FSE 2014 @ London, UK
3 March 2014

Improved All-Subkeys Recovery Attacks on FOX, KATAN and SHACAL-2 Block Ciphers

Takanori Isobe and Kyoji Shibutani
Sony Corporation

Background

- All-Subkeys Recovery Attacks
  - Proposed at SAC 2012 by Isobe and Shibutani
  - Recover all subkeys instead of user-provided key
  - Applied to CAST-128, KATAN32/48/64, SHACAL-2, Blowfish, FOX

- Function Reduction Technique
  - Proposed at ASIACRYPT 2013 by Isobe and Shinbutani
  - Advanced Technique for All-Subkeys Recovery Attack
  - Applied to *Feistel* Construction
    - Improved generic attacks on several Feistel constructions
    - 8-round attack on CAST-128 (best attack w.r.t. # attacked rounds)
    - Extremely-Low data attacks on 8/10/12-round Camellia-128/192/256

Our Contribution

- Improve "All-Subkeys Recovery Attack"
  - Extend "Function Reduction" to other constructions
    - Exploit structure-dependent properties of Lai-Massey and LFSR-type
  - Utilize "Repetitive All-Subkeys Recovery Attack"
    - Reduce data requirement in each inner ASR attacks
      - Variant of inner loop technique
  - Apply to FOX64/128, KATAN32/48/64, SHACAL-2
    - FOX64/128 : Lai-Massey construction
    - KATAN32/48/64 : Stream-type (LFSR) construction
    - SHACAL-2 : Source-heavy Generalized Feistel construction

Our Results

Best Attacks

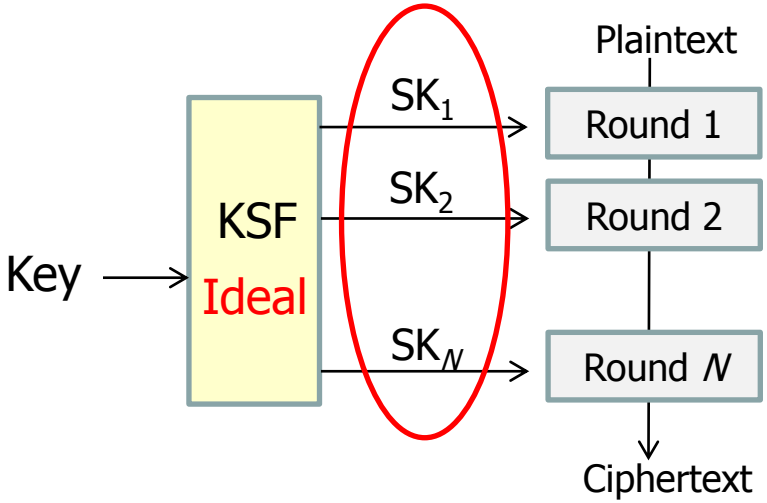
Target	# Attacked rounds	Attack Type	Time	Memory	Data	Paper
FOX64	5	Integral	$2^{109.4}$	Not given	2^{9}	[26]
	5	Impossible differential	2^{71}	Not given	2^{90}	[27]
	6	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	**15**	**Ours**
	7	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	$\mathbf{2^{30.9}}$	**Ours**
FOX128	5	Integral	$2^{205.6}$	Not given	$2^{116.3}$	[26]
	5	Impossible differential	2^{135}	Not given	28	[27]
	5	ASR	2^{228}	2^{228}	14	[16]
	6	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	**15**	**Ours**
	7	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	$\mathbf{2^{30.9}}$	**Ours**
KATAN32	110	ASR	2^{77}	$2^{75.1}$	138	[16]
	114	Differential	2^{77}	Not given	$2^{31.9}$	[2]
	119	**ASR**	$\mathbf{2^{79.1}}$	$\mathbf{2^{79.1}}$	**144**	**Ours**
KATAN48	100	ASR	2^{78}	2^{78}	128	[16]
	105	**ASR**	$\mathbf{2^{79.1}}$	$\mathbf{2^{79.1}}$	**144**	**Ours**
KATAN64	94	ASR	$2^{77.1}$	$2^{79.1}$	116	[16]
	99	**ASR**	$\mathbf{2^{79.1}}$	$\mathbf{2^{79.1}}$	**142**	**Ours**
SHACAL-2	41	ASR	2^{500}	2^{492}	244	[16]
	42	**ASR**	$\mathbf{2^{508}}$	$\mathbf{2^{508}}$	$\mathbf{2^{25}}$	**Ours**

Agenda

1. All-Subkeys Recovery Attacks
2. Function reduction technique
3. Improved Attacks on FOX-64/128
4. Improved Attacks on KATAN-32/48/64
5. Improved Attacks on SHACAL-2
6. Conclusion

All-Subkeys Recovery Attack

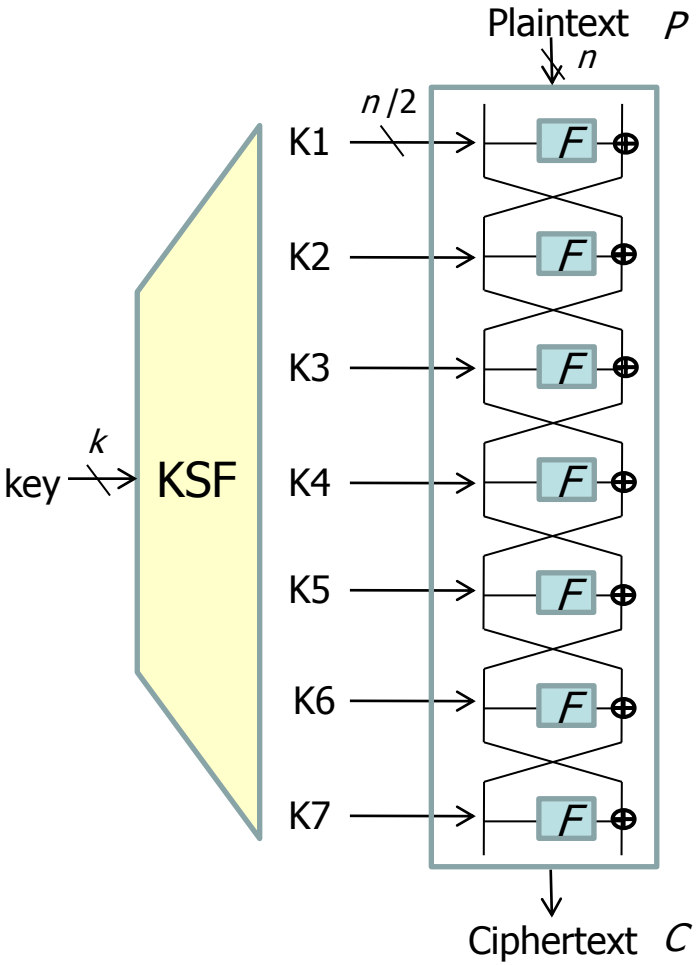
- Find all subkeys instead of user-provided key
  - "Finding all subkeys" ≒ "Finding user-provided key"



- Advantage
  - Analyzing KSF is not necessary
    - Easy to apply to complex KSF!
  - Focus only on data processing part
    - Work on any KSF (even ideal KSF) ⇨ **generic attack!**

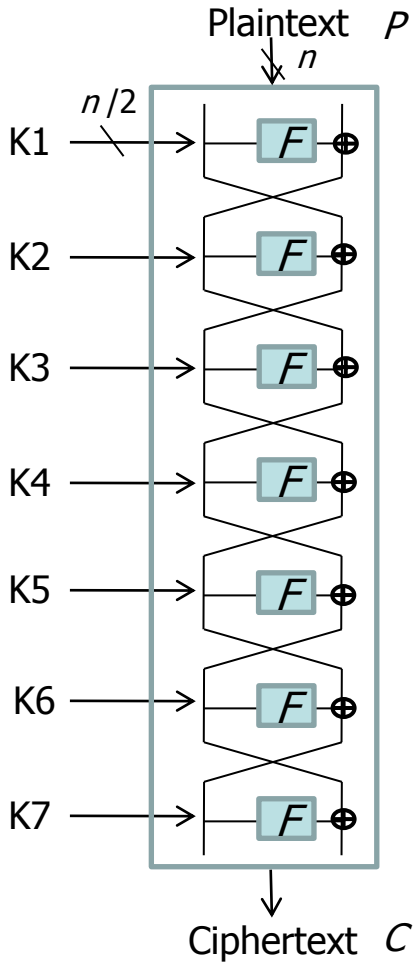
How to Recover All Subkeys

- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block



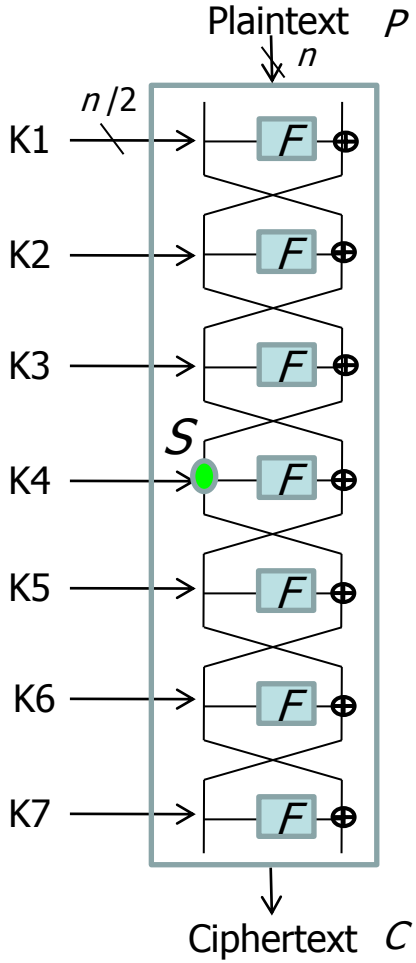
How to Recover All Subkeys

- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block



How to Recover All Subkeys

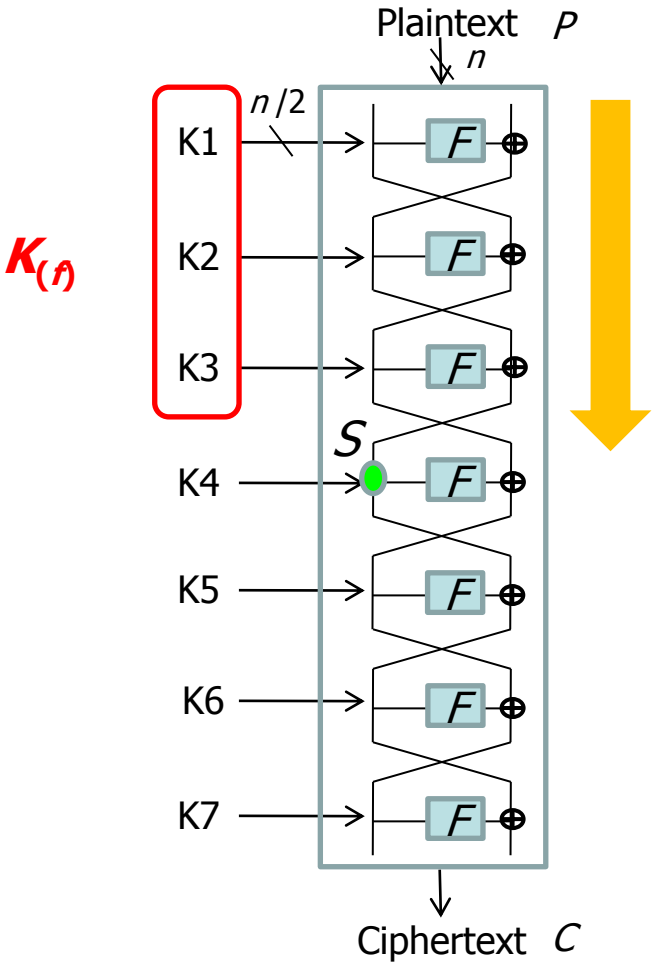
- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block



1. Choose s -bit matching state S

How to Recover All Subkeys

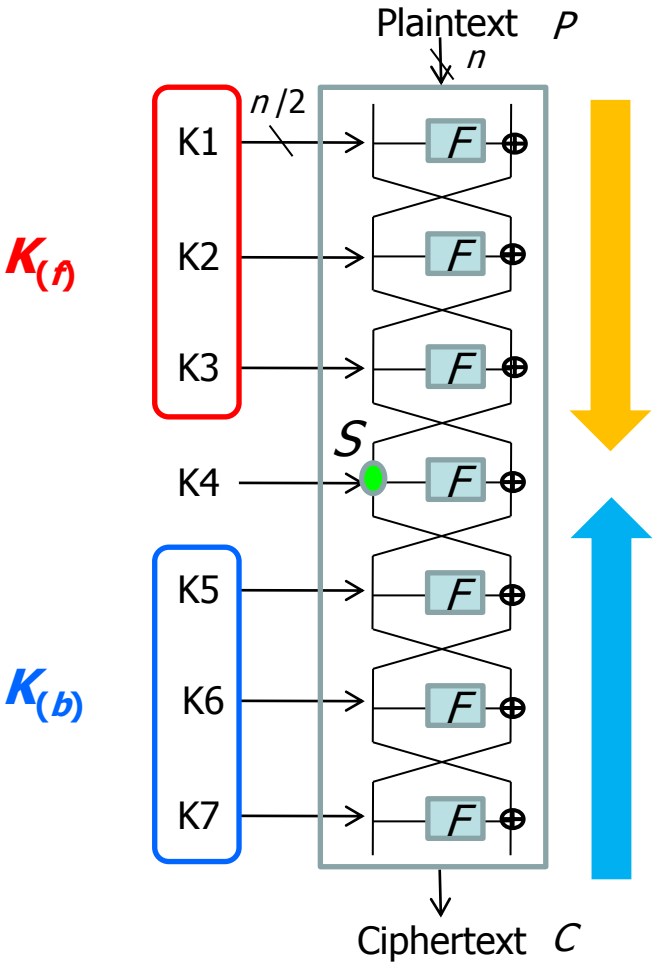
- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block



1. Choose s -bit matching state S
2. Compute $S = F_{(f)}(K_{(f)}, P)$ with all $K_{(f)}$ and Make Table of (s, $K_{(f)}$) pairs

How to Recover All Subkeys

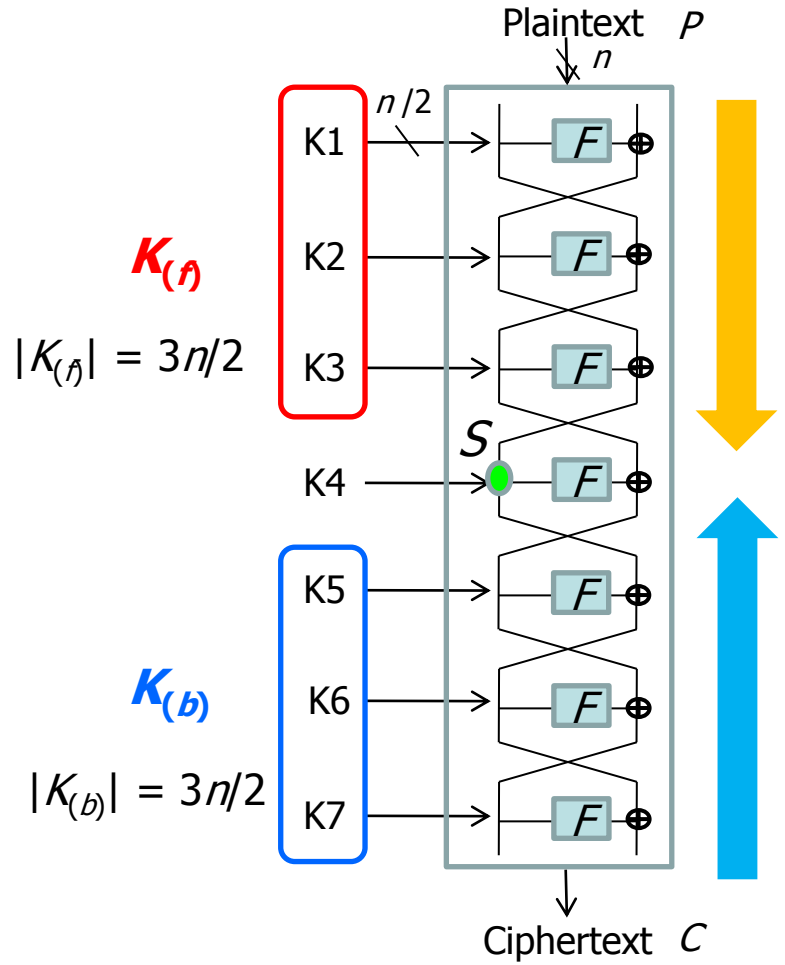
- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block



1. Choose s -bit matching state S
2. Compute $S = F_{(f)}(K_{(f)}, P)$ with all $K_{(f)}$ and Make Table of (s, $K_{(f)}$) pairs
3. Compute $S' = F_{(f)}(K_{(b)}, C)$ with all $K_{(b)}$

How to Recover All Subkeys

- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block

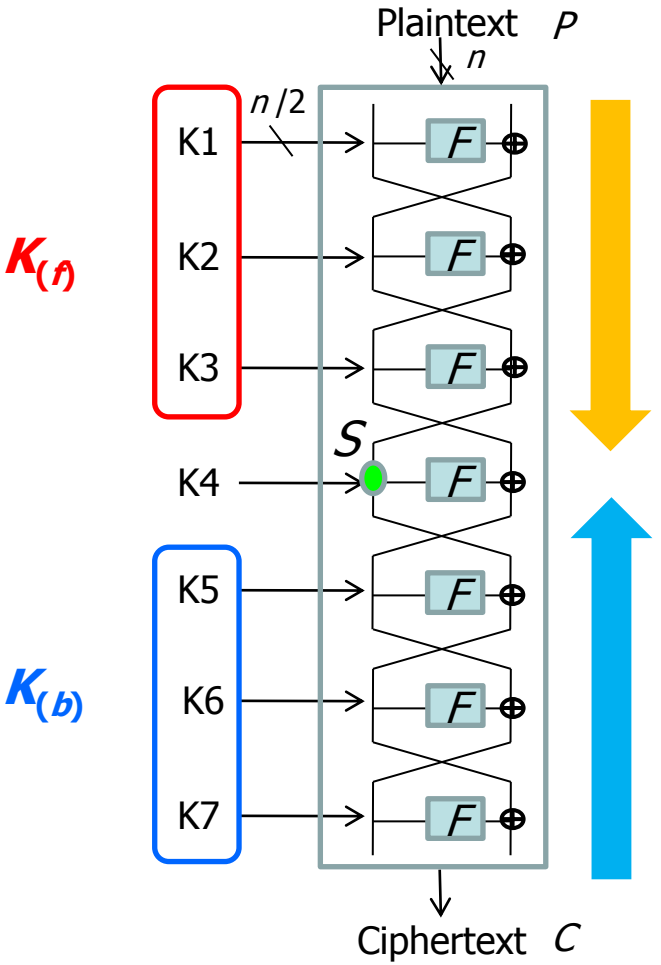


1. Choose s -bit matching state S
2. Compute $S = F_{(f)}(K_{(f)}, P)$ with all $K_{(f)}$ and Make Table of (s, $K_{(f)}$) pairs
3. Compute $S' = F_{(f)}(K_{(b)}, C)$ with all $K_{(b)}$
4. If $S = S'$, regard it as key candidate

surviving key candidates
: $2^{6n/2 - n/2} = 2^{5n/2}$

How to Recover All Subkeys

- Based on Meet-in-the-Middle Approach
  - Example: 7-round Feistel Cipher w/ k (=2n)-bit key and n-bit block



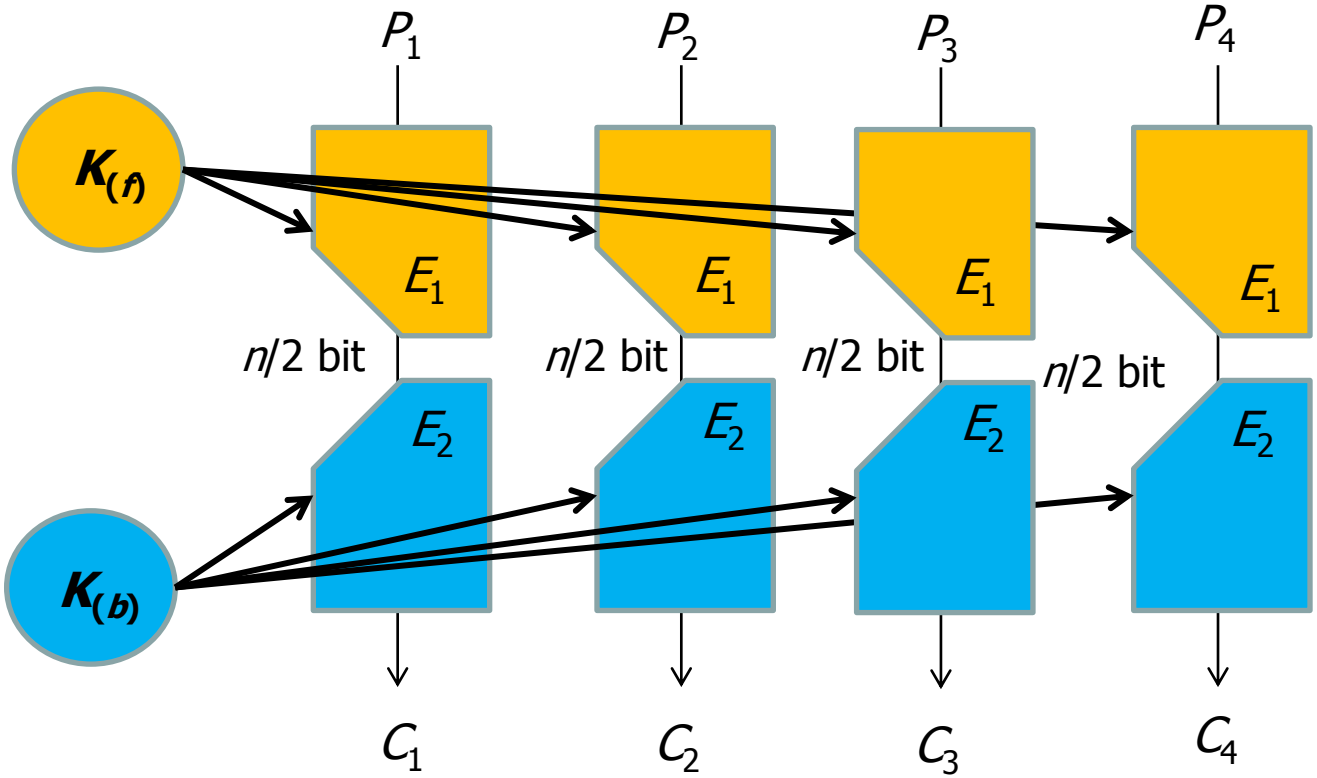
1. Choose s -bit matching state S
2. Compute $S = F_{(f)}(K_{(f)}, P)$ with all $K_{(f)}$ and Make Table of (s, $K_{(f)}$) pairs
3. Compute $S' = F_{(f)}(K_{(b)}, C)$ with all $K_{(b)}$
4. If $S = S'$, regard it as key candidate

surviving key candidates
: $2^{6n/2 - n/2} = 2^{5n/2}$

For successful attack
we need to reduce key space

Parallel MitM attack

- Given N (4) plaintext/ciphertexts

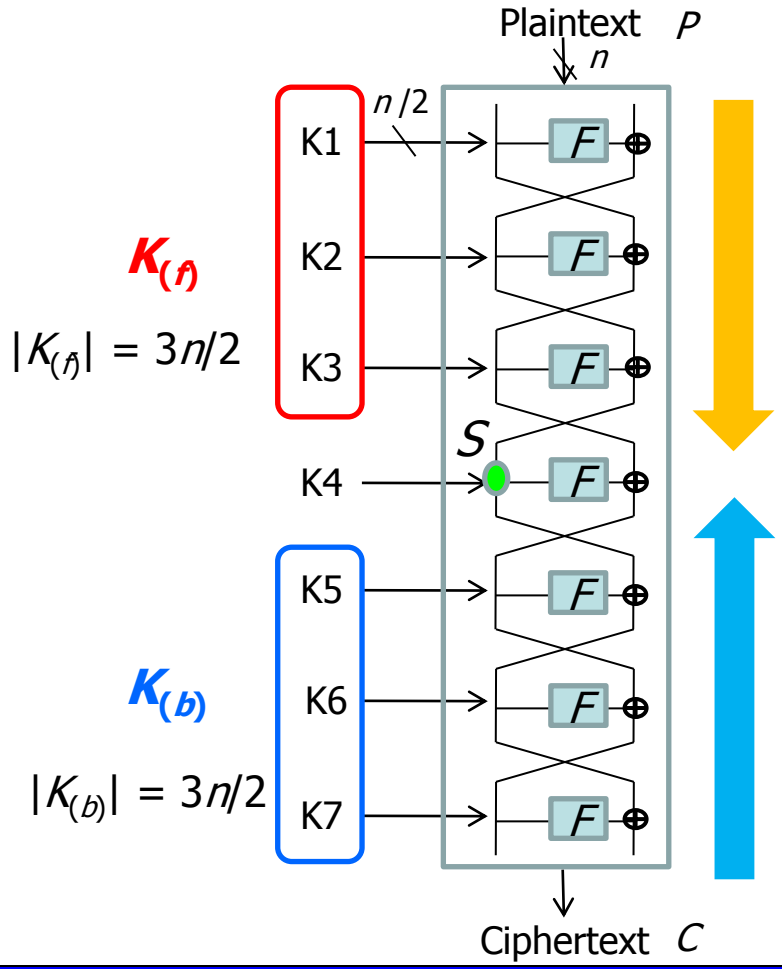


Filter out wrong keys by using N (4) matching state

surviving key candidates : $2^{6n/2 - 4 \cdot n/2} = 2^n (=2^{k/2})$

Evaluation

- Time complexity for filtering wrong keys



-Time complexity :

$$\max\left(2^{|K_{(f)}|}, 2^{|K_{(b)}|}\right) \times N$$

$$= \max\left(2^{3n/2}, 2^{3n/2}\right) \times 4$$

$$= 2^{3n/2+2} = 2^{3n/4+2} < 2^{k}$$

Point for successful attack :

Reduce involved keys $K_{(f)}$ and $K_{(b)}$ in forward and backward computation

Agenda

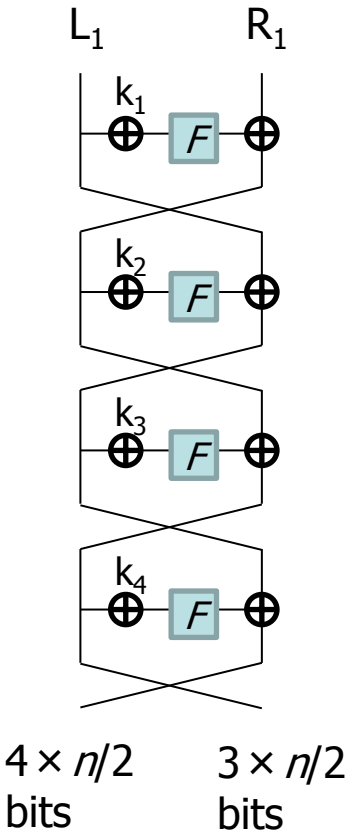
1. All-Subkeys Recovery Attacks
2. Function reduction technique
3. Improved Attacks on FOX-64/128
4. Improved Attacks on KATAN-32/48/64
5. Improved Attacks on SHACAL-2
6. Conclusion

Function Reduction Technique [IS 2013]

- Technique to reduce bits used in $K_{(f)}$ and $K_{(b)}$
  - Using degree of freedom of plaintext/ciphertext

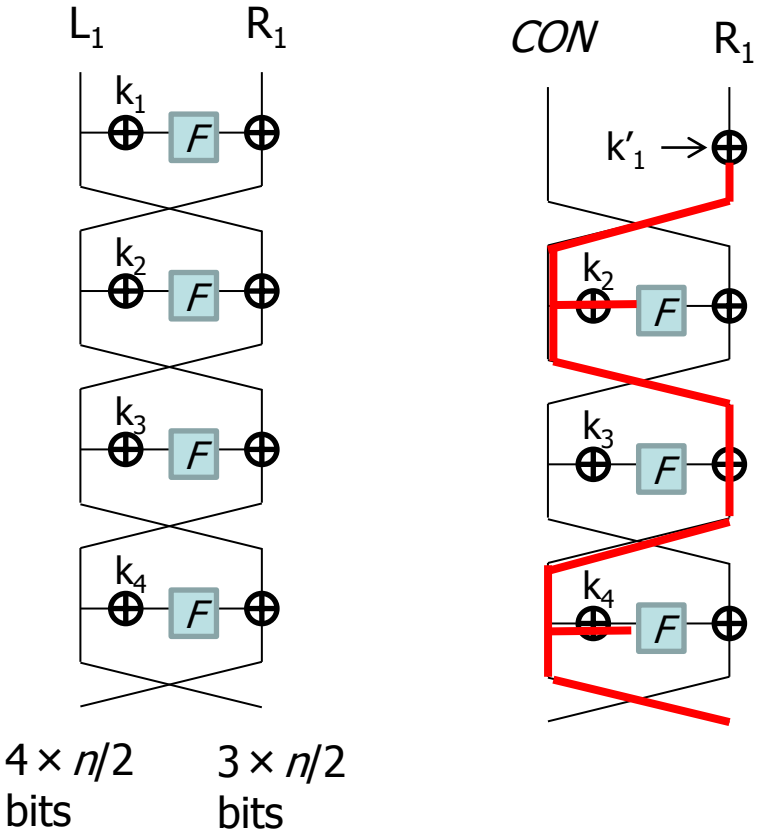
Function Reduction Technique [IS 2013]

- Technique to reduce bits used in $K_{(f)}$ and $K_{(b)}$
  - Using degree of freedom of plaintext/ciphertext



Function Reduction Technique [IS 2013]

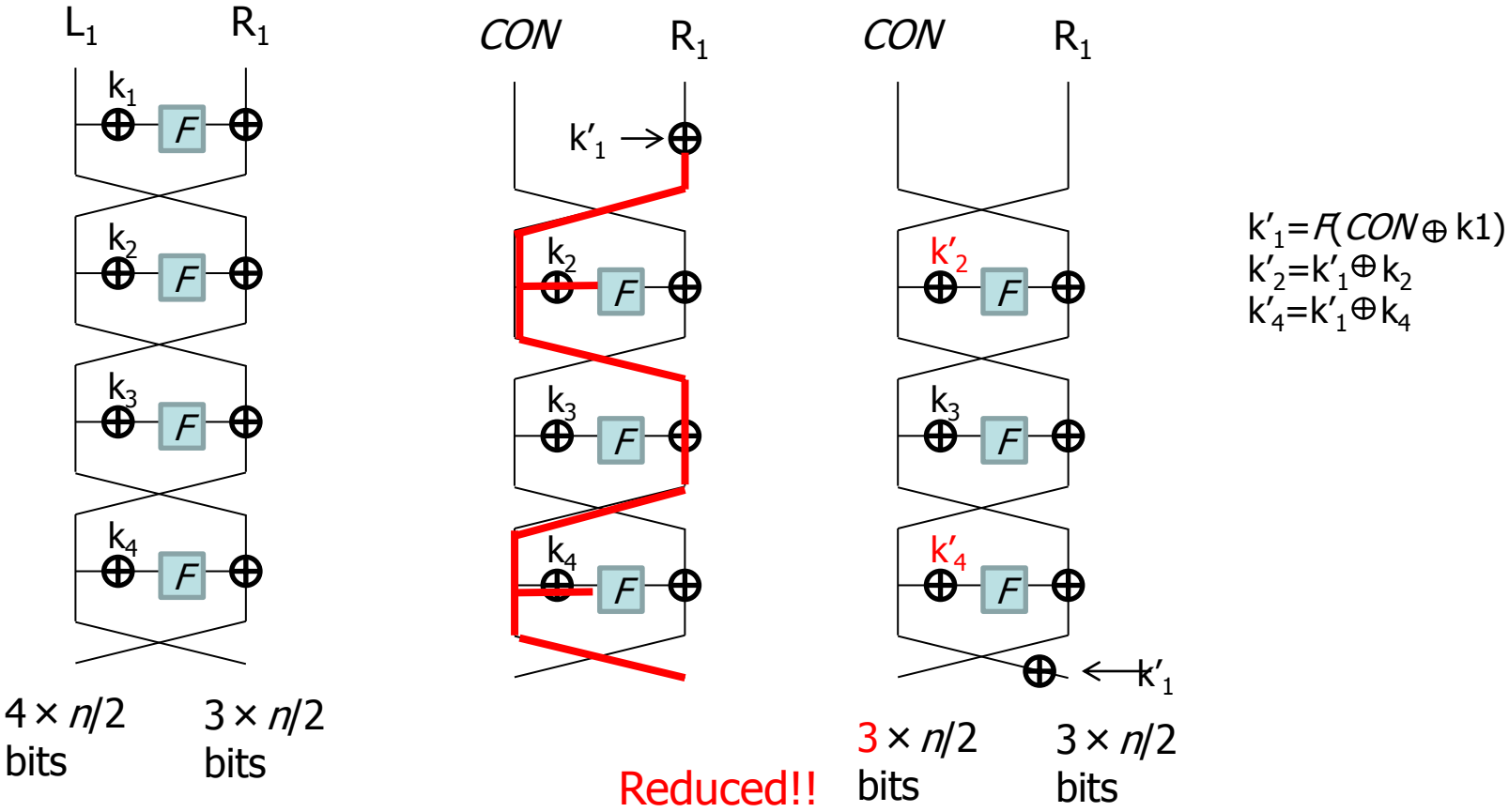
- Technique to reduce bits used in $K_{(f)}$ and $K_{(b)}$
  - Using degree of freedom of plaintext/ciphertext



$k'_1 = F(CON \oplus k1)$

Function Reduction Technique [IS 2013]

- Technique to reduce bits used in $K_{(f)}$ and $K_{(b)}$
  - Using degree of freedom of plaintext/ciphertext



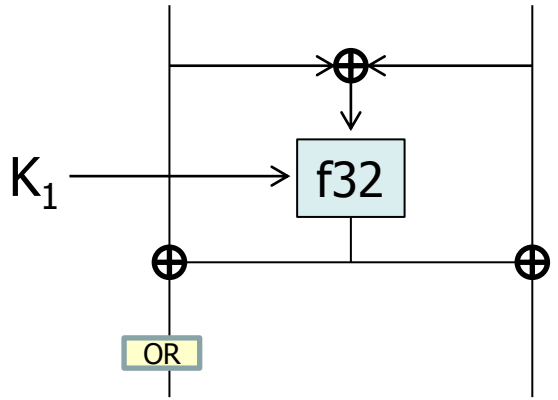
Agenda

1. All-Subkeys Recovery Attacks
2. Function reduction technique
3. Improved Attacks on FOX-64/128
4. Improved Attacks on KATAN-32/48/64
5. Improved Attacks on SHACAL-2
6. Conclusion

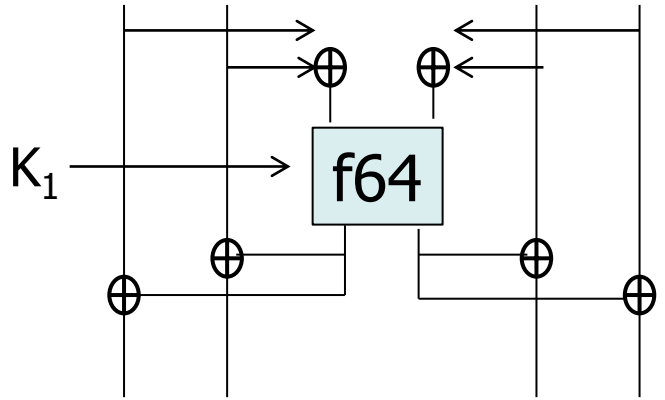
FOX

- Proposed by Junod and Vaudenay @ SAC 2004
- Also known as IDEA NXT (successor of IDEA)
- Based on Lai-Massey scheme
- *Strong* key scheduling function
- Two Variants
  - FOX 64 : 64-bit block and 128-bit key
  - FOX 128 : 128-bit block and 256-bit key

FOX 64

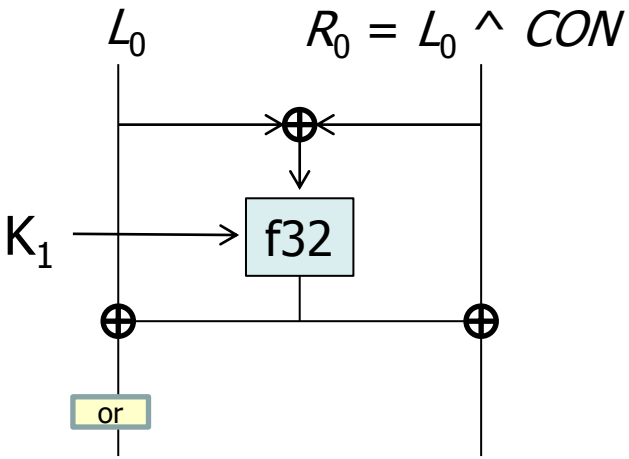


FOX 128



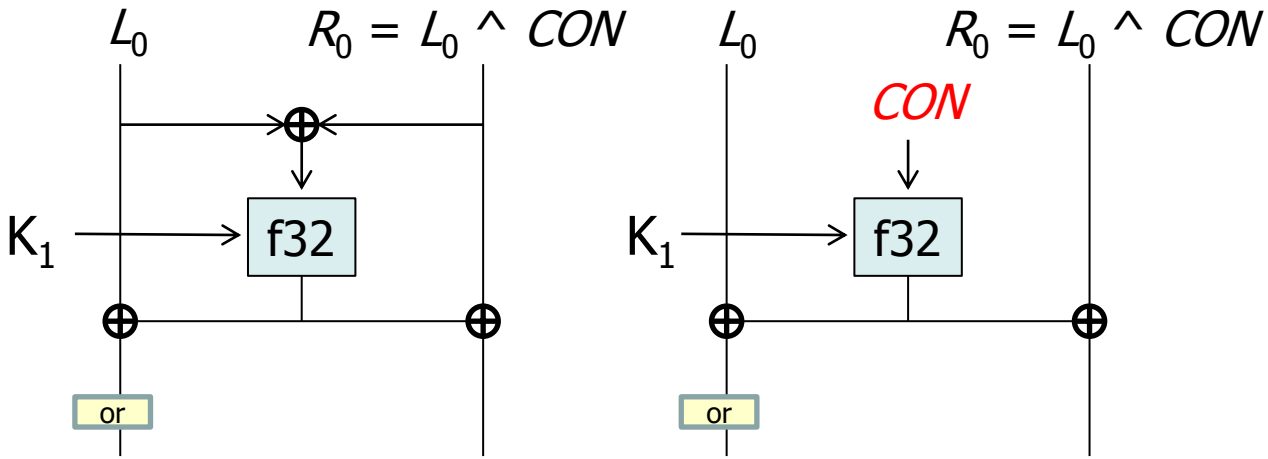
Function Reduction of FOX 64

- Choose plaintext s.t., $R_0 = L_0 \wedge CON$
  - Input to f32 is fixed to constant *CON*



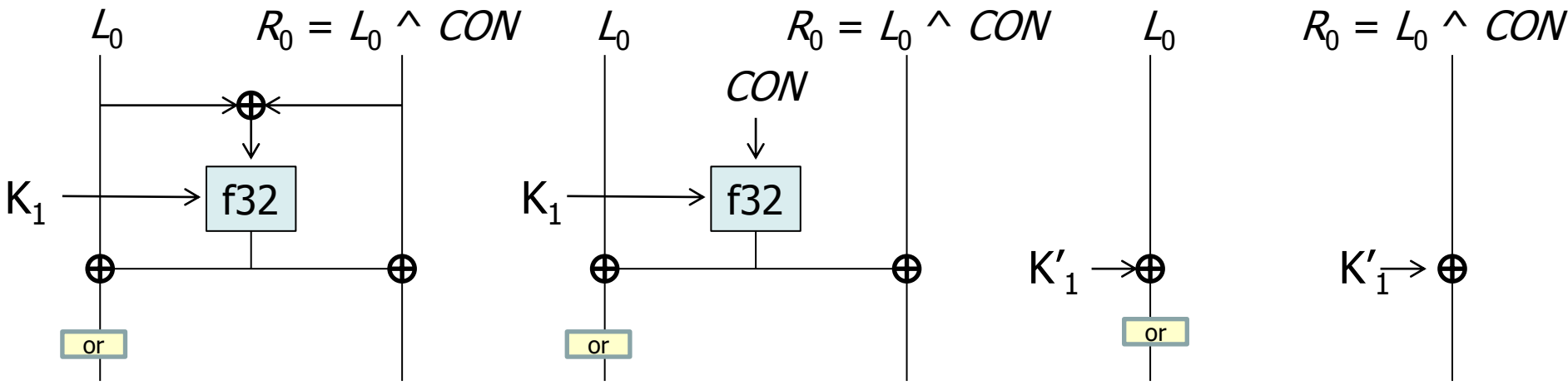
Function Reduction of FOX 64

- Choose plaintext s.t., $R_0 = L_0 \wedge CON$
  - Input to f32 is fixed to constant *CON*



Function Reduction of FOX 64

- Choose plaintext s.t., $R_0 = L_0 \wedge CON$
  - Input to f32 is fixed to constant *CON*

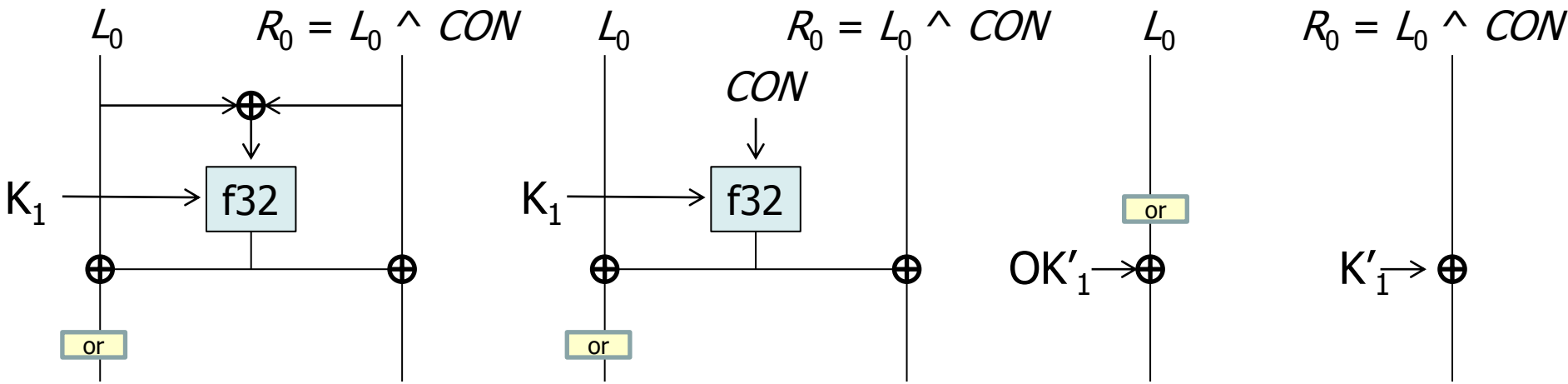


$K'_1 = f32(CON, K_1)$

depend only on key bits

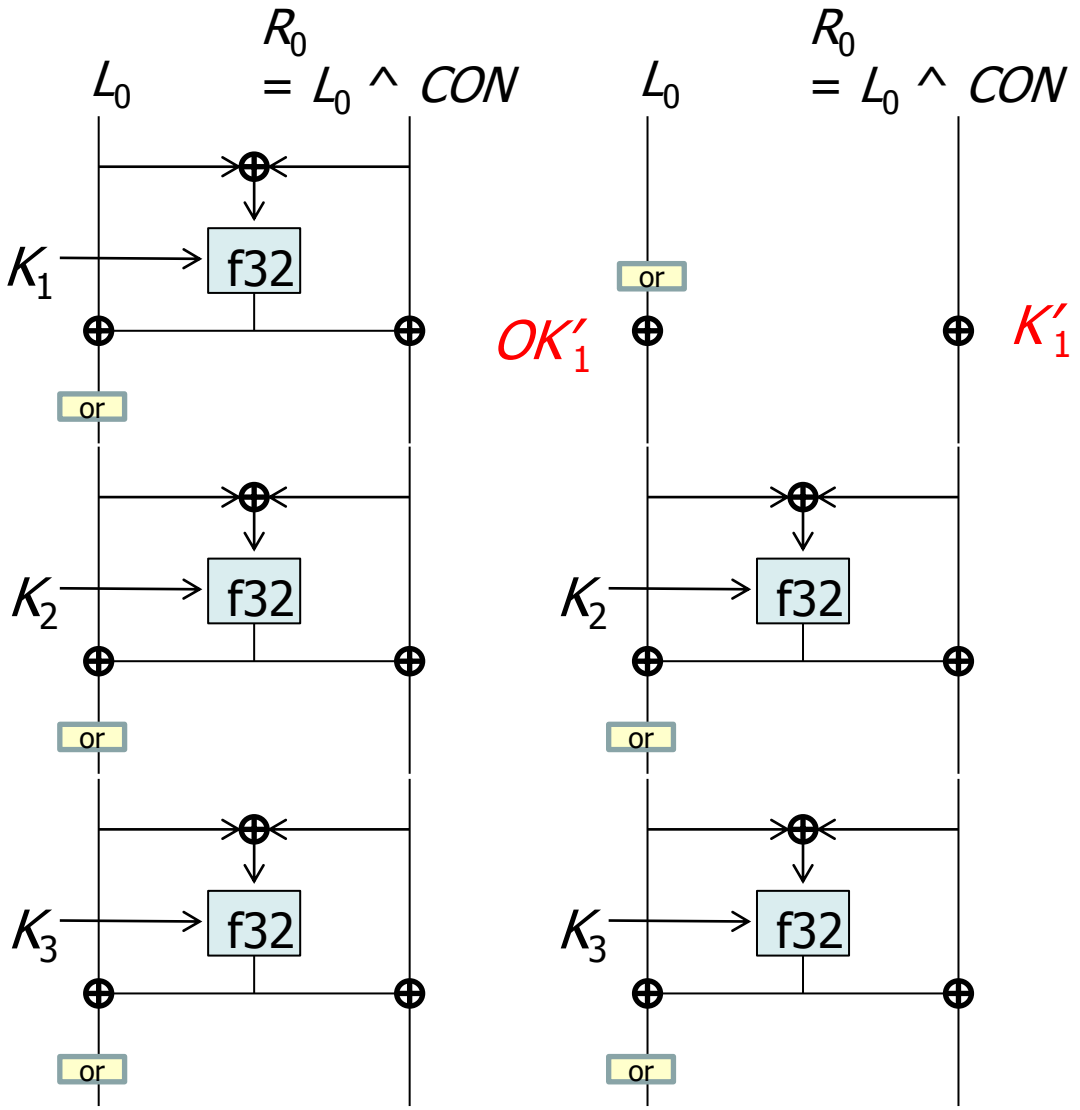
Function Reduction of FOX 64

- Choose plaintext s.t., $R_0 = L_0 \wedge CON$
  - Input to f32 is fixed to constant *CON*



$K'_1 = f32(CON, K_1)$
$OK'_1 = or\ (f32(CON, K_1))$

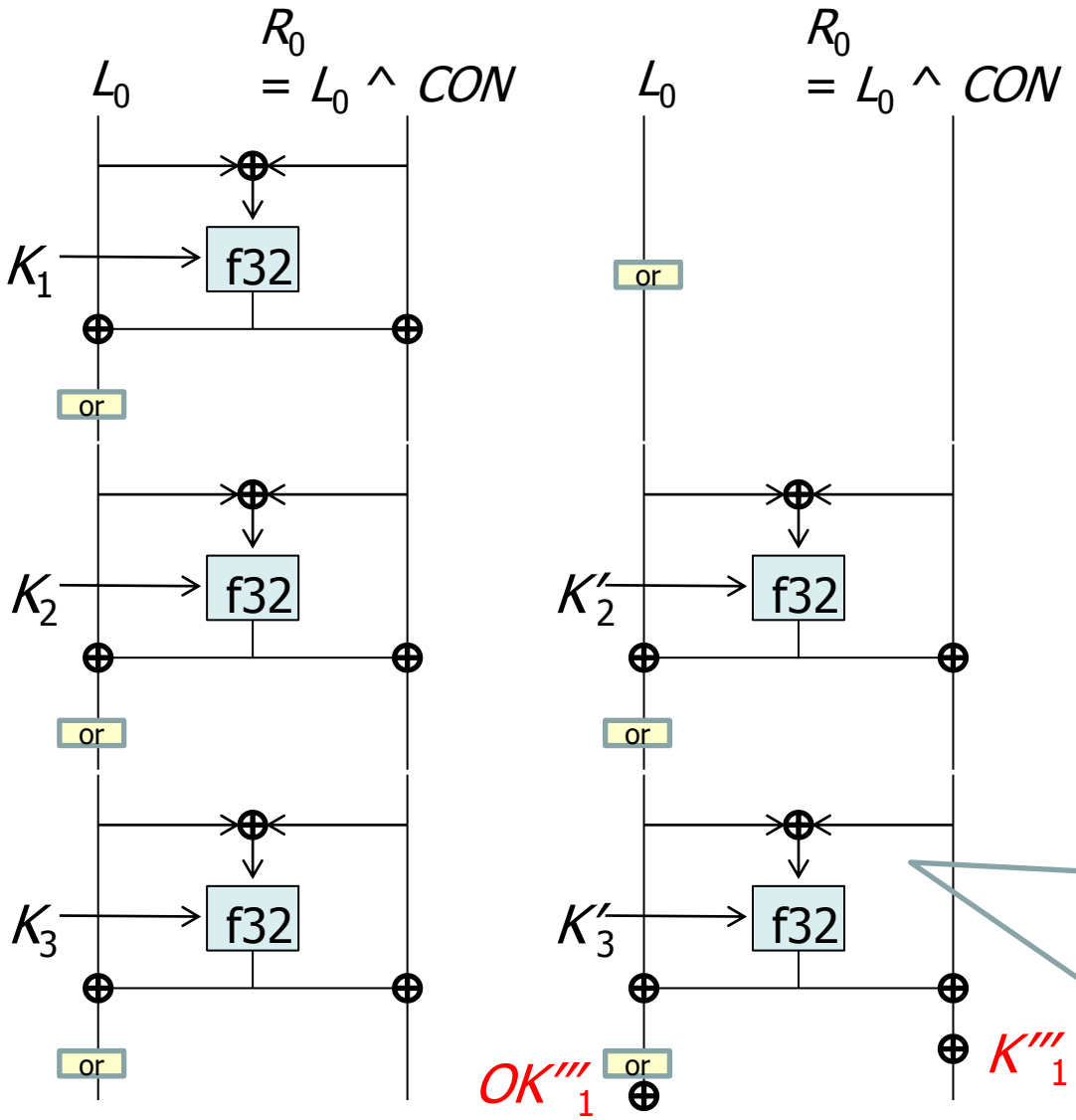
3-round Function Reduction of FOX64



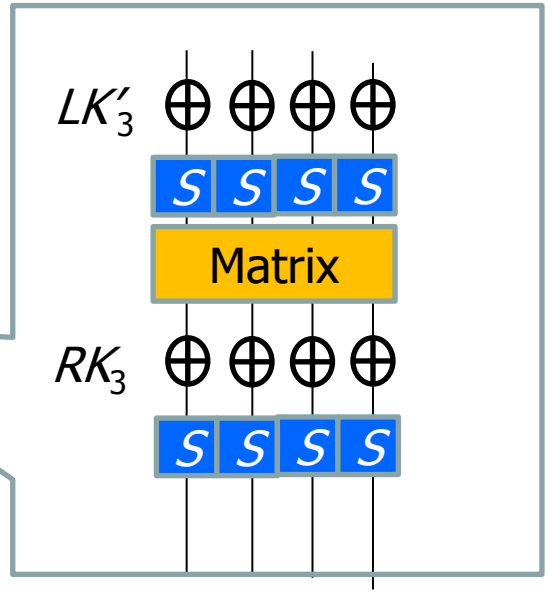
$K'_1 = \text{f32}(CON, K_1)$

$OK'_1 = \text{or}\,(\text{f32}(CON, K_1))$

3-round Function Reduction of FOX64



$K'_1 = \text{f32}(CON, K_1)$
$OK'_1 = \text{or}(\text{f32}(CON, K_1))$
$LK'_2 = LK_2 \wedge OK'_1 \wedge K'_1$
$K''_1 = K'_1 \wedge LK_2$
$OK''_1 = \text{or}(OK'_1 \wedge LK_2)$
$K'_2 = LK_2 \,||\, RK_2$
$LK'_3 = LK_3 \wedge OK''_1 \wedge K''_1$
$K'''_1 = K''_1 \wedge LK_3$
$OK'''_1 = \text{or}(OK''_1 \wedge LK_3)$
$K'_3 = LK_3 \,||\, RK_3$

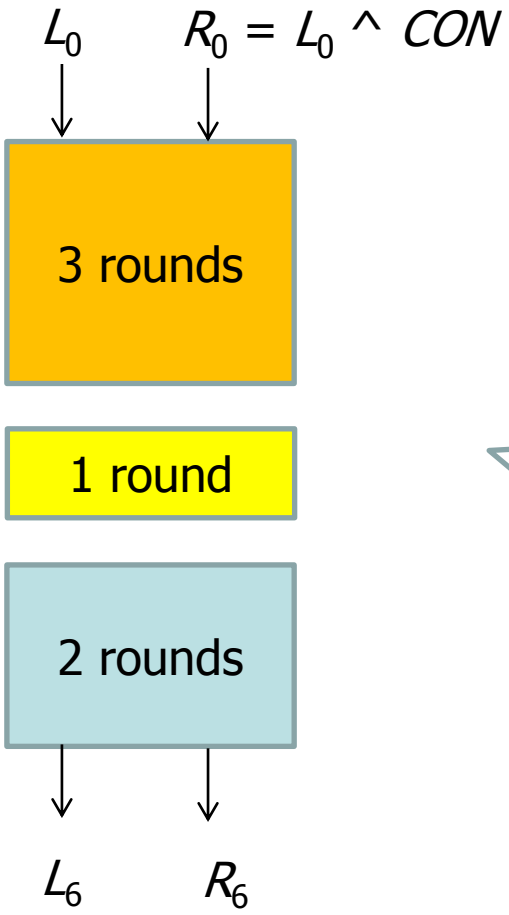


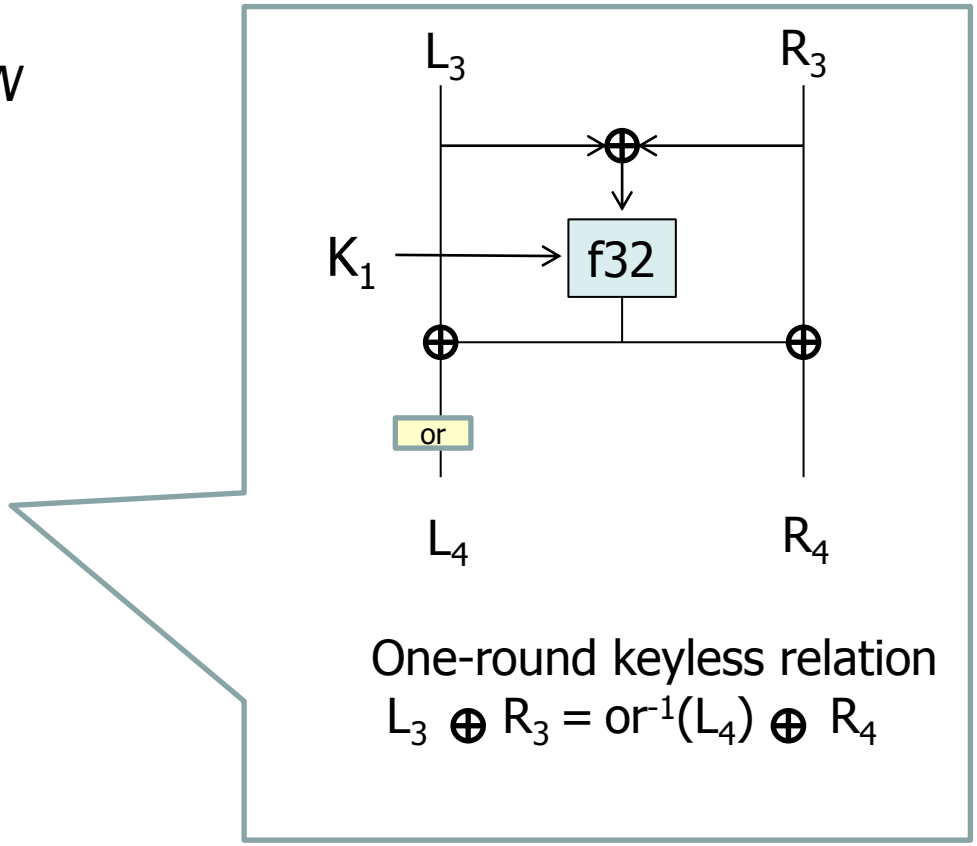
6-round Attack on FOX 64

- 3-round Function Reduction in forward direction

6-round Attack on FOX 64

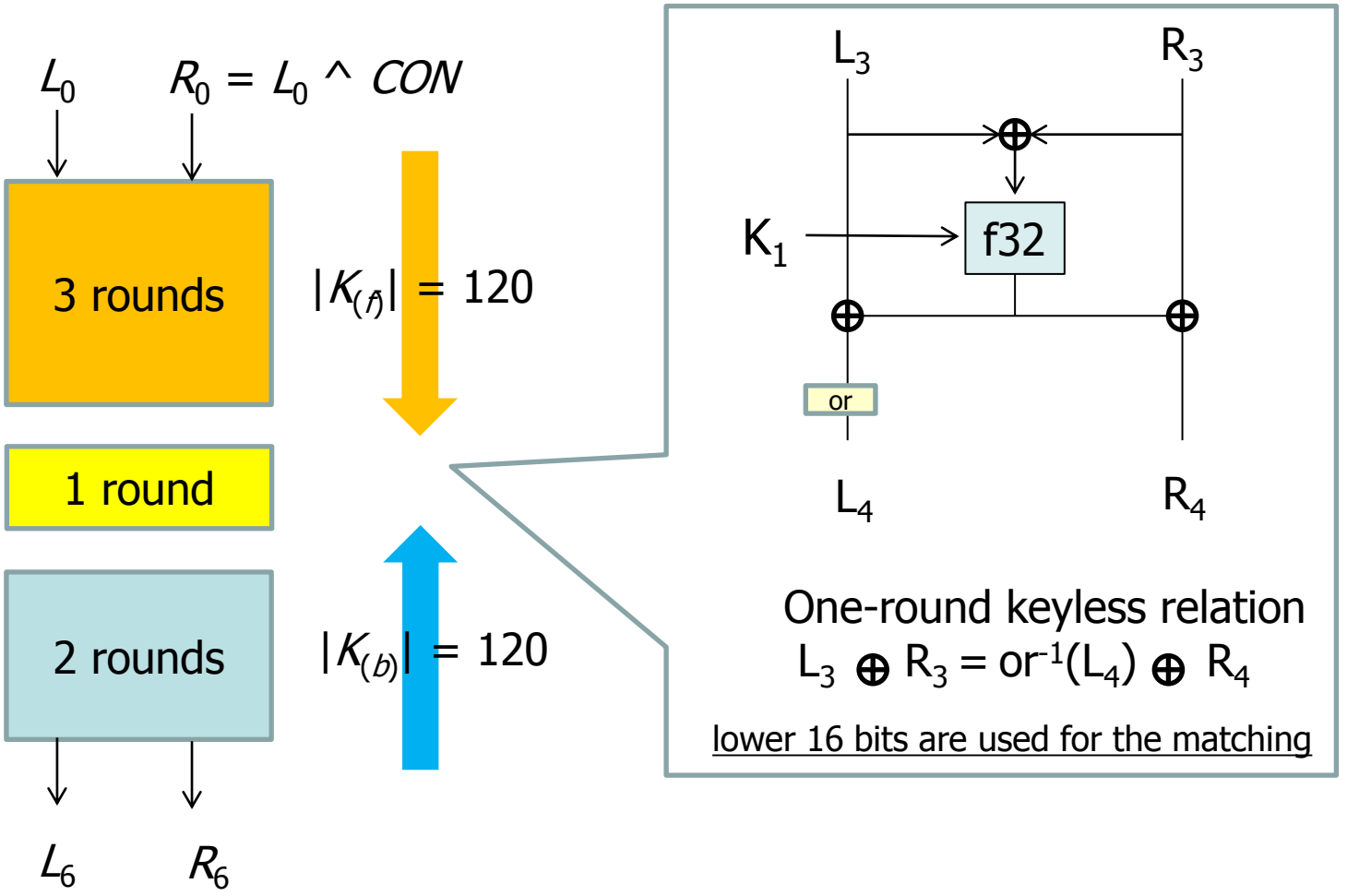
- 3-round Function Reduction in forward direction





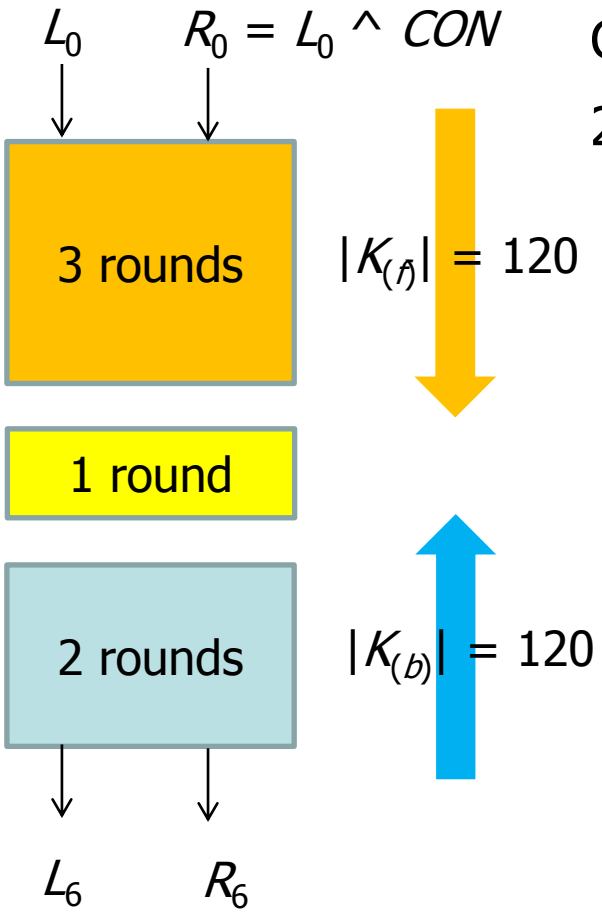
6-round Attack on FOX 64

- 3-round Function Reduction in forward direction



6-round Attack on FOX 64

- 3-round Function Reduction in forward direction



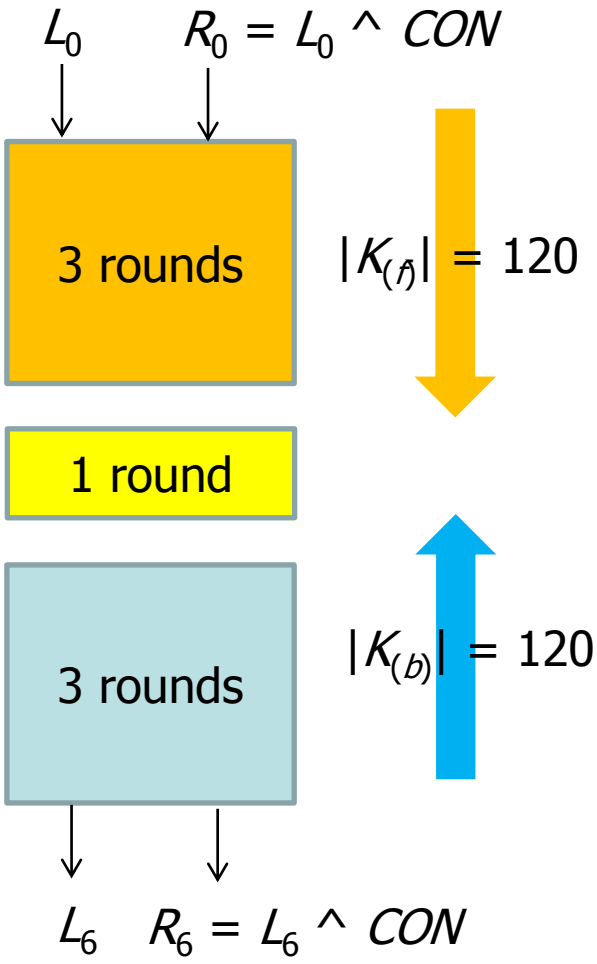
Given 15 data (15 parallel ASR attacks),

2^{240} candidates are reduced to $2^0(2^{240-16\cdot15}) = 1$

Time : $2^{120} \times 15 = 2^{124}$
Data : 15
Memory : $2^{120} \times 14 = 2^{124}$

7-round Attack on FOX64

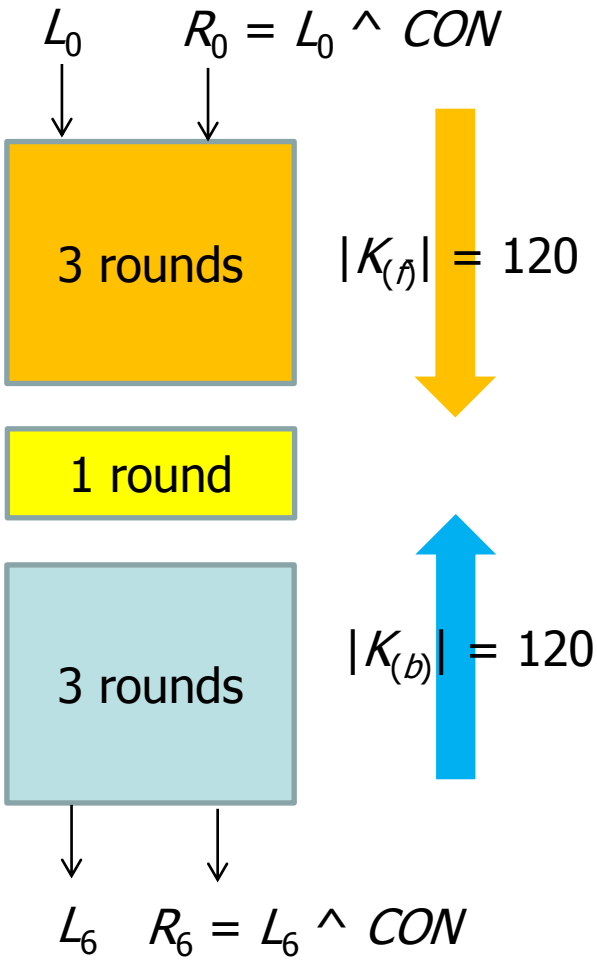
- 3-round Function Reduction in both directions



- Require 15 plaintext/ciphertext pairs s.t.
  - Plaintext = $L_0 || L_0 \wedge CON$ (32 bit condition)
  - Ciphertext = $L_6 || L_6 \wedge CON$ (32 bit condition)
- To find these data, it requires more than 2^{32} plaintexts satisfying $L_0 || L_0 \wedge CON$ (32 bit condition).
- However, degree of freedom of plaintext is only 32 bit L_0

7-round Attack on FOX64

- 3-round Function Reduction in both directions

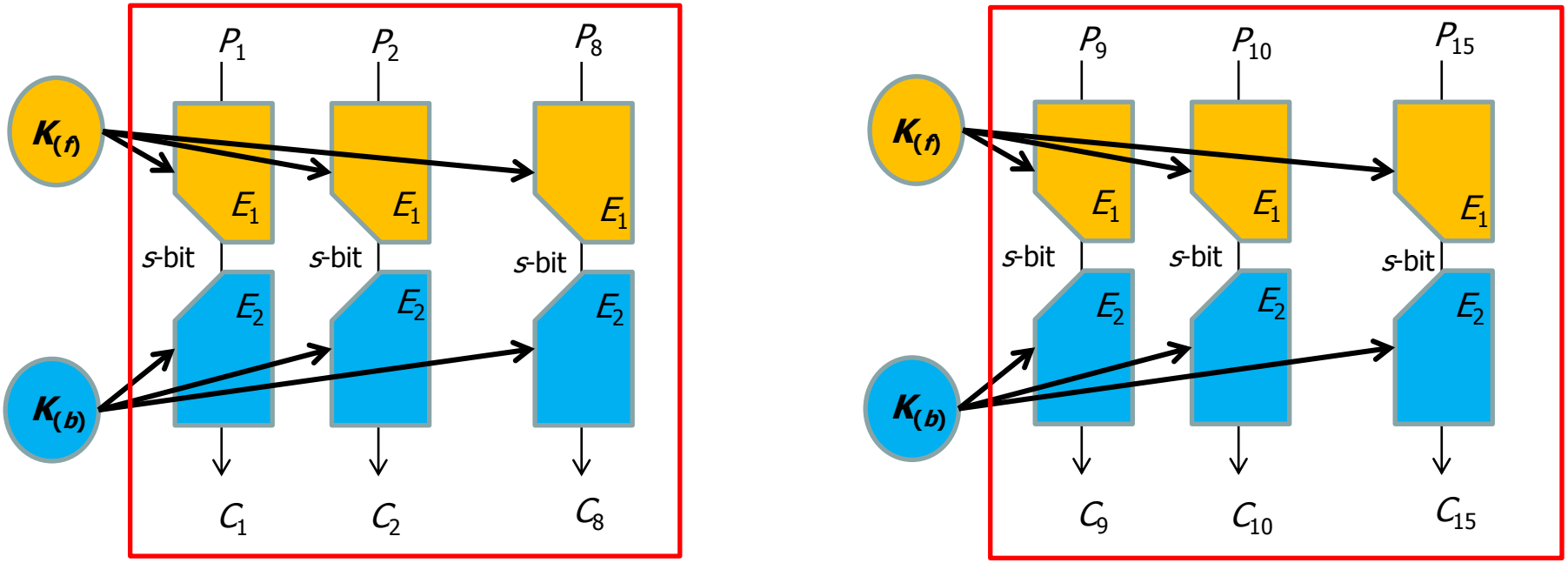


- Require 15 plaintext/ciphertext pairs s.t.
  - Plaintext = $L_0 || L_0 \wedge CON$ (32 bit condition)
  - Ciphertext = $L_6 || L_6 \wedge CON$ (32 bit condition)
- To find these data, it requires more than 2^{32} plaintexts satisfying $L_0 || L_0 \wedge CON$ (32 bit condition).
- However, degree of freedom of plaintext is only 32 bit L_0

Repetitive ASR approach

Repetitive All-Subkeys Recovery Attack

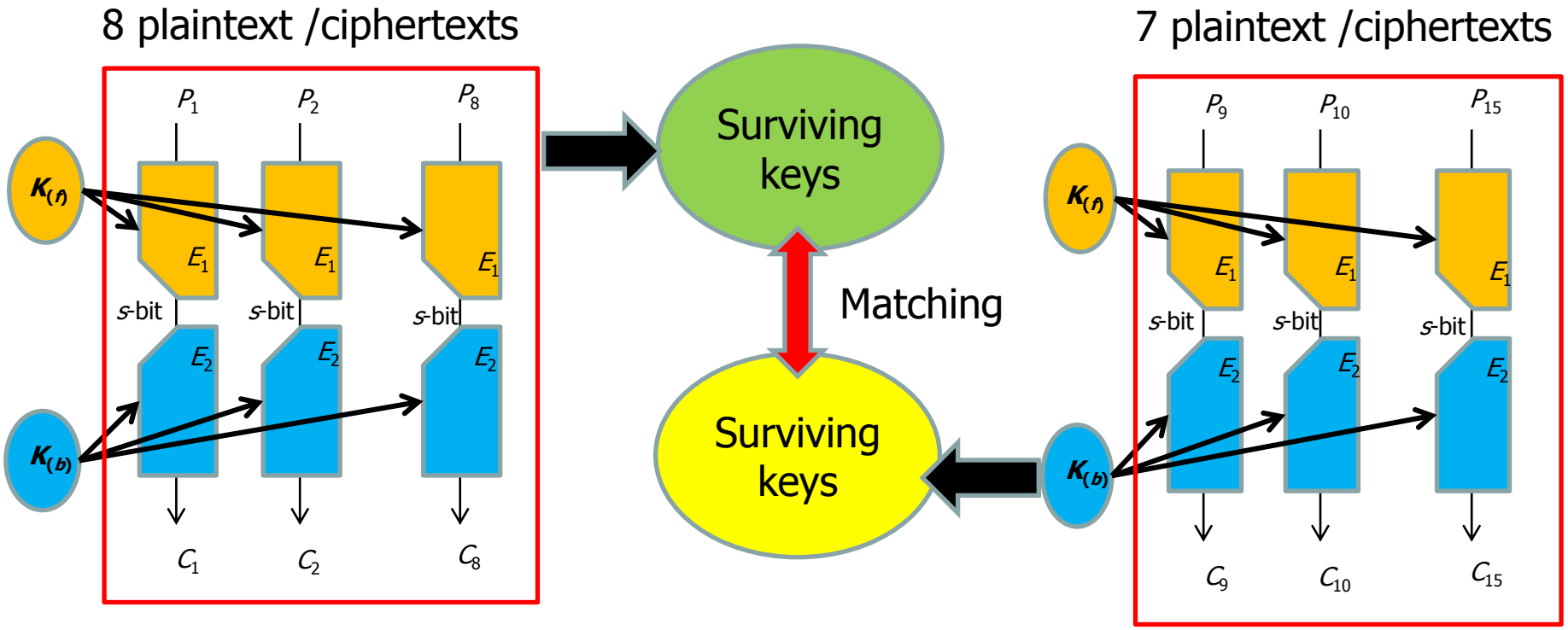
- Divide Parallel MitM into M parts
  - Enable mounting each ASR with less data
  - Example : $M = 2$ (8 and 7)



8 desired data can be obtained from 2^{31} ciphertexts

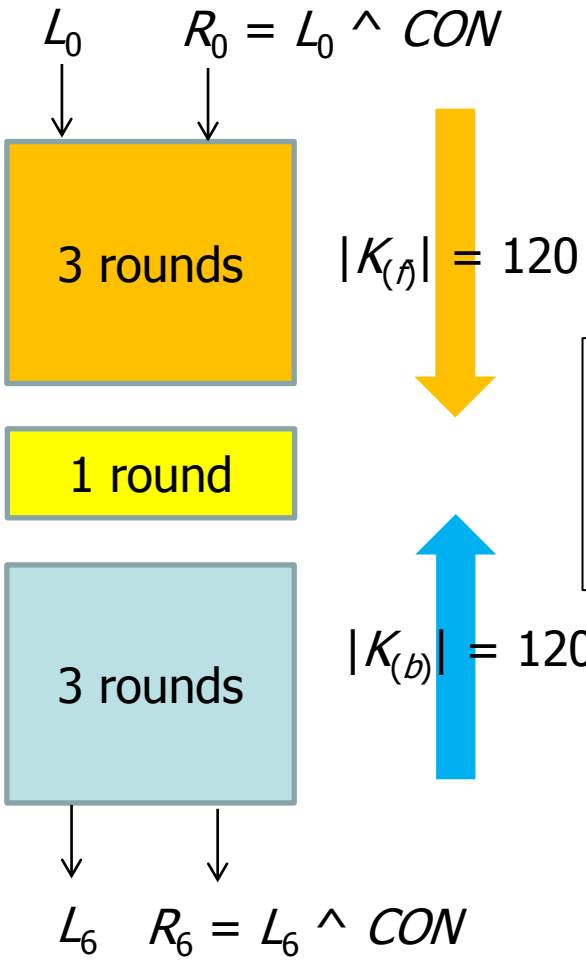
Repetitive All-Subkeys Recovery Attack

- Divide Parallel MitM into M parts
  - Enable mounting each ASR with less data
  - Example : $M = 2$ (8 and 7)



7-round Attack on FOX64

- 3-round Function Reduction in both directions



Time : $2^{120} \times 8 + 2^{120} \times 7 = 2^{124}$
Data : 2^{31}
Memory : $2^{120} \times 14 = 2^{123}$

Results

- Update best single-key attacks on FOX64 and FOX128 w.r.t. number of attacked rounds

Best Attacks

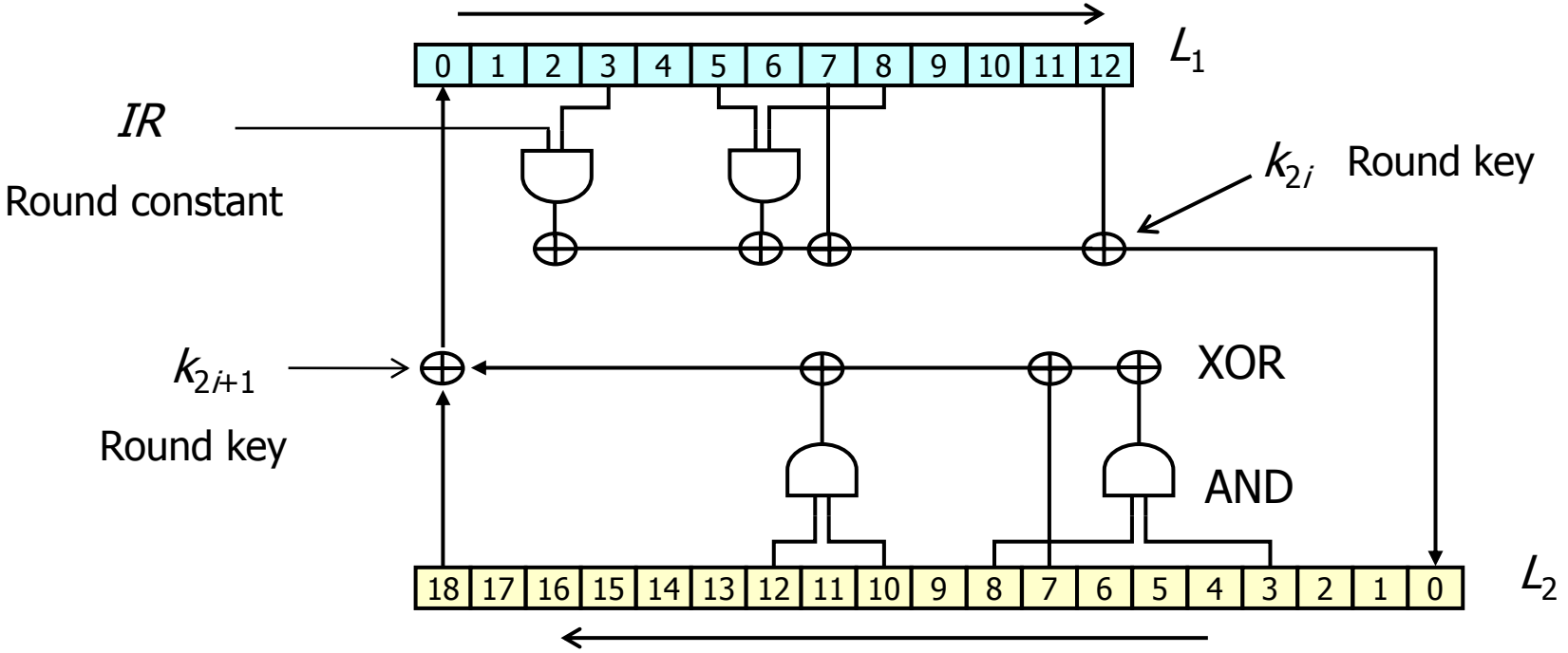
Target	#Attack round	Attack Type	Time	Memory	Data	Paper
FOX64	5	Integral	$2^{109.4}$	Not given	2^{9}	[26]
	5	Impossible differential	2^{71}	Not given	2^{90}	[27]
	6	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	**15**	**Ours**
	7	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	$\mathbf{2^{30.9}}$	**Ours**
FOX128	5	Integral	$2^{205.6}$	Not given	$2^{116.3}$	[26]
	5	Impossible differential	2^{135}	Not given	28	[27]
	5	ASR	2^{228}	2^{228}	14	[16]
	6	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	**15**	**Ours**
	7	**ASR**	$\mathbf{2^{124}}$	$\mathbf{2^{124}}$	$\mathbf{2^{30.9}}$	**Ours**

Agenda

1. All-Subkeys Recovery Attacks
2. Function reduction technique
3. Improved Attacks on FOX-64/128
4. Improved Attacks on KATAN-32/48/64
5. Improved Attacks on SHACAL-2
6. Conclusion

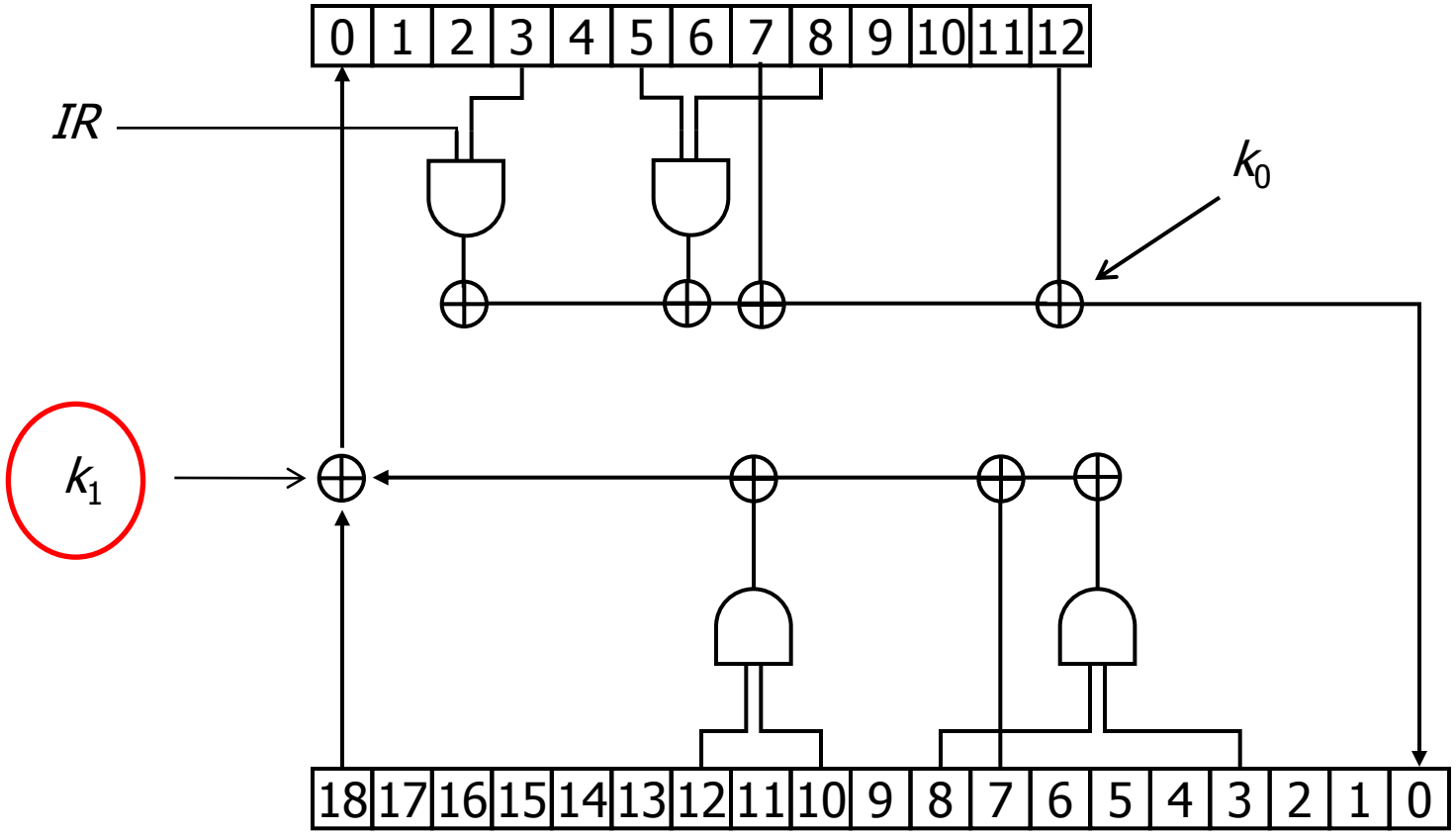
KATAN Family

- Ultra lightweight block cipher (CHES 2010)
- block size : 32/48/64 bits, key size : 80 bits
- Based on Stream cipher Trivium
  - 254-round LFSR-type construction



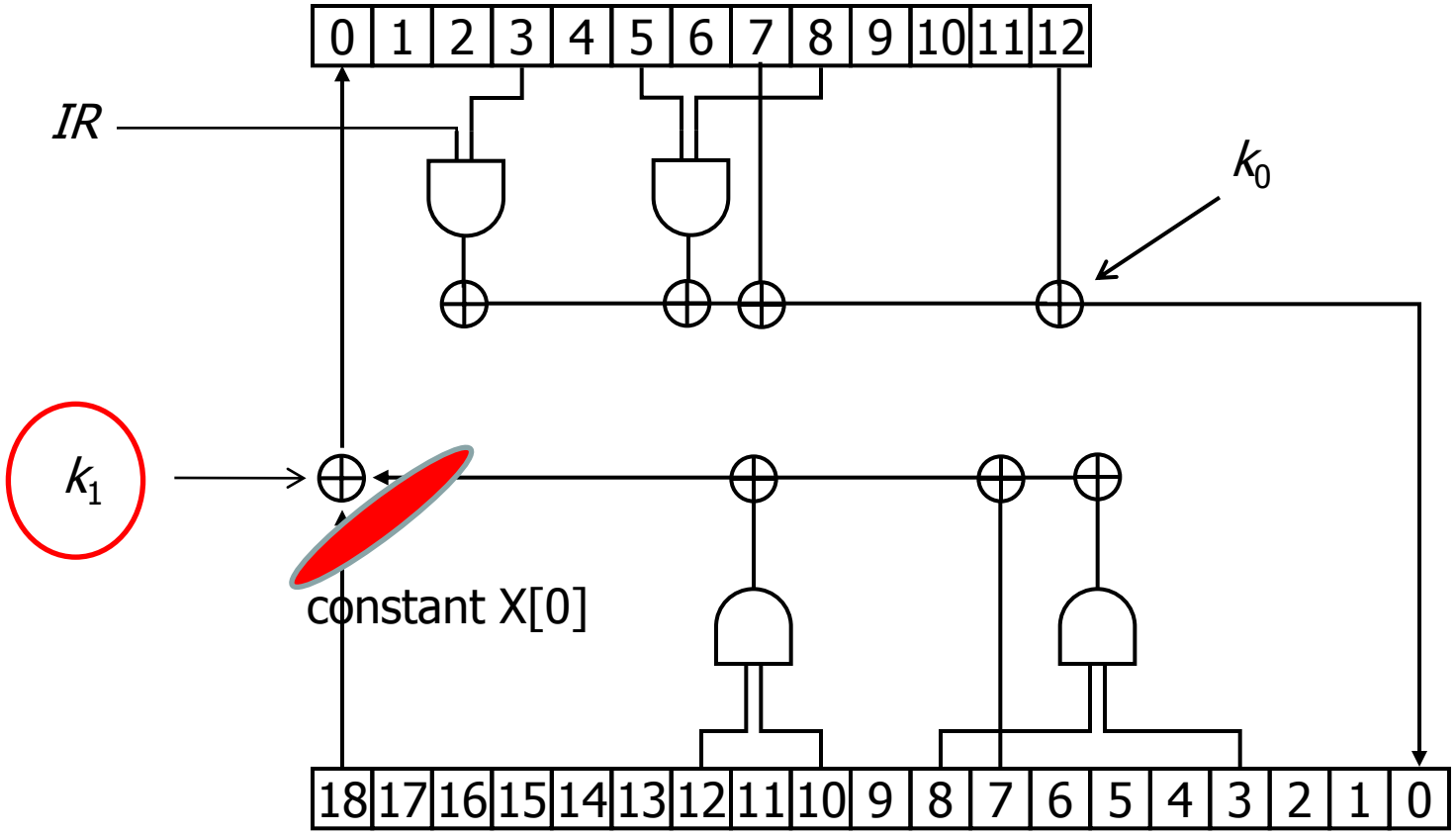
Function Reduction of KATAN32

$i = 0$

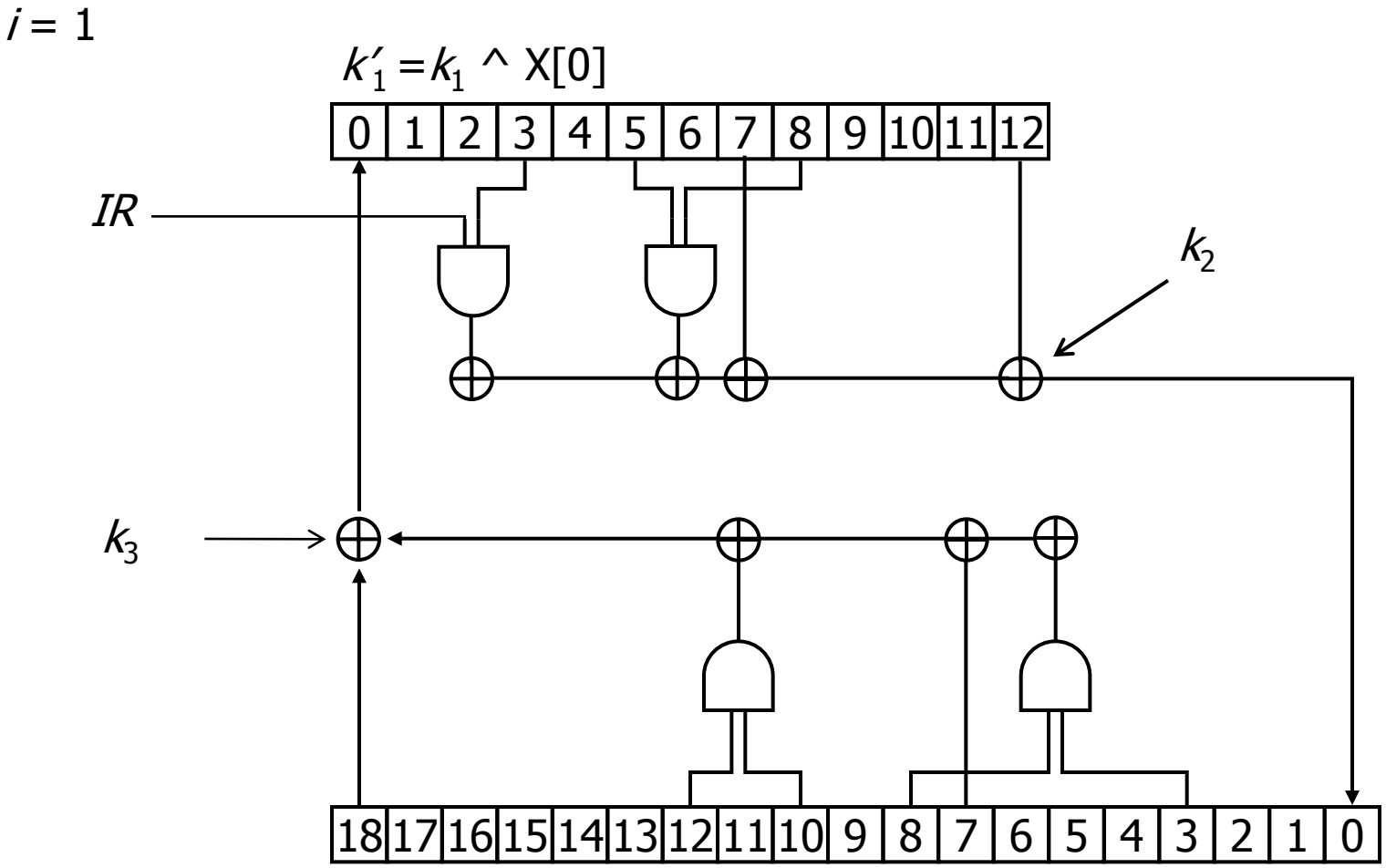


Function Reduction of KATAN32

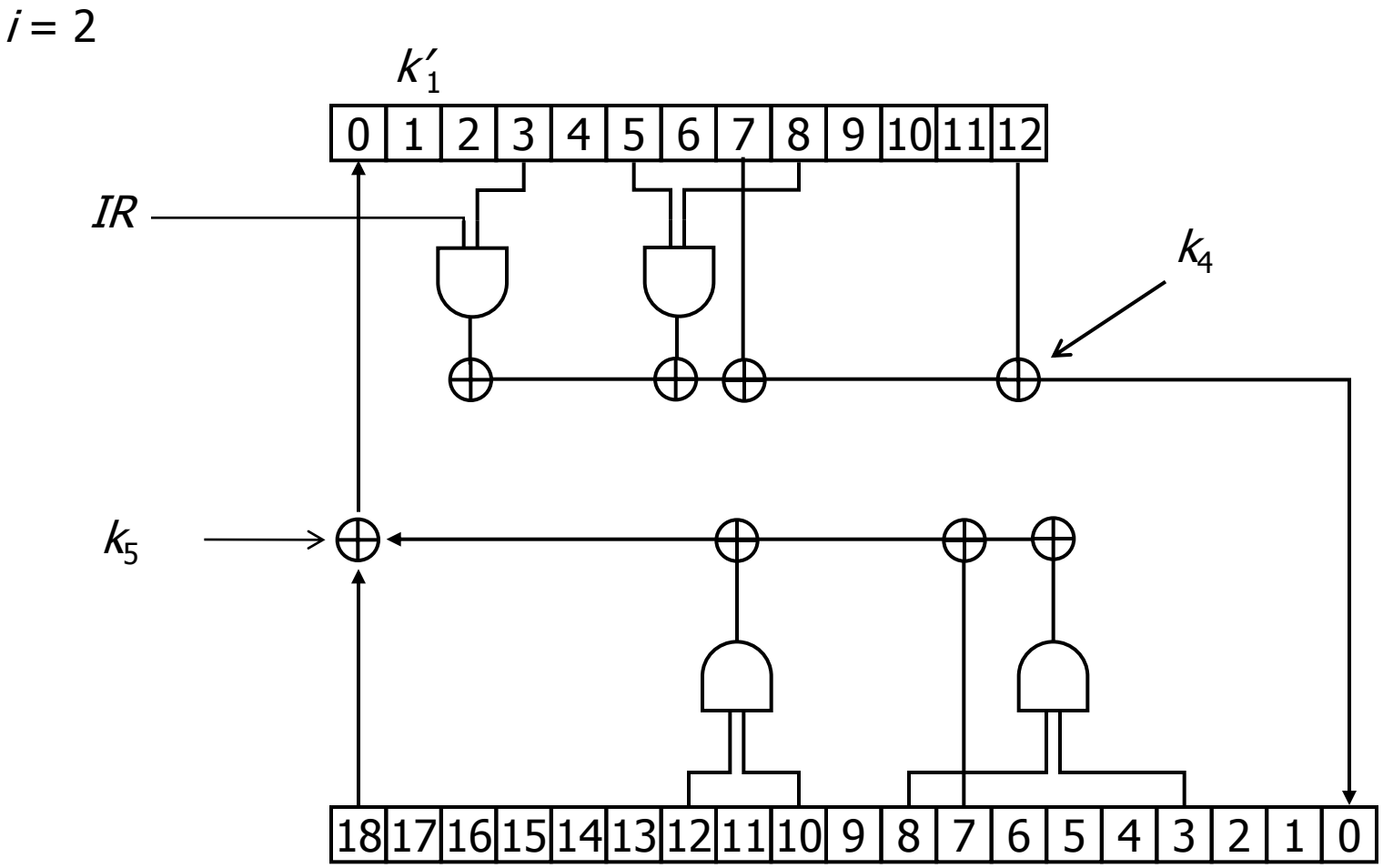
$i = 0$



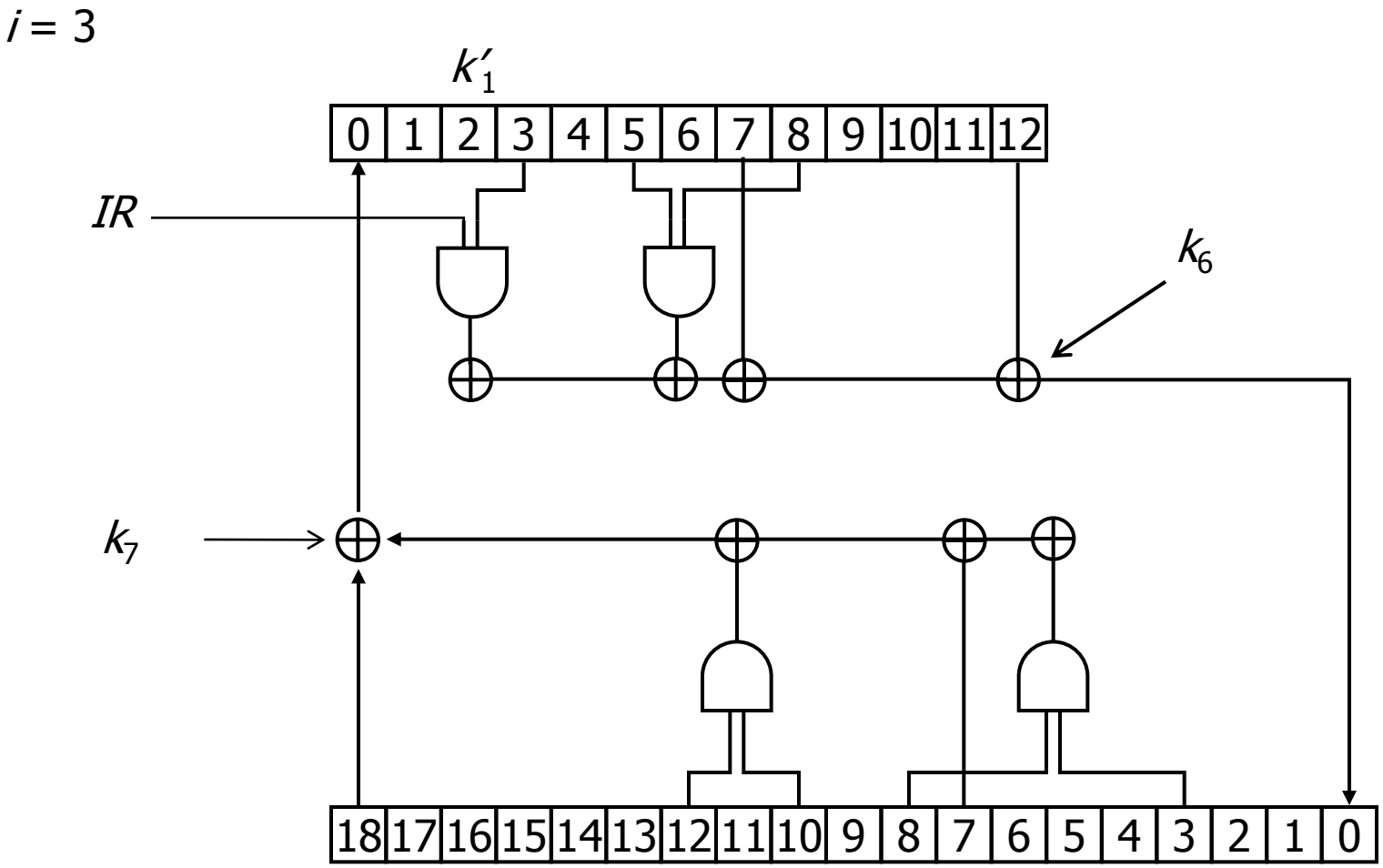
Function Reduction of KATAN32



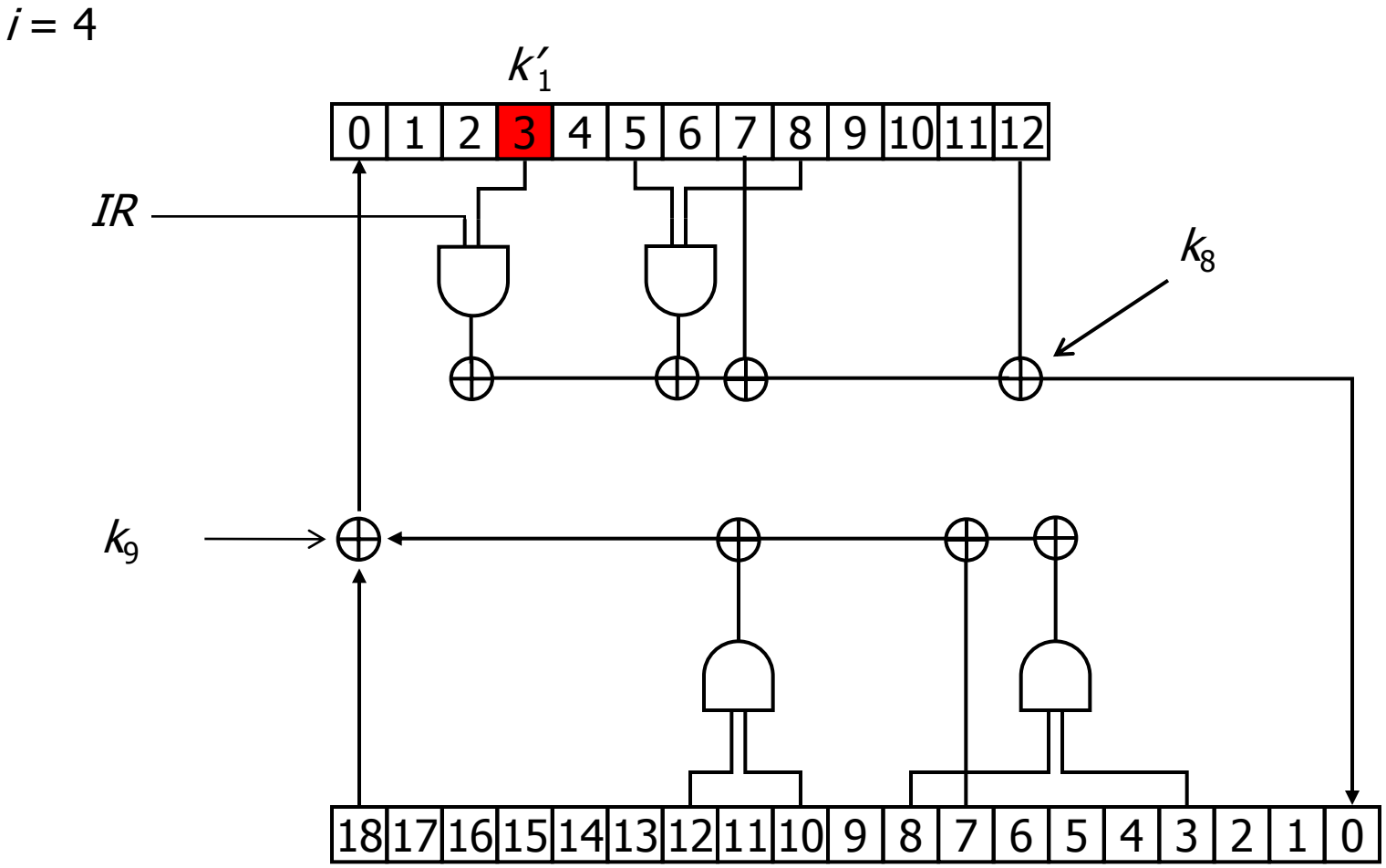
Function Reduction of KATAN32



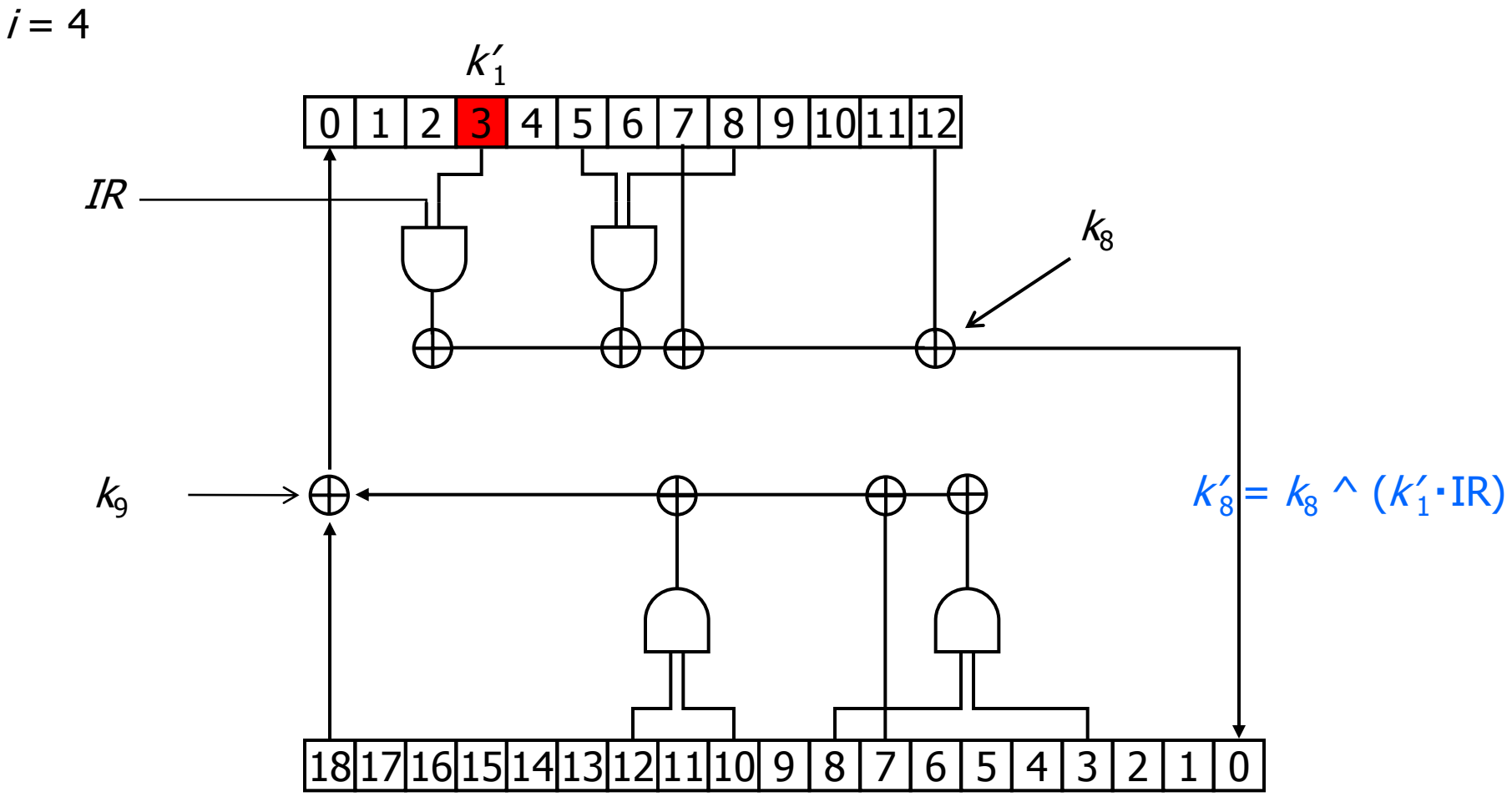
Function Reduction of KATAN32



Function Reduction of KATAN32

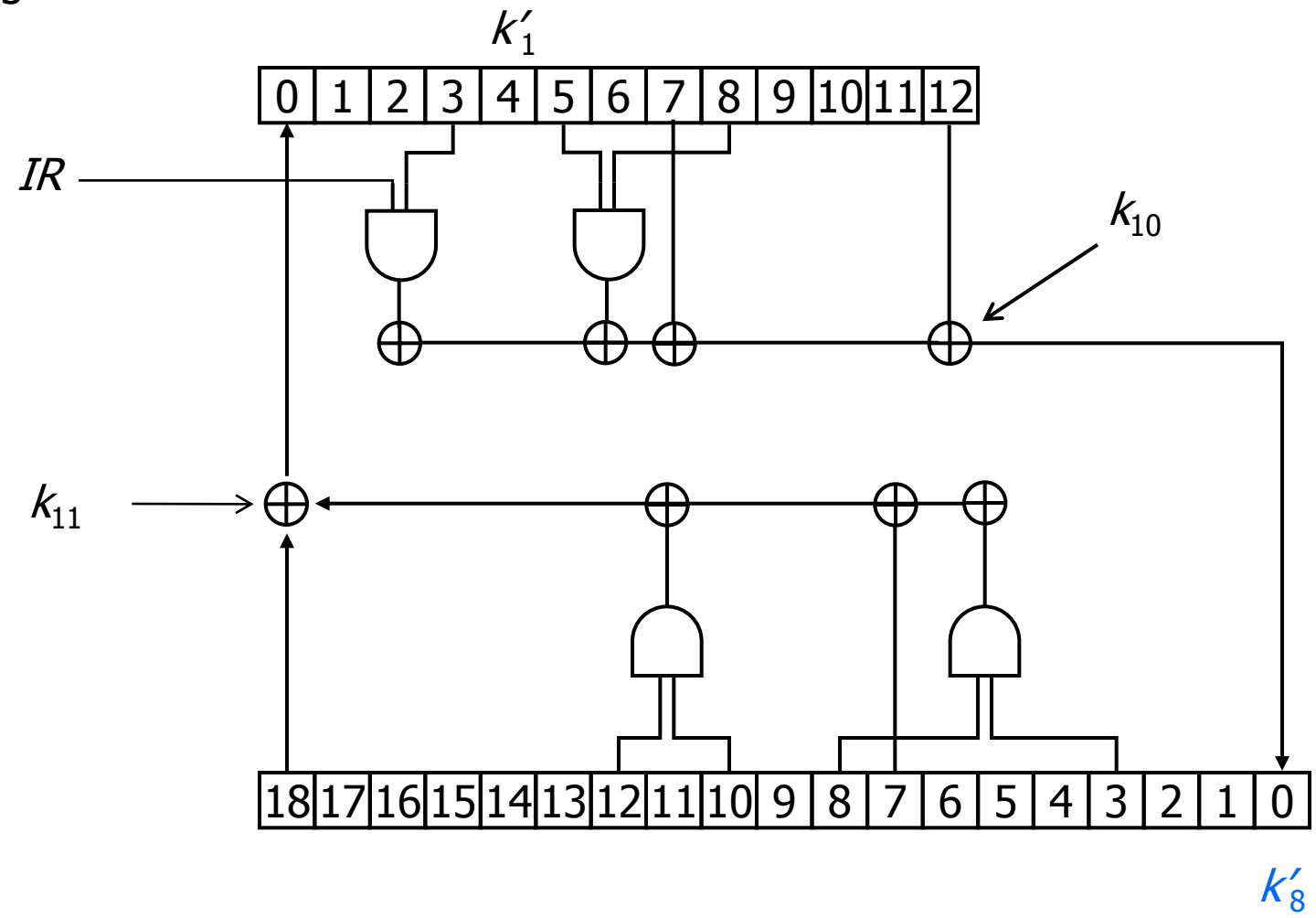


Function Reduction of KATAN32

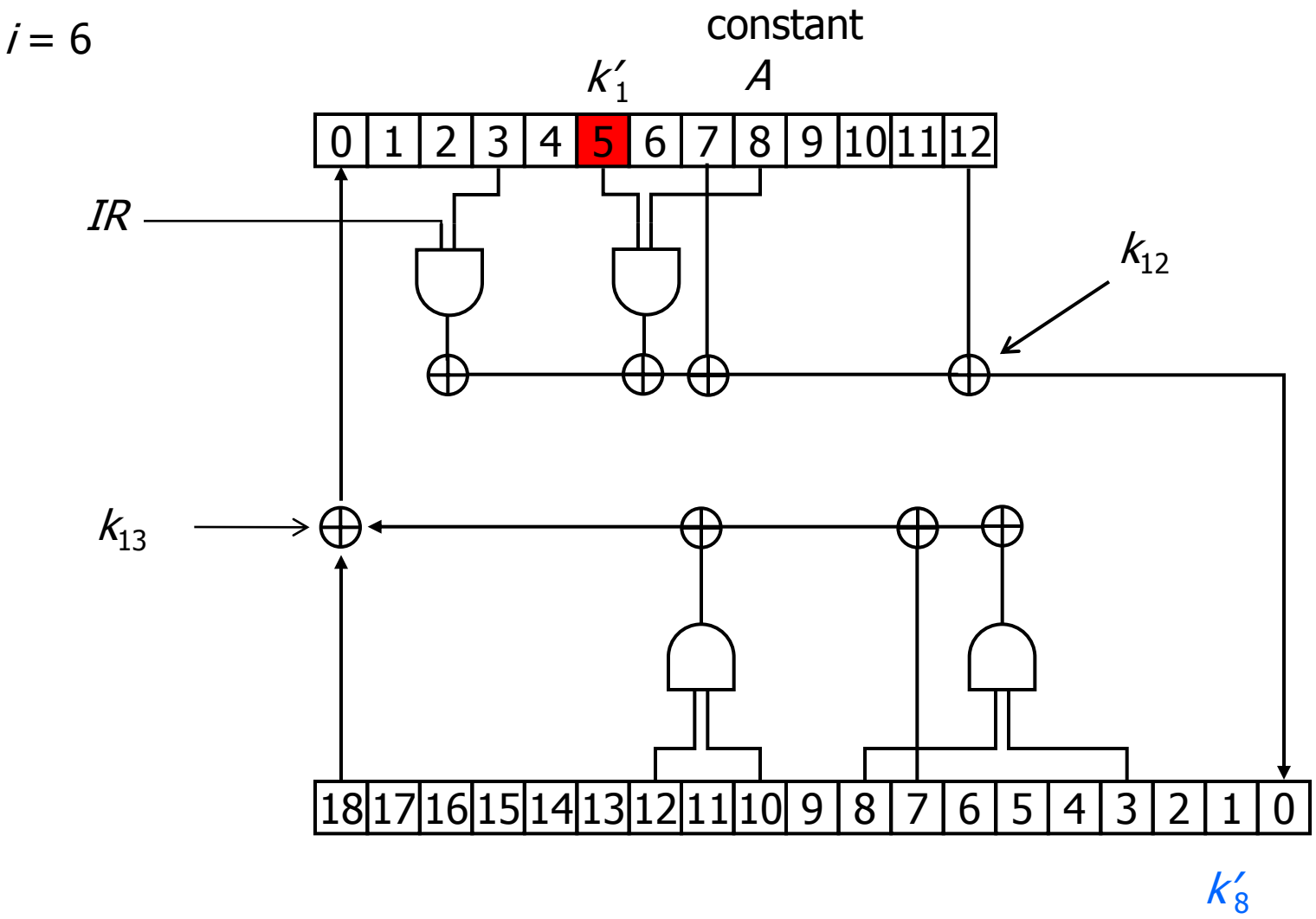


Function Reduction of KATAN32

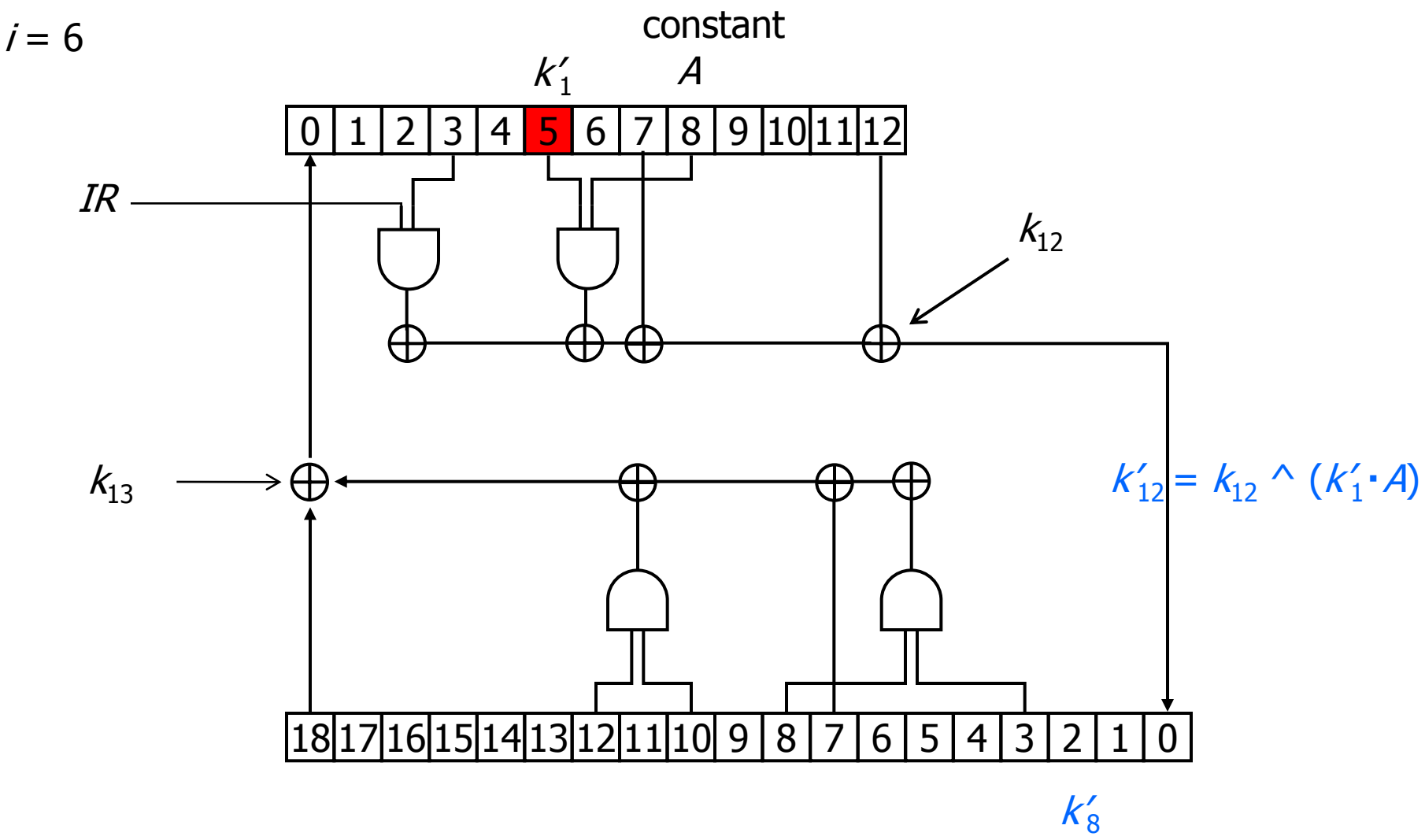
$i = 5$



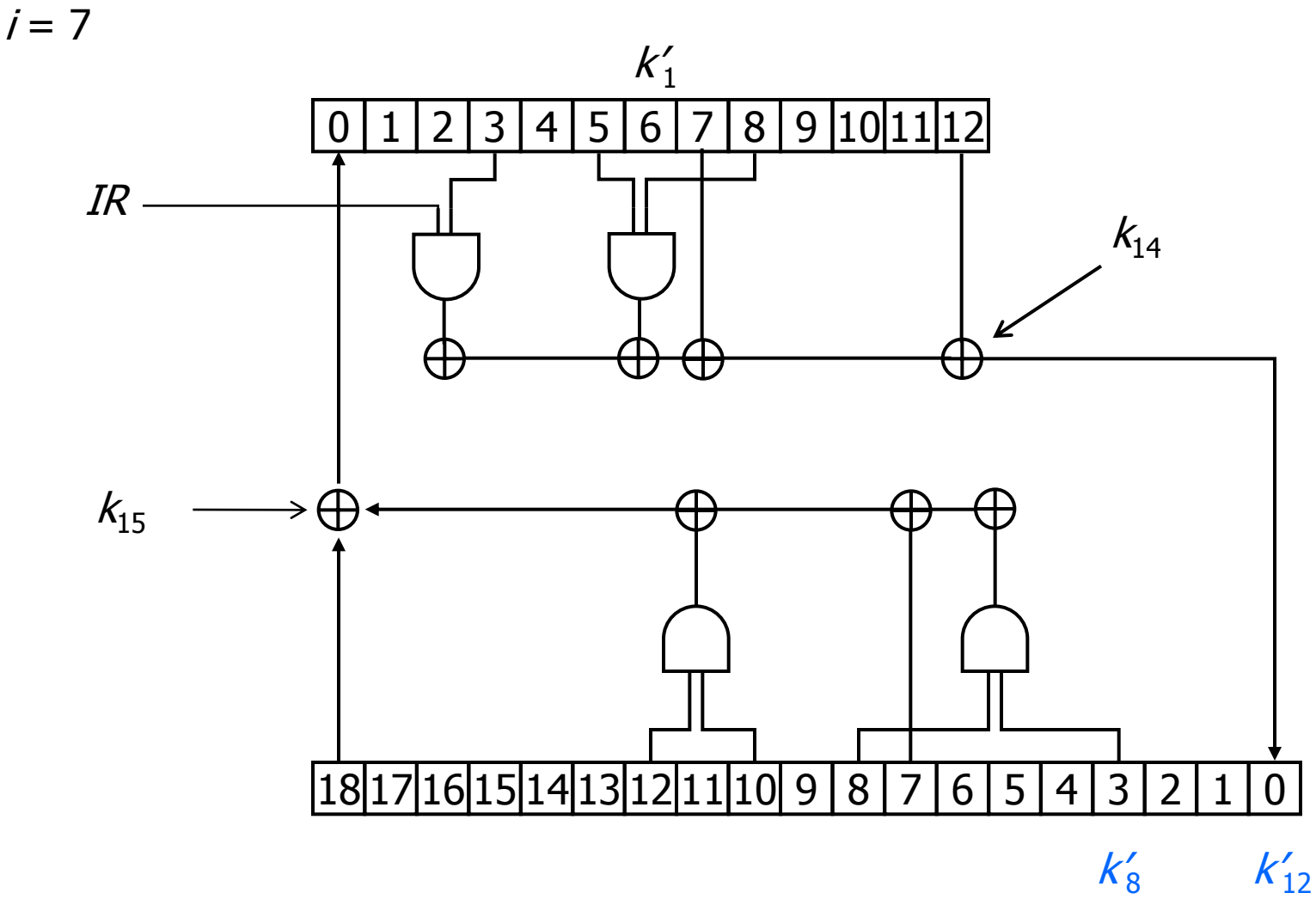
Function Reduction of KATAN32



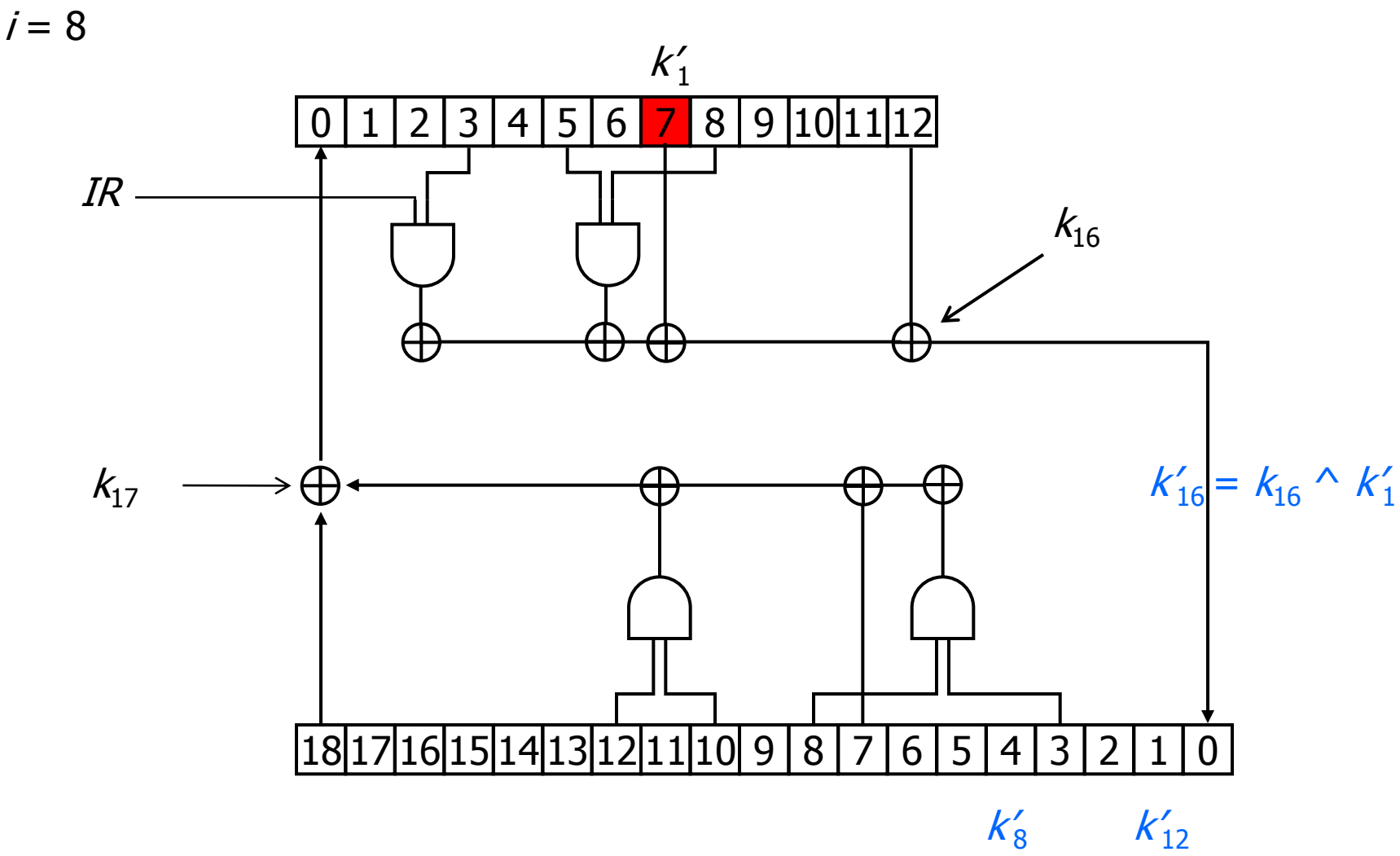
Function Reduction of KATAN32



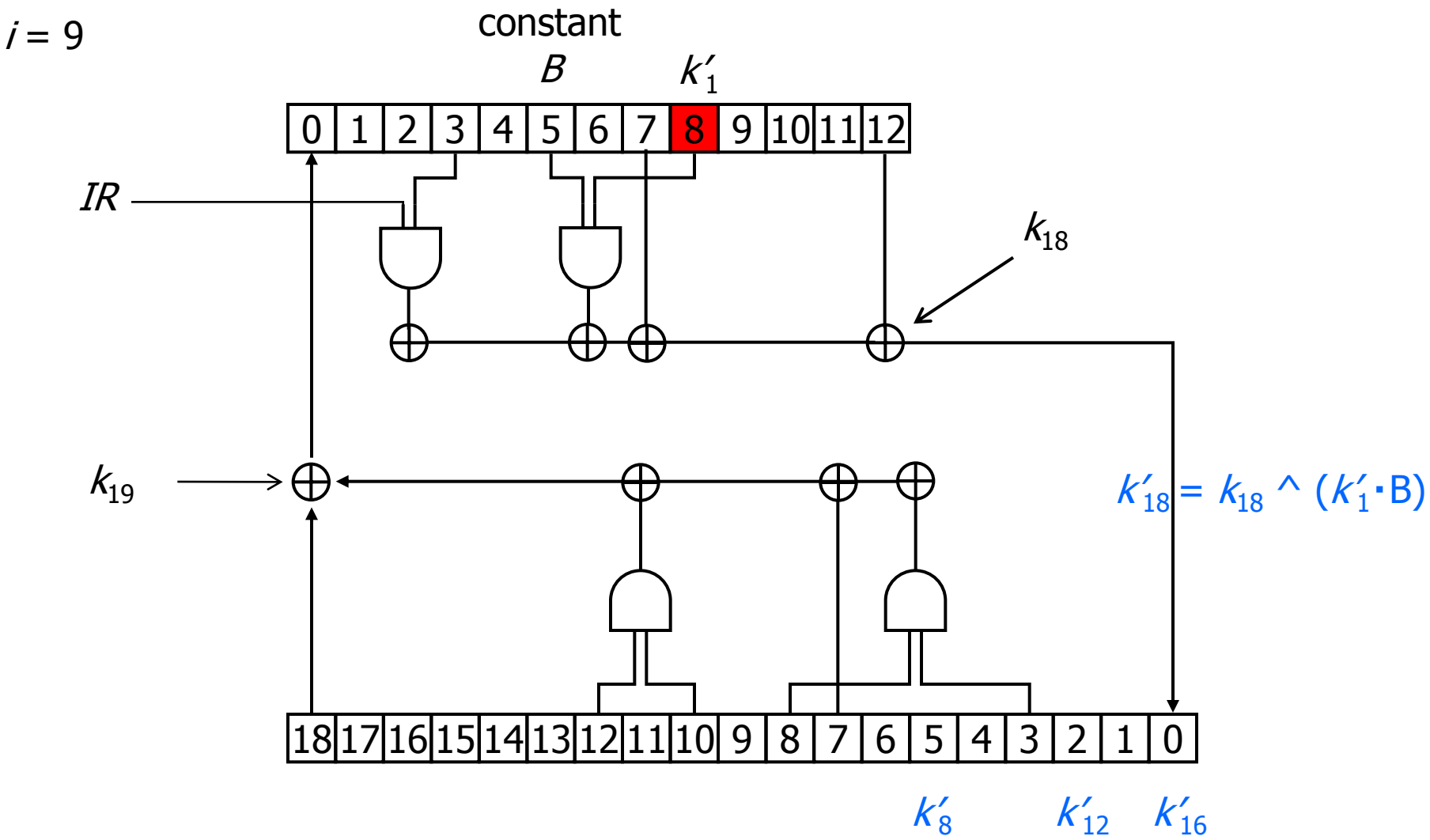
Function Reduction of KATAN32



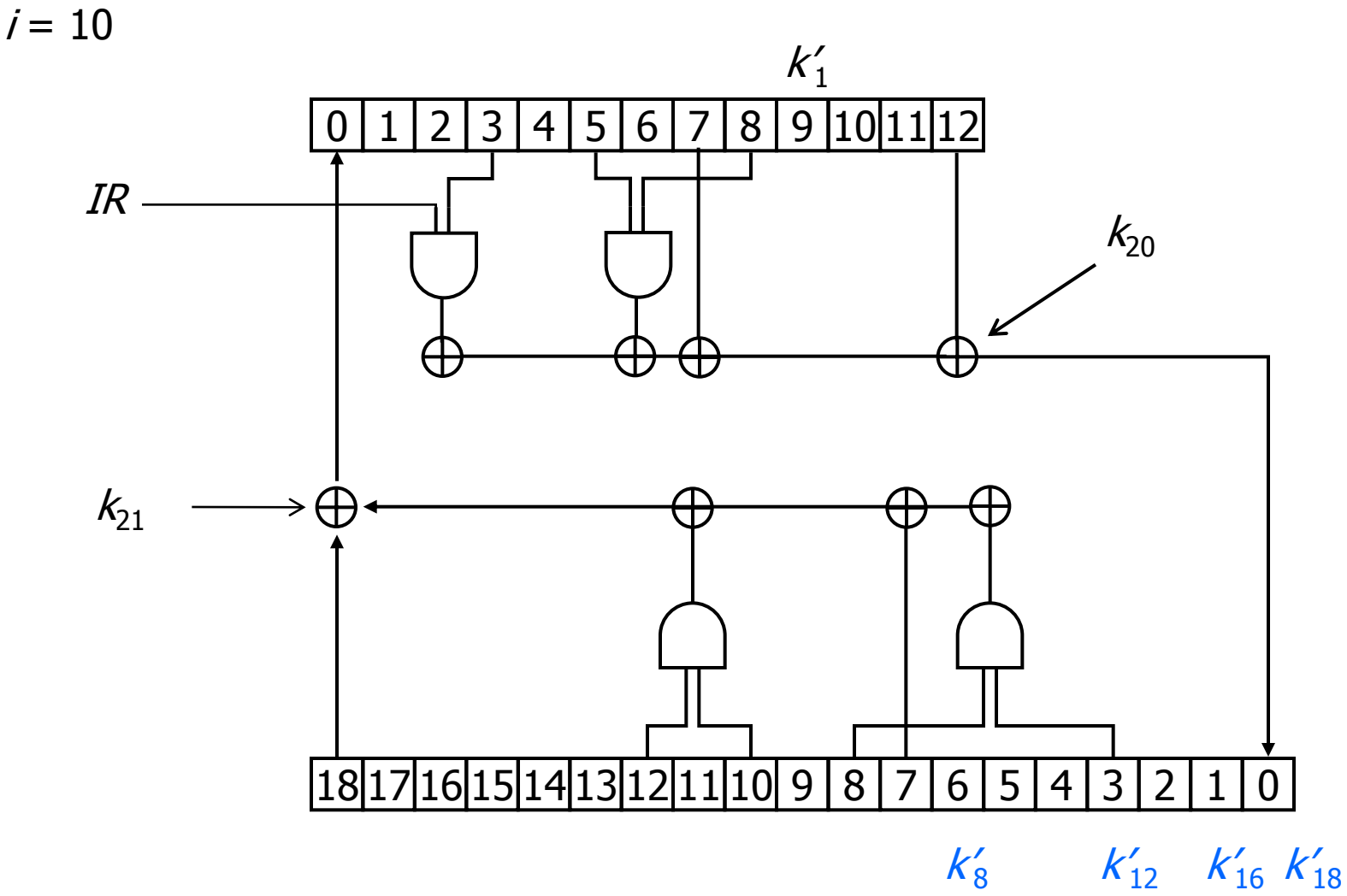
Function Reduction of KATAN32



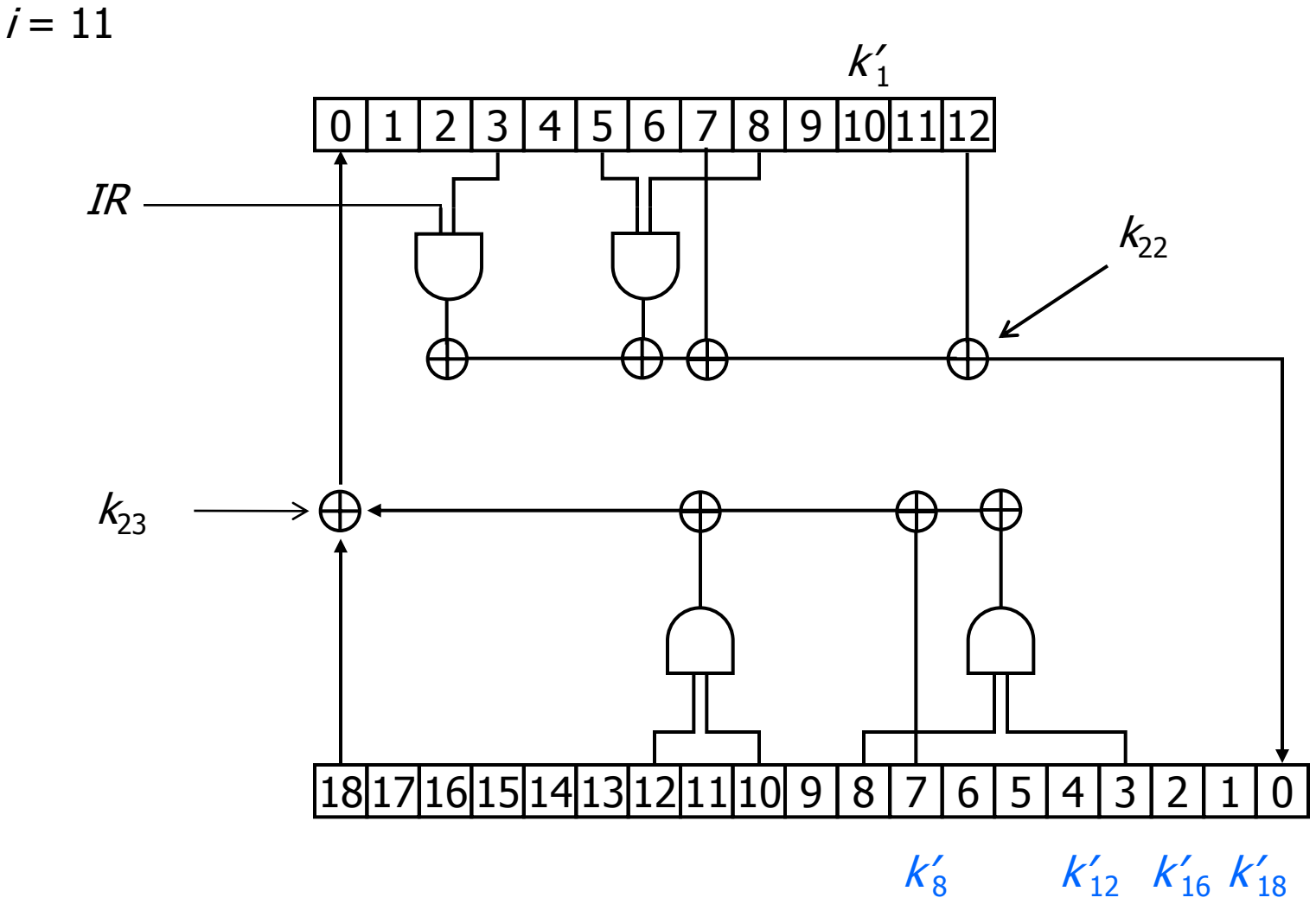
Function Reduction of KATAN32



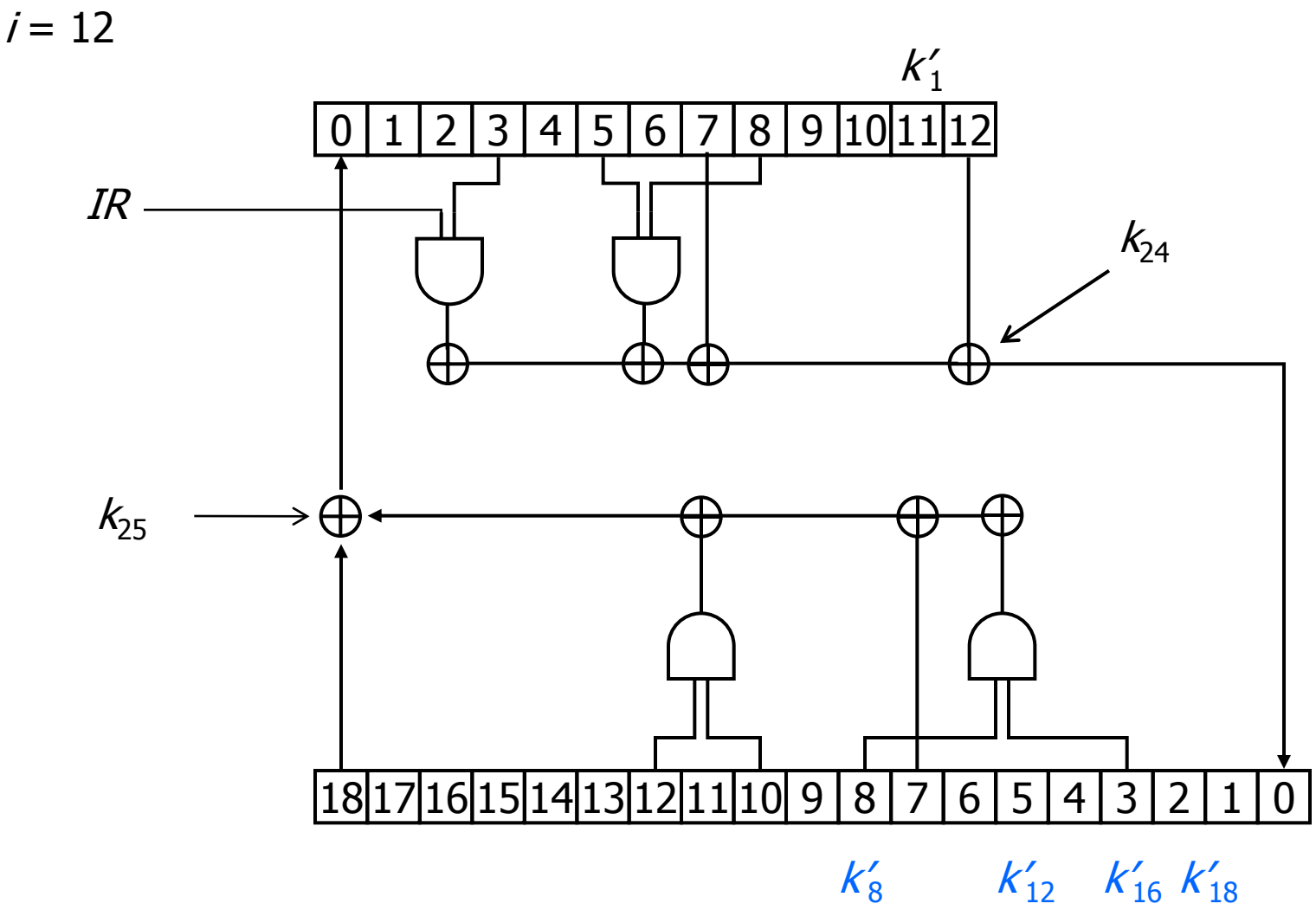
Function Reduction of KATAN32



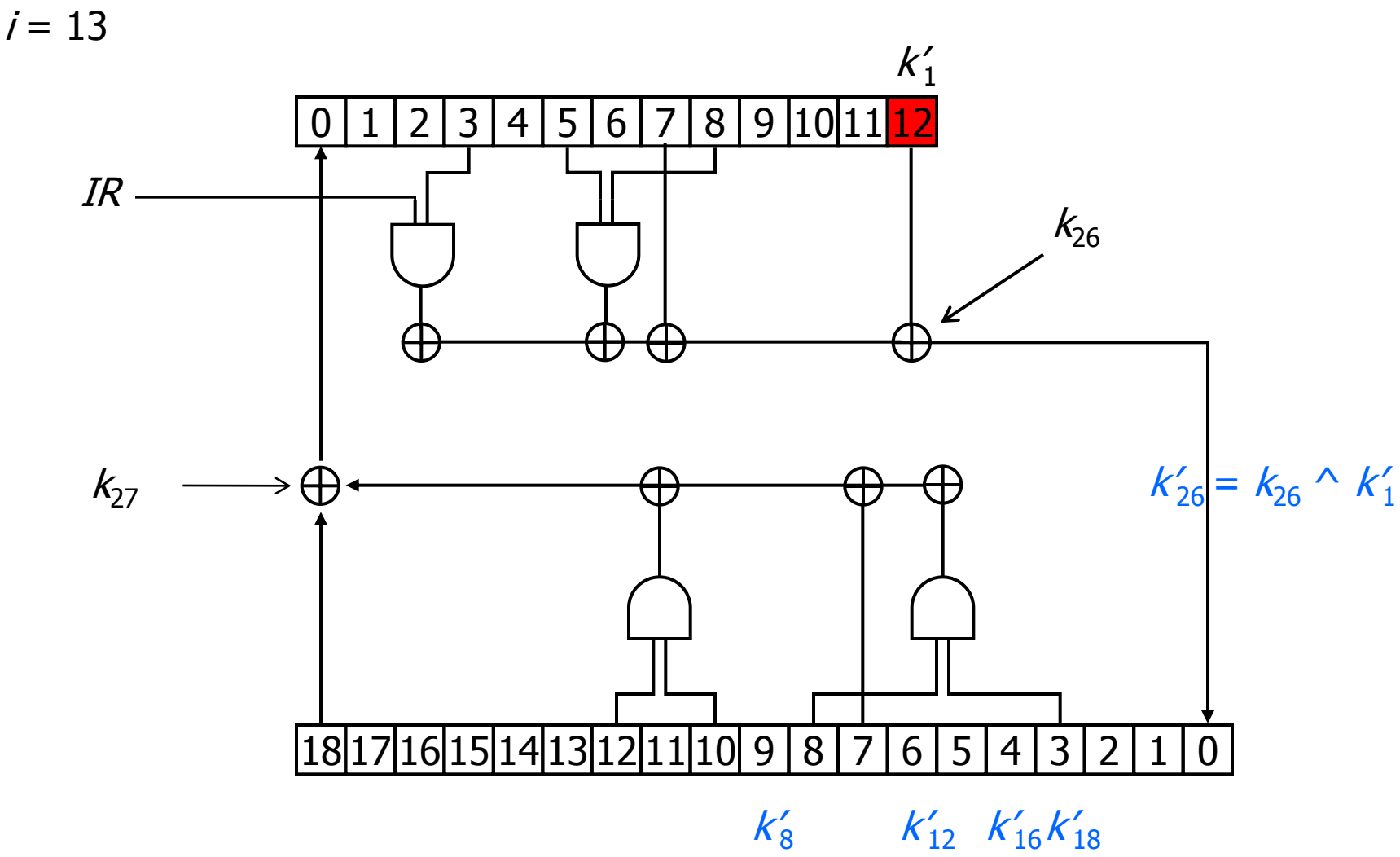
Function Reduction of KATAN32



Function Reduction of KATAN32

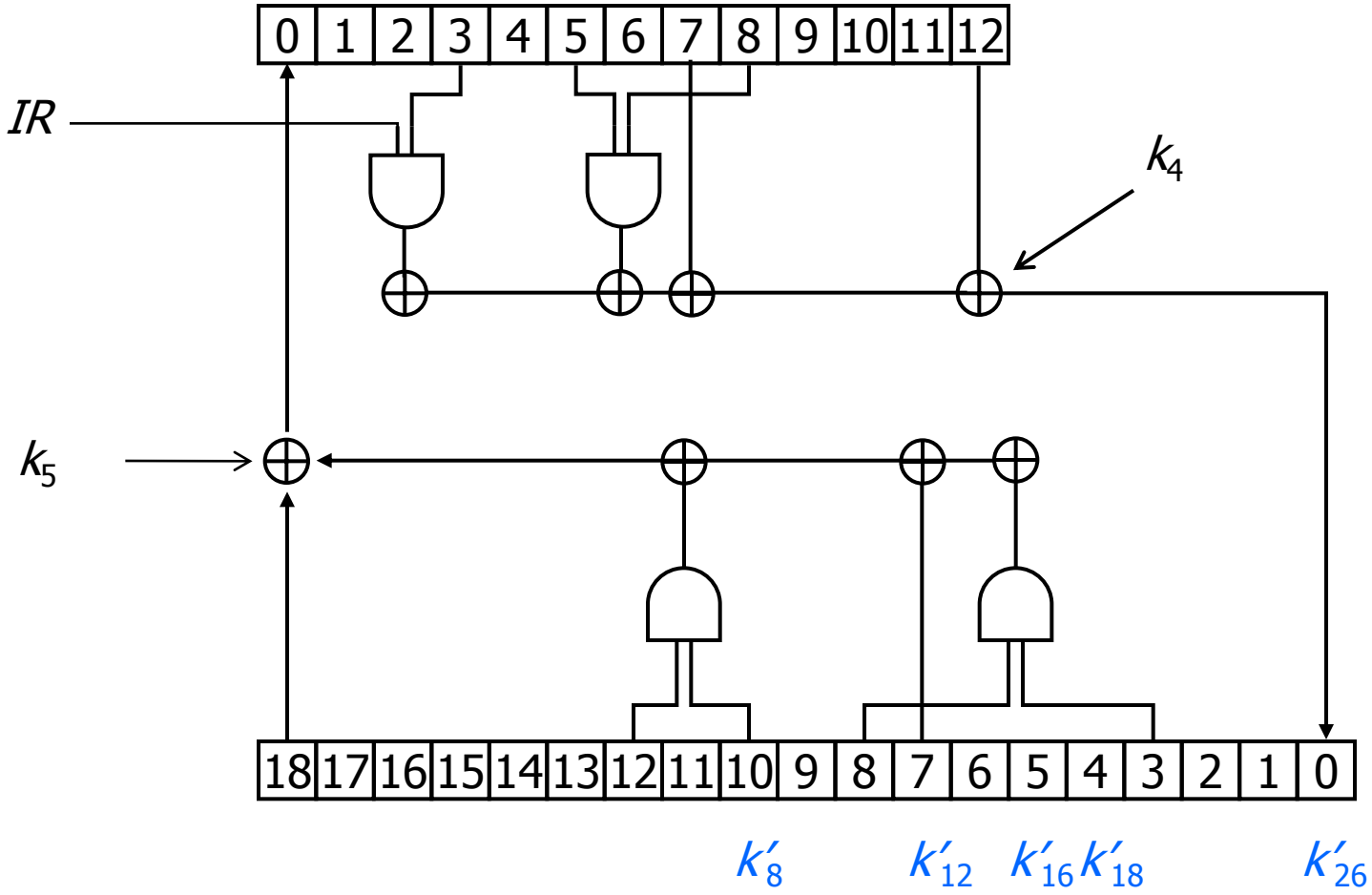


Function Reduction of KATAN32



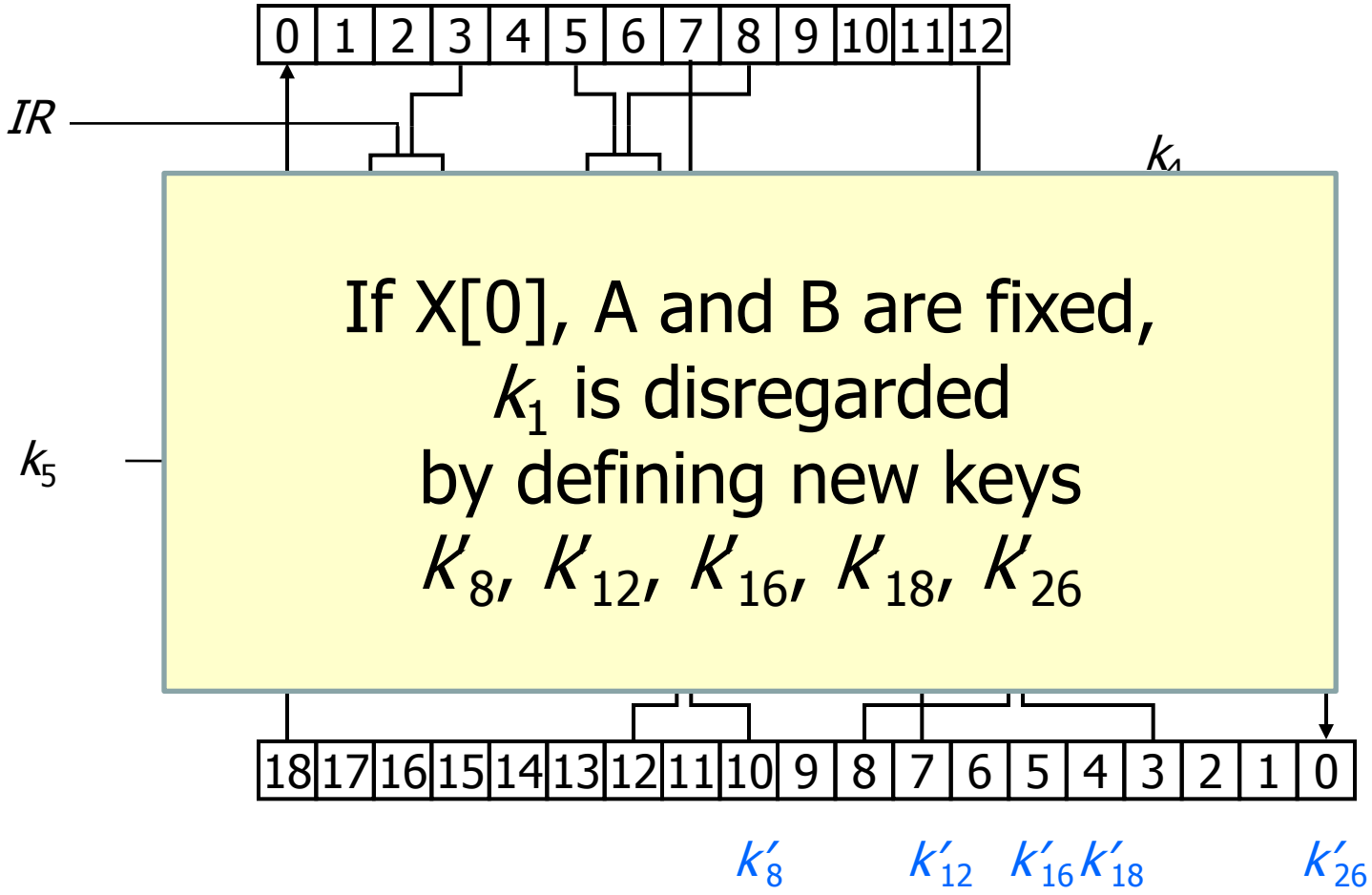
Function Reduction of KATAN32

$i = 13$



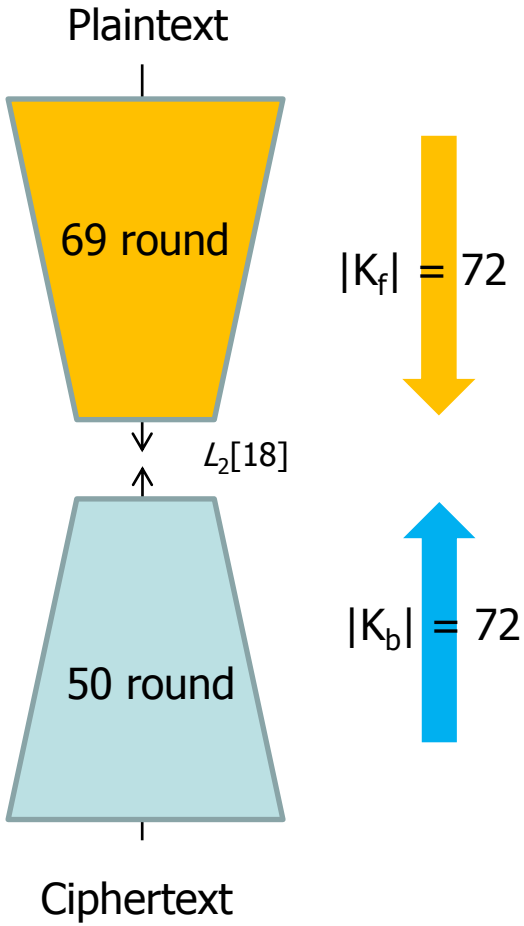
Function Reduction of KATAN32

$i = 13$



119-round Attack on KATAN32

- 8-bit function reduction in forward direction
  - by controlling 23 bits of plaintexts



Given 144 data,

2^{144} candidates are reduced to $2^0(2^{240-1\cdot144}) = 1$

Time : $2^{72} \times 144 = 2^{79.1}$
Data : 144
Memory : $2^{72} \times 144 = 2^{79.1}$

Results

- Update best single-key attacks on KATAN 32/48/64 w.r.t. number of attacked rounds

Best Attacks

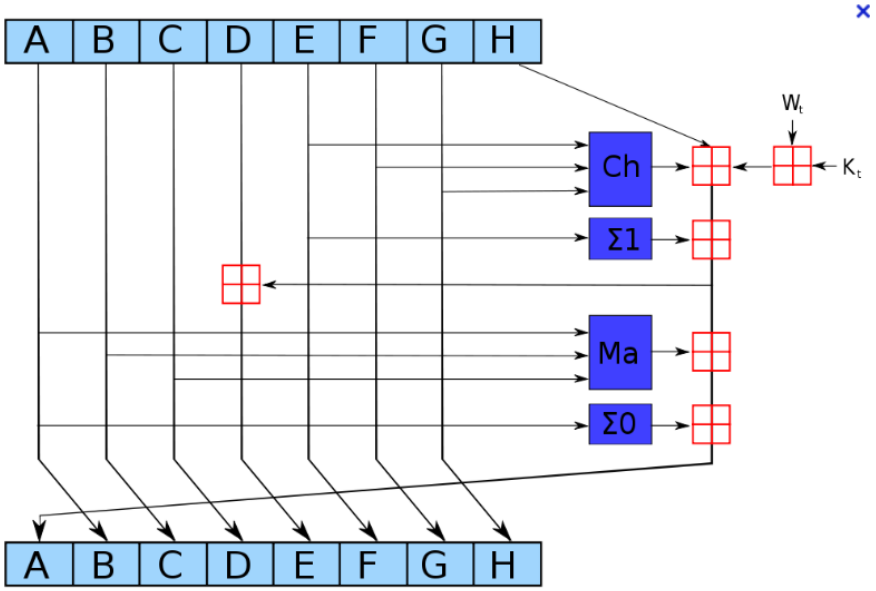
Target	#Attack round	Attack Type	Time	Memory	Data	Paper
KATAN32	110	ASR	2^{77}	$2^{75.1}$	138	[16]
	114	Differential	2^{77}	Not given	$2^{31.9}$	[2]
	119	**ASR**	**$2^{79.1}$**	**$2^{79.1}$**	**144**	**Ours**
KATAN48	100	ASR	2^{78}	2^{78}	128	[16]
	105	**ASR**	**$2^{79.1}$**	**$2^{79.1}$**	**144**	**Ours**
KATAN64	94	ASR	$2^{77.1}$	$2^{79.1}$	116	[16]
	99	**ASR**	**$2^{79.1}$**	**$2^{79.1}$**	**142**	**Ours**

Agenda

1. All-Subkeys Recovery Attacks
2. Function reduction technique
3. Improved Attacks on FOX-64/128
4. Improved Attacks on KATAN-32/48/64
5. Improved Attacks on SHACAL-2
6. Conclusion

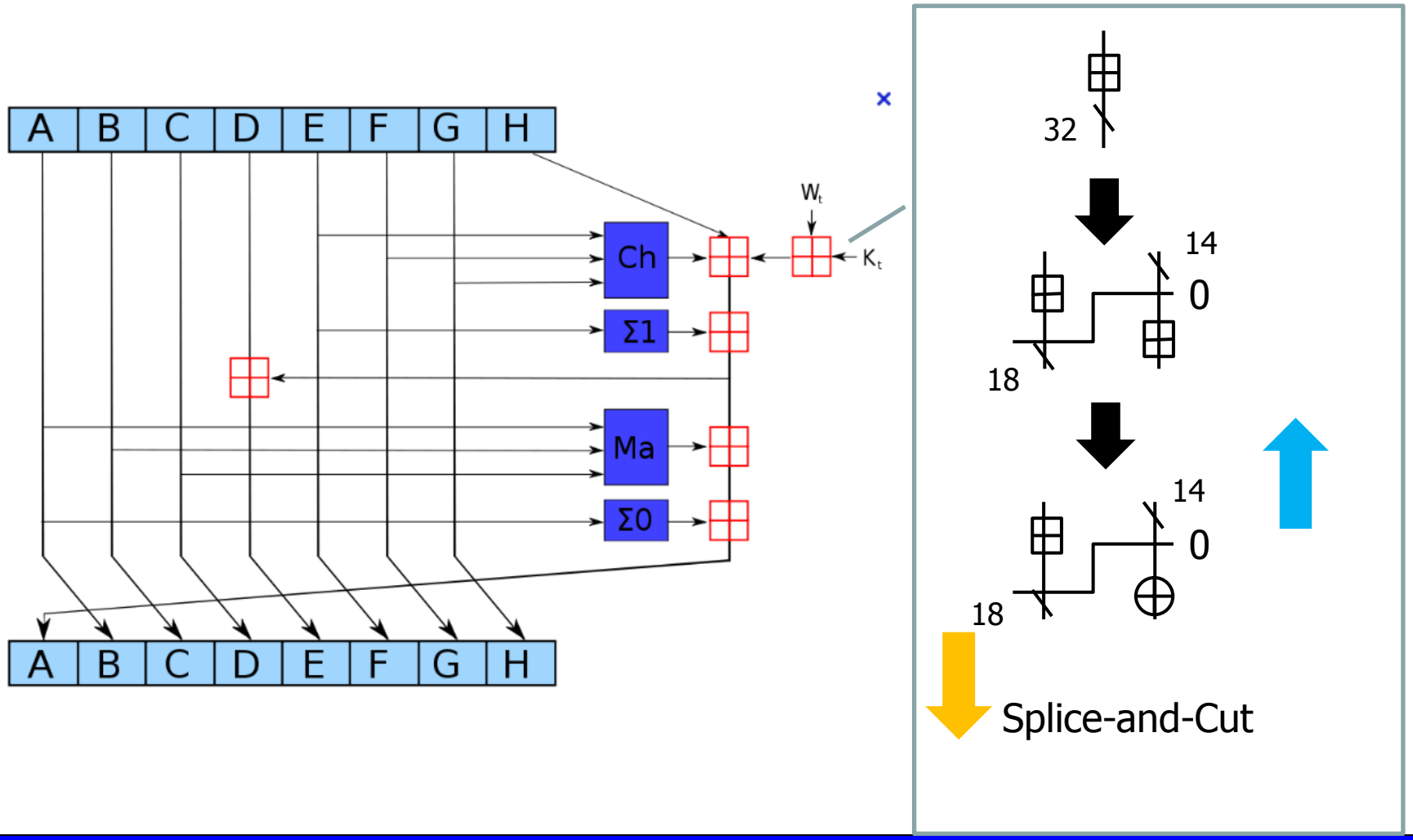
SHACAL-2

- Selected by NESSIE portfolio
- block size : 256 bits, key size : <= 512 bits
- Based on SHA-256 compression function
  - 64-round GFN like construction
- Best Attack : 41 round ASR attack



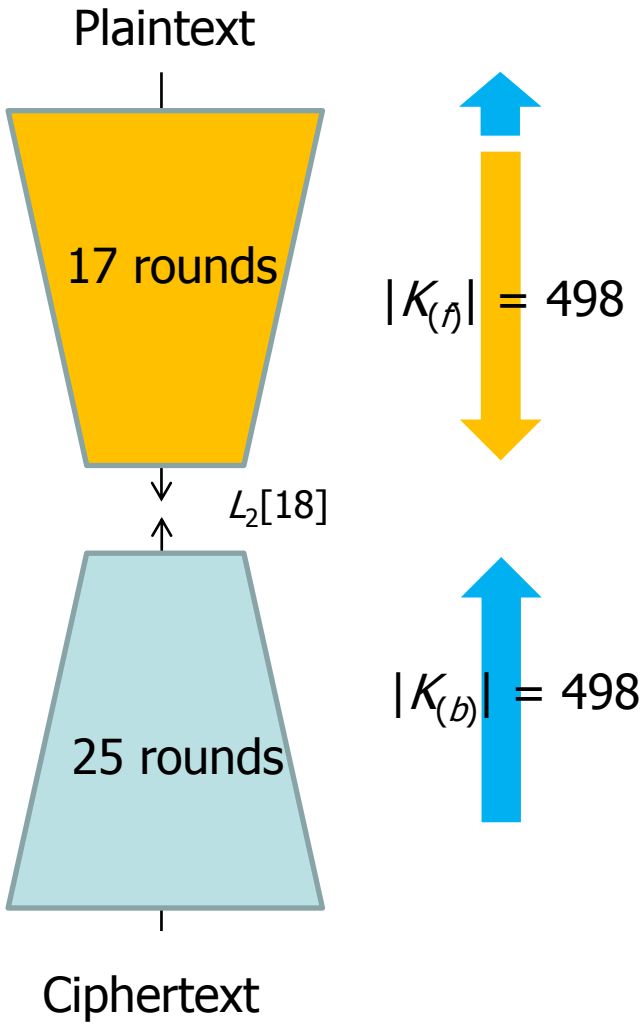
Key Linearization of SHACAL-2

- Key Linearization w/ splice-and-cut



42-round Attack on SHACAL-2

- Splice-and-Cut approach



Given 996 data,

2^{996} candidates are reduced to $2^0(2^{240-1\cdot144})$
$= 1$

Time : $2^{498} \times 996 = 2^{508}$
Data : 2^{25}
Memory : $2^{498} \times 996 = 2^{508}$

Conclusion

- Improved "All-Subkeys Recovery Attack"
  - Extended "Function Reduction" to other constructions
    - Exploit structure-dependent properties of Lai-Massey and LFSR-type
  - Utilized "Repetitive All-Subkeys Recovery Attack"
  - Updated best single-key attacks of FOX64/128, KATAN32/48/64, SHACAL-2
    - 7-round attacks on FOX64/128
    - 119/110/99-round attacks on KATAN32/48/64
    - 42-round attack on SHACAL-2

Thank you for your attention!