Cryptography Workarounds For Law Enforcement

Snake oil Crypto, D4 and other tricks

Team CIRCL

CIRCL
Computer Incident
Response Center
Luxembourg

2019/11/27

Jean-Louis Huynen

Outline

- Cryptography 101,
- Brute Force 101,
- Encryption an Law Enforcement,
- Pretty Good Privacy / GnuPG
- Use-Case: RSA,
- First Hands-on: Understanding RSA,
- Snake-Oil-Crypto: a primer,
- Second Hands-on: RSA in Snake-Oil-Crypto,
- D4 passiveSSL Collection,
- Interactions with MISP.

Cryptography 101

Cryptography Concepts

- **Plaintext** P: Text in clear,
- **Encryption** E: Process of disguising the plaintext to hide its content,
- **Ciphertext** C: Result of the Encryption process,
- **Decryption** D: Process of reverting encryption, transforming C into P,
- **Encryption Key** EK: Key to encrypt P into C,
- **Decryption Key** DK: Key to decrypt C into P,
- **Cryptanalysis**: Analysis of C to recover P without knowing K.

Cryptography Services

- **Confidentiality** : Ensure the secrecy of the message except for the **intended** recipient,
- **Authentication** : Proving a party's identity,
- **Integrity** : Verifying that data transmitted were not altered,
- **Non-repudiation** : Proving that the sender sent a given message.

Type of Encryption Applications

- **In-transit encryption**: protects data while it is transferred from one machine to another,
- **At-rest encryption**: protects data stored on one machine.

Encryption most important concepts

- **Confusion**: Obscures the relationship between the Cipher Text and the key. In a perfect cipher, changing one bit of the key should change all bits of the Cipher Text.
- **Diffusion**: Hides relationship between the Plain Text and the Cipher Text (eg. symbols frequencies). In a perfect cipher changing a single bit of the Plain Text bit affects at least half of the Cipher Text bits.
- **Kerckhoffs's Principle**: The algorithm can be public:

  It [cipher] should not require secrecy, and it should not be a problem if it falls into enemy hands.

There is no security in obscurity.

Attackers model I

Black Box - Attackers may only see inputs / outputs:

- **Ciphertext-Only Attackers (COA) :** see only the ciphertext,
- **Known-Plaintext Attackers (KPA):** see ciphertext and plaintext,
- **Chosen-Plaintext Attacker (CPA):** encrypt plaintext, and see ciphertext,
- **Chosen-Ciphertext Attakers (CCA):** encrypt plaintext, decrypt ciphertext.

Attackers model II

Grey Box - Attackers see cipher's implementation:

- **Side-Channel Attacks:** study the behavior of the implementation, eg. **timing attacks** [1]:
  - Osvik, Shamir, Tromer [**?**]: Recover AES-256 secret key of LinuxâĂŹs dmcrypt in just 65 ms
  - AlFardan, Paterson [**?**]: âĂIJLucky13âĂİ recovers plaintext of CBC-mode encryption in pretty much all TLS implementations
  - Yarom, Falkner [**?**]: Attack against RSA-2048 in GnuPG 1.4.13: âĂIJOn average, the attack is able to recover 96.7% of the bits of the secret key by observing a single signature or decryption round.âĂİ
  - Benger, van de Pol, Smart, Yarom [**?**]: âĂIJreasonable level of success in recovering the secret keyâĂİ for OpenSSL ECDSA using secp256k1 âĂIJwith as little as 200 signaturesâĂİ

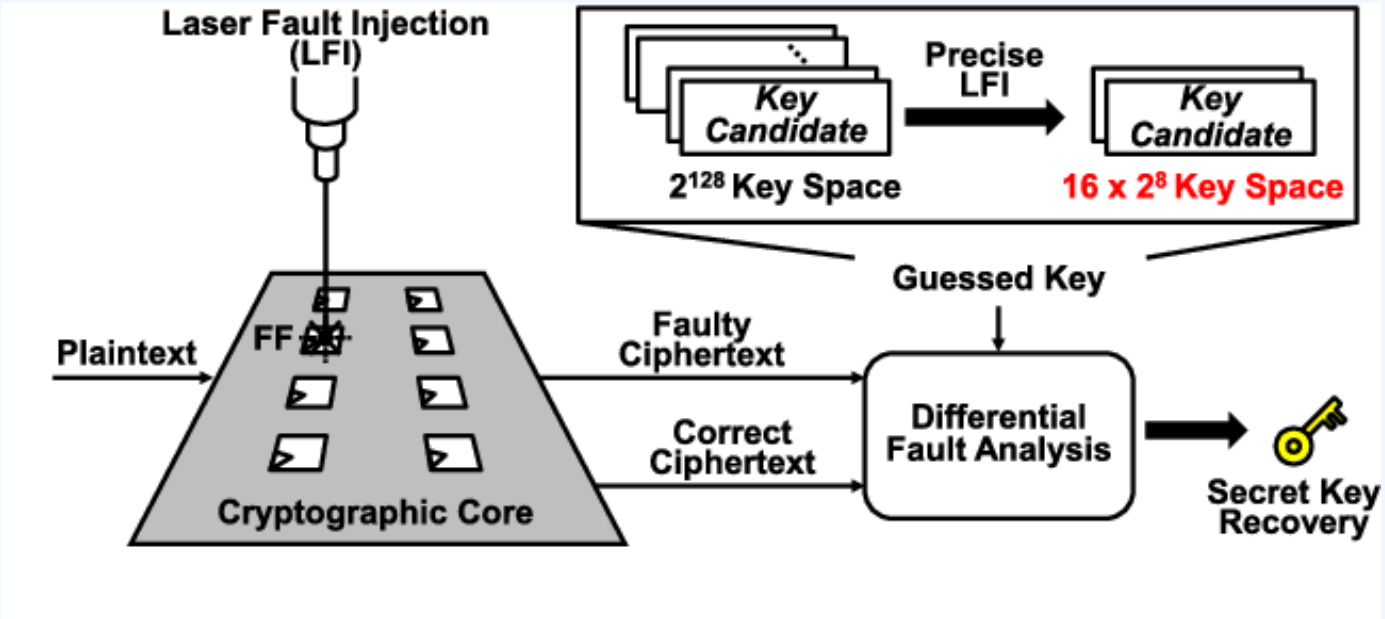
Attackers model III

Most recent timing attack: **TPM-fail** [?]

> We discovered timing leakage on Intel firmware-based TPM (fTPM) as well as in STMicroelectronics' TPM chip. Both exhibit secret-dependent execution times during cryptographic signature generation. While the key should remain safely inside the TPM hardware, we show how this information allows an attacker to recover 256-bit private keys from digital signature schemes based on elliptic curves.

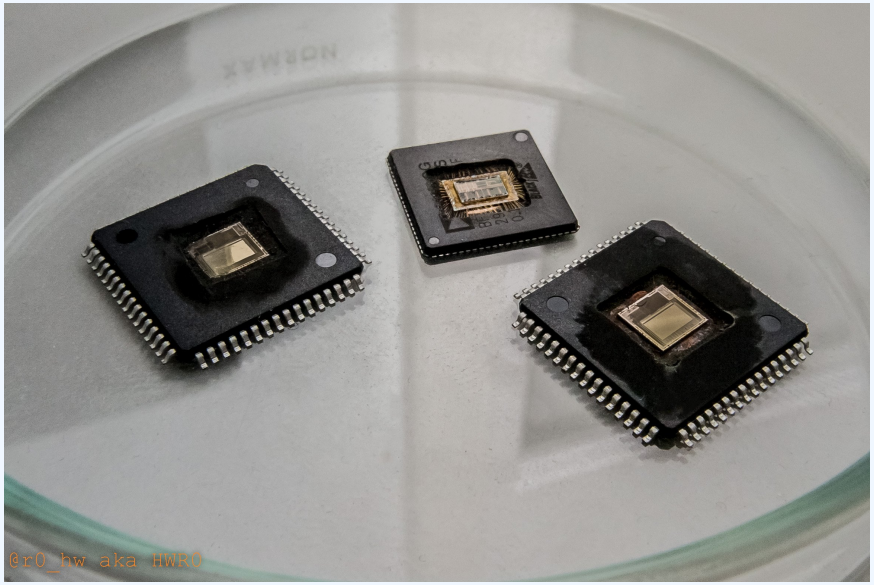
Attackers model IV

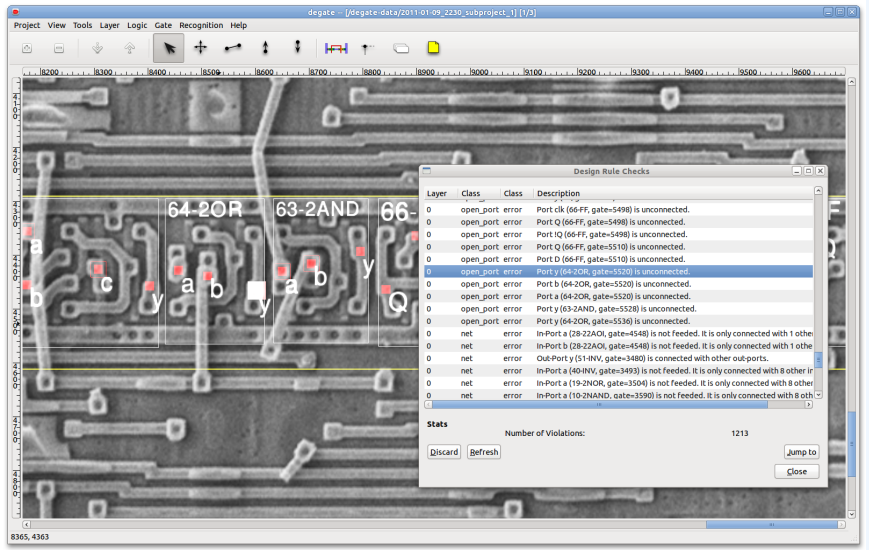
- **Invasive Attacks:**
  - injecting faults [**?**],



Attackers model V

- decapping chips [2], reverse engineering [3] [4], etc [?].





[1] https://cryptojedi.org/peter/data/croatia-20160610.pdf
[2] https://siliconpr0n.org/wiki/doku.php?id=decap:start
[3] http://siliconzoo.org
[4] http://degate.org

Security Notions

- **Indistinguishability (IND) :** Ciphertexts should be indistinguishable from random strings,
- **Non-Malleability (MD):** "Given a ciphertext $C_1 = E(K, P1)$, it should be impossible to create another ciphertext, C_2 , whose corresponding plaintext, P_2 , is related to P_1 in a meaningful way."

Semantic Security (IND-CPA) is the most important security feature:

- Ciphertexts should be different when encryption is performed twice on the same plaintext,
- To achieve this, randomness is introduced into encryption / decryption:
  - $C = E(P, K, R)$
  - $P = D(C, K, R)$

Semantic Security

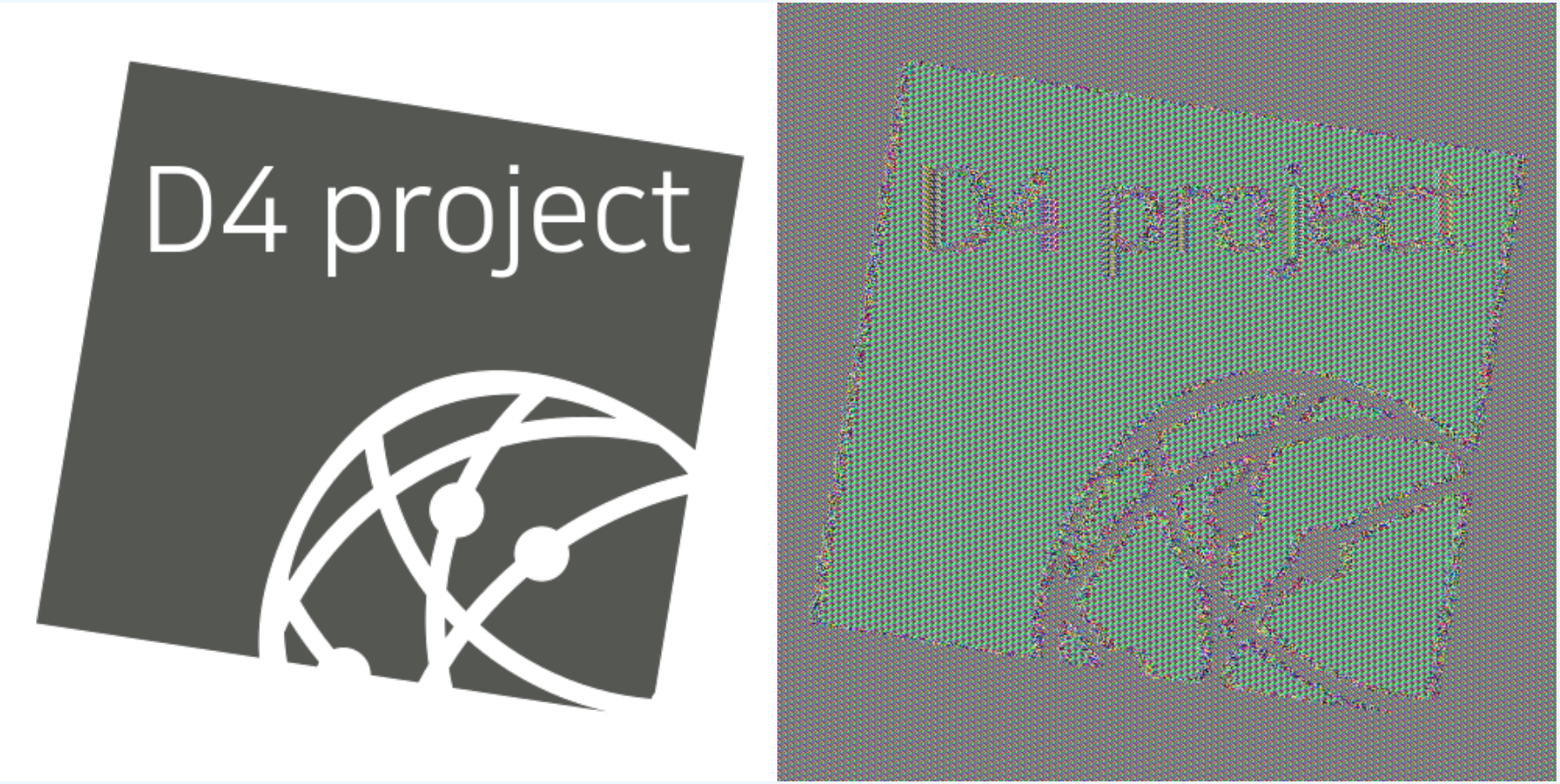


Figure: Image encrypted with AES-ECB

Semantic Security

IND-CPA should not leak information about the PlainText as long as the key is secret:

- $C^1 = E(K, P^1)$, $C^2 = E(K, P^2)$, what are the couples?
- the same message encrypted twice should return two different CipherText,
- one way to achieve this is to introduce randomness in the encryption process: $C = E(K, R, P)$ where R is fresh random bits,
- C should not be distinguishable from random bits.

No Semantic Security without randomness

Randomness

- **Entropy**: (measure of) disorder in a system,
- **Random Number Generator**: a source of entropy, or uncertainty,
- **Pseudo Random Number Generator**: a crypto algorithm that produces a stream of random (hopefully) bits from the RNG.
- there are cryptographic and non-cryptographic (predictable) PRNG,
- there are software-based, and hardware-based PRNG.

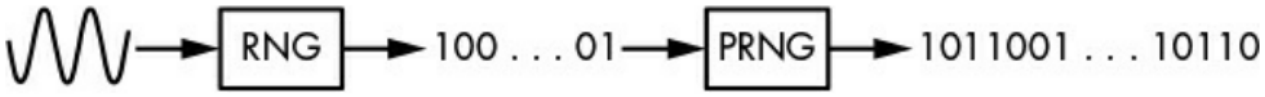


Figure 2-1: RNGs produce few unreliable bits from analog sources, whereas PRNGs expand those bits to a long stream of reliable bits.

Bad entropy sources are a disaster for crypto-systems (ask casinos).

Quantifying Security

RSA 2048 is roughly 100 bits security.

- The key size is different for the “bits of security”,
- “n-bits” of security means that 2^n operations are needed to compromise break a cipher.

Type of encryption

- Symmetric encryption: two parties share a key to encrypt and decrypt,
- Asymmetric encryption, there are two keys:
  - one can encrypt – this one is public – so public can send you encrypted messages,
  - another one can decrypt – this one is private – so you can decrypt the message encrypted for you.
- Obviously, one can not compute the private key from the public key.
- as the public key is public, the attacker model of public-key cryptography is Chosen Plaintext Attacker.

Brute Force 101

Brute Forcing - Basics

2 Approaches:

- **Exhaustive Key Search:**
  - n bits key : 2^n trials,
  - most likely around half of the trials (2^{n-1}),
  - no memory needed.
- **Code Book Attack**
  - Pre-Compute $C = E(P, K)$ for all keys K,
  - store 2^k keys,
  - for a given C, look up for K.

Brute Forcing - Key Search

Key search is testing each possible keys by trial and errors:

- We usually consider that one trial requires 1 ns to complete,
  - n bits key : 2^n trials,
  - 128 bits of security : 2^{128} trials,
  - 2^{88} ns = age of the universe,
  - with ns by trial, we need 2^{40} times the age of the universe to cover all keys,
- some attacks can be done in parallel (sequentially independent operations):
  - For one million cores:
  - length of one million in bits is $log2(1000000) = 19,93$
  - $2^{128}/2^{20} = 2108$
  - 2^{20} times the age of the universe.

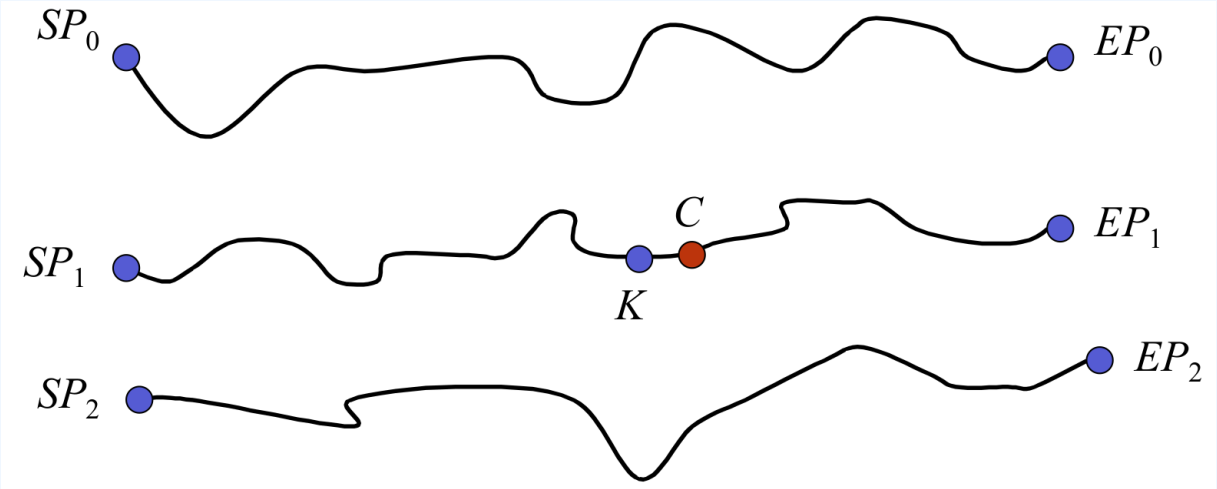
Brute Forcing - TMTO I

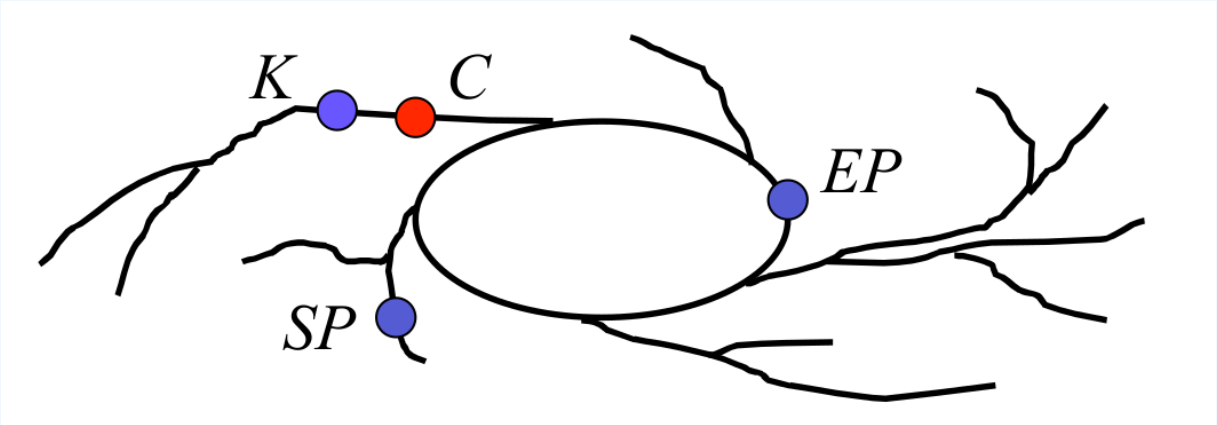
"It usually takes a long time to find a shorter way."

Time-**M**emory **T**rade **O**ff:

- Chosen Plaintext Attack,
- Hellman in 1980,
- It is a trade-off between Exhaustive Key Search, and Code Book Attacks,
- more expensive than an exhaustive search as it requires:
  - 2^n one-time pre-computations, using one known plaintext,
  - the storage of these 2^n results,
  - the results are chains, that also have a cost to invert.
- speed-up attacks against memory space,
- useful when routinely attacking a cipher (eg. computing 1.68 To of tables allows for almost instant cracking of A5/1 cipher used in GSM communications).

Brute Forcing - TMTO II





Rainbow Tables are an improved version of Hellman's algorithm.

How Keys are generated anyway?

There are three ways keys can be generated:

- By **Randomly** choosing the key from a PRNG,
- by **Deriving** the key from a password using a Key Derivation Function,
- by using a **Key agreement protocol** that requires interactions between involved parties.

Encryption and Law Enforcement

2016 ENISA / EUROPOL JOINT STATEMENT

- In the arms race between cryptographers and crypto-analysts. In terms of practical breaks, cryptographers are miles ahead.
- In a society that is ever more depending on the correct functioning of electronic communication services, technical protection of these service is mandatory,
- In the face of serious crimes, law enforcement may lawfully intrude privacy or break into security mechanisms of electronic communication,
- **proportionality** - collateral damages (class breaks)
- Resolving the encryption dilemma: collect and share best practices to circumvent encryption.

Encryption Workarounds [?] I

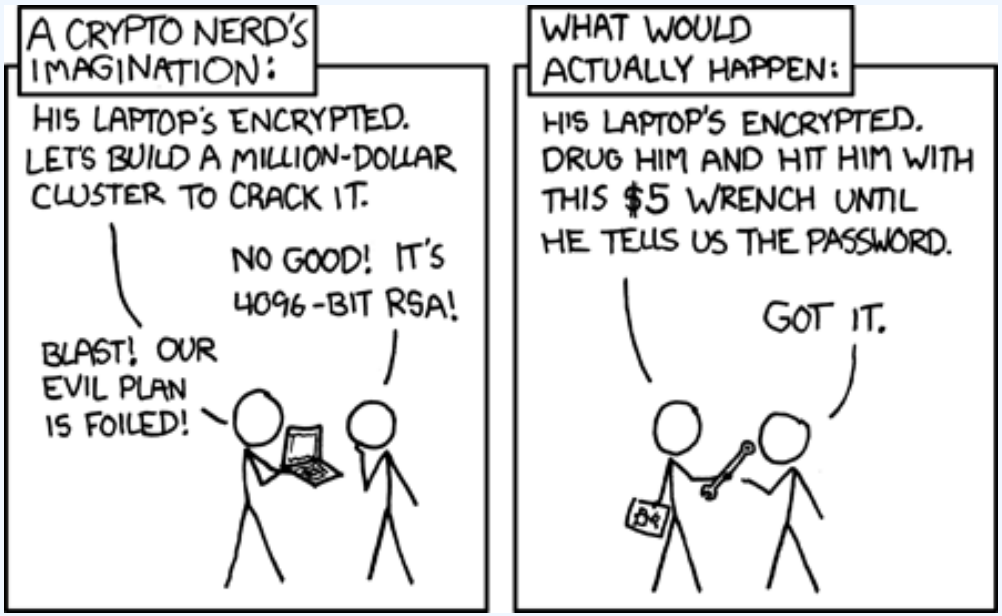
Any effort to reveal an unencrypted version of a target's data that has been concealed by encryption.

- **Try to get the key:**
  - **Find the key:**
    - physical searches for keys,
    - password managers,
    - web browser password database,
    - in-memory copy of the key in computer's HDD / RAM.
    - seize the key (keylogger).
  - **Guess the key:**,
    - Whereas encryption keys are usually too hard to guess (eg. 128bits security is 2^{128} trials (universe is 2^{88} ns old)),
    - passphrases are usually shorter to be memorizable, and are linked to the key,
    - some systems have limitations on sorts of passwords (eg. 4/6 digits banking application),

Encryption Workarounds [?] II

- educated guess on the password from context,
- educated guess from owner's other passwords,
- dictionaries and password generation rules ([5]).
- Offline / online attacks (eg. 13 digits pw: 25.000 on an iphone VS matter of minutes offline),
- + beware devices protection when online (eg. iphone erase on repeated failures).

- **Compel the key:**



Encryption Workarounds [?] III

- **Try to access the PlainText without the key:**
  - **Exploit a Flaw:**
    - Weakness in the algorithm (more on that later),
    - weakness in the random-number generator (more on that later),
    - weakness in the implementation,
    - bugs (eg. Gordon's exploit on android in 2015[6]),
    - backdoors (eg. NSA NOBUS -Bullrun program- Dual EC-DRBG [**?**]
  - **Access PlainText when in use:**
    - Access live system memory,
    - especially useful against Full Disk Encryption,
    - Seize device while in use,
    - remotely hack the device,
    - "Network Investigative Technique" (eg. Playpen case against tor).

Encryption Workarounds [?] IV

- **Locate a PlainText copy:**
  - Avoid encryption entirely,
  - cloud providers (eg. emails),
  - remote cloud storage (eg. iCloud),

Takeaways:

- **No workaround works every time:** the fact that a target used encryption does not mean that the investigation is over.
- **some workarounds are expensive:** exploiting.
- **expertise may be have to be found outside of the governments:** vendors' assistance?

Encryption Workarounds [?] V

Technically, we can retain that crypto-systems have weaknesses:

- key generation,
- key length,
- key distribution,
- key storage,
- how users enter keys into the crypto-system,
- weakness in the algorithm itself / implementation,
- system / computer running the algorithm,
- crypto system used in different points in time,
- **users.**

[5] https://hashcat.net/hashcat/
[6] https://cve.circl.lu/cve/CVE-2015-3860

When cryptography helps investigations

- authentication mechanisms between peers,
- openGPG can leak a lot of metadata
  - key ids,
  - subject of email in thunderbird,
- Bitcoin's Blockchain is public,
- correlating these data with external sources can yields interesting insights,
- More on this in AIL workshop.

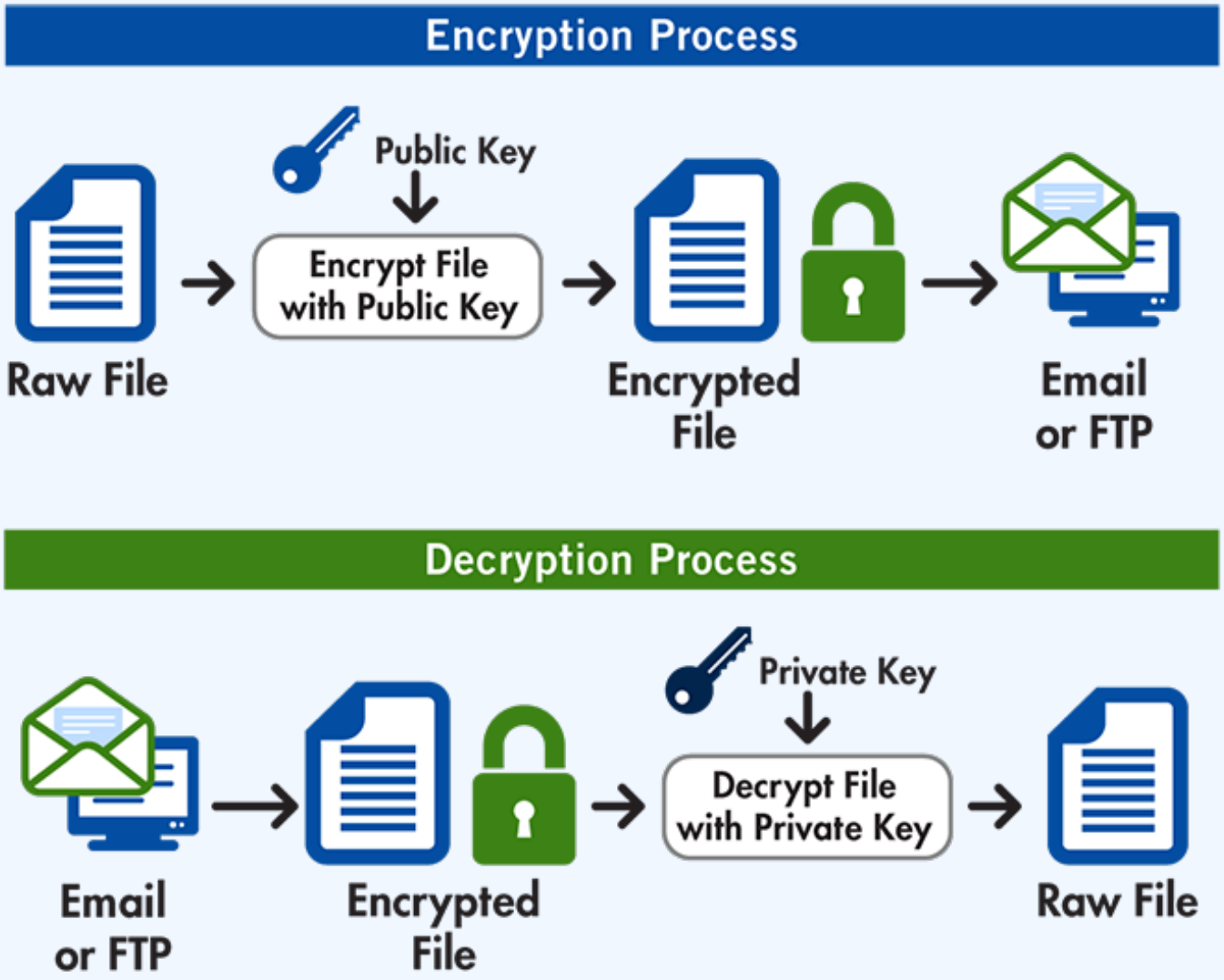
Pretty Good Privacy / Gnu Privacy Guard

Pretty Good Privacy / Gnu Privacy Guard

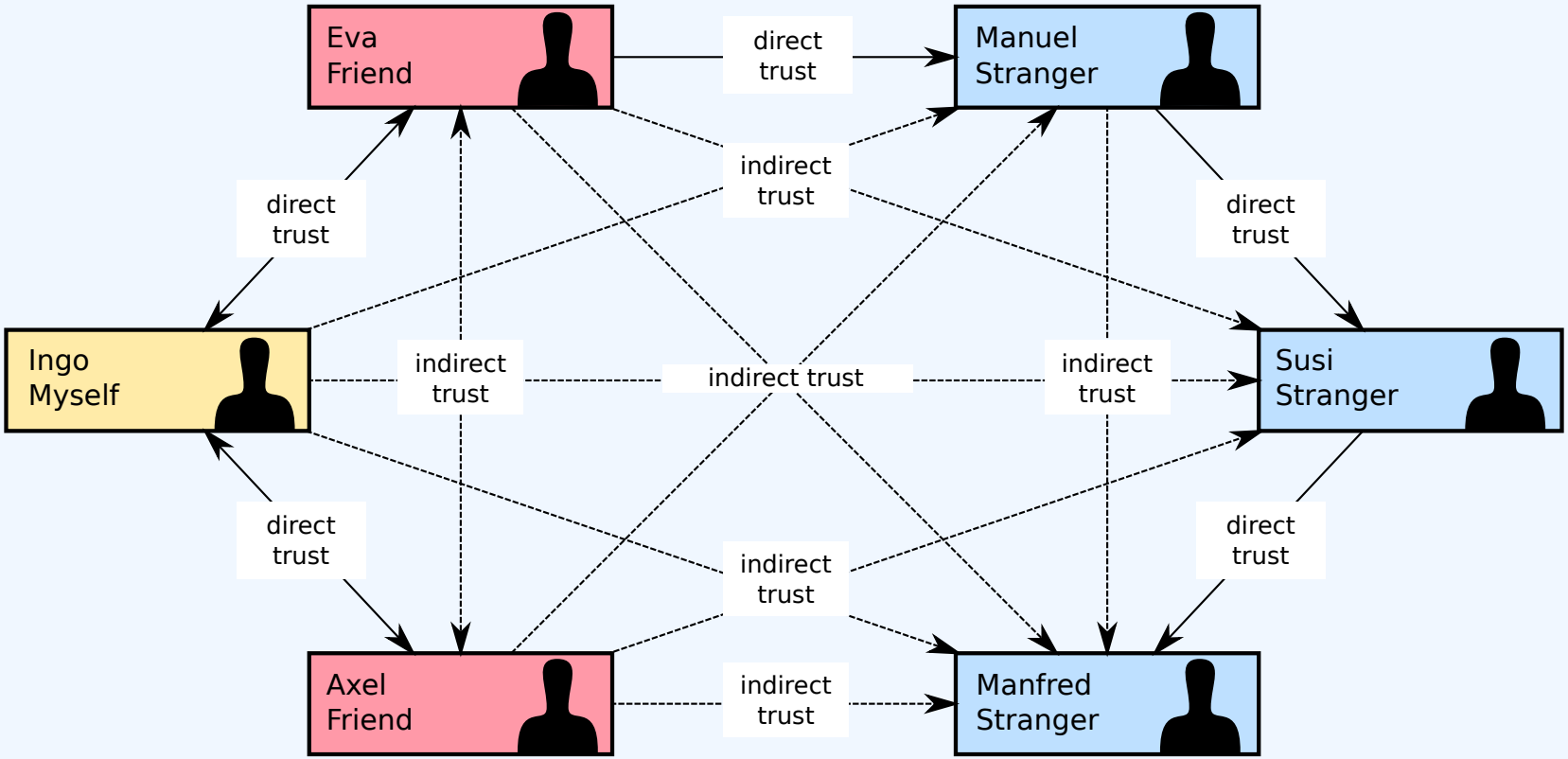
- PGP was Invented By Phil Zimmermann in 1991,
- Hybrid Cipher: asymmetric encryption with symmetric encryption,
- allows to sign communications and files for authentication,
- very low vulnerability count over the years [7],
- One can generate collisions on short IDs though[8],
- no Perfect Forward Secrecy,
- but sessions keys.

[7] https://cve.circl.lu/search/gnupg/gnupg
[8] https://github.com/lachesis/scallion/

Pretty Good Privacy / Gnu Privacy Guard



Pretty Good Privacy / Gnu Privacy Guard



Gnu Privacy Guard: Session keys

- **Hands-on**

  ```
  Move into ~/hands-on/GPGsessions
  ```

- We create two keys, one for the person being the focused of an investigation A (The Very Bad Guy), and one for a witness B (Mr. Good Guy),
- then, we encrypt two messages:
  - one from A to B: to_encrypt_relevant.asc,
  - and a note, form B to B (note): to_encrypt_irrelevant.asc,
- B's passphrase is "goodguypassphrase",
- act as B and extract the session key for to_encrypt_relevant.asc,
- act as a cop and use the session key to decrypt to_encrpyt_relevant.asc,
- verifies that it does not work to decrypt to_encrypt_irrelevant.asc.

Broken Implementations

Default private keys I

SonarG/sonarfinder-ibm-4.1.8.el7.jsonar.x86_64.rpm:

Sonar Finder is part of SonarG and is distributed from https://gbdi-packages.jsonar.com/ within

```
md5sum: a3e4792e1f37b58ff054e05499f69bad rhel7.x_IBM_Guardium_big_data_security_installer_4
```

As

```
./sonarfinder-ibm-4.1.8.el7.jsonar.x86_64.rpm
```

Inside this rpm resides default configuration for an apache catalina server:

```
./opt/sonarfinder/sonarFinder/conf/server.xml
```

with the following default:

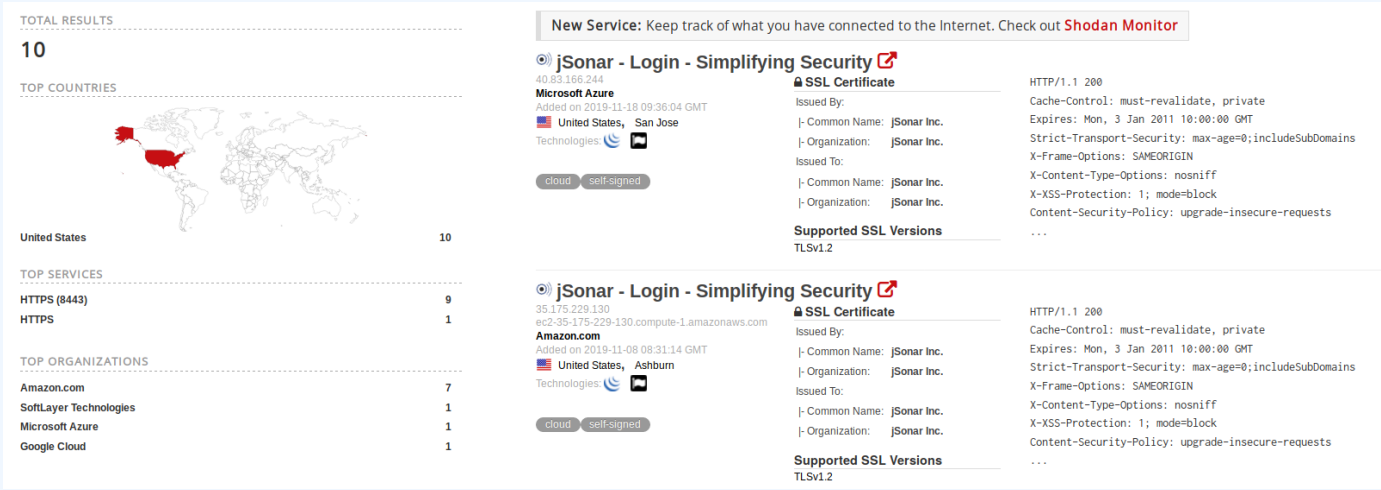
```
            <Certificate certificateKeyFile="${catalina.home}/sslCerts/jsonar.key"
                         certificateFile="${catalina.home}/sslCerts/jsonar.crt"
                         type="RSA" />
```

jsonar.key and jsonar.crt files are indeed present in the rpm.
They should instead be generated during the installation because otherwise they offer no protection to the users who to not take care of rotating these keys.

Impact

Loss of confidentiality, integrity and authenticity.

Default private keys II



Hello @jlhuynen, thank you for your report. Our product team has performed analysis on the reported issues and have determined the reported issue is not applicable due to reasoning below. Please let us know if you have any further questions or can provide additional information on the reported vulnerability.

They are generated and separate for every customer. This is the tomcat cert that the browser verifies. The customer generates these (we can't - because it is tied to a hostname of the customer).

XOR ENCRYPTION

```
class SecureFileHandler:
@staticmethod
def encrypt_file(filepath, content, hash_source, encrypt, return_string_only=False):
    msg_header = "SecureFileHandler encrypt_file"
    enc_string = content
    if encrypt:
        with open(hash_source, 'r') as fp:
            key = DispatcherUtils.get_hash_from_string("".join(fp.readlines()))
        try:
            cipher = XOR.new(key[:32])
```

“Custom” Key derivation function I

```
sonar_salt = bytes.fromhex('a462e2029fffc63b')
sonar_crypt_rounds = 5
```

“Custom” Key derivation function II

```
def evp_bytes_to_key(salt, data, count):
    """
    Derive the key and the IV from the given password and salt.
    """
    iv_len = 16
    key_len = 32

    data_bytes = bytes(data.encode('ascii'))

    data_and_salt = (data_bytes + salt)

    dtot = bytes_to_key__md(hashlib.sha1, data_and_salt, count)

    d = [dtot]
    while len(dtot) < (iv_len + key_len):
        d.append(bytes_to_key__md(hashlib.sha1, d[-1] + data_and_salt, count))
        dtot += d[-1]

    return dtot[:key_len], dtot[key_len:key_len + iv_len]
```

AES-ECB

- Check out opt/sonarfinder/sonarFinder/sonardispatch/encryption.py

```
def _get_new_cipher(self):
    return Cipher(algorithms.AES(key=self.key), modes.ECB(), backend=default_backend())
```

Where the Electronic Code Book mode is chosen.

- One can easily try to guess passwords length as padding is not randomized, but use rjust instead:

```
def _make_encryptable_as_64bytes(some_string):
    return some_string.rjust(64).encode()
```

for instance using this small snippet of code:

```
    seed = "123456789abcde"
    p = lambda n: (seed * n)
    blocks = []
    for i in range(1,5):
        print(self.encrypt_text(p(i)))
```

You will obtain an output similar to this depending on your certificate cert.pem:

```
A+p/5lk1p+4BVy8IKE2YowPqf+ZZNafuAVcvCChNmKMD6n/mWTWn7gFXLwgoTZij978tU8k7UVjH4i4Wo2rDsQ==
A+p/5lk1p+4BVy8IKE2YowPqf+ZZNafuAVcvCChNmKMCLqgaP4UVu+oRcNMkgB7wqSx6/yCifkS18mG+FifbkQ==
A+p/5lk1p+4BVy8IKE2Yo2JaxRTcVnYtYHFMCkJXXVjbxU/x+qCMzOQ47lcbY2QTqSx6/yCifkS18mG+FifbkQ==
hRkVCQa2sjq2QTpX00AmDvo7MHzZ+C8qie62/pem7cjbxU/x+qCMzOQ47lcbY2QTqSx6/yCifkS18mG+FifbkQ==
```

One can then easily guess how many 16 bytes blocks are needed to encipher this password.

Understanding RSA

RSA Basics

Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman in 1977:

- asymmetric crypto system,
- can encrypt and sign,
- messages are big numbers,
- encryption is basically multiplication of big numbers,
- creates a *trapdoor permutation*: turning x in y is easy, but finding x from y is hard.

RSA "by hand"

- **Hands-on,** a sagemath script that is a toy example of RSA:

```
cd ~/hands-on/UsingRSA
sage rsa.sage
```

- **Outputs:**

```
PlainText is: 1234567890
p = random_prime(2^32) = 2312340619
q = random_prime(2^32) = 2031410981
n = p*q = 4697314125248937239
phi = (p-1)*(q-1) = 4697314120905185640
e = random_prime(phi) = 2588085603940229747
d = xgcd(e,phi)[1] = -2102894211931680277
Does d*e == 1?
 mod(d*e, phi) = 1
CipherText y = power_mod(x, e, n) = 14546069107110627 45
Decrypted CT is: 1234567890
```

RSA - Use with openssl

- **Hands-on**:

  `~/hands_on/UsingRSA`

- Decrypt message.bin
- generate a new private key,
- generate the corresponding public key,
- use this new key to encrypt a message,
- use this new key to decrypt a message.

With only one key

Several potential weaknesses:

- Key size too small: keys up to 1024 bits are breakable given the right means,
- close p and q,
- unsafe primes, smooth primes,
- broken primes (FactorDB, Debian OpenSSL bug).
- signing with RSA-CRT (instead of RSA-PSS)

With a set of keys

Several potential weaknesses:

- share moduli: if $n_1 = n_2$ then the keys share p and q,
- share p or q,

In both case, it is trivial to recover the private keys.

Breaking small keys[9]

- **Hands-on**:

  `~/hands_on/SmallKey`

- what is the key size of smallkey?
- what is n?
- what is the public exponent?
- what is n in base10?
- what are p and q?

Let's generate the private key: using p, then using q.

[9] https://www.sjoerdlangkemper.nl/2019/06/19/attacking-rsa/

Close Prime Factors

- **Hands-on**:

  ~/hands_on/ClosePQ

- use Fermat Algorithm[10] to find **both p and q:**

```
def fermatfactor(N):
  if N <= 0: return [N]
  if is_even(N): return [2,N/2]
  a = ceil(sqrt(N))
  while not is_square(a^2-N):
    a = a + 1
  b = sqrt(a^2-N)
  return [a - b,a + b]
```

[10] http://facthacks.cr.yp.to/fermat.html

Shared prime factors

Researchers have shown that several devices generated their keypairs at boot time without enough entropy[11]:

```
prng.seed(seed)
p = prng.generate_random_prime()
// prng.add_entropy()
q = prng.generate_random_prime()
n = p*q
```

Given n=pq and n' = pq' it is trivial to recover the shared p by computing their **Greatest Common Divisor (GCD)**, and therefore **both private keys**[12].

"They cracked cracked about 13000 of them"

[11] Bernstein, Heninger, and Lange: http://facthacks.cr.yp.to/

[12] http://www.loyalty.org/~schoen/rsa/

Shared prime factors

- **Hands-on**:

  `~/hands_on/SharedPrimeFactor`

- Read README.txt, you have a challenge to solve :
  - the *answers* folder should be left alone for now,
  - *scripts* contains scripts that may be useful to solve the challenge,
  - *attempts* may hold your attempt are generating private keys.
  - *bgcd-bd.sage* contains Daniel J. Berstein's algorithm for computing RSA collisions in batches.

Hands-on: Exploiting Weaknesses in RSA
– at bigger scale –

Snake Oil Crypto[13] - Problem Statement

We reckon that IoT devices **are often the weakest devices** on a network:

- Usually the result of cheap engineering,
- sloppy patching cycles,
- sometimes forgotten–not monitored (remember the printer sending sysmon?),
- few hardening features enabled.

We feel a bit safer when they use TLS, but we what you now know about RSA, should we?

[13] https://github.com/d4-project/snake-oil-crypto

Snake Oil Crypto - GCD

In Snake-Oil-Crypto we compute GCD[14] between:

- between certificates having the same issuer,
- between certificates having the same subject,
- on keys collected from various sources (PassiveSSL, Certificate Transparency, shodan, censys, etc.),
- python + redis + postgresql [15]

“Check all the keys that we know of for vendor X”

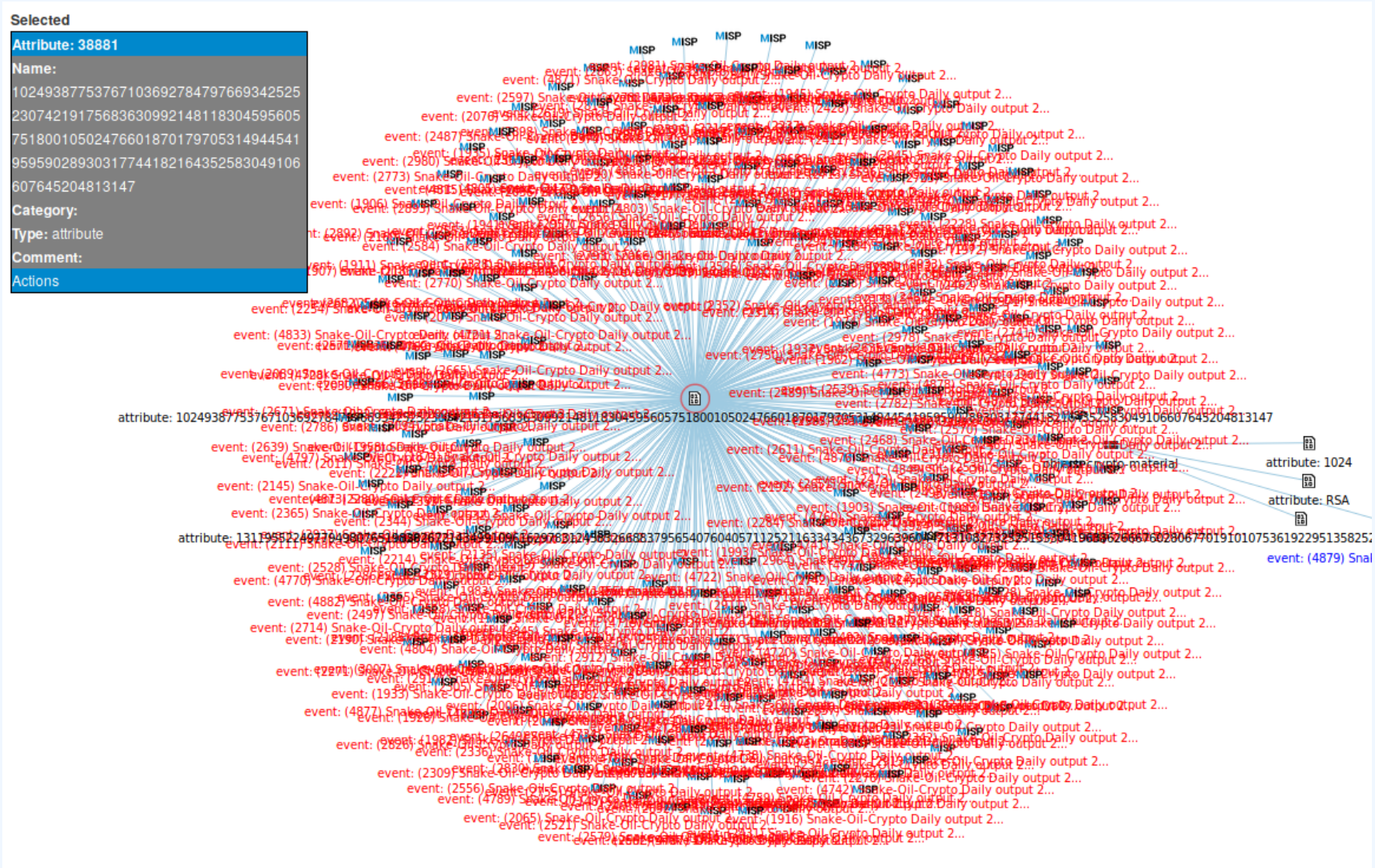
[14] using Bernstein’s Batch GCD algorithm
[15] `https://github.com/D4-project/snake-oil-crypto/`

Snake Oil Crypto - GCD

Quick Demo:

- Let's check how strong are the RSA keys in our database...
- check some results on https://misp-eurolea.enforce.lan
- how bad can it be?
- do you find some vendors we should notify?

Snake Oil Crypto - MISP feed



Snake Oil Crypto - MISP feed

The MISP feed:

- **Allows** for checking automatic checking by an IDS on hashed values,
- **contains** thousands on broken keys from a dozen of vendors,
- **will be accessible upon request (info@circl.lu).**

In the future:

- **Automatic** the vendor checks by performing TF-IDF on x509's subjects,
- **automatic** vendors notification.

Hands-on: Exploiting Weaknesses in RSA
– enter D4-project –

Problem statement

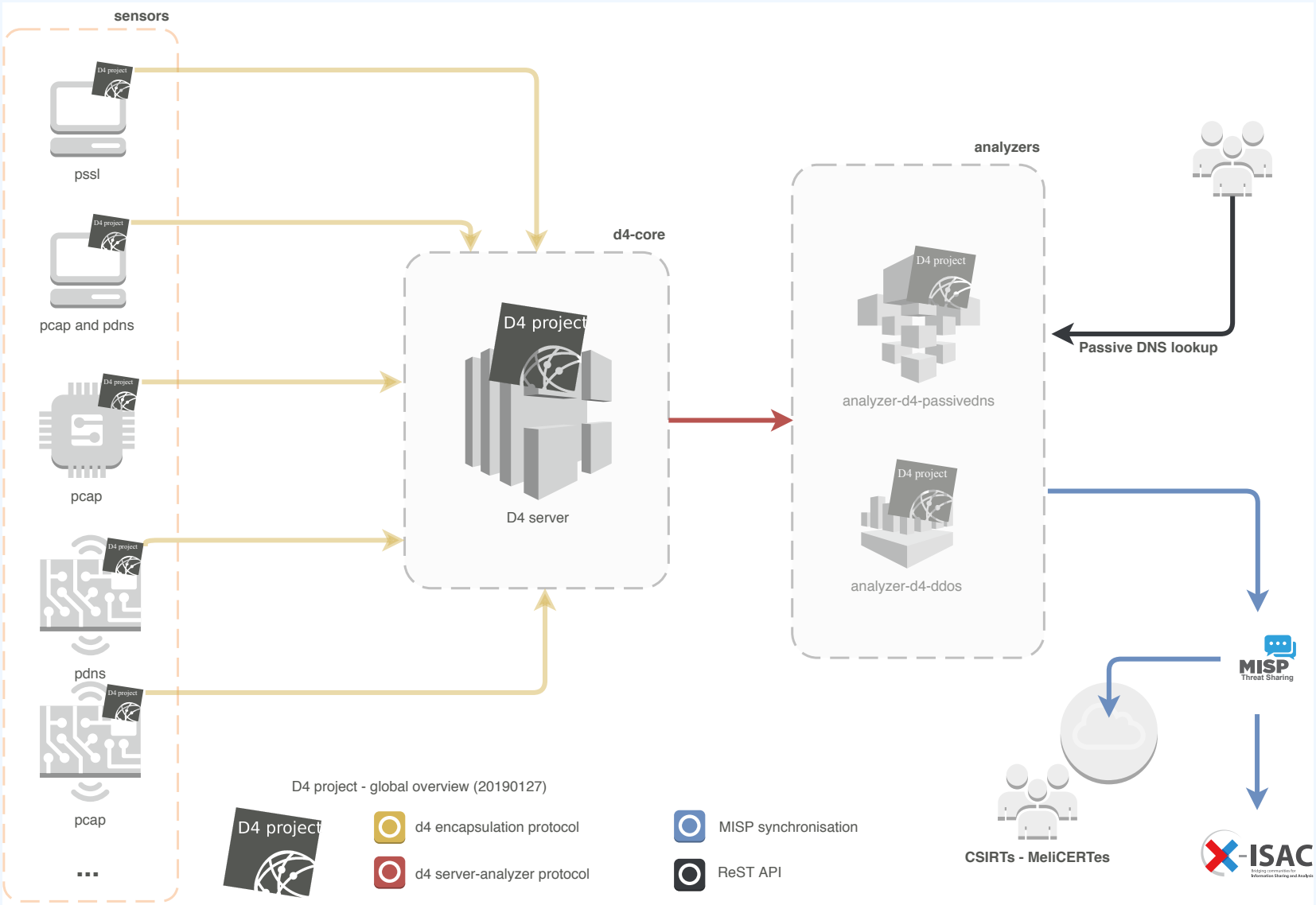
- CSIRTs (or private organisations) build their **own honeypot, honeynet or blackhole monitoring network**
- Designing, managing and operating such infrastructure is a tedious and resource intensive task
- **Automatic sharing** between monitoring networks from different organisations is missing
- Sensors and processing are often seen as blackbox or difficult to audit

Objective

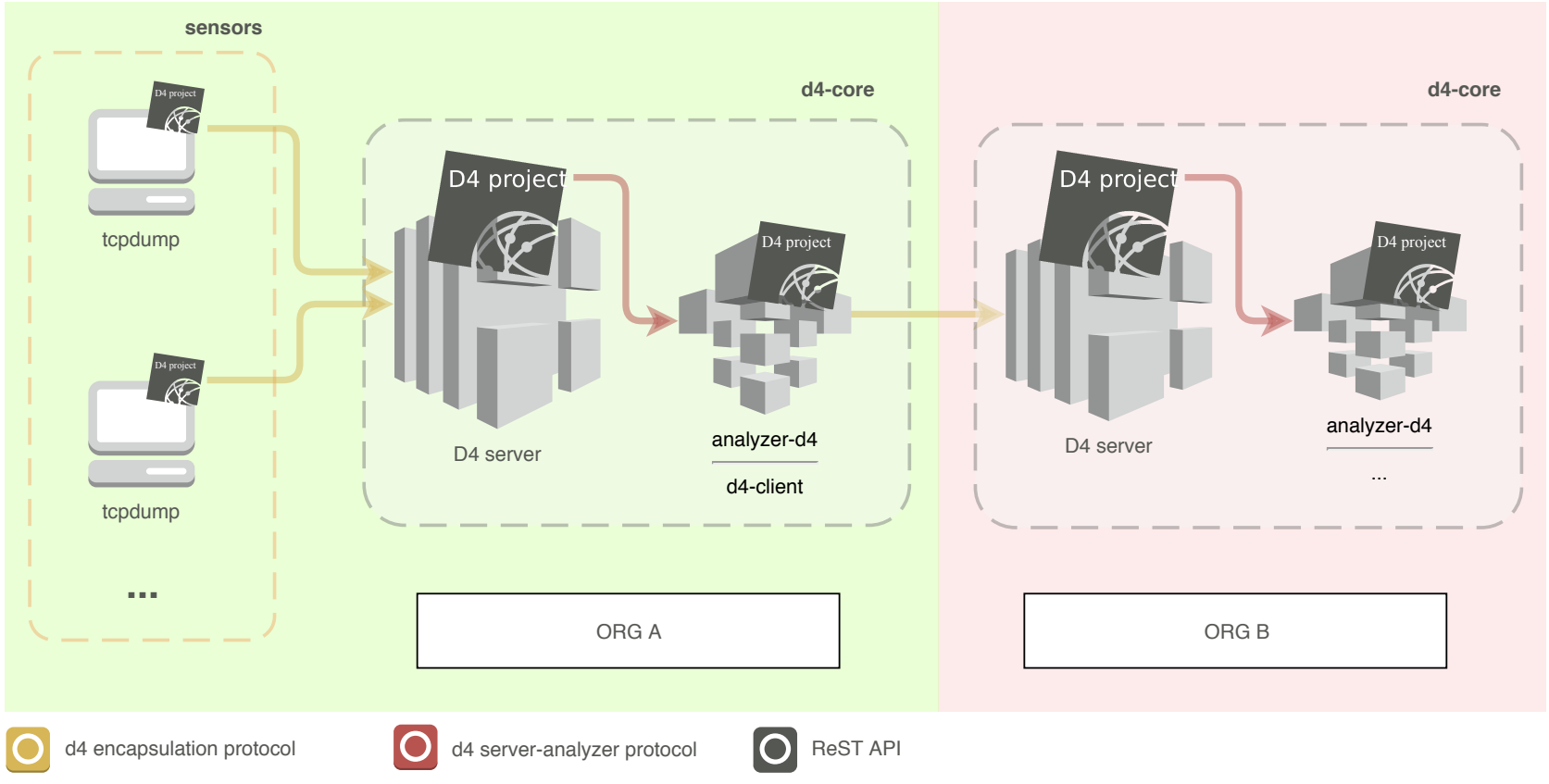
- Based on our experience with MISP[16] where sharing played an important role, we transpose the model in D4 project
- Keeping the protocol and code base **simple and minimal**
- Allowing every organisation to **control and audit their own sensor network**
- Extending D4 or **encapsulating legacy monitoring protocols** must be as simple as possible
- Ensuring that the sensor server has **no control on the sensor** (unidirectional streaming)
- Don't force users to use dedicated sensors and allow **flexibility of sensor support** (software, hardware, virtual)

[16] https://github.com/MISP/MISP

D4 Overview


sensors
pssl
pcap and pdns
pcap
pdns
pcap
...
d4-core
D4 project
D4 server
analyzers
analyzer-d4-passivedns
analyzer-d4-ddos
Passive DNS lookup
MISP
Threat Sharing
CSIRTs - MeliCERTes
-ISAC
D4 project - global overview (20190127)
D4 project
d4 encapsulation protocol
d4 server-analyzer protocol
MISP synchronisation
ReST API

D4 Overview - Connecting Sensor Networks



D4 - TLS Fingerprinting

Keep a log of links between:

- x509 certificates,
- ports,
- IP address,
- client (ja3),
- server (ja3s),

  "JA3 is a method for creating SSL/TLS client fingerprints that should be easy to produce on any platform and can be easily shared for threat intelligence."[17]

Pivot on additional data points during Incident Response

[17] https://github.com/salesforce/ja3

D4 - TLS Fingerprinting

- **Hands-on**:

  `~/hands_on/TLSinspection`

- open stripped.pcap
- what is the admin password?
- bummer, it's encrypted,
- what is the admin password?

D4 - full chain demo.

First release

- ✓ sensor-d4-tls-fingerprinting [18]: **Extracts** and **fingerprints** certificates, and **computes** TLSH fuzzy hash.
- ✓ analyzer-d4-passivessl [19]: **Stores** Certificates / PK details in a PostgreSQL DB.
- snake-oil-crypto [20]: **Performs** crypto checks, push results in MISP for notification
- lookup-d4-passivessl [21]: **Exposes** the DB through a public REST API.

[18] github.com/D4-project/sensor-d4-tls-fingerprinting
[19] github.com/D4-project/analyzer-d4-passivessl
[20] github.com/D4-project/snake-oil-crypto
[21] github.com/D4-project/lookup-d4-passivessl

Get in touch if you want to join/support the project, host a passive SSL sensor or contribute

- Collaboration can include research partnership, sharing of collected streams or improving the software.
- Contact: info@circl.lu
- `https://github.com/D4-Project` - `https://twitter.com/d4_project`

References I