Uncover more crypto secrets with Radare2

r2con2024

?E

Sylvain Pelissier

Applied Cryptographer @ Zellic

@ipolit@mastodon.social

Karim Sudki

IoT Security Expert @ Kudelski

@_Az0x_

About

- Memory analysis with Radare2
  - Pattern recognition
  - Recovery of cryptographic materials

- Show some evolutions since r2con 2020
  - *In radare2 /c means Cryptography*

- Demonstrate practical use cases in real-world situations

Searching with r2

/* all the things

- / - string
- /x - hex string
- /d - pattern
- /c - cryptography
- /h - hash
- /m - magic
- …

e search.?

```
    search.align: only catch aligned search hits
 search.badpages: scan and stop searching when finding bad pages
    search.chunk: chunk size for /+ (default size is asm.bits/8)
search.contiguous: accept contiguous/adjacent search hits
 search.distance: search string distance
search.esilcombo: stop search after N consecutive hits
    search.flags: all search results are flagged, otherwise only printed
    search.named: name flags with given string instead of search.prefix
 search.overlap: look for overlapped search hits
    search.kwidx: store last search index count
     search.show: show search results
 search.verbose: make the output of search commands verbose
     search.from: search start address
       search.to: search end address
  search.prefix: prefix name in search hits label
       search.in: specify search boundaries (raw, bin.sections.r/w/x,...)
 search.maxhits: maximum number of hits (0: no limit)
```

More complex search

- Perform actions on search results
  - e cmd.hit => execute command on hit

- How to perform more advanced searches ?
  - Logical pattern match (OR | AND | NOT)

News from plugins

r2yara

- YARA (Yet Another Recursive Acronym)
- Open-source tool
- Identification and classification of malwares
- Rules based on text or binary patterns
- Used by many tools (IDS, antivirus, MaskRomTool)
- Usable in radare2 via r2yara

YARA rules

```
rule Curve25519 {
    meta:
        author = "spelissier"
        description = "Basepoint and coefficients"
        date = "2023-03"
        reference= "https://www.rfc-editor.org/rfc/rfc7748.html#page-8"
    strings:
        $basepoint = {09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        $coefficient1 = {41 db 01 00} // The constant a24
        $coefficient2 = {42 db 01 00} // Go language uses a24 + 1
    condition:
        $basepoint and ($coefficient1 or $coefficient2)
}
```

r2yara

- Allows to load and run yara rules in r2
  - Install r2pm -ci r2yara
- Got its own repository:
  - https://github.com/radareorg/r2yara

```
[r2con@r2con ~]$ r2 spi.bin
[0x00000000]> yr-
INFO: Rules cleared
[0x00000000]> yr ./cryptography-yara-rules/ecc.yar
[0x00000000]> yrl
Curve25519
ecc_order
[0x00000000]> yrs
Curve25519
0x00095040: yara0.Curve25519_0 : 0900000000000000000000000000000000000000000000000000000000000000
0x0027b24a: yara0.Curve25519_1 : 0900000000000000000000000000000000000000000000000000000000000000
0x00037e84: yara0.Curve25519_2 : 42db0100
0x002132ac: yara0.Curve25519_3 : 42db0100
[0x00000000]> s yara0.Curve25519_
yara0.Curve25519_2   yara0.Curve25519_0   yara0.Curve25519_3   yara0.Curve25519_1
[0x00000000]> s yara0.Curve25519_0
[0x00095040]>
```

```
[0x00095040 [Xadvc]0 0% 384 spi.bin]> xc @ yara0.Curve25519_0
- offset -  4041 4243 4445 4647 4849 4A4B 4C4D 4E4F  0123456789ABCDEF  comment
0x00095040  0900 0000 0000 0000 0000 0000 0000 0000  ................  ; yara0.Curve25519_0
0x00095050  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095060  f7e9 7a2e 8d31 092c 6bce 7b51 ef7c 6f0a  ..z..1.,k.{Q.|o.
0x00095070  0000 0000 0000 0000 0000 0000 0000 0008  ................
0x00095080  429a a3ba 23a5 bfcb 115b 9dc5 7495 f3b6  B...#....[..t...
0x00095090  ffff ffff ffff ffff ffff ffff ffff ff07  ................
0x000950a0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000950b0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000950c0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000950d0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000950e0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000950f0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095100  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095110  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095120  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095130  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095140  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095150  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095160  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095170  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095180  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00095190  0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Where to find the rules ?

- Official Yara rules repository got COVID
  - https://github.com/Yara-Rules/rules
  - Latest commit 2 years ago
  - No PR merged anymore (15 opened)

- New repository for cryptographic rules
  - https://github.com/sylvainpelissier/cryptography-yara-rules
  - If you have nice rules, it will be integrated

Introducing yrg

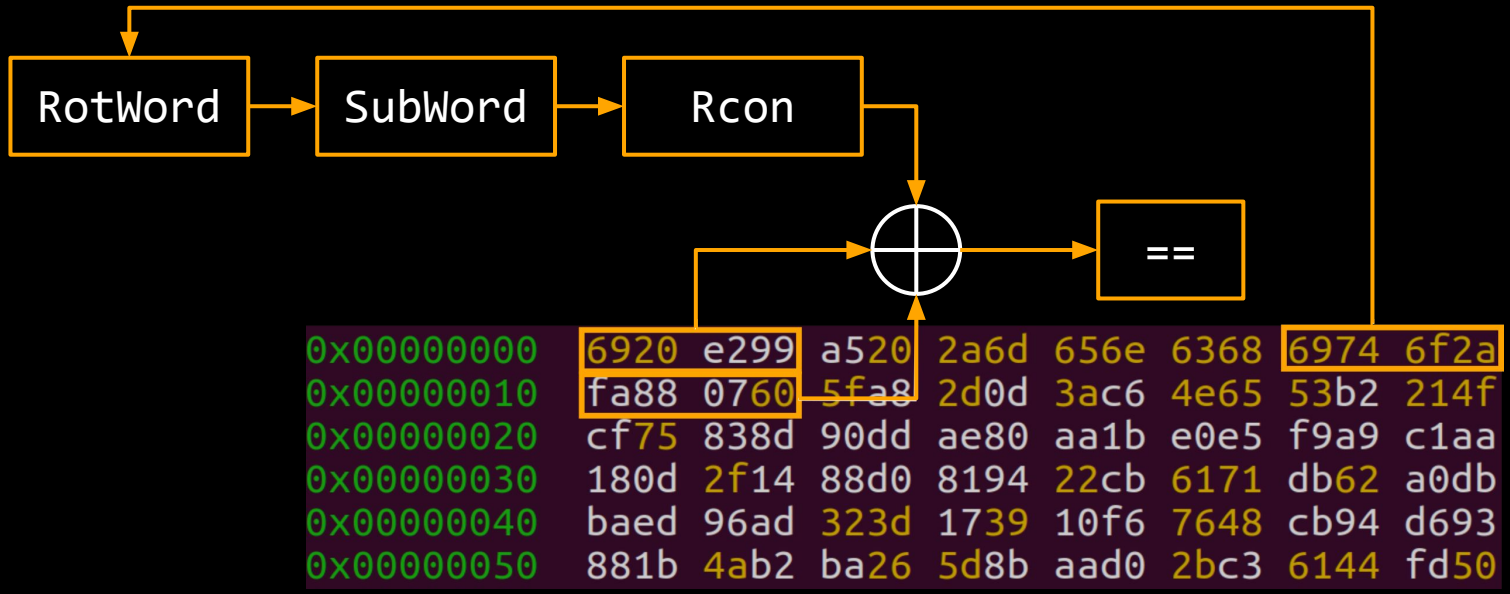
- Write your own rules while reversing !

```
Usage: yrg [action] [args..]   load and run yara rules inside r2
| yrg-           delete last pattern added to the yara rule
| yrg-*          delete all the patterns in the current rule
| yrgs ([len])   add string (optionally specify the length)
| yrgx ([len])   add hexpairs of blocksize (or custom length)
| yrgf ([len])   add function bytepattern signature
| yrgz           add all strings referenced from current function
```

```
    version = "0.1"
}
[0x00132700]> yrgs 16
[0x00132700]> yrg
rule rulename : test {
  meta:
    author = "r2con"
    description = "My first yara rule"
    date = "2024-10-20"
    version = "0.1"
  strings:
    $ = "expand 32-byte k"
  condition:
    all of them
}
[0x00132700]> e yara.rule = my_new_rule
[0x00132700]> yr+
INFO: Rule successfully added
[0x00132700]> yrl
my_new_rule
[0x00132700]> yrs
my_new_rule
0x00132700: yara0.my_new_rule_0 : 657870616e642033322d62797465206b
[0x00132700]>
```

News from symmetric crypto

Remember AES key schedule search ?

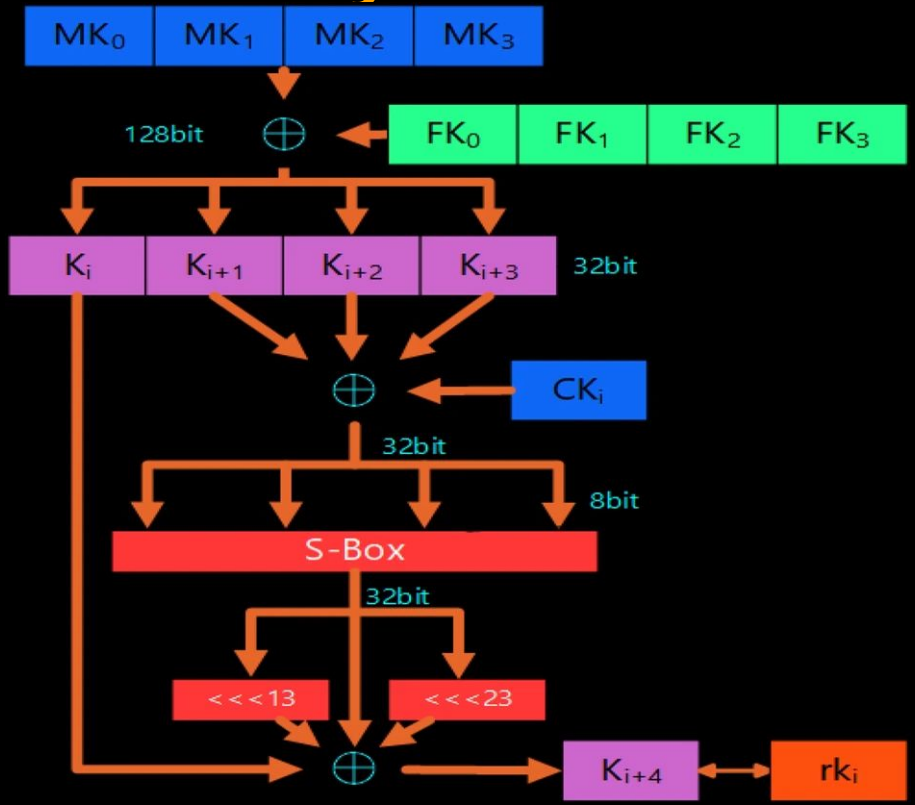


Available using /ca command

SM4 key schedule search

- /ca [algo] can be used to search SM4 key schedule

SM4 key recovery



SM4 key recovery

```
[0x00000000]> /ca sm4

INFO: Master key found: 0123456789abcdef123456789abcdef0 @    0xff
0x000000ff hit0_0 f98621f1612b6641db28e44757dbe32c
```

Print encryption decryption

- Existing commands woD and woE
  - Write encryptions or decryptions
- New commands poE and poD for printing only
  - Usage: poE [algo] [key] [iv]

Print encryption decryption

```
[0x00000000]> poE
c      aes-ecb  Block Cipher Mode for Rijndael Encryption
c      aes-cbc  Cipher Block Chaining Mode for Rijndael Encryption
c     aes-wrap  Advanced Encryption Standard Key Wrap Algorithm (RFC 3394)
c     blowfish  Bruce Schneier's symetric-key block cipher
c         cps2  Capcom Play System 2
c      des-ecb  Simplest and weakest Electronic Code Book for DES
c          rc2  Ron Rivest's Code symmetric key encryption also known as ARC2
c          rc4  Rivest Cipher 4
c          rc6  Rivest's Cipher 6
c  serpent-ecb  Bouncy Castle Cryptography
c      sm4-ecb  ShāngMì4 block cipher
c          xor  Byte level Exclusive Or Encryption
| poE [algo] [key] [iv]  Print block encryption
```

Practical application

- Target (PC)
  - Linux OS
  - Enabled security features
    - Secure Boot
    - Full Disk encryption w/TPM (Trusted Platform Module)

- Goal
  - Decrypt disk content for fun and profit !

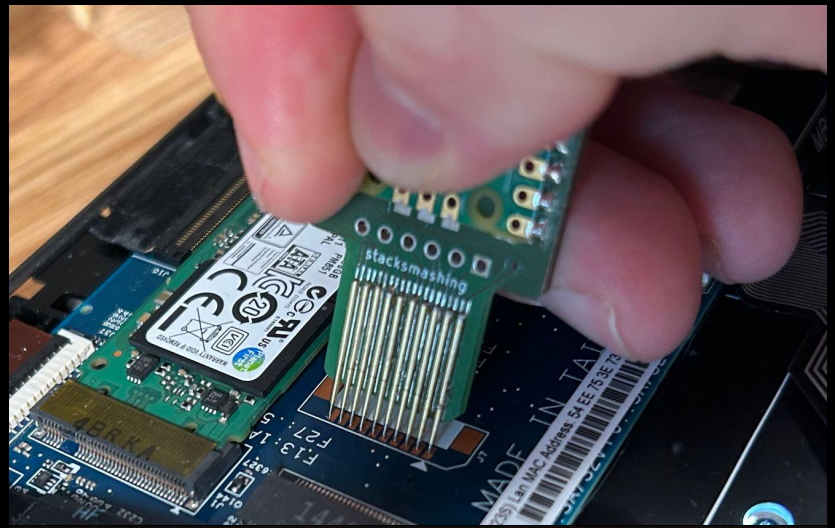
TPM 101

- TPM stores a sealed key or passphrase (!= Master Key)
- During boot, BIOS/UEFI and OS bootloader measure components
- Extend the hash value of some Platform Configuration Registers
  - PCR0 = Core system firmware executable code
  - PCR1 = Core system firmware data/host platform configuration

- Key is released only if current PCRs match the ones at seal time
- Any unauthorized changes prevent TPM from releasing the key

Disk Encryption key recovery

- Simple, Low-Cost Attack
- Solder wires to TPM pins
  - Speed-up with tools
- Monitor signals at boot
  - Wait for Unseal command



Disk Encryption key recovery

- But...
  - Target uses fTPM (Intel PTT)
  - Where to put the probe ?

Smart move :)

Really ?

Luckily

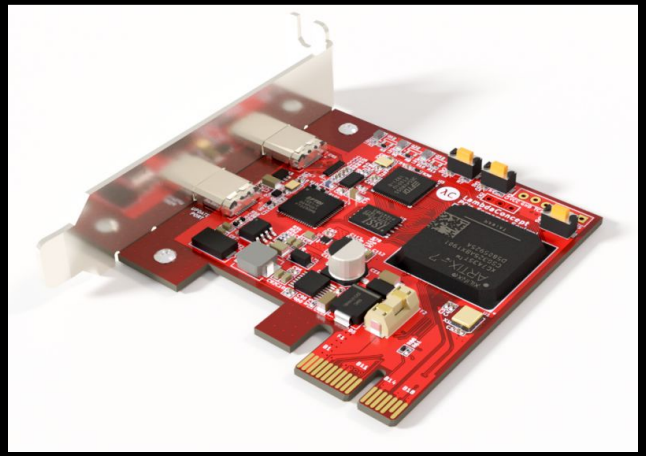
- For some reasons…
  - ✅ BIOS password not set
  - ✅ BIOS settings (PCR #1) not measured…(only 0,2,7)

- Which BIOS settings to modify ?
  - Secure boot ?
    - ❌ Secureboot status and list of enrolled keys (PCR #7)

  - Intel VT-d ?
    - ✅ Disables the IOMMU
    - Note: Not required if booting UKI without `intel_iommu=on`

- Direct Memory Access (DMA) attack ftw !

Dump memory



- Plug PCI Leech in the target
- Boot the system
- Dump entire memory

```
pcileech dump -out dump.img -v -vv -device fpga
```

- Use /ca aes|sm4 to find key expansion(s) in memory
  - Decrypt the disk with the recovered keys
- Demo ! (replicated setup on VMs)

```
[r2con@r2con ~]$ r2 -e search.from=0x27000000 -e search.maxhits=2 -n aes_dump.raw
[0x00000000]> /ca aes
[# ]0x29d7c234 hit0_0 d1d581b9d605beb7f629f771683f3b4d5875e766c90d9fdc0b17e78bafeab3d2
0x29d7c424 hit0_1 eba04af0b4455da3db925df46a65a2d73d752cffefdf09471a2e9c60e270fefe
[0x00000000]>
```

```
LUKS header information for aes.img
Cipher name:    aes
Cipher mode:    xts-plain64
Payload offset: 32768
UUID:           c68a3485-bf79-45a4-8378-458558a2d7f6
MK bits:        512
MK dump:        eb a0 4a f0 b4 45 5d a3 db 92 5d f4 6a 65 a2 d7
                3d 75 2c ff ef df 09 47 1a 2e 9c 60 e2 70 fe fe
                d1 d5 81 b9 d6 05 be b7 f6 29 f7 71 68 3f 3b 4d
                58 75 e7 66 c9 0d 9f dc 0b 17 e7 8b af ea b3 d2
```

```
[r2con@r2con ~]$ r2 -e search.from=0x23000000 -e search.maxhits=2 -n sm4_dump.raw
[0x00000000]> /ca sm4
[# ]INFO: Master key found: 7b7b8f3520a4f9b95d7686c220ebd09c @0x23914224
[# ]INFO: Master key found: 75e012fceaa98c84dfb6d7721404cd13 @0x23976c2c
0x23914224 hit0_0 7908d45834bc2457371501453bb1f5dd
0x23976c2c hit0_1 280793659dc9c2e2234528c231f23e72
[0x00000000]>
```

```
LUKS header information for sm4.img
Cipher name:    sm4
Cipher mode:    xts-plain64
Payload offset: 32768
UUID:           00b98fcd-5407-4e30-bbeb-810b23e91f75
MK bits:        256
MK dump:        75 e0 12 fc ea a9 8c 84 df b6 d7 72 14 04 cd 13
                7b 7b 8f 35 20 a4 f9 b9 5d 76 86 c2 20 eb d0 9c
```

Results

- Recover the disk encryption key
- Dump content of the disk
- Reverse engineer of some binaries
  - Discovery of multiple RCE

News from asymmetric crypto

More r2 commands

- Search commands
  - /cd ASN.1/DER certificates
  - /cr ASN.1/DER private keys (RSA and ECC)
  - /cg GPG/PGP   keys and signatures
- Decode commands
  - pFa[jqt] [len]   print decoded ASN.1/DER
  - pFo[j]   [len]   print decoded ASN.1 OID
  - pFp[j]   [len]   print decoded PKCS7
  - pFx[j]   [len]   print decoded X509

Real-world scenario

- IoT device
- ARM microcontroller
- Unprotected console (UART)
- Command to read/write to arbitrary locations
  - Dump microcontroller firmware
  - Dump external memory

External memory

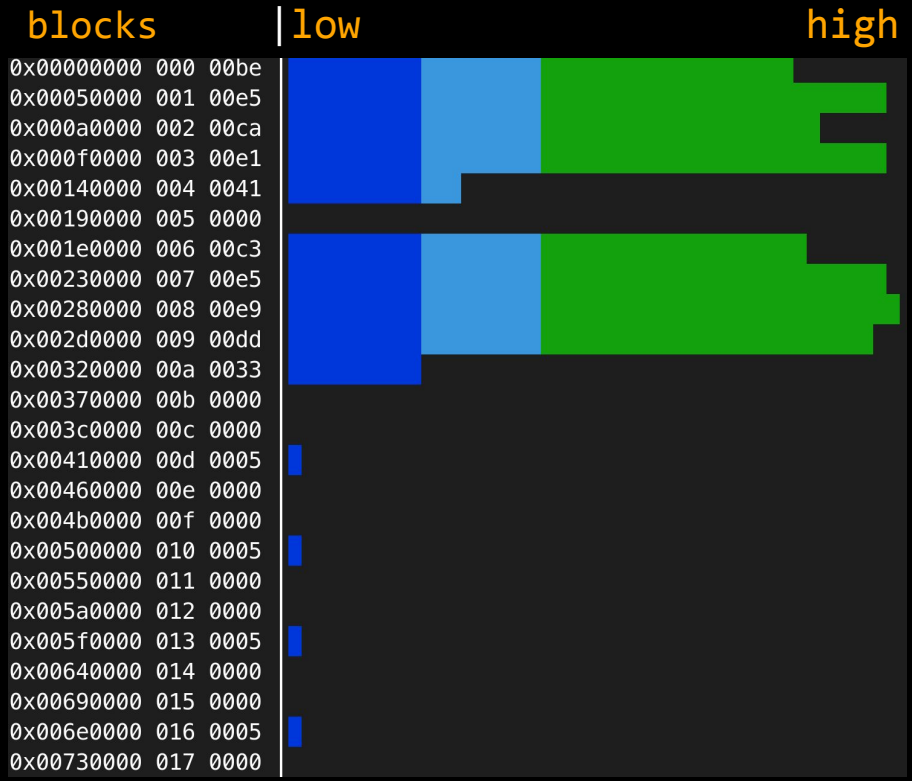
- Often stores sensitive data
- Device connected to the cloud via Wifi
- Time to hunt for certificates !!

```
[r2con@r2con ~]$ r2 -n spi.bin
[0x00000000]> b 1000
[0x00000000]> /cr
0x0009b5fc hit0_0 304f020100301c311a301806035504030c11302a300506032b6570032100a000300506032b657003...
0x002803a6 hit0_1 304f020100301c311a301806035504030c11a000300506032b65700341003a206d697373696e6700...
[0x00000000]> pFa @hit0_0
  OFFSET  HDR +    OBJ DEPTH FORM NAME                : VALUE
       0  0x2 +   0x4f     0 cons SEQUENCE            : 30
     0x2  0x2 +    0x1     1 prim INTEGER             : 00
     0x5  0x2 +   0x1c     1 cons SEQUENCE            : 30
     0x7  0x2 +   0x1a     2 cons SET                 : 31
     0x9  0x2 +   0x18     3 cons SEQUENCE            : 30
     0xb  0x2 +    0x3     4 prim OBJECT_IDENTIFIER   : commonName
    0x10  0x2 +   0x11     4 prim UTF8String          : 0*0...+ep.!
    0x23  0x2 +   0x2b     1 prim BIT_STRING          : 700341006373723d00686d61633d002d2d2d2d2d424547494e204345525449464943415445205245515545 (235 bits)

[0x00000000]> pFa @hit0_1
  OFFSET  HDR +    OBJ DEPTH FORM NAME                : VALUE
       0  0x2 +   0x4f     0 cons SEQUENCE            : 30
     0x2  0x2 +    0x1     1 prim INTEGER             : 00
     0x5  0x2 +   0x1c     1 cons SEQUENCE            : 30
     0x7  0x2 +   0x1a     2 cons SET                 : 31
     0x9  0x2 +   0x18     3 cons SEQUENCE            : 30
     0xb  0x2 +    0x3     4 prim OBJECT_IDENTIFIER   : commonName
    0x10  0x2 +   0x11     4 prim UTF8String          : .
    0x23  0x2 +   0x2c     1 cons Application_19      :  (840 bits)
    0x25  0x2 +   0x2a     2 cons Application_14      :  (824 bits)
    0x27  0x2 +   0x28     3 prim EOC                 :  (784 bits)

[0x00000000]> /cd
0x000009b0 hit1_0 308201093081bca0030201
[0x00000000]>
```

```
[0x00000000]> pFa @hit1_0
  OFFSET  HDR +    OBJ DEPTH FORM NAME                : VALUE
       0  0x4 +  0x109     0 cons SEQUENCE            : 30
     0x4  0x3 +   0xbc     1 cons SEQUENCE            : 30
     0x7  0x2 +    0x3     2 cons Context [0]         :
     0x9  0x2 +    0x1     3 prim INTEGER             : 02
     0xc  0x2 +   0x10     2 prim INTEGER             : 2ab59455b5d0dc96101ed8a52fb5e9ac
    0x1e  0x2 +    0x5     2 cons SEQUENCE            : 30
    0x20  0x2 +    0x3     3 prim OBJECT_IDENTIFIER   : id-Ed25519
    0x25  0x2 +   0x11     2 cons SEQUENCE            : 30
    0x27  0x2 +    0xf     3 cons SET                 : 31
    0x29  0x2 +    0xd     4 cons SEQUENCE            : 30
    0x2b  0x2 +    0x3     5 prim OBJECT_IDENTIFIER   : commonName
    0x30  0x2 +    0x6     5 prim UTF8String          : MY_CA_
    0x38  0x2 +   0x20     2 cons SEQUENCE            : 30
    0x3a  0x2 +    0xd     3 prim UTCTime             : 03/08/2023 00:00:00 GMT
    0x49  0x2 +    0xf     3 prim GeneralizedTime     : 02/08/2103 23:59:59 GMT
    0x5a  0x2 +   0x1c     2 cons SEQUENCE            : 30
    0x5c  0x2 +   0x1a     3 cons SET                 : 31
    0x5e  0x2 +   0x18     4 cons SEQUENCE            : 30
    0x60  0x2 +    0x3     5 prim OBJECT_IDENTIFIER   : commonName
    0x65  0x2 +   0x11     5 prim PrintableString     : aa:bb:cc:dd:ee:ff
    0x78  0x2 +   0x2a     2 cons SEQUENCE            : 30
    0x7a  0x2 +    0x5     3 cons SEQUENCE            : 30
    0x7c  0x2 +    0x3     4 prim OBJECT_IDENTIFIER   : id-Ed25519
    0x81  0x2 +   0x21     3 prim BIT_STRING          : f0d889ba13bd559c1fa82e32ca784b0bce14bd6ebfb2c614d56654ad98be6006a3 (256 bits)
    0xa4  0x2 +   0x1d     2 cons Context [3]         : 1b300c0603551d130101ff04023000300b0603551d0f0404030205e030 (176 bits)
    0xa7  0x2 +   0x1a     3 prim GeneralString       : 0c0603551d130101ff04023000300b0603551d0f0404030205e0 (384 bits)
    0xc3  0x2 +    0x5     1 cons SEQUENCE            : 30
    0xc5  0x2 +    0x3     2 prim OBJECT_IDENTIFIER   : id-Ed25519
    0xca  0x2 +   0x41     1 prim BIT_STRING          : 9fec0ebe966ca24fb1e5e555f14b010ecb590332f49a15bdabbe2b0a70ed437f2a159c8f2d1a91ba3bc0eb1de627f4bed0e
0... (512 bits)
```

Now what ?

- /cr No private key found ;(
- p=e Entropy on memory dump is low
- Reverse engineering of the firmware
  - Pointers to external memory
  - Not ASN.1 format
- How to find the private key?



Ed25519 signature in a nutshell

- Public-key signature system
  - EdDSA ≠ ECDSA
  - No dependency on random number generator
- Fast
  - Key generation
  - Signature verification (up to 64 in batch)
- Small
  - Keys (32 bytes)
  - Signatures (64 bytes)

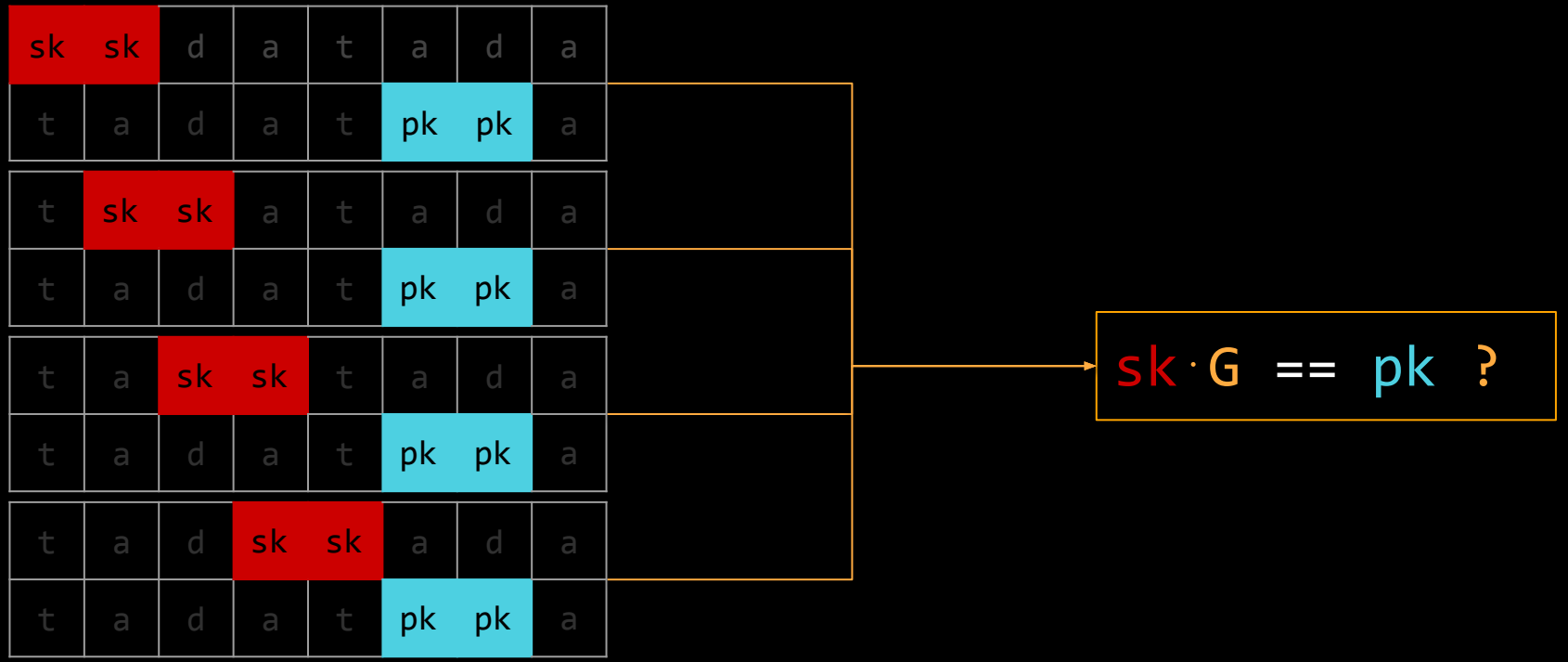
Searching private key from public

- The relation between a Ed25519 public key (pk) and its private key (sk) is:

$$pk = sk \cdot G$$

- Where G is the base point on the elliptic curve

Searching private key from public



Searching private key from public

sk	sk	d	a
t	a	d	a
t	sk	sk	a
t	a	d	a
t	a	sk	sk
t	a	d	a
t	a	d	sk
t	a	d	a

= pk ?

Python to the rescue !

init pk

load sk

generate pk'

pk == pk' ?

```python
from cryptography import x509
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.serialization import load_der_private_key
from cryptography.hazmat.primitives.asymmetric import ed25519


cert_data = open("public.der","rb").read()
public_key = x509.load_der_x509_certificate(cert_data, default_backend()).public_key()
public_key_bytes = public_key.public_bytes(encoding=serialization.Encoding.Raw,format=serialization.PublicFormat.Raw)


binary_data = open("spi.bin","rb").read()
for i in range(0, len(binary_data)+1 - 32, 1):
    private_key_bytes = binary_data[i:i + 32]
    try:

        private_key = ed25519.Ed25519PrivateKey.from_private_bytes(private_key_bytes)


        generated_public_key = private_key.public_key()
        test = generated_public_key.public_bytes(encoding=serialization.Encoding.Raw,format=serialization.PublicFormat.Raw)

        if public_key_bytes == test:
            print("FOUND")
            print(f"offset = {hex(i)}")
            print(private_key_bytes.hex())
```

```
FOUND
offset = 0x98a
0842f389403daa889b5e919587af4b47a330f2848c9536170a8f0852342c6d5a
```

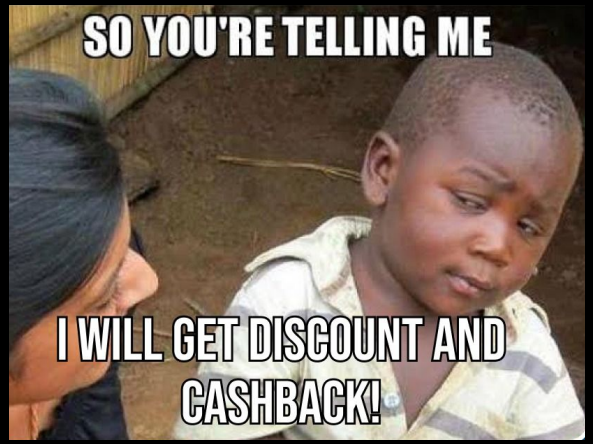
What if...

- such feature could be useful to others...
- Feature request to ~~r2 crypto team~~ @ipolit
  - Let’s add another command to r2 !
  - PR merged (#23195)

Introducing /cp

- Find private key from public key within r2
  - /cp [algo] [pubkey]

- For now, only Ed25519 supported
  - As always... PR welcome !
  - Plan to support SSL algorithms
  - 100% money back guarantee



- Coming with another new commands:
  - Print signature of a block (poS [algo] [key])

```
    0x4  0x3 +   0xbc     1 cons SEQUENCE           : 30
    0x7  0x2 +    0x3     2 cons Context [0]        :
    0x9  0x2 +    0x1     3 prim INTEGER            : 02
    0xc  0x2 +   0x10     2 prim INTEGER            : 2ab59455b5d0dc96101ed8a52fb5e9ac
   0x1e  0x2 +    0x5     2 cons SEQUENCE           : 30
   0x20  0x2 +    0x3     3 prim OBJECT_IDENTIFIER  : id-Ed25519
   0x25  0x2 +   0x11     2 cons SEQUENCE           : 30
   0x27  0x2 +    0xf     3 cons SET                : 31
   0x29  0x2 +    0xd     4 cons SEQUENCE           : 30
   0x2b  0x2 +    0x3     5 prim OBJECT_IDENTIFIER  : commonName
   0x30  0x2 +    0x6     5 prim UTF8String         : MY_CA_
   0x38  0x2 +   0x20     2 cons SEQUENCE           : 30
   0x3a  0x2 +    0xd     3 prim UTCTime            : 03/08/2023 00:00:00 GMT
   0x49  0x2 +    0xf     3 prim GeneralizedTime    : 02/08/2103 23:59:59 GMT
   0x5a  0x2 +   0x1c     2 cons SEQUENCE           : 30
   0x5c  0x2 +   0x1a     3 cons SET                : 31
   0x5e  0x2 +   0x18     4 cons SEQUENCE           : 30
   0x60  0x2 +    0x3     5 prim OBJECT_IDENTIFIER  : commonName
   0x65  0x2 +   0x11     5 prim PrintableString    : aa:bb:cc:dd:ee:ff
   0x78  0x2 +   0x2a     2 cons SEQUENCE           : 30
   0x7a  0x2 +    0x5     3 cons SEQUENCE           : 30
   0x7c  0x2 +    0x3     4 prim OBJECT_IDENTIFIER  : id-Ed25519
   0x81  0x2 +   0x21     3 prim BIT_STRING         : f0d889ba13bd559c1fa82e32ca784b0bce14bd6ebfb2c614d56654ad98be6006a3 (256 bits)
   0xa4  0x2 +   0x1d     2 cons Context [3]        : 1b300c0603551d130101ff04023000300b0603551d0f0404030205e030 (176 bits)
   0xa7  0x2 +   0x1a     3 prim GeneralString      : 0c0603551d130101ff04023000300b0603551d0f0404030205e0 (384 bits)
   0xc3  0x2 +    0x5     1 cons SEQUENCE           : 30
   0xc5  0x2 +    0x3     2 prim OBJECT_IDENTIFIER  : id-Ed25519
   0xca  0x2 +   0x41     1 prim BIT_STRING         : 9fec0ebe966ca24fb1e5e555f14b010ecb590332f49a15bdabbe2b0a70ed437f2a159c8f2d1a91ba3bc0eb1de627f4bed0e
0... (512 bits)

[0x000009b0]> s+0x81+0x2+0x1
[0x00000a34]> /cp ed25519 `p8 32`
0x0000098a hit3_0 0842f389403daa889b5e919587af4b47a330f2848c9536170a8f0852342c6d5a
[0x00000a34]>
```

Results

- Recover public and private certificates
- Impersonate the device against cloud
  - Retrieve new firmware updates
  - Fake device telemetry
  - Further attacks at scale

Conclusion

- Radare2 as an all-in-one tool
- Evolution of cryptography related commands
- Demonstration on real-world situations

- Thank you !

Questions