Tweakable Block Cipher Based Lightweight Cryptography

Thomas Peyrin

NTU - Singapore

Lightweight Crypto Day 2019

Tel Aviv - March 31, 2019

Outline

Outline

Lightweight cryptography ?

RFID tags to be deployed widely (supply chain management, e-passports, contactless applications, etc.)

- we need to ensure authentication and/or confidentiality
- block ciphers are used as basic blocks for RFID device authentication and privacy-preserving protocols
- it was estimated in 2005 that a basic RFID tag may have a total gate count of anywhere from 1000-10000 gates, with **only 200-2000 gates** budgeted for security

Standard block ciphers were not designed with lightweight cryptography in mind

Lightweight cryptography ?

RFID tags to be deployed widely (supply chain management, e-passports, contactless applications, etc.)

- we need to ensure authentication and/or confidentiality
- block ciphers are used as basic blocks for RFID device authentication and privacy-preserving protocols
- it was estimated in 2005 that a basic RFID tag may have a total gate count of anywhere from 1000-10000 gates, with **only 200-2000 gates** budgeted for security

~~Standard block ciphers were not designed with lightweight cryptography in mind~~
Latest `AES-128` implementations only need 1600 GE [JMPS17]
Is `AES-128` a lightweight cipher ?

Is AES-128 a lightweight cipher?

YES! Latest AES-128 implementations only need about 1600 GE

NO! This small implementation requires 1500/2000 cycles! Slow and not energy efficient.

cipher	impl. type	area (GE)	cycles	area*cycles
AES-128	1-bit serial	~1600	~1750	~2800000
AES-128	32-bit serial	~5400	54	~292000
AES-128	round based	~7200	11	~80000
SKINNY-128	1-bit serial	~1300	~7000	~9450000
SKINNY-128	round based	~2400	40	~96000
SKINNY-128	fully unrolled	~32000	1	~32000

What really matters is the flexibility of the cipher to easily offer tradeoffs

Difficult comparison

Comparing crypto algorithms
for hardware is difficult

Difficult comparison : many different platforms

Application-Specific Integrated Circuit (ASIC)

+ high-performance
+ low power consumption
− very expensive non-recurring cost
− one can't change anything once produced
− time consuming to develop

Bottom-line : for high volume production

Field-Programmable Gate Arrays (FPGA)

+ can be reprogrammed
+ simple to develop
− more waste compared to ASIC (higher recurring cost)

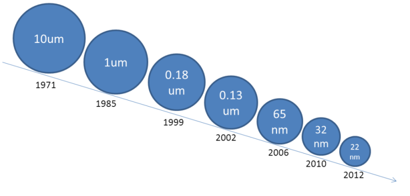
Bottom-line : for low volume production

Microcontrollers and ARM

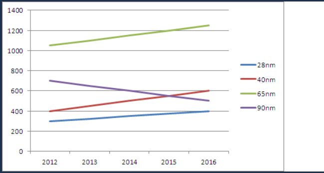
for embedded systems, mobile devices, etc.

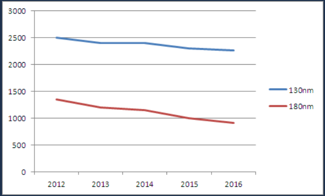
Many different platforms : ASIC

ASIC : different technology nodes



anysilicon.com





Technology nodes usage evolution

Many different platforms : ASIC

ASIC : different cell libraries (depending on the manufacturer)

Library	Logic process	NAND NOR	NOT	XOR XNOR	AND OR	ANDN ORN	NAND3 NOR3	XOR3 XNOR3	MAOI1	MOAI1
UMC	180nm	1.00	0.67	3.00	1.33	1.67	1.33	4.67	2.67	2.00
sxlib	130nm	1.00	0.75	2.25	1.25	1.25	1.25	-	-	-
TSMC	65nm	1.00	0.50	3.00	1.50	1.50	1.50	5.50	2.50	2.50
NanGate	45nm	1.00	0.67	2.00	1.33	-	1.33	-	-	-
NanGate	15nm	1.00	0.75	2.25	1.50	-	1.50	-	-	-

TABLE – Comparisons of several standard cell libraries for typical combinatorial cells. The values are given in GE.

Gate Equivalence : area of a NAND gate

- Comparing implementations with different technologies does not make sense
- Comparing only one technology gives only a narrow view.

Many different platforms : FPGA, Microcontrollers and ARM

FPGA

- **Manufacturers :** Xilinx, Altera
- **Lookup table :** 4-input LUT, 6-input LUT, etc.

Microcontrollers and ARM

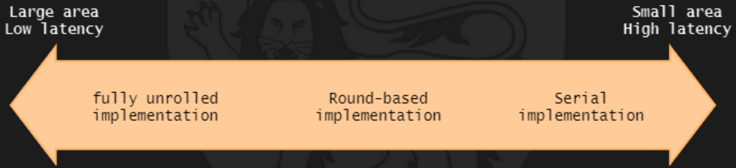
- **Word-size :** 4-bit, 8-bit, 16-bit, 32-bit
- **Memory :** ROM and RAM
- **Instructions set**

Difficult comparison : many different implementations

Implementation tradeoffs (from smaller to bigger) :

- **bit-serial** implementation (one bit at a time)
- nibble or **byte-serial** implementation (one Sbox at a time)
- **round-based** implementation (one round at a time)
- **fully unrolled** implementation (entire cipher)

Also implementation tricks (scan flip-flops vs D flip-flops)



For lightweight applications, serial and round-based implementations are the most important

Difficult comparison : many different goals

- **Area** (GE in ASIC, slices in FPGA, RAM/ROM on μcontrollers) : especially for very constrained devices, but a criterion to minimize anyway
- **Throughput :** not necessarily a critical aspect, but has to be not too bad
- **Energy :** for battery-powered devices
- **Power :** for passive RFID tags
- **Latency :** for disk encryption, automotive industry, etc.
- **FOM/FOAM :** a figure for taking into account the time/area/power/(security margin) tradeoffs
- Performance for **small messages** is particularly important, for ex. Electronic Product Code (EPC)

For lightweight applications, area/energy/power are generally the most important

Difficult comparison : other considerations to make things even worse

What about side-channels ?

Small devices will likely be easily accessible, so more subject to SCA.

What about the API

Implement or not the API ? Custom API ?

What about software implementations on the server side ?

It is likely that many lightweight devices will be communicating with a single server. The cipher has to be efficient on high-end software as well. Bitsliced implementations can help.

Chip production flow

There are many different stages in an hardware implementation : Synthesis, Place and Route, ... **we usually stop as the synthesis**. In theory, we should be measuring the silicon area of the final circuit (the only way to know for sure is to produce the chip).

In this talk, we will only consider ASICs for simplicity of comparison

Rough numbers to remember :

- a **NAND/NOR gate** : 1 GE
- an **XOR gate** : about 3 GE
- a **2-to-1 multiplexer** on 1 bit : about 2.75 GE
- a **memory bit** : about 6 GE

Outline

Primitive design : problem solved ?

Latest primitives are now almost achieving minimal possible area, for both round-based and serial implementations.

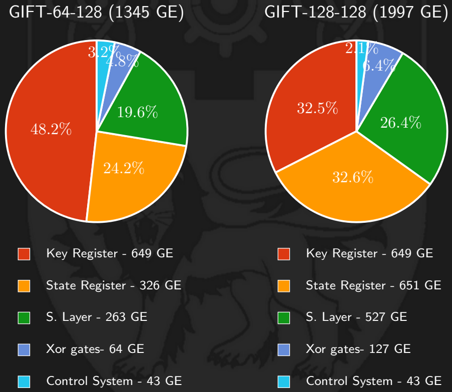


FIGURE – Component-wise area requirements for round-based implementations of `GIFT-64-128` and `GIFT-128-128`

Primitive design : problem solved ?

Latest primitives are now almost achieving minimal possible area, for both round-based and serial implementations.

Throughput on high-end servers can also be very good using bitslice implementations

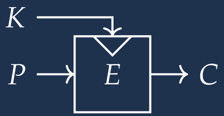
Lightweight AE modes

How to design a lightweight AE mode ?

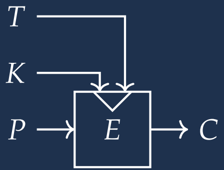
What internal primitive to use ?

(Tweakable) Block Ciphers

A block cipher is a family of permutations parametrized by a secret key K.



A tweakable block cipher is a family of permutations parametrized by a secret key K and a tweak value T [LRW02].

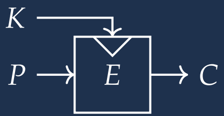


We denote

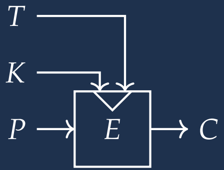
- P the n-bit plaintext
- C the n-bit ciphertext
- K the k-bit key
- T the t-bit tweak

(Tweakable) Block Ciphers

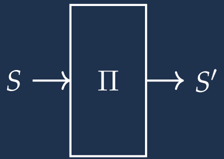
A block cipher is a family of permutations parametrized by a secret key K.



A tweakable block cipher is a family of permutations parametrized by a secret key K and a tweak value T [LRW02].



A permutation on $b = c + r$ bits, where c is the capacity and r is the rate (sponge framework [BDPV07])



Use case 1 : minimal area

Use case 1 : minimal area

In this scenario, we don't care if the ciphering process is really slow, we just want to minimize area
(typically bit-serial or word-serial implementation).

- We will cipher m-bit at a time (m is small)
- We want at least n-bit security
- We will use a n-bit key

Use case 2 : low energy consumption and lightweight

Use case 2 : low energy consumption and lightweight

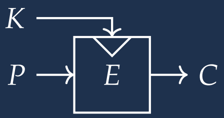
In this scenario, we want a small area and good throughput performances (typically round-based implementation)

- We will cipher n-bit at a time
- We want at least n-bit security
- We will use a n-bit key

Case of block-cipher

Issue with BC :

Most BC modes will provide only birthday security (BBB BC-based modes are not lightweight nor fast) thus for n-bit security you need to use at least $2n$-bit block cipher with n bit key, at very minimal you will need $3n$ (probably impossible ?).



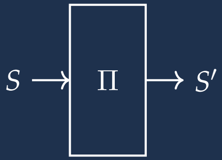
The smallest known, COFB [CIMN17], actually requires :

- $1.5n + k$ state for $n/2$-bit security, thus $3n + k = 4n$ in our scenario.
- the internal BC will handle $2n$-bit words with a $2n$-bit primitive, so rate if 1.
- minimum state is $4n$ and efficiency is $state/rate = 4n$

Case of sponges

Issue with sponges :

You need at least a capacity of $2n$, which is the minimum state size for n-bit security. If one needs to handle n-bit at a time, then $3n$ state is needed. Throughput not so great because you use a $2n$ or $3n$-bit permutation each time (this effect is usually reduced by using a non-hermetic sponge)



`SpongeAE` [BDPA11] requires :

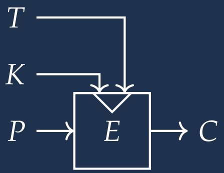
- $3n$-bit state for n-bit security with n-bit message
- the permutation works on $3n$-bit so rate is $1/3$.
- minimum state is $2n$ and efficiency is $state/rate = 9n$

What about tweakable block-ciphers ?

The case of TBC :

We will show Romulus and Remus TBC-based modes which can achieve :

- minimal state $2n$
- efficiency as low as $3n$



128-bit security

Scheme	Number of Primitive Calls	State Size (S)	Rate (R)	S/R	Inverse Free
Romulus-N1	$\lceil \frac{\lvert A\rvert-n}{2n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3.5n$	1	$3.5n$	Yes
Romulus-N2	$\lceil \frac{\lvert A\rvert-n}{1.75n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3.2n$	1	$3.2n$	Yes
Romulus-N3	$\lceil \frac{\lvert A\rvert-n}{1.75n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3n$	1	$3n$	Yes
Remus-N2	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$	$3n$	1	$3n$	Yes
COFB	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$4n$	1	$4n$	Yes
ΘCB3	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$4.5n$	1	$4.5n$	No
SpongeAE	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3n$ $(2n)$	1/3	$9n$	Yes
BEETLE	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$	$2.1n$	1/2	$4.2n$	Yes
ASCON-128	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3.5n$	1/5	$17.5n$	Yes
Ascon-128a	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3.5n$	2/5	$8.75n$	Yes

64-bit security

Scheme	Number of Primitive Calls	State Size (S)	Rate (R)	S/R	Inverse Free
Remus-N1	$\lceil \frac{\|A\|}{n} \rceil + \lceil \frac{\|M\|}{n} \rceil + 1$	$2n$	1	$2n$	Yes
Remus-N3	$\lceil \frac{\|A\|}{n} \rceil + \lceil \frac{\|M\|}{n} \rceil$	$2.06n$	1	$2.06n$	Yes
COFB	$\lceil \frac{\|A\|}{n} \rceil + \lceil \frac{\|M\|}{n} \rceil + 1$	$2.5n$	1	$2n$	Yes
ΘCB3	$\lceil \frac{\|A\|}{n} \rceil + \lceil \frac{\|M\|}{n} \rceil + 1$	$3.5n$	1	$3.5n$	No
SpongeAE	$\lceil \frac{\|A\|}{n} \rceil + \lceil \frac{\|M\|}{n} \rceil + 1$	$2n$ (n)	1/2	$4n$	Yes
BEETLE	$\lceil \frac{\|A\|}{n} \rceil + \lceil \frac{\|M\|}{n} \rceil + 2$	$1.08n$	1/2.25	$2.43n$	Yes

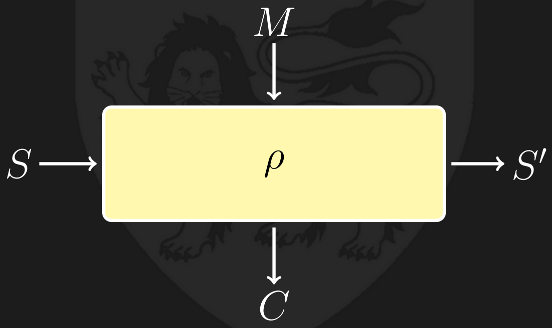
Outline

Romulus and Remus

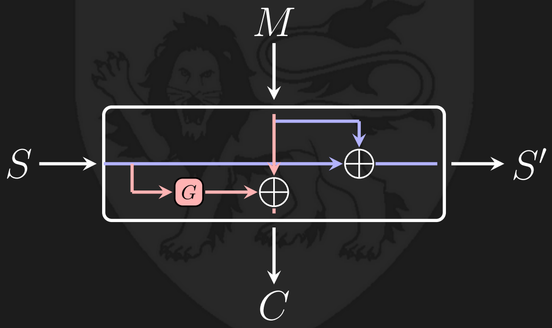
Romulus and Remus :

two lightweight TBC-based AE schemes

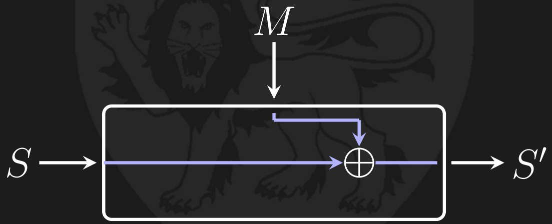
(joint work with T. Iwata, M. Khairallah and K. Minematsu)


M
S
ρ
S′
C

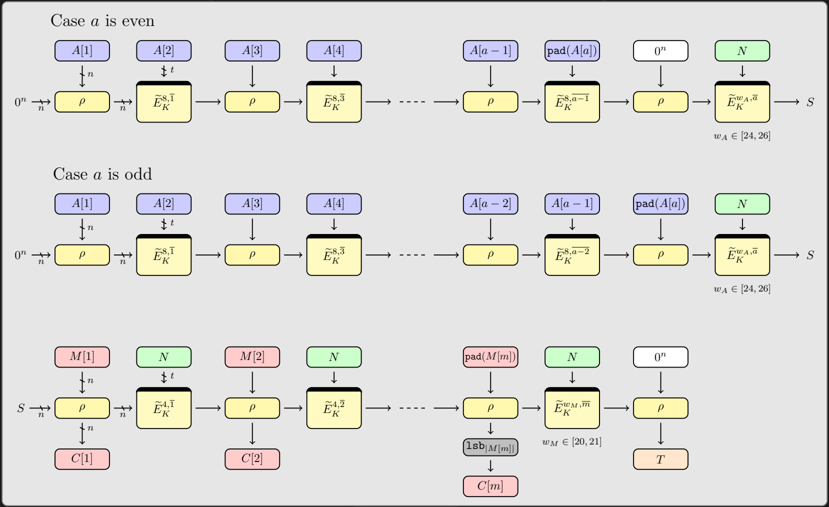
$$G = \begin{pmatrix} G_s & 0 & 0 & \dots & 0 \\ 0 & G_s & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & G_s & 0 \\ 0 & \dots & 0 & 0 & G_s \end{pmatrix}, \quad G_s = \begin{pmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1 \end{pmatrix}$$



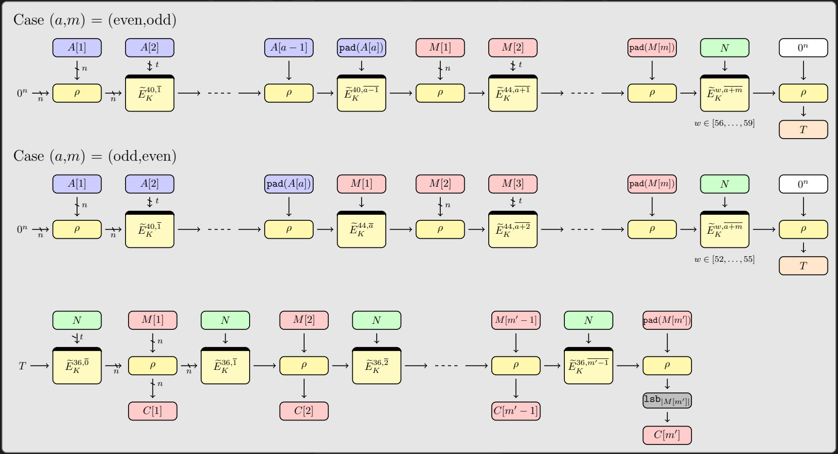
$$G = \begin{pmatrix} G_s & 0 & 0 & \dots & 0 \\ 0 & G_s & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & G_s & 0 \\ 0 & \dots & 0 & 0 & G_s \end{pmatrix}, \quad G_s = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



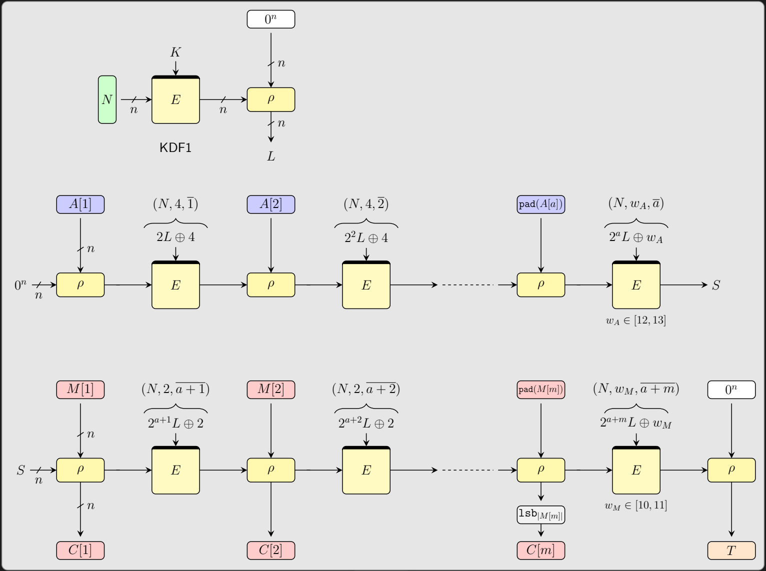
Romulus-N : nonce-respecting



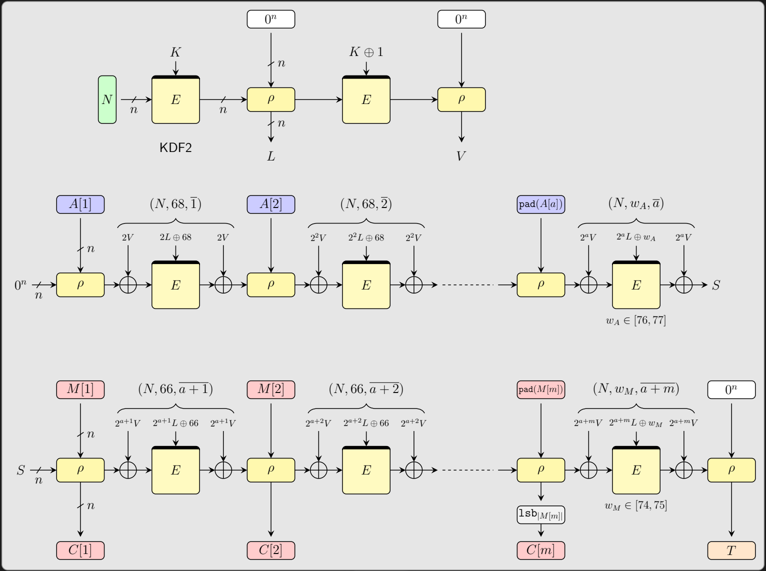
Romulus-M : nonce-misuse



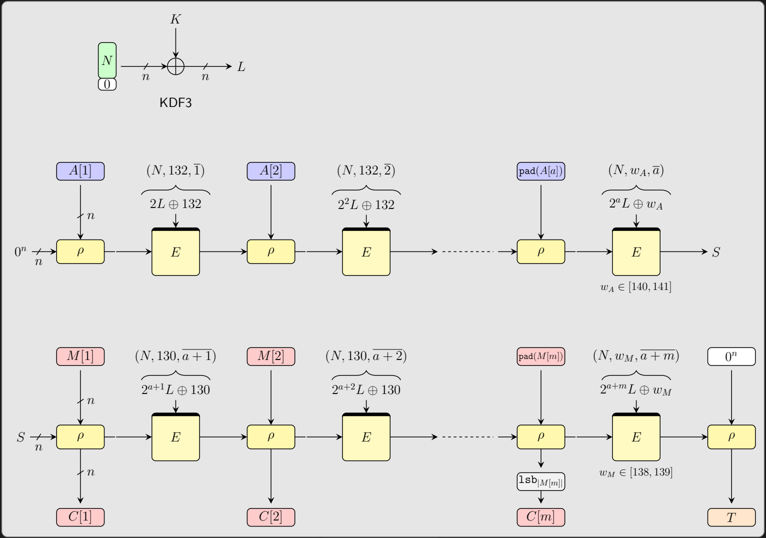
Remus-N with ICE1



Remus-N with ICE2



Remus-N with ICE3



Remus **and** Romulus **features :**

- provably secure
  (standard model for Romulus, ideal cipher for Remus)
- full 128-bit security
  (except Remus-N1/Remus-M1 and Remus-N3)
- rate 1 (rate $1 + t/n$ for authentication part in Romulus)
- minimize state registers, XORs and multiplexers
- easy nonce-misuse resistance mode
  (birthday with graceful degradation so ~full security in practice)
- no or low overhead for small messages
  Ex : 1 AD and 1 M n-bit blocks need 2 TBC calls with Romulus
- side-channel protection
  TBC protection : see threshold implementation of `SKINNY`
  Mode protection : TEDS (ePrint 2019/193) leakage resilient strategies can be easily applied to Remus
- simple and flexible
  you can trade nonce size/counter size/security/throughput for area (in contrary to sponges that don't offer area tradeoff)

Outline

Outline

SKINNY

- SKINNY -

C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi,
T. Peyrin, Y. Sasaki, P. Sasdrich and S.M. Sim

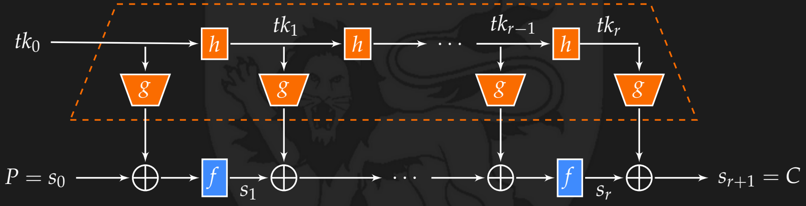
CRYPTO 2016

Our goal : to propose an academy alternative to SIMON, with better security properties and tweak capability

https://sites.google.com/site/skinnycipher/

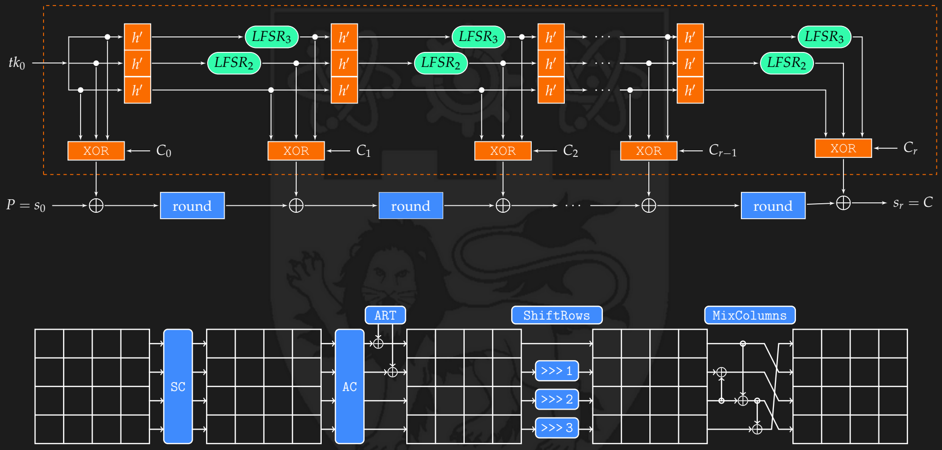
The TWEAKEY framework

The TWEAKEY framework rationale [ASIACRYPT'14] :
tweak and key should be treated the same way ⟶ tweakey



TWEAKEY **generalizes the class of key-alternating ciphers**

The SKINNY round function



SKINNY results

SKINNY versions

Block size n	Tweakey size t: n	$2n$	$3n$
64	32 rounds	36 rounds	40 rounds
128	40 rounds	48 rounds	56 rounds

SKINNY

- A ultra-lightweight family of tweakable block ciphers
- Security guarantees for differential/linear cryptanalysis (both single and related-key)
- Efficient and competitive software/hardware implementations
- Scalable security
- Suitable for most lightweight applications
- Perform and share publicly full security analysis

Security of SKINNY and comparison with SIMON and others

Ratio of rounds required for Diff/Lin trail resistance

Cipher	Single Key (SK)	Related Key (RK)
SKINNY-128-128	15/40 = 37%	19/40 = 47%
SIMON-128-128	37/68 = 54%	no bound known
AES-128	4/10 = 40%	6/10 = 60%

Ratio of attacked rounds

Cipher	Single Key (SK)	Related Key (RK)
SKINNY-128-128	18/40 = 45%	19/40 = 48%
SIMON-128-128	49/68 = 72%	? ≥ 72%
AES-128	7/10 = 70%	7/10 = 70%

There were SKINNY cryptanalysis competitions :
https://sites.google.com/site/skinnycipher/cryptanalysis-competition

Remus + SKINNY

Remus mode with SKINNY

Variant	Cycles	Area w/o interface (GE)	Area (GE)	Throughput (Gbps)	Thput/Area (Gbps/kGE)
Remus-N1	44	3106	3611	2.96	0.82
Remus-N2	44	4230	4774	3.46	0.72
Remus-M1	44(AD)/88(M)	3115	3805	2.16	0.56
Remus-M2	44(AD)/88(M)	4295	4962	2.34	0.47

TABLE – ASIC Round-Based Implementations of Remus using the TSMC 65nm standard cell library. Power and Energy are estimated at 10 Mhz.

Romulus + SKINNY

Romulus mode with SKINNY

Variant	Cycles	Area w/o interface (GE)	Area (GE)	Throughput (Gbps)	Thput/Area (Gbps/kGE)
Basic Iterative	60	5514	6620	2.78	0.42
Unrolled x4†	18	8231	9286	6.18	0.67
Unrolled x4‡	18	9632	10748	9.27	0.86

† Minimum Area ;
‡ 1 GHz ;

TABLE – ASIC Round-Based Implementations of Romulus-N1 using the TSMC 65nm standard cell library. Power and Energy are estimated at 10 Mhz.

Outline

1. Lightweight Cryptography : a Multi-Dimensional Problem
2. BC/TBC/SPONGE for AE?
3. Remus and Romulus
4. Instantiating Remus and Romulus
   - SKINNY
   - TGIF
5. Conclusion

SKINNY

- Thank Goodness It's Friday - (TGIF)

T. Iwata, M. Khairallah, K. Minematsu,
T. Peyrin, Y. Sasaki, S.M. Sim and L. Sun

Our goal : to propose a tweakable block-cipher based on `GIFT` design principles : performant everywhere

`https://sites.google.com/site/tgif-ae/`

Classical view of `GIFT-64` (CHES 2017)

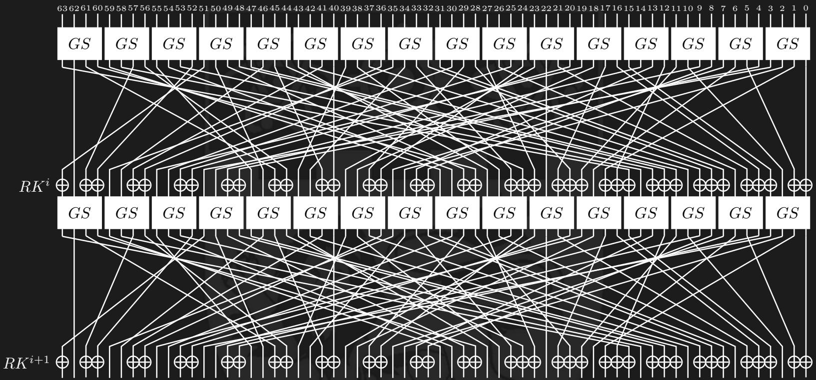


FIGURE – 2 Rounds of `GIFT-64`.

Classical view of GIFT-64 (CHES 2017)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

Input bits

slice 0

0	16	32	48
12	28	44	60
8	24	40	56
4	20	36	52

slice 1

5	21	37	53
1	17	33	49
13	29	45	61
9	25	41	57

slice 2

10	26	42	58
6	22	38	54
2	18	34	50
14	30	46	62

slice 3

15	31	47	63
11	27	43	59
7	23	39	55
3	19	35	51

Output bits

Classical view of `GIFT-64` (CHES 2017)

Input bits

slice 0

0	16	32	48
12	28	44	60
8	24	40	56
4	20	36	52

slice 1

5	21	37	53
1	17	33	49
13	29	45	61
9	25	41	57

slice 2

10	26	42	58
6	22	38	54
2	18	34	50
14	30	46	62

slice 3

15	31	47	63
11	27	43	59
7	23	39	55
3	19	35	51

Output bits

slice 0

0	12	8	4
48	60	56	52
32	44	40	36
16	28	24	20

slice 1

21	17	29	25
5	1	13	9
53	49	61	57
37	33	45	41

slice 2

42	38	34	46
26	22	18	30
10	6	2	14
58	54	50	62

slice 3

63	59	55	51
47	43	39	35
31	27	23	19
15	11	7	3

Classical view of `GIFT-64` (CHES 2017)

Input bits

slice 0

0	12	8	4
48	60	56	52
32	44	40	36
16	28	24	20

slice 1

21	17	29	25
5	1	13	9
53	49	61	57
37	33	45	41

slice 2

42	38	34	46
26	22	18	30
10	6	2	14
58	54	50	62

slice 3

63	59	55	51
47	43	39	35
31	27	23	19
15	11	7	3

Output bits

slice 0

0	48	32	16
4	52	36	20
8	56	40	24
12	60	44	28

slice 1

17	1	49	33
21	5	53	37
25	9	57	41
29	13	61	45

slice 2

34	18	2	50
38	22	6	54
42	26	10	58
46	30	14	62

slice 3

51	35	19	3
55	39	23	7
59	43	27	11
63	47	31	15

Classical view of GIFT-64 (CHES 2017)

slice 0

0	48	32	16
4	52	36	20
8	56	40	24
12	60	44	28

slice 1

17	1	49	33
21	5	53	37
25	9	57	41
29	13	61	45

slice 2

34	18	2	50
38	22	6	54
42	26	10	58
46	30	14	62

slice 3

51	35	19	3
55	39	23	7
59	43	27	11
63	47	31	15

Input bits

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

Output bits

A new view of GIFT-64 (unpublished)

We found a new way to represent and compute GIFT-64

A new view of GIFT-64 (unpublished)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

Input bits

← ←← ←←←

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

5	9	13	1
21	25	29	17
37	41	45	33
53	57	61	49

slice 2

10	14	2	6
26	30	18	22
42	46	34	38
58	62	50	54

slice 3

15	3	7	11
31	19	23	27
47	35	39	43
63	51	55	59

Output bits

A new view of `GIFT-64` (unpublished)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

5	9	13	1
21	25	29	17
37	41	45	33
53	57	61	49

slice 2

10	14	2	6
26	30	18	22
42	46	34	38
58	62	50	54

slice 3

15	3	7	11
31	19	23	27
47	35	39	43
63	51	55	59

Input bits

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

↑

slice 1

21	25	29	17
37	41	45	33
53	57	61	49
5	9	13	1

↑↑

slice 2

42	46	34	38
58	62	50	54
10	14	2	6
26	30	18	22

↑↑↑

slice 3

63	51	55	59
15	3	7	11
31	19	23	27
47	35	39	43

Output bits

A new view of GIFT-64 (unpublished)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

21	25	29	17
37	41	45	33
53	57	61	49
5	9	13	1

slice 2

42	46	34	38
58	62	50	54
10	14	2	6
26	30	18	22

slice 3

63	51	55	59
15	3	7	11
31	19	23	27
47	35	39	43

Input bits

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

→

slice 1

17	21	25	29
33	37	41	45
49	53	57	61
1	5	9	13

→→

slice 2

34	38	42	46
50	54	58	62
2	6	10	14
18	22	26	30

→→→

slice 3

51	55	59	63
3	7	11	15
19	23	27	31
35	39	43	47

Output bits

A new view of `GIFT-64` (unpublished)

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

17	21	25	29
33	37	41	45
49	53	57	61
1	5	9	13

slice 2

34	38	42	46
50	54	58	62
2	6	10	14
18	22	26	30

slice 3

51	55	59	63
3	7	11	15
19	23	27	31
35	39	43	47

Input bits

slice 1: ↓ slice 2: ↓↓ slice 3: ↓↓↓

slice 0

0	4	8	12
16	20	24	28
32	36	40	44
48	52	56	60

slice 1

1	5	9	13
17	21	25	29
33	37	41	45
49	53	57	61

slice 2

2	6	10	14
18	22	26	30
34	38	42	46
50	54	58	62

slice 3

3	7	11	15
19	23	27	31
35	39	43	47
51	55	59	63

Output bits

A new view of GIFT-64 (unpublished)

We found a new way to represent and compute GIFT-64

- It will compute exactly 4 rounds of GIFT-64
- It “magically" comes back to the normal representation at the end of the 4 rounds
- Gives excellent micro-controller performance, without use of tables
- Exactly the same cipher, so we maintain GIFT best performances on ASIC (energy and area)

A new view of GIFT-128 (unpublished)

It also works for GIFT-128, but :

- more complicated
- you have to "reset" the representation after 4 rounds, so less gain than for GIFT-64

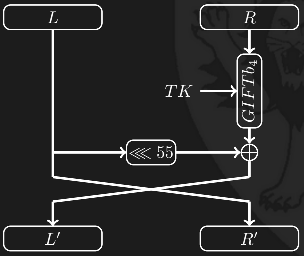
It will benefit to NIST submisssions that use GIFT-128 :

- GIFT-COFB
  (S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S.M. Sim, Y. Todo)
- SUNDAE-GIFT
  (S. Banik, A. Bogdanov, T. Peyrin, Y. Sasaki, S.M. Sim, E. Tischhauser, Y. Todo)

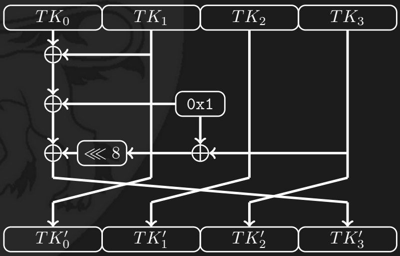
GIFT to TGIF-TBC

Main ideas underlying TGIF-TBC :

- we load the input and save the output directly in bitslice representation to avoid packing/unpacking
- 4 rounds of GIFT-64 is very efficient, so we use it as a blackbox
- ... in a Misty construction to get 128-bit block
- designed a strong and lightweight key schedule, specially adapted to the bitslice representation



TGIF-TBC step function



TGIF-TBC key schedule

TGIF-TBC features

Results for TGIF-TBC :

- surprisingly resistant against diff/lin attacks, even in related-tweakey model
- this allowed to get even better performances than GIFT-128 : same for ASIC, better for μ-controllers and high-end servers
- now with a tweak capability
- special tweakey schedule feature : the tweakey state naturally comes back to its original value at the end

Ratio of rounds required for Diff/Lin trail resistance

Cipher	Single Key (SK)	Related Key (RK)
TGIF-TBC	12/18 = 67%	12/18 = 67%
GIFT-128	25/40 = 62%	no bound known
SKINNY-128-128	15/40 = 37%	19/40 = 47%
SIMON-128-128	37/68 = 54%	no bound known
AES-128	4/10 = 40%	6/10 = 60%

TGIF = Remus + TGIF−TBC

TGIF = Remus mode with TGIF−TBC

Variant	Cycles	Area w/o interface (kGE)	Area (kGE)	Throughput (Gbps)	Thput/Area (Gbps/kGE)
TGIF-N1	22	4307	4813	3.68	0.76
TGIF-N2	22	5406	5950	3.68	0.62
TGIF-M1	22(AD)/44(M)	4250	4940	2.45	0.5
TGIF-M2	22(AD)/44(M)	5569	6236	2.45	0.39
TGIF-N1	22	-	5945	5.9	1
TGIF-N2	22	-	7009	5.9	0.85
TGIF-M1	22(AD)/44(M)	-	6133	3.87	0.63
TGIF-M2	22(AD)/44(M)	-	7345	3.87	0.52

TABLE – ASIC Step-Based Implementations of TGIF using the TSMC 65nm standard cell library at minimum area. Power and Energy are estimated at 10 Mhz.

Outline

1. Lightweight Cryptography : a Multi-Dimensional Problem
2. BC/TBC/SPONGE for AE ?
3. Remus and Romulus
4. Instantiating Remus and Romulus
   - SKINNY
   - TGIF
5. Conclusion

Future Works

Possible future works :

- Analysis of NIST submitted designs
- More TBC-based modes for various cryptographic needs ! Especially side-channel resistance
- Sponges are good for absorbing (full state absorption), not for squeezing or encryption (waste of computation)
- Hybrid Sponge-TBC modes ? best of both worlds : sponge for absorb, TBC for encryption

Thank you !