W00T'17

11th USENIX Workshop on Offensive Technologies

One Car, Two Frames: Attacks on Hitag-2 Remote Keyless Entry Systems Revisited

Ryad BENADJILA[1]

Mathieu RENARD[2]

José LOPES-ESTEVES[2]

Chaouki KASMI[2]

[1] Thales C&S, ryadbenadjila@gmail.com

[2] ANSSI, forename.name@ssi.gouv.fr

THALES
Communications & Security
S.A.S.

August 15, 2017

Vancouver, BC, Canada

Introduction

Car locking systems

- Used to open/close a car and for anti-theft immobilizers.

Car locking systems

- Used to open/close a car and for anti-theft immobilizers.

Mechanical

Smart-key

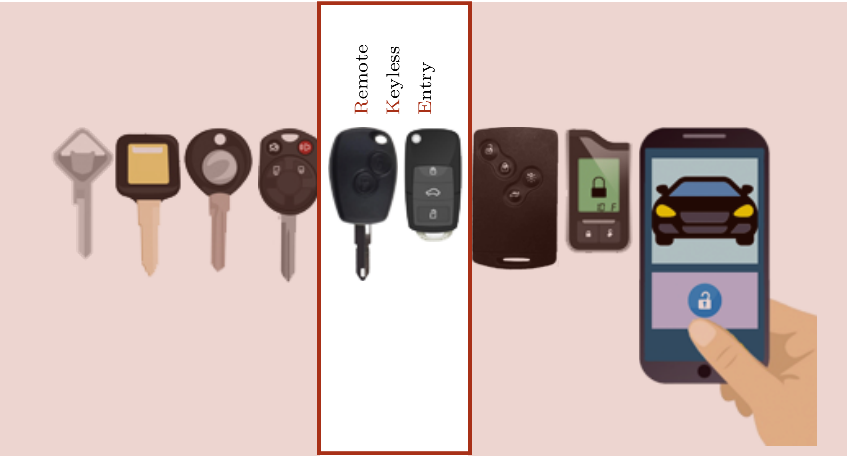
Car locking systems

- Used to open/close a car and for anti-theft immobilizers.

Mechanical
Smart-key

Car locking systems

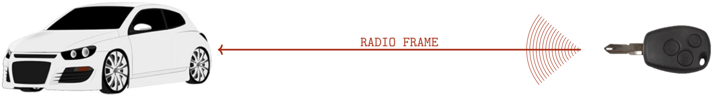
- This talk: focus on open/close Remote Keyless Entry systems.



Context

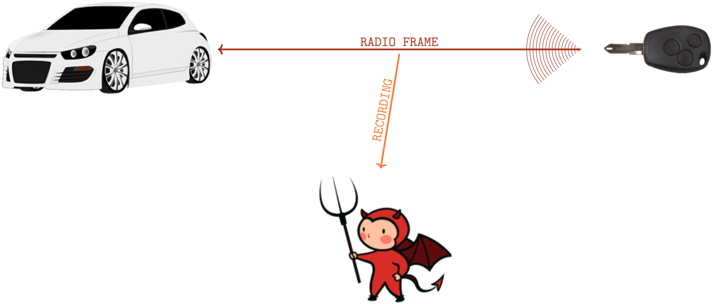
Remote Keyless Entry

- RKE:
  1. Monodirectional communication between remote key and ECU.



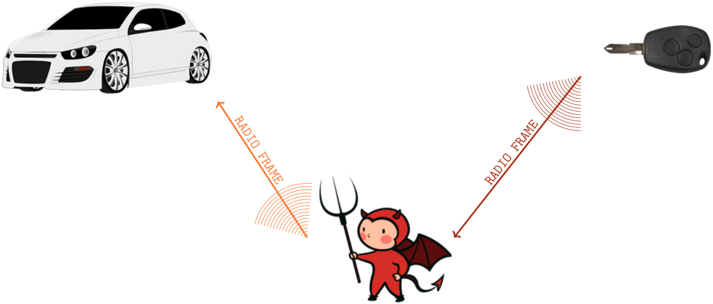
Remote Keyless Entry

- RKE:
  1. Monodirectional communication between remote key and ECU.
  2. Threats: recording,



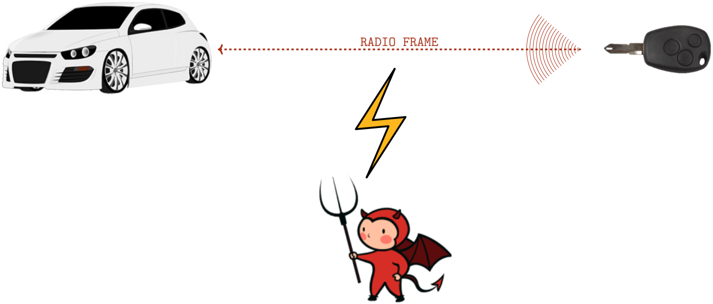
Remote Keyless Entry

- RKE:
  1. Monodirectional communication between remote key and ECU.
  2. Threats: recording, replaying,



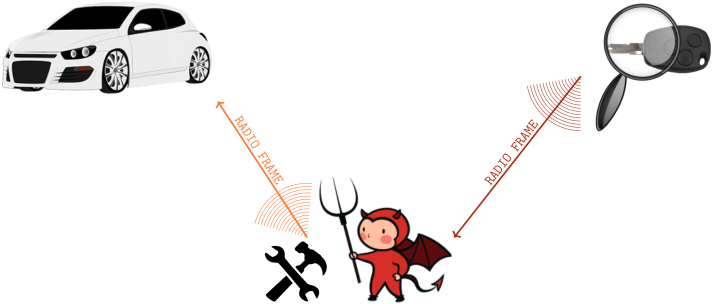
Remote Keyless Entry

- RKE:
  1. Monodirectional communication between remote key and ECU.
  2. Threats: recording, replaying, jamming,

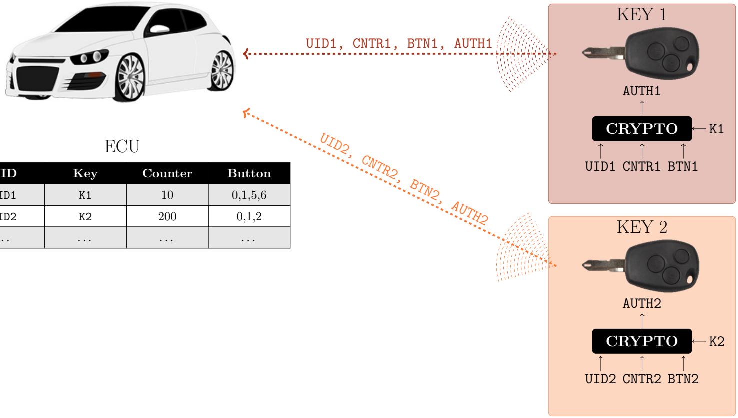


Remote Keyless Entry

- RKE:
  1. Monodirectional communication between remote key and ECU.
  2. Threats: recording, replaying, jamming, spoofing.



Remote Keyless Entry



UID1, CNTR1, BTN1, AUTH1

UID2, CNTR2, BTN2, AUTH2

ECU

UID	Key	Counter	Button
UID1	K1	10	0,1,5,6
UID2	K2	200	0,1,2
...	...	...	...

KEY 1

AUTH1

CRYPTO ← K1

UID1 CNTR1 BTN1

KEY 2

AUTH2

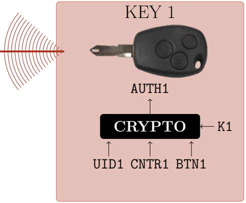
CRYPTO ← K2

UID2 CNTR2 BTN2

Remote Keyless Entry

UID1, CNTR1, BTN1, AUTH1

UID2, CNTR2, BTN2, AUTH2



KEY 1

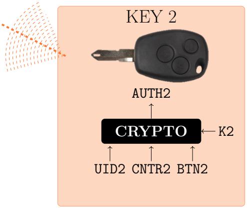
AUTH1

↑

CRYPTO ← K1

↑ ↑ ↑

UID1 CNTR1 BTN1



KEY 2

AUTH2

↑

CRYPTO ← K2

↑ ↑ ↑

UID2 CNTR2 BTN2

ECU

UID	Key	Counter	Button
UID1	K1	10	0,1,5,6
UID2	K2	200	0,1,2
...	...	...	...

BTN1 $\in \{0,1,5,6\}$?

CNTR1 $\in [10, 10+\delta]$?

UID1 CNTR1 BTN1

↓ ↓ ↓

K1 → CRYPTO

↓

AUTH_ECU =?= AUTH1

Remote Keyless Entry

UID1, CNTR1, BTN1, AUTH1

UID2, CNTR2, BTN2, AUTH2

ECU

UID	Key	Counter	Button
UID1	K1	11	0,1,5,6
UID2	K2	200	0,1,2
...	...	...	...

BTN1 $\in$ {0,1,5,6} ?

CNTR1 $\in [10, 10 + \delta]$?

UID1 CNTR1 BTN1

K1 → CRYPTO

AUTH_ECU =?= AUTH1

KEY 1

AUTH1

CRYPTO ← K1

UID1 CNTR1 BTN1

KEY 2

AUTH2

CRYPTO ← K2

UID2 CNTR2 BTN2

USENIX 2016: attacks on RKE systems

- USENIX 2016 article: "Lock It and Still Lose It - On the (In)Security of Automotive Remote Keyless Entry Systems".
- Two attacks are discussed:
  1. Volkswagen – good crypto but master keys are shared amongst all vehicles since 2000!
  2. PCF7946 – Philips/NXP transponder using Hitag-2. Correlation attack unveiled.

USENIX 2016: attacks on RKE systems

- Goal: setup the attack targeting the `PCF7946`.
  1. Capture and decode the radio frames.
  2. Implement the correlation attack.
  3. Find the secret key using the attack.
  4. Craft valid radio frames and profit.

- Constraints: **black-box** approach.
  - Breaking the car was not an option!
  - Neither invasive nor semi-invasive attacks on the `PCF7946` considered.
    - Time and resource costly!

Radio signal analysis

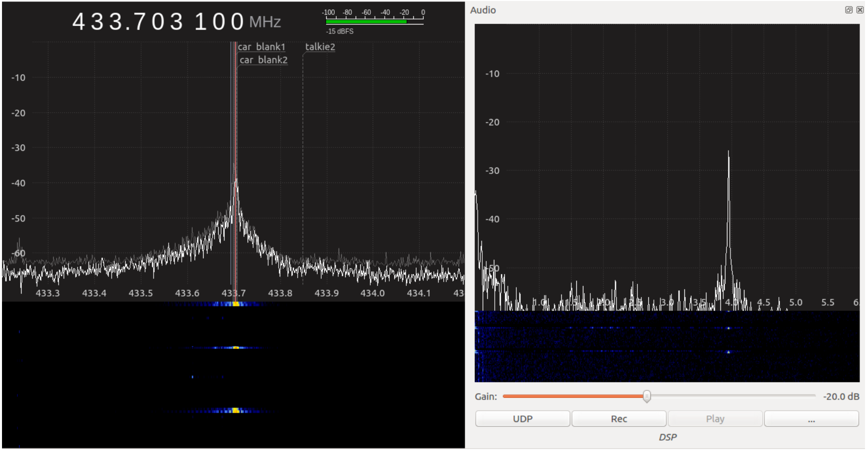
From RF signal to bits

- Useful information to be gathered:
  - Central frequency and channel bandwidth.
  - Modulation.
  - Channel encoding.
  - Packet format.

- **White-box** analysis:

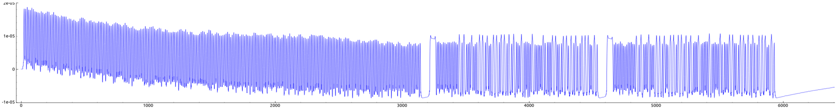
Parameter	Value
Working frequency	ISM 433 MHz
Modulation	ASK/FSK
Channel encoding	Manchester/NRZ
Packet format	see USENIX 2016

Demodulation: spectral analysis

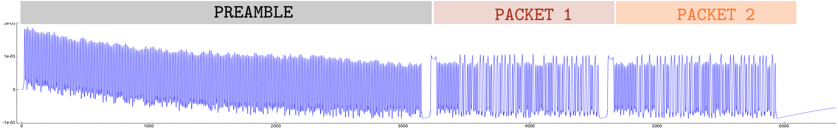


- Modulation: ASK (Amplitude Shift Keying).

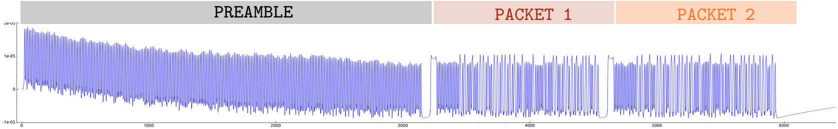
Decoding

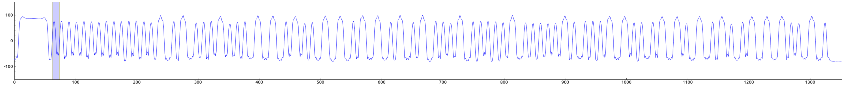


Decoding

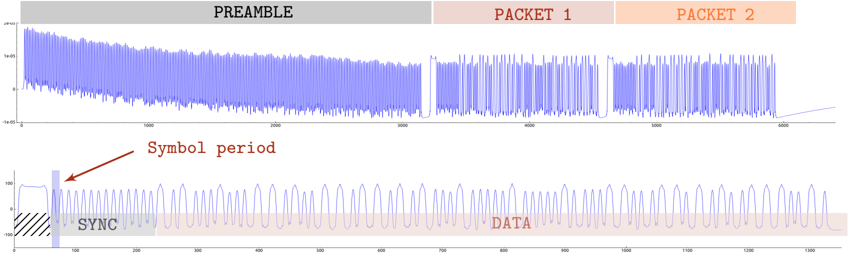

PREAMBLE
PACKET 1
PACKET 2
2e-05
1e-05
0
-1e-05
0
1000
2000
3000
4000
5000
6000

Decoding


PREAMBLE
PACKET 1
PACKET 2
2e-05
1e-05
0
-1e-05
0
1000
2000
3000
4000
5000
6000

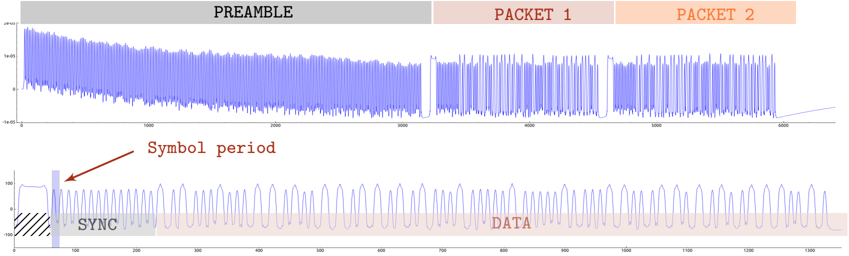

100
0
-100
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300

Decoding



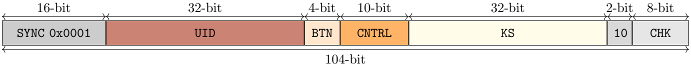
Symbol period

Decoding



Symbol period

- Results:
  - Modulation: ASK.
  - Channel encoding: Manchester.
  - Observing invariants to get back to the data.
  - Using the checksum for sanity check.



16-bit	32-bit	4-bit	10-bit	32-bit	2-bit	8-bit
SYNC 0x0001	UID	BTN	CNTRL	KS	10	CHK

104-bit

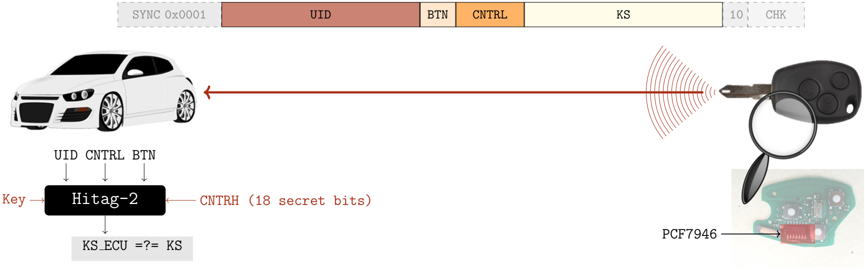
Hitag-2

The Hitag-2 algorithm

- Late 90's stream cipher from Philips (NXP).
- Hardware reverse engineered in 2007.

The Hitag-2 algorithm

- Late 90's stream cipher from Philips (NXP).
- Hardware reverse engineered in 2007.
- Using the algorithm in a RKE context:



SYNC 0x0001	UID	BTN	CNTRL	KS	10	CHK

UID CNTRL BTN

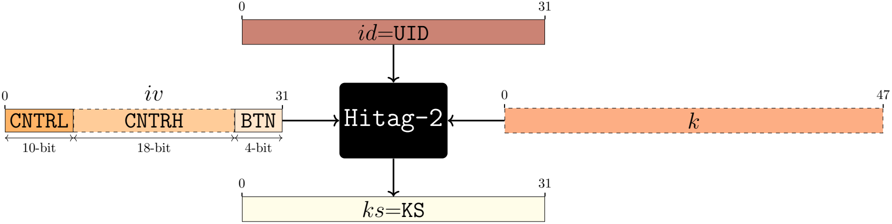
Key → Hitag-2 ← CNTRH (18 secret bits)

KS_ECU =?= KS

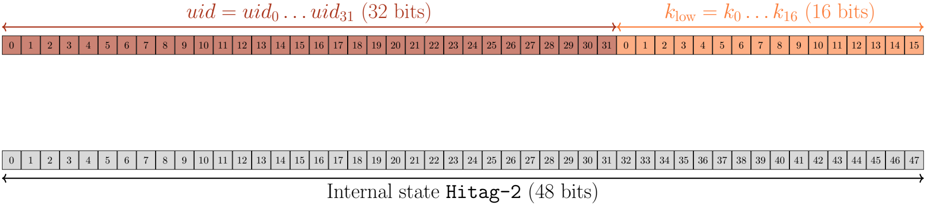
PCF7946

The Hitag-2 algorithm

- Late 90's stream cipher from Philips (NXP).
- Hardware reverse engineered in 2007.
- Using the algorithm in a RKE context:



Hitag-2: initialization phase



$uid = uid_0 \ldots uid_{31}$ (32 bits)

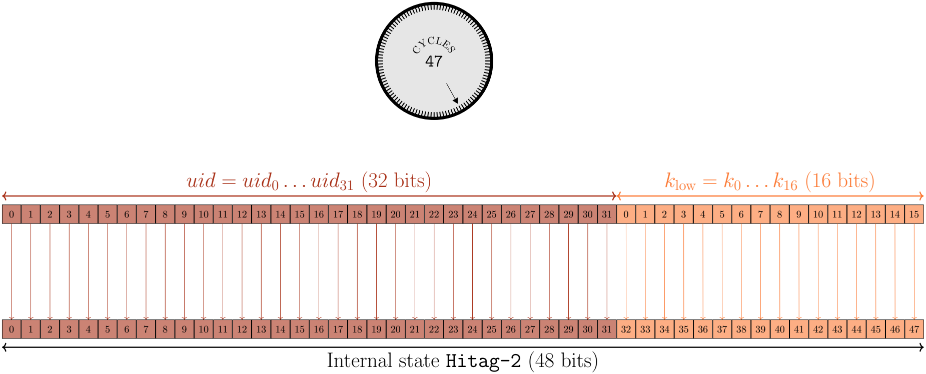
$k_{\text{low}} = k_0 \ldots k_{16}$ (16 bits)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

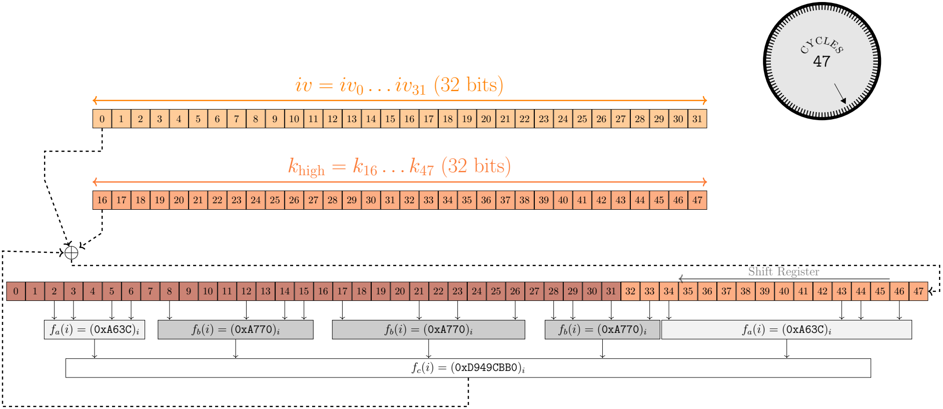
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

Internal state `Hitag-2` (48 bits)

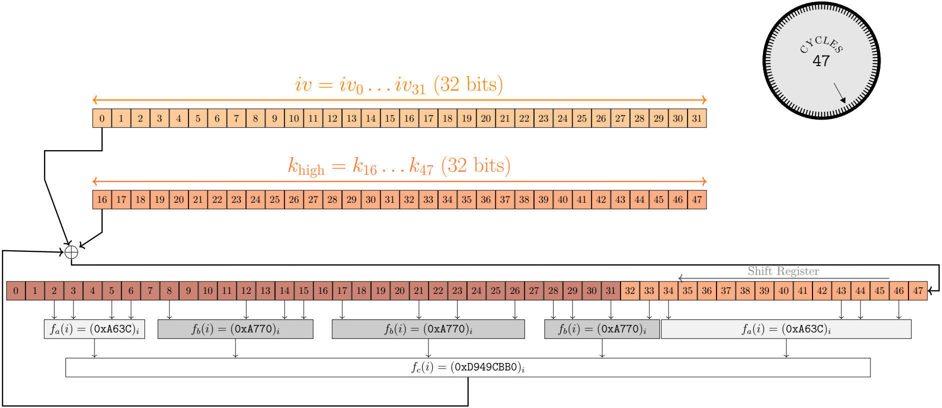
Hitag-2: initialization phase



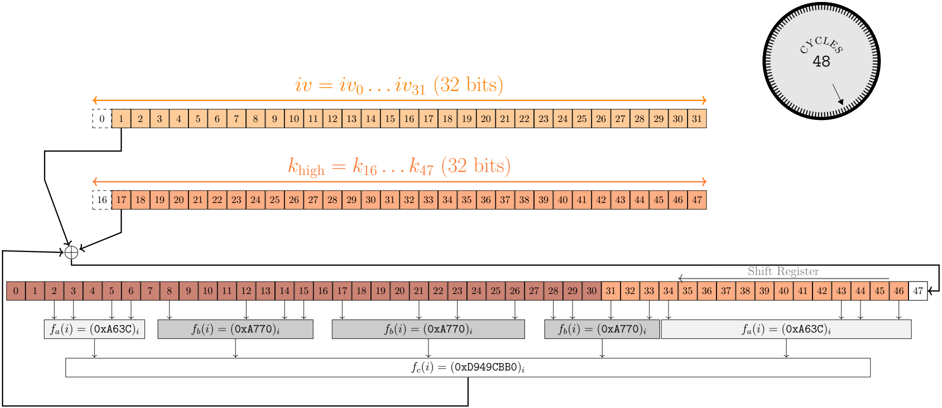
Hitag-2: randomization phase



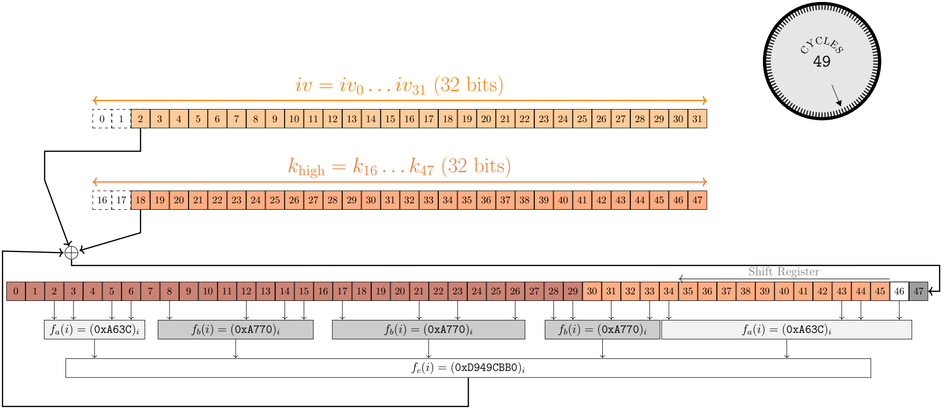
Hitag-2: randomization phase



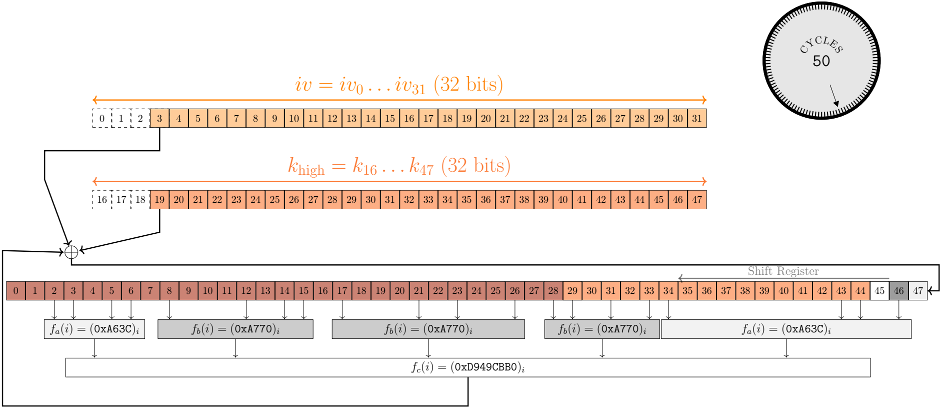
Hitag-2: randomization phase



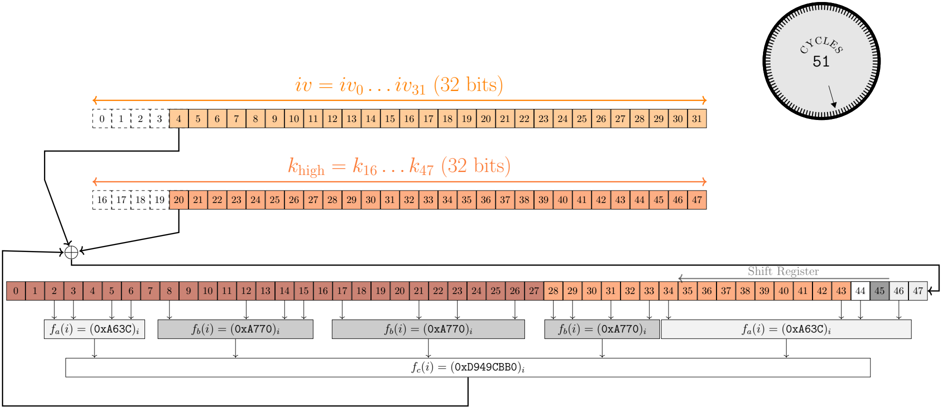
Hitag-2: randomization phase



Hitag-2: randomization phase



Hitag-2: randomization phase



CYCLES
51

$iv = iv_0 \ldots iv_{31}$ (32 bits)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

$k_{\text{high}} = k_{16} \ldots k_{47}$ (32 bits)

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

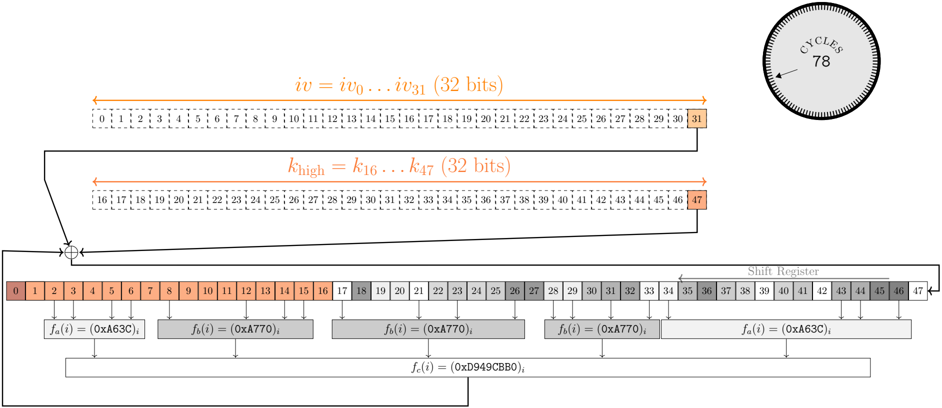
Shift Register

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

$f_a(i) = (\texttt{0xA63C})_i$ $f_b(i) = (\texttt{0xA770})_i$ $f_b(i) = (\texttt{0xA770})_i$ $f_b(i) = (\texttt{0xA770})_i$ $f_a(i) = (\texttt{0xA63C})_i$

$f_c(i) = (\texttt{0xD949CBB0})_i$

Hitag-2: randomization phase



$iv = iv_0 \dots iv_{31}$ (32 bits)

$k_{\text{high}} = k_{16} \dots k_{47}$ (32 bits)

$f_a(i) = (\texttt{0xA63C})_i$

$f_b(i) = (\texttt{0xA770})_i$

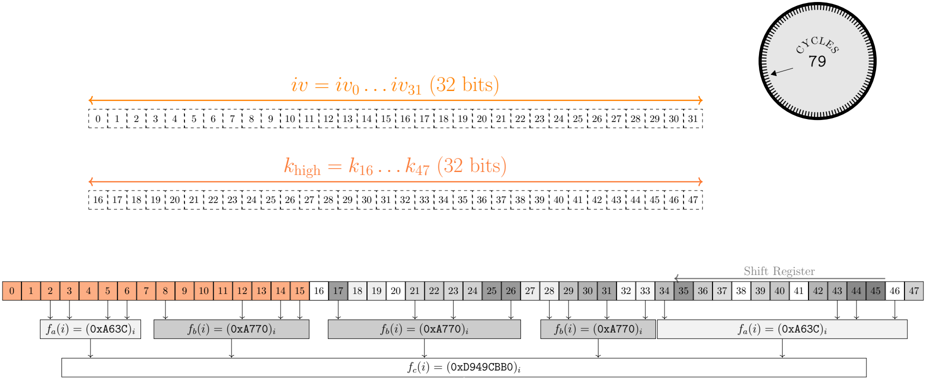
$f_b(i) = (\texttt{0xA770})_i$

$f_b(i) = (\texttt{0xA770})_i$

$f_a(i) = (\texttt{0xA63C})_i$

$f_c(i) = (\texttt{0xD949CBB0})_i$

Hitag-2: randomization phase



CYCLES
79

$iv = iv_0 \dots iv_{31}$ (32 bits)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

$k_{\text{high}} = k_{16} \dots k_{47}$ (32 bits)

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

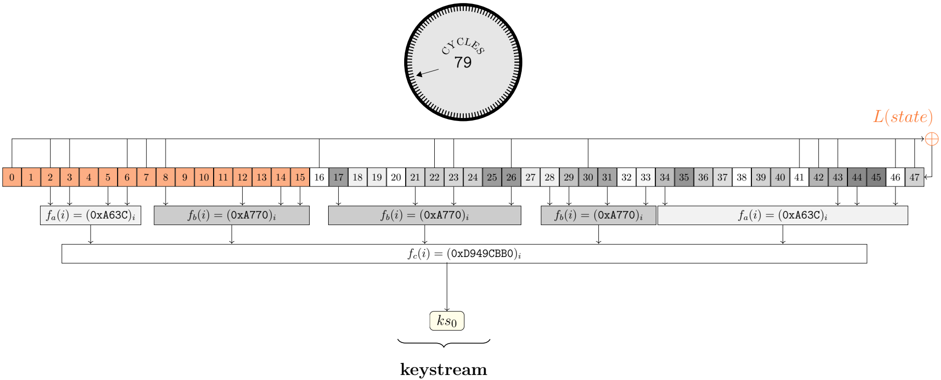
Shift Register

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

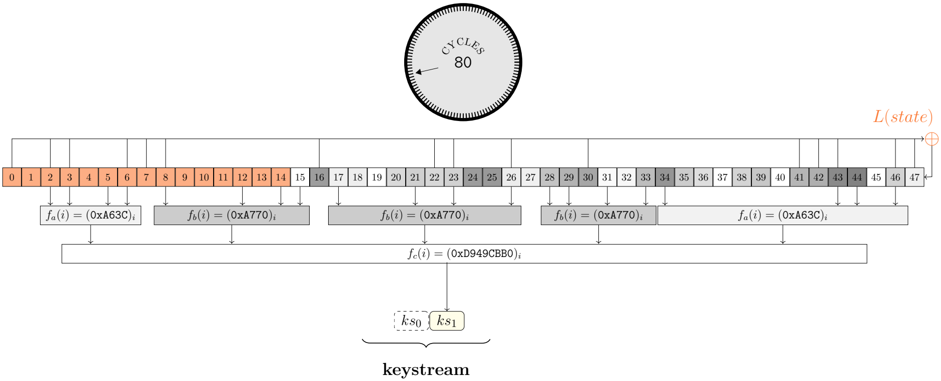
$f_a(i) = (\texttt{0xA63C})_i$ $f_b(i) = (\texttt{0xA770})_i$ $f_b(i) = (\texttt{0xA770})_i$ $f_b(i) = (\texttt{0xA770})_i$ $f_a(i) = (\texttt{0xA63C})_i$

$f_c(i) = (\texttt{0xD949CBB0})_i$

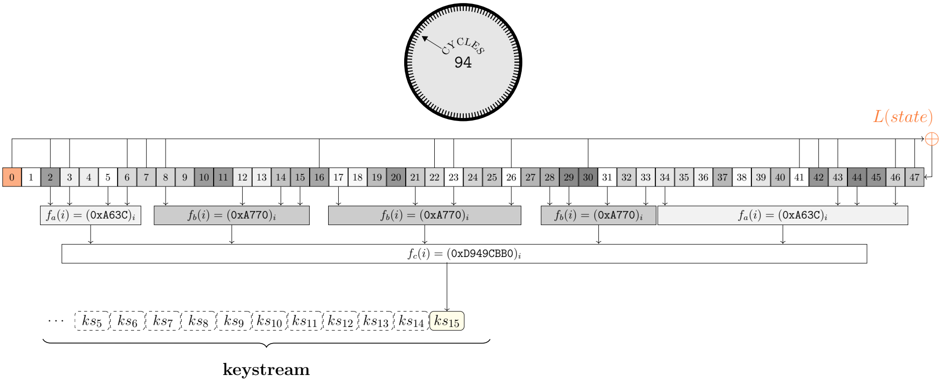
Hitag-2: nominal phase



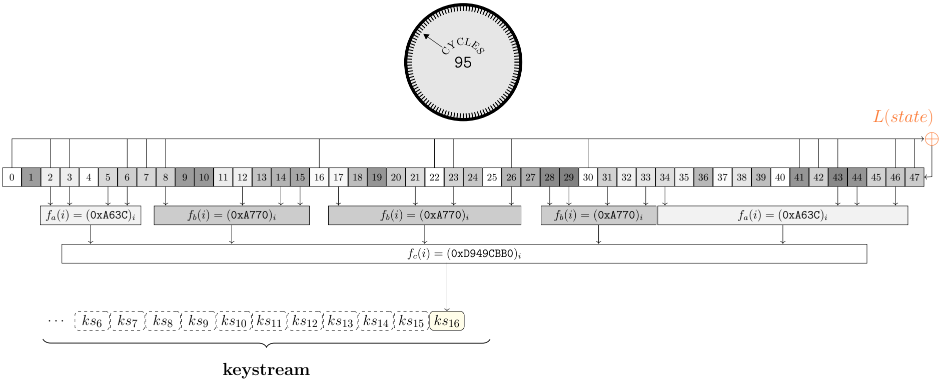
Hitag-2: nominal phase


CYCLES
80
$L(state)$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
$f_a(i) = (\texttt{0xA63C})_i$
$f_b(i) = (\texttt{0xA770})_i$
$f_b(i) = (\texttt{0xA770})_i$
$f_b(i) = (\texttt{0xA770})_i$
$f_a(i) = (\texttt{0xA63C})_i$
$f_c(i) = (\texttt{0xD949CBB0})_i$
ks_0
ks_1
keystream

Hitag-2: nominal phase



Hitag-2: nominal phase


CYCLES
95
L(state)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
$f_a(i) = (\texttt{0xA63C})_i$
$f_b(i) = (\texttt{0xA770})_i$
$f_b(i) = (\texttt{0xA770})_i$
$f_b(i) = (\texttt{0xA770})_i$
$f_a(i) = (\texttt{0xA63C})_i$
$f_c(i) = (\texttt{0xD949CBB0})_i$
$\cdots$ ks_6 ks_7 ks_8 ks_9 ks_{10} ks_{11} ks_{12} ks_{13} ks_{14} ks_{15} ks_{16}
keystream

Hitag-2: the correlation attack

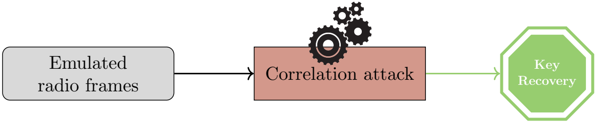
- Introduced by the USENIX 2016 article:
    - Key recovery with 4 to 8 radio frames.
    - The key search space is significantly reduced.
    - Uses key candidates scoring deduced from the observed keystream.

Hitag-2: the correlation attack

- Introduced by the USENIX 2016 article:
  - Key recovery with 4 to 8 radio frames.
  - The key search space is significantly reduced.
  - Uses key candidates scoring deduced from the observed keystream.
- Solving the unknown CNTRH issue:
  - Supposed to be set to zero at manufacturing time.
  - Authors suggest to estimate the vehicle's age.

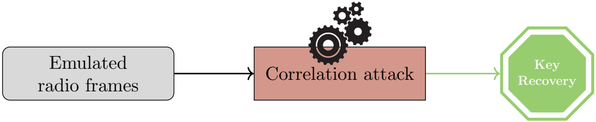
Implementing the correlation based cryptanalysis

- Tests on emulated radio frames.
  - Our implementation works.
  - The key is found in a few minutes.

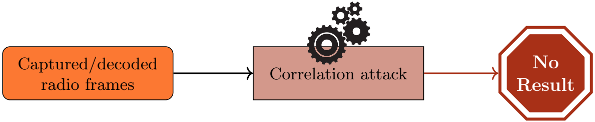


Implementing the correlation based cryptanalysis

- Tests on emulated radio frames.
  - Our implementation works.
  - The key is found in a few minutes.

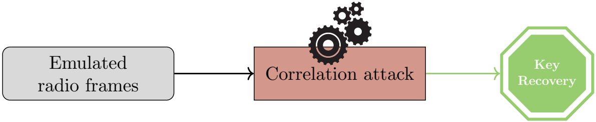


- Tests on real radio frames (with unknown CNTRH).
  - Cryptanalysis does not converge towards a proper key.

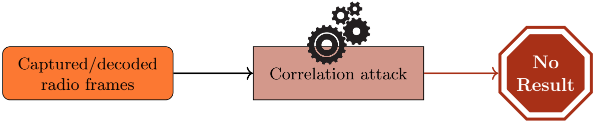


Implementing the correlation based cryptanalysis

- Tests on emulated radio frames.
  - Our implementation works.
  - The key is found in a few minutes.



- Tests on real radio frames (with unknown `CNTRH`).
  - Cryptanalysis does not converge towards a proper key.



- Our `Hitag-2` RKE system might be different!
  - We need to understand the discrepancies.

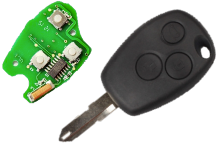
New attacks

Black box reverse engineering

- How can it be performed?
  - We had access to the vehicle but no access to the ECU.
  - No NDA with NXP: neither datasheets nor SDKs.

Black box reverse engineering

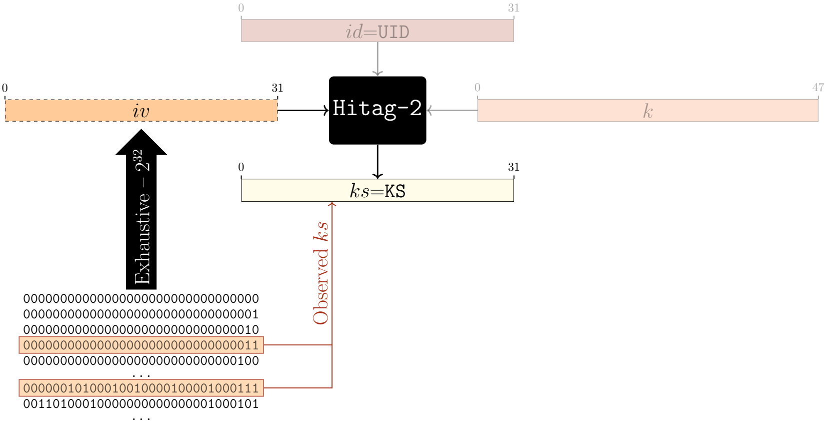
- How can it be performed?
  - We had access to the vehicle but no access to the ECU.
  - No NDA with NXP: neither datasheets nor SDKs.
- We found programmable blank keys containing the `PCF7946`!



  - They use the manufacturing default key `0x4f4e4d494b52`.

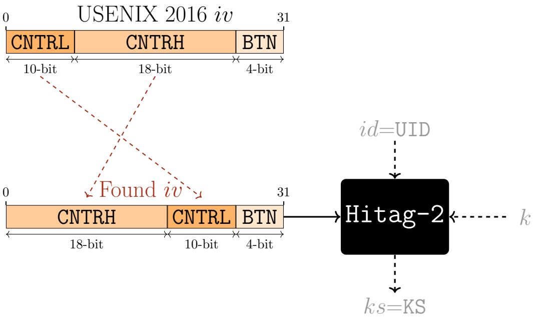
Finding the *iv* format: a black box approach

- Brute forcing the 2^{32} *iv* and finding explicit patterns for observed *ks*, with fixed and known *id* and *k*.



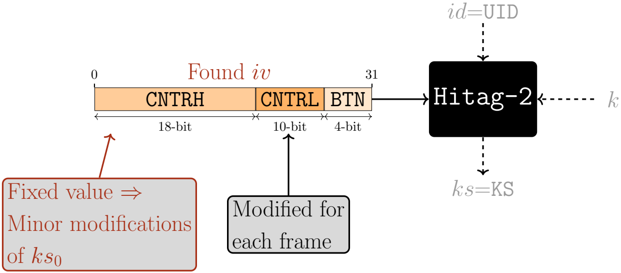
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
. . .
00000010100010010000100001000111
00110100010000000000000001000101
. . .

Finding the *iv* format: the discrepancies

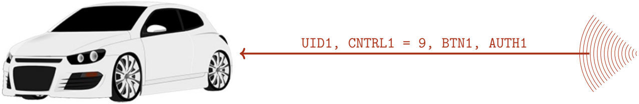


Finding the *iv* format: the discrepancies

- Explains why the USENIX 2016 correlation attack fails.



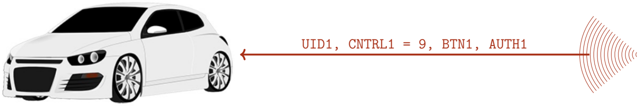
Uncovering an ECU mitigation



ECU

UID	Key	Counter	Button
UID1	K1	10	0,1,5,6
UID2	K2	200	0,1,2
...	...	...	...

Uncovering an ECU mitigation



UID1, CNTRL1 = 9, BTN1, AUTH1

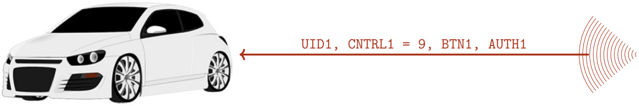
ECU

UID	Key	Counter	Button
UID1	K1	**10**	0,1,5,6
UID2	K2	200	0,1,2
...	...	...	...

CNTR1 = 9 $<$ 10

Uncovering an ECU mitigation

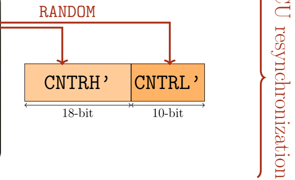
- Resynchronization with near-field 125 KHz when starting the engine.



ECU

UID	Key	Counter	Button
UID1	K1	10	0,1,5,6
UID2	K2	200	0,1,2
...	...	...	...

CNTR1 = 9 < 10



Optimized exhaustive search

- Uses two triplets (id, iv, ks):
  - Searches over 2^{48} keys the one realizing the observed keystreams.
  - Implementation of a heavily parallelized and optimized brute-forcer on CPU and GPU (in OpenCL).

Optimized exhaustive search

- Uses two triplets (id, iv, ks):
  - Searches over 2^{48} keys the one realizing the observed keystreams.
  - Implementation of a heavily parallelized and optimized brute-forcer on CPU and GPU (in OpenCL).
- Tested on Amazon EC2 instances:

Platform	Time
GeForce GTX 780Ti	18 hours
One Amazon EC2[†] instance	45 minutes
Three Amazon EC2[†] instances	15 minutes

[†] p2.16xlarge: 16 Tesla K80, 128 CPU

Optimized exhaustive search

- Uses two triplets (id, iv, ks):
  - Searches over 2^{48} keys the one realizing the observed keystreams.
  - Implementation of a heavily parallelized and optimized brute-forcer on CPU and GPU (in OpenCL).

- Tested on Amazon EC2 instances:

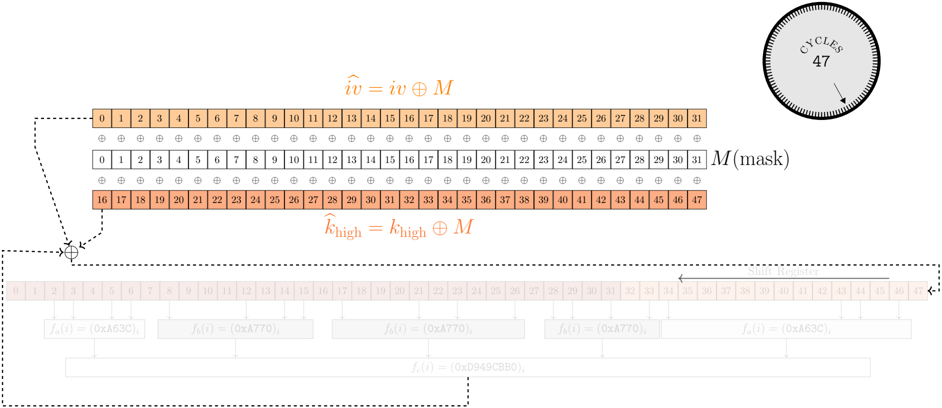
Platform	Time
GeForce GTX 780Ti	18 hours
One Amazon EC2† instance	45 minutes
Three Amazon EC2† instances	15 minutes

† p2.16xlarge: 16 Tesla K80, 128 CPU

- How to deal with the unknown part of CNTRH?

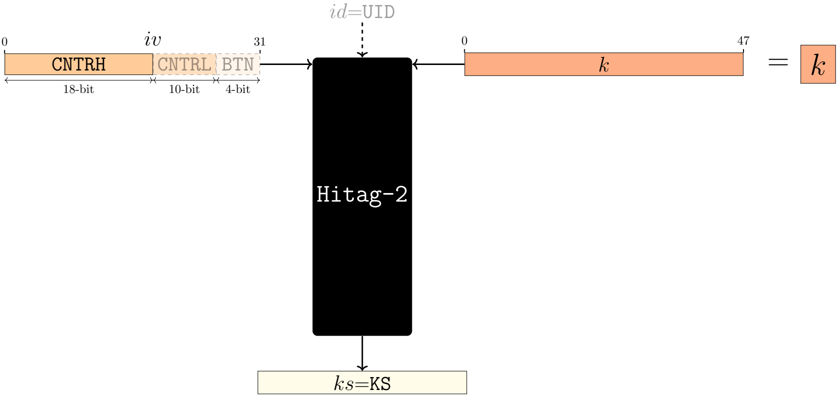
Hitag-2 equivalent keys

- Masking can be inserted during the randomization phase.



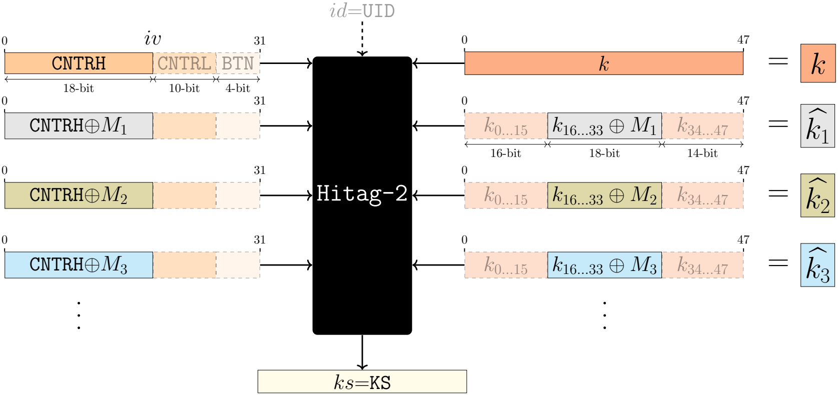
Hitag-2 equivalent keys

- Many equivalent keys generating the same keystream can be exposed through *iv* masking.



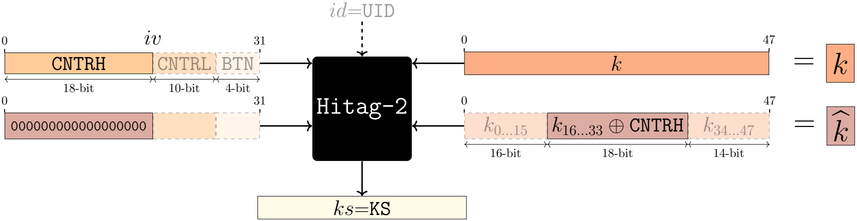
Hitag-2 equivalent keys

- Many equivalent keys generating the same keystream can be exposed through iv masking.



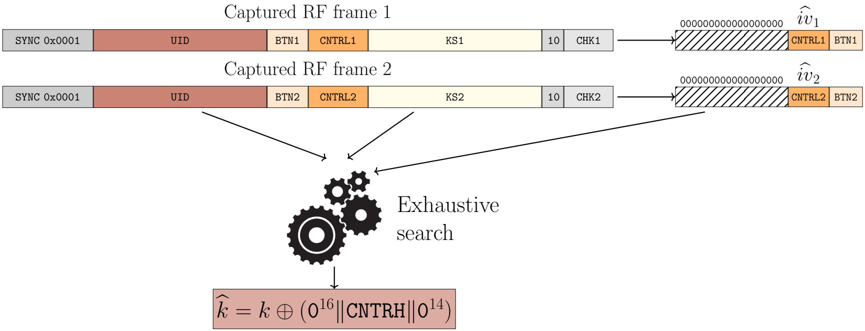
Hitag-2 equivalent keys

- Many equivalent keys generating the same keystream can be exposed through *iv* masking.
- Particular case of interest: when the mask is CNTRH.



Hitag-2 equivalent keys

- Many equivalent keys generating the same keystream can be exposed through *iv* masking.
- An exhaustive search with equivalent $\widehat{iv}$ produces an equivalent key $\widehat{k}$ masked with CNTRH.



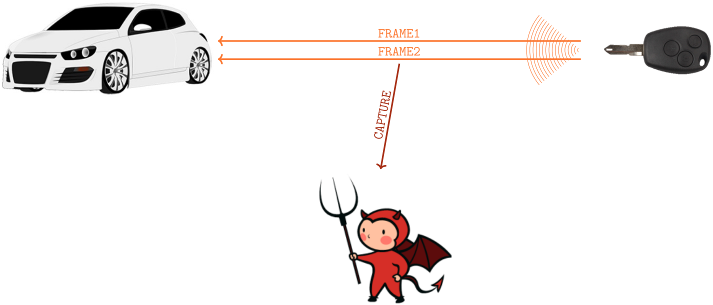
$$\widehat{k} = k \oplus (0^{16} \| \texttt{CNTRH} \| 0^{14})$$

Hitag-2 equivalent keys

- Many equivalent keys generating the same keystream can be exposed through iv masking.
- An exhaustive search with equivalent $\widehat{iv}$ produces an equivalent key $\widehat{k}$ masked with `CNTRH`.
- No need to find the real key k to craft legitimate frames!

New attacks 1/2: capture two frames and guess

- **Without** ECU resynchronization.



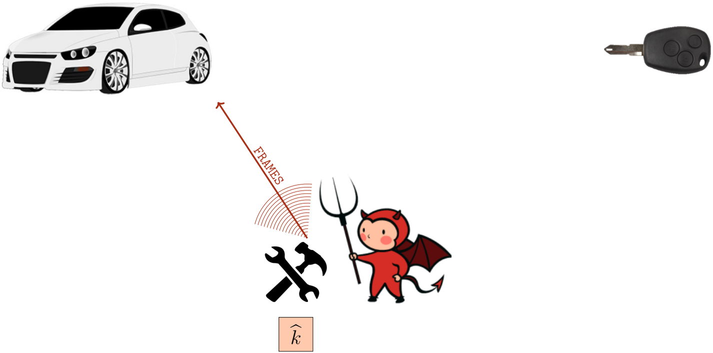
New attacks 1/2: capture two frames and guess

- **Without** ECU resynchronization.

$\widehat{k}$ = Exhaustive search of 2^{48}
$\Rightarrow$ 15 minutes on Amazon EC2

New attacks 1/2: capture two frames and guess

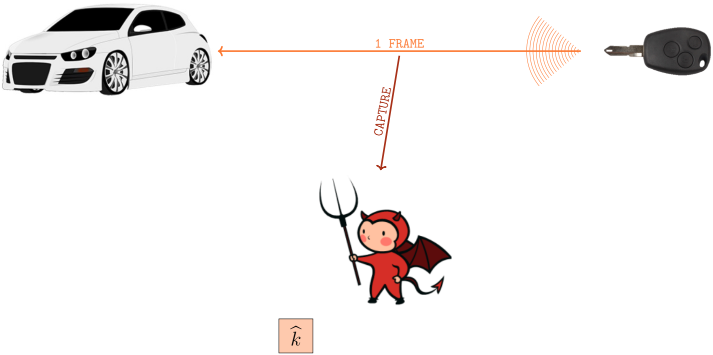
- **Without** ECU resynchronization.



$\leq (1024 - \texttt{CNTRL})$ frames OK
$> (1024 - \texttt{CNTRL}) \Rightarrow$ increment

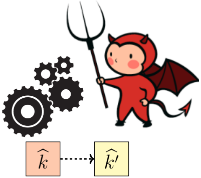
New attacks 2/2: recapture and adapt

- **With** ECU resynchronization.



New attacks 2/2: recapture and adapt

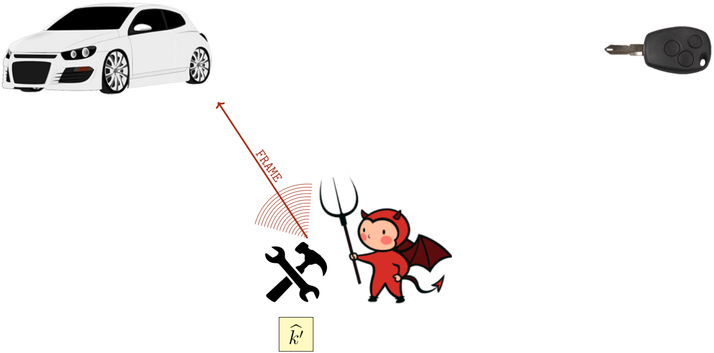
- **With** ECU resynchronization.



$\widehat{k'}$ = Exhaustive search of 2^{18}
⇒ 15 minutes on a laptop

New attacks 2/2: recapture and adapt

- **With** ECU resynchronization.



$\leq (1024 - \texttt{CNTRL})$ frames OK
$> (1024 - \texttt{CNTRL}) \Rightarrow$ increment

Conclusion

Conclusion

- Results:
    - A hardened RKE `Hitag-2` exposed.
        - Mitigation through ECU resynchronization.

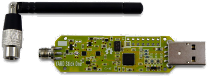
Conclusion

- Results:
  - A hardened RKE **Hitag-2** exposed.
    - Mitigation through ECU resynchronization.
  - Attack cost $\approx$ $\boxed{20} + \boxed{45} + \boxed{100}$ €.

RTL-SDR (capture)

Cloud brute force (find equivalent key)

YARD Stick One (forge frames)



  - Attack complexity = 2 RF frames, +1 with the ECU mitigation.

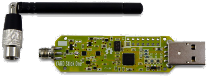
Conclusion

- Results:
  - A hardened RKE `Hitag-2` exposed.
    - Mitigation through ECU resynchronization.
  - Attack cost ≈ $\boxed{20} + \boxed{45} + \boxed{100}$ €.

RTL-SDR (capture)

Cloud brute force (find equivalent key)

YARD Stick One (forge frames)



  - Attack complexity = 2 RF frames, +1 with the ECU mitigation.
- Obsolete and proprietary cryptography is broken:
  - Time to make a change!

W00T'17

11th USENIX Workshop on Offensive Technologies

One Car, Two Frames: Attacks on `Hitag-2` Remote Keyless Entry Systems Revisited

?