Tweaks and Keys for Block Ciphers: the TWEAKEY Framework

Thomas Peyrin
(joint work with Jérémy Jean and Ivica Nikolić)

NTU - Singapore

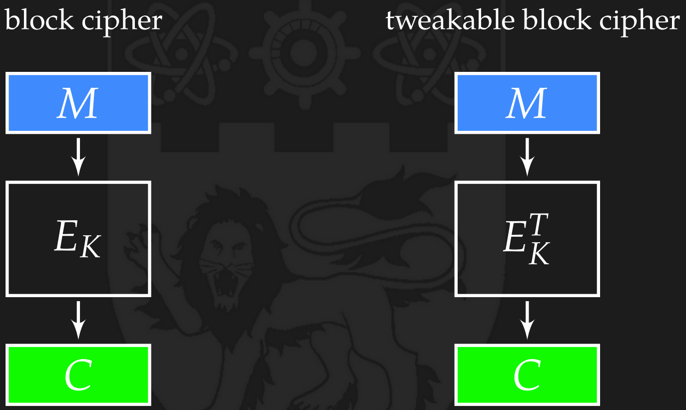
ASK 2014

Chennai, India - December 19, 2014

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

Block ciphers and tweakable block ciphers

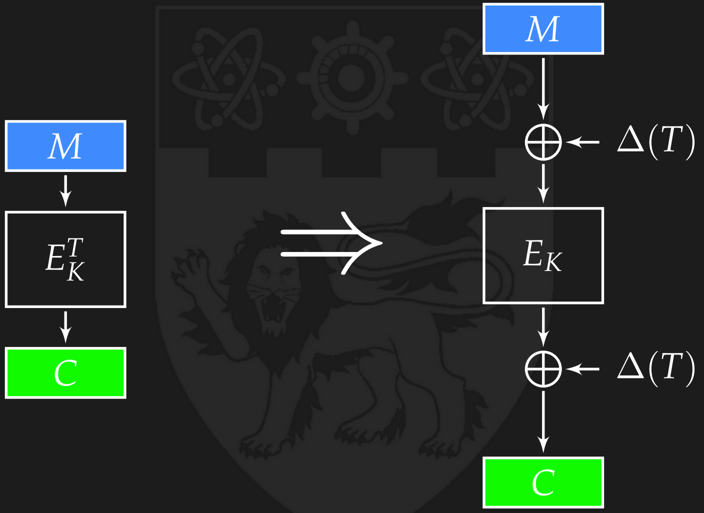


Tweakable block ciphers

Tweakable block ciphers are very useful building blocks:

- block cipher, stream cipher
- parallel MAC
- parallel authenticated encryption: like `OCB3` or `COPA`, but simpler design/proofs and much higher security bounds
- hash function: use the tweak input as block counter (HAIFA framework) or to perform randomized hashing
- tree hashing: use the tweak to encode the position in the tree
- PRNG, KDF, disk encryption

XEX-like constructions



Contributions

- block cipher based TBC constructions (like XEX) usually provide birthday security
- building an ad-hoc TBC with full security is not easy (very little number of proposals)
- even designing a key schedule remains a risky task, especially for long keys (see related-key attacks on AES-256)

Our contributions

- we propose the TWEAKEY framework to help designers to create tweakable block ciphers
- we provide one cipher example KIASU-BC , the first ad-hoc AES-based TBC
- in the TWEAKEY framework, we propose the STK construction for SPN ciphers
- we provide two cipher examples Joltik-BC and Deoxys-BC

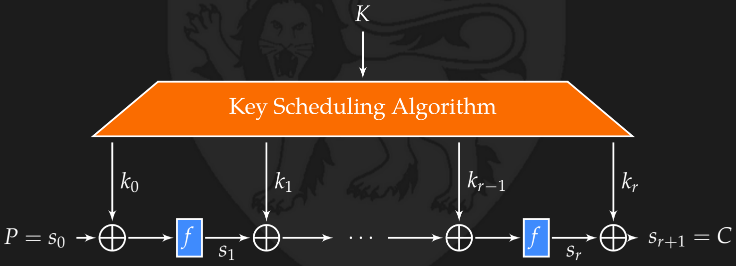
Outline

Outline

Block ciphers

Iterated SPN block ciphers

- internal permutation: f
- number of iterations: r
- SPN: $f = \mathsf{P} \circ \mathsf{S}$ applies Substitution (S) and Permutation (P).
- secret key: K
- key scheduling algorithm: $K \rightarrow (k_0, \ldots, k_r)$
- Ex: AES



Tweakable block ciphers ?

From an **efficiency** point of view, updating the tweak input of a TBC should be doable very efficiently

$\rightarrow$ **the tweak schedule should be lighter than the key schedule**

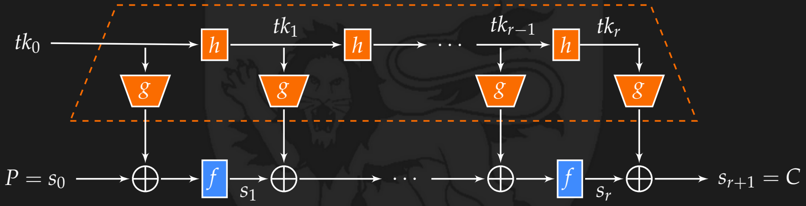
From a **security** point of view, the tweak is fully known and controllable, not the key

$\rightarrow$ **the tweak schedule should be stronger than the key schedule**

Thus, for a TBC designer, this paradox leads to $\mathit{tweak} = \mathit{key}$

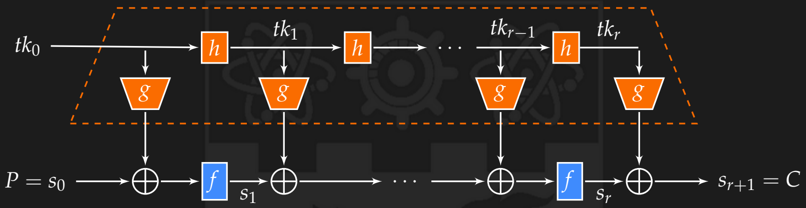
The TWEAKEY framework

Rationale:
tweak and key should be treated the same way ⟶ tweakey



TWEAKEY generalizes the class of key-alternating ciphers

The TWEAKEY framework



The TWEAKEY framework

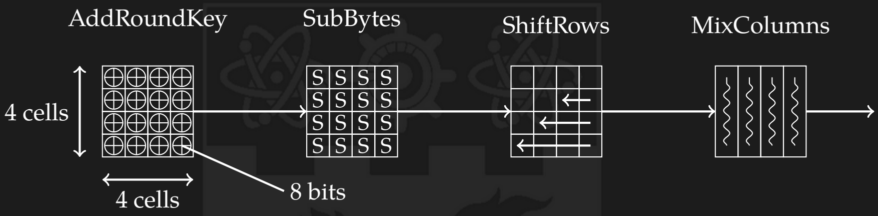
The regular key schedule is replaced by a TWEAKEY schedule that generates subtweakeys. An n-bit key n-bit tweak TBC has $2n$-bit tweakey and g compresses $2n$ to n bits:

- such a primitive would be a TK−2 primitive (TWEAKEY of order 2).
- the same primitive can be seen as a $2n$-bit key cipher with no tweak (or $1.5n$-bit key and $0.5n$-bit tweak, etc).

Outline

1. Introduction
2. The TWEAKEY Framework
   - ▷ TWEAKEY
   - ▷ The tweakable block cipher KIASU-BC
3. The STK Construction
   - ▷ STK
   - ▷ Joltik-BC and Deoxys-BC
4. Authenticated encryption with TBC
5. Future works

The AES−128 round function

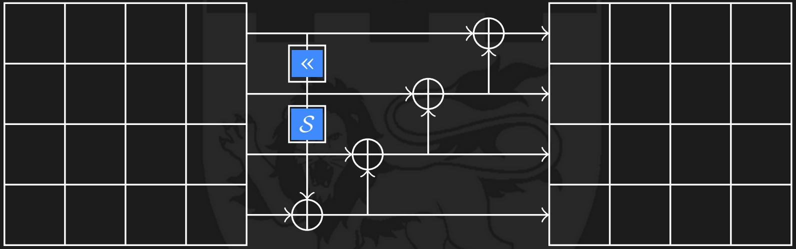


The 128-bit round function of AES−128 is an SP-network:

- **AddRoundKey:** xor incoming 128-bit subkey
- **SubBytes:** apply the 8-bit Sbox to each byte
- **ShiftRows:** rotate the i-th line by i positions to the left
- **MixColumns:** apply the AES−128 MDS matrix to each columns independently

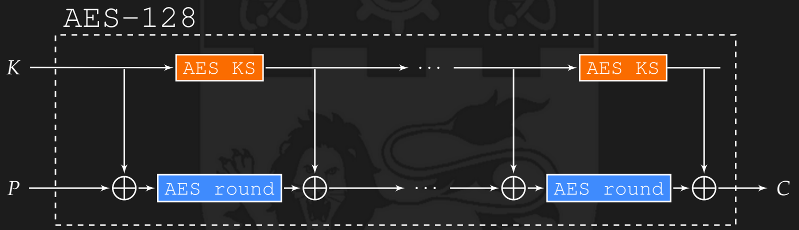
The AES−128 key schedule

AES−128 key schedule



The tweakable block cipher KIASU−BC

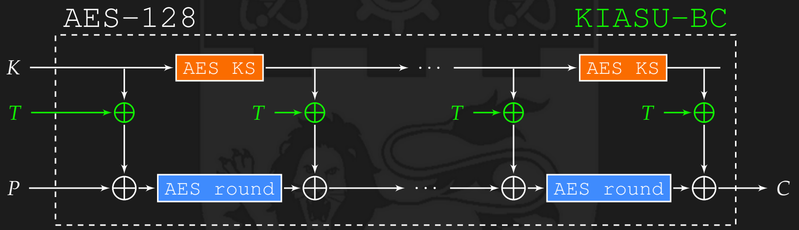
KIASU−BC is **exactly** the AES−128 cipher, but with a fixed 64-bit tweak value T XORed to each subkey (two first rows)



$$T = \begin{array}{|c|c|c|c|} \hline T_0 & T_2 & T_4 & T_6 \\ \hline T_1 & T_3 & T_5 & T_7 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$$

The tweakable block cipher KIASU−BC

KIASU−BC is **exactly** the AES−128 cipher, but with a fixed 64-bit tweak value T XORed to each subkey (two first rows)



$T =$

T_0	T_2	T_4	T_6
T_1	T_3	T_5	T_7
0	0	0	0
0	0	0	0

Security of KIASU−BC

The security of KIASU−BC is the same as AES−128 for a fixed tweak. The tricky part is to analyse what happens when the tweak varies.

If the key is fixed and one varies the tweak:

KIASU−BC's tweak schedule has been chosen such that it is itself a good key schedule.

Bad idea: adding a tweak on the entire 128-bit state, since trivial and very good related-tweakey differential paths would exist.

If both the key and tweak vary (aka related-tweakey):

KIASU−BC was designed such that no interesting interaction between the key schedule and the tweak schedule will exist. We put a special focus on attacks which are highly impacted by the key schedule:

- related-key related-tweak attacks (aka related-tweakey)
- meet-in-the-middle attacks

Security of KIASU−BC

Related-tweakey attacks

We prove that no good related-key related-tweak (aka related-tweakey) attacks differential path exist for KIASU (even boomerang), with a computer-aided search tool.

rounds	active SBoxes	upper bound on probability	method used
1-2	0	2^{0}	trivial
3	1	2^{-6}	Matsui's
4	8	2^{-48}	Matsui's
5	≥ 14	2^{-84}	Matsui's
7	≥ 22	2^{-132}	ex. split (3R+4R)

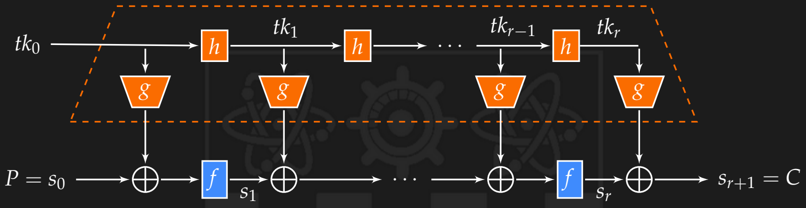
KIASU features

- first adhoc tweakable AES-128 ...
- ... which provides 2^{128} security - not only birthday security
- extremely fast in software: less than 1 c/B on Haswell
- quite small in hardware
- very simple - almost direct plug-in of AES-128 (reuse existing security analysis and implementations)
- backward compatible with AES-128 (simply set $T = 0$)

Outline

Outline

Building fast ad-hod tweakable block ciphers is not easy

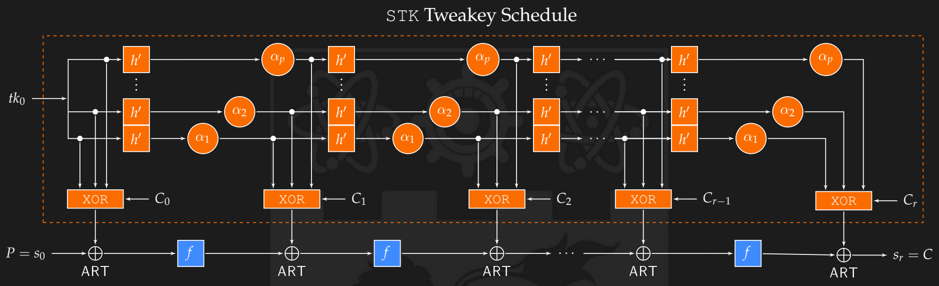


The case of AES-like ciphers

- KIASU is limited to 64-bit tweak for AES (insecure otherwise)
- we could do a LED-like design, but slow due to high number of rounds
- **the main issue:** adding more tweakey state makes the security drop, or renders security hard to study, even for automated tools

Idea: separate the tweakey material in several words, design a secure tweakey schedule for one word and then superpose them in a secure way

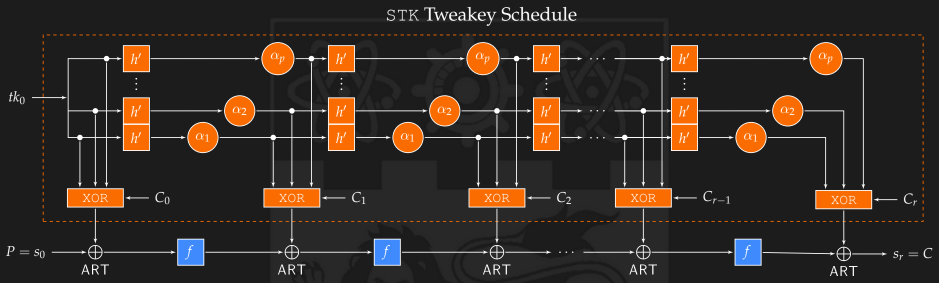
The STK construction (Superposition-TWEAKEY)



From the TWEAKEY framework to the STK construction:

- the tweakey state update function h consists in the same subfunction h' applied to each tweakey word
- the subtweakey extraction function g consists in XORing all the words together
  - reduce the implementation overhead
  - reduce the area footprint by reusing code
  - simplify the security analysis

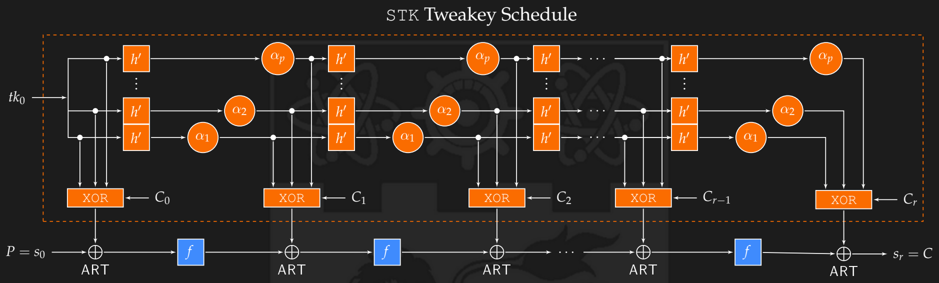
The STK construction (Superposition-TWEAKEY)



From the TWEAKEY framework to the STK construction:

- **problem**: strong interaction between the parallel branches of tweakey state
- **solution**: differentiate the parallel branches by simply using distinct multiplications in a small field

The STK construction (Superposition-TWEAKEY)



In details:

- assume the n-bit internal state of the cipher is divided into p nibbles of c bits: we divide the tweakey material into n-bit words, and then c-bit nibbles
- h' will simply be a permutation of the nibbles positions
- each nibble of the k-th tweakey word is multiplied by a value $\alpha_k \in GF(2^c)$

The STK construction: rationale

Design choices

- multiplication in $GF(2^c)$ controls the number of cancellations in g, when the subtweakeys are XORed to the internal state
- rely on a linear code to bound the number of cancellations

Implementation

- very simple transformations: linear and lightweight
- multiplications constants chosen as $1, 2, 4, \ldots$ for efficiency

Security analysis

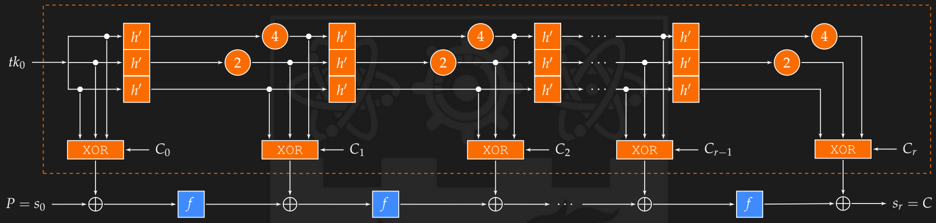
- a security analysis is now possible with STK:
  - when considering one tweakey word, we ensure that function h' is itself a good tweakey schedule
  - when considering several tweakey words, we reuse existing tools searching for good differential paths: for these tools it is easy to add the cancellation bound

Outline

1. Introduction
2. The TWEAKEY Framework
   - TWEAKEY
   - The tweakable block cipher KIASU-BC
3. The STK Construction
   - STK
   - Joltik-BC and Deoxys-BC
4. Authenticated encryption with TBC
5. Future works

STK with a 4×4 internal state matrix

STK construction (for TK−3) with a 4×4 internal state matrix



- multiplication factors are 1, 2 and 4 in $GF(2^c)$
- h' is a simple permutation of the 16 nibbles:

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix} \xmapsto{h'} \begin{pmatrix} 1 & 5 & 9 & 13 \\ 6 & 10 & 14 & 2 \\ 11 & 15 & 3 & 7 \\ 12 & 0 & 4 & 8 \end{pmatrix}$$

Joltik-BC tweakable block cipher

Joltik-BC tweakable block cipher:

- 64-bit TBC, instance of the STK construction
- two members: Joltik-BC-128 and Joltik-BC-192
  - 128 bits for TK-2: $|key| + |tweak| = 128$ (2 tweakey words)
  - 192 bits for TK-3: $|key| + |tweak| = 192$ (3 tweakey words)
- AES-like design:
  - 4-bit S-Box from the Piccolo block cipher (compact in hardware)
  - involutive MDS matrix $\Longrightarrow$ low decryption overhead
  - light constant additions to break symmetries (from LED cipher)
- Joltik-BC-128 has 24 rounds (TK-2)
- Joltik-BC-192 has 32 rounds (TK-3)
- HW implementations estimation: about 1500 GE for TK-2 version

Deoxys-BC tweakable block cipher

Deoxys-BC tweakable block cipher:

- 128-bit TBC, instance of the STK construction
- two members: Deoxys-BC-256 and Deoxys-BC-384
  - 256 bits for TK-2: $|key| + |tweak| = 256$ (2 tweakey words)
  - 384 bits for TK-3: $|key| + |tweak| = 384$ (3 tweakey words)
- the round function is **exactly** the AES round function (AES-NI)
- constants additions to break symmetries (RCON from AES key schedule)
- Deoxys-BC-256 has 14 rounds (TK-2): can replace AES-256
- Deoxys-BC-384 has 16 rounds (TK-3)
- software performances: about 1.30 c/B with AES-NI

Outline

1. Introduction
2. The TWEAKEY Framework
   - TWEAKEY
   - The tweakable block cipher KIASU-BC
3. The STK Construction
   - STK
   - Joltik-BC and Deoxys-BC
4. **Authenticated encryption with TBC**
5. Future works

Authenticated Encryption

=

Authentication + Encryption

KIASU≠, Joltik≠ and Deoxys≠

One can easily build a nonce-based parallelizable AE mode from a TBC (similar to OCB3 or TAE): simply ensure that every call to the TBC will have a distinct tweak input value

We can directly reuse the OCB3 security proofs:

- but ensuring full security instead of birthday bound
- the proofs are simpler (see ΘCB3 and OCB3 proofs)
- no long initialization required anymore: fast for short inputs

We plug KIASU-BC, Joltik-BC and Deoxys-BC
in such modes and we obtain:
KIASU≠, Joltik≠ and Deoxys≠ for nonce-respecting scenario
KIASU=, Joltik= and Deoxys= for nonce-misuse scenario

KIASU≠, KIASU= and KIASU-BC

We have two operating modes KIASU≠ and KIASU= , both built upon the same tweakable block cipher named KIASU-BC .

Operating modes:

- KIASU≠ is for nonce-respecting (based on OCB3)
- KIASU= is for nonce-misuse resistance (based on COPA)
- both modes are parallelizable

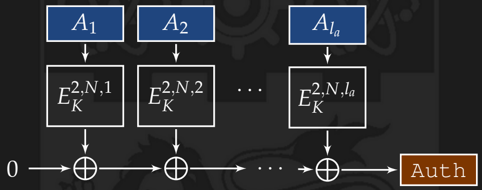
The tweakable block cipher KIASU-BC :

- message of $n = 128$ bits
- key of $k = 128$ bits
- tweak of $t = 64$ bits

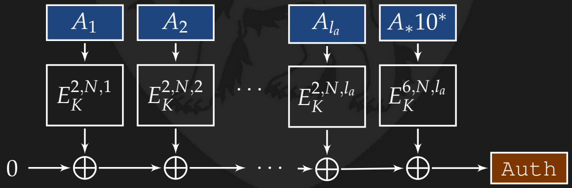
nonce-respecting mode: KIASU≠

KIASU≠ is based on OCB3

For Associated Data (full block):



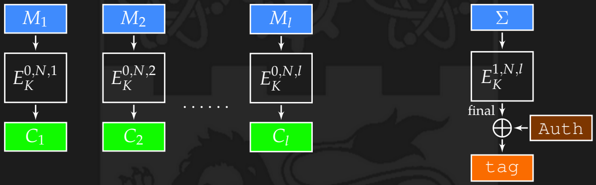
For Associated Data (partial block):



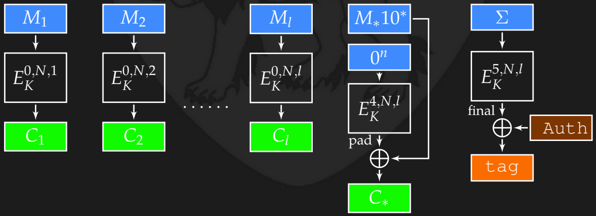
nonce-respecting mode: KIASU≠

KIASU≠ is based on OCB3

For Plaintext (full block):



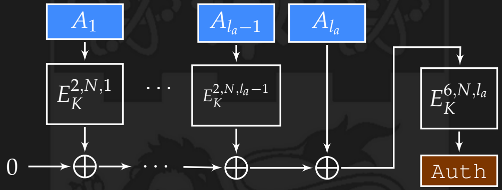
For Plaintext (partial block):



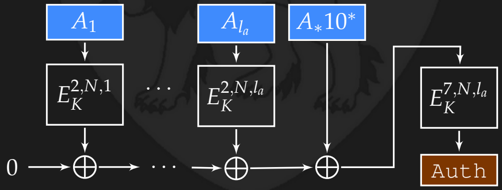
nonce-misuse resistant mode: KIASU≠

KIASU≠ is based on COPA

For Associated Data (full block):



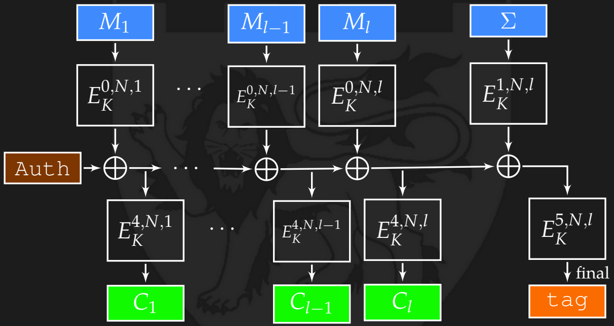
For Associated Data (partial block):



nonce-misuse resistant mode: KIASU≠

KIASU≠ is based on COPA

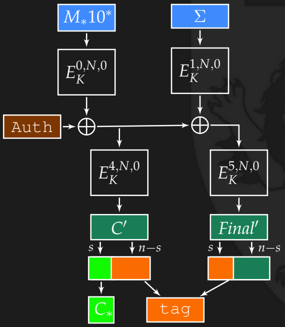
For Plaintext (full block):



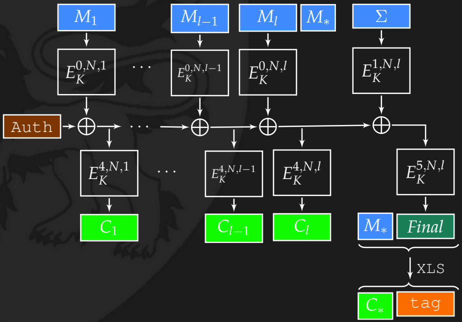
nonce-misuse resistant mode: KIASU=

KIASU= is based on COPA

For Plaintext (single partial block):



For Plaintext (partial block):



Security claims (in $\log_2$)

	Security (bits)	
nonce-respecting user	KIASU≠	KIASU=
Confidentiality for the plaintext	128	64
Integrity for the plaintext	128	64
Integrity for the associated data	128	64

	Security (bits)	
nonce-misuse user	KIASU≠	KIASU=
Confidentiality for the plaintext	none	64
Integrity for the plaintext	none	64
Integrity for the associated data	none	64

Conjectured security claims (in $\log_2$)

nonce-respecting user	Security (bits) KIASU≠	Security (bits) KIASU=
Confidentiality for the plaintext	128	128
Integrity for the plaintext	128	128
Integrity for the associated data	128	128

nonce-misuse user	Security (bits) KIASU≠	Security (bits) KIASU=
Confidentiality for the plaintext	none	64
Integrity for the plaintext	none	64
Integrity for the associated data	none	64

Outline

Future works

- cryptanalysis of STK?
- proofs for STK?
- other better/faster/stronger constructions than STK?
- adding a layer on top of KIASU to increase the tweak size ?

Thank you !