\cdots 70+x\}} = s_{\{68+x\cdots 70+x\}} \oplus FrameBits_{\{0\cdots 2\}}$
  Update the state using Permu1
  $lenp = adlen$ mod 32 /* number of bits in partial block */
  $startp = adlen - lenp$ /* starting position of partial block */
  $s_{\{128+x\cdots 128+x+lenp-1\}} = s_{\{128+x\cdots 128+x+lenp-1\}} \oplus ad_{\{startp\cdots adlen-1\}}$
  /* the length (bytes) of the last partial block is XORed to the state*/
  $s_{\{64+x\cdots 65+x\}} = s_{\{64+x\cdots 65+x\}} \oplus (lenp/8)$
end if

/*After processing the associated data, the state is updated again*/
/* Omit this step when Permu4 is undefined for the variant */
Update the state using Permu4

1.4.3 The encryption

The plaintext is divided into 32-bit blocks and get encrypted.

/* processing the full blocks of plaintext, each block 32 bits*/
for i from 0 to $\lfloor mlen/32 \rfloor$:
    $s_{\{68+x \cdots 70+x\}} = s_{\{68+x \cdots 70+x\}} \oplus FrameBits_{\{0 \cdots 2\}}$
    Update the state using Permu2
    $s_{\{128+x \cdots 159+x\}} = s_{\{128+x \cdots 159+x\}} \oplus m_{\{32i \cdots 32i+31\}}$
    $c_{\{32i \cdots 32i+31\}} = s_{\{128+x \cdots 159+x\}}$
end for
/* processing the partial block of plaintext, less than 32 bits*/
if ($mlen$ mod 32) > 0:
    $s_{\{68+x \cdots 70+x\}} = s_{\{68+x \cdots 70+x\}} \oplus FrameBits_{\{0 \cdots 2\}}$
    Update the state using Permu2
    $lenp = mlen$ mod 32 /* number of bits in partial block */
    $startp = mlen - lenp$ /* starting position of partial block */
    $s_{\{128+x \cdots 128+x+lenp-1\}} = s_{\{128+x \cdots 128+x+lenp-1\}} \oplus m_{\{startp \cdots mlen-1\}}$
    $c_{\{startp \cdots mlen-1\}} = s_{\{128+x \cdots 128+x+lenp-1\}}$
    /* the length (bytes) of the last partial block is XORed to the state*/
    $s_{\{64+x \cdots 65+x\}} = s_{\{64+x \cdots 65+x\}} \oplus (lenp/8)$
end if

1.4.4 The finalization

We generate the tag in two steps. In each step, the framebits for finalization (with value 7) are XORed to the state, then the state gets updated and 32 bits of the tag are generated.

for i from 0 to 1:
    $s_{\{68+x \cdots 70+x\}} = s_{\{68+x \cdots 70+x\}} \oplus FrameBits_{\{0 \cdots 2\}}$
    Update the state using Permu3
    $t_{\{32i \cdots 32i+31\}} = s_{\{128+x \cdots 159+x\}}$
end for

1.4.5 The decryption

In a decryption process, the initialization and processing the associate data are the same as the encryption process. After processing the associated data, we decrypt the ciphertext C. The ciphertext is divided into 32-bit blocks and get decrypted.

/* processing the full blocks of plaintext, each block 32 bits*/
for i from 0 to $\lfloor mlen/32 \rfloor$:

$s_{\{68+x\cdots 70+x\}} = s_{\{68+x\cdots 70+x\}} \oplus FrameBits_{\{0\cdots 2\}}$
Update the state using Permu2
$m_{\{32i\cdots 32i+31\}} = s_{\{128+x\cdots 159+x\}} \oplus c_{\{32i\cdots 32i+31\}}$
$s_{\{128+x\cdots 159+x\}} = c_{\{32i\cdots 32i+31\}}$

end for
/* processing the partial block of plaintext, less than 32 bits*/
if ($mlen$ mod 32) > 0:

$s_{\{68+x\cdots 70+x\}} = s_{\{68+x\cdots 70+x\}} \oplus FrameBits_{\{0\cdots 2\}}$
Update the state using Permu2
$lenp = mlen$ mod 32 /* number of bits in partial block */
$startp = mlen - lenp$ /* starting position of partial block */
$m_{\{startp\cdots mlen-1\}} = s_{\{128+x\cdots 128+x+lenp-1\}} \oplus c_{\{startp\cdots mlen-1\}}$
$s_{\{128+x\cdots 128+x+lenp-1\}} = c_{\{startp\cdots mlen-1\}}$
/* the length (bytes) of the last partial block is XORed to the state*/
$s_{\{64+x\cdots 65+x\}} = s_{\{64+x\cdots 65+x\}} \oplus (lenp/8)$

end if

1.4.6 The verification

After decrypting the ciphertext, we generate a 64-bit authentication tag T', then compare T' with the received tag T. In each step, the framebits for finalization (with value 7) are XORed to the state.

for i from 0 to 1:

$s_{\{68+x\cdots 70+x\}} = s_{\{68+x\cdots 70+x\}} \oplus FrameBits_{\{0\cdots 2\}}$
Update the state using Permu3
$t_{\{32i\cdots 32i+31\}} = s_{\{128+x\cdots 159+x\}}$

end for
$T' = t'_{\{0\cdots 63\}}$. Accept the message if $T' = T$; otherwise, reject.

1.5 CLX-Hash

CLX-Hash hashes a byte sequence less than 2^{50} bytes and generates a 256-bit authentication tag. CLX-Hash is based on the sponge mode. A 288-bit permutation $P'_{288,n}$ is used in CLX-Hash. The state update function of CLX-128 is part of that of CLX-Hash, so it is efficient to implement both CLX-128 and CLX-Hash on hardware.

1.5.1 The permutation $P'_{288,n}$

The permutation $P'_{288,n}$ is an extension of the (160+128)-bit permutation.

The state update function for $P'_{288,n}$

StateUpdate(S) for $x > 0$:
  feedback $= s_0 \oplus s_{19} \oplus s_{128} \oplus s_{163} \oplus (\sim (s_{221} \& s_{234})) \oplus s_{255}$
  for i from 0 to 286: $s_i = s_{i+1}$
  s_{287} = feedback
end

1.5.2 The initialization

The state is set to constant in the initialization.

1. Set the 288-bit state S as 0.
2. Set $s_{196} = 1$.
3. Update the state using $P'_{288,1024}$.

1.5.3 Processing the message

The message is divided into 32-bit blocks, then a 32-bit message block is hashed in each step. When there is partial block, the length (bytes) of the last partial block is XORed to the state.

/* processing the full blocks of message, each block 32 bits*/
for i from 0 to $\lfloor mlen/32 \rfloor$:
  $s_{196} = s_{196} \oplus 1$
  $s_{\{256 \cdots 287\}} = s_{\{256 \cdots 287\}} \oplus m_{\{32i \cdots 32i+31\}}$
  Update the state using $P'_{288,2560}$
end for

/* processing the partial block of message*/
if ($mlen$ mod 32) > 0:
  $s_{196} = s_{196} \oplus 1$
  $lenp = mlen$ mod 32 /* number of bits in partial block */
  $startp = mlen - lenp$ /* starting position of partial block */
  $s_{\{256 \cdots 256+lenp-1\}} = s_{\{256 \cdots 256+lenp-1\}} \oplus m_{\{startp \cdots mlen-1\}}$
  $s_{\{192 \cdots 193\}} = s_{\{192 \cdots 193\}} \oplus (lenp/8)$
  Update the state using $P'_{288,2560}$
end if

1.5.4 The finalization

We generate the message digest, MD, as follows.

for i from 0 to 7:

$md_{\{32i \cdots 32i+31\}} = s_{\{256 \cdots 287\}}$

$s_{196} = s_{196} \oplus 1$

Update the state using $P'_{288,256}$

end for

Chapter 2

Security Goals

2.1 Security goals with unique nonce

In CLX authenticated encryption algorithms, each pair of key and nonce is used to protect only one message. If verification fails, the new tag and the decrypted ciphertext should not be given as output.

The security goals of CLX for unique nonce are given in Table 2.1. We assume that each each key is used to process at most 2^{50} byes of messages (associated data, plaintext/ciphertext), and each message is at least 8 bytes. Note that the authentication security in Table 2.1 includes the integrity security of plaintext, associated data and nonce.

Table 2.1: Security Goals of CLX with Unique Nonce

	Encryption	Authentication
CLX-128	112-bit	64-bit
CLX-128Q	112-bit	64-bit
CLX-128H	112-bit	64-bit
CLX-192Q	168-bit	64-bit
CLX-192H	168-bit	64-bit
CLX-256Q	224-bit	64-bit
CLX-256H	224-bit	64-bit

2.2 Security goals with repeated nonce

When nonce is reused in CLX authenticated encryption algorithms, the secret key security of CLX-128H, CLX-196H and CLX-256 remains strong.

When nonce is reused in CLX, the authentication security of CLX-128H, CLX-196H and CLX-256 remains strong.

When nonce is reused, an attacker is able to decrypt a ciphertext since the CLX encryption is somehow similar to the cipher feedback mode.

Table 2.2: Security Goals of CLX with Repeated Nonce (*the security of CLX-128 is for the nonce being repeated only 2^{12} times)

	Secret Key	Authentication
CLX-128	112-bit*	64-bit*
CLX-128Q	–	–
CLX-128H	112-bit	64-bit
CLX-192Q	–	–
CLX-192H	168-bit	64-bit
CLX-256Q	–	–
CLX-256H	224-bit	64-bit

2.3 Security goals of hash function

The security goals of CLX-Hash are given in Table 2.3.

Table 2.3: Security Goals of CLX-Hash

	Preimage	Second Preimage	Collision
CLX-Hash	112-bit	112-bit	112-bit

Chapter 3

Security Analysis

3.1 Security of the Duplex Mode

The Duplex mode [3] was proposed to use the sponge hash mode to construct an authenticated encryption scheme. In [10], the security of the Duplex mode was thoroughly analysed. For a Duplex mode design with key size k, block size b, rate r and capacity c, the security is approximately $\min\{2^{b/2}, 2^c, 2^k\}$. The security bound can be explained as follows. The key can be attacked using brute force, so the security cannot go beyond 2^k. c bits of the state are the only unknown information of the state, so the security cannot go beyond 2^c. The complexity $2^{b/2}$ is required for a state collision to occur (b-bit state).

In the lightweight applications, the amount of data being processed is assumed to be at most 2^{50} bytes for a single key. For CLX, it means that there are at most 2^{48} data blocks (assume that each message is at least 8 bytes). To obtain a state collision, an attacker needs to try 2^{b-48} random states in order to obtain a collision with the cipher states. The security of Duplex mode becomes $\min\{2^{b-48}, 2^c, 2^k\}$ for a lightweight authenticated encryption algorithm. For a lightweight authenticated encryption algorithm, the minimum state size is thus 160-bit for 112-bit security.

3.2 Properties of the Permutation $P_{160+x,n}$

3.2.1 Differential properties of the permutation

In this section, we analyse the differential properties [4, 5] of the CLX permutation $P_{160+x,n}$. The following five types of differences will be analysed.

- Type 1. Input differences at $S_{128+x\cdots159+x}$
- Type 2. Arbitrary input differences, output differences at $S_{128+x\cdots159+x}$
- Type 3. Input differences at $S_{128+x\cdots159+x}$, output differences at $S_{128+x\cdots159+x}$

- Type 4. Differences in two-block case
- Type 5. Arbitrary input differences, arbitrary output differences

In the following, we will analyse the differential propagation using the Mixed Integer Linear Programming (MILP) [13]. We will use the Gurobi optimizer [8] to find the exact bound for some rounds.

Type 1 Differences

For the Type 1 differences, the input differences are at $S_{128+x \cdots 159+x}$, and there is no restriction on the output differences. The largest differential probabilities of the Type 1 differences are summarized in Table 3.1.

Table 3.1: Type 1 Differential Properties of the Permutation

Design	Round	Probability	Method
$P_{160,n}$	384	2^{-39}	MILP
	512	2^{-64}	MILP
$P_{192,n}$	384	2^{-41}	MILP
	512	2^{-69}	MILP
$P_{256,n}$	384	2^{-41}	MILP
	512	2^{-69}	MILP
$P_{320,n}$	384	2^{-38}	MILP
	544	2^{-67}	MILP

Type 2 Differences

For the Type 2 differences, there is no restriction on the input differences, and output differences are at $S_{128+x \cdots 159+x}$. The largest differential probabilities of the Type 2 differences are summarized in Table 3.2.

Table 3.2: Type 2 Differential Properties of the Permutation

Design	Round	Probability	Method
$P_{160,n}$	384	2^{-23}	MILP
$P_{192,n}$	384	2^{-28}	MILP
$P_{256,n}$	384	2^{-7}	MILP
$P_{320,n}$	384	2^{-2}	MILP

Type 3 Differences

For the Type 3 differences, the input differences are at $S_{128+x\cdots159+x}$, and output differences are at $S_{128+x\cdots159+x}$. The largest differential probabilities of the Type 3 differences are summarized in Table 3.3.

Table 3.3: Type 3 Differential Properties of the Permutation

Design	Round	Probability	Method
$P_{160,n}$	384	$\leq 2^{-89}$	MILP
$P_{192,n}$	384	$\leq 2^{-111}$	MILP
$P_{256,n}$	384	0	MILP
$P_{320,n}$	384	0	MILP

Type 4 Differences

For the Type 4 differences, the arbitrary input differences injected to $S_{128+x\cdots159+x}$ after 384 rounds. This is to capture the differential propagation through more than one message blocks. The largest differential probabilities of the Type 4 differences are summarized in Table 3.4.

Table 3.4: Type 4 Differential Properties of the Permutation

Design	Round	Probability	Method
$P_{160,n}$	576	2^{-67}	MILP
$P_{192,n}$	576	2^{-74}	MILP
$P_{256,n}$	576	2^{-76}	MILP
$P_{320,n}$	576	2^{-66}	MILP

Type 5 Differences

The Type 5 differences are arbitrary input difference and arbitrary output difference. Our observation of the MILP results shows that the optimal differential (of Type 5) of CLX appears when there is one bit difference in a middle state. Based on this observation, we use MILP to find the largest differential probabilities of the Type 5 differences. The results are summarized in Table 3.5.

3.2.2 Linear properties of the permutation $P_{160+x,n}$

In this section, we analyse the linear properties of the CLX permutation $P_{160+x,n}$. The linear bias for output bits at $S_{96+x\cdots127+x}$ will be analysed.

Table 3.5: Type 5 Differential Properties of the Permutation

Design	Round	Probability	Method
$P_{160,n}$	1280	2^{-160}	MILP
$P_{192,n}$	1280	2^{-180}	MILP
$P_{256,n}$	1280	2^{-95}	MILP
$P_{320,n}$	1920	2^{-99}	MILP

In the analysis, we use the Mixed Integer Linear Programming (MILP) [13]. We will use the Gurobi optimizer [8] to find the exact linear bias for some rounds. The results are summarized in Table 3.6.

Table 3.6: Linear bias of the permutation

Design	Round	Bias
$P_{160,n}$	480	2^{-31}
$P_{192,n}$	480	2^{-33}
$P_{256,n}$	448	2^{-31}
$P_{320,n}$	448	2^{-27}

3.2.3 Algebraic properties of the permutation $P_{160+x,n}$

We consider the algebraic property for the input bits at $S_{128+x\cdots159+x}$. Our experiment shows that after 500 rounds, every output bit at $S_{96+x\cdots127+x}$ is affected by the 32-bit input cube tester at $S_{128+x\cdots159+x}$. The results are summarized in Table 3.7.

Table 3.7: Round number for 32-bit cube tester to affect output bits

Design	Round	Cube Tester
$P_{160,n}$	493	32-bit
$P_{192,n}$	493	32-bit
$P_{256,n}$	481	32-bit
$P_{320,n}$	493	32-bit

3.3 Forgery Attacks

For an authenticated encryption scheme, an internal state collision will directly lead to a forgery attack. To produce a state collision, an attacker can inject difference into nonce, associated data or plaintext/ ciphertext, then eliminate the difference in the state using the difference in the later input blocks.

3.3.1 Forgery attacks on nonce and associated data

In CLX, each 32-bit nonce block and associated data block is processed using $P_{160+x,384}$ (with different Framebits for nonce and associated data). The associated data also plays the role of nonce in CLX.

Nonce and associated data are processed in a very similar way in CLX (the only difference is the used of different FrameBits). In the following, we only need to consider the forgery attacks on associated data. There are two cases of forgery attacks on the associated data:

Case 1. Forgery attacks with differences at only two adjacent associated data blocks, i.e., $\Delta ad_i \neq 0$ and $\Delta ad_{i+1} \neq 0$

For this type of forgery attacks, the input difference to $P_{160+x,384}$ is at $s_{128+x \cdots 159+x}$, and the output difference is at $s_{128+x \cdots 159+x}$. This is the Type 3 differences analysed in Sect. 3.2.1. According to Table 3.3, the largest differential probability of Type 3 differences of P_{384} is at most 2^{-89}. It means that the forgery attack succeeds with probability at most 2^{-89} using this type of differential attack.

Case 2. Forgery attacks involving more than two associated data blocks, i.e., $\Delta ad_i \neq 0$ and $\Delta ad_j \neq 0$, where $j > i+1$. (Δad_w may or may not be zero for $i < w < j$.)

For this type of forgery attacks, at least two permutations $P_{160+x,384}$ are involved. This is the Type 4 differences analysed in Sect. 3.2.1. According to Table 3.4, the largest differential probability of Type 4 differences is at most 2^{-66}. Since there are at least 768 rounds in this case, the forgery attack succeeds with probability should be less than 2^{-66} using this type of differential attack.

The above analysis shows that the differential forgery attack on nonce and associated data succeeds with probability at most $\max(2^{-89}, 2^{-66}) = 2^{-66}$. Note that 2^{-66} is for 576 rounds of $P_{320,576}$, while the differential needs to pass through at least 768 rounds of $P_{320,768}$. So the security margin is very large (we computed the differential for only 576 rounds due to the high computational complexity of MILP for large rounds).

3.3.2 Forgery attacks on plaintext/ciphertext

We analyse the forgery attacks when differences are introduced into plaintext/ciphertext. When an adversary introduces a difference in a plaintext

block M_i or ciphertext block C_i , the difference introduces input difference at $s_{128+x \cdots 159+x}$ of the permutation P_n. According to Table 3.1, the differential is at most 2^{-64} after 544 rounds.

In CLX, at least 1024 rounds are used to encrypt a plaintext block, so the differential forgery attack on plaintext/ciphertext succeeds with probability much smaller than 2^{-64}.

3.4 State Recovery Based on State Collision

The state recovery attack based on state collision [1] is useful for analyzing stream ciphers. The idea is that an attacker can try many random states offline, then generate keystreams from the states, then compare the keystreams with the received keystreams to identify the state.

For lightweight applications, there is constraint that a key is used to process at most 2^{50} bytes of data. If we assume that a message is at least 8 bytes, a key is used to process at most 2^{48} data blocks in CLX. To provide n-bit security, the state size should be at least $n+48$ bits when non-keyed permutation is used in the Duplex mode to prevent the state recovery attack using state collision. When nonce is misused in the Duplex mode, message block can be applied to set part of the state to constant. When 32-bit message block is used, the state size should be at least $n + 48 + 32$ bits when non-keyed permutation is used in the Duplex mode.

The state sizes of CLX are chosen to be large enough to resist the state collision when nonce is unique. CLX-128 is able to provide 112-bit security since its state size is $112 + 48 = 160$ bits. This is the smallest state size for 112-bit security when less than 2^{50} bytes of data are processed by a single key. The state sizes of other variants are $n + 48 + 32$ bits, providing even higher security.

The state sizes of CLX-128H, CLX-192H, CLX-256H are large enough to resist the state collision when nonce is misused since their state sizes are $n + 48 + 32$ bits.

The state size of CLX-128 is 160-bit, so CLX-128 is not able to provide 112-bit security when nonce is repeated 2^{112} times. However, if we assume that the nonce of CLX-128 being repeated only 2^{16} times (one message from each repeated nonce), then the attacker still needs to try 2^{112} states in order to recover the state through state collision. Our security goal is that CLX-128 provides 112-bit security when nonce is repeated 2^{12} times.

3.5 Key Recovery Attacks for Repeated Nonce

In this section ,we analyse the security of CLX-128, CLX-128H, CLX-192H and CLX-256H for repeated nonce.

3.5.1 Differential cryptanalysis

When nonce is misused, a difference can be injected into the state at $s_{128+x\cdots159+x}$ through a plaintext block, then the output difference can be observed in the next ciphertext block.

According to Table 3.1, the maximum differential probability is less than 2^{-60} after 512 rounds. In CLX-128H, CLX-192H and CLX-256H, at least 1280 rounds are used to encrypt a 32-bit message block. The differential probability for 1280 rounds is much smaller than 2^{-32}, we thus believe that it is impossible to recover the key of CLX using the differential cryptanalysis.

3.5.2 Linear cryptanalysis

When nonce is misused, an attacker can try to find the linear relation [11, 12] between the input $s_{128+x\cdots159+x}$ and the output $s_{96+x\cdots127+x}$ of $P_{160+x,n}$.

According to Table 3.6, the linear bias is at most 2^{-27} after 480 rounds. This linear bias is much smaller than 2^{-16} (the message block size is 32-bit). At least 1280 rounds are used in the encryption of one plaintext block in CLX-128H, CLX-192H and CLX-256H, we thus believe that it is impossible to recover the key of TinyJAMBU using the linear cryptanalysis.

3.5.3 Algebraic attacks

We experimentally tested the number of rounds for each output bit being affected by 32-bit cube [7] for the input.

According to Table 3.7, after 500 rounds, every output bits is affected by the 32-bit cube tester. Hence, we believe that the 1024-round encryption provides large security margin against the algebraic cryptanalysis since the message block size is 32-bit.

3.6 Related-key Attacks

CLX provides resistance against the related-key key recovery attack. According to Table 3.5, the related-key differential probability is at most 2^{-95} for CLX-128, CLX-128Q, CLX-128H, CLX-192Q, and CLX-192H, because the key was processed with at least $1024+384 = 1408$ rounds before the nonce being injected into the state.

For CLX-256Q and CLX-256H, the key was processed with at least $1536 + 384 = 1920$ rounds before the nonce being injected into the state. According to Table 3.5, the related-key differential probability is at most 2^{-99}.

Each key is used to process at most 2^{50} bytes of data, so CLX is strong against the related-key key recovery attack.

3.7 Slide attack

The slide attack is an effective tool to analyse the cipher with self-similarity round functions. Although CLX permutation has the sliding property, the frame bits being added to the state will prevent the slide attack since the position of the frame bits is fixed. Further, due to the large state size of CLX (at least (160-bit), and the data being processed is at most 2^{50} bytes, the chance that there are slide states is negligently small.

3.8 Security Analysis of CLX-Hash

CLX-Hash is built on the 288-bit permutation $P'_{288,n}$ using the sponge mode. We compute the Type 5 difference as in Section 3.2.1 for the permutation $P'_{288,n}$. The probability of the 2048-round Type 5 differential for P'_{288} is 2^{-307}. Since each 32-bit message block is compressed using 2560-round permutation $P'_{288,2560}$, we expect that CLX-Hash has strong collision resistance.

Chapter 4

The Performance of CLX

4.1 Hardware Performance

We implement CLX-128 VHDL using the CAESAR Hardware API [9]. In our lightweight implementations of CLX, we compute 8 and 32 rounds in one clock cycle. We synthesis our implementations with the Synopsys Design Compiler for an ASIC using 90 nm UMC technology. The results are summarized in Table 4.1. The hardware cost of the whole cipher (including initialization, processing associated data, encryption and decryption, tag generation and verification) is counted. We exclude the cost of the CAESAR Hardware API (preprocessor and postprocessor) which is the same for all the authenticated encryption algorithms.

4.2 Software Performance

In software implementation, the amount of RAM and ROM required by CLX is expected to be very small. We will test CLX later on the low end processors for the exact cost of RAM and RAM. Some estimations are given below.

1. 160+x bits of RAM are used to store the state.
2. The cipher does not need the storage of the key on the device.
3. Only 12 bits of ROM are needed to store constants (Framebits). (Alternatively, we may use 4 bytes of ROM to store the constants)
4. According to the reference code, the binary code of CLX is expected to be very small since we repeatedly using the permutation in the implementation, and the permutation is implemented using only 16 lines of code. The amount of ROM being used to store the binary code is expected to be small.

Table 4.1: The hardware performance of CLX (the whole cipher is implemented, including the initialization, processing AD, encryption/decryption, tag generation and verification.

Implementation	Area (GE)	Throughput for long AD (Mbps)	Throughput for long P (Mbps)
CLX-128 (8 rounds)	1501	81.7	27.2
CLX-128 (32 rounds)	1884	223.7	74.5
CLX-128Q (8 rounds)	1730	80.1	48.0
CLX-128Q (32 rounds)	2195	225.2	135.1
CLX-192Q (8 rounds)	1743	80.1	40.0
CLX-192Q (32 rounds)	2597	223.7	111.8
CLX-256Q (8 rounds)	2564	81.6	35.0
CLX-256Q (32 rounds)	3013	223.7	95.8
CLX-Hash (8 rounds)	2301	-	12.2
CLX-Hash (32 rounds)	2696	-	35.4

Chapter 5

Features

- Lightweight Permutation. The permutation is based on a (160+x)-bit nonlinear feedback shift register with only 5 taps (6 taps for register larger than 160 bits). It is thus efficient to implement the permutation in hardware.
- Lightweight Input Loading. The nonce, associated data and the plaintext/ciphertext can be loaded into the state bit-by-bit when the nonlinear feedback shift register of the permutation is clocked. It is thus efficient to load the input stream into the cipher in hardware.
- Parallel Computation. In CLX, 32 steps can be computed in parallel. This parallel feature benefits fast hardware and software implementations.
- Associated data can be treated as part of the nonce. As long as the pairs (nonce, associated data) are different, we consider that nonce is unique.

Chapter 6

Design Rationale

- Design goal
  We aim to design a lightweight authenticated encryption algorithm which is optimized for the devices in which changing keys are used, and the keys are not stored in the devices.

- Use the Duplex authenticated encryption mode

  When the key is not stored in the devices, it is desirable to use the non-keyed permutation to design lightweight authenticated encryption algorithms so as to avoid the cost of storing the key.

  When non-keyed permutation is used, Duplex mode is suitable for lightweight design, so we choose the Duplex mode in CLX.

  [Note that when the key is already stored in the devices (or when the key must be stored in the devices), it is desirable to use the keyed permutation to design lightweight authenticated encryption since much smaller state can be used in the keyed permutation to obtain the same security (because the constraint on the amount of data being processed by each key of a lightweight cipher in practice, and the state size of non-keyed permutation must be sufficiently large to resist the state recovery attack using collision [1]).]

- Choose the permutation size

  Non-keyed permutation is used in CLX, the permutation should be large enough to resist the state recovery attack based on state collision [1]. We assume that each key is used to process less than 2^{50} bytes of data, so there are at most 2^{48} data blocks (we assume that each message is at least 8 bytes). To obtain 112-bit security of the secret key, the minimum state size should be 160 bits.

  In case of nonce misuse, the attacker is able to set part of the state to a constant value in the Duplex mode, so the effective state size is reduced. If

we use 32-bit message block size, then 192-bit permutation must be used for 112-bit security for reused nonce.

- Design the permutation

  The permutation is based on a simple nonlinear feedback shift register with only 5 taps (6 taps for register larger than 160 bits). There are three reasons for using the nonlinear feedback shift register to update the state:

  - The hardware cost of nonlinear feedback shift register is low.
  - A number of steps of the nonlinear feedback shift registers can be computed in parallel for efficient hardware and software implementation.
  - The stream input data can be easily loaded into the state.

  There is slide property in the permutation based on a feedback shift register. We use the FrameBits for each permutation to protect the cipher against the slide property.

- Design the nonlinear feedback shift register

  All the CLX algorithms are based on the 160-bit nonlinear feedback shift register with 5 taps. A (160+x)-bit feedback shift register is to extend the 160-bit register by x bits without modifying the 160-bit feedback shift register. In this way, all the variants of CLX algorithms share the main component. It saves the hardware cost when we need to implement more than one CLX algorithms in hardware. It also allows the CLX algorithms being described and understood easily.

  In the 160-bit nonlinear feedback shift register, we need to choose four tap positions.

  - The tap positions are chosen to ensure that 32 steps can be computed in parallel.
  - There are 15 tap distances for the feedback register with 6 taps. The tap positions are chosen to ensure that most of those 15 tap distances are co-prime to each other. When two tap distances are not co-prime, the greatest common divisor should be small.
  - After the above filtering of tap positions, we tested the differential and linear property of the permutation (with 32-bit message block). We choose the tap distances that give high security against the forgery attack on associated data and the plaintext/ciphertext.

- Different processing of associated data and plaintext

  In CLX, we use a strong permutation for encrypting plaintext, while we use a weak permutation for processing associated data so that we can get better efficiency. The reason is that as long as there is no state collision due to the associated data, there is no state information leakage, so a weak

permutation can be used for processing the associated data. However, the ciphertext leaks the state information, so strong permutation is needed for encrypting the plaintext.

Bibliography

[1] S. Babbage. "A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers". *European Convention on Security and Detection*, IEE Conference Publication No. 408, May 1995.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "Sponge Functions". Ecrypt Hash Workshop 2007.

[3] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. "Duplexing the sponge: Single-pass authenticated encryption and other applications". In *Selected Areas in Cryptography – SAC 2011*, pages 320–337.

[4] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[5] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, London, UK, 1993.

[6] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available at `https://competitions.cr.yp.to/caesar-submissions.html`

[7] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, 2009. Springer Berlin Heidelberg.

[8] Gurobi Optimizer. Available at `http://www.gurobi.com/`

[9] Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj. CAESAR hardware API. *IACR Cryptology ePrint Archive*, 2016:626, 2016.

[10] Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. Beyond conventional security in sponge-based authenticated encryption modes. *Journal of Cryptology*, Jun 2018.

[11] Mitsuru Matsui. Linear Cryptanalysis Method for DES cipher. In *Advances in Cryptology–EUROCRYPT'93*, pages 386–397. Springer, 1994.

[12] Mitsuru Matsui and Atsuhiro Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *Advances in Cryptology – EUROCRYPT'92*, pages 81–91. Springer, 1993.

[13] Nicky Mouha, Qingju Wang, Dawu Gu, Bart Preneel. Differential and linear cryptanalysis using Mixed-Integer Linear Programming. *Information security and cryptology – Inscrypt 2011*, pages 57–76.