Dismantling MIFARE Classic

Flavio D. Garcia

Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.

ESORICS 2008

Joint work with: Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur and Bart Jacobs

Outline

Timeline

Dec 2007	CCC presentation by Nohl and Plotz
March 2008	We recover CRYPTO1 and found attacks.
March 2008	We notified the manufacturer and other stakeholders (without disclosure).
Jun 2008	NXP tries to stop “irresponsible” publication, via injunction (court order).
July 2008	Judge refuses to prohibit, basically on freedom of expression. Also:
	“University acted with due care, warning stakeholders early on”
	“Damage is not result of publication, but of apparent deficiencies in the cards”
	NXP did not appeal

RFID Tags

MIFARE

MIFARE product family from NXP

- Ultralight
- Classic or Standard (320B, 1KB and 4KB)
- DESFire
- SmartMX

MIFARE

MIFARE product family from NXP

- Ultralight
- Classic or Standard (320B, 1KB and 4KB)
- DESFire
- SmartMX

MIFARE dominance

- Over 1 billion MIFARE cards sold
- Over 200 million MIFARE Classic cards in use covering 85% of the contactless smart card market

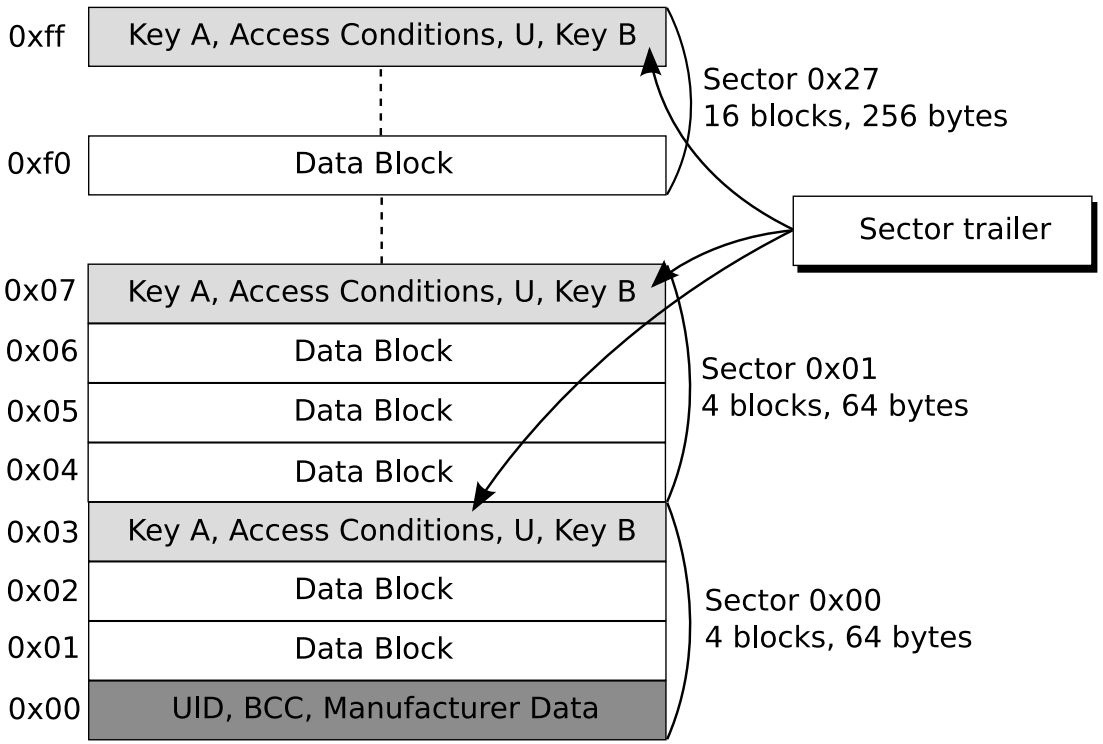
MIFARE Classic

Some systems using MIFARE Classic

- Access to our university building
- Used in many office and official buildings
- Public transport systems
  - OV-Chipkaart (Netherlands)
  - Oyster card (London)
  - Smartrider (Australia)
  - EMT (Malaga) ¨‿
- Personnel entrance to Schiphol Airport (Amsterdam)
- Access to Dutch military bases
- Popular payment system in Asia

Reverse Engineering MIFARE Classic

Logical structure of the MIFARE Classic 4K



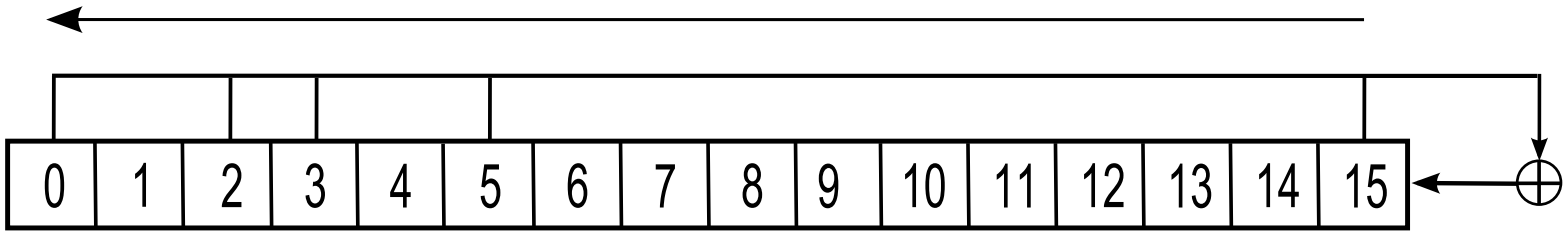
MIFARE Classic

- Proprietary stream cipher CRYPTO1.
- Key length of only 48 bits.

Weak pseudo-random generators

- 16 bit state nonce pseudo-random generator on the tag.
- 32 bit nonces.
- Reader gives the same sequence of nonces after power up.
- The pseudo-random generator on the tag iterates over time.
- Generated nonces on the tag only depend on uptime.

Nonce generating LFSR on the tags



Authentication Trace

Example (uid $\oplus n_T = C$)

Step	Sender	Hex	Abstract
01	Reader	26	req type A
02	Tag	04 00	answer req
03	Reader	93 20	select
04	Tag	c2 a8 2d f4 b3	uid,bcc
05	Reader	93 70 c2 a8 2d f4 b3 ba a3	select(uid)
06	Tag	08 b6 dd	MIFARE 1k
07	Reader	60 30 76 4a	auth(block 30)
08	Tag	42 97 c0 a4	n_T
09	Reader	7d db 9b 83 67 eb 5d 83	$n_R \oplus \mathrm{ks}_1, a_R \oplus \mathrm{ks}_2$
10	Tag	8b d4 10 08	$a_T \oplus \mathrm{ks}_3$

Another Authentication Trace

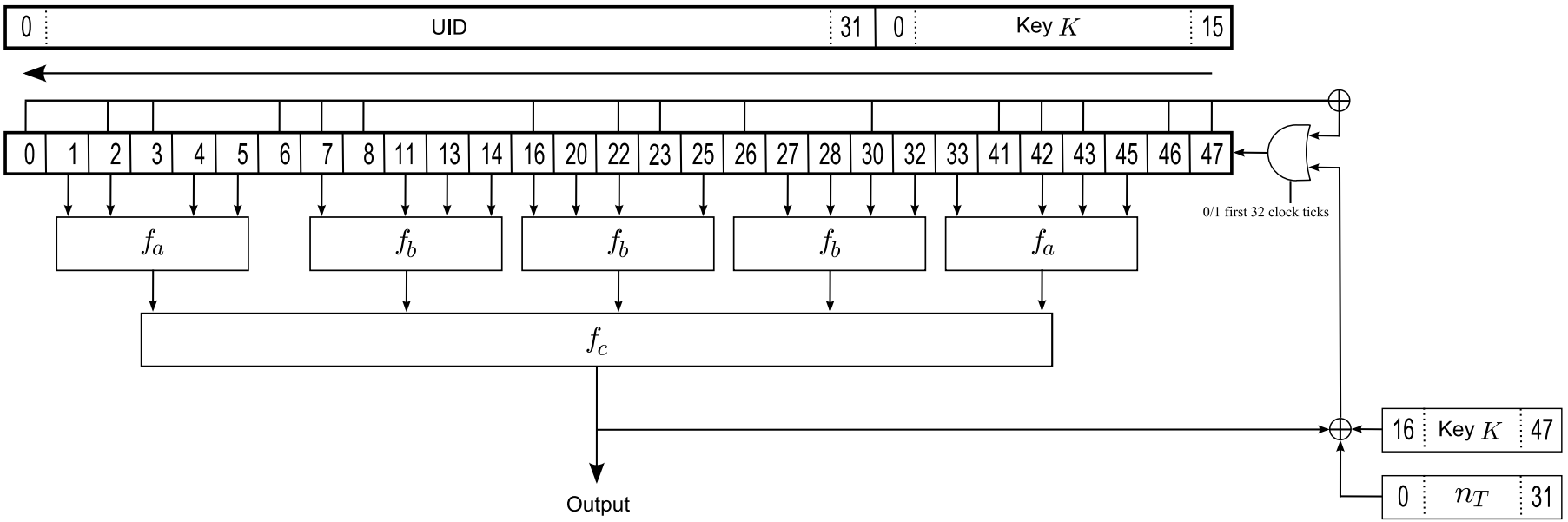
Example (uid$'$ $\oplus n'_T = C$)

Step	Sender	Hex	Abstract
01	Reader	26	req type A
02	Tag	04 00	answer req
03	Reader	93 20	select
04	Tag	1d fb e0 33 35	uid',bcc
05	Reader	93 70 1d fb e0 33 35 d3 55	select(uid')
06	Tag	08 b6 dd	MIFARE 1k
07	Reader	60 30 76 4a	auth(block 30)
08	Tag	9d c4 0d 63	n'_T
09	Reader	7d db 9b 83 42 95 c4 46	$n_R \oplus \mathrm{ks}_1, a'_R \oplus \mathrm{ks}_2$
10	Tag	eb 3e f7 da	$a'_T \oplus \mathrm{ks}_3$

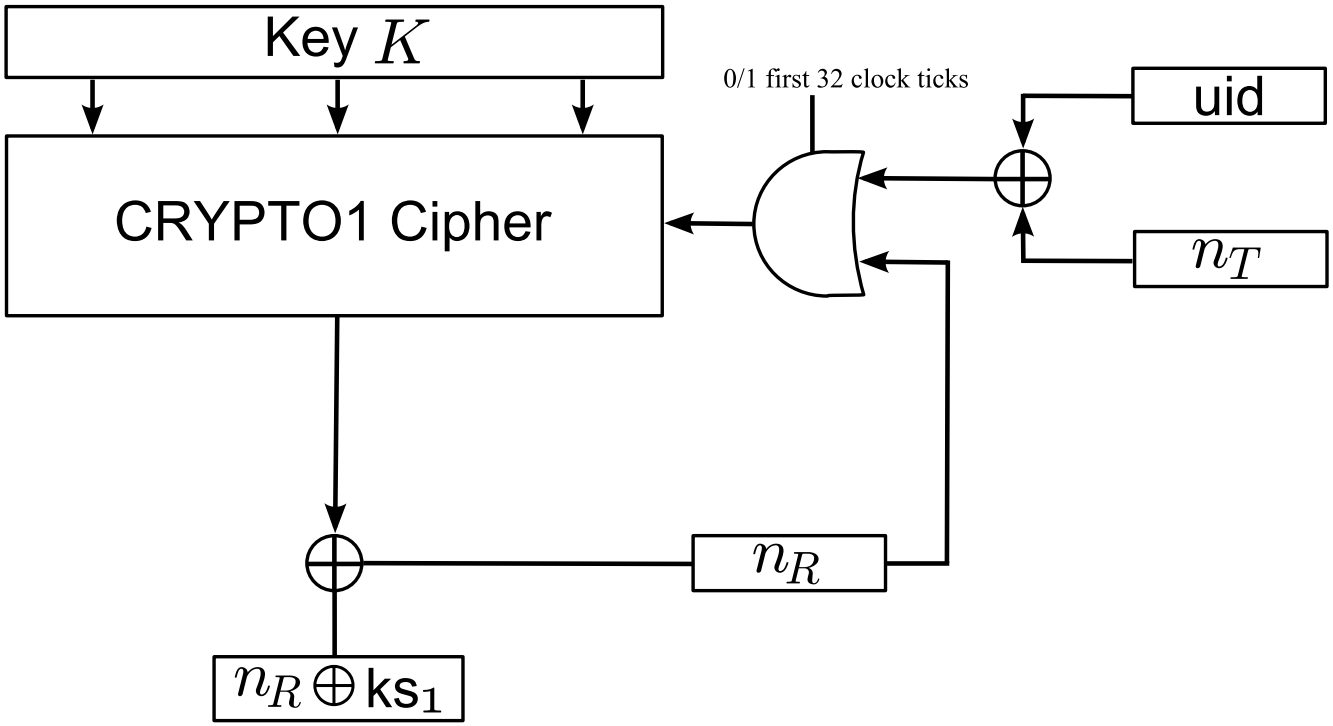
Authentication Protocol

	Tag		Reader
0		anti-c(uid) $\longrightarrow$	
1		$\longleftarrow$ auth(block)	
2	picks n_T		
3		n_T $\longrightarrow$	
4	$ks_1 \leftarrow \text{cipher}(K, \text{uid}, n_T)$		$ks_1 \leftarrow \text{cipher}(K, \text{uid}, n_T)$
5			picks n_R
6			$ks_2, ks_3 \ldots \leftarrow \text{cipher}(K, \text{uid}, n_T, n_R)$
7		$n_R \oplus ks_1, \text{suc}^2(n_T) \oplus ks_2$ $\longleftarrow$	
8	$ks_2, ks_3 \ldots \leftarrow \text{cipher}(K, \text{uid}, n_T, n_R)$		
9		$\text{suc}^3(n_T) \oplus ks_3$ $\longrightarrow$	

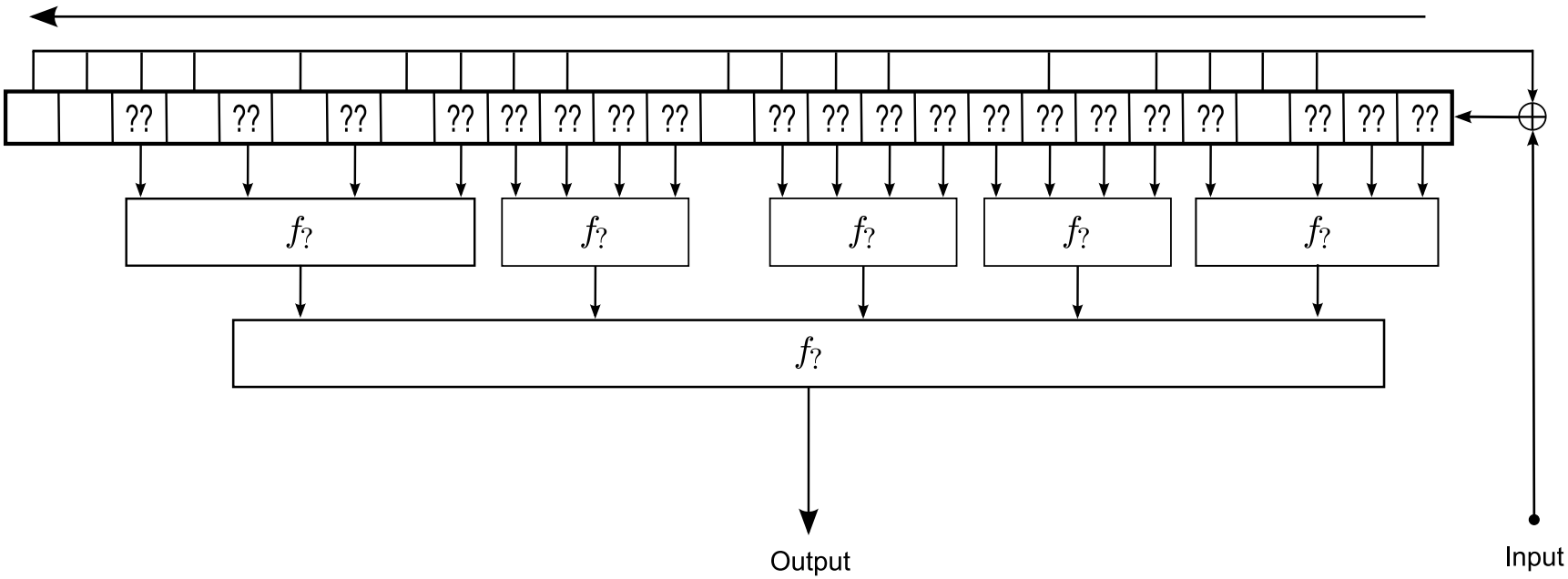
Hitag2 Cipher



Initialization Diagram



Guessed structure for CRYPTO1



Recovering the input taps to the filter function

Example

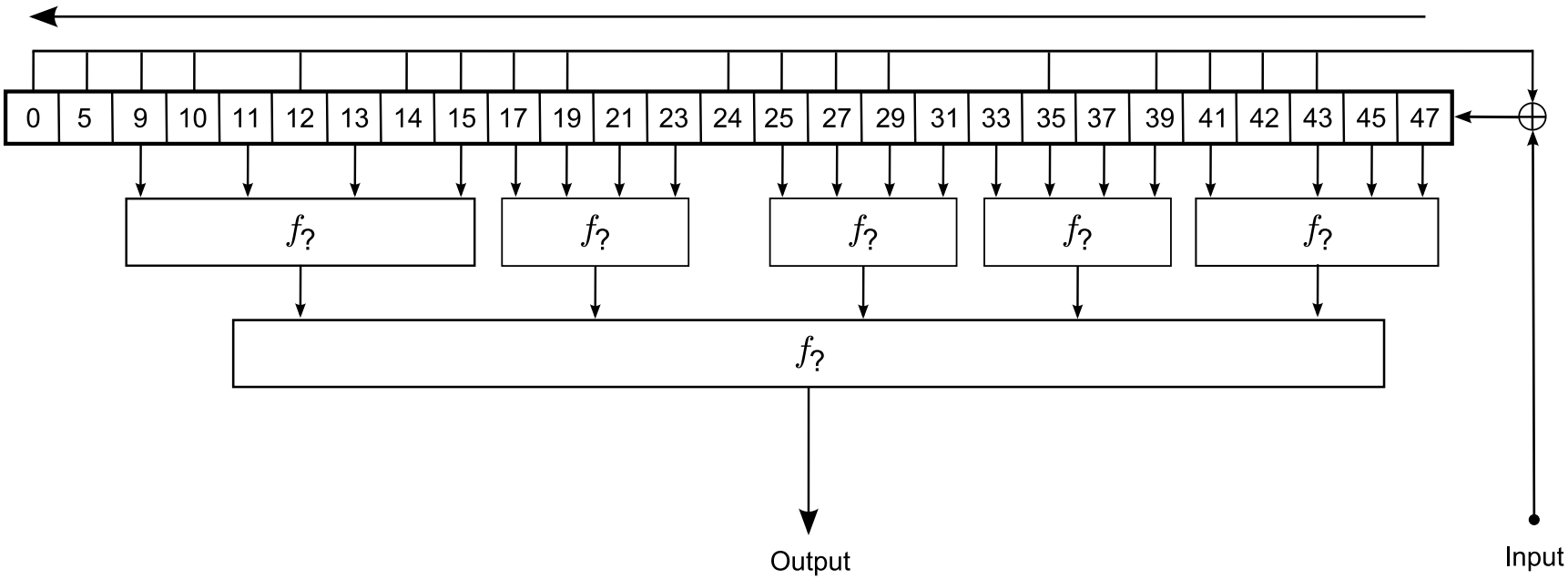
Sender	Hex	
Reader	26	req type A
Ghost	04 00	answer req
Reader	93 20	select
Ghost	00 00 00 00 00	uid,bcc
Reader	93 70 00 00 00 00 00 9c d9	select(uid)
Ghost	08 b6 dd	MIFARE 1k
Reader	60 00 f5 7b	auth(block 0)
Ghost	6d c4 13 ab d0 f3	n_T
Reader	df 19 d5 7a e5 81 ce cb	$n_R \oplus \mathrm{ks}_1, \mathrm{suc}^2(n_T) \oplus \mathrm{ks}_2$

Recovering the input taps to the filter function

Example (one bit difference LFSR state)

Sender	Hex	
Reader	26	req type A
Ghost	04 00	answer req
Reader	93 20	select
Ghost	00 00 00 00 00	uid,bcc
Reader	93 70 00 00 00 00 00 9c d9	select(uid)
Ghost	08 b6 dd	MIFARE 1k
Reader	60 00 f5 7b	auth(block 0)
Ghost	6d c4 13 ab d0 73	n'_T
Reader	5e ef 51 1e 5e fb a6 21	$n_R \oplus \mathrm{ks}'_1, \mathrm{suc}^2(n'_T) \oplus \mathrm{ks}'_2$

Guessed structure for CRYPTO1

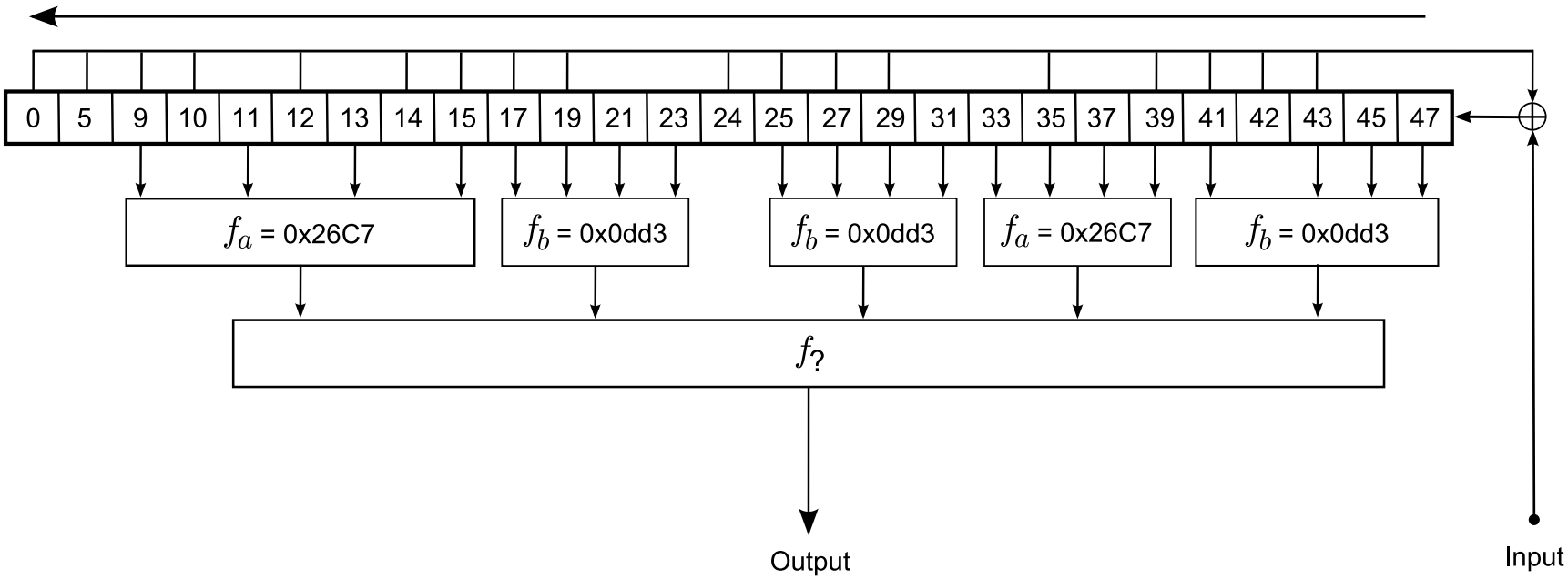


Recovering one component of the filter function

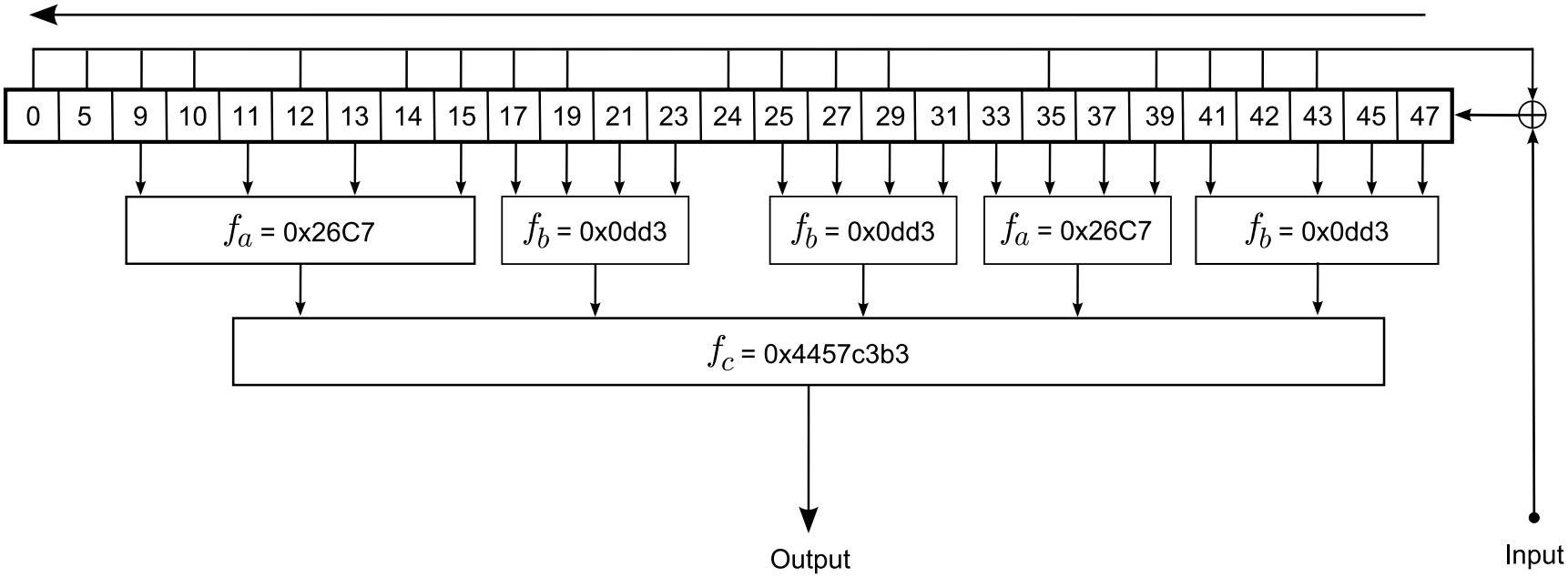
Example (First bit of encrypted reader nonce)

LFSR \ XX	55	54	51	50	45	44	41	40	15	14	11	10	05	04	01	00
0xb05d53bfdbXX	0	0	0	0	1	1	0	1	1	1	0	1	0	0	1	1
0xfbb57bbc7fXX	1	1	1	1	0	0	1	0	0	0	1	0	1	1	0	0
0xe2fd86e299XX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Guessed structure for CRYPTO1



The CRYPTO1 Cipher



Cryptanalysis of MIFARE Classic

Authentication Protocol with Timeout

	Ghost		Reader
0		anti-c(uid) $\longrightarrow$	
1		auth(block) $\longleftarrow$	
2	picks n_T		
3		n_T $\longrightarrow$	
4			$\text{ks}_1 \leftarrow \text{cipher}(K, \text{uid}, n_T)$
5			picks n_R
6			$\text{ks}_2, \text{ks}_3 \ldots \leftarrow \text{cipher}(K, \text{uid}, n_T, n_R)$
7		$n_R \oplus \text{ks}_1, \text{suc}^2(n_T) \oplus \text{ks}_2$ $\longleftarrow$	
8	wait for timeout		
9		halt $\oplus$ ks_3 $\longleftarrow$	

Authentication Protocol with Timeout

	Ghost		Reader
0		anti-c(uid) →	
1		auth(block) ←	
2	picks n_T		
3		n_T →	
4			$\text{ks}_1 \leftarrow \text{cipher}(K, \text{uid}, n_T)$
5			picks n_R
6			$\text{ks}_2, \text{ks}_3 \ldots \leftarrow \text{cipher}(K, \text{uid}, n_T, n_R)$
7		$n_R \oplus \text{ks}_1, \text{suc}^2(n_T) \oplus \text{ks}_2$ ←	
8	wait for timeout		
9		$\text{halt} \oplus \text{ks}_3$ ←	

It is possible to recover ks_2, ks_3!

Splitting the search space

Off-line table. 2^{36} entries.

LFSR State	$ks_2\ ks_3$
00 00 00 00 00 00	a0 91 5b 02 8f c5 a7 b5
⋮	⋮
00 0f ff ff ff ff	6f ea 4c af 0b fb 5c 5b

On-line table. 2^{12} entries.

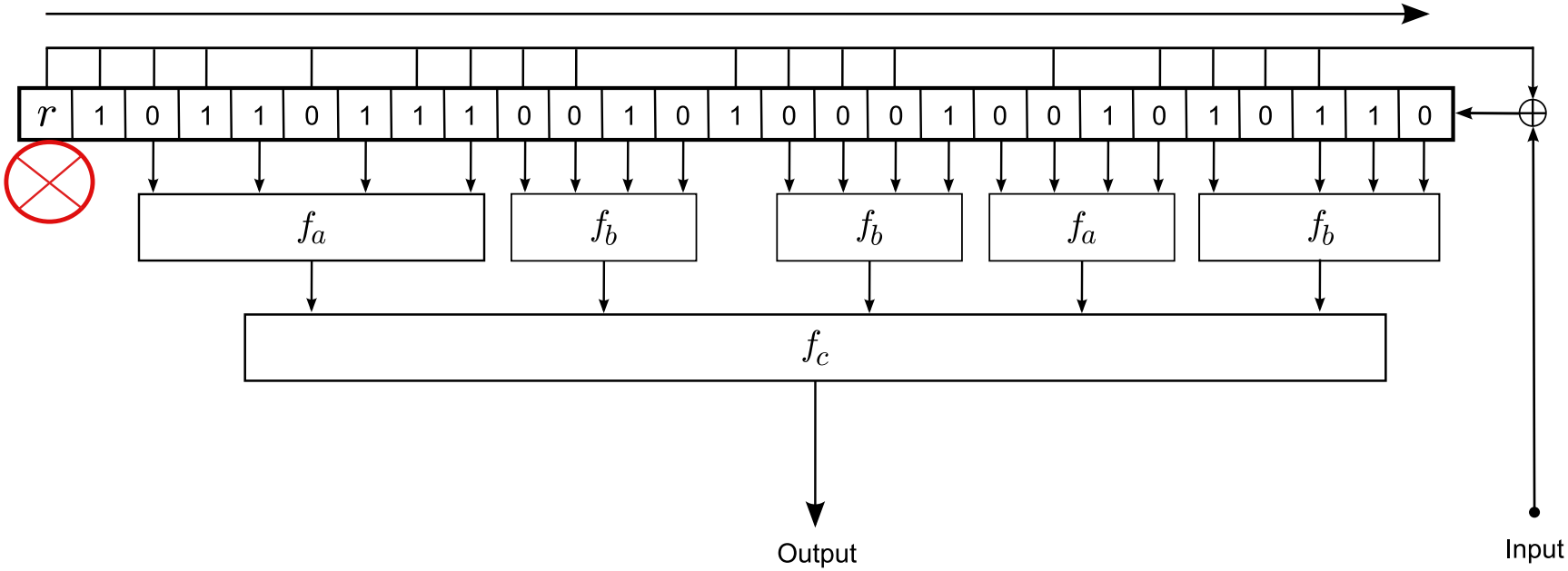
n_T	$ks_2\ ks_3$
00 00 00 00	d2 95 11 02 2f 5d a1 bb
⋮	⋮
00 00 ff f0	88 de 6b bf 3c 0a 22 5f

There is one n_T producing LFSR = YY YY YY YY 00 0Y

Authentication Protocol

	Tag		Reader
10		anti-c(uid) $\longrightarrow$	
11		$\longleftarrow$ auth(block)	
12	picks n_T		
13		n_T $\longrightarrow$	
14	$ks_1 \leftarrow \text{cipher}(K, \text{uid}, n_T)$		$ks_1 \leftarrow \text{cipher}(K, \text{uid}, n_T)$
15			picks n_R
16			$ks_2, ks_3 \ldots \leftarrow \text{cipher}(K, \text{uid}, n_T, n_R)$
17		$n_R \oplus ks_1$, Here! $\text{suc}^2(n_T) \oplus ks_2$ $\longleftarrow$	
18	$ks_2, ks_3 \ldots \leftarrow \text{cipher}(K, \text{uid}, n_T, n_R)$		
19		$\text{suc}^3(n_T) \oplus ks_3$ $\longrightarrow$	

Rolling back n_R



Recovering the secret key

Get back in time

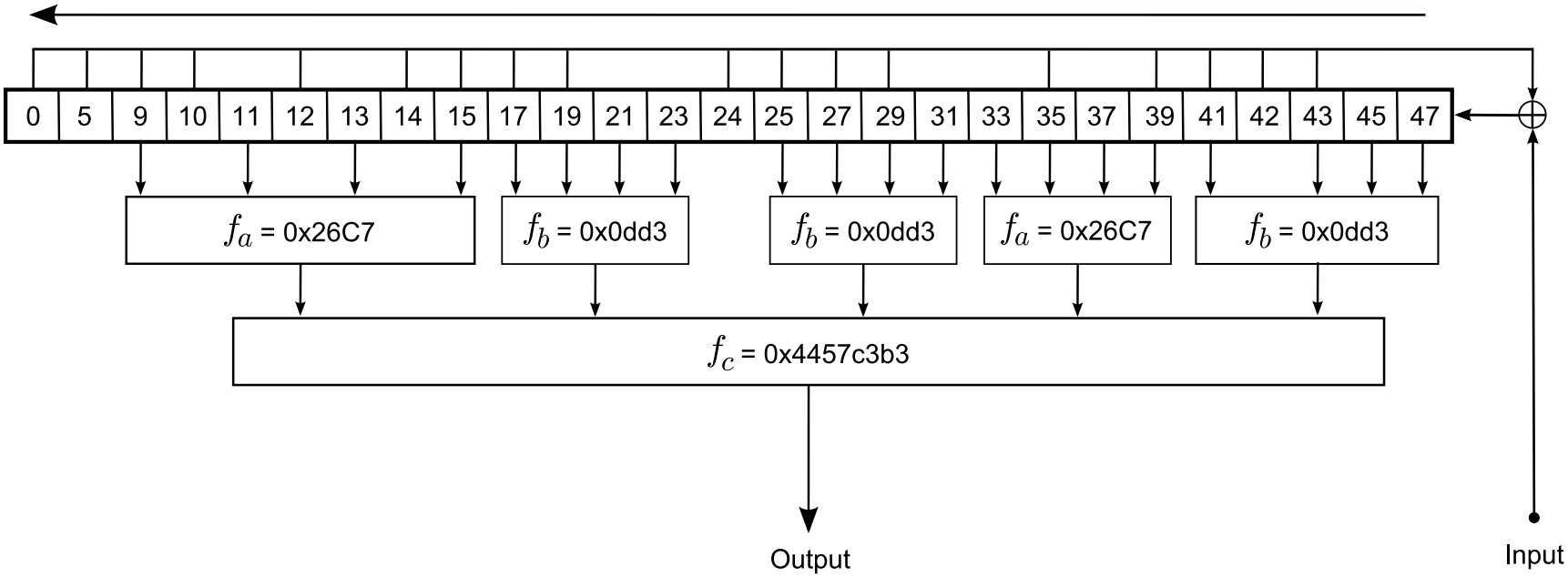
- Rollback n_R
- Rollback $n_T \oplus$ uid
- Recover the key!

Summary Attack 1

Typical attack times

- 4 to 8 hours pre-computation (this can be reused for any key).
- Gathering 4096 authentication sessions takes something between 2 and 14 minutes.
- Two minutes to recover the key.

The CRYPTO1 Cipher - Odd input bits

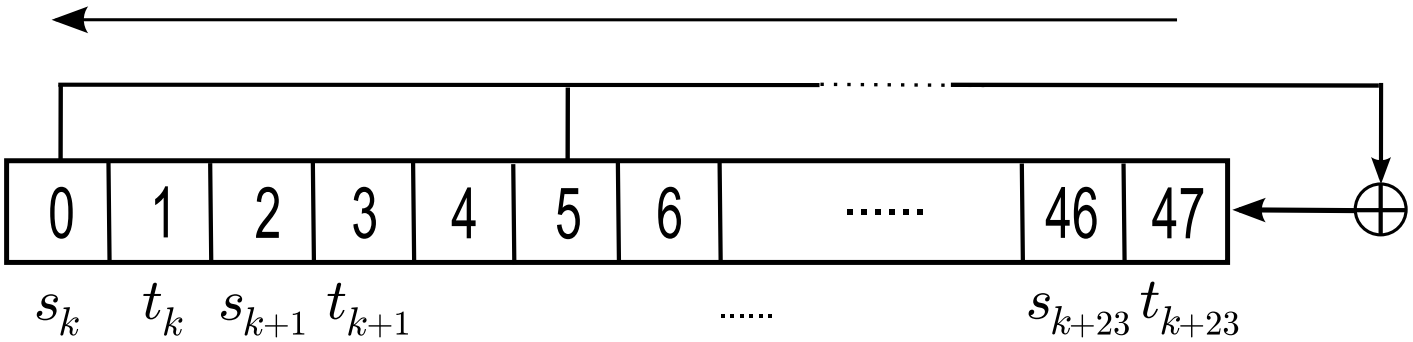


Attack 2

All the input bits to the filter function are on odd numbered bits.

- Let $b_0 b_1 \ldots b_{n-1}$ be n consecutive bits of keystream
- We build two tables of approximately 2^{19} elements.
- These tables contain the even and the odd numbered bits of the LFSR.
- Each table produce the evenly and oddly numbered bits of the required keystream.

Subsequences $\bar{s}$ and $\bar{t}$



Attack 2

$t_0, t_{k+1}, \ldots, t_{19}$ if $f(t_0, t_{k+1}, \ldots, t_{19}) = b_0$

Attack 2

$t_0, t_{k+1}, \ldots, t_{19}$ if $f(t_0, t_{k+1}, \ldots, t_{19}) = b_0$

We extend the odd table

$t_0, t_{k+1}, \ldots, t_{19}, 0$ if $f(t_1, t_{k+1}, \ldots, t_{19}, 0) = b_2$

$t_0, t_{k+1}, \ldots, t_{19}, 1$ if $f(t_1, t_{k+1}, \ldots, t_{19}, 1) = b_2$

~~$t_0, t_{k+1}, \ldots, t_{19}$~~ otherwise.

Attack 2

$s_0, s_{k+1}, \ldots, s_{19}$ if $f(s_0, s_{k+1}, \ldots, s_{19}) = b_1$

Attack 2

$s_0, s_{k+1}, \ldots, s_{19}$ if $f(s_0, s_{k+1}, \ldots, s_{19}) = b_1$

We extend the even table

$s_0, s_{k+1}, \ldots, s_{19}, 0$ if $f(s_1, s_{k+1}, \ldots, s_{19}, 0) = b_3$

$s_0, s_{k+1}, \ldots, s_{19}, 1$ if $f(s_1, s_{k+1}, \ldots, s_{19}, 1) = b_3$

~~$s_0, s_{k+1}, \ldots, s_{19}$~~ otherwise.

Attack 2

- We keep extending until we have sequences of 24 bits.
- We compute their (partial) contribution to the feedback at each stage (4 bits).
- We sort the tables on the newly computed feedback bits.
- We match two states entries and get a state $t_0 s_0 t_1 \ldots s_{23}$

Summary Attack 2

Requirements for the attack

- No pre-computation needed.
- Need only one partial authentication from a reader.
- Under 40 ms computation time to recover a secret key.
- Under 8MB of memory consumption.

Conclusions

- Cards can be cloned easily (within a second!).
- Only one trace is sufficient to clone.
- Only the reader is needed to get the secret key of a card.
- Security by obscurity is volatile.
- Do not develop your own crypto but use standards.