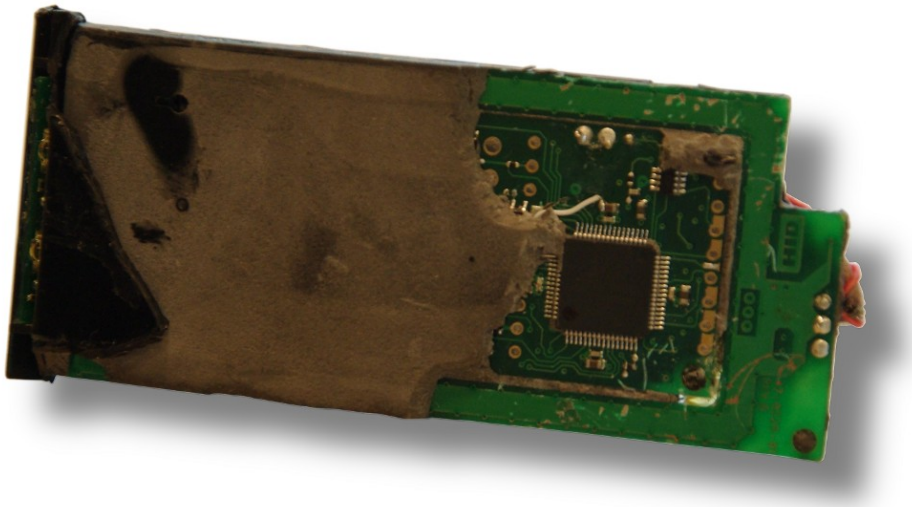
Usenix WOOT 2011

Flavio D. Garcia
Gerhard de Koning Gans
Roel Verdult

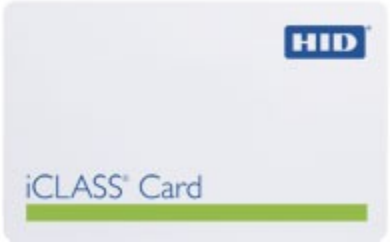
Exposing iClass Key Diversification



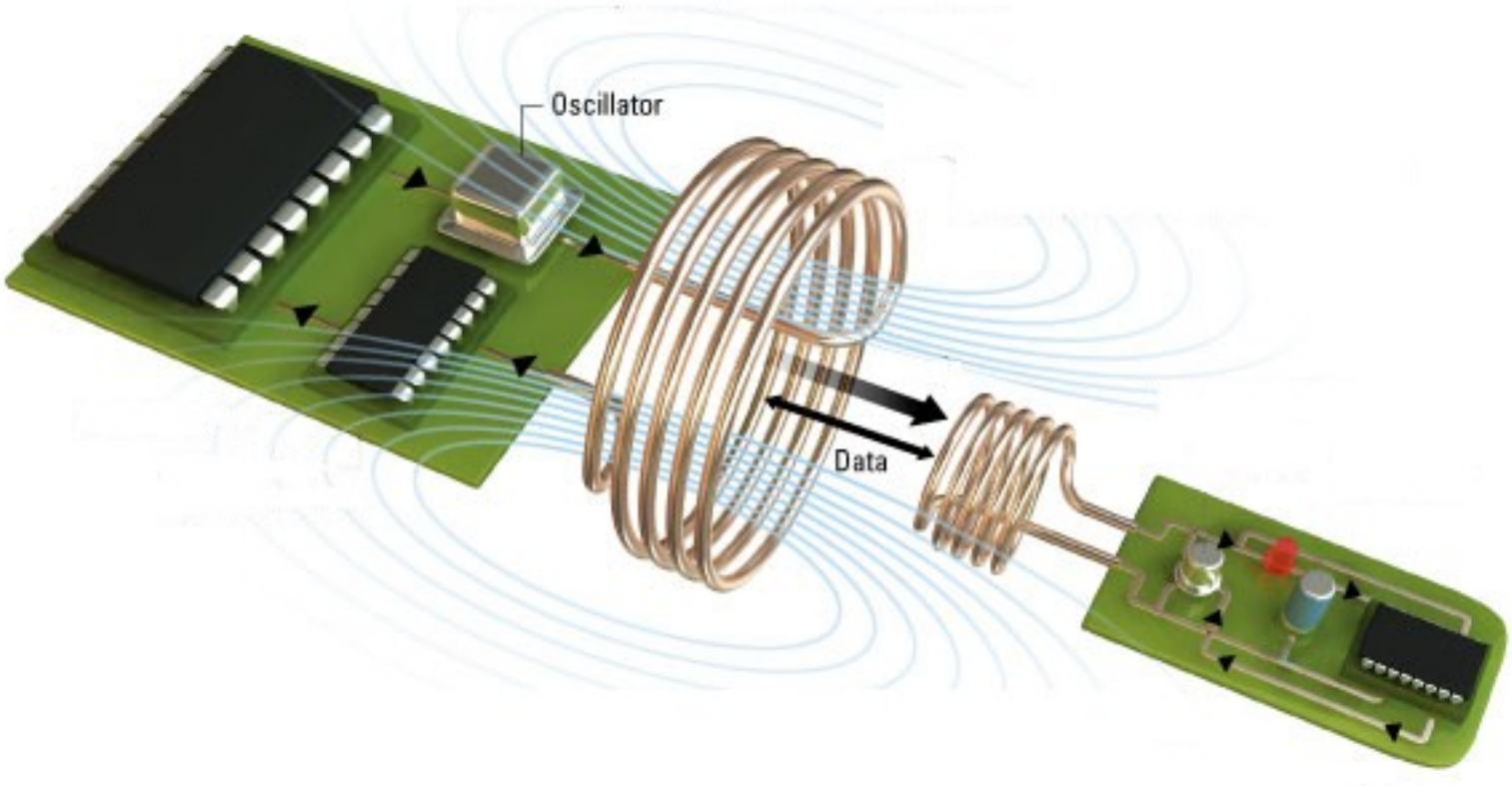
Radboud University Nijmegen

Contents

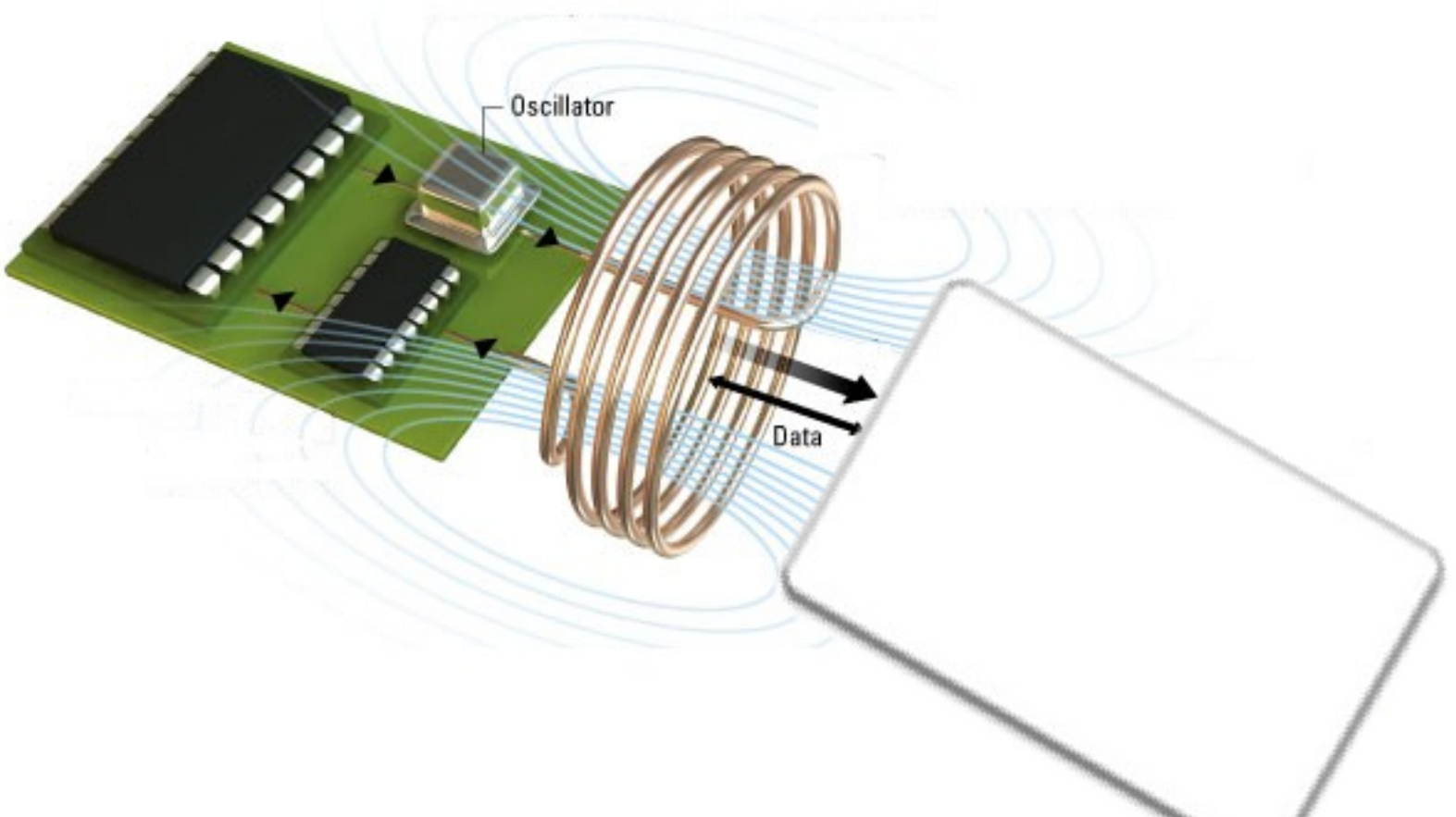
- Introduction
  - RFID
  - iClass and Picopass
  - Key Diversification
- iClass Key Diversification
  - DES and Fortify
  - Reader Control and Key Updates
  - Finding hash0 and $hash0^{-1}$
- Key Recovery Attack
- Conclusion



Radio Frequency Identification (RFID)



Radio Frequency Identification (RFID)



iClass and PicoPass

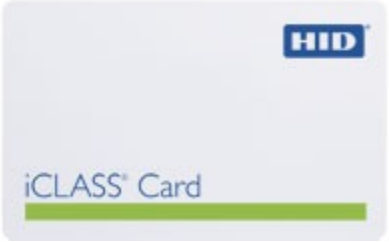
picopass®

HID®

iCLASS® by HID®

iClass (HID Global)

- ISO 15693 compatible smartcard
- Introduced in 2002 as replacement of HID prox
- Over 300 million cards sold (according to HID)



iClass (HID Global)

- Widely used in access control (examples from HID)
  - The Bank of America Merrill Lynch
  - Int. Airport of Mexico City
  - Navy base of Pearl Harbor
- Used as secure authentication
  - NaviGO (Dell Latitude and Precision)
  - e-Payment
  - Billing systems

iClass

- One master key for **every system**
- Built-in Key Diversification

Security by Obscurity?

- We know the examples of
  - Mifare Classic
  - KeeLoq
  - Hitag2
- How is the key diversification implemented?
- Important question since it is **built-in**!

Our Contribution

- Reverse engineering of built-in key diversification
  - Encryption of ID
  - 'Hashing' by **hash0**
- By-pass encryption mode of Omnikey Secure Mode
  - New library to communicate in Secure Mode
- Custom firmware for Proxmark3 (RFID Tool)
  - To eavesdrop ISO 15693 communication
- Released all of above (proxmark.org)
- We show that **hash0** can be inverted and give an attack to find the **master key!**

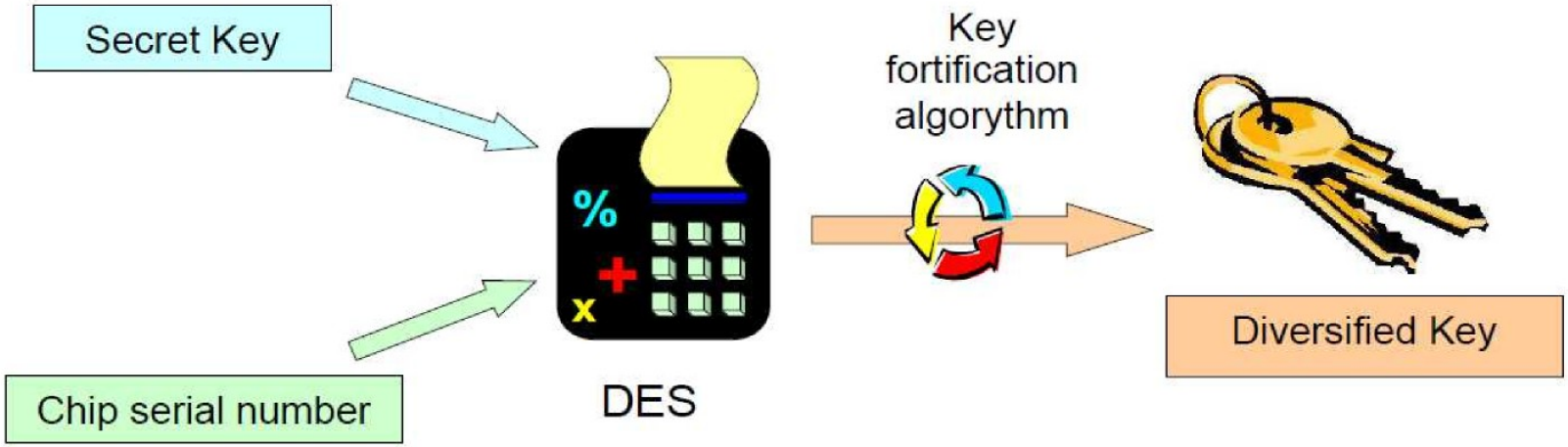
Key Diversification

Request ID

45

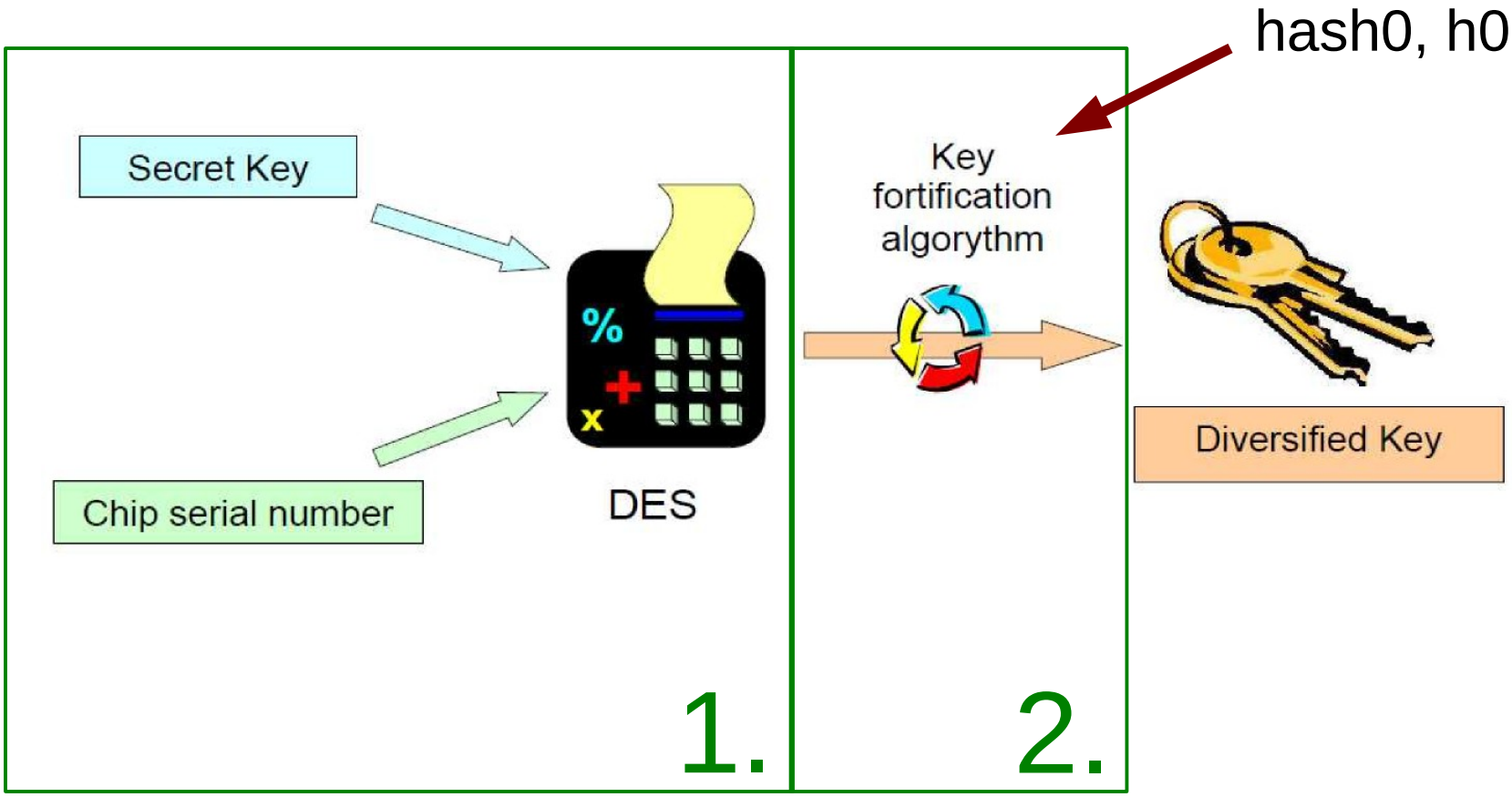
card key = diversify(MK,45)

iClass Key Diversification/Fortification



[Source: PicoPass Datasheets]

iClass Key Diversification/Fortification



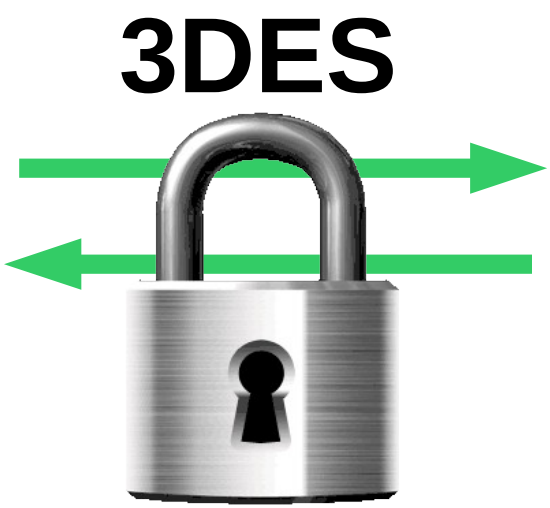
[Source: PicoPass Datasheets]

Omnikey (HID Global)

ISO 24727 requires encryption of USB connection

Omnikey *Secure Mode*

`iCLASSCardLib.dll`

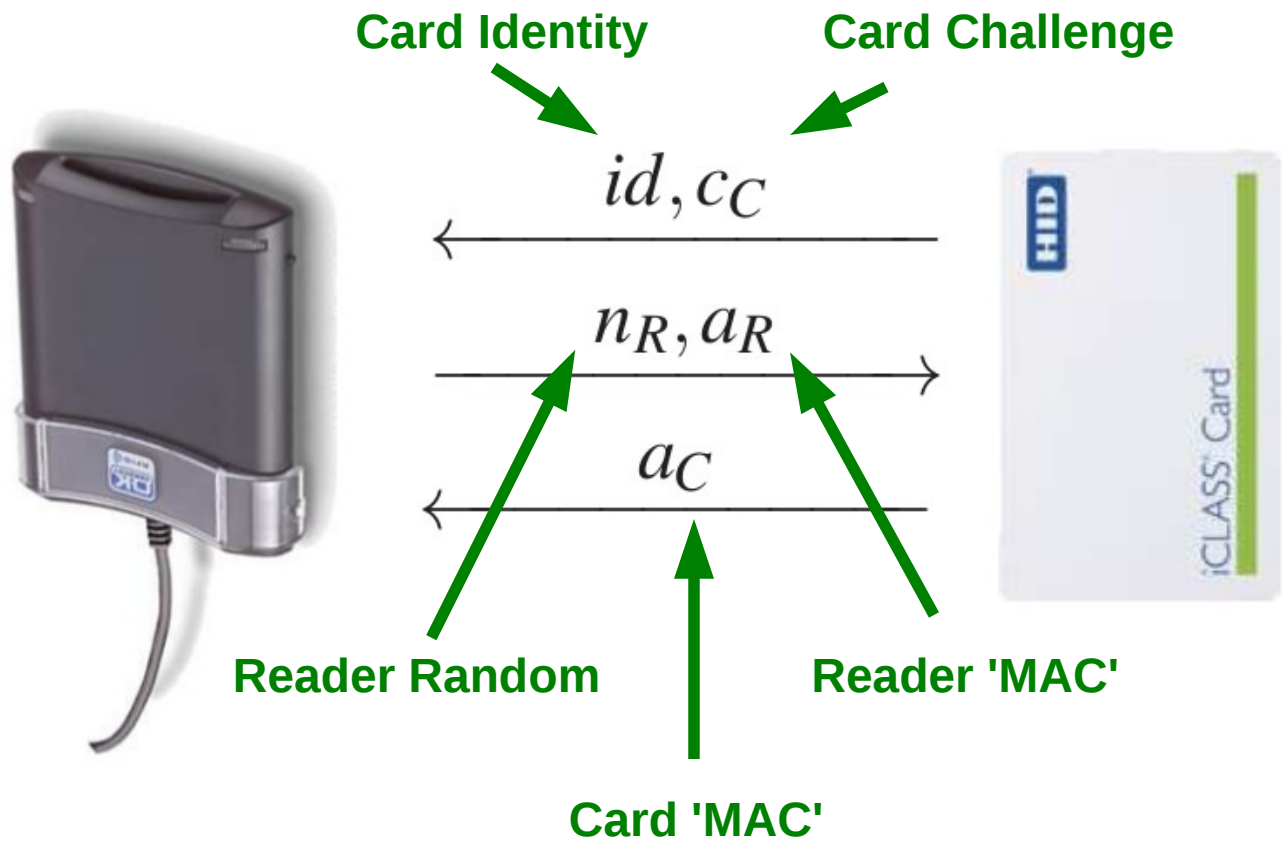


iClass Memory Layout

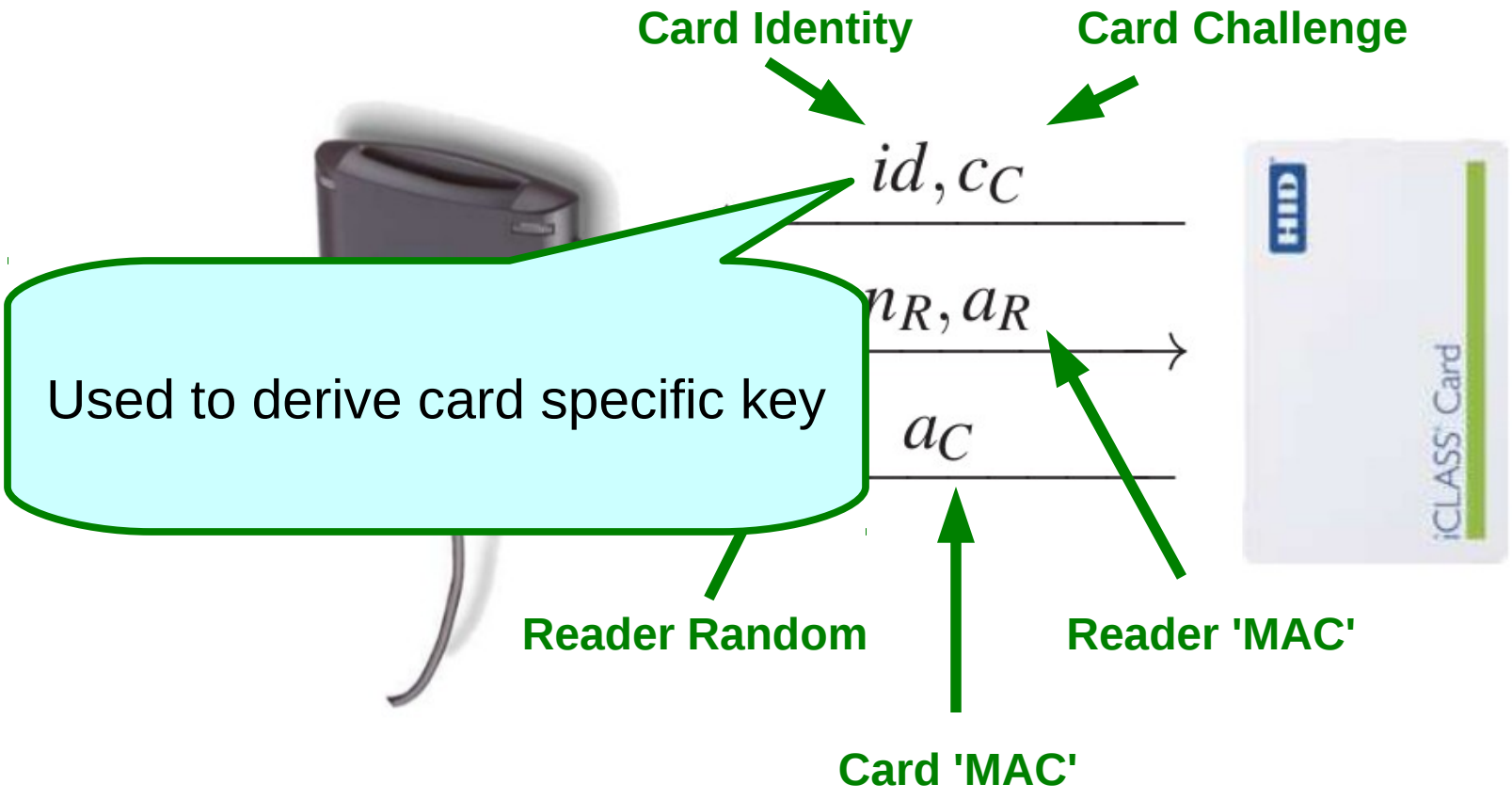
Key Slot	Value
00	
01	
02	
..	
..	

Block	Content	Denoted by
0	Card serial number	Identifier *id*
1	Configuration	
2	e-Purse	Card challenge c_C
3	Key for application 1	Debit key kd_{id}
4	Key for application 2	Credit key kc_{id}
5	Application issuer area	
6...18	Application 1	HID application a_{HID}
19...n	Application 2	$n = 16x - 1$ for xKS

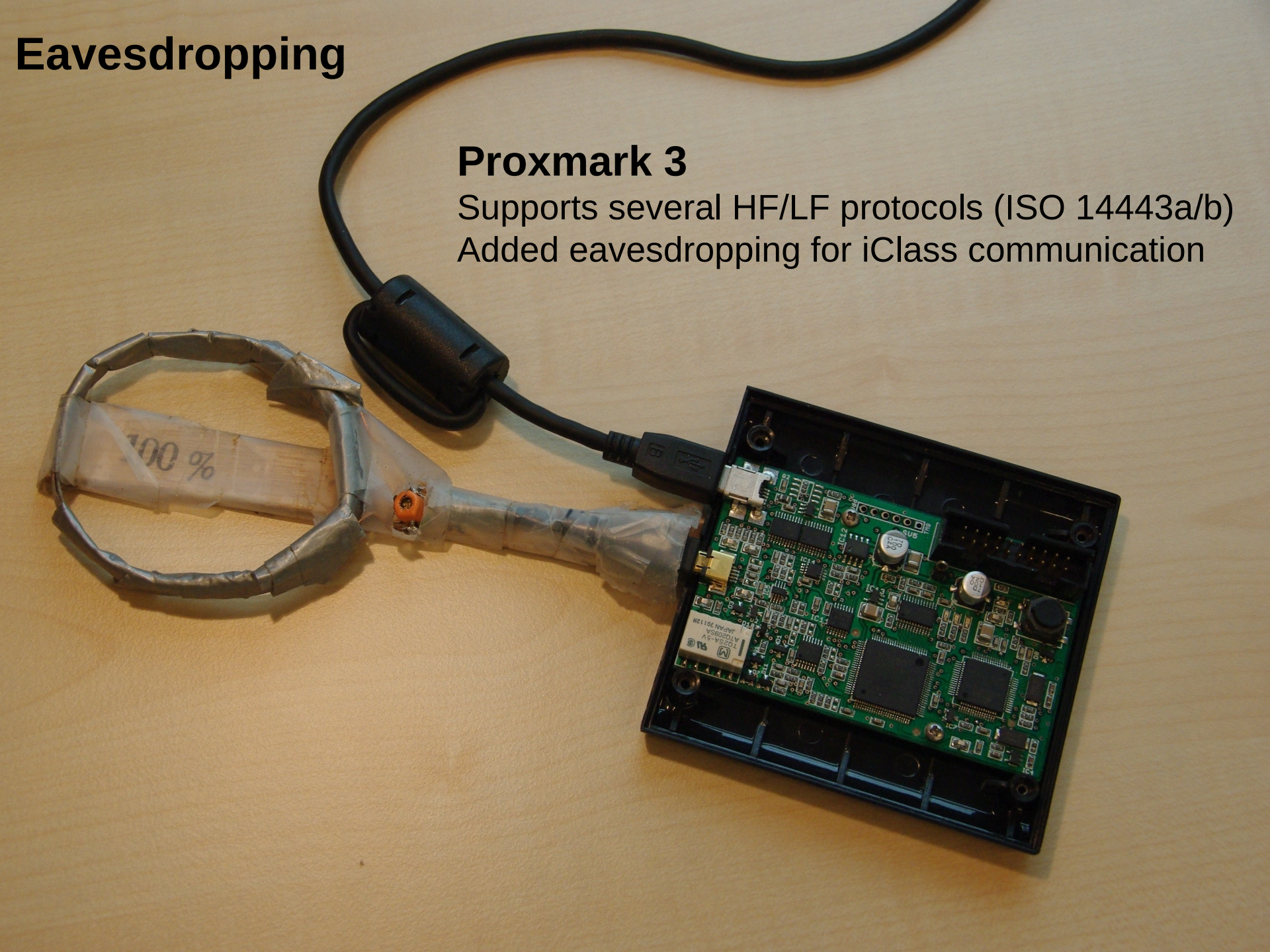
Authentication Protocol



Authentication Protocol



Eavesdropping

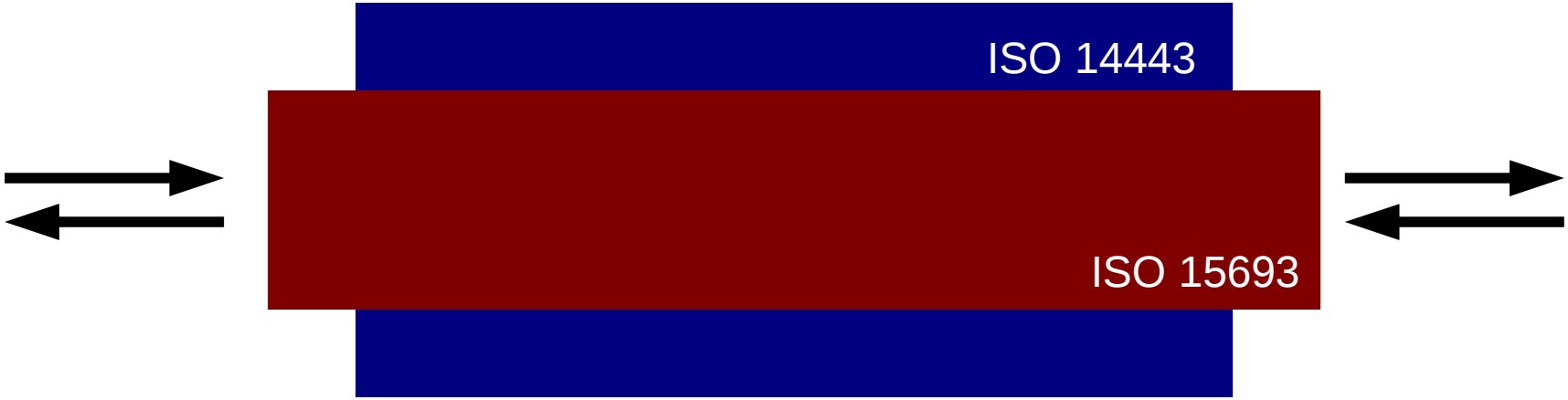


Proxmark 3

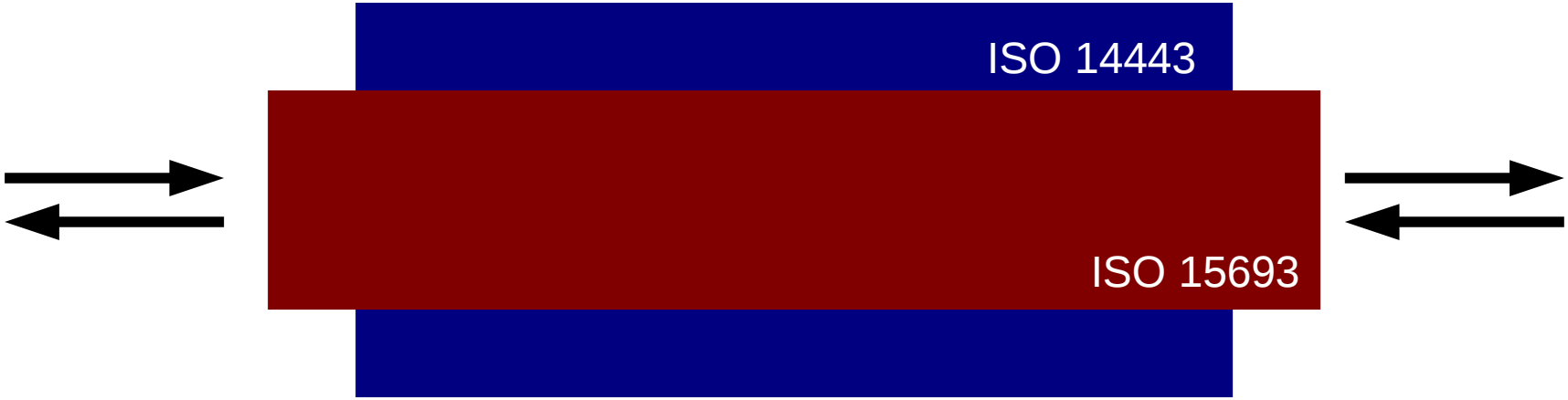
Supports several HF/LF protocols (ISO 14443a/b)

Added eavesdropping for iClass communication

Implementation side effect: “ISO Tunneling”



Implementation side effect: “ISO Tunneling”



Emulate iClass using existing software from **libnfc**

Card Key Update

Origin	Message	Description
Reader	0c 00 73 33	Read identifier
Tag	86 1d c1 00 f7 ff 12 e0	Card serial number *id*
Reader	0c 01 fa 22	Read configuration
Tag	12 ff ff ff 7f 1f ff 3c	iClass 2KS configuration
Reader	18 02	Authenticate with kc_{id}
Tag	fe ff ff ff ff ff ff ff	Card challenge c_C
Reader	05 00 00 c1 d9 7e 99 bb f4	Reader challenge (05, n_R, a_R)
Tag	46 3c 62 98	Response (a_C)
Reader	87 04 fc b4 32 3e 6a 86 56 26 8a b5 18 cc	Update kc_{id} (87 04, $kc'_{id} \oplus kc_{id}$, 8a b5 18 cc)
Tag	ff ff ff ff ff ff ff ff	Update succesful
Reader	0c 00 73 33	Read *id*
	...	
Reader	87 04 76 98 db 5d 01 78 0a 8f 67 25 c1 08	Update kc_{id} (87 04, $kc''_{id} \oplus kc'_{id}$, 67 25 c1 08)
	...	
Reader	87 04 8a 2c e9 63 6b fe 5c a9 e2 a5 bc 55	Update kc_{id} (87 04, $kc_{id} \oplus kc''_{id}$, e2 a5 bc 55)

Card Key Update

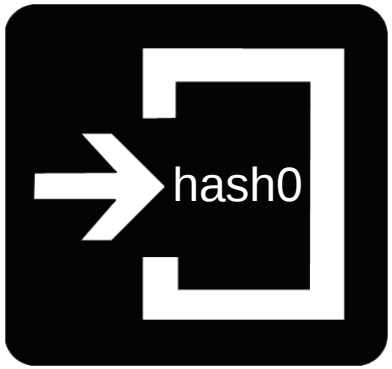
Origin	Message	
Reader	0c 00 73 33	
Tag	86 1d c1 00 f7 ff 12 e0	
Reader	0c 01 fa 22	
Tag	12 ff ff ff 7f 1f ff 3c	
Reader	18 02	
Tag	fe ff ff ff ff ff ff ff	
Reader	05 00 00 c1 d9 7e 99 bb f4	
Tag	46 3c 62 98	
Reader	87 04 fc b4 32 3e 6a 86 56 26 8a b5 18 cc	Update kc_{id} (87 04, $kc'_{id} \oplus kc_{id}$, 8a b5 18 cc)
Tag	ff ff ff ff ff ff ff ff	Update succesful
Reader	0c 00 73 33	Read *id*
	...	
Reader	87 04 76 98 db 5d 01 78 0a 8f 67 25 c1 08	Update kc_{id} (87 04, $kc''_{id} \oplus kc'_{id}$, 67 25 c1 08)
	...	
Reader	87 04 8a 2c e9 63 6b fe 5c a9 e2 a5 bc 55	Update kc_{id} (87 04, $kc_{id} \oplus kc''_{id}$, e2 a5 bc 55)

```
fcb4323e6a865626
7698db5d01780a8f  ⊕
----------------
8a2ce9636bfe5ca9
```

XOR Difference of Card Keys is send over the air

Determine Input of hash0

$$DES_{enc}(id, MK) \; ?$$



Pick any 64-bit string c and compute with two different keys (k and k'):

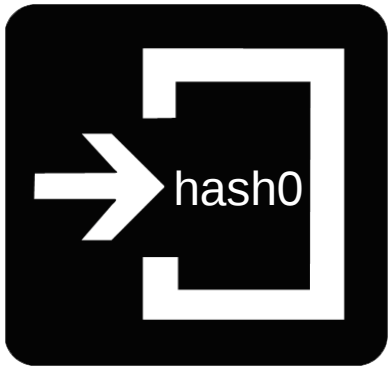
$$DES_{dec}(c,k) = p$$
$$DES_{dec}(c,k') = p'$$

Determine Input of hash0

$DES_{enc}(p,k)$

$DES_{enc}(p,k)$

c

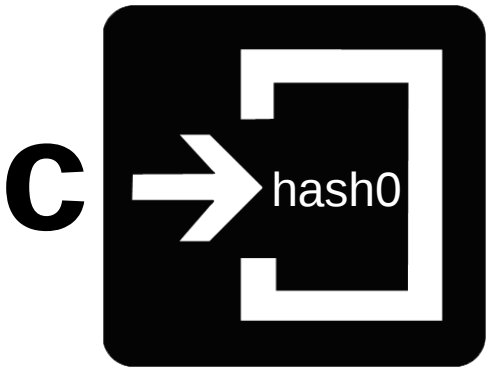


Same XOR difference!

Determine Input of hash0

$DES_{enc}(p,k)$

$DES_{enc}(p,k)$



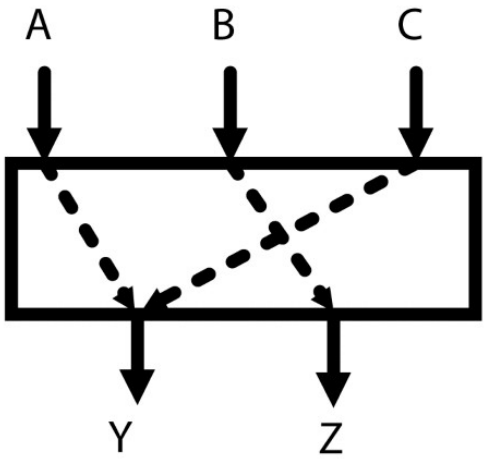
Same XOR difference!

Card key = hash0($DES_{enc}(id,kc)$)

Recovering hash0

- XOR Difference
- Learn Input/Output Relations
- Step-by-step Recovery of Partial Input/Outputs
- Reconstruct hash0

Input/Output Relations



h0(0000000000000001) = 0606000000000000

h0(0000000000000002) = 0400040000000000

h0(0000000100000000) = 0000000000080000

h0(0000000200000000) = 0000000000100000

h0(8000000000000000) = 0306050c07060d00

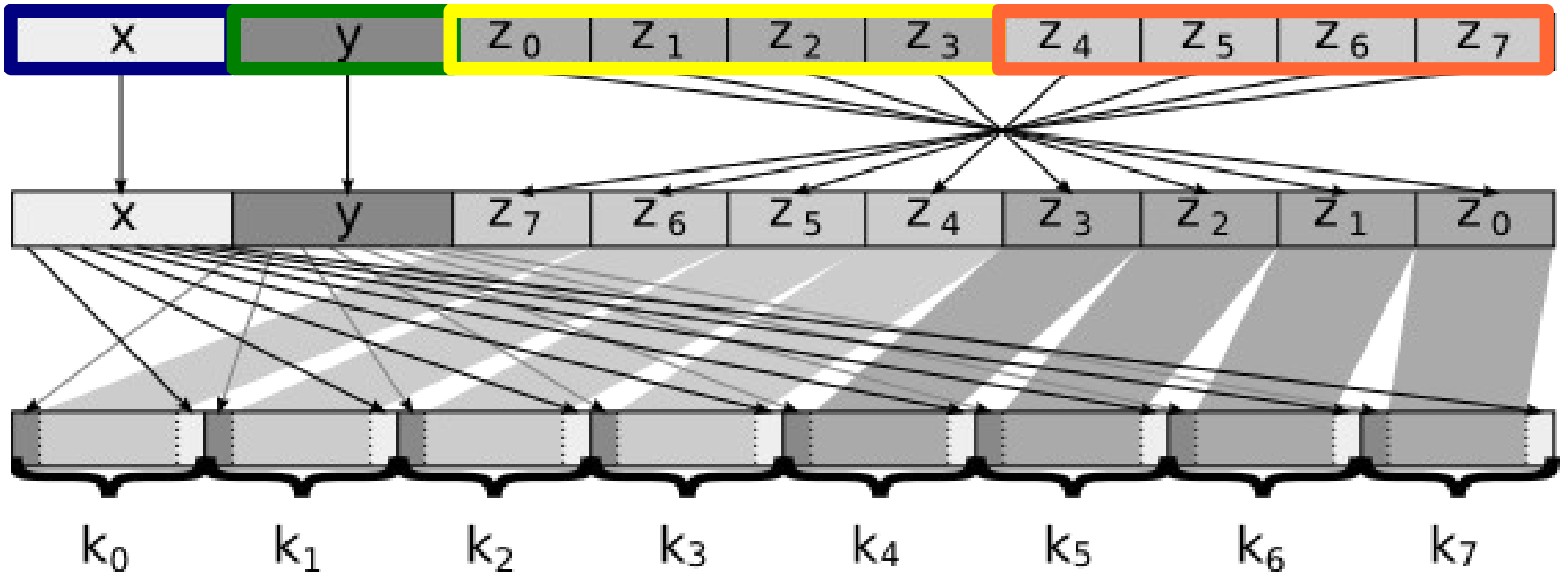
h0(4000000000000000) = 0306050c04050d00

0	**7e7e7e7e**00000000	24	00000000**027e7e7e**
1	**7e7e7e7e**00000000	25	00000000**047e7e7e**
2	**7a7e7e7e**00000000	26	00000000**087e7e7e**
3	**727e7e7e**00000000	27	00000000**107e7e7e**
4	**627e7e7e**00000000	28	00000000**207e7e7e**
5	**427e7e7e**00000000	29	00000000**407e7e7e**
6	00**7e7e7e**00000000	30	0000000000**7e7e7e**
	...		...
11	00**7e7e7e**00000000	35	0000000000**7e7e7e**
12	0000**7e7e**00000000	36	000000000000**7e7e**
	...		...
17	0000**7e7e**00000000	41	000000000000**7e7e**
18	000000**7e**00000000	42	00000000000000**7e**
	...		...
23	000000**7e**00000000	47	00000000000000**7e**

	or-mask	and-mask
48	**fc**00000000000000	**80**00000000000000
49	00**fc**000000000000	00**80**000000000000
50	0000**fc**0000000000	0000**80**0000000000
51	000000**fc**00000000	000000**80**00000000
52	00000000**fe**000000	00000000**fe**000000
53	0000000000**fe**0000	0000000000**fe**0000
54	000000000000**fe**00	000000000000**fe**00
55	00000000000000**fe**	00000000000000**fe**
56	**7f7f7f7e7e7f7f7f**	**01010100000101**01
57	0000**7f7e7f**000000	0000**01**00**01**000000
58	**7f7e7e7e7f**000000	**01**0000000**01**000000
59	**7f7e7e7e7e7f**0000	**01**000000000**01**0000
60	0000**7f7e7e7e7f**00	0000**01**000000**01**00
61	**7f7e7f7f7f7f7f**00	**01**00**01010101**0100
62	**7f7e7f7e7e7f7f**00	**01**00**01**0000**01**0100
63	**7f7e7f7e7f7e7f**00	**01**00**01**00**01**00**01**00

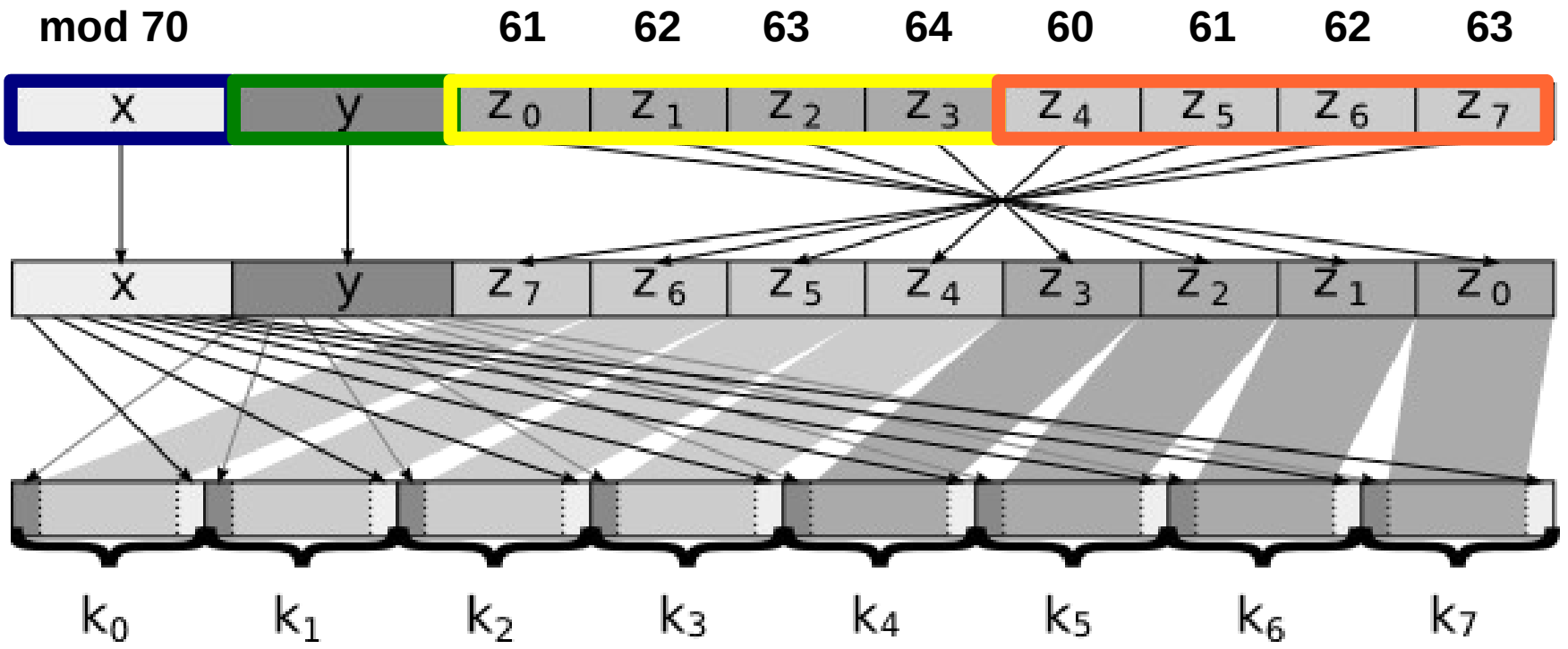
		or-mask	and-mask
NEGATION	48	**fc**000000000000000	**80**000000000000000
	49	00**fc**00000000000000	00**80**00000000000000
	50	0000**fc**0000000000	0000**80**0000000000
	51	000000**fc**00000000	000000**80**00000000
	52	00000000**fe**000000	00000000**fe**000000
	53	0000000000**fe**0000	0000000000**fe**0000
	54	000000000000**fe**00	000000000000**fe**00
	55	00000000000000**fe**	00000000000000**fe**
PERMUTATION	56	**7f7f7f7e7e7f7f7f**	**01010100000101**01
	57	0000**7f7e7f**000000	0000**01001**01000000
	58	**7f7e7e7e7f**000000	**01**00000**01**000000
	59	**7f7e7e7e7e7f**0000	**01**0000000**01**0000
	60	0000**7f7e7e7e7f**00	0000**01**000000**01**00
	61	**7f7e7f7f7f7f7f**00	**01**00**1010101**00
	62	**7f7e7f7e7e7f7f**00	**01**00**1**0000**101**00
	63	**7f7e7f7e7f7e7f**00	**01**00**1**00**1**00**1**00

Structure of hash0



permute negate

Structure of hash0



permute negate

hash0

- We fully recovered hash0
- It is clearly **not**
  - Collision resistant
  - One-way
- We were able to invert **hash0**
  - On average we have 4 candidate pre-images
- Recovering the master key comes down to a brute force on single DES (Few days on RIVYERA)

Key Recovery Attack (Phase 1)

Key Update: $k_{master} \rightarrow k_{new}$

emulated **id**

The attacker knows k_{new}

and therefore learns $hash0(DES_{enc}(id, k_{master}))$

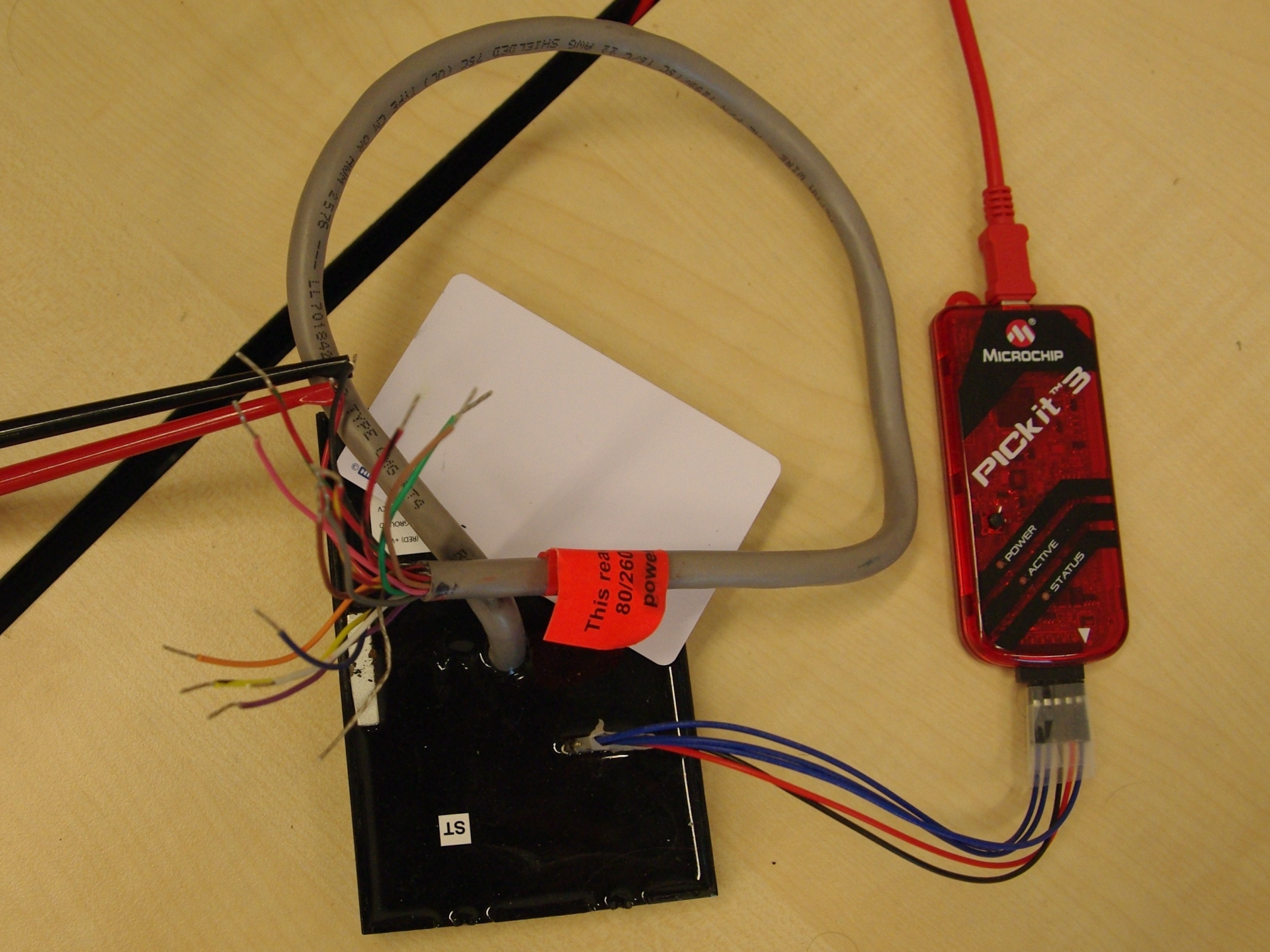
Key Recovery Attack (Phase 2)

- For every DES key **k** check if **DES$_{enc}$(id,k)** equals one of the pre-images from phase 1.
- When the check above succeeds the corresponding key **k** needs to be verified against another emulated **id.**

- A single DES key can be broken within days. We checked the recovered candidates against the master key that we obtained from the reader firmware.


12VDC and
Check
necting


BEEPER (YEL)
HOLD (BLU)
TAMPER (VIO)
DATA1 (WHT)
DATA0 (GRN)
GRN LED (ORN)
RED LED (BRN)


MICROCHIP
PICkit 3
POWER
ACTIVE
STATUS
ST

Verification of Results

- We recovered the master key from firmware

  as done by Meriac and Plotz in [***HID iClass Demystified*, 27th CCC, Dec 2010**]

- This verified that we found the correct key

Conclusion

- Single DES for diversification (broken since 1997)
- The **hash0** function is not:
  - pre-image resistant
  - collision resistant
- **hash0** can be inverted (on average 4 pre-images)
- ...recover the master key from key update message!
- **One master key for every iClass system**

Next step...

- iClass Authentication Algorithm

Questions?