CYBERSECURITY LAB #3

Jocelyne Elias
Giulio Tripi – Tutor didattico

<jocelyne.elias, giulio.tripi>@unibo.it

Exercise

Decrypt the files uploaded on virtuale, hints included

Write a **small** report containing **the steps and the decrypted flags** and upload it on Virtuale

Remember: write name, surname and the number of the lab session on the report!

Applied cryptography

AES

(Advanced Encryption Standard)

Symmetric-key algorithm

- Key length: 128, 192 or 256 bits

Block cipher

- Block size: 128 bits

Lightweight

- Low RAM consumptions
- High speed

Block cipher modes

Confidentiality-only modes

- **ECB** (Electronic Code Block)
- **CBC** (Cipher Block Chaining)
- **CFB** (Cipher Feedback)
- **OFB** (Output Feedback)
- **CTR** (Counter)

OpenSSL

- We will use **OpenSSL to play around with crypto algorithms**
- OpenSSL is an open-source library that implements Basic cryptographic primitives
    - Hashing algorithms
    - **SSL** and **TLS** protocols
    - Various utilities (prime number generator, PRNG, ...)
- It comes with a handy **command line interface** (CLI)
    - We can do everything from our terminal

OpenSSL
Cryptography and SSL/TLS Toolkit

Check website certificate

View the informations of a website's certificate:

```
openssl s_client -connect www.unibo.it:443 2>/dev/null | openssl x509 -noout -text
```

View the TLS connection details

```
openssl s_client -connect www.unibo.it:443 2>/dev/null
```

Check website certificate

Check the supported TLS version:

```
openssl s_client -connect www.unibo.it:443 -servername www.unibo.it -tls1_2
```

Or:

- -tls1
- -tls1_1

https://www.ssllabs.com/ssltest/

Check website certificate

Qualys. SSL Labs

Home Projects Qualys Free Trial Contact

You are here: Home > Projects > SSL Server Test > www.unibo.it

SSL Report: www.unibo.it (137.204.24.208)

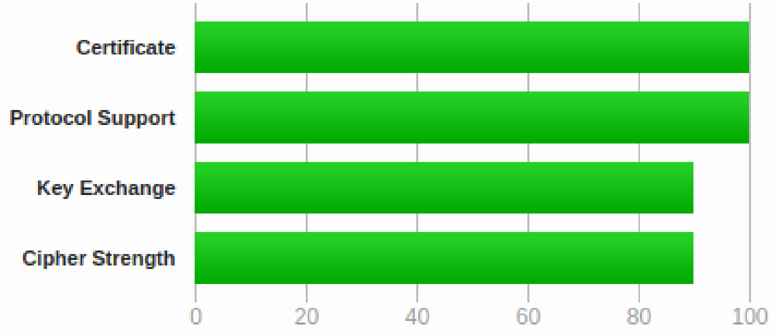
Assessed on: Fri, 19 Apr 2024 22:45:53 UTC | Hide | Clear cache

Scan Another »

Summary

Overall Rating

A



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

Check website certificate

Cipher Suites

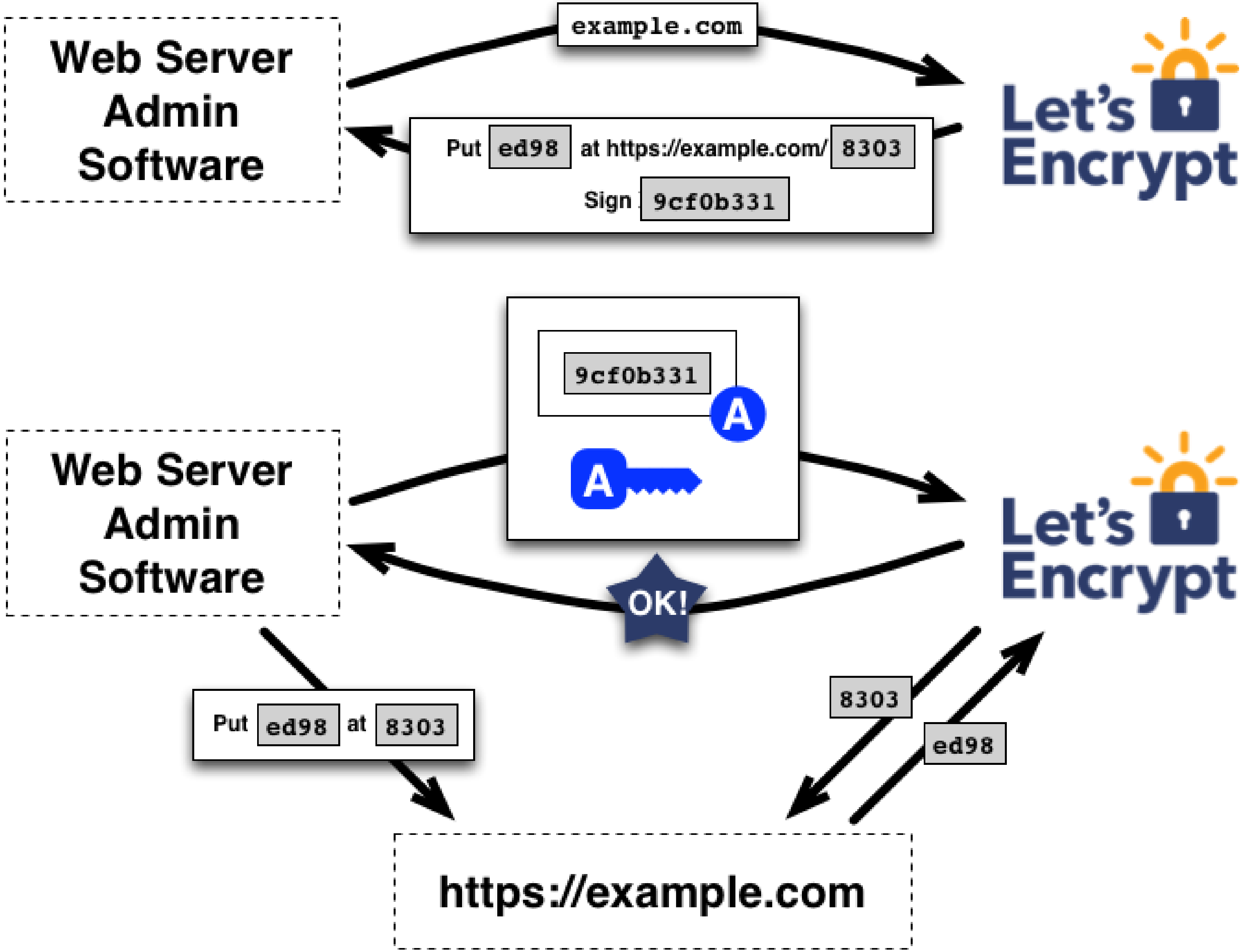
 # TLS 1.2 (suites in server-preferred order)

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH x25519 (eq. 3072 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH x25519 (eq. 3072 bits RSA) FS	256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x9e) DH 2048 bits FS	128
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x9f) DH 2048 bits FS	256
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (0x45) DH 2048 bits FS **WEAK**	128
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256 (0xc0) **WEAK**	256

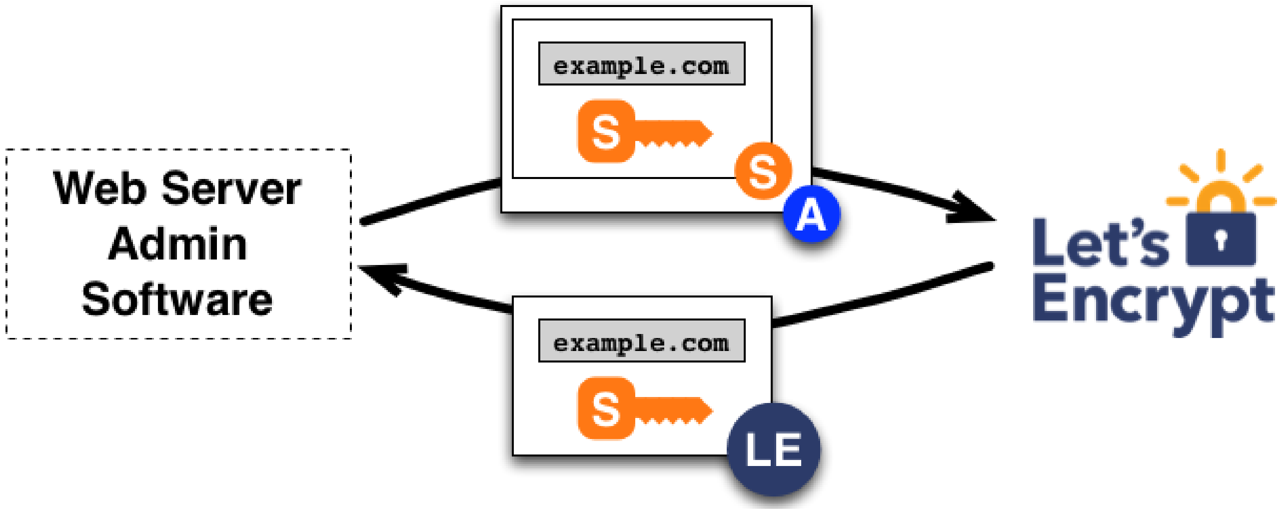
Handshake Simulation

Android 4.4.2	RSA 3072 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Android 5.0.0	RSA 3072 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Android 6.0	RSA 3072 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Android 7.0	RSA 3072 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH x25519 FS
Android 8.0	RSA 3072 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH x25519 FS

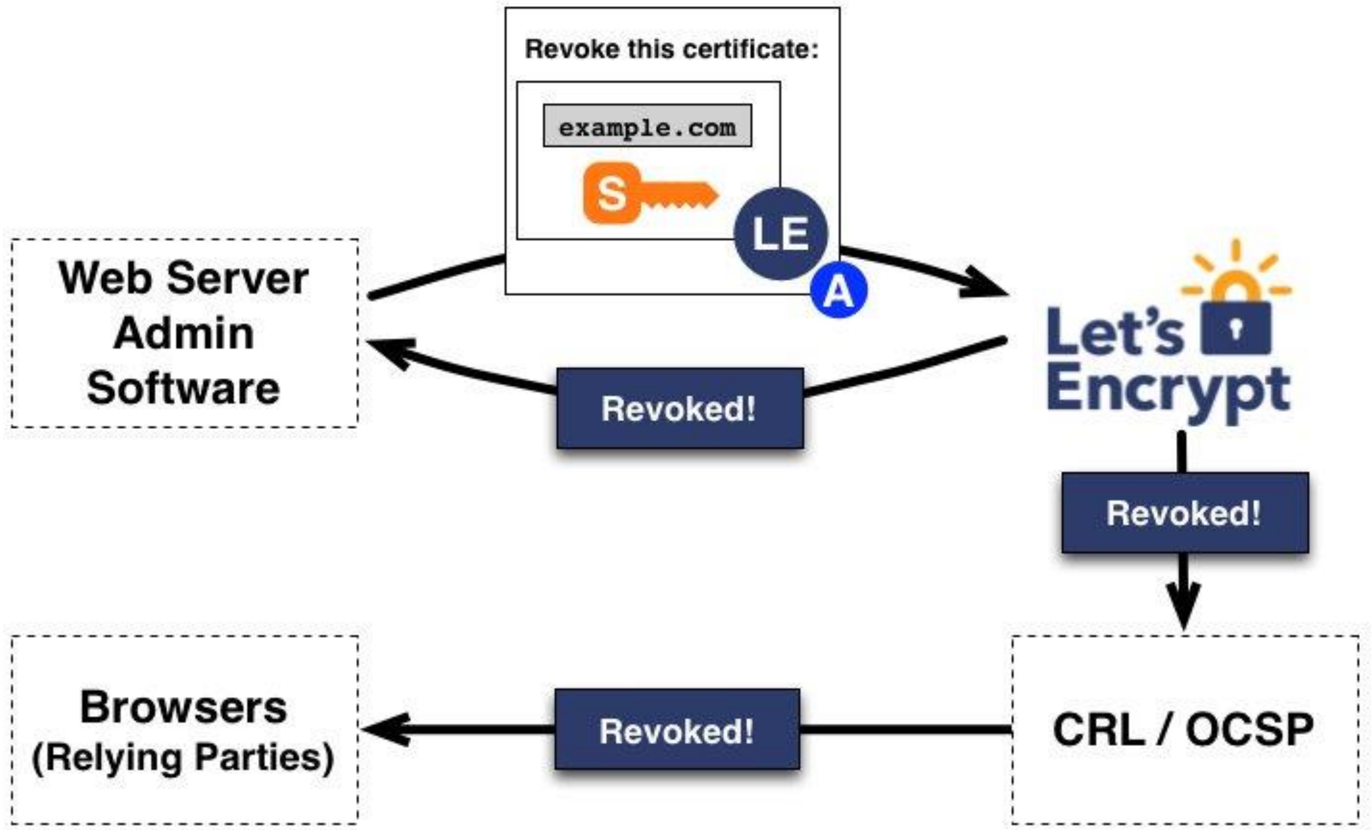
Let's Encrypt



Let's Encrypt (2)



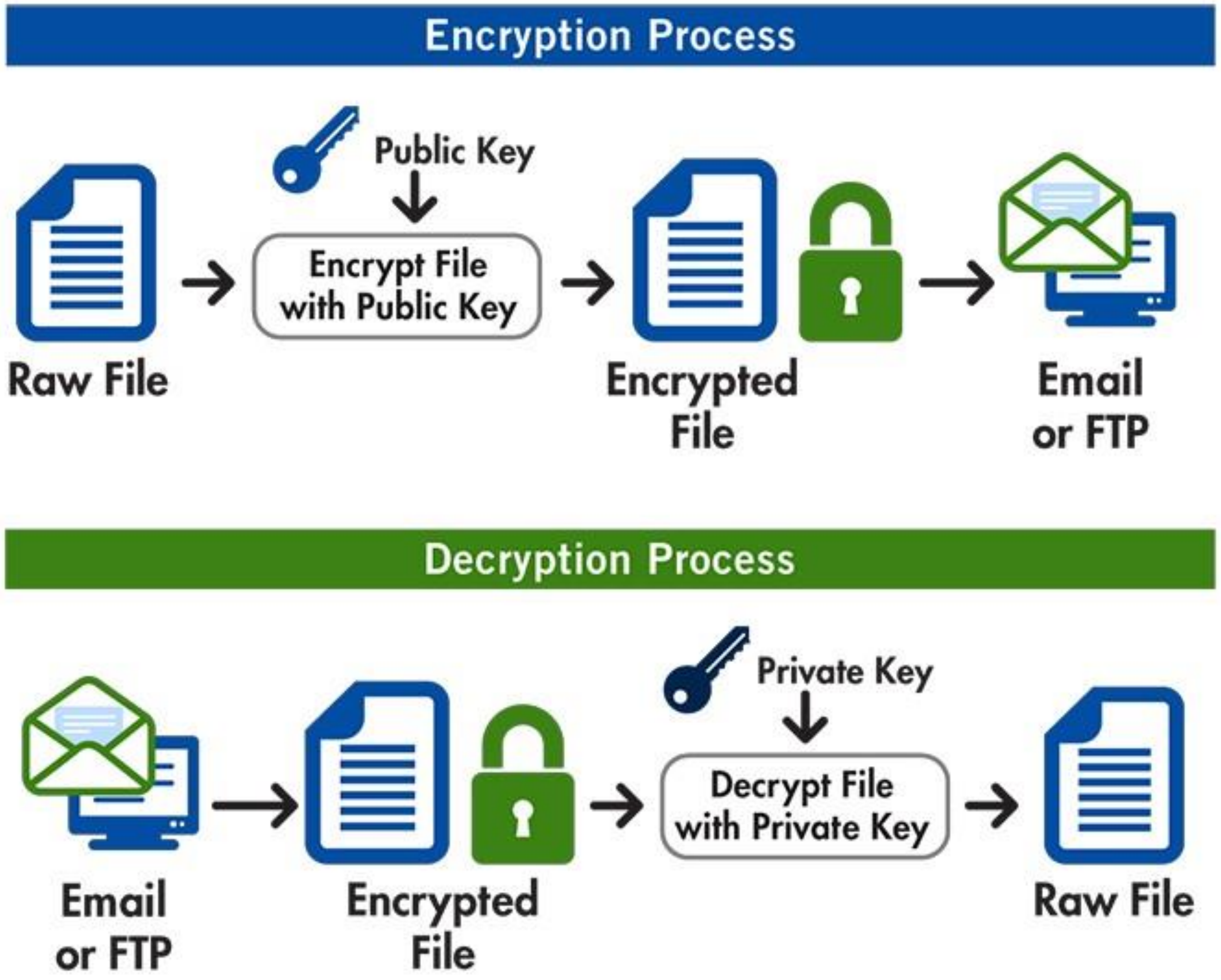
Let's Encrypt (3)



https://letsencrypt.org/how-it-works/

https://letsencrypt.org/docs/dst-root-ca-x3-expiration-september-2021/

GnuPG (GNU Privacy Guard) - OpenPGP



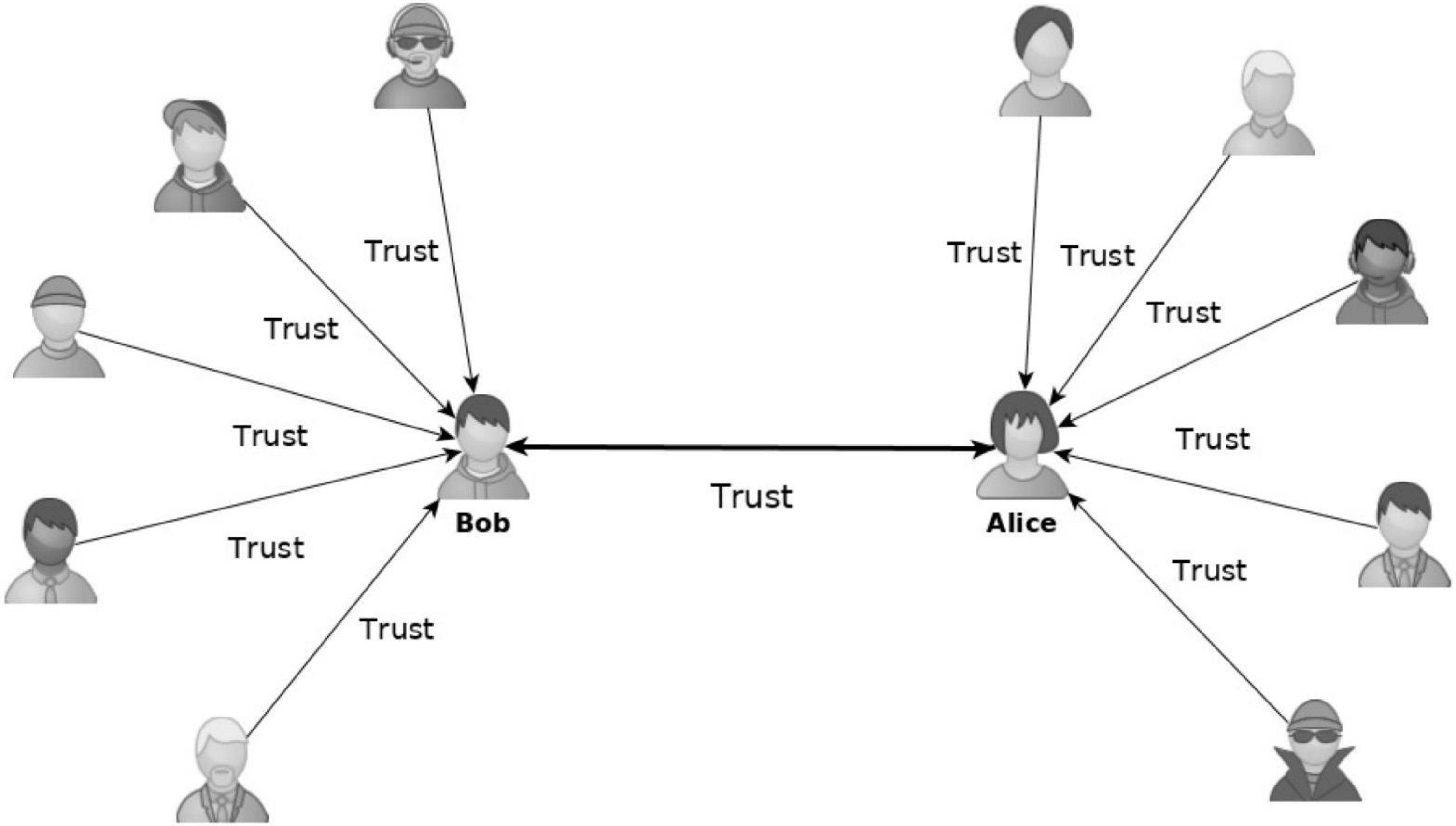
```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

- -------------------------------------------------------------------------
Debian Security Advisory DSA-5649-1                   security@debian.org
https://www.debian.org/security/                     Salvatore Bonaccorso
March 29, 2024                        https://www.debian.org/security/faq
- -------------------------------------------------------------------------

Package        : xz-utils
CVE ID         : CVE-2024-3094

Andres Freund discovered that the upstream source tarballs for xz-utils,
the XZ-format compression utilities, are compromised and inject
malicious code, at build time, into the resulting liblzma5 library.

Right now no Debian stable versions are known to be affected.
Compromised packages were part of the Debian testing, unstable and
experimental distributions.

Users running Debian testing and unstable are urged to update the xz-utils packages.

For the detailed security status of xz-utils please refer to
its security tracker page at:
https://security-tracker.debian.org/tracker/xz-utils

Mailing list: debian-security-announce@lists.debian.org
-----BEGIN PGP SIGNATURE-----
```


iQKTBAEBCgB9FiEERkRAmAjBceBVMd3uBUy48xNDz0QFAmYG4XBfFIAAAAAALgAo
aXNzdWVyLWZwckBub3RhdGlvbnMub3BlbnBncC5maWZ0aGhvcnNlbWFuLm5ldDQ2
NDQ0MDk4MDhDMTcxRTA1NTMxRERFRTA1NENCOEYzMTM0M0NGNDQACgkQBUy48xND
z0QBZg/9HMXAGIvBC12v8PSnp6EjnagxXBjTqLIJzEwQFgmC1cS58Kmv214c3fD+
rxHEfqQxcgjVSWPbIgI5ZXf1XZtx1YiMGRd9aEvKQSwLu0ox0/UR5igZakLrZb+n
t1qvH8AGYQhK41ysFJVwNulUXqqopvGEPgwopLfGPn8P3zjOrs0BoLqYmQ0nbsv3
92l9rAYk6W7G+L3Gwp/cQVzqmyErlEk/QB3Ld+6HLP7a8shY+A8a7iVHE1vkzNjw
JeZ2shIrvkCJqb1/BVSJU92fy2P4xjiMY8phDum7dzWnyy0WZLa90B/tDF9WB7Ok
nuUa020yxjflnabSM112We1V8D5sh18X30NK8scXiCD5cbPEysGqaUf8Baik9qux
Wkn60oqLKFN0VdrUxeqyLp1AC7wEiysQaNqv/8ZqhYF3/KxrbzgBOVy9XeB3pEfk
oLLPtUeH3kuXGw2Qp+Kqg3Zlfe04XZZX5kme/7PFkBvjZ8JFH7dWW+eEO9MbnsPD
br0tWxod0jhvLdZ6YLFad6q2jkjqO3LH3+SYAhp+otcY1TNpIe7xWAB+Phj0TJqu
IoSnYutqEb4mwoUzn9vZRzOxLvyePEJwbFG89sQf4GCYm4FwDhyB51Eo5piC7Fre
EtfsmdU7xAl6tljtUkzTHz27dBIokrgw4W0YrYaeUSmm3jKttPA=
=522l

-----END PGP SIGNATURE-----

https://xkcd.com/1181/

There's no central authority for trust, so everyone is responsible for signing other's keys



https://xkcd.com/364/

Basic usage for AES

Encryption of a simple text file using AES-256 in ECB mode:

openssl aes-256-ecb -e -in (plaintxt) -out (ciphertxt)

Decryption:

openssl aes-256-ecb -d -in (ciphertxt) -out (plaintxt)

Reasoning about the key..

The key in AES must be 128, 192, 256 bits in length…

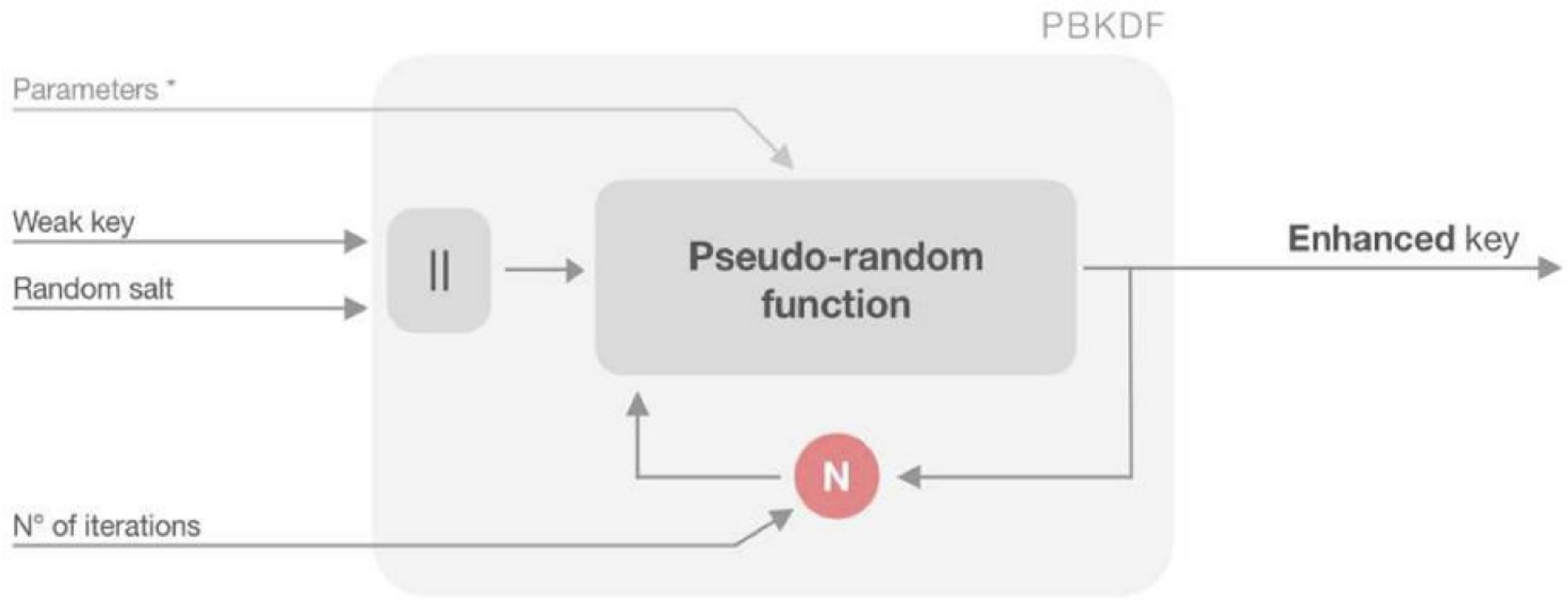
So, can’t we use a human-friendly password to protect our data?

Key concept: Our human-friendly but weak password is used to generate a stronger (enhanced) key with higher entropy

These functions are called **Password Based Key Derivation Functions (PBKDFs)**

In OpenSSL use

- *-p*, to print the actual enhanced key, salt and IV (if used)
- *-nosalt*, to disable the usage of salting to increase the key randomness



By default, OpenSSL **applies a trivial PBKDF**

- If salting is not enabled
  - *key = sha256(passphrase)*
- If salting is enabled
  - *key = sha256(passphrase || salt)*

A better option is to **use more iterations** or **PBKDF2**

- Use the flag *–iter (number of iterations)*, or *–pbkdf2*

Reasoning about the file size..

Size of the plaintext and ciphertext *may* be different

- Ciphertext > plaintext

This happens for two reasons

- **The salt is stored** in the header of the ciphertext (unless –nosalt is used)
- **The plaintext is padded** before being encrypted (*ECB* and *CBC* modes only)
  - *Ciphertext size is always multiple of the cipher block size (128-bit = 16 bytes)*

Visualizing an encrypted file using a normal text editor (or printing on the console) **can't work:**

- The plaintext usually contains ASCII **printable** characters..
- But the ciphertext contains **non-printable** characters

When dealing with such kind of data, we need to view our files using **hexdumps**

- This way, we can visualize binary data encoded in hexadecimal format, e.g.:
  - *0x0a = "\n"*
  - *0x00 = NULL*
  - *0x41 = "A"*

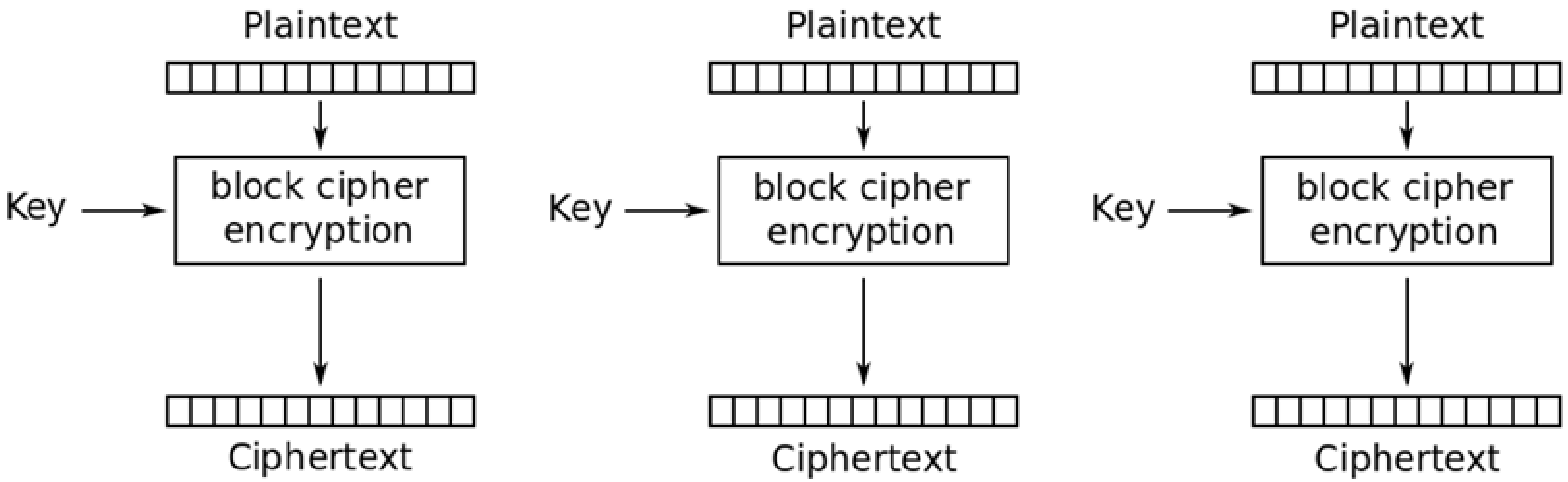
Use **xxd** to visualize the hexdump of a given input file

Weaknesses of ECB mode

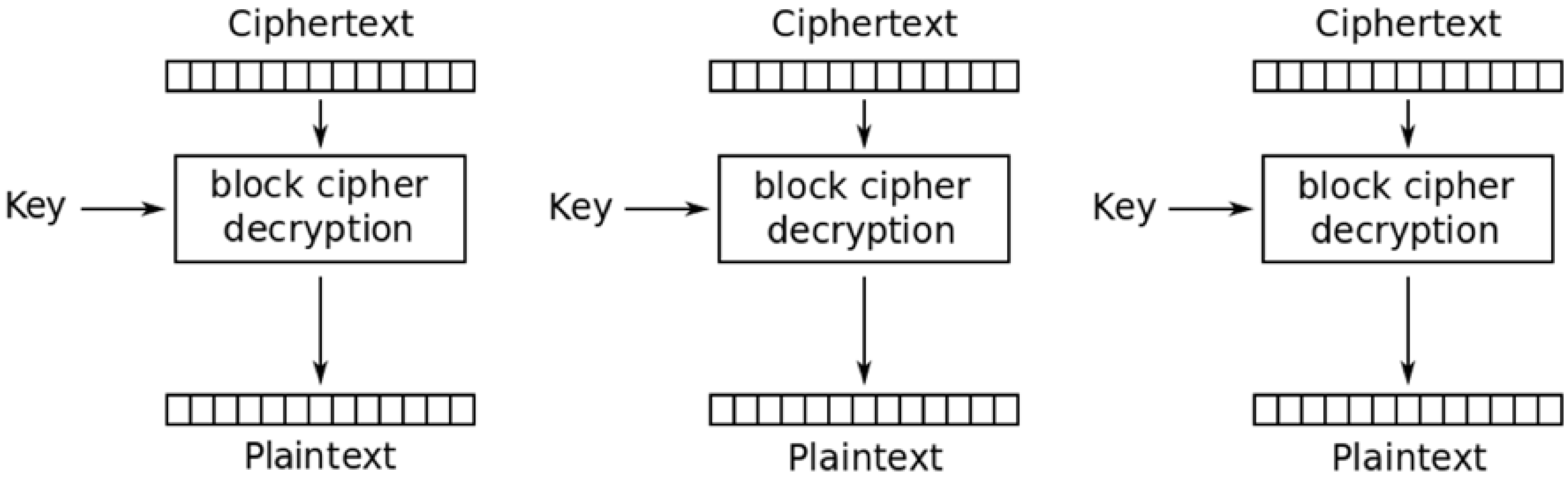
ECB mode

ECB mode **lacks diffusion:**

Identical plaintext blocks produce identical ciphertext blocks



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

We can verify this behaviour encrypting a simple bitmap image

The Tux experiment

- Let’s **encrypt the Linux (*tux*) logo** in **ECB** mode and see what happens
- For the sake of simplicity, the input file will be a simple **bitmap**
  - *.ppm format*

PPM (Portable PixMap) seems a bit exoteric, but in reality **it’s the simplest image format**.

- You can see it using: *xxd –g 3 –c 15 tux.ppm*

```
00000000: 50360a 323635 203331 340a32 35350a  P6.265 314.255.
0000000f: ffffff ffffff ffffff ffffff ffffff  ...............
0000001e: ffffff ffffff ffffff ffffff ffffff  ...............
0000002d: ffffff ffffff ffffff ffffff ffffff  ...............
0000003c: ffffff ffffff ffffff ffffff ffffff  ...............
0000004b: ffffff ffffff ffffff ffffff ffffff  ...............
0000005a: ffffff ffffff ffffff ffffff ffffff  ...............
00000069: ffffff ffffff ffffff ffffff ffffff  ...............
00000078: ffffff ffffff ffffff ffffff ffffff  ...............
00000087: ffffff ffffff ffffff ffffff ffffff  ...............
```

- You may want to install GIMP to view the image
  - *sudo apt update && sudo apt install gimp*

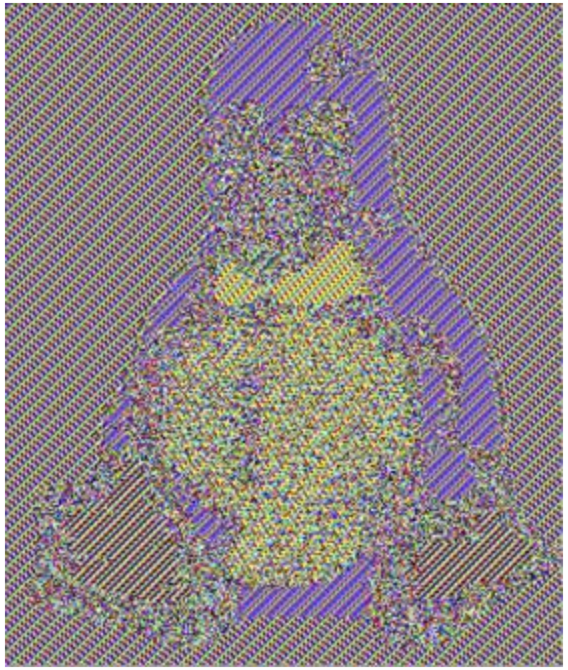
- **Split header and body in two different files**
  - *head -n 3 Tux.ppm > Tux.header*
  - *tail -n +4 Tux.ppm > Tux.body*

- **Encrypt the body**
  - *openssl aes-256-ecb -e -in Tux.body -out Tux.body.ecb*

- **Reassembling everything together**
  - *cat Tux.header Tux.body.ecb > Tux.ecb.ppm*

- Now look at the image…. **Is it familiar?**

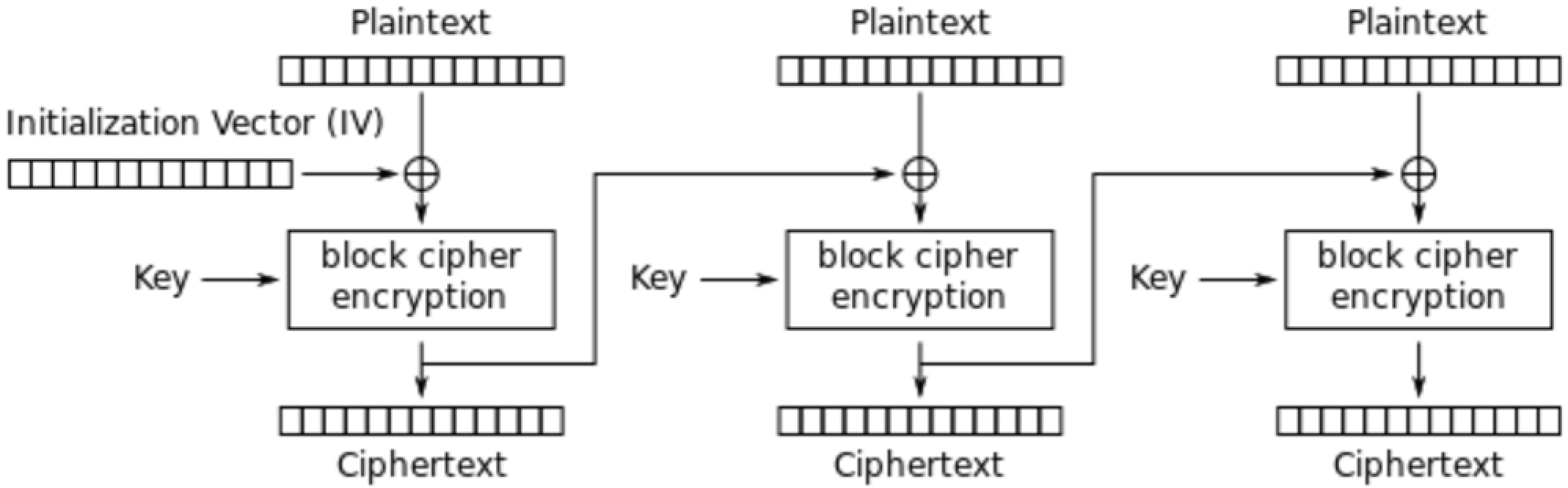
Original



ECB encrypted

- **Try to repeat the experiment with CBC mode!**
  - *But you must provide an IV with the –iv option*

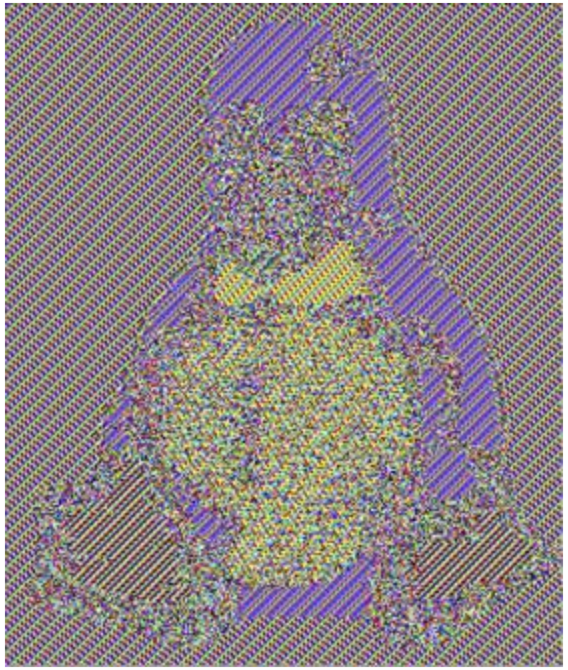
- **CBC hide away patterns in the plaintext** thanks to the **XOR-ing of the first plaintext block with an IV**, before encrypting it
  - Moreover, it involves **block chaining** as every subsequent plaintext block is XOR-ed with the ciphertext of the previous block



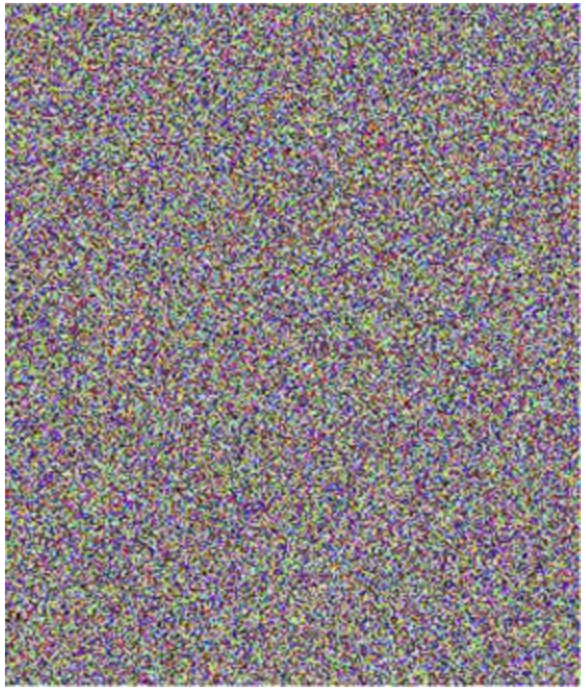
Cipher Block Chaining (CBC) mode encryption

Try **CBC mode** yourself!

Original



ECB encrypted



CBC encrypted

Exercise 1: check certificates

Steps:

1. Find a website that uses **Let's Encryp**t and one website that doesn't. Which one was more difficult to find?
2. Out of the first 10 websites you can think about, how many uses Let's Encrypt?
3. For the two websites of point (1) check which version of TLS they support.
4. Choose a website of your preference (e.g. the website of your football team), check it with ssllabs and write some comments.

Exercise 2: decrypt the files

Steps:

1. Download the files on Virtuale
2. Understand the **modes** (CBC,ECB,…)
3. Find the **passwords** and **use them to decrypt**
4. Write steps and the **FLAG** in the report