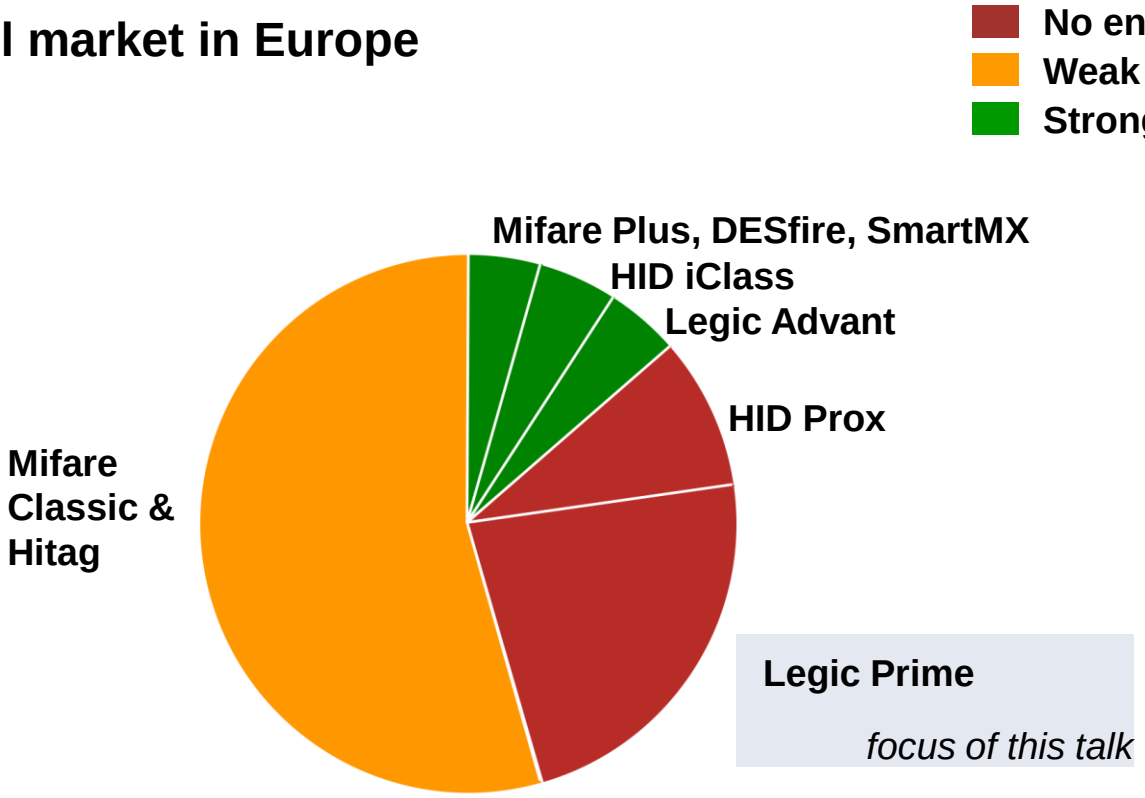
Legic Prime – Obscurity in Depth

Henryk Plötz <ploetz@informatik.hu-berlin.de>
Karsten Nohl <nohl@virginia.edu>

SECURITY
RESEARCH
LABS

RFID access control does not widely use appropriate encryption

Access control market in Europe (Illustrative) :



SECURITY RESEARCH LABS

SECURITY RESEARCH LABS

Contents

SECURITY RESEARCH LABS

Legic‘s RFID access and payment cards have an aura of mistery

- Contactless smart cards at 13.56MHz
  - **A** Legic Prime – Proprietary, marketed since 1992
  - **B** Legic Advant – ISO compliant, marketed since 2004
- Predominantly used in access control, but payment applications exist (i.e., cafeteria)
- Can hold several applications, with easier management compared to Mifare Classic
- Main difference to other systems: Master Token System Control

Legic takes obscurity to the extreme

- Shrouded in a cloud of closed-ness and exclusivity
- Compared to Mifare – Much harder to get cards and readers
- No documentation available beyond layer 1/2

A Legic Prime is outdated but popular

- Old card type, as old as Mifare Classic (and at least as insecure)
- Proprietary radio protocol (applied to become ISO 14443 Appendix F) – „LEGIC RF“
- Proprietary „Legic Encryption“
- Slow data rate (~10 kbit/s), comparatively high read range (supposedly up to 70 cm)
- Card types
  - MIM22 (outdated)
  - MIM256 (234 bytes storage)
  - MIM1024 (1002 bytes storage)

SECURITY RESEARCH LABS

Legic Advant uses modern encryption

- New card type, developed in the 2000's
- Based on ISO 14443A or ISO 15693
- 3DES or AES, also backward compatible to „Legic Encryption“
- Several ATC card types with varying sizes (15693: 128-944 bytes, 14443: 544-3680 bytes)
- Not yet analyzed by us, therefore not covered in this talk

SECURITY RESEARCH LABS

Contents

SECURITY RESEARCH LABS

Master Token System Control enables hierachichal access rights management

„The powerful LEGIC Master-Token System Control (MTSC) (...) is unique in the security industry. With MTSC no sensitive passwords are needed. Instead, a special physical Master-Token (...) is used containing a unique genetic code which securely links cards and readers“

—Legic web site

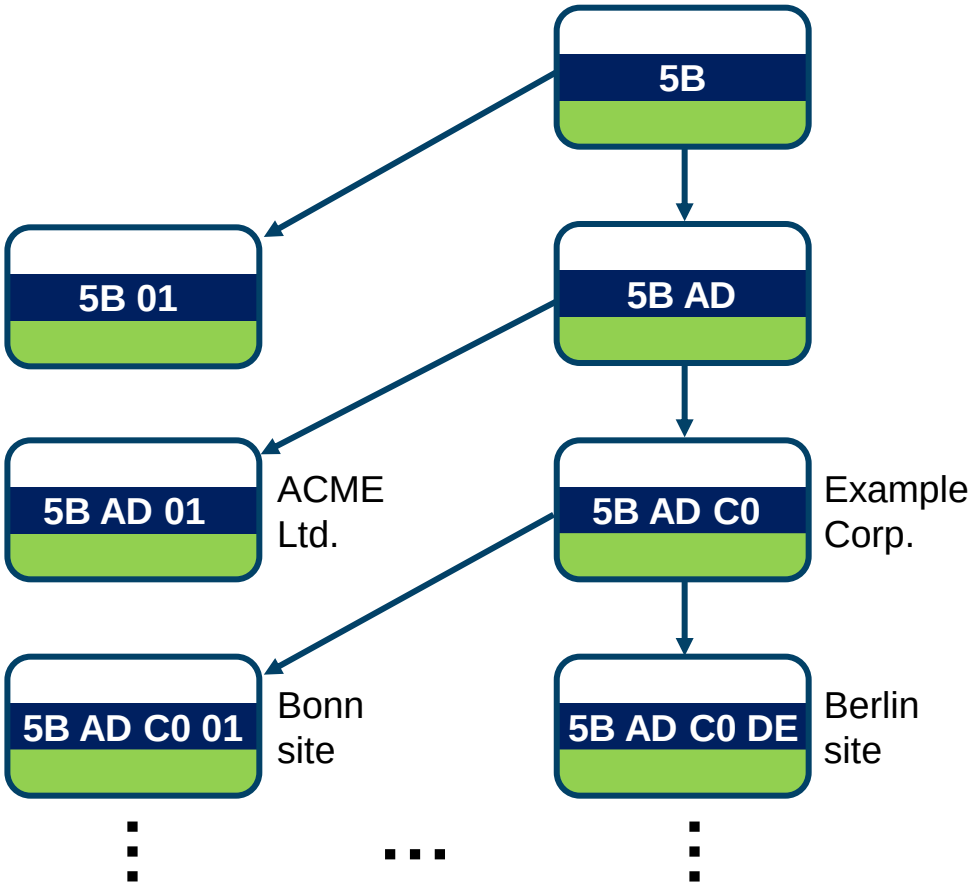
- Hierarchical structure with Legic or a license partner at its root; system integrators, customers and customer subdivisions at the nodes
- Node identifer is called the stamp (or „genetic code“)
- Each master token is associated with one stamp

SECURITY RESEARCH LABS

Source: http://www.legic.com/unique security.html

MTSC structures tokens in tree

A token can only create objects with higher nesting level than its own → longer stamp, but same prefix



Legic / Legic license partner

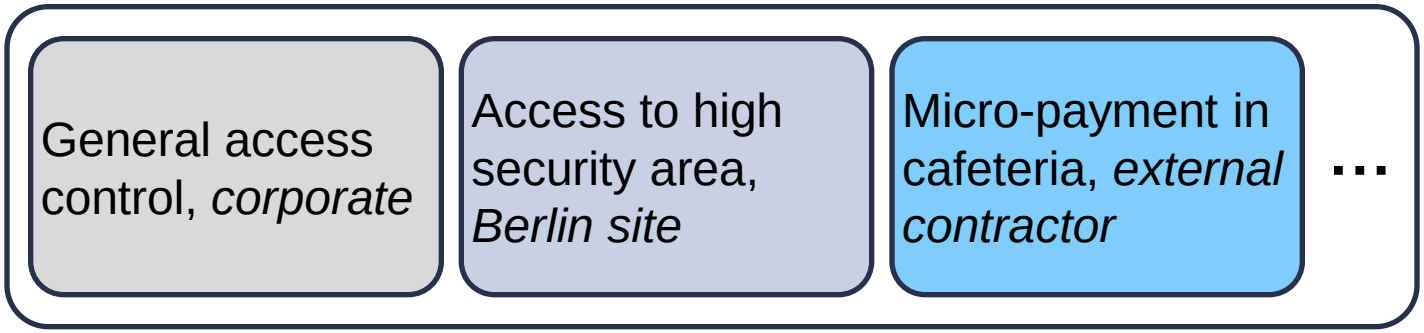
Systems integrator

Customer

Customer divisions / sites / etc.

SECURITY RESEARCH LABS

Multiple applications can be stored in separate segments



Example—employee card

- Cards are segmented and read / write access is regulated on a per-segment basis
- Segment access is bestowed through physical token instead of keys or passwords
- The MSTC token itself is a Legic Prime or Advant card

SECURITY RESEARCH LABS

Segments pretend to be protected

Segments on cards are imprinted with a stamp on creation

- Stamp comes from the token that authorized the creation
- Stamp can not be changed

Optionally, segments can be „read protected“

Readers are loaded with access rights for none / one / multiple stamps

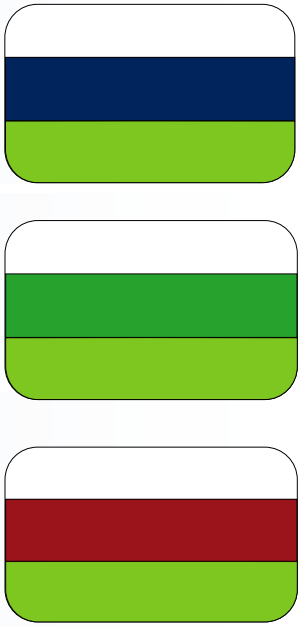
Card-Reader interaction

- Read read-protected segment and write – Only if reader has access rights for that segment's stamp
- Read non-read-protected segments – All readers can do this

SECURITY RESEARCH LABS

Three types of tokens with special privileges exist

General Authorization Media (GAM)	Token-creating token that carries the **temporary** authorization to **create sub-tokens**
Identification Authorization Media (IAM)	Segment-creating token that carries the **temporary** authorization to **create segments** on cards
System Authorization Media (SAM)	„Reader-creating“ token that bestows the **permanent** authorization to **write** to existing segments on cards (and **read** read-protected segments)



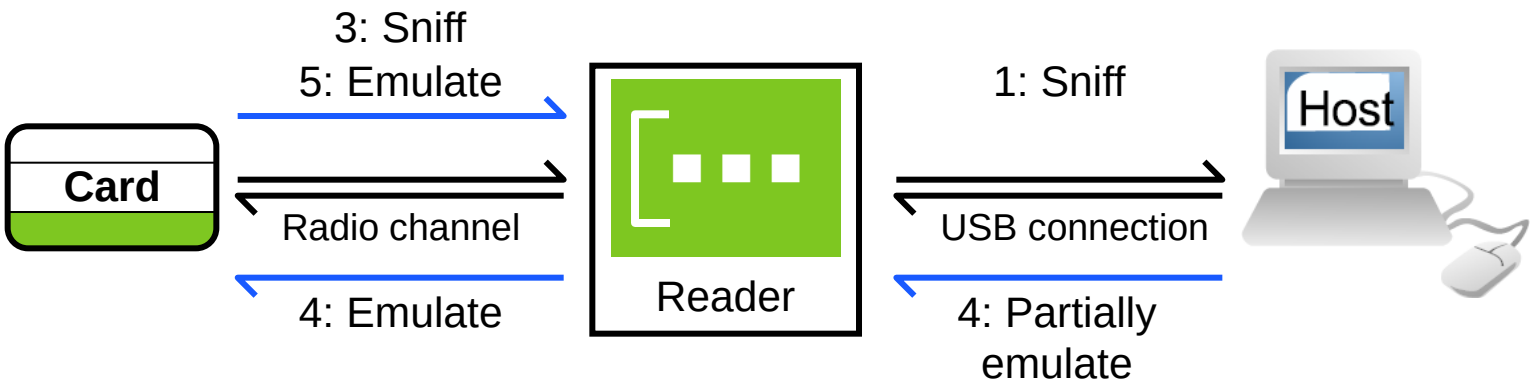
For the SAM (a.k.a. SAM63, a.k.a, Taufkarte‘), which ‚launches‘ readers (‚taufen‘), there is a counterpart – SAM64 (a.k.a ‚Enttaufkarte‘) to de-launch readers (‚enttaufen‘)

SECURITY RESEARCH LABS

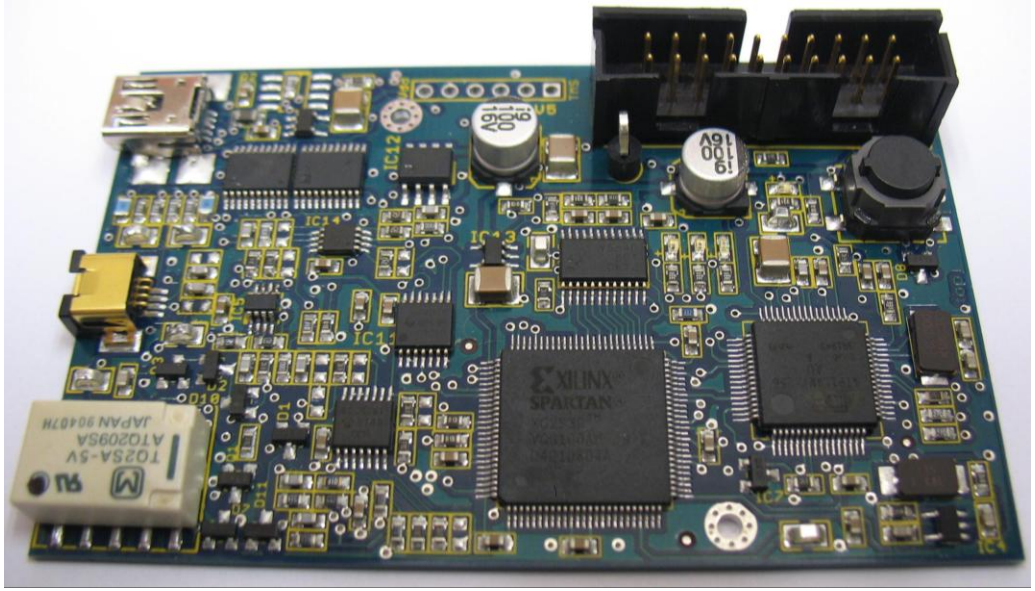
Contents

SECURITY RESEARCH LABS

Legic Prime can be analyzed and attacked with standard tools



Attacks were implemented using the Proxmark3



SECURITY RESEARCH LABS

Contents

SECURITY RESEARCH LABS

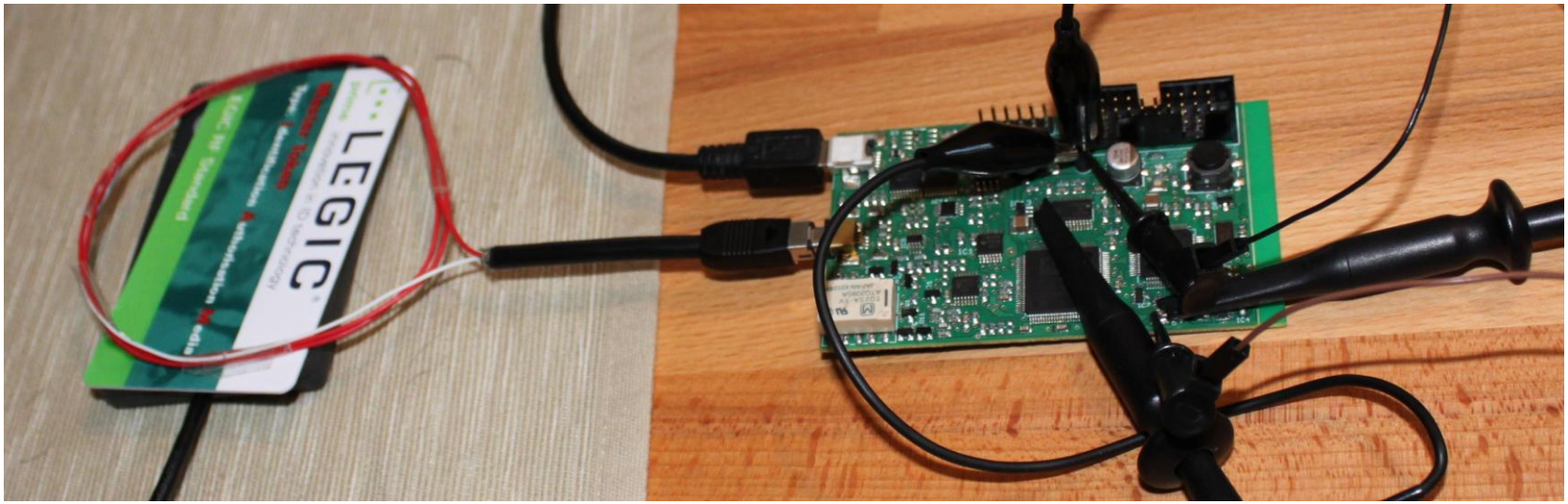
LEGIC RF Layer 1 is publicly documented

ISO 14443 Annex F gives general parameters:

- RWD to TAG – Pulse-pause modulation, 100% AM, off-duration: 20µs, 0‘-bit: on-duration 40µs, ‚1’-bit: on-duration 80µs, data rate 10 kHz–16.6…kHz (data-dependent)
- TAG to RWD – On-off-keying, load-modulation, subcarrier f_c / 64 (~212kHz), bit-duration: 100µs
- Framing „defined by the synchronization of the communication“
- No frame start / stop information for tag originated frames

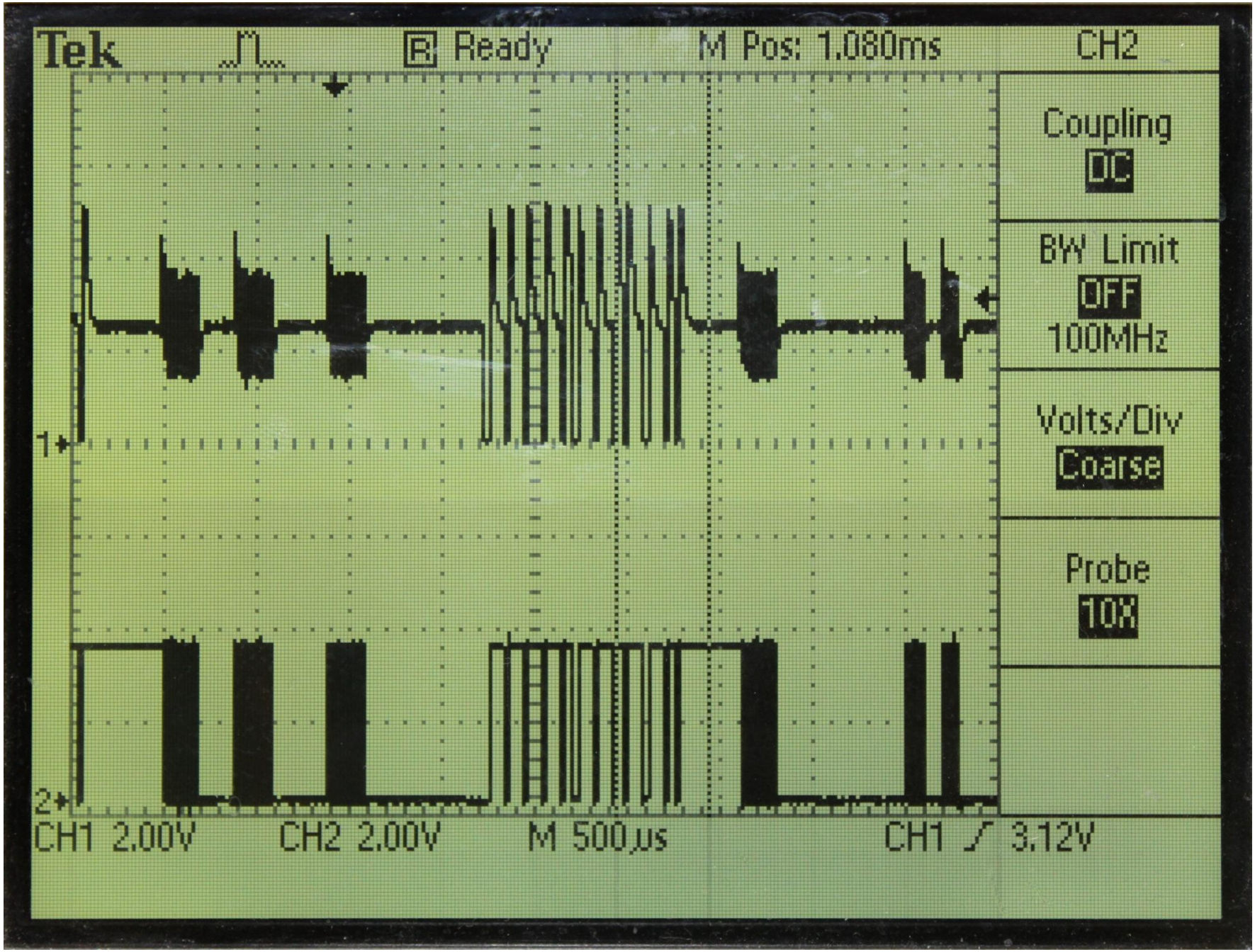
SECURITY RESEARCH LABS

LEGIC RF can be sniffed with standard RFID tools



Sniffing with OpenPICC2 (fixed threshold, not so good) or Proxmark3 (hysteresis, much better) and oscilloscope or logic analyzer

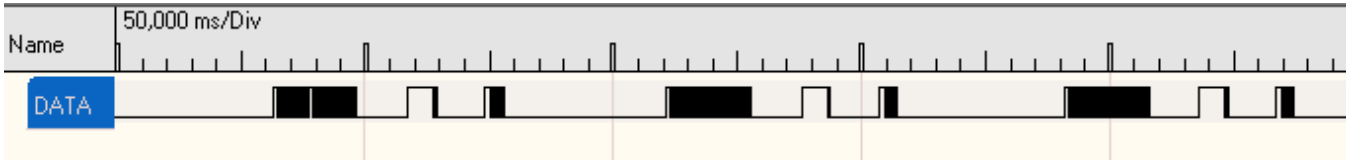
SECURITY RESEARCH LABS


Tek
Ready
M Pos: 1.080ms
CH2
Coupling
DC
BW Limit
OFF
100MHz
Volts/Div
Coarse
Probe
10X
1
2
CH1 2.00V
CH2 2.00V
M 500µs
CH1 3.12V

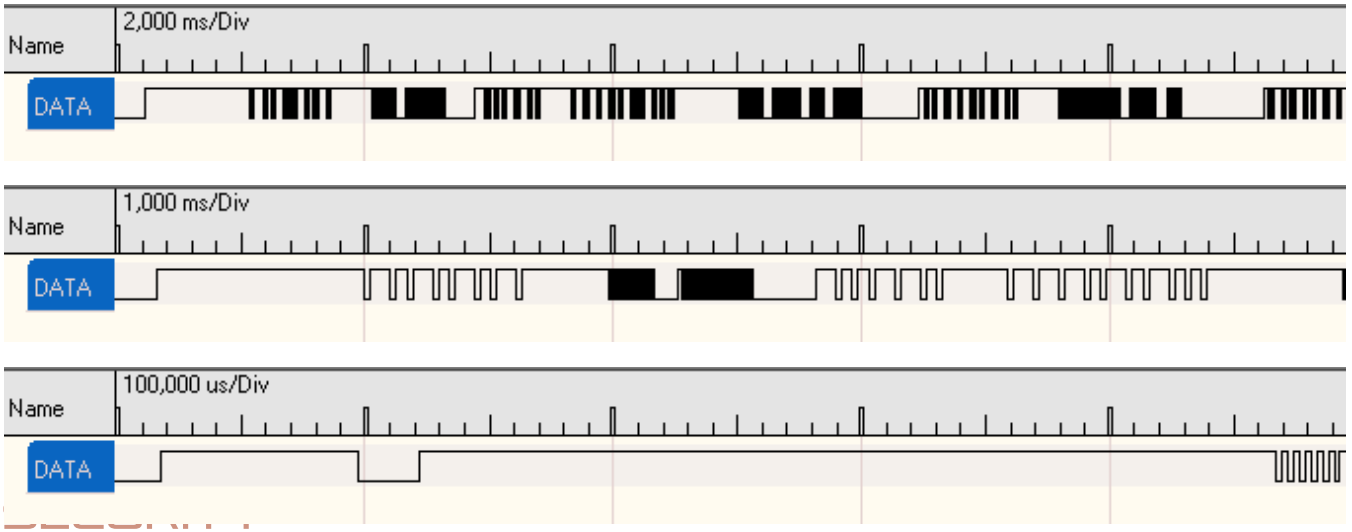
Protocol phases can be visualized with cheap logic analyzer

Logic analyzer data

„Get UID“ type command, cycles through LEGIC RF, then ISO 14443-A, then ISO 15693



„Get UID“ transaction consists of multiple exchanges by card and reader



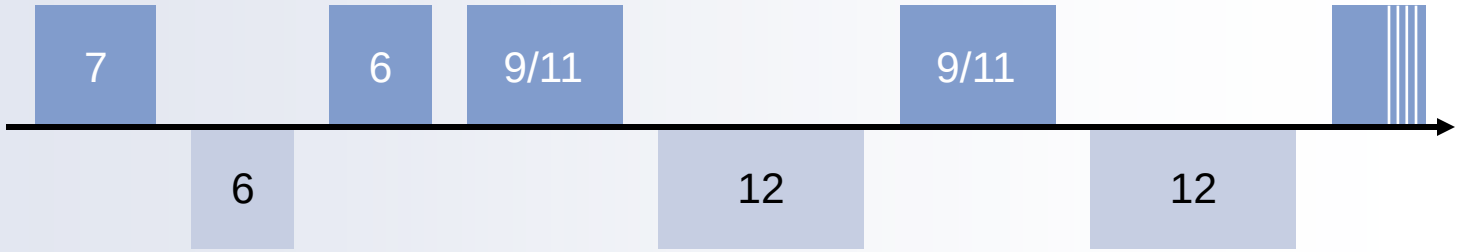
RESEARCH LABS

Legic Layer 2 decoder had to be written

Custom decoder in C

Delay between RWD command and TAG response seems to be constant, approx 330µs

As expected – TAG-originated frames are not delimited, length unclear



Comparing many traces yields the protocol structure

- Setup, once per session
  - 7 bits from RWD
  - 6 bits from TAG
  - 6 bits from RWD
- Repeat several times, once for each byte requested
  - 9 bits from RWD (Depending on card type: 11 bits for MIM1024)
  - 12 bits from TAG

Data stream appears to be encrypted

Let the 7-6-6 exchange be the ‚setup phase' and the remainder of the session be the ‚main phase'

First 7-bit-command from RWD is more or less random, but always has first bit set, name it RAND. Assumption: IV of a stream cipher

- RNG is weak:
  a) Too small; b) 0x55 in ~10% percent of cases (vs. expected 1.5%)

For a given RAND the rest of the setup phase is identical over all cards of the same type (MIM256 and MIM1024 differ by one bit)

Within a card type, for a fixed RAND, all reader command sequences are identical

Looks like a stream cipher with weak IV from reader and no random from the card

SECURITY RESEARCH LABS

First command in transaction queries card UID

UID 3e 17 44 3e		
Src	Len	Bits
R	7	1010101
T	6	010001
R	6	111000
R	9	010010100
T	12	100010001101
R	9	001011100
T	12	111111000110
R	9	010100101
T	12	110011111011
R	9	001011000
T	12	101100001000
R	9	111101111
T	12	011001100001

UID 3e 58 b8 79		
Src	Len	Bits
R	7	1010101
T	6	010001
R	6	111000
R	9	010010100
T	12	100010001101
R	9	001011100
T	12	000111101111
R	9	010100101
T	12	111100001010
R	9	001011000
T	12	010000101111
R	9	111101111
T	12	001100101010

SECURITY RESEARCH LABS

Note: These examples are synthetic and do not use the actual generator taps or CRC polynoms

First command in transaction queries card UID

UID 3e 17 44 3e		
Src	Len	Hex
R	7	055
T	6	022
R	6	007
R	9	052
T	12	B11
R	9	074
T	12	63F
R	9	14A
T	12	DF3
R	9	034
T	12	10D
R	9	1EF
T	12	866

UID 3e 58 b8 79		
Src	Len	Hex
R	7	055
T	6	022
R	6	007
R	9	052
T	12	B11
R	9	074
T	12	F78
R	9	14A
T	12	50F
R	9	034
T	12	F42
R	9	1EF
T	12	54C

⊕: 00 4f fc 47
B11 ⊕ B11 = 000
63F ⊕ F78 = 947
DF3 ⊕ 50F = 8FC
10D ⊕ F42 = E4F

SECURITY RESEARCH LABS

Note: These examples are synthetic and do not use the actual generator taps or CRC polynoms

Transaction is armored by two CRC check sums

- Each reader-card request-response pair only transmits one byte of payload
- The UID is transmitted in order first byte, fourth / last byte, third byte, second byte (Compared to the display in the GUI)
- Each response is protected by a 4 bit CRC
- A fifth byte is transmitted after the UID, this is an 8 bit CRC over the UID, stored on the card itself

All commands are read commands

- Hypothesis – Command is 1 bit command code, 8 bit (or 10 bit) address, response is 8 bit data and 4 bit CRC
- „Get UID“ isn't really requesting the UID, but simply reading the first 5 bytes of memory
- Hypothesis confirmed – Lowest bit (first bit transmitted) of command is command code, must not be changed; remaining 8 bits are address
- First command of „Get UID“ sequence is really „Read Byte 0“

Cmd	Arg								
C	x	x	x	x	x	x	x	x	**Original („Read Byte 0“)**
C	$\overline{x}$	x	x	x	x	x	x	x	**New – „Read Byte 1“**
C	x	$\overline{x}$	x	x	x	x	x	x	**New – „Read Byte 2“**
C	$\overline{x}$	$\overline{x}$	x	x	x	x	x	x	**New – „Read Byte 3“**

etc. pp.

- Timing is important. Also: Only one command per setup phase → **4s** to read a full MIM256 card

SECUR
RESEA
LABS

Contents

SECURITY RESEARCH LABS

CRC provides no protection

CRC in stream cipher is well known to be malleable (WEP, Mifare Classic, ...)

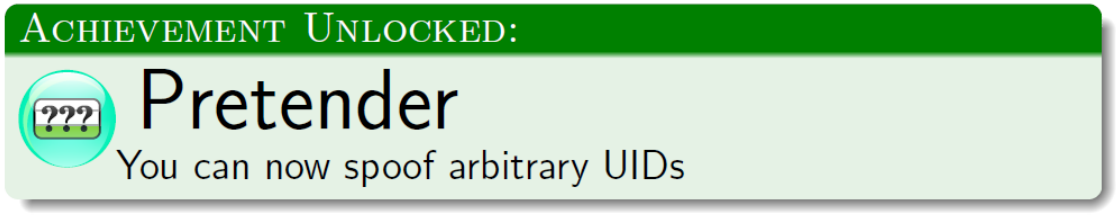
With unknown CRC function, a simple approach is to brute-force the difference values for all 1-bit changes. The Differences are fully additive

Data								**CRC**				
x	x	x	x	x	x	x	x	x	x	x	x	Original (valid)
$\overline{x}$	x	x	x	x	x	x	x	$\overline{x}$	$\overline{x}$	x	$\overline{x}$	1st Difference (valid)
x	$\overline{x}$	x	x	x	x	x	x	$\overline{x}$	x	$\overline{x}$	x	2nd Difference (valid)
x	x	$\overline{x}$	x	x	x	x	x	x	$\overline{x}$	x	$\overline{x}$	3rd Difference (valid)
Use as follows												
$\overline{x}$	x	$\overline{x}$	x	x	x	x	x	$\overline{x}$	x	x	x	Modified data (valid CRC)

After being able to freely anticipate the transport CRC, the UID-CRC can be attacked in a similar manner

Yields two tables: A) Transport CRC: 8 entries of 4 bits; B) UID CRC: 32 entries of 8 bits

Gather known UID transactions for as many RANDs as possible (we managed 59 out of theoretically 64), modify responses for requested UID



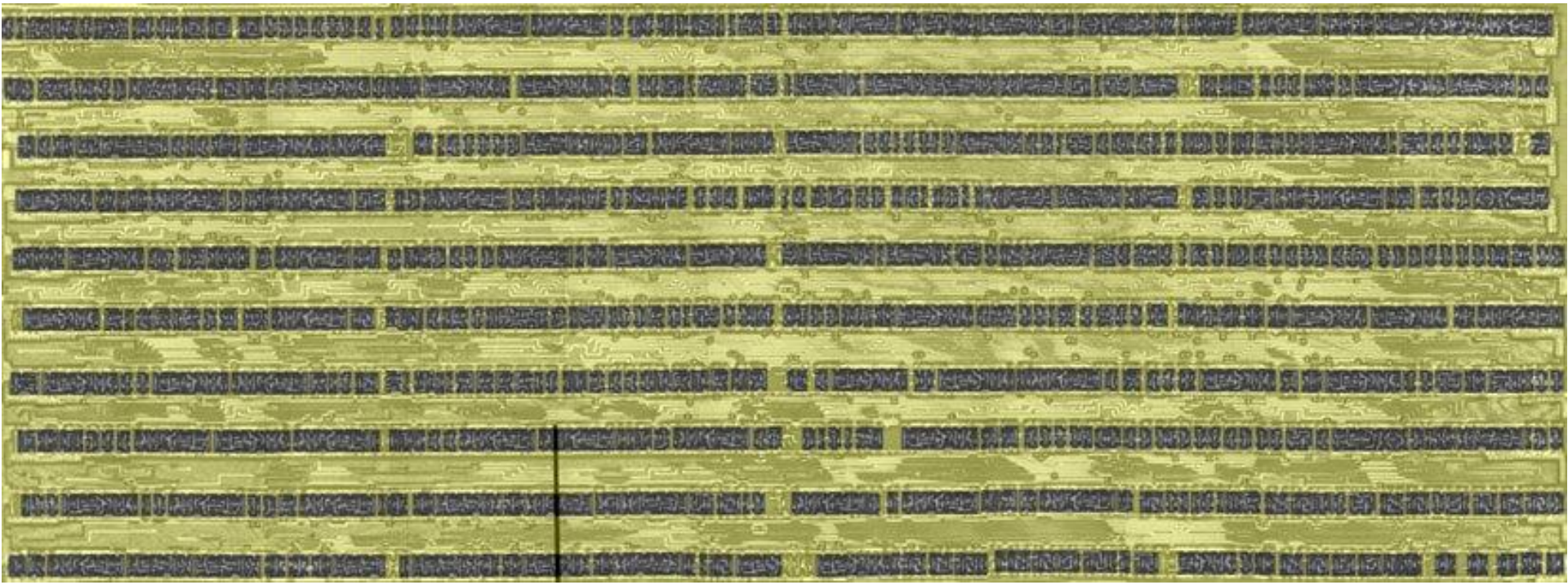
SECURITY RESEARCH LABS

Note: These examples are synthetic and do not use the actual CRC polynomial

Contents

SECURITY RESEARCH LABS

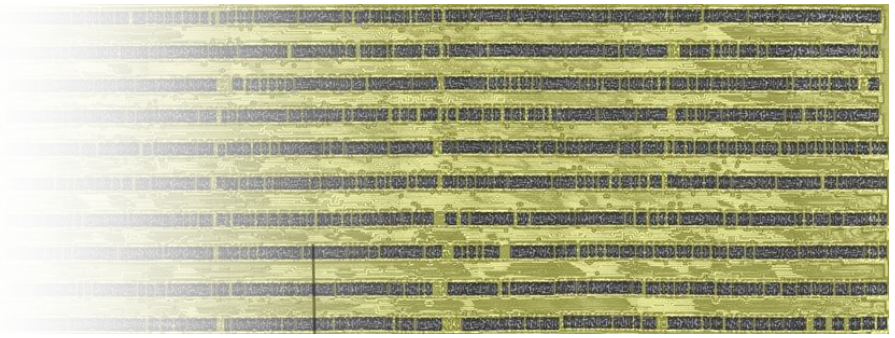
Legic Prime tag was reverse-engineered completely from its silicon implementation

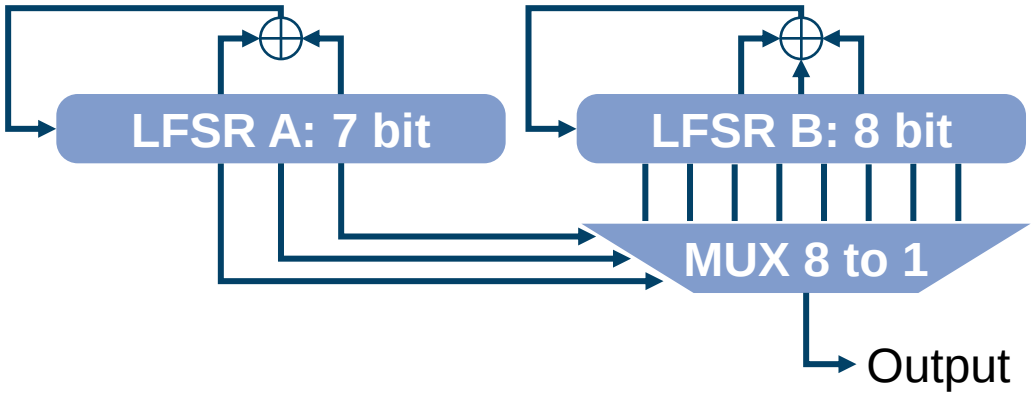


SECURITY RESEARCH LABS

Obfuscation function quickly found from reversed circuit

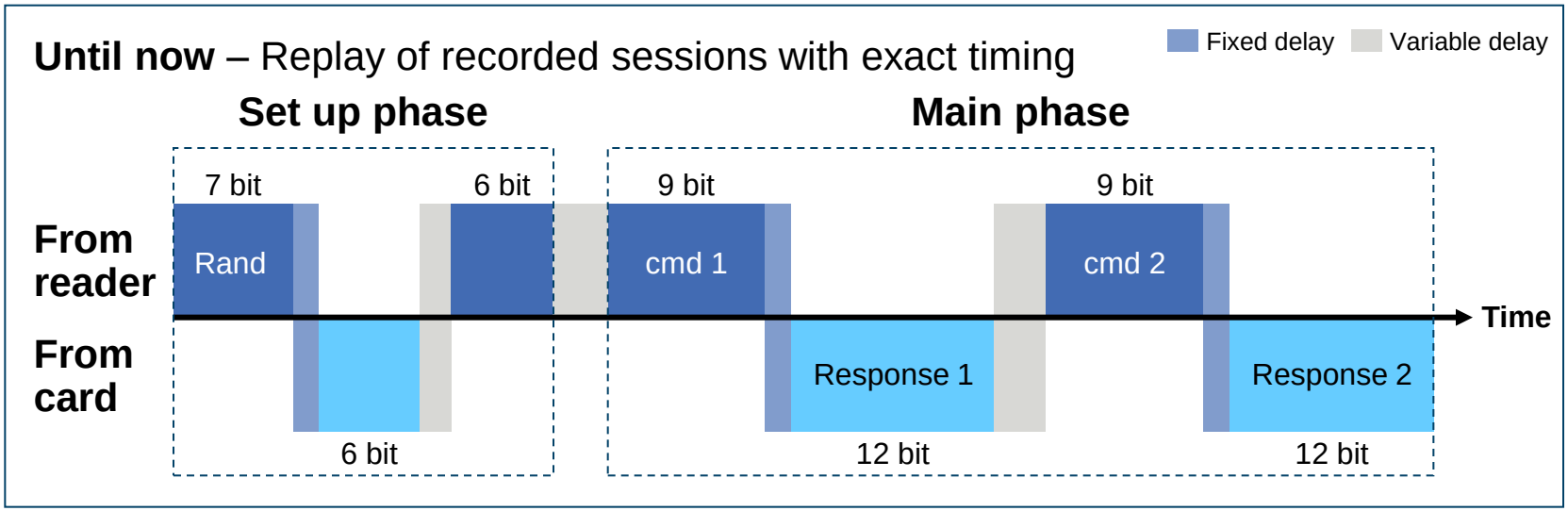
- The LEGIC obfuscation function consists of two LFSRs
- Easily invertible, but not even needed for a state this small





SECURITY RESEARCH LABS

Playing with timing discloses further protocol details

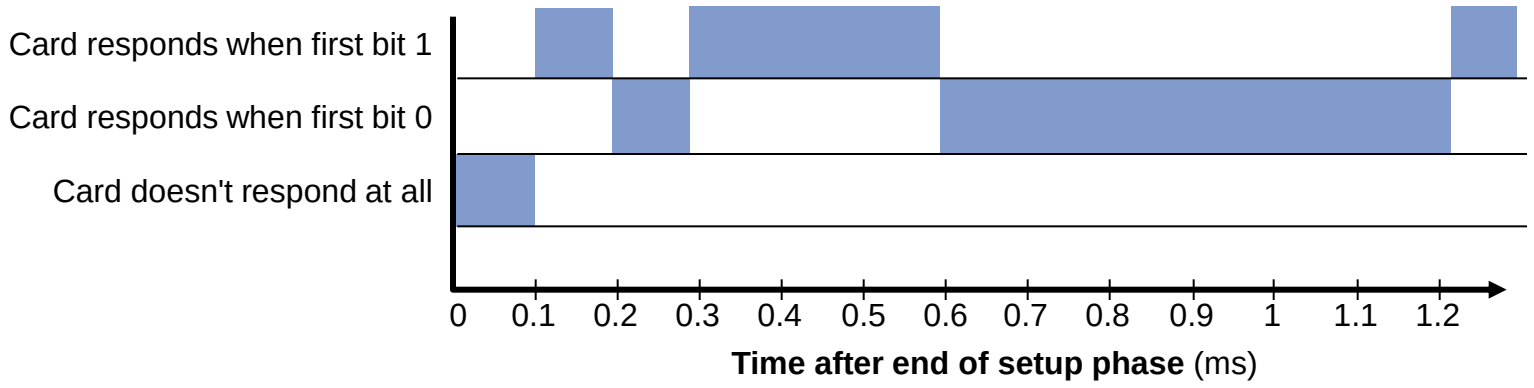


Experi-ment	Vary the timing before the first command
Result	Card response for some delays, no card response for others
Interpre-tation	Result of the de-obfuscation changed, which changes the command bit
Conclu-sion	The obfuscation stream generator is continuously running (at period time ~100µs)

SECURITY RESEARCH LABS

Obfuscation stream can be determined experimentally

Vary first bit of command at each time off set → gives first bit of obfuscation stream at that time offset



Interpre-tation

The obfuscation stream generator generates a new bit approx. every 100µs (more like 99.1µs, might be reader-specific)

Complete break, even without a microscope – Generate arbitrary amounts of obfuscation stream by leveraging a few bits of known plaintext (optimized attack: 14 hours preparation, 4 kilobytes storage; naive attack: 4 days preparation, 80 kilobytes storage)

SECURITY RESEARCH LABS

Note: These examples are synthetic and do not use the actual obfuscation stream

Knowledge of functions and protocol enables compatible Legic reader

Knowledge of the experimentally determined obfuscation stream allows to find the initialization for the function (brute force)

Initialization

1st step Load R_a = RAND and R_b = (RAND << 1)|1

2nd step That's it, there's no 2nd step

- No key input → not technically an encryption

Can now generate obfuscation stream at any point in time

Can send as many read commands in one single session as necessary → 0.69s for a full dump of a MIM256

SECURITY RESEARCH LABS

Contents

Full emulation requires deeper understanding of CRC

- Both the slow and the fast reader ignore the transport CRC, but for a full card emulator we need to generate the CRC
- Look at the sniffed communication (de-obfuscated)

Src	Len	Binary	Hex	Interpretation
(setup phase omitted)				
RWD	9	1 0000 0000	001	Read byte 0
TAG	12	0111 1100 1111	F3E	Answer: 3e, CRC f
RWD	9	1 1000 0000	003	Read byte 1
TAG	12	0111 1100 0110	63E	Answer: 3e, CRC 6
RWD	9	1 0100 0000	005	Read byte 2
TAG	12	0010 0010 0000	044	Answer: 44, CRC 0
RWD	9	1 1100 0000	007	Read byte 3
TAG	12	1110 1000 0010	417	Answer: 17, CRC 4
RWD	9	1 0010 0000	009	Read byte 4
TAG	12	0001 0010 0111	E48	Answer: 48, CRC e

SECURITY RESEARCH LABS

Note: These examples are synthetic and do not use the actual CRC polynom

CRCs can be reversed with generic approach

- **A CRC is determined by four parameters –** Register width, polynom, initial value, final XOR
- **Storage CRC is 8 bits, transport CRC is 4 bits –** Easy to brute-force over the full parameter space
- **If all the known inputs are of the same length, initial value and final XOR are equivalent –** Fixing one to an arbitrary value gives a solution for the other
- **Better than brute force –** Analysis of the 1-bit differences allows direct determination of the CRC parameters

Knowledge of transport CRC enables card emulation …

Differently sized commands (9 bit for MIM256, 11 bit for MIM1024) allows to disambiguate initial value and final XOR

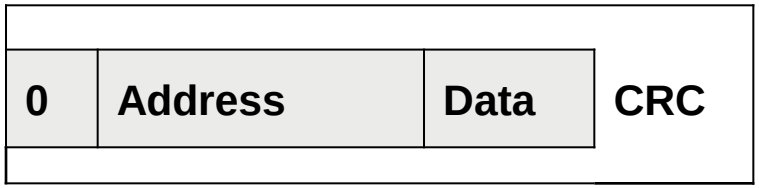
Result – Transport CRC is made over the full command and the full payload of the response

From reader		From card	
1	Address	Data	CRC
8/10 bits		8 bits	4 bits

SECURITY RESEARCH LABS

… and provides write access to cards

- Write commands are 21 bit for MIM256 and 23 bit for MIM1024
- Contains command code („0“), 8/10 bit address, 8 bit data, 4 bit CRC
- Same CRC as for read commands, calculated over the full 17/19 bits



- Card acknowledges with a single „1“-bit, after 3.6 ms
- Obfuscation stream is unaffected by ACK

LABS

Contents

SECURITY RESEARCH LABS

Master tokens are not read protected

Analysis of a „load IAM“ or „launch reader“ process reveals

- UID is read, UID-CRC is read
- Bytes 6 and 5 are read (in that order)
- Byte 7…(7+stamp length) are read
- Byte 21 is read

Launch process takes a long time, ~15s, providing the illusion that something profound is happening (key-derivation? lengthy EEPROM reprogramming?)

- On the radio channel, byte 4 (UID-CRC) is read every 1s, to ping whether the card is still there

Combining the address / data information from a sniff, the following structure of an IAM is revealed

Address	Data							
0	3e	3e	44	17	48	2f	f8	04
8	5b	ad	c0	de				
16						0e		

Note: These examples are synthetic and do not use the actual CRC polynom. Also, the stamp is fake. Obviously.

Tokens can be fully copied and emulated

- Naive transfer to physical card not successful
- Bytes 5 and 6 behave strange when writing → can only be decremented
- Complete emulation is successful
- Playing with the emulated card reveals – Byte 21 is a CRC, secures UID and stamp
- Exhaustive search over the CRC byte enables emulation of an IAM for different stamps

SECURITY RESEARCH LABS

Knowledge of content CRC enables easier token creation and emulation

Analysis of CRC bit differences reveals – Same CRC polynom as for the UID

Further analyses find a common set of parameters for the UID CRC and the master token CRC

- Disambiguates initial value / final XOR
- Master token CRC is calculated over
  - UID, bytes 0 thru 3
  - Bytes 6 and 5
  - Byte 7
  - Stamp, bytes 8 thru (8+(stamp length)-1)

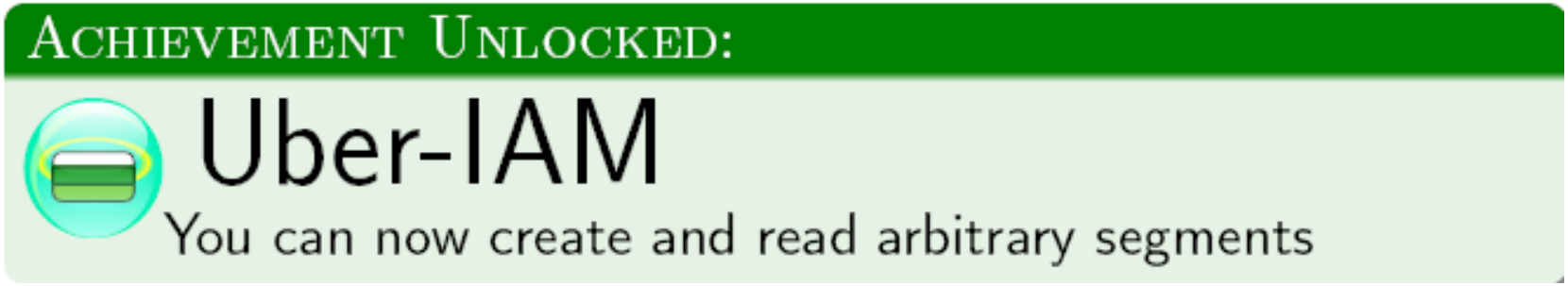
Can now emulate IAM and SAM for arbitrary stamps of length 4

SECURITY RESEARCH LABS

S(tamp s)ize matters

Now that we can generate the master token CRC, let's play with the different bytes

- Byte 5 seems to control the token type
- Byte 6 seems to control the stamp length, in coordination with byte 7
- Byte 7 is 0x04 for the IAM and 0x44 for the SAM (both of stamp length 4)
- Wrong values for byte 6 tend to freak out the software – Differing error messages, exceptions, crashes or the mute pretense that the card is empty
- **Lucky accident –** Set byte 7 to 0x00, byte 6 to 0xfc and we got ourselves an IAM of stamp length 0



Contents of system master tokens is deterministic

Byte 7 is RD / WRP / WRC

- Low nibble controls the stamp size
- High nibble controls the stamp size for the launch process

Byte 5 is token type – MSBit controls whether the token can create sub-tokens (OLE), remaining 7 bits are

- **0x00 – 0x2f** IAM
- **0x30 – 0x6f** SAM
- **0x70 – 0x7f** GAM

Byte 6 is the organisational level? Must be 0xfc – (Stamp length)

SECURITY RESEARCH LABS

Extent of pwnage

Can create IAMs and SAMs for arbitrary stamps of arbitrary lengths (including 0!)

- If the SAM should launch readers, its stamp length must be at least 1
- Uber-IAM allows full read and creation access to arbitrary stamps

Can create GAMs for arbitrary stamps of length 2 or higher

- The software seems to specifically lock out shorter GAMs, pretends the card is empty

Contents

SECURITY RESEARCH LABS

Card data needs to be interpreted in light of the application

Reverse engineering card contents not necessary for the standardized types (e.g., cash, access, biometric) – Simply use the regular software together with the Uber-IAM

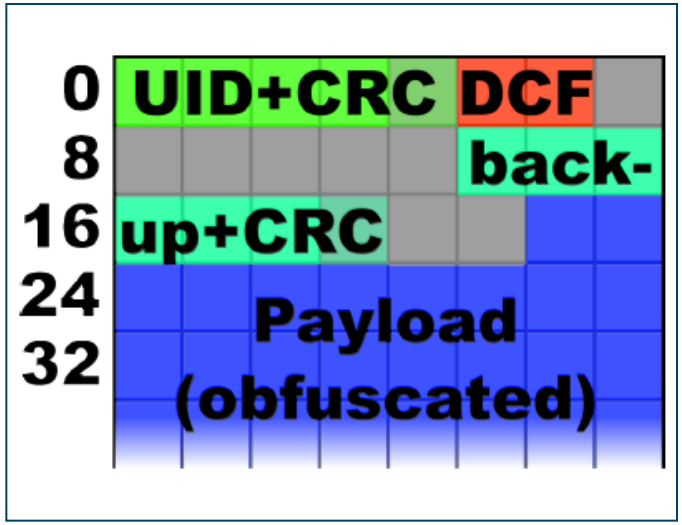
Otherwise, if available, use csg files (Legic segment definition) to aid in interpretation

Data on the card is further obfuscated – All payload bytes are XORed with some value. That value is the CRC of the UID (which is also stored on the card)

- **Obscurity In Depth**

Cards are divided in header and application data

- 4 bytes UID + 1 byte CRC
- 2 bytes decremental field (DCF), is 0x60 0xea for all cards that aren't master token
- 6 bytes unknown / unused / fixed, might be a version identification, possibly related to old unsegmented cards
- 6 bytes segment header backup area + 1 byte CRC
- 2 bytes unknown / unused
- Remainder – Obfuscated payload



SECURITY RESEARCH LABS

Each application has its own segment

Segment header is 4 bytes + 1 byte CRC	▪ **1st byte –** Lower byte of segment length (including header) ▪ **2nd byte, lower nibble –** High nibble of segment length ▪ **2nd byte, high nibble –** Flags: 0x8 == last segment flag, 0x4 == segment valid flag (if flag is not set, the segment is deleted) ▪ **3rd byte –** WRP, length of write protected area of the segment. Always includes the stamp length ▪ **4th byte, bits 4 thru 6 –** WRC ▪ **4th byte, MSBit –** RD, read protection
Segment header write procedure	▪ Save old segment header to backup area ▪ First byte of backup area: = 0x80 (‚dirty`) | segment number ▪ Write new segment header ▪ Clear dirty flag in backup area

SECURITY RESEARCH LABS

Contents

SECURITY RESEARCH LABS

Legic Prime design is fundamentally flawed

No keys, (no key management, no card authentication, no reader authentication)

- Spoofing, skimming
- Segments can be created out of thin air
- Master token can be created out of thin air

No authorisation necessary for master token use, master token not inherently necessary for segment creation

- Master token clonable

SECURITY RESEARCH LABS

Legic Prime's weaknesses can be softened through software checks and cryptography …

Protection	Effect	Effort
1 Check UID	▪ Fraudster must use expensive emulation device	▪ Minimal software modifications
2 Encrypt data	▪ Reverse-engineering data format becomes difficult	▪ Additional layer in software stack; new keys; potentially faster reader hardware
3 Sign data	▪ Only previously valid states can be cloned	
4 Sign counter	▪ Card value cannot be restored	
5 Fingerprint cards (timing, emulation bit, etc.)	▪ Most emulators will be detected	▪ Requires programmable RFID part in reader

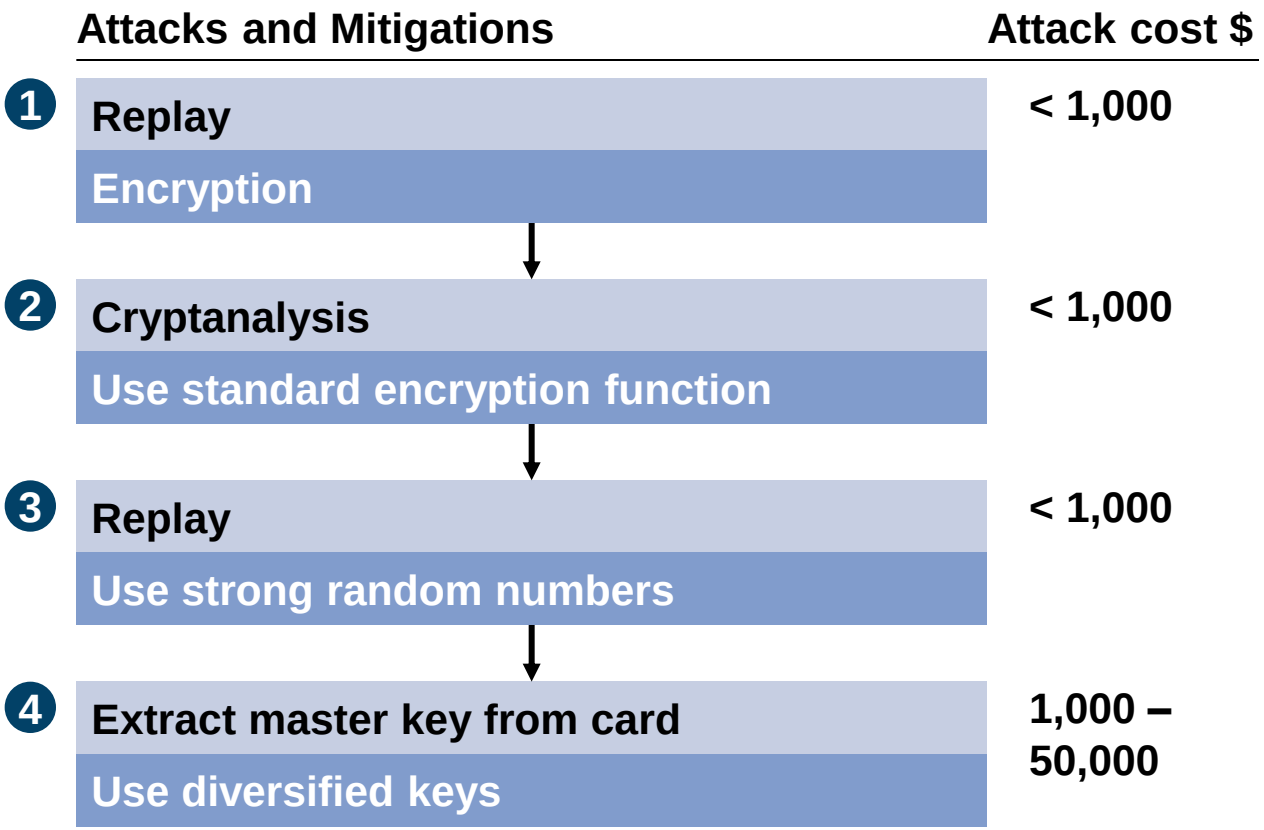
These Mifare / Legic-specific measures are supplemental to generic protections; in particular

- Key diversification
- Anomaly detection in (semi-)online systems

SECURITY RESEARCH LABS

… mitigated through better cryptographic designs in the mid-term…

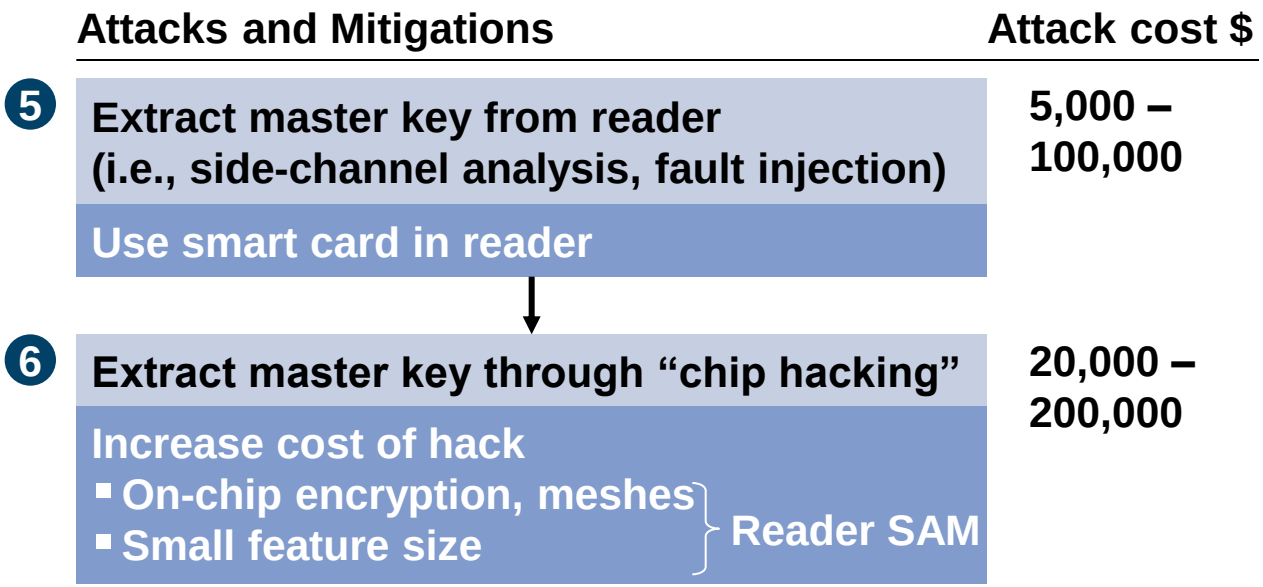
Attack
Mitigation



SECURITY RESEARCH LABS

… and solved in the long-term by using a new generation of access cards

Attack
Mitigation



Modern RFID cards that combine standard cryptography with secure key storage (SAM modules) can be considered secure in excess of EUR 100,000 per card. At this security level, attacks on the system backend might very well be the system’s weakest link

SECURITY RESEARCH LABS

Proxmark3 allows pen-testing RFID systems

We released in December:

Legic Prime Reader

- Test whether an access cards is Legic Prime (or HID, Mifare Classic) and hence vulnerable
- Test whether private data is stored on the card (including in read-protected segments)

We do not release: Legic Emulator & Full Protocol

- Reverse-engineering these components is not hard
- Therefore – Upgrade ASAP
- Have since been reverse-engineered

Legic Prime writer has appeared in the Proxmark 3 repository

SECURITY RESEARCH LABS

Conclusions

Even multi-level obfuscation does not prevent reverse-engineering

Access cards at the very least need inherent protection in form of good crypto and secret keys

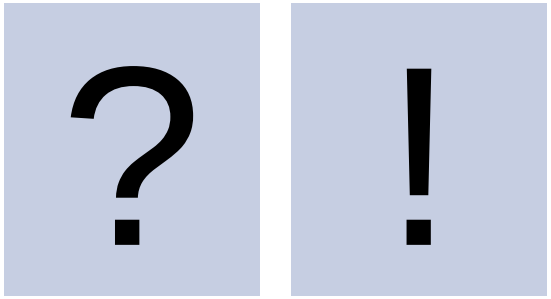
Legic Prime analyzed head to toe

- No actual, inherent security found
- Advertised range ~70 cm and card completely unprotected against skimming → more significant break than with Mifare Classic

Once again – Security by obscurity does not work

SECURITY RESEARCH LABS

Questions?



Henryk Plötz—ploetz@informatik.hu-berlin.de
Karsten Nohl—nohl@virginia.edu

SECURITY RESEARCH LABS

Backup

Please upgrade, just not to HID!

Several RFID cards have been publicly broken over the past years – Mifare Classic, NXP Hitag2, Legic Prime

Meanwhile, HID Prox – The card with the least security – Still has a reputation of being secure

Let us recap

- HID Prox cards can be read and emulated with a $20 device
- Reading distance is at least 20cm
- No crypto, no obfuscation, no protection; but: good lawyers

SECURITY RESEARCH LABS

Token Sub-Types

For the SAM (a.k.a. SAM63, a.k.a, Taufkarte‘), which ‚launches‘ readers (‚taufen‘), there is a counterpart – SAM64 (a.k.a ‚Enttaufkarte‘) to de-launch readers (‚enttaufen‘)

Other types (possibly restricted to advent)

- **XAM** Permanent permission to create segments (e.g. a launching version of IAM)
- **IAM+** Restricted version of IAM, which only allows to create a given number of segments

There are references to SAM4 ‚Parametrierkarte’, which changes reader parameters. Also some systems may use other ‚SAM... ‘ types for sneakernet purposes

SECURITY RESEARCH LABS