Key-Schedule in (Lightweight) Symmetric-Key Cryptography

Thomas Peyrin

NTU - Singapore

TCCM-CACR 2016

Yinchuan, China - August 31, 2016

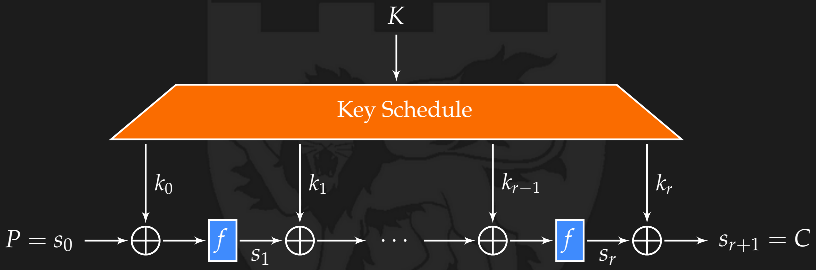
Outline

1. Permutations for symmetric key primitives
2. Key schedule role
   - ▷ Meet-in-the-middle attacks
   - ▷ Slide attacks
   - ▷ Symmetry attacks
   - ▷ Weak keys
   - ▷ Related-key attacks
3. Key schedule constructions
   - ▷ AES and PRESENT
   - ▷ WHIRLPOOL key schedule
   - ▷ LED key schedule
   - ▷ The TWEAKEY framework
4. The Skinny tweakable block cipher
   - ▷ SKINNY security
   - ▷ SKINNY performances
5. Future directions and open problems

Iterated block ciphers

An iterated block cipher is composed of two parts :

- an internal permutation f repeated r times (also named round function)
- a key schedule that generates $r+1$ subkeys $K \rightarrow (k_0, \ldots, k_r)$



For a compression function, the key schedule is also named the message expansion

Iterated block ciphers

An iterated block cipher is composed of two parts :

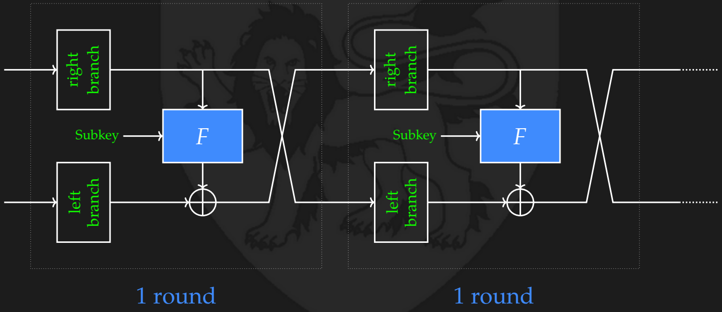
- an internal permutation f repeated r times (also named round function)
- a key schedule that generates $r+1$ subkeys $K \rightarrow (k_0, \ldots, k_r)$

For a compression function, the key schedule is also named the message expansion

Permutations

We know how to design a good permutation :

- **Feistel network** - DES, SHA-2
- **Substitution-Permutation network (SPN)** - AES, Keccak (SHA-3)



Permutations

We know how to design a good permutation :

- **Feistel network** - DES, SHA−2
- **Substitution-Permutation network (SPN)** - AES, Keccak (SHA−3)

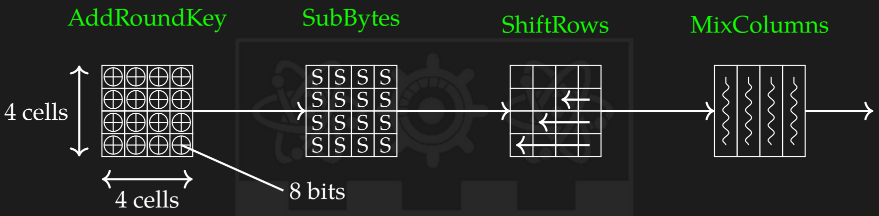
Permutations

We know how to design a good permutation :

- **Feistel network** - DES, SHA-2
- **Substitution-Permutation network (SPN)** - AES, Keccak (SHA-3)

Many recent primitives try to use only permutations to avoid the key schedule (sponge functions, Grøstl, LED)

Ex : the AES−128 round function



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

- **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 (branching number = 5)

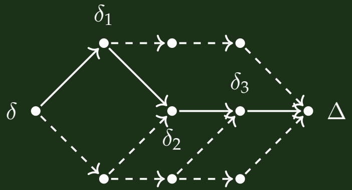
10/12/14 rounds for AES−128/AES−192/AES−256

Differentials and differential characteristics

Differential (characteristics)

- used in differential cryptanalysis
- sequence of differences at each round for an iterated primitive
- a differential is a collection of characteristics

Example



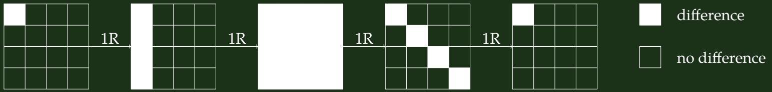
- $\delta \to \Delta$ is a differential
- $\delta \to \delta_1 \to \delta_2 \to \delta_3 \to \Delta$ is a differential characteristic
- $\mathbb{P}(\delta \to \delta_1 \to \delta_2 \to \delta_3 \to \Delta)$ is its differential probability

Differentials and differential characteristics

Differential characteristics

- differential characteristics are easier to handle than differentials
  $\Longrightarrow$ we usually focus on characteristics
- designers' goal :
  upper-bound the differential probability of characteristics

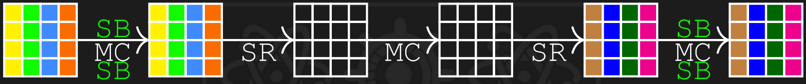
Example : 4-round AES



- 4-round characteristic with 25 active S-Boxes (minimal)
- AES S-Box : $p_{max} = 2^{-6}$
- differential probability : $p \leq 2^{-6\times 25} = 2^{-150}$

Proving 25 active Sboxes for 4 AES rounds (part I)

Proving 25 active Sboxes for 4 AES rounds (part II)



Theorem 1

Any active Super-box will contain at least 5 active Sboxes

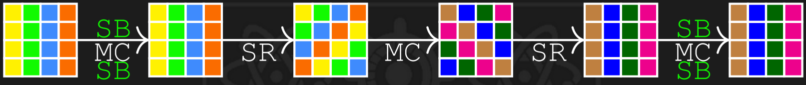
Theorem 2

There will be at least 5 active Super-boxes in 4 AES rounds

Corollary

There are at least 25 active Sboxes in 4 AES rounds

Proving 25 active Sboxes for 4 AES rounds (part II)



Theorem 1

Any active Super-box will contain at least 5 active Sboxes

Theorem 2

There will be at least 5 active Super-boxes in 4 AES rounds

Corollary

There are at least 25 active Sboxes in 4 AES rounds

Min. num. of active Sboxes for AES in the SK model

Rounds	1	2	3	4	5	6	7	8	9	10
min	1	5	9	25	26	30	34	50	51	55

Question :

What would this table look like for the AES structure in the RK model ?

Outline

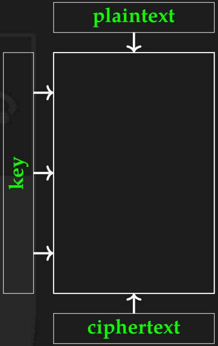
1. Permutations for symmetric key primitives
2. Key schedule role
    - Meet-in-the-middle attacks
    - Slide attacks
    - Symmetry attacks
    - Weak keys
    - Related-key attacks
3. Key schedule constructions
    - AES and PRESENT
    - WHIRLPOOL key schedule
    - LED key schedule
    - The TWEAKEY framework
4. The Skinny tweakable block cipher
    - SKINNY security
    - SKINNY performances
5. Future directions and open problems

Outline

Meet-in-the-middle attacks

Attack sketch :

- choose two independent subparts K_F and K_B of the key K and guess the remaining bits $K \setminus K_F \cap K_B$
- compute X forward from the plaintext (does not depend on K_B)
- compute X backward from the ciphertext (does not depend on K_F)
- check if you get a match on X. If so, test this key candidate.
- complex improvements exist (splice-and-cut, etc.)



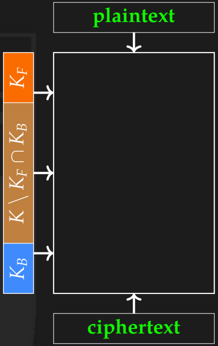
Can be used for **key-recovery** on block ciphers or **preimage** on hash functions

Meet-in-the-middle attacks are strongly depending on the strength of the key-schedule

Meet-in-the-middle attacks

Attack sketch :

- choose two independent subparts K_F and K_B of the key K and guess the remaining bits $K \setminus K_F \cap K_B$
- compute X forward from the plaintext (does not depend on K_B)
- compute X backward from the ciphertext (does not depend on K_F)
- check if you get a match on X. If so, test this key candidate.
- complex improvements exist (splice-and-cut, etc.)



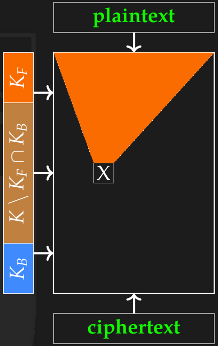
Can be used for **key-recovery** on block ciphers or **preimage** on hash functions

Meet-in-the-middle attacks are strongly depending on the strength of the key-schedule

Meet-in-the-middle attacks

Attack sketch :

- choose two independent subparts K_F and K_B of the key K and guess the remaining bits $K \setminus K_F \cap K_B$
- compute X forward from the plaintext (does not depend on K_B)
- compute X backward from the ciphertext (does not depend on K_F)
- check if you get a match on X. If so, test this key candidate.
- complex improvements exist (splice-and-cut, etc.)



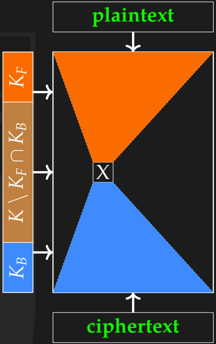
Can be used for **key-recovery** on block ciphers or **preimage** on hash functions

Meet-in-the-middle attacks are strongly depending on the strength of the key-schedule

Meet-in-the-middle attacks

Attack sketch :

- choose two independent subparts K_F and K_B of the key K and guess the remaining bits $K \setminus K_F \cap K_B$
- compute X forward from the plaintext (does not depend on K_B)
- compute X backward from the ciphertext (does not depend on K_F)
- check if you get a match on X. If so, test this key candidate.
- complex improvements exist (splice-and-cut, etc.)



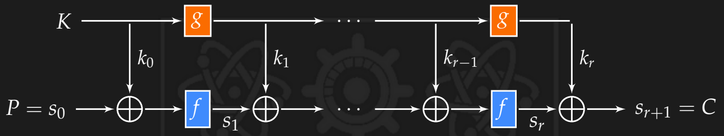
Can be used for **key-recovery** on block ciphers or **preimage** on hash functions

Meet-in-the-middle attacks are strongly depending on the strength of the key-schedule

Outline

1. Permutations for symmetric key primitives
2. Key schedule role
   - Meet-in-the-middle attacks
   - Slide attacks
   - Symmetry attacks
   - Weak keys
   - Related-key attacks
3. Key schedule constructions
   - AES and PRESENT
   - WHIRLPOOL key schedule
   - LED key schedule
   - The TWEAKEY framework
4. The Skinny tweakable block cipher
   - SKINNY security
   - SKINNY performances
5. Future directions and open problems

Slide attacks



Slide attacks :

- can happen if the very same round function f_k is used to build the permutation
- find a slid pair $P' = f_k(P \oplus k)$, then you will have $C' = f_k(C)$
- once a slid pair is found, easy to recover the key if f_k is weak enough

To prevent them :

Easy to patch using constants or a counter in the key schedule or in the internal state function

Slide attacks

Slide attacks :

- can happen if the very same round function f_k is used to build the permutation
- find a slid pair $P' = f_k(P \oplus k)$, then you will have $C' = f_k(C)$
- once a slid pair is found, easy to recover the key if f_k is weak enough

To prevent them :

Easy to patch using constants or a counter in the key schedule or in the internal state function

Slide attacks

$$P = s_0 \oplus k \longrightarrow \boxed{f_k} \xrightarrow[s_1]{} \boxed{f_k} \xrightarrow[s_2]{} \boxed{f_k} \xrightarrow[s_3]{} \cdots \longrightarrow \boxed{f_k} \xrightarrow[s_{r-1}]{} \boxed{f_k} \xrightarrow[s_r]{} \boxed{f_k} \longrightarrow s_{r+1} = C$$

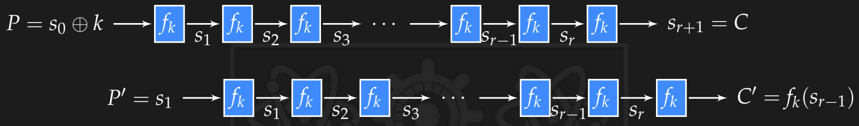
Slide attacks :

- can happen if the very same round function f_k is used to build the permutation
- find a slid pair $P' = f_k(P \oplus k)$, then you will have $C' = f_k(C)$
- once a slid pair is found, easy to recover the key if f_k is weak enough

To prevent them :

Easy to patch using constants or a counter in the key schedule or in the internal state function

Slide attacks



Slide attacks :

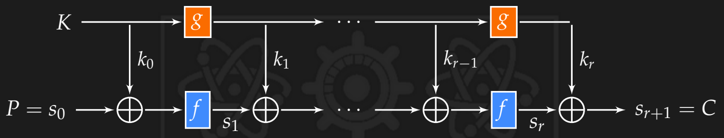
- can happen if the very same round function f_k is used to build the permutation
- find a slid pair $P' = f_k(P \oplus k)$, then you will have $C' = f_k(C)$
- once a slid pair is found, easy to recover the key if f_k is weak enough

To prevent them :

Easy to patch using constants or a counter in the key schedule or in the internal state function

Outline

Symmetry attacks



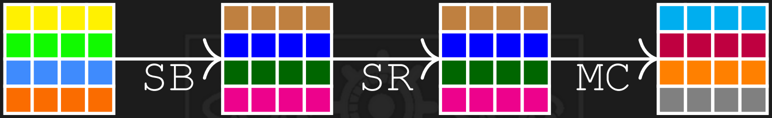
Symmetry attacks :

- can happen if a certain property can be maintained after application of f_k
- allows to maintain a low entropy in the internal state
- more generally : invariant subspace attacks

To prevent them :

Easy to patch using constants or a counter in the key schedule or in the internal state function

Symmetry attacks



Symmetry attacks :

- can happen if a certain property can be maintained after application of f_k
- allows to maintain a low entropy in the internal state
- more generally : invariant subspace attacks

To prevent them :

Easy to patch using constants or a counter in the key schedule or in the internal state function

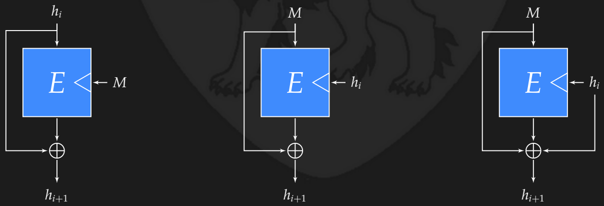
Outline

Weak-keys

A weak-key class is a set of keys for which the attack can break the cipher faster than exhaustive search

Weak-keys for block cipher and hash functions

- weak-keys are not too problematic for a block cipher as long as the weak-key class remains small
- situation is completely different for a hash function : a single weak key can potentially be catastrophic (ex. IDEA cipher)



Outline

1. Permutations for symmetric key primitives
2. Key schedule role
   - Meet-in-the-middle attacks
   - Slide attacks
   - Symmetry attacks
   - Weak keys
   - Related-key attacks
3. Key schedule constructions
   - AES and PRESENT
   - WHIRLPOOL key schedule
   - LED key schedule
   - The TWEAKEY framework
4. The Skinny tweakable block cipher
   - SKINNY security
   - SKINNY performances
5. Future directions and open problems

Related-key attacks for block ciphers

The related-key security model

The attacker is allowed to make queries to the key K, but also to other keys K', K'', etc. “related” to the key K

Why studying related-keys attacks ?

- some protocols might use simple updates to generate new keys
- related-key analysis helps to understand hash functions
- more generally, in the ideal case, a cipher shouldn’t have any structural flaw, so we could even extend this model to known-key/chosen-key attacker

A LOT of block ciphers have been broken in this model (`AES-256` for example)

Message expansion for hash function

"related-key" attacks are actually the base of most hash function collision attacks

The case of hash functions :

- key-schedule for block ciphers = message expansion for hash functions
- the message expansion is crucial in a hash function, because fully controlled by the attacker
- must resist collision attacks, but also any distinguishing property

A LOT of hash functions have been broken because of an insufficiently secure message expansion (`SHA-0`, `SHA-1`, many `SHA-3` candidates for example)

Outline

Key schedule design

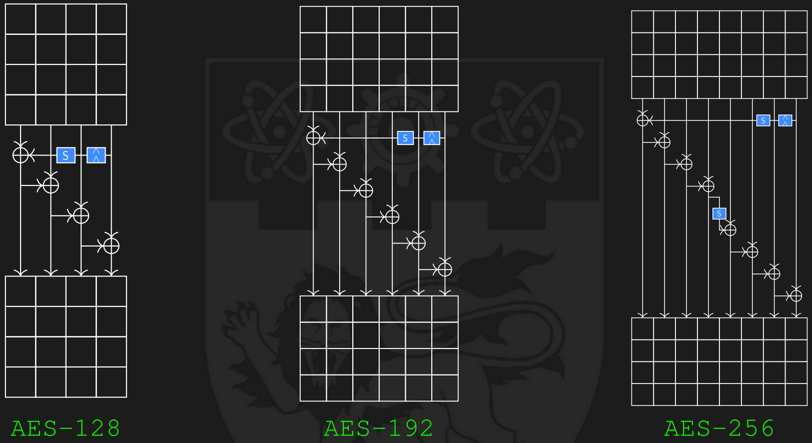
we don't really know
how to design an efficient and secure key schedule

Our current knowledge for building key schedules/message expansion is sparse :

- general technique use ad-hoc KS to decorrelate the KS and the internal BC, so hard ot prove anything and hard to analyse
- AES has a rather efficient key schedule (about 25% to 40% of the internal permutation part), but no clue about its security
- in order to get simple provable confidence in the key schedule, designers proposed inefficient solutions :
    - WHIRLPOOL has a very strong message expansion, but then one round is not efficient
    - LED has no key schedule, but requires more rounds to resist RK

Outline

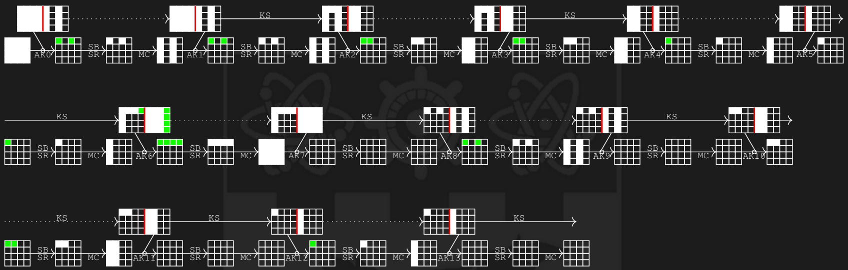
The AES key schedules



Rationale :

- XORs for inter-column diffusion, shift for inter-row diffusion, Sbox for non-linearity, counter to break symmetries
- quite different from the AES round function

Security issues with the AES key schedule



Related-key attacks on the full AES−256 and AES−192

- existence of 2-round local collision paths [BKN09]
- 14-round path with only 24 active Sboxes (5 in the key schedule, 19 in the internal state)
- later improved in [BK09] using boomerang technique (since very good small differential paths exist) : key recovery attack with $2^{99.5}$ time and data
- harder to attack AES−192 and so far no attack on AES−128

Proven bounds for AES-128

Single-key model

Rounds	1	2	3	4	5	6	7	8	9	10
min	1	5	9	25	26	30	34	50	51	55

Related-key model (truncated differences)

Rounds	1	2	3	4	5	6	7	8	9	10
min	0	1	3	9	11	13	15	21	23	25

Related-key model (actual differences)

Rounds	1	2	3	4	5	6	7	8	9	10
min	0	1	5	13	17	?	?	?	?	?

The PRESENT key schedule

PRESENT is a 64-bit block cipher - based on SPN, but using 4-bit Sboxes and bit permutation as permutation layer.

The key schedule of the PRESENT−80 block cipher

- The key is 80 bits and the subkeys 64 bits
- **Extract :** the round subkey is the 64 MSB of the key state
- **Shift :** rotate the key state by 19 bit positions to the right
- **Sbox :** apply one Sbox to the 4 MSBs of the key state
- **Counter :** add a 5-bit round counter to the key state

- very simple and hardware friendly
- quite different from the round function
- still no related-key attack on full PRESENT
- even better : the best attacks on PRESENT are not in related-key

Outline

The key-schedule of WHIRLPOOL internal block cipher

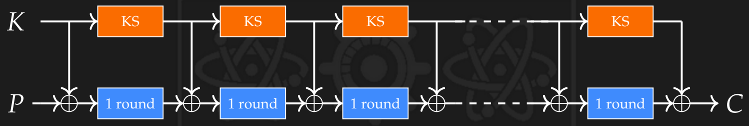
Recent lessons learned in block ciphers design :

- **designing key schedules seems hard**
- obtaining **security proofs** when also considering differences in the key schedule seems hard as well

WHIRLPOOL rationale : use an entire round function as key schedule update

- **only leverages the quality of the permutation** since we do know how to build good permutations
- trivial to **prove a minimal number of active Sboxes in the RK model**

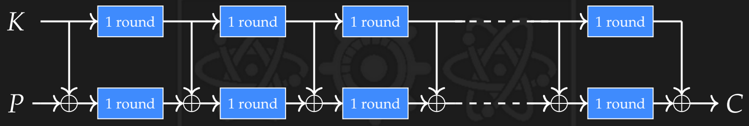
The key-schedule of WHIRLPOOL internal block cipher



Issues with WHIRLPOOL's key schedule :

- security is greatly reduced when used inside a hash construction ([LMRRS09]), but probably ok when used in a classical block cipher scenario (unknown key)
- it is quite slow ($\times 2$ slower if a new key has to be used)

The key-schedule of WHIRLPOOL internal block cipher



Issues with WHIRLPOOL's key schedule :

- security is greatly reduced when used inside a hash construction ([LMRRS09]), but probably ok when used in a classical block cipher scenario (unknown key)
- it is quite slow ($\times 2$ slower if a new key has to be used)

Outline

The key-schedule of LED

Recent lessons learned in block ciphers design :

- **designing key schedules seems hard**
- obtaining **security proofs** when also considering differences in the key schedule seems hard as well

LED rationale : use NO key schedule

- much **simpler for cryptanalysts** : not relying on the difficulty to analyze (a lot of cryptanalysis has been performed since publication of LED)
- **only leverages the quality of the permutation** since we do know how to build good permutations
- **you can directly hardwire the key** in some particular scenarios
- can benefit from **security proofs** (see recent security proofs on iterated Even-Mansour schemes)
- easy to **prove a minimal number of active Sboxes in the RK model**

The key-schedule of LED : first attempt

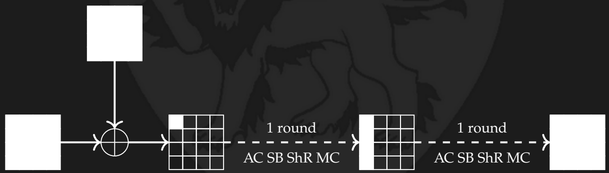
Key repeated every round

Paths exist with only **1 active Sbox per round** on average

The key-schedule of LED : second attempt

Key repeated every two rounds

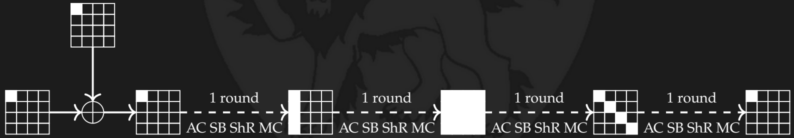
Paths exist with only **2.5 active Sboxes per round** on average



The key-schedule of LED : third attempt

Key repeated every four rounds

Paths exist with only **3.125 active Sboxes per round** on average



The key-schedule of LED

For 64-bit key :

XOR the key to the internal state **every four rounds**, for a total of **8 steps (or 32 rounds)** :

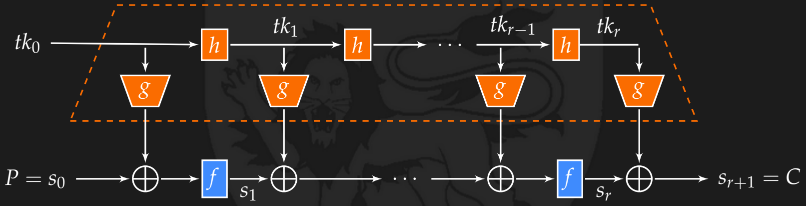
For 128-bit key :

Divide the key into **two equal chunks** K_1 and K_2 and alternatively XOR them to the internal state **every four rounds**, for a total of **12 steps (or 48 rounds)** :

Outline

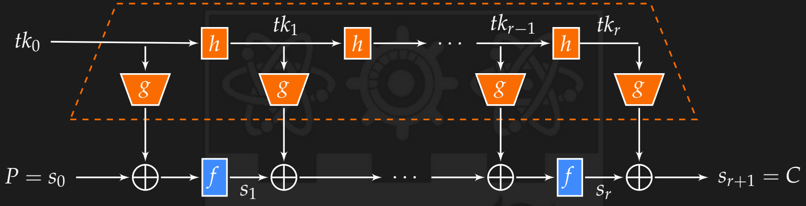
The TWEAKEY framework

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 TWEAKEY framework



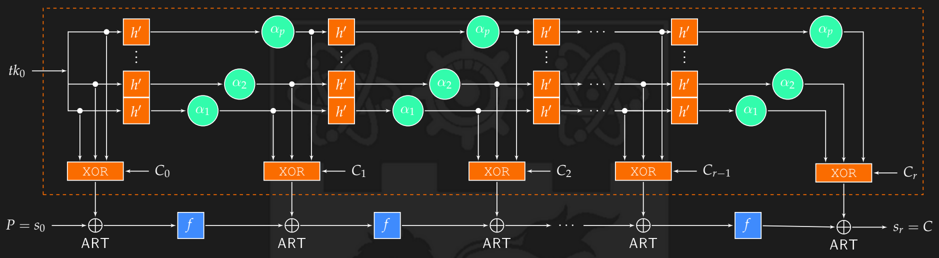
The main issue :

adding more tweakey state makes the security drop, or renders security hard to study, even for automated tools

Idea : the STK construction (Superposition-TWEAKEY)

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

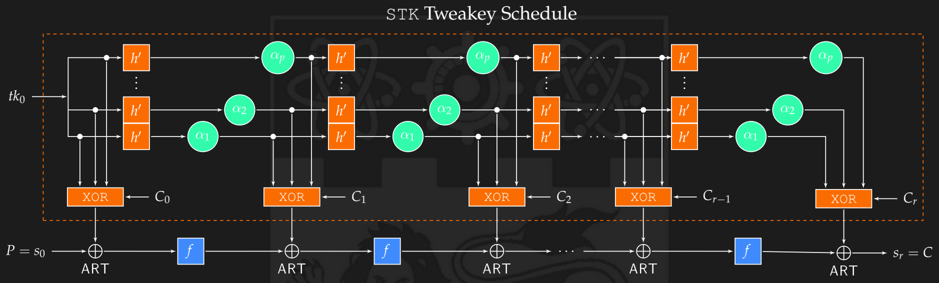
The STK construction (Superposition-TWEAKEY)



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 (for example a simple permutation of the cells positions)
- the subtweakey extraction function g consists in XORing all the words together
  - reduce the implementation overhead
  - 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 (for example by simply using distinct multiplications in a small field or LFSRs)

The STK construction : rationale

Design choices

- very simple transformations : linear and lightweight
- multiplication in $GF(2^c)$ or LFSRs control the number of cancellations in g, when the subtweakeys are XORed to the internal state
- one can bound the number of cancellations

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

SKINNY website

Joint work with C. Beierle, S. Kölbl, G. Leander, A. Moradi, Y. Sasaki, P. Sasdrich and S.M. Sim
(CRYPTO 2016)

Paper, Specifications, Results and Updates available at :
https://sites.google.com/site/skinnycipher/

Any new cryptanalysis of SKINNY is welcome !

SKINNY goals and results

Goals

- Provide an alternative to NSA-designed SIMON block cipher
- Construct a lightweight (tweakable) block cipher
- Achieve scalable security
- Suitable for most lightweight applications
- Perform and share full security analysis
- Efficient software/hardware implementations in many scenarios

Results

- SKINNY family of (tweakable) block ciphers
- Block sizes n : 64 and 128 bits
- Various key+tweak sizes : n, $2n$ and $3n$ bits
- Security guarantees for differential/linear cryptanalysis (both single and related-key)
- Efficient and competitive software/hardware implementations
  - Round-based SKINNY-64-128 : 1696 GE (SIMON : 1751 GE)
  - on Skylake (avx2) : 2.78 c/B (SIMON : 1.81 c/B) for fixed-key

SKINNY general design strategy

- Start from weak crypto components, but providing very efficient implementations
  - Opposed to AES : strong Sbox and diffusion ⇒ only 10 rounds
  - Similar to SIMON : only AND/XOR/ROT ⇒ many rounds
- Reuse AES well-understood design
- Remove all operations not strictly necessary to security
- **Result : removing *any* operations from SKINNY results in an unsecure cipher**

SKINNY specifications : overview

Specifications

- SKINNY has a state of either 64 bit ($s = 4$) or 128 bits ($s = 8$).
- tweakey schedule generalises the **STK** construction
- Internal state IS : viewed as a 4×4 matrix of s-bit elements.
  $\Rightarrow |IS| = n = 16s \in \{64, 128\}$.
- The tweakey size can be n, $2n$ or $3n$.

Number of rounds

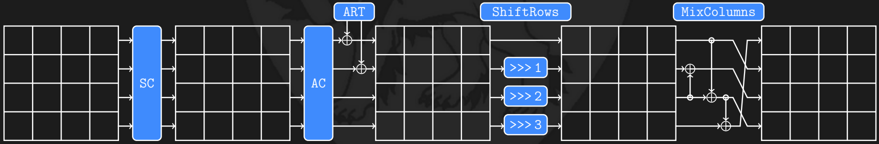
	Tweakey size		
Block size n	n	$2n$	$3n$
64	32	36	40
128	40	48	56

Comparison : SKINNY-64-128 has 36 rounds, SIMON-64-128 has 44 rounds.

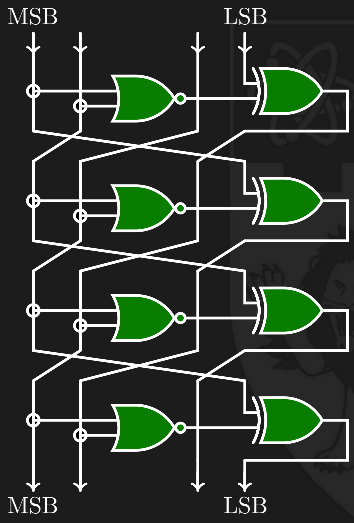
SKINNY round function

AES-like round function

- **SubCells (SC)** : Application of a s-bit Sbox to all 16 cells
- **AddConstants (AC)** : Inject round constants in the state
- **AddRoundTweakey (ART)** : Extract and inject the subtweakeys to half the state
- **ShiftRows (SR)** : Right-rotate Line i by i positions
- **MixColumns (MC)** : Multiply the state by a binary matrix



SKINNY 4-bit Sbox



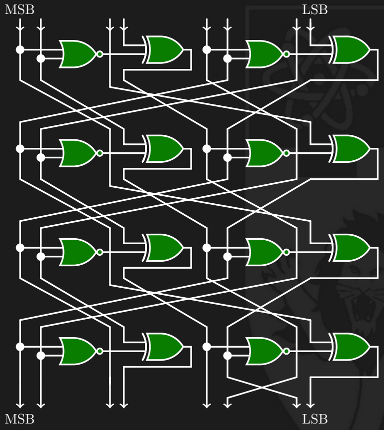
$\mathcal{S}_4$: 4-bit Sbox for SKINNY-64-*

- Almost PICCOLO Sbox
- Implementation : 4 NOR and 4 XOR
- Hardware cost : 12 GE

Properties

- Maximal diff. probability : 2^{-2}
- Maximal abs. linear bias : 2^{-2}
- $\deg(\mathcal{S}_4) = \deg(\mathcal{S}_4^{-1}) = 3$
- One fixed point : $\mathcal{S}_4(\texttt{0xF}) = \texttt{0xF}$
- Branch number : 2

SKINNY 8-bit Sbox



$\mathcal{S}_8$: 8-bit Sbox for SKINNY-128-*

- Generalize the $\mathcal{S}_4$ construction
- Implementation :
  8 NOR and 8 XOR
- Hardware cost : 24 GE

Properties

- Maximal diff. probability : 2^{-2}
- Maximal abs. linear bias : 2^{-2}
- $\deg(\mathcal{S}_8) = \deg(\mathcal{S}_8^{-1}) = 6$
- One fixed point :
  $\mathcal{S}_8(\texttt{0xFF}) = \texttt{0xFF}$
- Branch number : 2

Outline

Overview of SKINNY security

Claims

- Security against known classes of attacks
- Security in the related-key model
- No guarantees for known or chosen key
- No claim for related-cipher security
  (the constant does not encode the cipher parameters)

Attack vectors considered

- Differential/Linear cryptanalysis
- Integral attack
- Division property
- Meet-in-the-middle attack
- Impossible differential attack
- Invariant subspace attack
- Slide attack
- Algebraic attack

Comparing differential/linear bounds

- We adapt the number of rounds to get resistance (+ margin) :
  - SKINNY-64-64/128/192 has 32/36/40 rounds
  - SKINNY-128-128/256/384 has 40/48/56 rounds
- As a result, for all SKINNY variants :
  - **SK** security reached in 20 – 40% of the rounds
  - **TK2** security reached in 40 – 50% of the rounds

Comparison with other 64/128 and 128/128 ciphers

Cipher	Single Key (SK)	Related Key (RK)
SKINNY-64-128	8/36 = 22%	15/36 = 42%
SIMON-64-128	19/44 = 43%	no bound known
SKINNY-128-128	15/40 = 37%	19/40 = 47%
SIMON-128-128	41/72 = 57%	no bound known
AES-128	4/10 = 40%	6/10 = 60%
NOEKEON-128	12/16 = 75%	12/16 = 75%

Outline

Theoretical performances of SKINNY

Cipher	Rounds	#operations per bit without KS	#operations per bit with KS	Round-based area estimation
SKINNY-64-128	36	117	139.5	8.68
SIMON-64-128	44	88	**154**	8.68
PRESENT-64-128	31	147.2	**161.8**	12.43
PICCOLO-64-128	31	162.75	**162.75**	12.35
SKINNY-128-128	40	130	130	7.01
SIMON-128-128	72	136	**204**	7.34
NOEKEON-128-128	16	100	**200**	30.36
AES-128-128	10	202.5	**248.1**	59.12

Example of SKINNY-64-128 (more in the paper)

- 1R : (4 NOR + 4 XOR)/4 [SB] + (3 XOR)/4 [MC] + (32 XOR)/64 [ART]
- That is (per bit per round) : 1 NOR + 2.25 XOR
- #operations per bit (without KS) : $(1 + 2.25) \times 36 = 117$
- **Very low number of operations per plaintext bit. Challenge : do better.**

Round-based implementation results

	Area	Delay	Through-put @100KHz	Through-put @maxi-mum
	GE	ns	KBit/s	MBit/s
SKINNY-64-128	**1696**	1.87	177.78	951.11
SKINNY-128-128	2391	2.89	320.00	1107.20
SKINNY-128-256	3312	2.89	266.67	922.67
SIMON-64-128	**1751**	1.60	145.45	870
SIMON-128-128	2342	1.60	188.24	1145
SIMON-128-256	3419	1.60	177.78	1081
LED-64-128	3036	-	133.0	-
PRESENT-64-128	1884	-	200.00	-
PICCOLO-64-128	1773	-	193.94	-

Outline

Open problems in key-schedule security analysis

For security proofs :

- tighter bounds ?
- bounds for more rounds ?
- actual differences instead of truncated differences ?
- generic proof for any state size ?

For automated tools :

- more efficient algorithms (what about `AES-128` after 5 rounds ?)
- design tools to analyse other types of functions (e.g. ARX functions)
- automated tools for other attack types (MitM, division property, etc.)

Open problems in key-schedule constructions

For key schedule design :

- `LED` and `WHIRLPOOL` are not so efficient, others designs security is hard to prove
  can we design efficient and easily provable key schedules ?
- `STK` construction from TWEAKEY framework seems to be a good tradeoff, but we need more analysis (differentials, linear hulls ?)
- linear/non linear key schedule ?
- invertible/non invertible ?

Thank you !