OSDP & PACS

BruCON 2023

Knud Højgaard / knud@fraktal.fi

Overview for this time slot

(Quick) PACS overview

Weaknesses in **P**hysical **A**ccess **C**ontrol **S**ystems

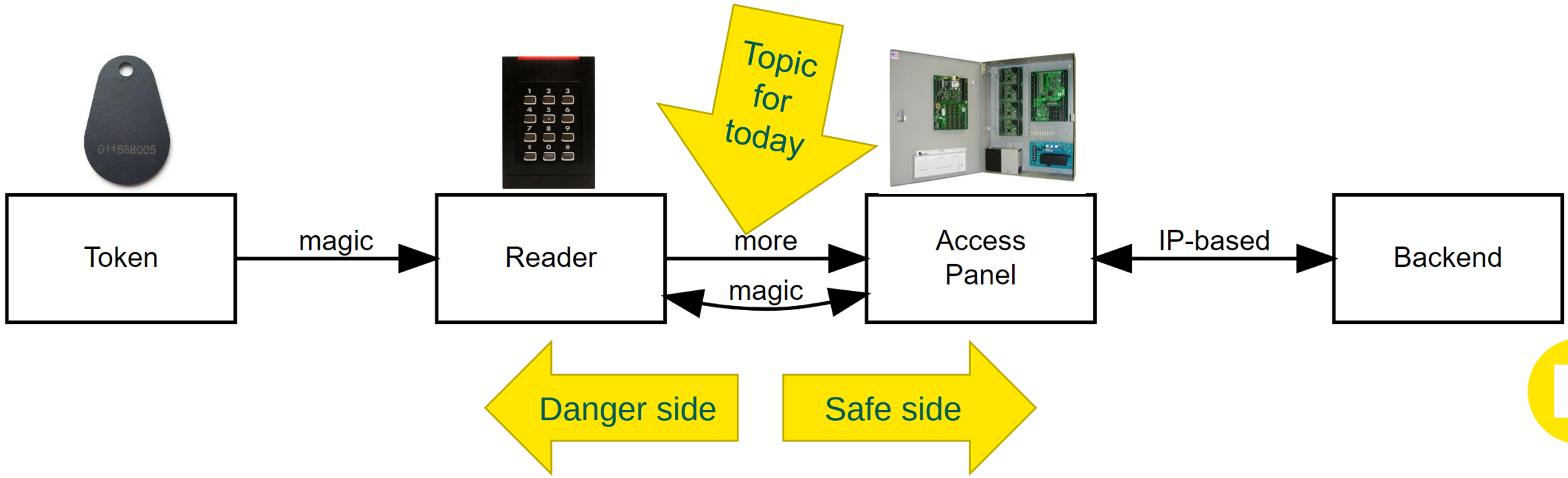
OSDP and the role it plays in PACS

How to audit OSDP installations & components

FRAKTAL

Physical Access Control System component overview

Custom, vendor-designed, magical black boxes full of proprietary stuff to guarantee lock-in vs. interoperable / standards-driven



FRAKTAL

Current PACS status = dumpster fire

- Cloneable low frequency tags
- Hackable / exploitable high frequency tags

- Exploitable readers
- Exploitable access panels
- Exploitable backends

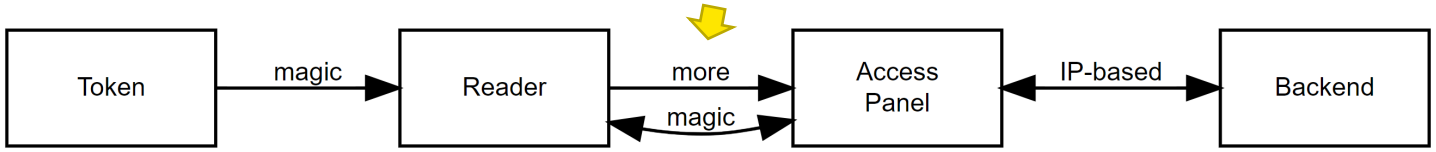
- Weak / legacy or misconfigured setups (MIFARE, Wiegand) still being installed

  Irresponsible and uncommitted installers, vendor lag

FRAKTAL

Wiegand

Wiegand is the “gold standard” for communications from reader to panel



The protocol is equivalent to plain text, sniffable and replayable to such an extent that there’s a commercial product to exploit it

ESPKey; $79 Wiegand interception tool

FRAKTAL

Modern PACS attack areas

Attack tokens

- Downside: must obtain token
- Cloning/duplicating & long-range reading of insecure tags main risk

Attack the reader; it's on the danger side

- Key extraction from the bike shed reader
- Reconfiguration, malicious firmware flashing
- EM attacks, DoS attacks

Attack communications to the access panel

- Recordings, replays, direct injection of manipulated token data
- Must be able to identify communication protocol on the wire

Attack the access panel or backend

- Requires network presence to attack via IP side, often objective of PACS tomfoolery is to gain network access, chicken||egg

FRAKTAL

Times are changing (slowly)

ESPKey is hopefully the final nail in the coffin for Wiegand

Enter **OSDP**
Open **S**upervised **D**evice **P**rotocol

With standards come *some* security, since test/edge cases can be made and tested for

More problems solved with this than just sniffing / relaying

FRAKTAL

Wiegand vs. OSDP V1 vs. V2

From unidirectional point to point bit spitting (Wiegand) to **a bus network** with addressing

OSDP V1 brings nothing interesting from a security perspective

(hopefully) not widely deployed

OSDP V2 is a **superset** of OSDP V1 and adds new functionality such as

- secure channel encryption (!)
- smart card passthrough communication
- biometric reader support

FRAKTAL

OSDP history + overview

OSDP V1 was jointly created in 2008 between HID Global, Mercury Security and Lenel. *

In 2012, the Security Industry Association (SIA) took ownership of the OSDP specification, and the SIA OSDP Working Group developed **OSDP V2**.

Standardized in IEC 60839-11-5:2020 (July 2020), still evolving, SIA up to version 2.2 now

- Free to use; standard document price ranges from €320 eur from iec.ch to $56 from normstream

Bidirectional communication, spoken on top of half-duplex **RS485**

- **1km cable runs, multi-drop bus network**

* plt-04025_a.0_-_hid-mercury_osdp_faq.pdf

FRAKTAL

OSDP verification

Mapping of mandatory functions in IEC 60839-11-1

The following Table F.1 to Table F.6 provide mapping of mandatory and optional requirements in IEC 60839-11-1 to OSDP specification.

Getting an OSDP verified reader signifies that it passes a set of tests

Mostly functional / conformance testing, with some very optimistic phrasing

6	Devices intended to be installed outside the controlled area or that could be accessible from outside the controlled area shall detect removal from mounting if that provides access to the internal elements and manipulation of these elements can cause an access granted condition	OP	OP	M	M	osdp_RSTATR

OSDP PROFILES

- **Basic**: These devices are Wiegand replacements; they provide the supervision benefits of a bidirectional protocol, protecting them from the common person-in-the-middle attacks.
- **Secure**: These devices meet the Basic profile but can also handle encrypted messages using Secure Channel and can enter and exit Basic and Secure modes as claimed.

FRAKTAL

OSDP terminology

We no longer have “readers” but “peripheral devices” (PD)

Multiple peripheral types, in order of complexity:
peripheral, basic, biometrics, extended packet mode

The access panel is now an access control unit (ACU) according to IEC, perhaps a CP according to SIA. Other than that, things are pretty much the same:

token/card -> PD <-> ACU <-> backend

FRAKTAL

OSDP V2 feature: smart card communication

Extended packet mode, OR transparent mode (hid/assa licensing encumbrance)

All card / security logic **theoretically** passed as-is to ACU on secure side of door

https://en.wikipedia.org/wiki/Contactless_smart_card

Comes with challenges in terms of over-the-internet relay attacks

ISO/IEC 14443 FWT of 77 milliseconds

1 verified vendor supports this (so far)

FRAKTAL

OSDP V2 feature: biometric reader support

Self-contained / isolated functionality in the protocol for some reason

Implemented as CMND 0x73, osdp_BIOREAD

1 verified vendor supports this (so far)

Table 24 – Biometric types

Value	Meaning
0x00	Not specified – default –
0x01	Right thumb print
0x02	Right index finger print
0x03	Right middle finger print
0x04	Right ring finger print
0x05	Right little finger print
0x06	Left thumb print
0x07	Left index finger print
0x08	Left middle finger print
0x09	Left ring finger print
0x0A	Left little finger print
0x0B	Right iris scan
0x0C	Right retina scan
0x0D	Left iris scan
0x0E	Left retina scan
0x0F	Full face image
0x10	Right hand geometry
0x11	Left hand geometry

Table 25 – Fingerprint formats

Value	Meaning
0x00	Not specified – use default method to scan, then report format used
0x01	Send raw fingerprint data as a PGM
0x02	ANSI/INCITS 378 Fingerprint template 84

FRAKTAL

OSDP V2 with secure channel briefly explained

OSDP secure channel is based on “secure channel protocol 03” by GlobalPlatform

OSDP standard (2.1.5):

Sample Secure Channel establishment session:

Sample Shared SCBK_D key = “303132333435363738393A3B3C3D3E3F”

This “sample key” has evolved into the default SCBK in 2.2

This default key is then used for protecting the channel during osdp_KEYSET events

- the spec requires the channel to be secure for keyset

an-321_configuring_ict_readers_for_osdp_communication.pdf

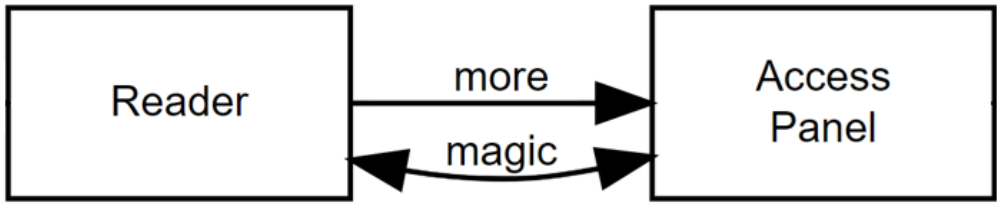
FRAKTAL

OSDP V2 communication flow

The reader/PD **cannot** initiate communication

The panel/ACU sends a steady stream of OSDP_POLL messages

Reader/PD events such as token reads are sent as responses to these polls



OSDP message format (header with CTRL & SCB)

Byte	Name	Meaning	Value
0	SOM	Start of Message	0x53
1	ADDR	Physical Address of the PD	0x00 – 0x7E
2	LEN_LSB	Packet Length Least Significant Byte	Any
3	LEN_MSB	Packet Length Most Significant Byte	Any
4	CTRL	Message Control Information	See Below

BIT	MASK	NAME	Meaning
0 - 1	0x03	SQN	Packet sequence number
2	0x04	CKSUM/ CRC	Set: 16-bit CRC; Clear: 8-bit CHECKSUM
3	0x08	SCB	Set: SCB is present; Clear: SCB is absent

Byte	Name	Meaning	Value
5	SEC_BLK_LEN	Length of Security Control Block	Any
6	SEC_BLK_TYPE	Security Block Type	Based on type
7 - m-1	SEC_BLK_DATA	Security Block Data	Based on type

Message header | CTRL block | sec ctrl block

OSDP message capture (SC DISABLED)

Previous slide omitted the actual message part and the trailing checksum or CRC

Byte 6 (if no SCB) holds the **command or reply** code

Sample poll-response without secure channel captured, easily decoded

PULL_HI	SOM	ADDR	LEN LSB	LEN MSB	CTRL	CMND	CRC	CRC
FF	0x53	**0x00**	0x08	0x00	0x07	**0x60**	0xb8	0xbf
FF	0x53	**0x80**	0x08	0x00	0x05	**0x40**	0x0a	0xf9
^^ covered in previous slide		^^ covered in previous slide		^^		^^ new	^^ new	^^ new

OSDP message capture (SC ENABLED)

Sample poll-response with secure channel enabled from test devices

HI	SOM	ADDR	LEN LSB	LEN MSB	CTRL	SEC_BLK_LEN	SEC_BLK_TYP	SEC_BLK_DAT	MAC	CRC
0xff	0x53	0x00	0x0e	0x00	0x0d	0x2	**0x15**	**0x60**	4cd24824	16c3
0xff	0x53	0x80	0x0e	0x00	0x0d	0x2	**0x16**	**0x40**	6f7e2898	fa85
0xff	0x53	0x00	0x0e	0x00	0x0e	0x2	**0x15**	**0x60**	2ef41207	ca7e
0xff	0x53	0x80	0x0e	0x00	0x0e	0x2	**0x16**	**0x40**	5364cd03	6377

SEC_BLK_TYP 15: req (panel to reader)

SEC_BLK_TYP 16: resp (reader to panel)

Secure channel is established, and a MAC is included

but the data field (SEC_BLK_DAT) is unencrypted

Secure channel is enabled, but encryption is not

OSDP sniffing & decoding

Async serial over RS485!

Once you have verified you are working with OSDP you can just do something like

stty -F /dev/ttyUSB0 raw 9600; modprobe usbmon
while true ; do cat /dev/ttyUSB0 > /dev/null ; done

Then you can use Wireshark coloring rules to make sense
of protocol flows and check if messages are flowing in plaintext

FRAKTAL

Wireshark decoding with colors

OSDP verified / compliant device:

@osdp RSTATR(usb.capdata[0-1]==ff:53 && usb.capdata[5]<8 &&
usb.capdata[6]==4B)@[57825,57825,57825][0,0,0]

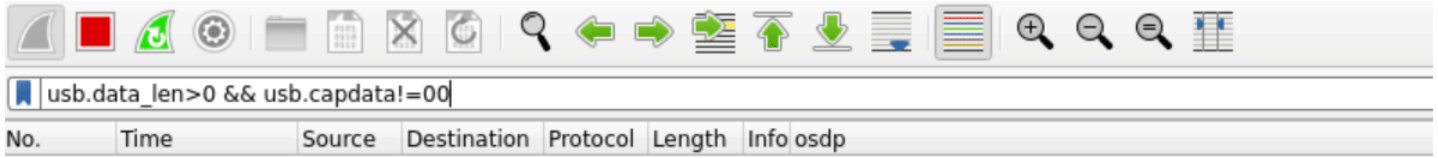
A device not pulling the wire high:

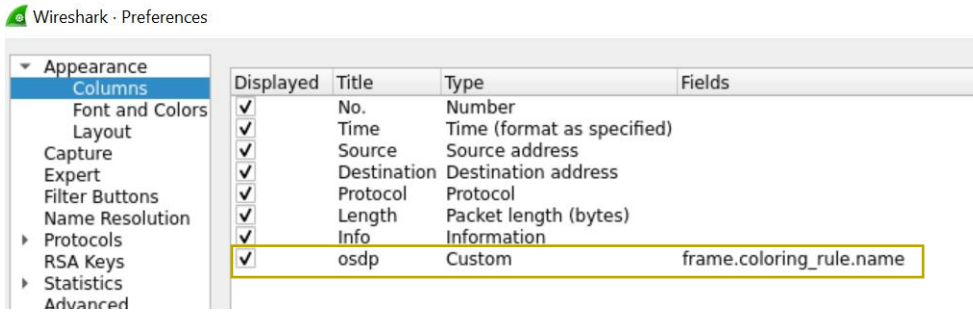
@osdp RSTATR@(usb.capdata[0]==53 && usb.capdata[4]<8 && usb.capdata[5]==4B)
@[57825,57825,57825][0,0,0]

Combined and unreadable:

@osdp RSTATR@(usb.capdata[0]==53 && usb.capdata[4]<8 && usb.capdata[5]==4B) ||
(usb.capdata[0-1]==ff:53 && usb.capdata[5]<8 &&
usb.capdata[6]==4B)@[57825,57825,57825][0,0,0]

Wireshark coloring





Free/open/available osdp tooling

osdp-python: https://github.com/ryanhz/osdp-python

OSDP.Net: https://github.com/bytedreamer/OSDP.Net

libosdp: https://github.com/goToMain/libosdp

libosdp-conformance: https://github.com/Security-Industry-Association/libosdp-conformance

- Has a sniffer which may be useful

Libosdp is (royalty) free – expect to see this as the base for some systems

FRAKTAL

Very basic OSDP comms framework

Libosdp & friends are ok if you are building things, less awesome for breaking things

Speak OSDP manually to be freed of the libosdp constraints

Implement the **crc** and/or **cksum** fields and some protocol practicalities such as sizers

```
def makepacket(pktdata):
        if type(pktdata) is str:
                payload = binascii.unhexlify(pktdata)
        else:
                payload = pktdata
        out = bytearray([0x53])                 # SOM / start of message
        out.extend(bytes.fromhex(addr))         # addr of pd or acu
        out.extend(pack('h', len(payload)+7))   # adding 7 to length (SOM, ADDR, LEN(2), BITFIELD, CRC(2))
        out.append(0x04)                        # lazy sqn bitfield
        out.extend(payload)                     # actual packet payload
        out.extend(pack('H',makecrc(out)))      # generate crc across entire packet
        return out
```

FRAKTAL

OSDP checksum

```
def makesum(input):
    if type(input) is str:
        input = binascii.unhexlify(input)
    data = bytearray(input)
    sum = 0xff # pull-high included in frame checksum calculation
    for i in range(len(data)):
        sum = sum + int(data[i])
    lsb = (sum).to_bytes(4, byteorder="big")[3]     # get lsb from sum
    cksum = pack("h",~lsb)[0]                       # 2's complement
    return cksum
```

OSDP CRC

```
def crc16_ccitt(crc, data):
    msb = crc >> 8
    lsb = crc & 255
    for c in data:
        x = c ^ msb
        x ^= (x >> 4)
        msb = (lsb ^ (x >> 3) ^ (x << 4)) & 255
        lsb = (x ^ (x << 5)) & 255
    return (msb << 8) + lsb

def makecrc(input):
        crc = crc16_ccitt(0x1D0F,input)
        return crc
```

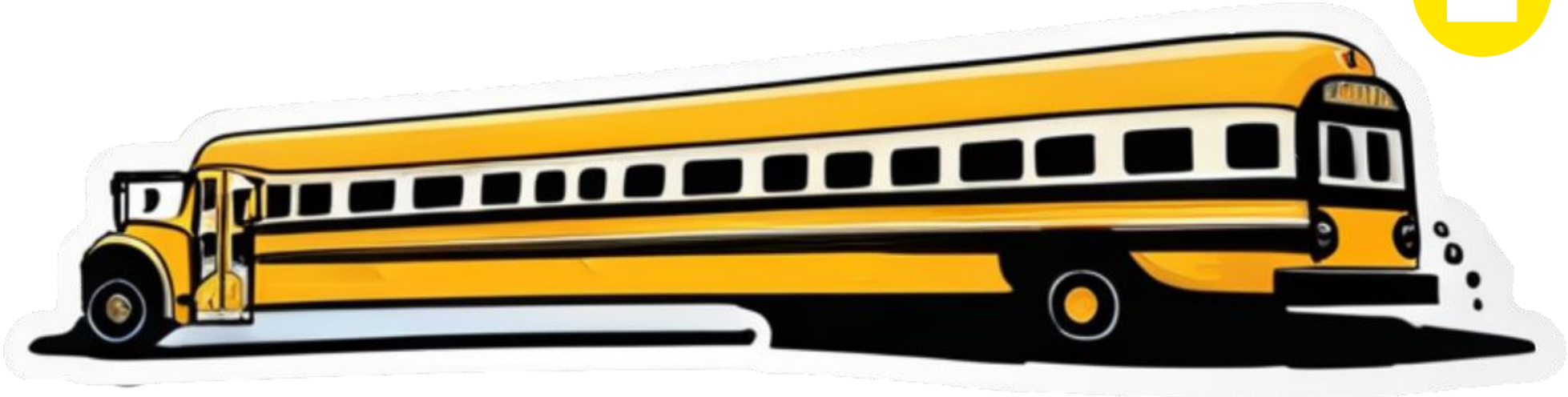

OSDP issues

Cable run problems

OSDP is multidrop and can handle ~127 devices on a single wire

This means there's very real potential for sending messages on the bus from somewhere along this potentially very long, up to 1km cable

- No longer just securing a single door, have to secure a long bus network



FRAKTAL

OSDP V2 problems

OSDP with secure channel still has plaintext metadata in the message passing, only (some) payload bodies are encrypted

Even when using secure channel, unencrypted messages are potentially still accepted

- **Implementation** & **configuration** specific

OSDP V2 without secure channel is **Wiegand 2.0** with tamper detect baked in

Exactly the same problems in terms of sniffing and replaying, no matter what SIA says in their marketing material

.. But more, new and complicated problems – firmware upgrades, reader messages

OSDP V2 master key scheme (use now discouraged)

OSDP 2.1.5:

SCBK = Enc(cUID || (~cUID), MK) // cUID is first 8 bytes of PDID response

To establish a secure connection between a CP and a PD, the PD presents its **cUID** in plain text. The CP performs the key diversification on the cUID and computes the PD's SCBK, thus establishing the common key for the secure session.

Creating key material by concatenating a shared secret and some public info is as good as deriving it from a shared secret

The shared secret has to be stored somewhere, in this case **outside the building**

OSDP attack: Reset the secure channel

The panel/ACU or reader/PD can invalidate an established session

- ACU by starting a new SC handshake
- PD by sending a NAK

NAK’s are 1-byte messages and a 4 byte MAC

Send a fake NAK to reset the channel to be able to capture initialization messages?

Not required; sending out of order data with a wrong mac or seqno may reset comms

Can also just choke the channel by spamming random data and waiting for an automated reset; may generate events, or sever the wire briefly; same outcome but different events

FRAKTAL

OSDP V2 attack: MAC implementation fails

Protocol uses a 4-byte MAC to prevent message spoofing, payload bitflips etc.

This *should* be MAC-then-decrypt

Possibility of implementations ignoring the MAC entirely (crc for integrity) and just decrypting & acting on payload, only checking 1 byte of the MAC, and so on and so on

A 4-byte MAC might seem small, in the context of a 9600-bps link it might be “ok”

FRAKTAL

OSDP V2 attack: Man in the middle with SCBK-D

The control panel establishes session keys inside a tunnel, maybe encrypted with **SCBK-D**

If SCBK-D is indeed in use, it is trivial to connect / place yourself on the bus and handle the connections for both sides, maybe with a single comms drop

Outcome: MITM capability; back to Wiegand / plain text security level

OSDP V2 pitfall: Install-mode & SCBK-D infoleak

A reader/PD in provisioning-time state has the default SCBK provided in the standard

In this state the ACU ***CAN*** set a new SCBK

When the new SCBK is set the pd ***SHOULD*** exit install mode

- Impossible to check without tampering a little with the system
- Must observe and decode entire communication flow from fresh boot

Sending an **osdp_PDCAP** message may reveal if the peripheral uses the SCBK-D

Described in IEC 60839-11-5:2020 Annex B:
This field is encoded to represent the key exchange capabilities 0x01 – (Bit-0) default AES128 key”

OSDP attacker challenges

Messages must fit inside the poll timing or risk triggering device offline alerts (poll timing * 2 -> 400ms)

The bus runs at **9600bps by default**; you have ~200ms to get a response in, including protocol overhead for simple messages

~(960 / 5)-~6 == room for ~186 chars

~(speed / timeslots because of OSDP poll timing) minus protocol overhead

- If you nail the timing **exactly**; if there is only 1 device on the bus

Constraints obviously apply to trigger as well as payloads

You are injecting onto an active bus!

Your payload might get clobbered by live responses

- Take over the connection entirely and prevent the device from sending responses for more accuracy


PD/READER attacks & auditing

Attacking & auditing readers

Key storage & vandalism/theft procedures

Shared keys between bike shed and vault?

Fallback modes for tokens

Lack of tamper-proof screws (lol)

Evaluation of tamper detection

- Is it easy to bypass?
- Does it work / is it connected and generate high priority alerts; does security staff react?
- Are your readers monitored via a camera, but not recording PIN codes? Who watches those videos?

Huge blind spot in terms of introspection capabilities for “regular auditors”

Readers drowned in potting compound, using obscure chips

No easy way of auditing readers, just have to trust the vendor

OSDP dangerous messages to peripherals

The standard supports some fixed length messages which are hard to get wrong from a technical perspective, but there's several messages which require thought and actual parsing; the ones that stand out as dangerous to me:

CMND	Desc	Danger
0x6b osdp_TEXT	Panel display control	Has length field for text string
0x74 osdp_BIOMATCH	Scan&match biometric template	Complex data, length fields
0x80 osdp_MFG	Manufacturer-specific	Who knows what's hidden here?
0xa4 osdp_GENAUTH	General authenticate	2 different sizers & 1 offset included
0xa5 osdp_CRAUTH	Challenge response auth	^^ same
0x7c osdp_FILETRANSFER	Fw update, conf changes etc.	Data "should" be sent in order; reassembly complex, content parsing, fw signatures?

PD attack: Basic fuzzer for display data

```
s_initialize("osdp_text")
s_byte(0x6b,name="CMND",fuzzable=False)
s_byte(0x00,name="reader_no")
s_byte(0x01,name="text_command", fuzz_values=[0x02,0x03,0x04])   # perm or temp, wrap or no wrap
s_byte(0x01,name="display_time")                                 # in seconds
s_byte(0x01,name="row")                                          # which row to show first char
s_byte(0x01,name="col")                                          # which column to show first char
s_size(block_name="text_string",length=1,fuzzable=True)
if s_block_start(name="text_string"):
        s_string("AAAABBBBCCCCDDDD")
        s_block_end(name="text_string")
```

boofuzz

PD/reader attack: Reconfigs (over OSDP)

Reconfiguring will affect availability

IF the reader accepts plain text commands over OSDP,

OSDP_comset a wrong baudrate or similar

-> reader drops out

-> installer checks and maybe just replaces

-> you have put in your device and capture the OSDP_keyset from the panel

-> victory

FRAKTAL

PD attack: Other interfaces

Managed using smartphone app or config cards

Supports everything under the sun in terms of fobs

OSDP key management & functionality less… flexible

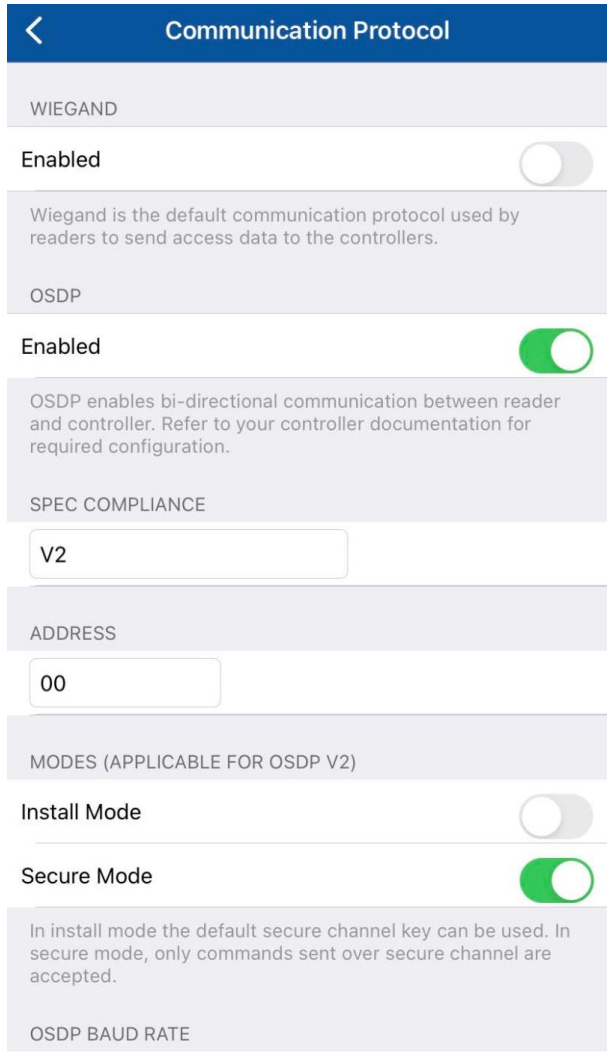
Install mode -> default 30313233… key

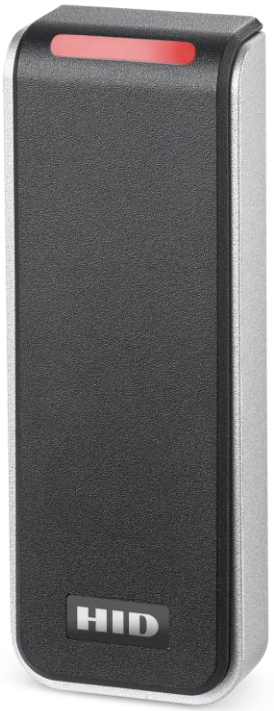
Secure mode -> “only secure channel”

No way of manually deploying a key during install

(potentially via special “configuration cards”)

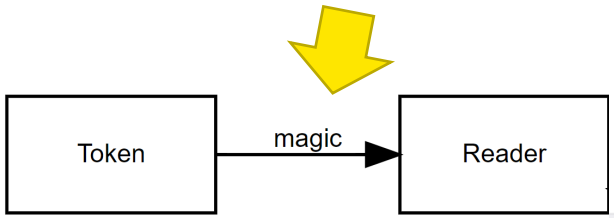
Must likely exchange keys protected by default key

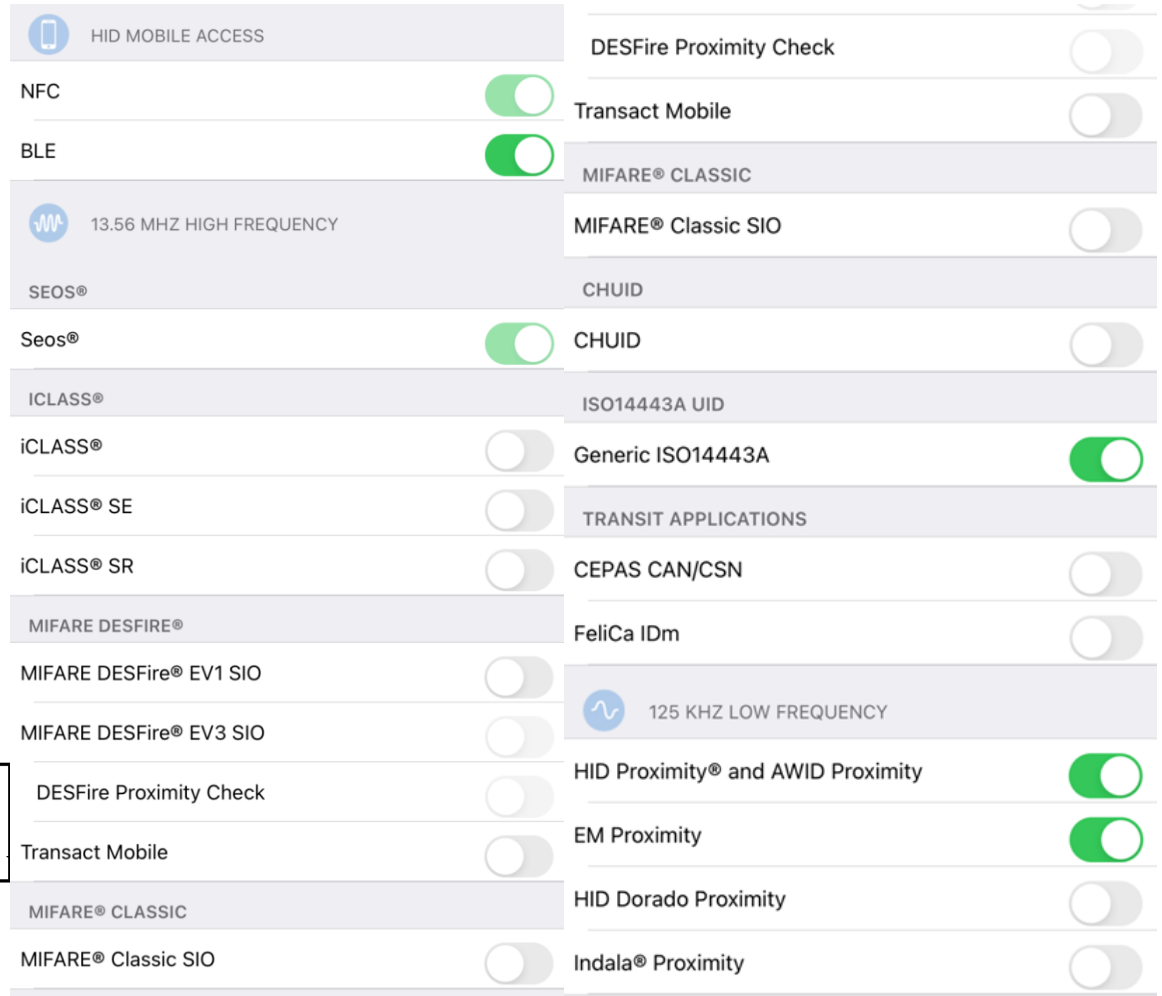




SIGNO 20 sidestep

Please check and disable all these things





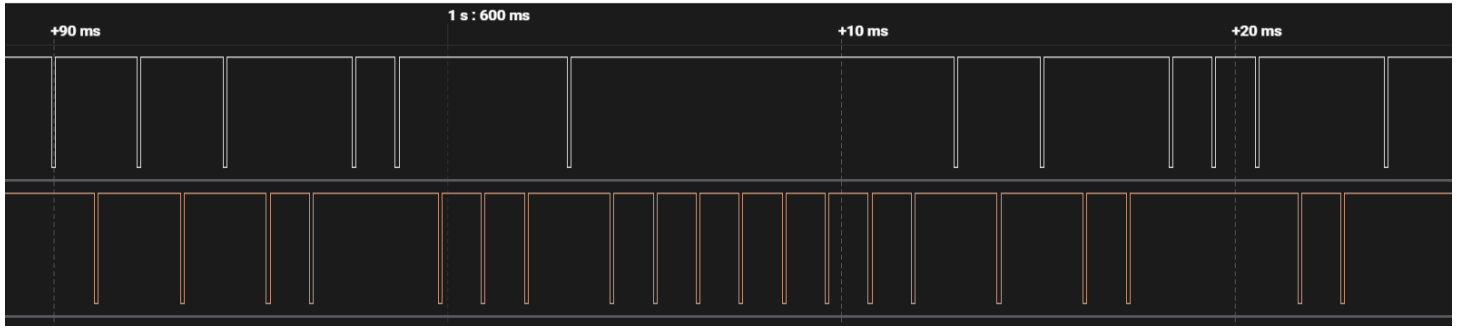

ACU/PANEL attacks & auditing

Auditing reader <-> ACU comms

Find reader wiring documentation!

Example for HID SIGNO: hook your logic analyzer to the green & white wires

Wiegand is easily recognizable and decodes with pulseview:



Clock & data will have constant pulses on 1, blips on other

OSDP + OSDP with SC is a matter of guessing baud rates on RS485 and inspecting the traffic as earlier described

Auditing ACU’s: OSDP attacks

The ACU/panel is now a multi-homed device with a leg on the internal side, and a cable dangling out the building on the danger side

Implementation-wise, the “first inside hop” / device is rs-485 all the way home, or has an IP side and OSDP side

It may be possible to register yourself as a full-blown PD to expand the attack surface

OSDP dangerous responses to panels

The PD/reader is actually less interesting as a target, compared to the ACU/panel.
Majority sent as poll responses - dangerous replies the panel must handle include:

CMND	Desc	Danger
0x50 osdp_RAW	Card data report	Length field
0x51 osdp_FMT	Card data report	Counter field
0x57 osdp_BIOREADR	Biometric read response	Length field, complex data
0x80 osdp_PIVDATAR	PIV data response	Multipart messaging, length fields
0x81 osdp_GENAUTHR	General auth response	Multipart messaging, length fields
0x82 osdp_CRAUTHR	Challenge response	Multipart messaging, length fields
0x83 0x84 0x90 osdp_MFG*	Manufacturer-specific	
… & more		

ACU attack: Basic fuzzer for card data parsing

```
s_initialize("osdp_raw")
s_byte(0x50,name="CMND",fuzzable=False)
s_byte(0x00,name="reader_no")
s_byte(0x00,name="format_code")              # 0 or 1 according to spec
s_size(block_name="card_data",length=2,endian='<',fuzzable=True)
if s_block_start(name="card_data"):
        s_string("AAAA")                     # normally just a number
        s_block_end(name="card_data")
```

FRAKTAL

ACU flaws: AXIS A1001 panel

Selected as test ACU for price & eBay availability, OSDP functionality without e.g. setting up Lenel OnGuard or other enterprise management solution

Investigation shows a MIPS CPU, Linux

OSDP handled by pacsiod, libpacsiod.so and libosdp.so

AXIS A1001

CVE-2023-21405 / CVSS 6.5
First public actual OSDP implementation vuln, wahey

Table 54 – Card data report, raw bit array (osdp_RAW)

Packet format field	Code	Name	Meaning
CMND	0x50	osdp_RAW	
DATA	0x00 – 0xFF	Reader number	0=First reader 1=Second reader
	0x00 = not specified, raw bit array 0x01 = P/data/P (wiegand)	Format code	Format of included data
	0x00 – 0xFF	Bit count (LSB)	2-byte size (in bits) of the data at the end of the record
	0x00 – 0xFF	Bit count (MSB)	
	0x00 – 0xFF	Data	8 bits of card data per data byte MSB to LSB (left justified)

Send an osdp_RAW packet of length 0 or 0xffff -> array indexing error leading to crash

CMND	rdr no	fmt code	BitC LSB	BitC MSB	data	result
0x50	00	00 (or 01)	00	00	<nothing>	Crash
0x50	00	00 (or 01)	ff	ff	<nothing>	Crash

Very fragile parser, many other problems leading to hangs & crashes

while true;do echo -ne "\xff\x53\x80"`dd if=/dev/urandom bs=1 count=16` >/dev/ttyUSB0; sleep 0.15;done

Upside; this works even with secure channel enabled / enforced

FRAKTAL

AXIS A1001 review remarks

Panel goes into sweep pattern if reader is disconnected

NO options for configuring secure channel in the end-user facing UI

NO key configuration options in regular UI

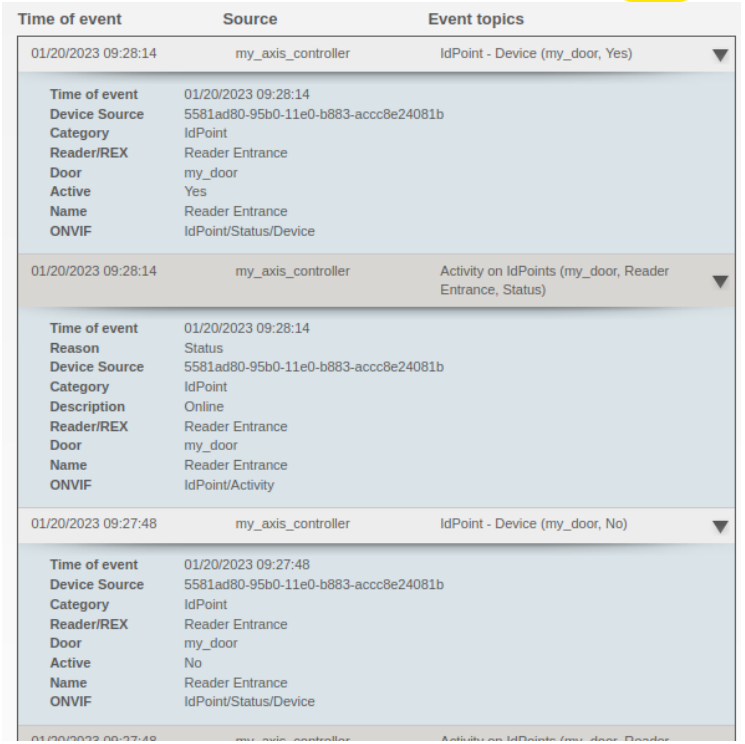
Can do it via convoluted API

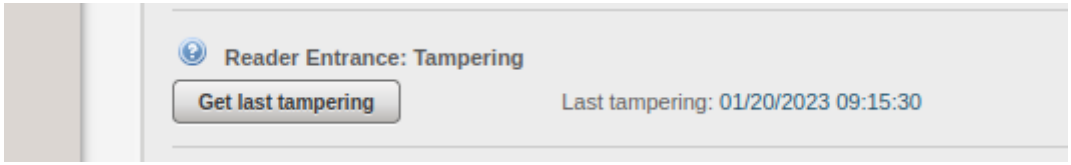
Tamper alerts hidden somewhere weird and need to be configured first but that's an OSDP tamper switch message, not me pulling wires.

Wire tampering / reader disconnect hidden under IdPoint -> Device; alert not very useful

NO detection of bus collisions; can just spam events

Not strictly OSDP conformant, does not pull line high **which breaks libosdp sniffing**





AXIS A1001 OSDP failures from the log

[WARNING] pacsiod[704]: 11:57:57.686636 (0x7ced00) OSDP NAK (EC=0x04) from reader at address 0x00 (state = 3).

[WARNING] pacsiod[704]: 11:57:57.689309 (0x7ced00) Unexpected sequence number, resetting to zero.

[WARNING] pacsiod[4850]: 14:56:42.062167 (0x75c08400) Incorrect SQN in response message.

[WARNING] pacsiod[704]: 11:59:35.765349 (0x7ced80) osdp_dev_int_message_timeout

[WARNING] pacsiod[704]: 11:59:52.348904 (0x7ced00) OSDP NAK (EC=0x01) from reader at address 0x00 (state = 3).

[WARNING] pacsiod[704]: 11:59:52.349946 (0x7ced00) CRC16 error, switching to 8-bit checksum.

[WARNING] pacsiod[704]: 11:59:57.447183 (0x7ced80) OSDP message crc-16 error, got 0x5380, calculated 0x8BC6

ACU flaws: <xxxx redacted>

Spamming random traffic on the bus causes the ACU to enter a failure state

Readers are silently offline, doors are closed (or open)

Fragile parser; possible to lock up ACU MCU with malformed OSDP traffic

- locked up to such an extent that the tamper switch loses function

ACU flaws: Cypress OSM-1000-BRD

Cheapest OSDP panel you can find. OSDP towards reader, Wiegand towards your legacy panel

Can work either as an access control unit or as a peripheral

Product - OSM-1000 - Cypress Integration Solutions / https://www.adiglobaldistribution.us/Product/ZE-OSM1000 (datasheets)

OSDP Default Parameters:

Secure Channel Base Key (SCBK): 30313233343536373839 3A3B3C3D3E3F

Polling Address: 0

Baud Rate: 9600

0123456789:;<=>?

SIA's Open Supervised Device Protocol (OSDP) v2.2.0 communication standard benefits

- **Security:** OSDP Secure Channel halts Wiegand hacking with AES-128 encryption

ACU flaws: Cypress OSM-1000-BRD (panel/ACU mode)

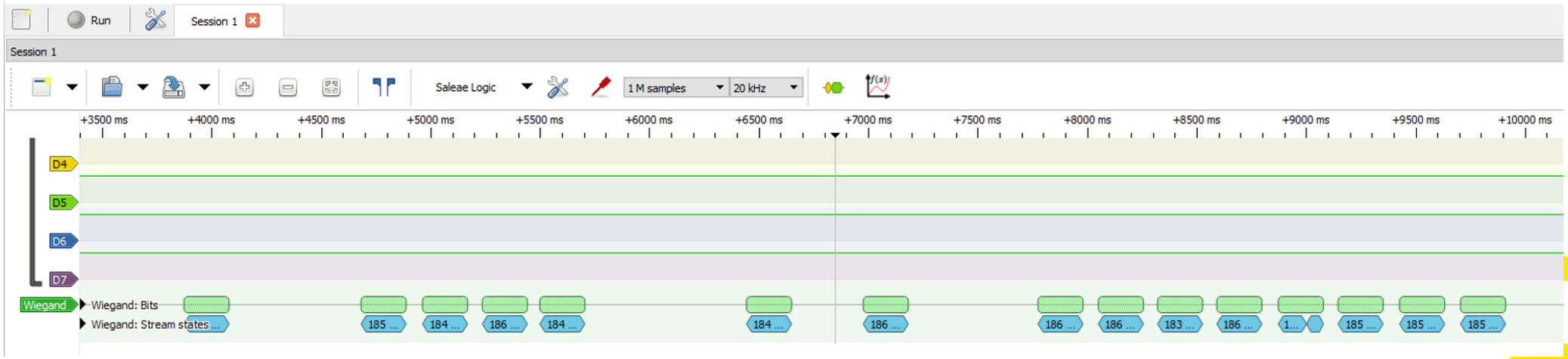
Timing issue; destroy secure channel, send unencrypted card read messages

- Passes as bueno to Wiegand side because SC does not exist yet

Documentation lacking; connect and cross your fingers

Device is too stupid, no way of detecting attacks except tamper

- This goes both ways; introspection is a lot of work
- Some crazy microcontroller on it I have no experience with




Conclusions & summary

OSDP & PACS status and future

OSDP -> OSDP v2 -> OSDP v2 with SC with master key -> OSDP v2 with SC with default SCBK (2020)

Creating standards people must pay to access is old hat and mainly assists failures (see SSCP)

OSDP over IP in the works, and a TLS version proposed

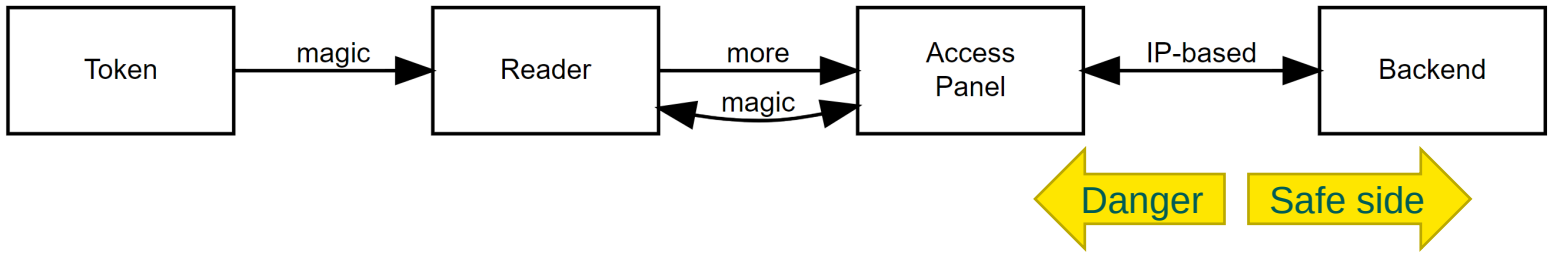
The future

FRAKTAL

Why was this interesting?

The attacker view is no longer confined to generic sniffing & replay of Wiegand or cloning of insecure tokens

The attack surface is moving closer to the enterprise and away from targeting end users or their tokens



Thank you!