Cryptography

From zero to dont-shoot-yourself-in-the-foot

microlab.red
lorenzo nicolodi

Rules of engagement

1. Try to contextualize as much as possible
2. Stop me as soon as something is not clear
3. Keep asking and questioning until you fully understand
4. If you think I am wrong (it happens, believe me), rise your hand
5. I am here to learn too so please, when possible, add:
   a) Details
   b) Real world examples
   c) Different points of view

Tools for learning

To improve the learning process, we will leverage:

- Active recalling
- Spaced repetitions
- Interleaving (when possible)
- Generation
- …

Disclaimer: I am a mind/learning nerd.

Topics

- Cryptography 101
- Randomness
- Cryptographic security
- Block ciphers
- Stream ciphers
- Hash functions
- RSA & Elliptic Curves
- Signing & key management

Warning!

It will be though and dense.

If you feel lost, welcome in the club.

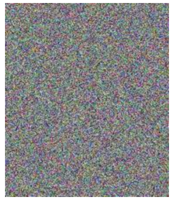
Cryptography 101

Golden rule in cryptography

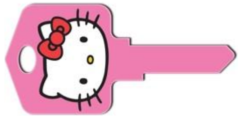
Algorithms and implementations must be open & peer reviewed for years

Cryptography terms

Plaintext (P)



Cyphertext (C)



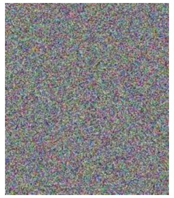
Key (K)

Cipher

Main concepts

→ Must be kept secret

Same **P**, different **K** → Different **C**



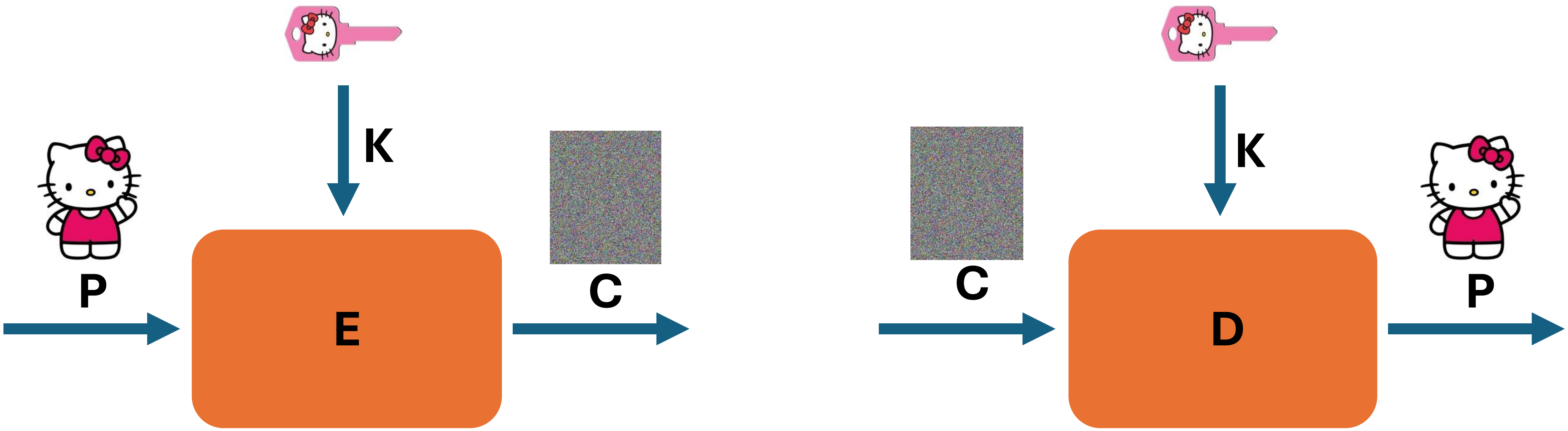
→ Looks really random (?)

Common types of cryptographic algorithms

- Symmetric encryption
- Asymmetric encryption
- Hashing
- Signing

Symmetric encryption

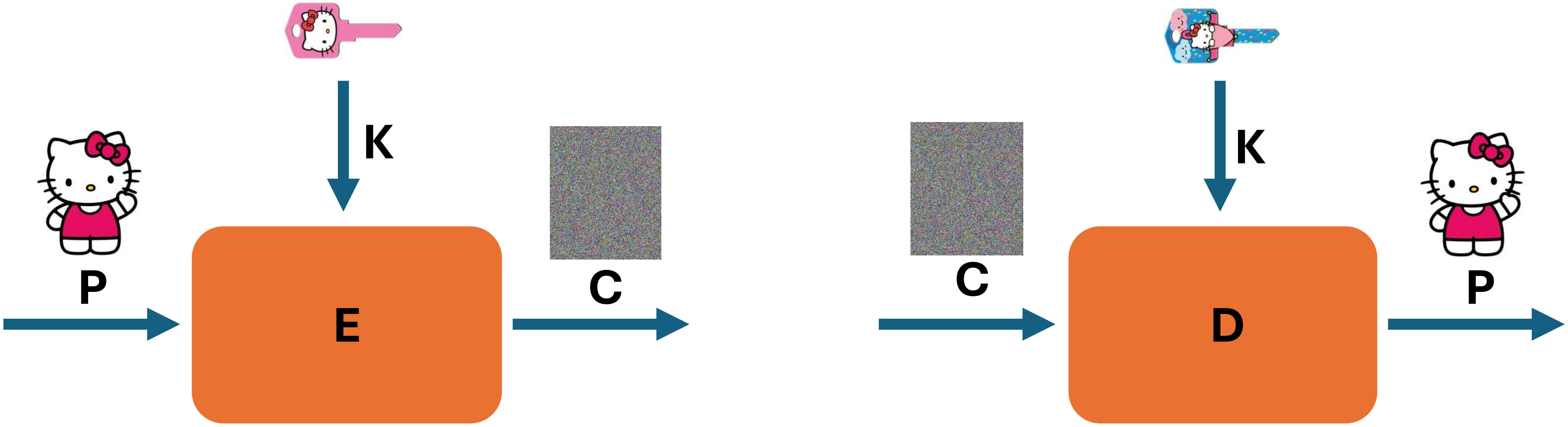
Same key for encryption and decryption



Asymmetric encryption

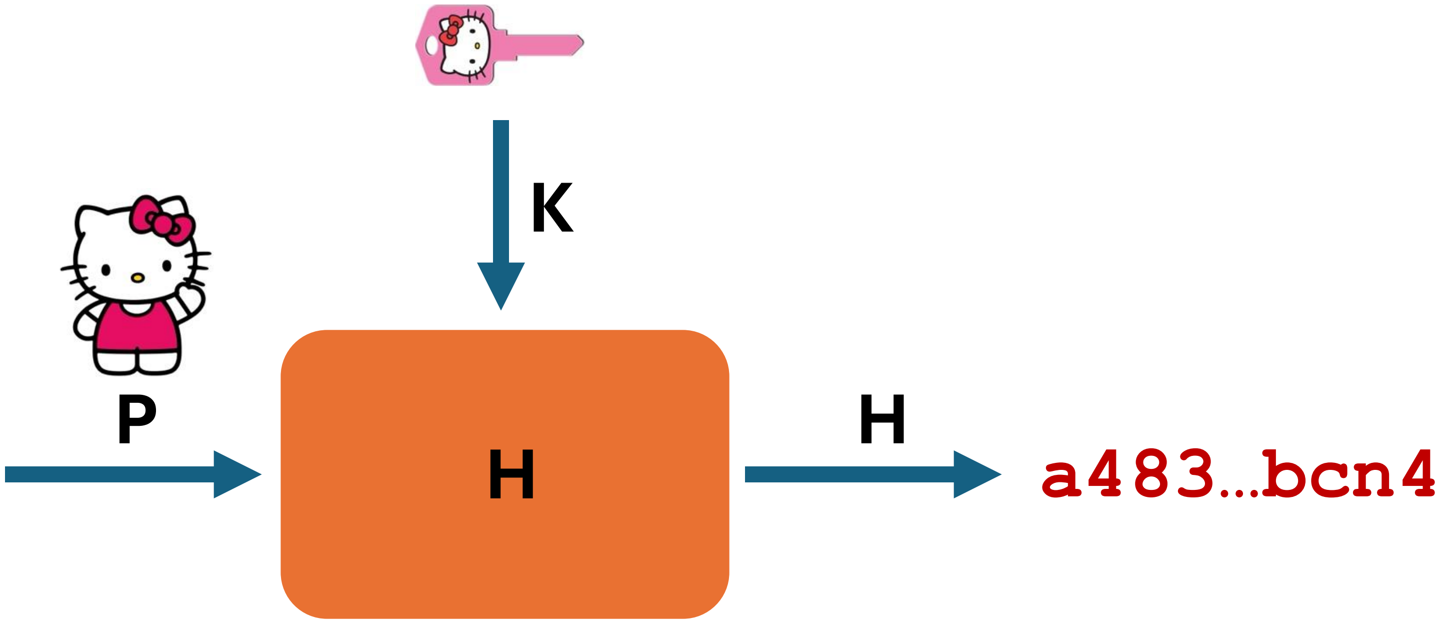
 **Public key** for encryption

 **Private key** for decryption



Hashing

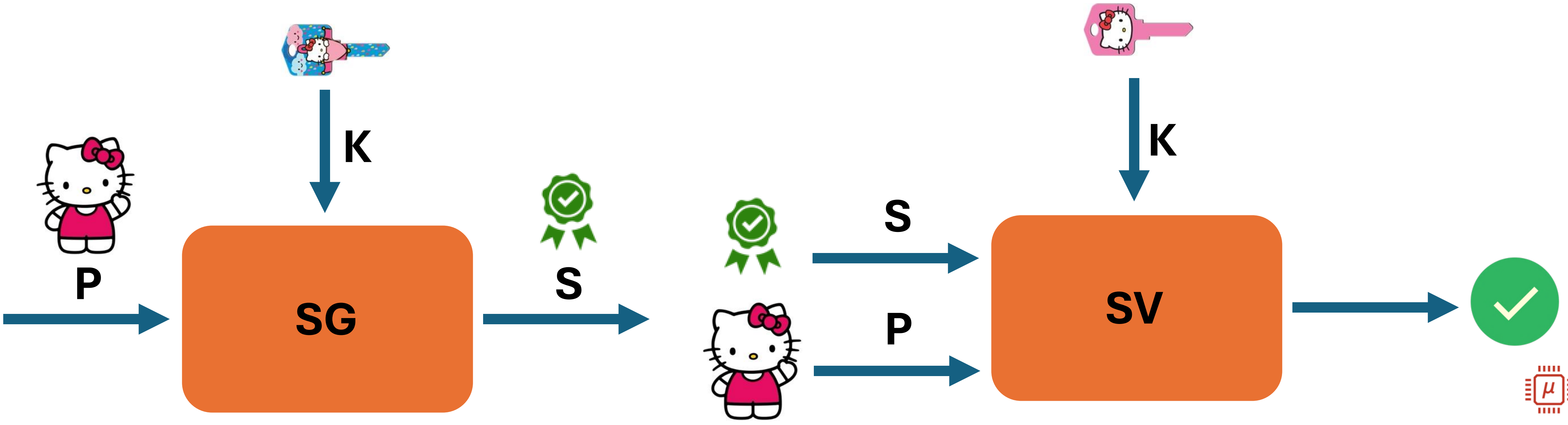
Possible key for authentication



Signing

Public key for signature verification

Private key for signature generation



Recall time

Measuring security

Informational VS computational security

Informational security → theorical definition of how strong a cipher is

it is binary: the cipher is either secure or insecure

Computational security → express the feasibility of an attack in real life

influenced by many factors (memory, parallelism, etc...)

Security level

Computational security → expressed as a tuple $(\boldsymbol{t}, \boldsymbol{\varepsilon})$

$\boldsymbol{t}$ is a limit of the number of operations an attacker can carry out

$\boldsymbol{\varepsilon}$ is the limit on the probability of success of an attack.

N-bit security notation → how many operations $\boldsymbol{t}$ are needed to succeed with probability ~1

$$\mathbf{1000000}\ \boldsymbol{operations} = \mathbf{log_2(1000000)} \cong \mathbf{20}\ \boldsymbol{bits}$$

Minimum security level

112-bits at minimum

Source:
Recommendation for Key Management: Part 1 – General
NIST Special Publication 800-57 Part 1 Revision 5

Question time

Does this provide any clue about the time it will take for the attack to complete?

Recall time

Randomness & entropy

Question time

Which string is more random?

1. 0000000000000000000000000
2. kwtlykwtlyRkwtlyRkwtlykwtlyR
3. 0010001000010000010000001
4. awijeoiajwdoiawejoidjannaehj
5. EeEeEeEeeeeeEeEeEeEeeeeee

Randomness as probability distribution

Probability distribution → Probability [0,1] of every event

↓

Uniform distribution → N events → each has $\frac{1}{N}$ probability to occur

Sum of probabilities and entropy

$p_1 + p_2 + p_3 + \ldots + p_n$ → set of all the probabilities in a distribution

Sum of all the probabilities:

$$p_1 + p_2 + p_3 + \ldots + p_n = 1$$

Entropy (i.e.: the measure of uncertainty)

$$-p_1 * \log_2(p_1) - p_2 * \log_2(p_2) - \ldots - p_n * \log_2(p_n)$$

Randomness & entropy in cryptography

Entropy and randomness are key elements in cryptography

Question time

Which elements must be considered while evaluating the randomness of a file/content?

Question time

How much entropy would be perfect to have for cryptographic purposes?

Randomness & entropy in *nix

`/dev/random` → evaluates the level of randomness before providing the requested bits

`/dev/urandom` → does not care and keeps providing bits without checking

Question time

Which one do you prefer?

Why?

Is it really random?

"Statistical test suites like TestU01, Diehard or NIST test suite are one way to test the quality of pseudorandom bits. [...]

But statistical tests are largely irrelevant on cryptographic security, and it's possible to design a cryptographically weak PRNG that will fool any statistical test."

Source:
"Serious cryptography"
Jean-Philippe Aumasson – 1st Ed. – page 29

Wanna learn more?

"The plain simple reality of entropy - Or how I learned to stop worrying and love urandom"

Filippo Valsorda – 32C3

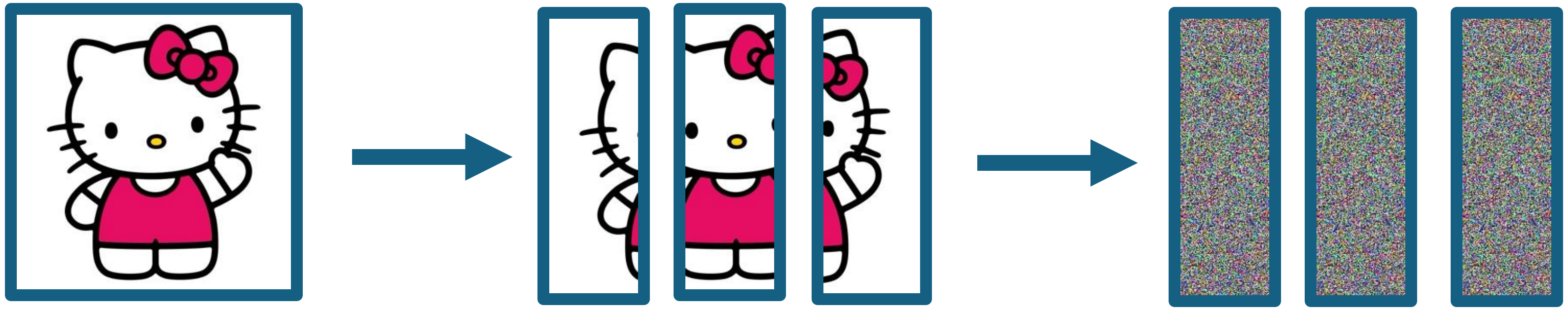
https://media.ccc.de/v/32c3-7441-the_plain_simple_reality_of_entropy

Recall time

Symmetric encryption

Symmetric ciphers

Block ciphers



Stream ciphers

Symmetric encryption

Block ciphers

Block ciphers

~~DES / 3DES (broken)~~

~~RC5 (it depends)~~

~~Blowfish/Twofish (broken/probably secure)~~

AES-128/192/256 (secure, if used properly)

Why should I use AES?

Still secure after more than 20 years

Quite fast

Widely available in software libraries

Hardware implementations are ubiquitous

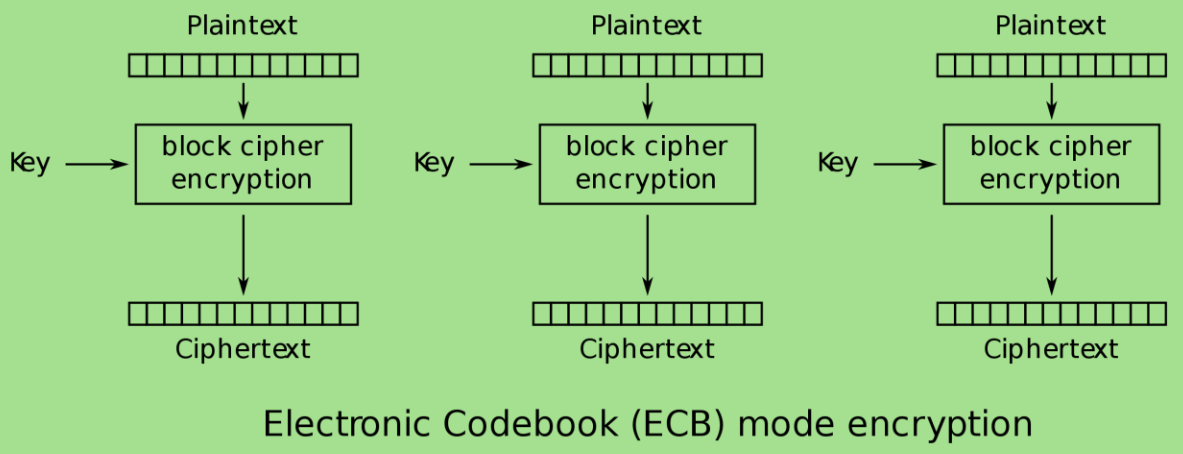
AES characteristics

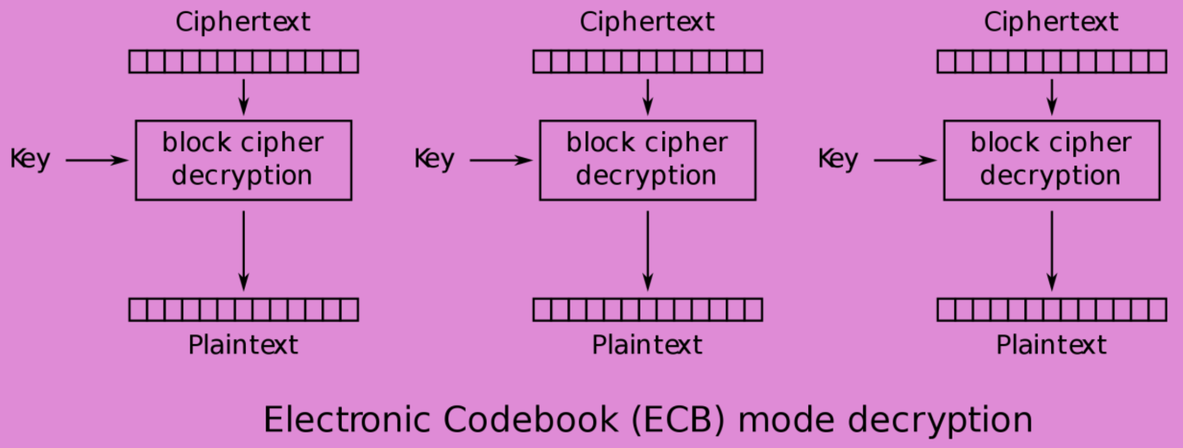
Key can be 128/192/256 bits long

Blocks are always 128 bits long, regardless of the key length

It operates in rounds: 10/12/14 rounds respectively

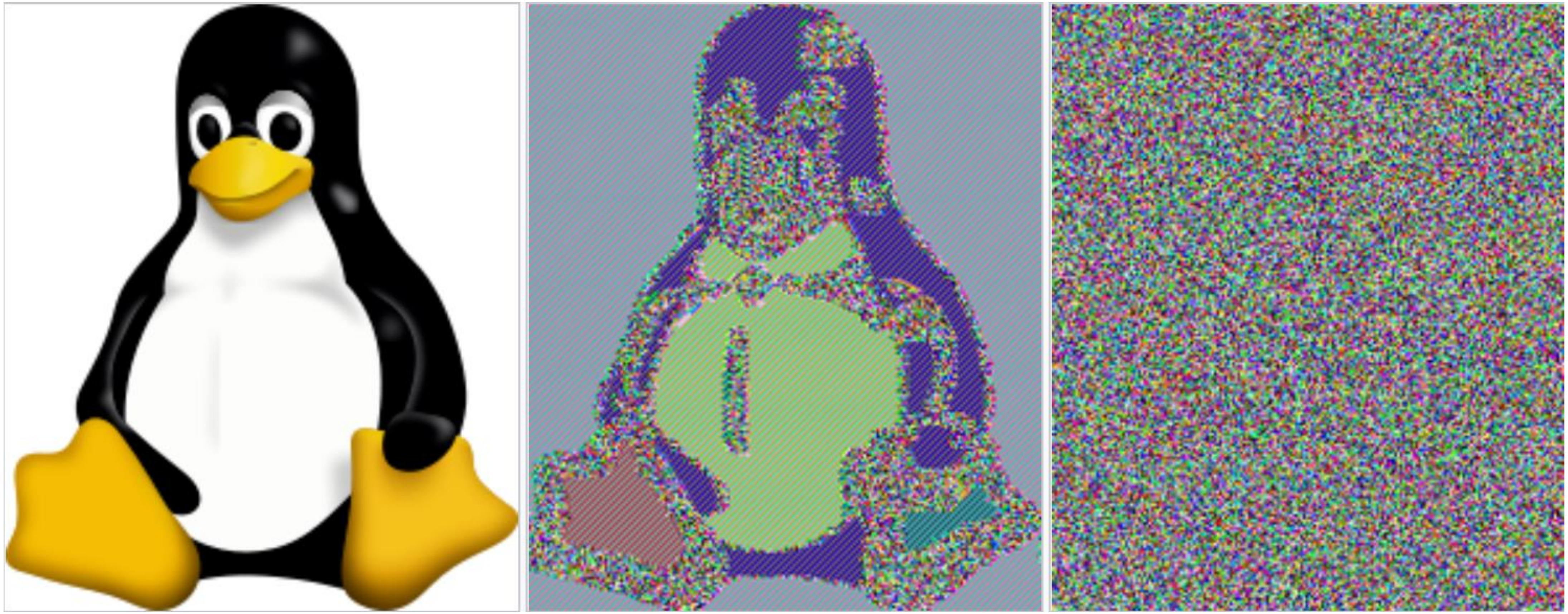
AES-ECB: Electronic Codeblock





Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

AES-ECB: Electronic Codeblock

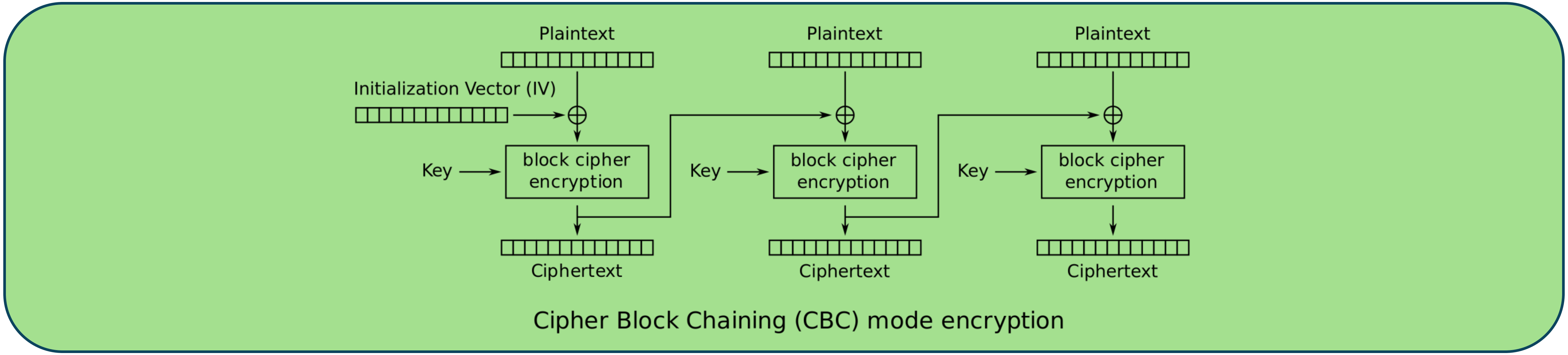


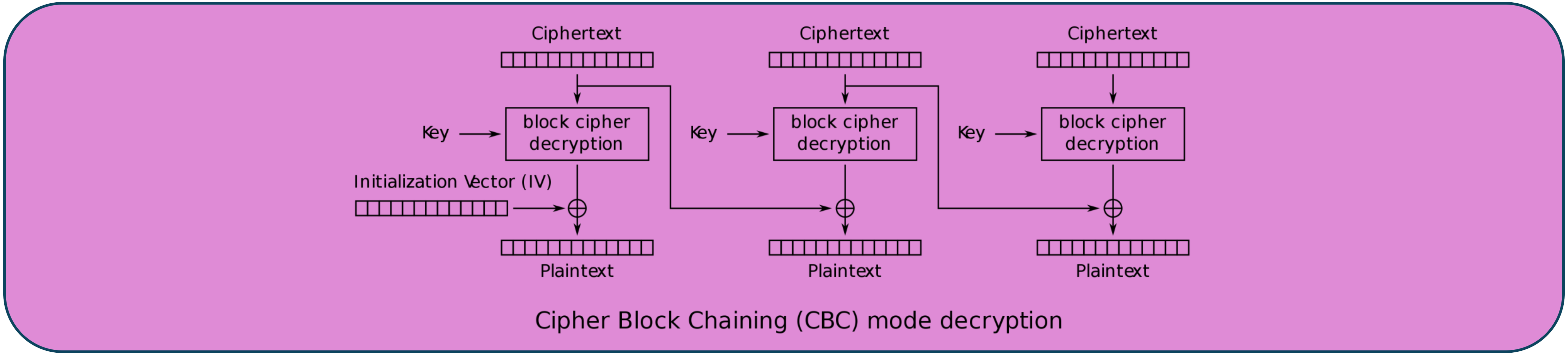
Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

AES-ECB: use it right

Don’t.

AES-CBC: Cipher Block Chaining





Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

AES-CBC: use it right

Initialization Vector (IV) → 128 random bits (= block length)

must change at every iteration

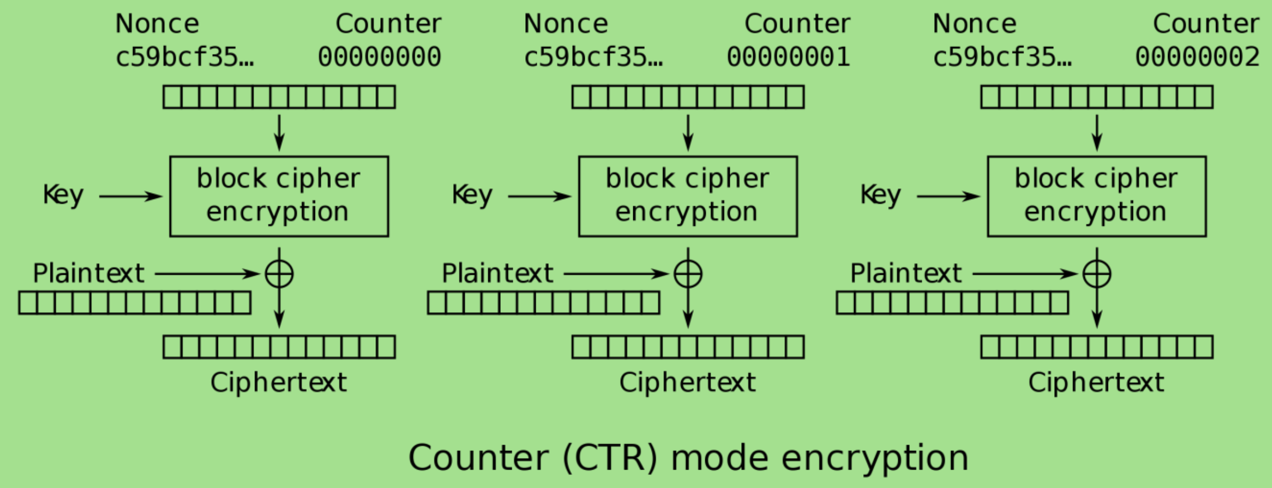
is passed in clear together with **C**

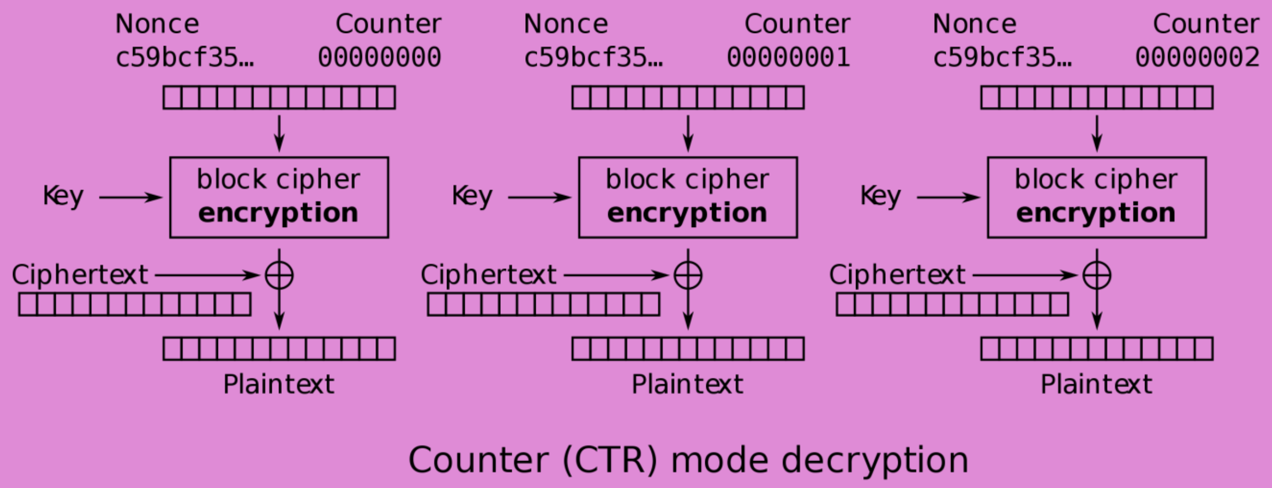
without it, the decryption cannot occur

Padding oracle attack → the implementation pads if needed

during decryption, if the pads is incorrect, report only a generic error

AES-CRT: the counter mode





Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

AES-CRT: use it right

Counter → must be different for every block

is passed in clear together with **C**

must not overflow / reuse the same value

without it, the decryption cannot occur

Nonce → random, must change at every iteration

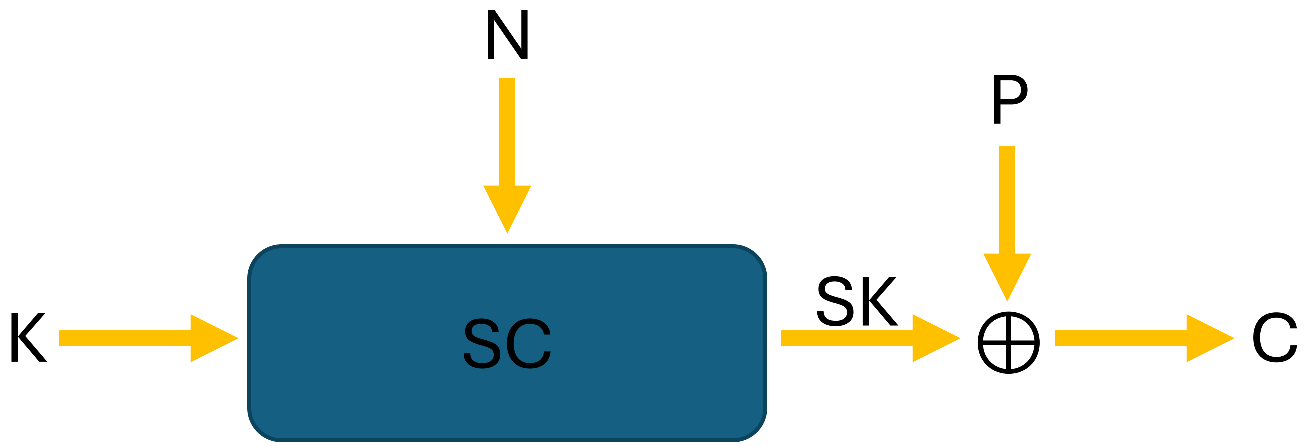
is passed in clear together with **C**

without it, the decryption cannot occur

Symmetric encryption

Stream ciphers

Stream cipher's internals



K	= Key
N	= Nonce
SC	= Stream Cipher
SK	= Stream Key
P	= Plaintext
C	= Ciphertext

It is mandatory for the cipher to be secure that **the nonce is used only once**!

Stream ciphers

Preferred when → constrained resources

AES is not available

padding is eating up too much space

Suggested algorithms → Grain128a

Salsa20

Recall time

Key management

Key length

Normally, the longer **THE KEY**, the better.

For the algorithms we saw → 128-bits is the minimum

192-bits is perfect but rarely used

256-bits is way too much

If possible, use 256-bits key and stop worrying.

Key protection

There are basically three possible solutions.

1. Key wrapping

You encrypt the key with another key/algorithm.

2. On-the-fly generation

Regenerate the key on-the-fly (e.g.: from a password).

3. Hardware secure storage

Put the key inside a secure storage that exposes encryption and decryption functions.

Question time

Can you spot the problems with each of these solutions?

Recall time

Hash and key derivation functions

Properties

Same input	→	Same output
Infinite long input	→	Fixed length output
Any change in input	→	Output is completely different
Affected by collisions	→	Two different input produce the same output
Usage scenario	→	Prove integrity Protect passwords Digital signature

Hash algorithms

~~MD5~~

~~SHA1~~

SHA2 (SHA-224, SHA256, SHA-384, SHA-512)

KEKKAK (aka SHA-3) / SHAKE-128 / SHAKE-256

BLAKE-2

Hashing & salt

Users do use common passwords → If hashed, they look the same

If breached, recovering the password is trivial

Salt → Different for every password

Stored in clear with the password

Precomputed hashes are ineffective

PBKDF2 and Aaron2

PBKDF2

(Password Based Key Derivation Function)

→ Derive a usable key from the user password

Requires iterations (more is better)

Weak if compared to Aaron2 or scrypt

Aaron2

→ Faster than PBKDF2

More secure than PBKDF2

Slowly replacing it

Recall time

RSA

Introduction to RSA

Asymmetric encryption algorithm

Public key (K_{pub}) → can be shared with anyone

used to encrypt

used to verify **K_{priv}** – generated signatures

Private key (K_{priv}) → must be kept private & secure

can be used for deriving the public key

used to generate signatures

RSA security

Key length	Bits of security	Status
512	~ 40	Deprecated
768	~ 60-64	Deprecated
896	~ 70	Deprecated
1024	~ 80	Deprecated
2048	~ 112	Still fine?
3072	~ 128	Should be ok
4096	~ 150	Should be ok

RSA problems

RSA algorithms → based on factorization and exponentiation

keys are used as exponents (more or less)

bigger keys == slower process

RSA key generation → must be done properly

randomness plays a crucial role

don’t do it manually, use **OpenSSL** or **OpenSSH**


Can’t I simply use 65537 as K_{Pub}? Or 3? Or even 1?

Textbook RSA insecurity

Never, ever use RSA in its “textbook” version. It is insecure.

RSA-OAEP → use it for encryption & decryption

RSA-PSS → use it for signature generation & verification

RSA – Do we really need it?

Probably, no.

https://blog.trailofbits.com/2019/07/08/fuck-rsa/

Recall time

Elliptic curves

If possible, move to EC

Asymmetric encryption algorithm

Guarantee 128-bits of security at least

Is a set of algorithms that provide encryption and signing

Way faster than RSA

EC security

Key length	Bits of security	Status
112	~ 56	**Deprecated**
160	~ 80	**Deprecated**
224	~ 112	**Still fine?**
256	~ 128	**Ok to use**
384	~ 192	**Ok to use**
512	~ 256	**Ok to use**

If possible, move to EC

NIST standard curves → with unexplained details that create some suspicious on their security

industry standard

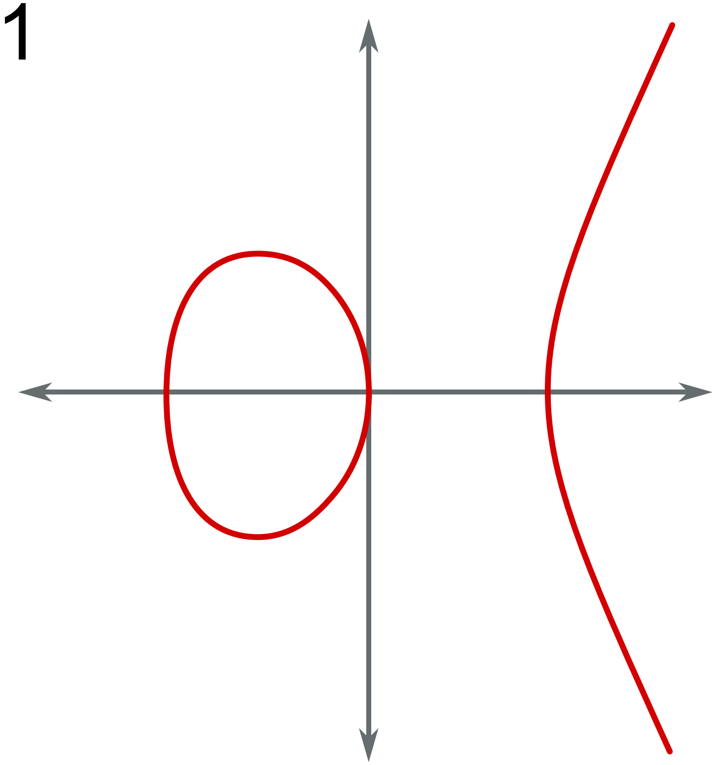
Curve25519 → not NIST certified

completely transparent

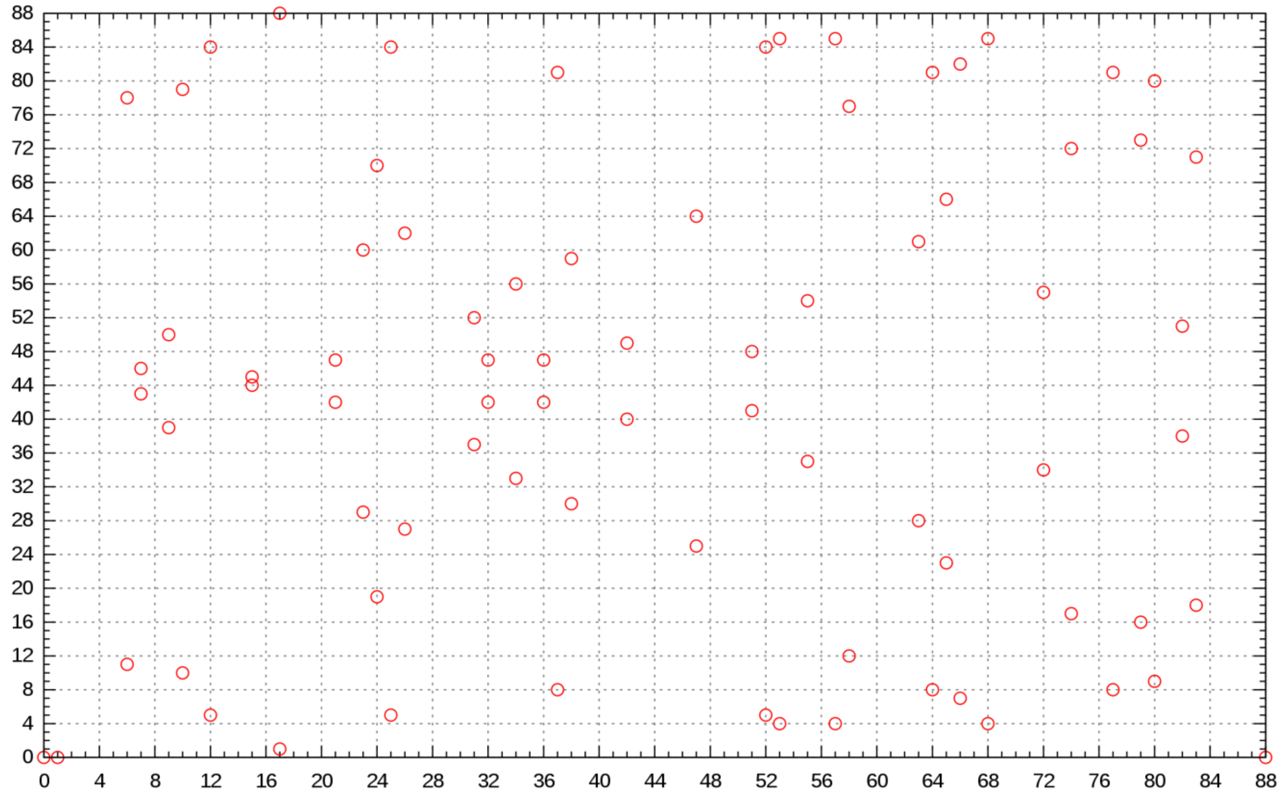
adopted by Google Chrome, Apple systems, OpenSSH and others.

Bonus: why are they called this way?

1



$y^2 = x^3 - x$



Recall time

Symmetric VS asymmetric encryption

Question time

Which one you choose?

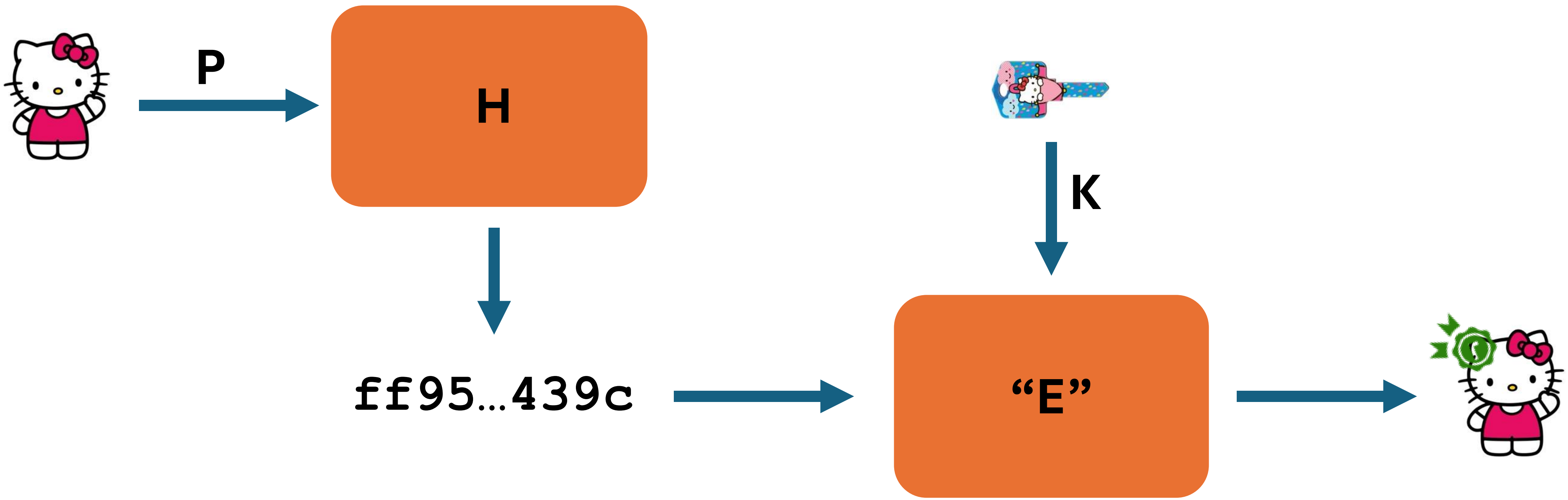
Why you choose it?

Can you use both?

Signing

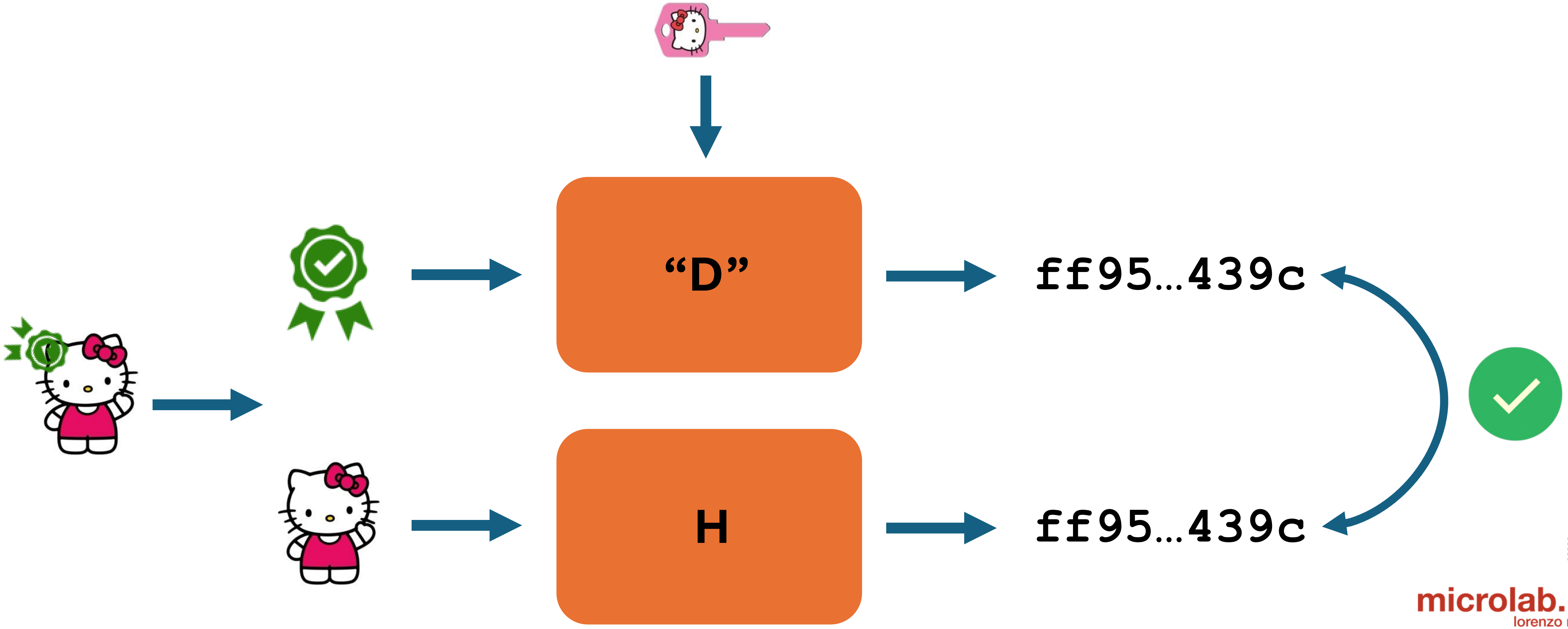
Signature generation

 **Private key** for signature generation



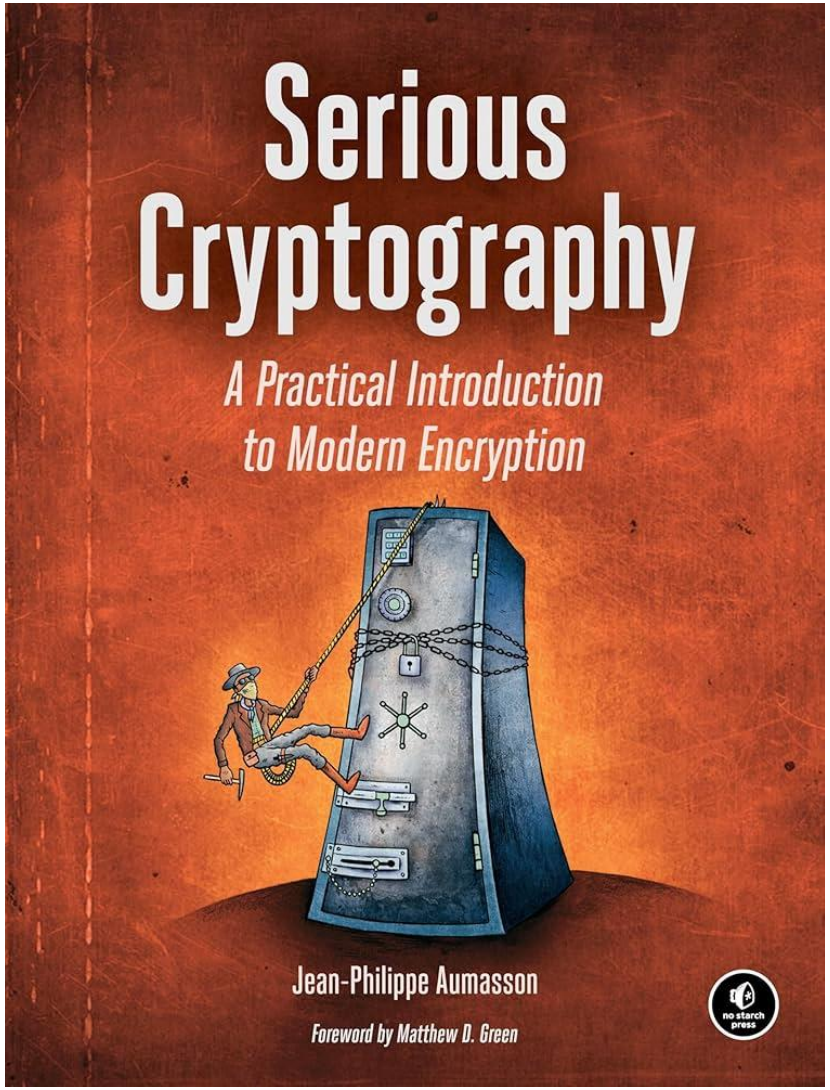
Signature verification

Public key for signature verification



Recall time

Want to learn more?



Question?

Lorenzo Nicolodi

lo@microlab.red