Block Ciphers, Hashes, MACs, the Advanced Encryption Standard (AES), and Key Agreement Protocols (DH)

12th of December 2023

Ing. Gabriel Maiolini Capez
Politecnico di Torino

Who am I?
Telecommunications Engineer (TCE)

By formation:

[2021-] PhD student in Electrical, Electronics, and Communications Engineering

[2017-2021] MSc. in Communications and Computer Network Engineering *(PoliTO, 110)*

[2014-2021] BSc. in Electrical Engineering with Emphasis in Telecommunication (*Polytechnic School of the University of São Paulo, Luiz de Queiroz Orsini Award)*

By trade:

[2022-] Multiple Access Sequences for Satellite Constellations Without Centralized Management *(ESA, TAS-I, CNIT)*

[2022-] Jamming Detection and Mitigation for Satellite Networks (Jeans), *(ESA, PoliTO, MBI, Wiser)*

[2022-] Advisory board for Space-based Data Centres *(ESA, KP Labs, IBM, …)*

[2022-2022] Consultancy services to NEVA SGR (Technical analysis of Leaf Space GSaaS)

[2021-2022] Lunar Radionavigation System (LRNS) *(ESA, PoliTO, TAS-I, Telespazio)*

[2021-2022] Principal Investigator: On the Use of Mega-Constellation Services in Space *(ESA, PoliTO, Vyoma GmbH, Surrey Space Centre, Auckland University)*

[2021-2022] (Satellite) Telecommunications Engineer *(Vyoma GmbH)*

[2021-2021] Sviluppo di transceiver per applicazione satellitare (short codes) *(PoliTO, Argotec)*

[2020-2020] Cubesat communication link and OSD applications *(PoliTO)*

[2019-2020] Narrowband Internet of Things Via Satellite *(PoliTO)*

[2018-2020] Design and Verification of a CubeSat Ground Station (OPS-SAT) *(PoliTO)*

[2016-2017] Use of Parallel Processing Accelerators for Real Time Brain Connectivity *(EPUSP)*

A few publications:

- *[Under review]* ***Patent on Alternative Code Direct Sequence Spread Spectrum***
- *[Under review]* ***On the Use of Mega Constellation Services in Space***
- ***Characterisation of Mega-Constellation Links for LEO Missions with Applications to EO and ISS Use Cases***
- *[ESA TT&C 2022]* ***An Academic Ground Station as a Service (GSaaS) Devoted to CubeSats***
- *[IEEE TAES 2022]* ***Sparse Satellite Constellation Design for Global and Regional Direct-to-Satellite IoT Services***
- *[SpaceOps 2021]* ***Distributed Space Traffic Management Solutions with Emerging New Space Industry***
- *[SpaceOps 2021]* ***On the use of Pseudo-Noise Ranging with high-rate spectrally-efficient modulations***
- *[IAC 2019]* ***CubeSat Control Centre: A development of an Educational Control Centre to Support CubeSat Space Operations***

Agenda

- Block Ciphers
- Substitution Permutation Networks
- Advanced Encryption Standard (AES)
  - Mathematical Preliminaries
  - Introduction
  - State
  - Algorithm
- Block Modes (EBC, CBC, CTR)
- Hash Functions (SHA3)
- Keyed Hashing (MAC, HMAC)
- Authenticated Encryption (GCM)
- Key Agreement Protocols (DH)

References:

Serious Cryptography – Jean-Philippe Aumasson

A Graduate Course in Applied Cryptography – Dan Boneh and Victor Soup

Introduction to Cryptography with Coding Theory - Lawrence C. Washington

The Rijndael Block Cipher (nist.gov)

Definitions

- Plaintext: unencrypted message
- Ciphertext: encrypted message
- Permutation: ***an invertible*** transform such that each item (i.e., binary sequence) has an ***unique*** inverse. To be secure:
  - Determined by the key
  - Different keys should produce different ciphertexts
  - Look random (to avoid patterns in the ciphertext revealing information about plaintext)
  - *Hard* to reverse without the key (computationally infeasible)
- Mode of operation: algorithm that uses the ***permutation*** to process messages of arbitrary size
- Classic ciphers were of limited complexity, ***modern ciphers*** such as AES have much greater complexity and security

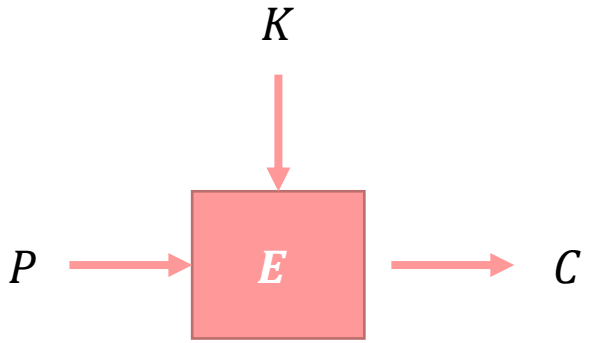
Block Ciphers

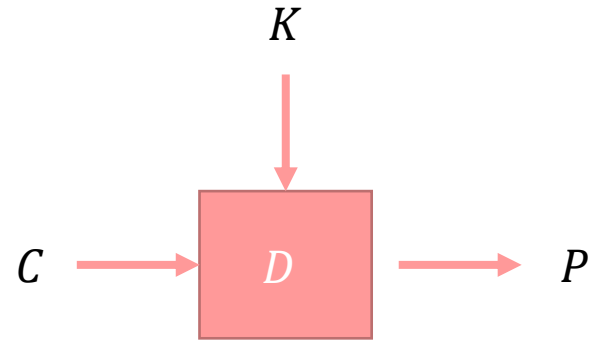
Set of encryption ($\boldsymbol{E}$) and decryption ($\boldsymbol{D}$) algorithms

- Encryption algorithm: $\boldsymbol{E}$ takes a key $\boldsymbol{K}$ and a plaintext block $\boldsymbol{P}$ to produce a ciphertext block $\boldsymbol{C} = \boldsymbol{E}(K, P)$

- Decryption algorithm: $\boldsymbol{D}$ takes a ciphertext $\boldsymbol{C}$ and a key $\boldsymbol{K}$ to produce a plaintext block $P = \boldsymbol{D}(C, K)$

Observation: the secrecy of the algorithm itself is not necessary and can be detrimental (Kerckhoff's Principle)





Block Ciphers

- Without K, attackers cannot derive information from $C = \boldsymbol{E}(K, P)$, nor identify patterns between C and P. That is, $\mathrm{H}(P|C) = 0$.

- Given black-box access to $\boldsymbol{E}$ and $\boldsymbol{D}$ it should be impossible to distinguish the output from a truly random permutation for some fixed unknown key

Two parameters are key to security:

- Block Size (i.e., 128-bit for AES)
- Key Length (i.e., 128/256-bit for AES)

Block Size	Memory
16-bit	128 KB
32-bit	16 GB
64-bit	128 EB

Trade-offs: binary, complexity (#operations, type of operations, memory, *implementation*), overhead (splitting blocks)

Codebook attack: if the block is too small (i.e., 16-bit), you can efficiently build a so-called codebook, that is, a look-up table (LUT) of all possible plaintexts to decrypt unknown ciphertext blocks

Block Ciphers

Block ciphers rely on a *sequence or repetition of many, simpler, rounds to be strong*.

Two main constructions:

- Substitution-Permutation Network (AES) *[today]*
- Feistel (i.e., DES, GOST)

Let $\{\boldsymbol{R_1}, \ldots \boldsymbol{R_N}\}$ be a set of *invertible transforms*. Then,

$$C = \boldsymbol{R_N}\left(\boldsymbol{R_{N-1}}\left(\boldsymbol{R}_{\ldots} \quad (\boldsymbol{R_1}\,(P))\right)\right)$$

$$P = \boldsymbol{R_N^{-1}}\left(\boldsymbol{R_{N-1}^{-1}}\left(\boldsymbol{R}_{\ldots}^{\boldsymbol{-1}} \quad (\boldsymbol{R_1^{-1}}(P))\right)\right)$$

Rounds are usually identical, but every round is parametrised by a *round key. Different round keys produce different outputs.*

Block Ciphers

But how do you generate a round key?

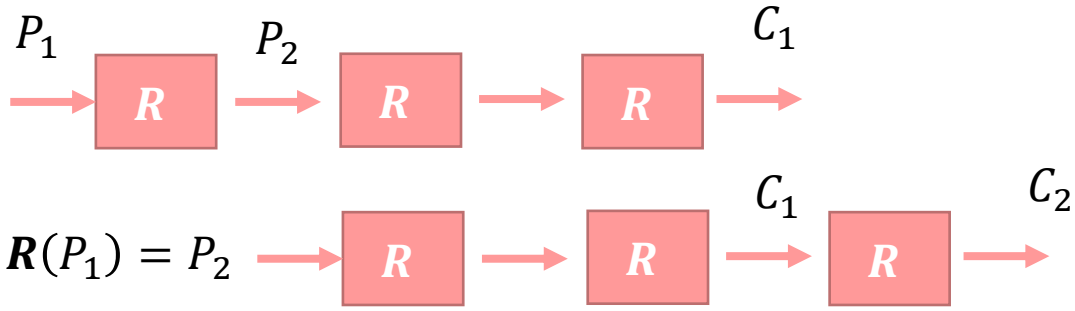
Round keys $\{K_i\}$ are derived from the main key K following a key schedule algorithm.

Every round should use a different key to avoid slide attacks

Knowing the output of a single round and the relationship between (P_1, C_1) and (P_2, C_2) *can help recover the key.*

Short simple rounds with round keys increase robustness against side-channel attacks

(+) If the key schedule is not invertible, gaining information about K_i does not allow attackers to find K (an issue with AES, where K can be recovered from any K_i)



Substitution Permutation Networks (SPN)

Two methods of "frustrating statistical analysis":

- *Diffusion (depth)*: "statistical structure of a message which leads to its redundancy is 'dissipated' into the long-range statistics" [Shannon]

Translation: "transformations depend equally on ***all bits of the input***." [Aumasson]

- *Confusion (breadth)*: "to make the relation between the simple statistics of E and the simple description of K a very complex and involved one." [Shannon]

Translation: "the ciphertext statistics should depend on the plaintext statistics in a manner too complicated [complex transformations] to be exploited by the cryptanalyst." [Massey]

Confusion + Diffusion → Substitution + Permutation

S-Boxes

Substitution ***Boxes:*** small LUTs that transform *k* input bits into *k* output bits

"Must be carefully chosen to be cryptographically strong: should be as non-linear as possible [...] and have no statistical bias" [Aumasson]

Additional Rijndael S-box Properties:

- Inverse mapping of the initial byte $\underline{x} \rightarrow \underline{x}^{-1}$ (non-linearity)
- Matrix rows are linear shifts of each other
- No input equals its S-box output or the complement of its output

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Source: The Rjindael Block Cipher S-Box Representation

Advanced Encryption Standard (AES)

Mathematical Preliminaries

In AES, operators are defined at the ***byte,*** that is, $GF(2^8)$ and ***word*** (4 bytes) levels.

Polynomial representation: $b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

Properties:

- Associative
- Commutativity
- Identity
- Inverse

Mathematical Preliminaries

Addition: sum of coefficients $\boldsymbol{mod}$ 2, byte-level eXclusive OR (XOR).

Multiplication: (mult. of polynomials $\boldsymbol{mod}$ irreducible polynomial $\underline{m(x)}$)

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

Issue: $c(x) = a(x)b(x)$ where both $a(x)$ and $b(x)$ are 4-bytes longs may not be representable by 4 bytes (word).

Solution (degree reduction): $d(x) = c(x)\ mod\ M(x)$, where $M(x)$ is an invertible polynomial of degree 4.

Then, $d(x)$ can be written as a multiplication of a circulant matrix:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Mathematical Preliminaries

Multiplication tricks. We want to compute: $b(x) * x \ mod \ m(x)$

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

$$b(x) * x = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

No simple byte-level operation!

However,…

If $b_7 = 0$, the reduction operation is the identity - we already have a polynomial of degree <= 7

If $b_7 = 1$, $m(x)$ must be subtracted (XOR). At the byte-level, this is a left shit (multiplication) and a conditional bit-wise XOR

Hardware support for this operation: ***xtime***

Mathematical Preliminaries

Coming back to our degree reduction, $c(x) = b(x) * x \; mod \; M(x)$ can be written in matrix form such that every multiplication by x is a cyclic shift of the bytes inside the word.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

AES - Introduction

Design goals:

- Resistance against known attacks
- High security
- Speed and code compactness across a variety of platforms and applications [efficiency]
- Simplicity

Three core layers:

- Linear mixing: for high diffusion across multiple rounds
- Non-linear: parallel application of S-boxes with optimum non-linearities
- Key addition: XOR of the round key to an intermediate state

Years of competition. Approved by NIST, implemented worldwide (“de-facto standard”). Cleared for top-secret information by the NSA.

AES - State

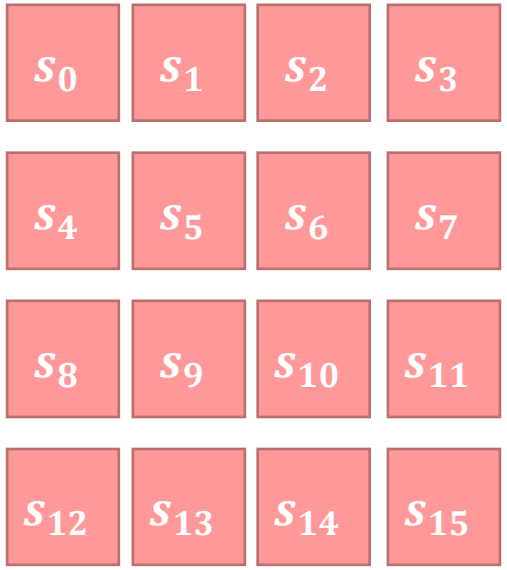
Intermediate cipher results are called ***State***

AES works on bytes, ***processing 128-bit blocks*** $\mathrm{s} = (s_0, s_1, \ldots, s_{15})$, composed of 4x4 byte arrays.

Key size (bit): ***128 (today)***, 192, and 256

Number of rounds: 10, 12, 14 to protect against known short-cut attacks with margin

By transforming bytes, columns, and rows, the final ciphertext is produced.

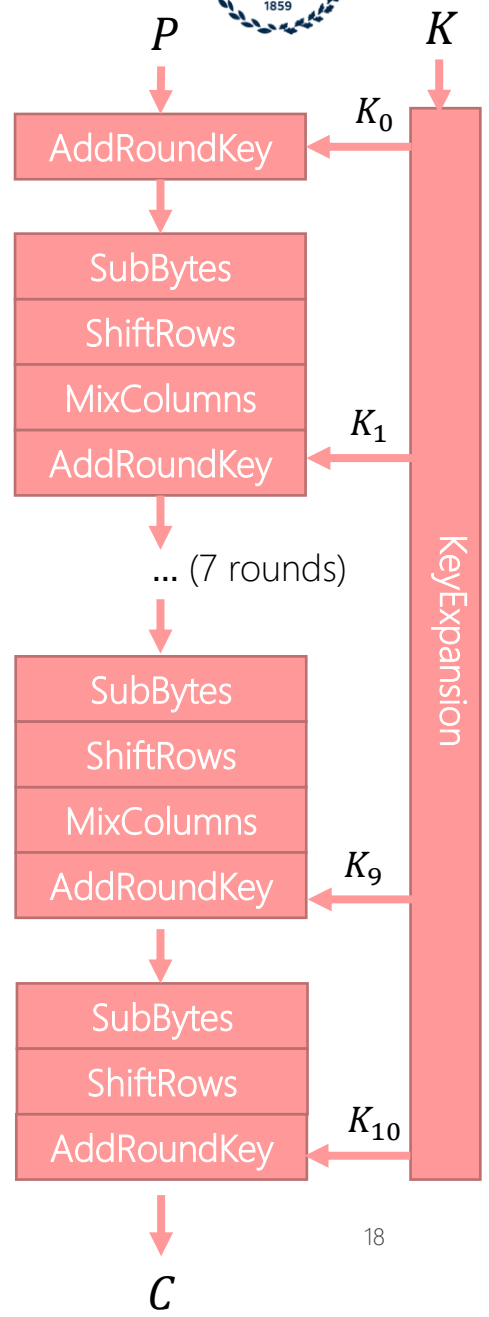


AES - Algorithm

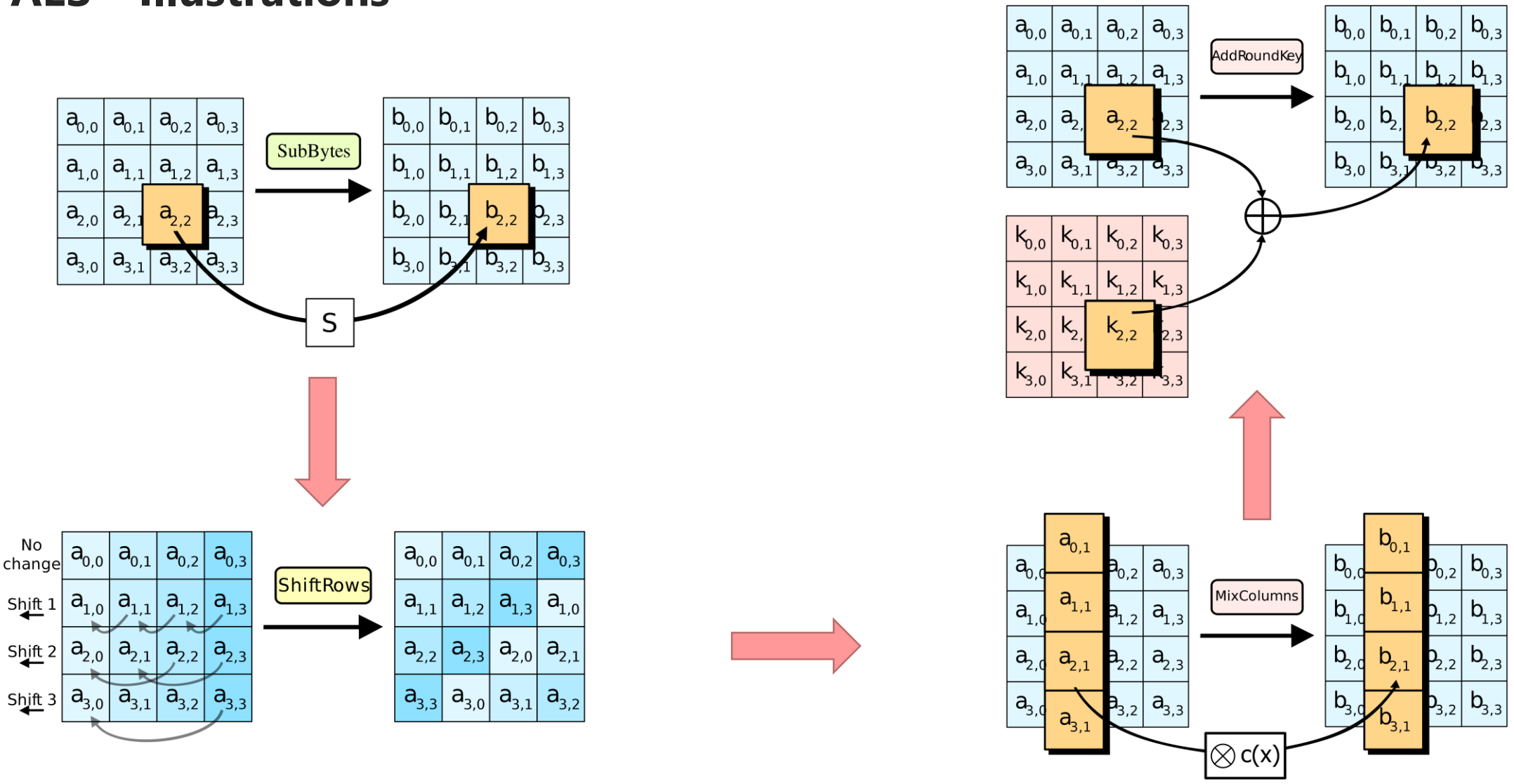
Five core blocks, 10 rounds:

1. AddRoundKey: XOR between state and round key [dependency on key, slide attack protection]
2. *SubBytes*: (non-linear) byte *substitution* using S-box [confusion, security]
3. *ShiftRows*: *(permutation)* shifts i-th row cyclically by i positions [diffusion, one column change propagates to all (code-book attack protection)]
4. *MixColumns*: *(permutation)* shifts j-th column cyclically by $\mathbf{j}$ positions [diffusion, changing a byte affects all bytes in the state (chosen plaintext attack protection)]
5. *KeyExpansion*: *(key schedule)* creates $\{K_i\}$ 128-bit keys from the original 128-bit key

Decryption operates in reverse



AES – Illustrations



By User:Matt Crypto - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1118874

AES – Algorithm

Key Schedule: need 10 rounds + 1 block lengths of keys (128 > 1408 bits)

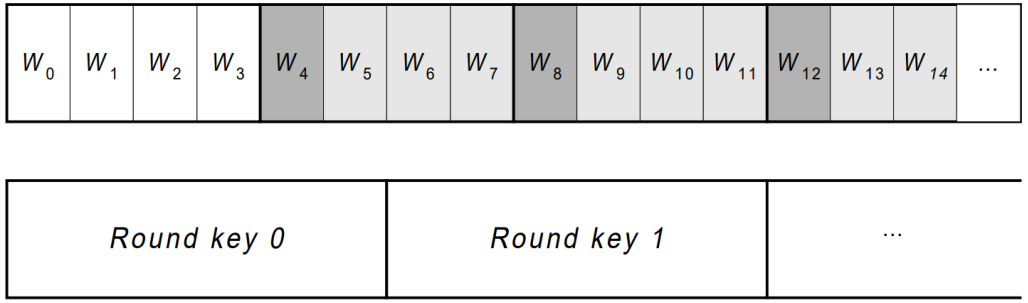
- *Key expansion:*
  - First Nk words contain the Cipher Key
  - Other words derived recursively
  - SubByte applies the Rijndael S-box to the Byte
  - RotByte is a cyclic permutation

- *Round Key Selection:* round key ***i*** given by buffer words W[Nb*I, Nb*(i+1)]

- Properties:
  - Fast & Simple
  - Chosen to be invertible (issue: knowing the round key (by a side-channel attack) allows the main key to be reversed. Hard, but happened in practice)
  - Round constants eliminate symmetries between rounds
  - Diffuses Cipher Key differences into Round Keys
  - Sufficiently non-linear to prevent determining round key differences from cipher key differences

```
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
{
      for(i = 0; i < Nk; i++)
            W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);

      for(i = Nk; i < Nb * (Nr + 1); i++)
      {
            temp = W[i - 1];
            if (i % Nk == 0)
                  temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
            W[i] = W[i - Nk] ^ temp;
      }
}
```



Source: Rijndael Proposal

AES - Algorithm

Initial key addition: *"any layer after the last key addition in the cipher (or before the first in the context of known-plaintext attacks) can be simply peeled off without knowledge of the key and therefore does not contribute to the security of the cipher"* [Rijndael]

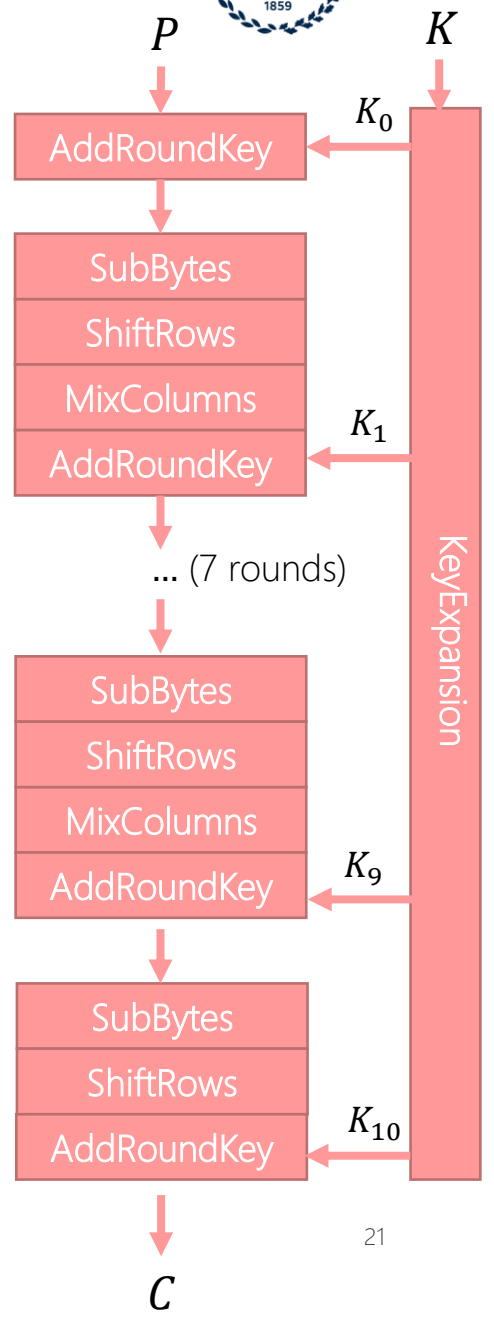
Last round: no MixColumns (linear transformation) to make encryption/decryption more similar without impacting security.

Implementation: table-based + hardware-based instructions (AES co-processor)

Security: as secure as possible. 15 years of research, hundreds of publications, essentially no reduction in security

But…

- *Encryption modes must be correctly selected and used (hard)*
- *Side-channel attacks must be defended against (constant time, power)*



Politecnico di Torino

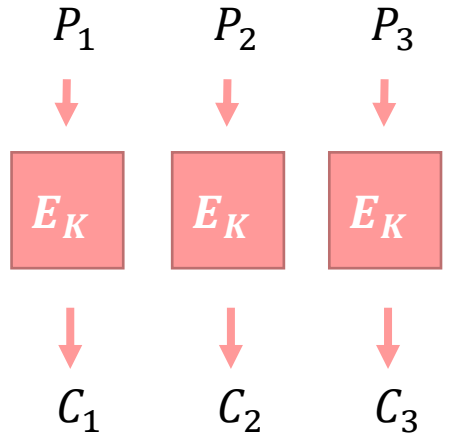
But how do we go from a 128-bit message to an arbitrary message length?

AES Block Modes

AES – Electronic Code Book (ECB) Mode

Simplest "mode":

1. Break a plaintext $P = [P_1 | P_2 | \ldots | P_N]$ into smaller chunks of appropriate size
2. Encrypt each chunk independently with the same key
3. Combine all chunks to form the ciphertext $C = [C_1 | C_2 | \ldots | C_N]$



Insecure, you should never use it:

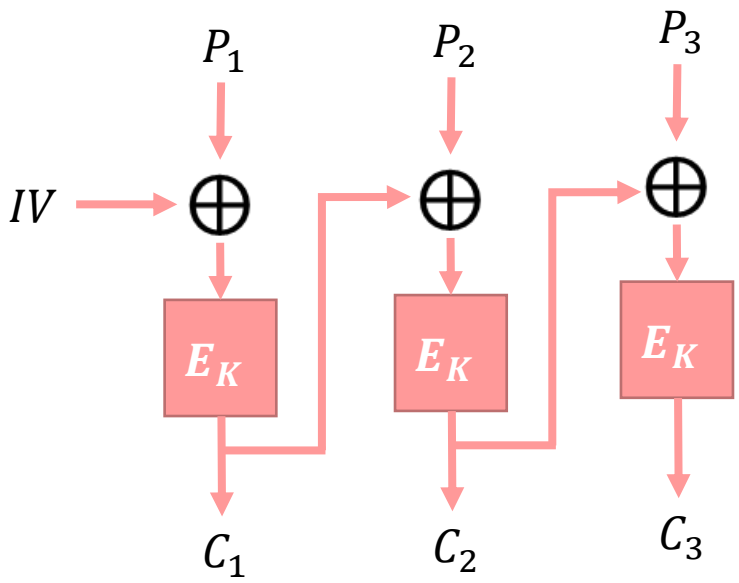
- Identical (plaintext block, key) produce identical cipher blocks, even across different ciphertexts
- In the long run, if an eavesdropper (Eve) finds out some plaintexts and their corresponding ciphertexts, it can build up a codebook to use in decrypting messages *without knowing the key (i.e., e-mail message headers)*

Source: Filippo.io

AES – Cipher Block Chaining (CBC) Mode

A much more secure alternative: block chaining

Starting from a ***random initialization vector (IV)***, we encrypt the i-th block as:
$C_i = \boldsymbol{E}(K, P_i \oplus C_{i-1})$



Thus,

- The ciphertext of the current block depends on all previous blocks
- By using a ***random IV***, identical plaintexts will produce different ciphertexts every time!

But how do we decrypt? *The IV must be sent in the clear along with the first ciphertext.*

AES – CBC Block Padding

But hold on… my plaintext's length is still a multiple of the block size.

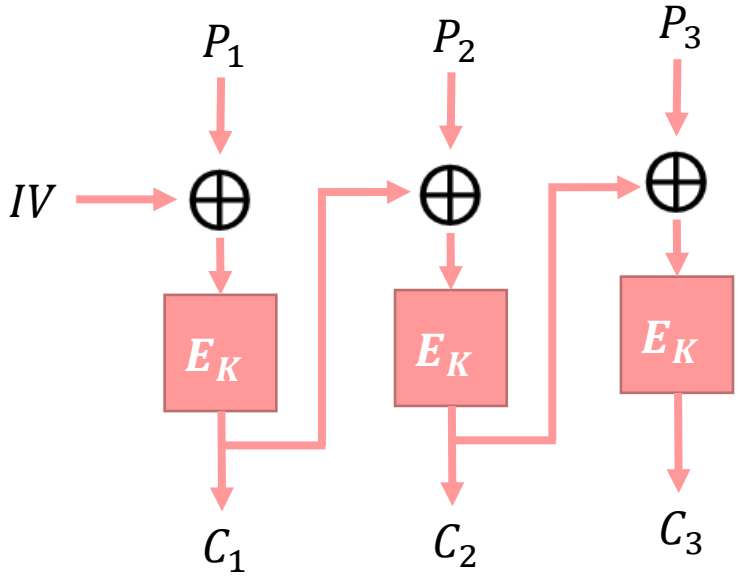
That's not arbitrary!

Indeed, to work with arbitrary lengths we must ***pad*** *the data (i.e., using special sequences) until we have a suitable message for encryption.*

Why special sequences (i.e., "post-amble")?

We need to distinguish data from padding!

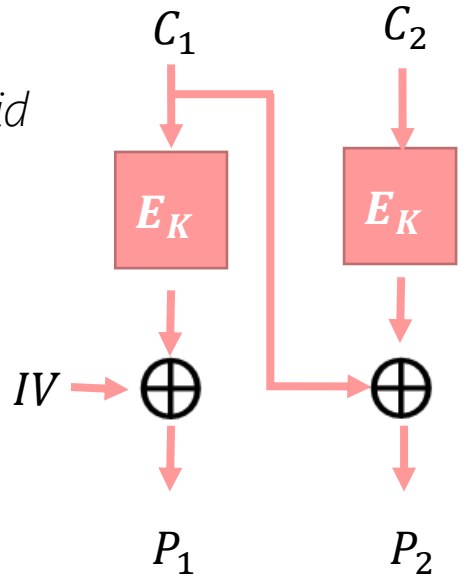
However…



AES – Padding Oracle Attacks

Padding oracle: *"a system that behaves differently based on whether the padding is valid or not" [Aumasson] (i.e., web request success/error).*

We can exploit this information (i.e., valid/invalid) to decrypt chosen cyphertexts.



Idea: Discover P_2 by asking the oracle if our guesses are correct.

We pick a random C_1 and send $(C_1 \mid C_2)$ to the oracle.

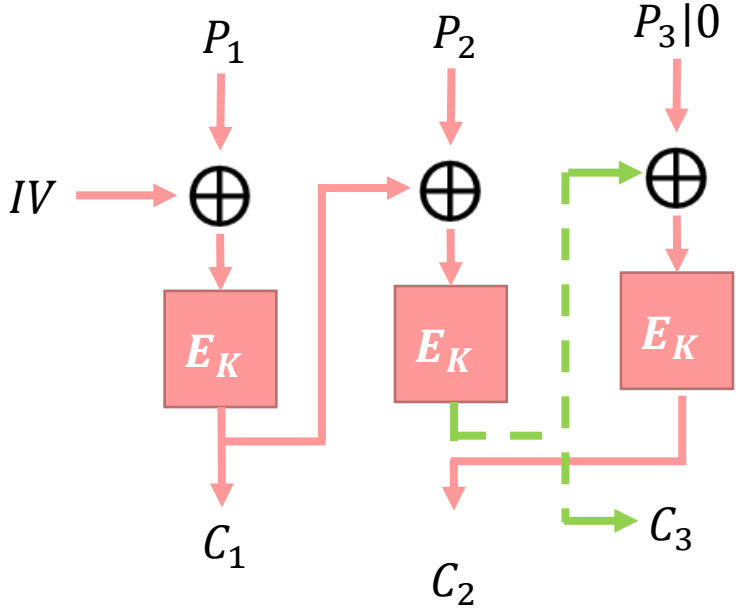
1. *Oracle will tell us* if $C_1 \oplus P_2$ *has a valid padding, allowing us to solve for a byte of* P_2 *(i.e.,* $C_1[15] \oplus P_2[15] = Pad\ Byte$*)* . If not, we change $C_1[15]$ until we obtain a positive answer.
2. We now find the value of $P_2[14]$ by setting $C_1[15] = P_2[15] + Pad\ Byte$ and repeating for $C_1[14]$
3. *We iterate (2,3) until all bytes have been decrypted.*

(harder in practice)

AES – CBC Cipher Stealing

A rarely used alternative that protects against padding oracle is *cipher stealing, that is, extending the bits of a plaintext using the previous ciphertext block*

Hard to implement correctly



AES – Counter (CTR) Mode

I don’t like padding after that fiasco… but cipher stealing is complex…

Stream ciphers were much simpler… we just generated pseudorandom bits from a key and added them to the plaintext

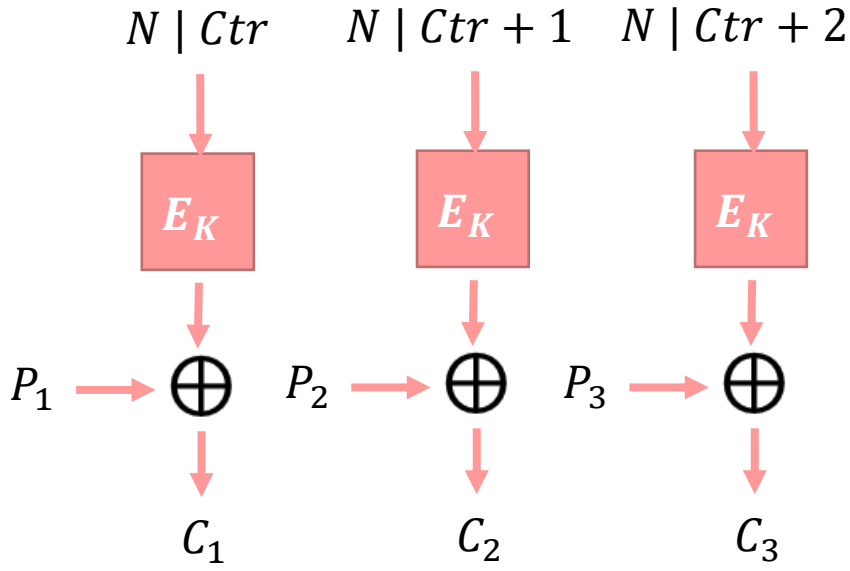
Can’t we go back to that? *Why, yes, we can. Let’s enter AES-CTR.*

AES – Counter (CTR) Mode

Using a random ***Nonce (N)*** and an integer ***Counter (Ctr)***, we *encrypt the nonce (transmitted in clear) + counter to obtain our pseudorandom bits*

Fast and easy, right? But remember: *nonces must never be reused with the same key for different messages, this can lead to complete failure.*

And… counters must be big enough for the amount of data you want to encrypt!



Still one mode to understand….

AES-GCM

But we must cover a lot of ground to get there…

Hash Functions

Definition

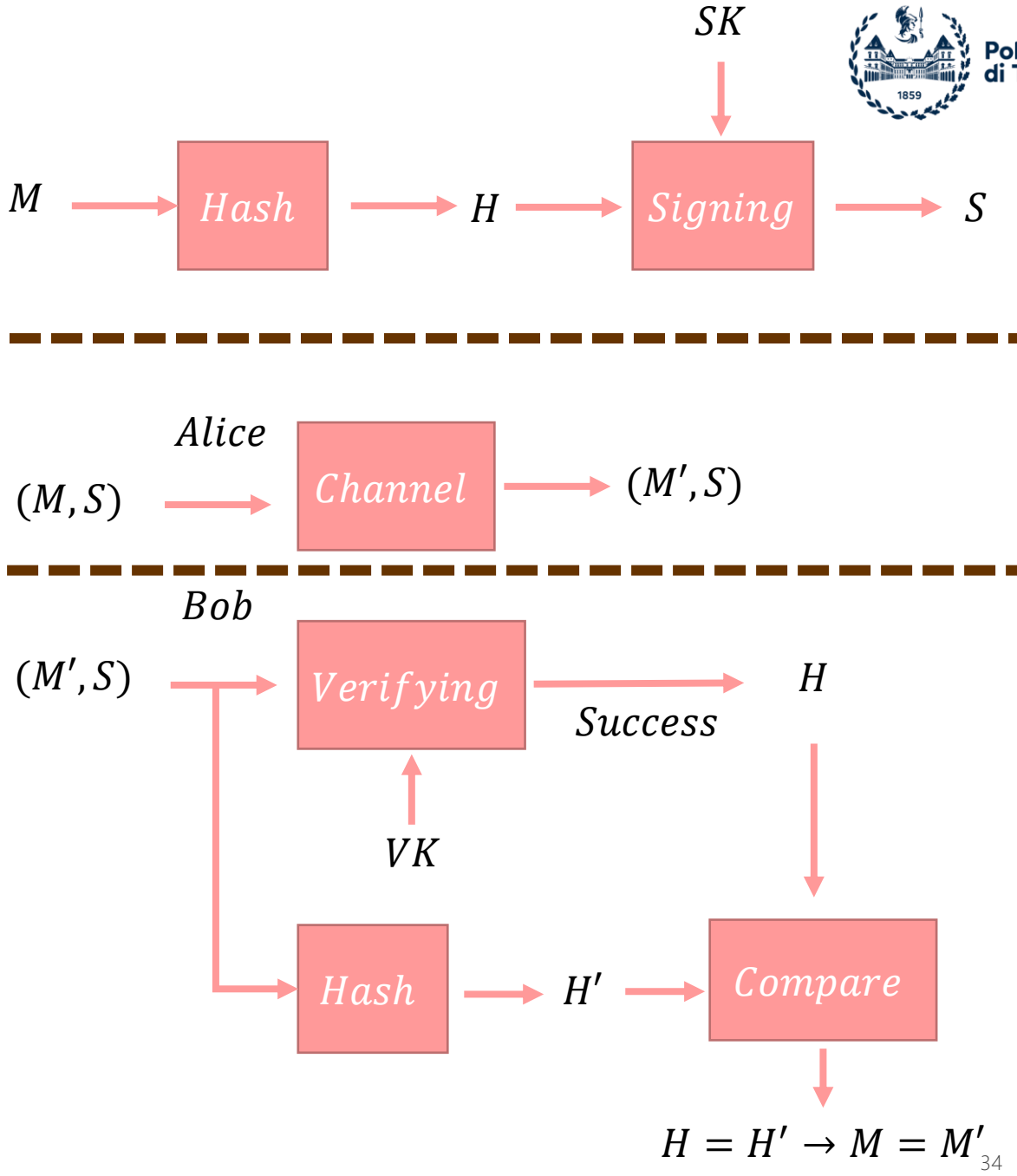
- A hash function $Hash$ takes an arbitrary length message M and produces a short fixed-length hash H (i.e., 128, 256, … bit), called a *hash value or digest.*

- Unlike a cipher, hashes do not protect *confidentiality*, but *integrity*. They ensure that the M has not been modified, as any change to M changes H.

- Multiple applications
  - Digital Signature Schemes
  - Public-Key Encryption
  - Integrity Verification
  - Authentication
  - Key Agreement

Digital Signature Schemes

- *Because hashes are unique to each message, they can be used as identifiers of that message.*
- *By signing the hash H using a signing key $\boldsymbol{SK}$ to produce a signature $\boldsymbol{S}$, the integrity of the message is as protected as if the message itself had been signed.*
- This is the foundation of digital signature schemes.
- *Applications process the hash and its signature rather than the message itself*
- *At much reduced complexity, because len(M) << len(H).*
- *Received message and signature verified using a verification key. Verification should fail if signature has been tampered with.*



Properties of a Secure Hash Function

A *secure hash function must behave as a truly random function*, that is, it should be impossible to predict the output from its input; even if inputs are related to each other.

In practice, they should satisfy two properties:

- *Preimage resistance*
- *Collision resistance*

Preimage Resistance

- A *preimage* of a hash H is any message M such that $H = Hash(M)$

- *Preimage resistance* means that for any random hash H, an attacker will never find a preimage of that hash. Put in a simpler way, a secure hash function is a *one-way function and is impossible to invert, even with unlimited computing power.*

- It should be impossible not only to find the message that was used to generate a hash, *but it should also be impossible to find* ***any*** *message that maps to a given hash.*

Message Length [bit]	# Preimages Per Hash
512	$2^{512}/_{2^{256}} = 2^{256}$
1024	$2^{1024}/_{2^{256}} = 2^{768}$

Why? Because many messages can hash to the same value.

Preimage Resistance

- For a N-bit hash function, there are only $\mathbf{2^N}$ possible hashes. But there are infinite messages (i.e., N, N+1, …, 2N, … N^N bit messages) that can hash to those $\mathbf{2^N}$ bits.
- Since a secure hash function behaves like a random function, the optimal way to search for the message that produces a hash would be…
- To randomly generate messages and compute their hashes, that is, brute force.
- On average, $\boldsymbol{2^N}$ *attempts are needed before finding a pre-image. N = 256? Hopeless.*

Message Length [bit]	# Preimages Per Hash
512	$2^{512}/_{2^{256}} = 2^{256}$
1024	$2^{1024}/_{2^{256}} = 2^{768}$

Collision Resistance

As indicated in the prior slides, many messages can hash to the same value. Thus, *collisions exist.*

This is the *pigeonhole principle*; if you have $M > N$ pigeons to put in N holes, at least one hole must contain more than one pigeon.

However, in a secure hash, *finding collisions must be as hard as recovering the original message*

Unfortunately, due to the *birthday attack*, finding collisions is easier than finding preimages (~ $2^{N/2}$)

Birthday Attack (I)

A secure hash function is a random function that distributes messages into a set of bins (hashes) with equal probability.

Let $0 \leq p_M \leq 1$ be the probability of collisions for M messages hashed to N possible hashes (bins). Then:

$$p_0 = p_1 = 0, p_M = 1$$

Let $q_j = 1 - p_j$ be the probability of no collisions among j hashes, $q_0 = q_1 = 1$, and p_j be the probability of at least one collision among j hashes.

1. $q_1 = \mathrm{P}(H_2 \neq H_1) = \mathrm{P}(A) = \left(\frac{N-1}{N}\right)$
2. $q_2 = \mathrm{P}\left((H_3 \neq H_1 \cap H_3 \neq H_2) \cap (H_2 \neq H_1)\right) = \mathrm{P}(B \cap A) = P(A)P(B) = \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right) = \left(\frac{N-1}{N}\right)\left(1 - \frac{2}{N}\right)$
3. …
4. $q_{j+1} = q_j(1 - {}^{j}/_{N})$

Birthday Attack (II)

Then,

$$p_M = (1 - q_M),\ q_M = \prod_{j=0}^{M-1}\left(1 - \frac{j}{N}\right) = \frac{N!}{(N-M)!(N^M)} = C(N,M)\left(\frac{M!}{N^M}\right)$$

$$0 \leq \left(1 - {}^{j}/_{N}\right) \leq 1, \qquad \forall\ (j \leq N) \in\ \mathbb{Z}$$

Applying the inequality $1 - x \leq e^{-x}$,

$$\prod_{j=0}^{M-1}\left(1 - \frac{j}{N}\right) = \prod_{j=0}^{M-1} e^{-i/N} = e^{-\sum_{j=0}^{M-1} i/N} = e^{M(M-1)/(2N)} \approx constant\ * 2^{N/2}$$

Thus,

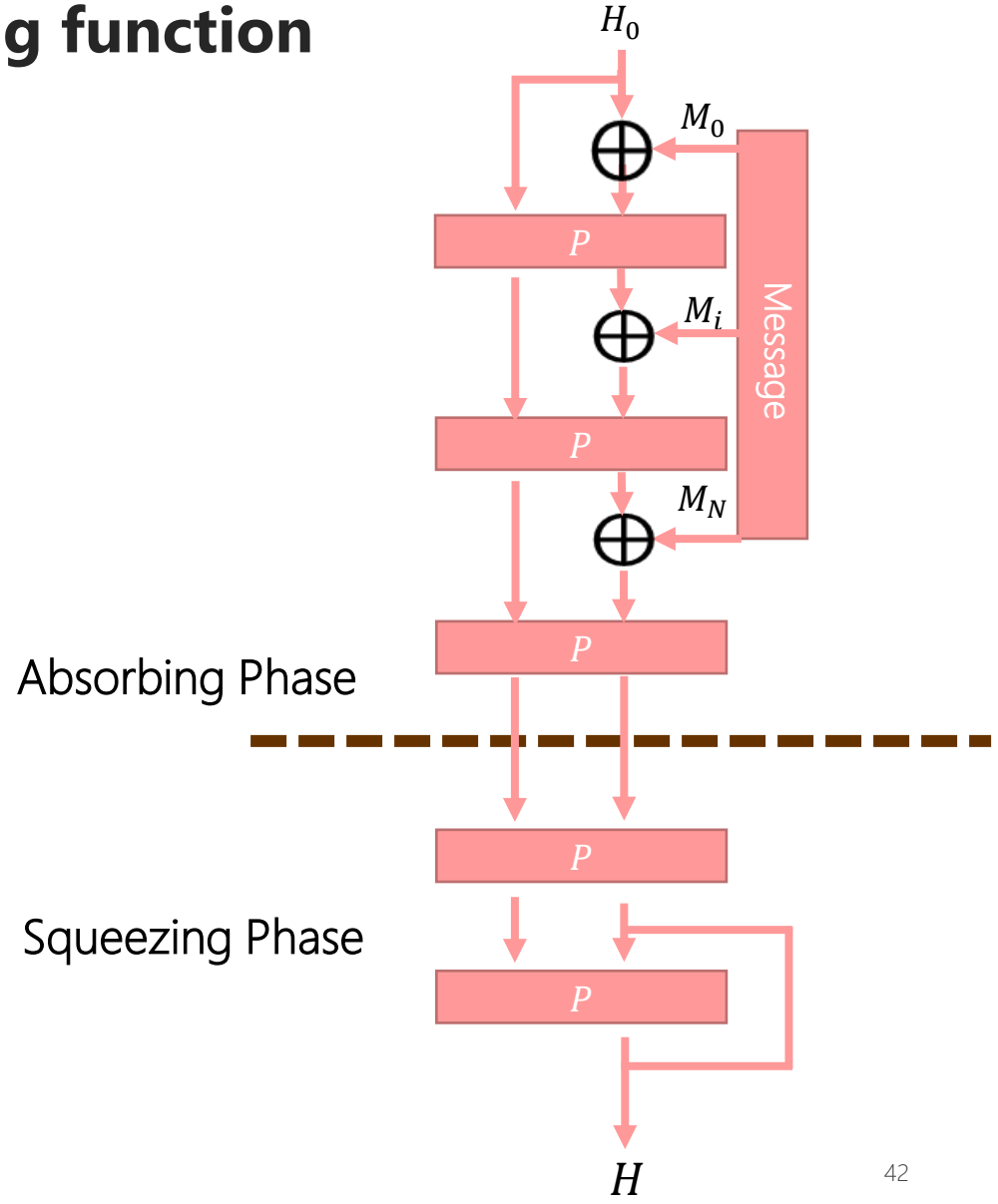
$$q_M \leq 2^{N/2}$$

And how do we build a hash function?

We don’t, cryptographers do…

A (very) brief look under the hood of a hashing function

- Modern hash functions (1980+) are built using *iterative hashing*
- Two constructs: *compression (Merkle-Damgard) x permutation hashing (sponge functions)*
- *SHA-3 (Keccak),* the latest Secure Hashing Algorithm standardised by NIST, is a *sponge function.*



A (very) brief look under the hood of a hashing function

- *How does it work?*

1. First message block M_0 XOR-ed against a pre-defined hash H_0
2. Permutation P applied to XOR-ed output
3. Output of previous permutation XOR-ed with M_i

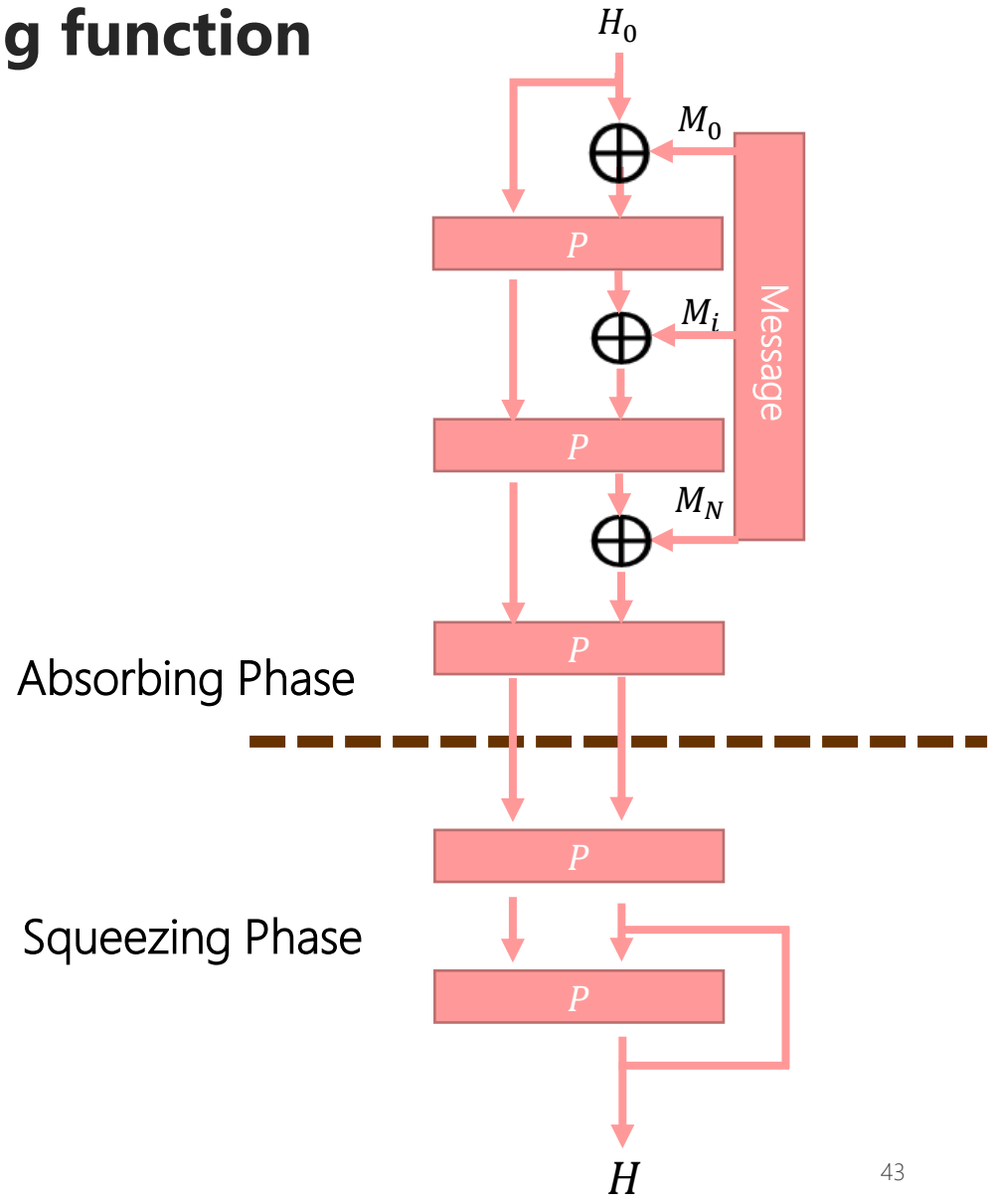
...

4. Output of previous permutation XOR-ed with the last message block M_N,

Closing the absorbing phase

5. Permutation P applied
6. A block of bits from the ***state*** are extracted to form the hash. If a longer hash is needed, the permutation P is applied again and another block of bits are extracted from the state.

This is the squeezing phase.



A (very) brief look under the hood of a hashing function

The security of the sponge depends on two parameters:

- Length of the internal state (w bits)
- Length of the message blocks (r bits)

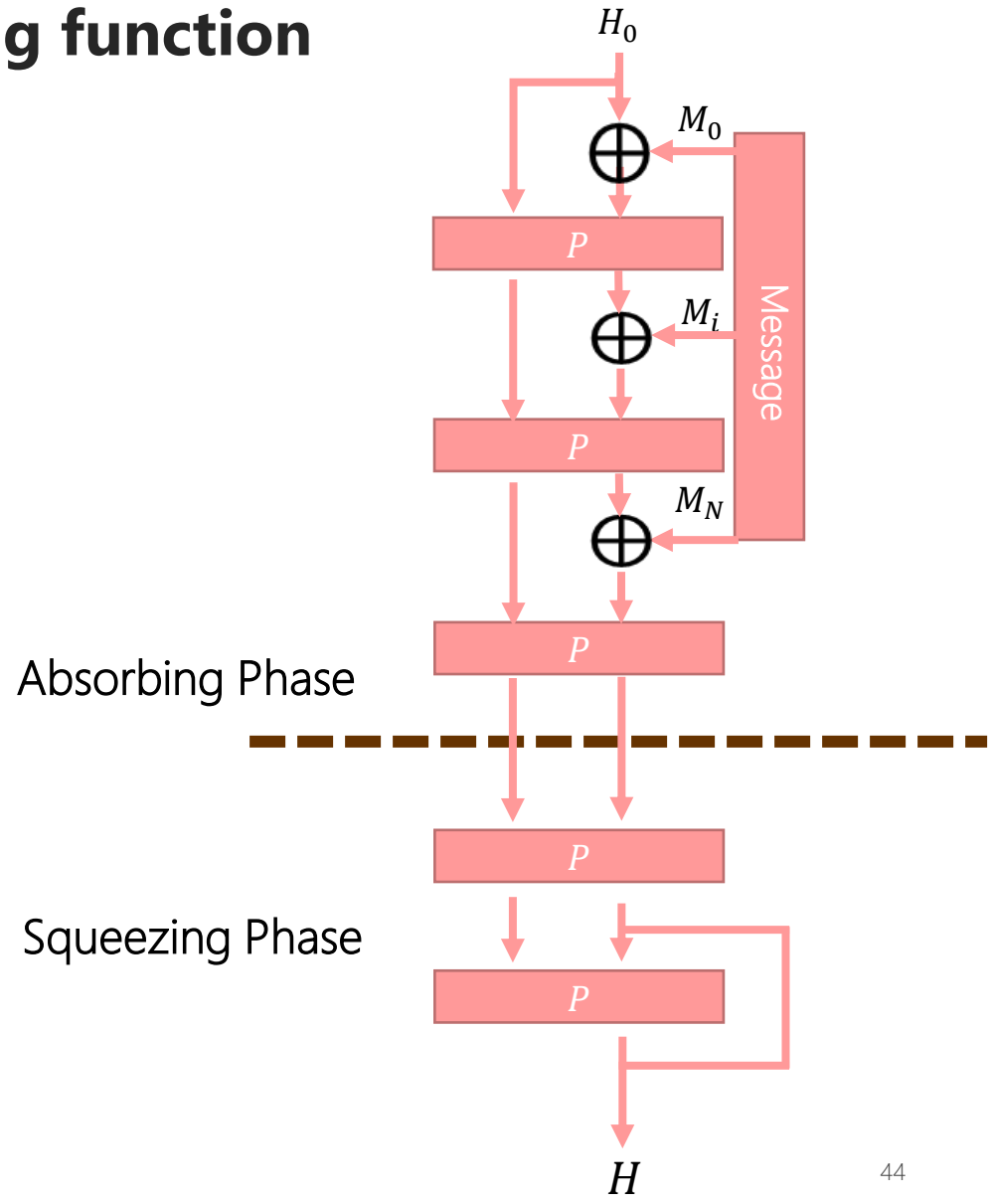
$$c = w - r$$

Is called the *sponge's capacity*. It defines the number of bits of the internal state that cannot be modified by message blocks.

The *security level* guaranteed by the sponge is $c/2$. For 256-bit security with $r = 64$, $c = 2 * 256 + r = 576$ bits.

As discussed before, to be sure, a *permutation must behave as random permutations*.

In this case, if the hash is of length n, then the complexity of an attack will be the smallest value between $2^{n/2}$ and $2^{c/2}$



Politecnico di Torino

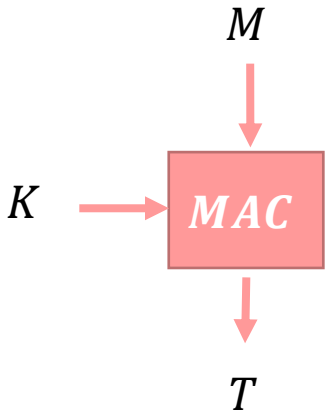
Anyone can hash… what if someone tampers with my message *and* hash?

Keyed Hashing

Message Authentication Code (MAC)

To ensure *integrity and authenticity*, we can create an authentication tag

$$T = MAC(K, M)$$



As long as the key K is known only by the sender and receiver, the message cannot be modified.

When a message M' is received, the receiver computes its MAC and checks against the received authentication tag T'. If $T' \neq T$, for example, because $M' \neq M$, then the message is not authentic.

MACs are widely used across many protocols, including Transport Layer Security (TLS), Signal, but not LTE or 5G (unless it's happening at higher layers, such as TLS and Sigal). Why?

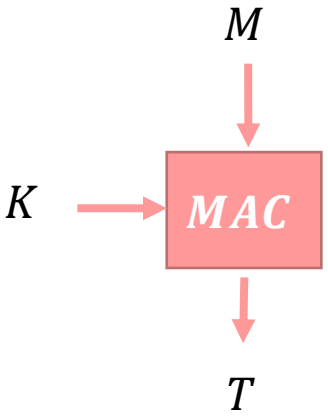
MACs represent additional overhead. If the data is encrypted with the right key (i.e., SIM key) and someone tampers with it, it will decrypt to noise.

Message Authentication Code (MAC)

- *Secure MACs are unforgeable*, that is, it is impossible to create a valid authentication tag if the attacker does not know the key,

- However, without a message number, it is possible to replay tags (i.e., tag used for a message re-used for another)

- To avoid this issue, messages have an identifier, and each tag includes the message number.

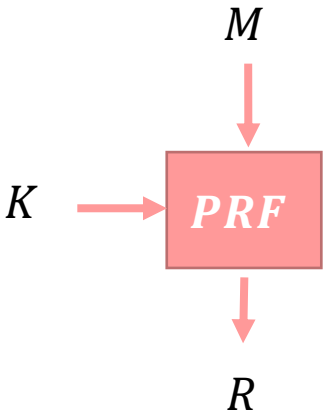


Pseudorandom Function (PRF)

- A secure pseudorandom function (PRF)

$$\mathrm{R} = PRF(K, M)$$

Uses a key to return an output that is indistinguishable from random to someone that does not know the key



- Several applications, among which:
  - Authentication challenges (i.e., send a random sequence and receive a "random" response)
  - Key derivation
  - Password-authenticated key exchange

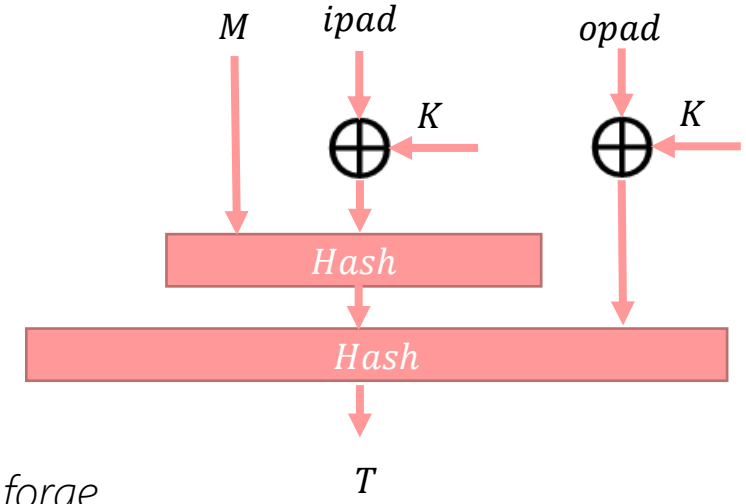
- Secure PRFs are also secure MACs; if you cannot distinguish the output from random, you also cannot forge tags.

- However, secure MACs are not secure PRFs; they may have structure that distinguishes them from random

Hash-based MAC (HMAC)

We can build MACs (and PRFs) from a ***secure hash function*** (RFC2104):

$$T = Hash\left((K \oplus opad) \mid Hash\left((K \oplus ipad) \mid M\right)\right)$$



Where:

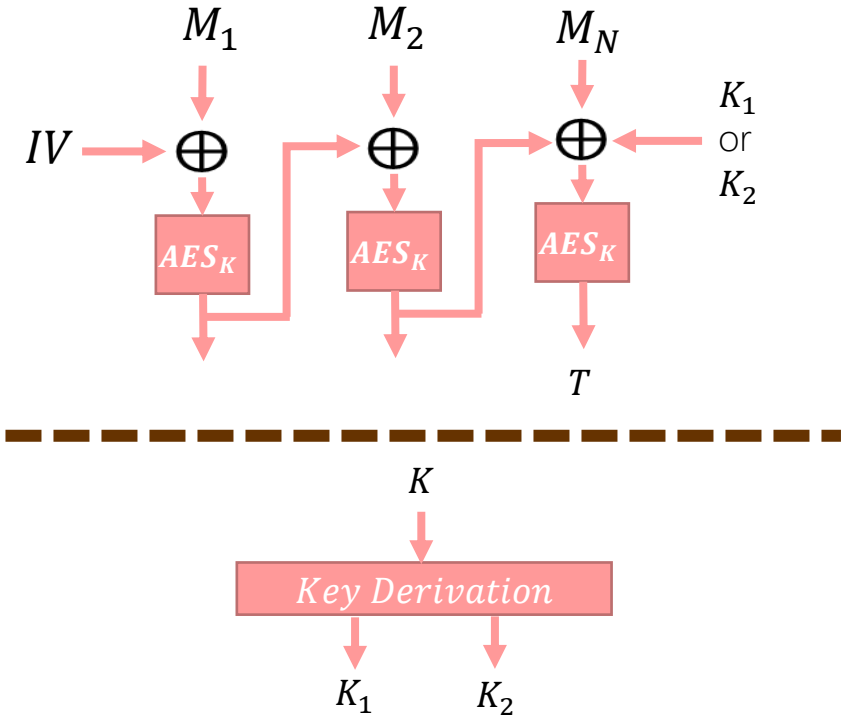
- $\boldsymbol{opad}$ *and* $\boldsymbol{ipad}$ *are strings as long as the block size*
- $\boldsymbol{K}$ *is the key*
- $\boldsymbol{M}$ *is the message*
- *Hash is a secure hash function (collision resistant), otherwise collisions could be used to forge MACs*

When a hash function is used as a HMAC, it is called HMAC-SHA3

For example, a combination of a HMAC and a cipher would be called AES-256-CBC + HMAC-SHA3

Cipher-based MAC (CMAC)

But we can also build MACs from *secure ciphers*, such as AES-CBC (RFC 4493):



1. We set $IV = 0$. Because it is a MAC, it does not need to be random. The same message and key will always produce the same authentication tag (*not a PRF!)*.

2. We derive two keys, K_1 and K_2, from a key K (outside of scope).

3. We apply AES-CBC to the message, using key K.

4. At the input of the last block, we XOR the last message block M_N with the output of the previous block and one of the two keys, K_1 or K_2, depending on the length of the last message block.

5. We discard all output, but the output of the last block, which becomes our authentication tag T

Wegman-Carter MAC

Repurposing hashes and ciphers as MACs and PRFs is resource-intensive. What can we do to improve it?

- We can speed-up the MAC generation using a ***universal hash*** $UH(K, M)$, rather than ***cryptographic hash functions***.
- *The only requirement is that* $P[\, UH(M_1) = \, UH(M_2)]$ *must be negligible for a random key* K
- *If this requirement is satisfied, we can generate a MAC and securely authenticate* ***a single message***.

To authenticate more than one message, we need an additional block: the Wegman-Carter construct.

Wegman-Carter MAC

More @ "New Hash Functions and Their Use in Authentication and Set Equality"

The Wegman-Carter MAC builds a MAC capable of authenticating multiple messages from a universal hash and a pseudorandom function, using two keys (K_1 and K_2):

$$MAC(K_1, K_2, N, M) = UH(K_1, M) \oplus PRF(K_2, N)$$

Where:

- N *is a nonce that should be unique for each key* K_2 *(i.e., counter)*
- UH *is a universal hash*
- PRF *is a strong pseudo-random function*
- *The output of* UH *and* PRF *have the same length and are sufficiently long to ensure security.*

Using independent keys protects the system from key interactions (i.e., UH and PRF both block ciphers using the same key)

"Like a hash encrypted with a one-time pad"

We saw how MACs can provide integrity and authentication.

We saw how ciphers, such as AES, can provide confidentiality.

Can we put them together?

Authenticated Encryption

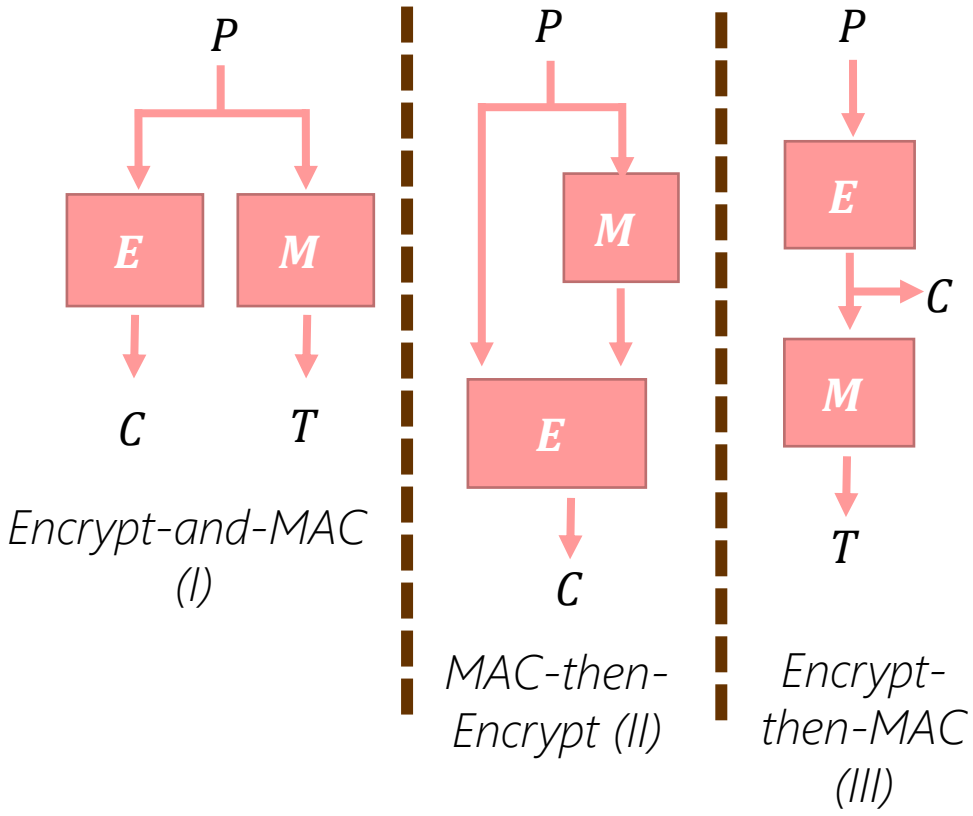
Authentication Encryption using MACs

MACs and Ciphers can be combined in three ways, each with unique trade-offs.

Recommended solution: Encrypt-then-MAC

- Because T is computed over P, (I) could leak information about P, facilitating its recovery unless the MAC was computed from a PRF and has no structure.
- (I) and (II) must decrypt C before authenticating T, which exposes the system to corrupted ciphertexts
- (II) encrypts T, preventing data about P from being leaked

- *(III) provides integrity of the ciphertext and allows the receiver to verify the authenticity of C before decrypting, avoiding decryption of corrupted ciphertext*
- *(III) prevents chosen ciphertext attacks, as the attacker must break the MAC before corrupting the ciphertext.*
- *(III) MAC is computed over ciphertext and cannot leak the plaintext*



Politecnico di Torino

Is there an alternative to combining ciphers and MACs?

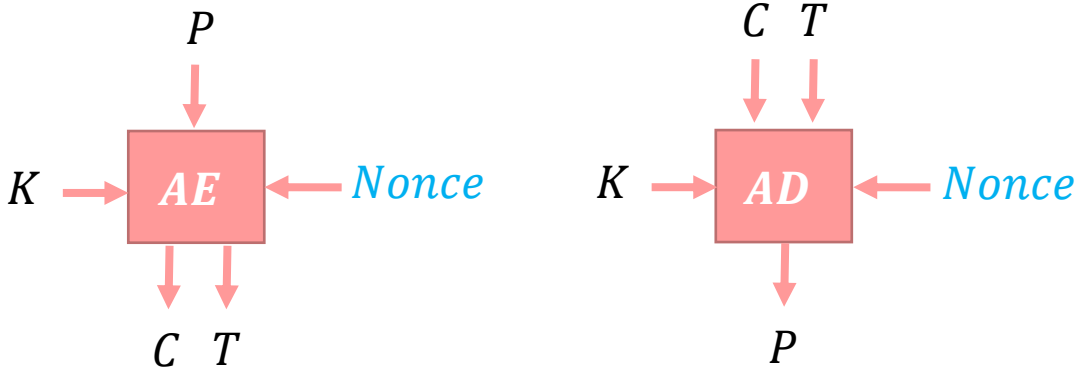
Authenticated Ciphers

Combine authentication and encryption to provide confidentiality, integrity, and authenticity.

Simpler, faster, and secure.

Two blocks:

- *Authenticated Encryption* AE
- *Authenticated Decryption* AD



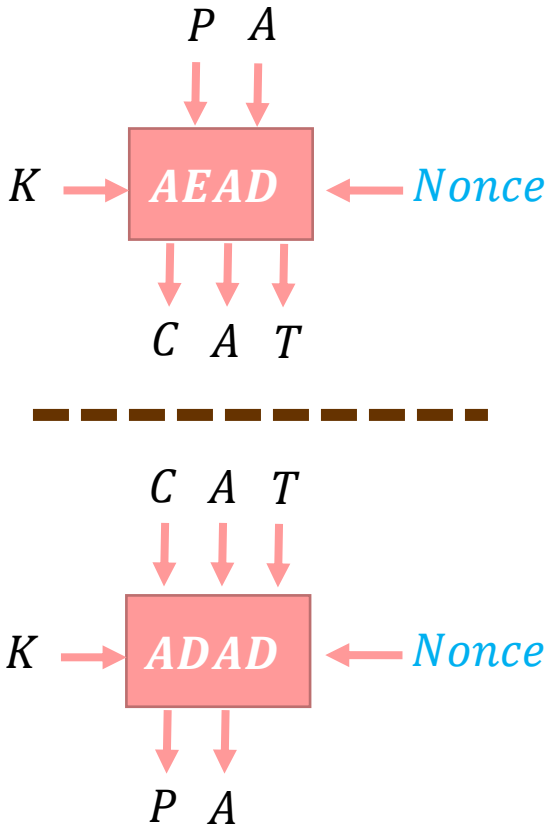
- Whenever (C, T) are wrong, AD *will return an* ***error rather than a plaintext forged or in error. If a plaintext is returned, it has been encrypted with the secret key,***

- ***Unforgeable tags,*** just as a secure MAC.

- Ciphertext is only decrypted if authenticity has been verified, otherwise it is discarded.

Authenticated Encryption with Associated Data (AEAD)

- Sometimes applications require data that is authenticated, but not encrypted (i.e., headers, routing, …)
- That is where Associated Data (AD) comes into play: if the cleartext data A, used for the computation of the authentication tag T, is corrupted, then the decryption will fail – even if C itself has not been corrupted.
- If A is empty, it is a secure authenticated encryption without any additional data
- If P is empty, it is a secure MAC.
- Just as with other block (AES-CBC) and stream ciphers (AES-CTR), we use a random *nonce* to ensure that the same plaintext encrypts to different plaintext. *The nonce must never be re-used for the same key and message.*
- Ideally, AEAD ciphers should be streamable (operate block-by-block) and parallelizable (operate on multiple blocks in parallel) for performance



AES-GCM

AES – Galois Counter Mode (GCM)

- *Encrypt-then-MAC Construction:*
  - *AES-CTR for data encryption with 128 or 256 bit key and a 96-bit nonce with counter starting from 1*
  - *Wegman-Carter MAC for authentication*

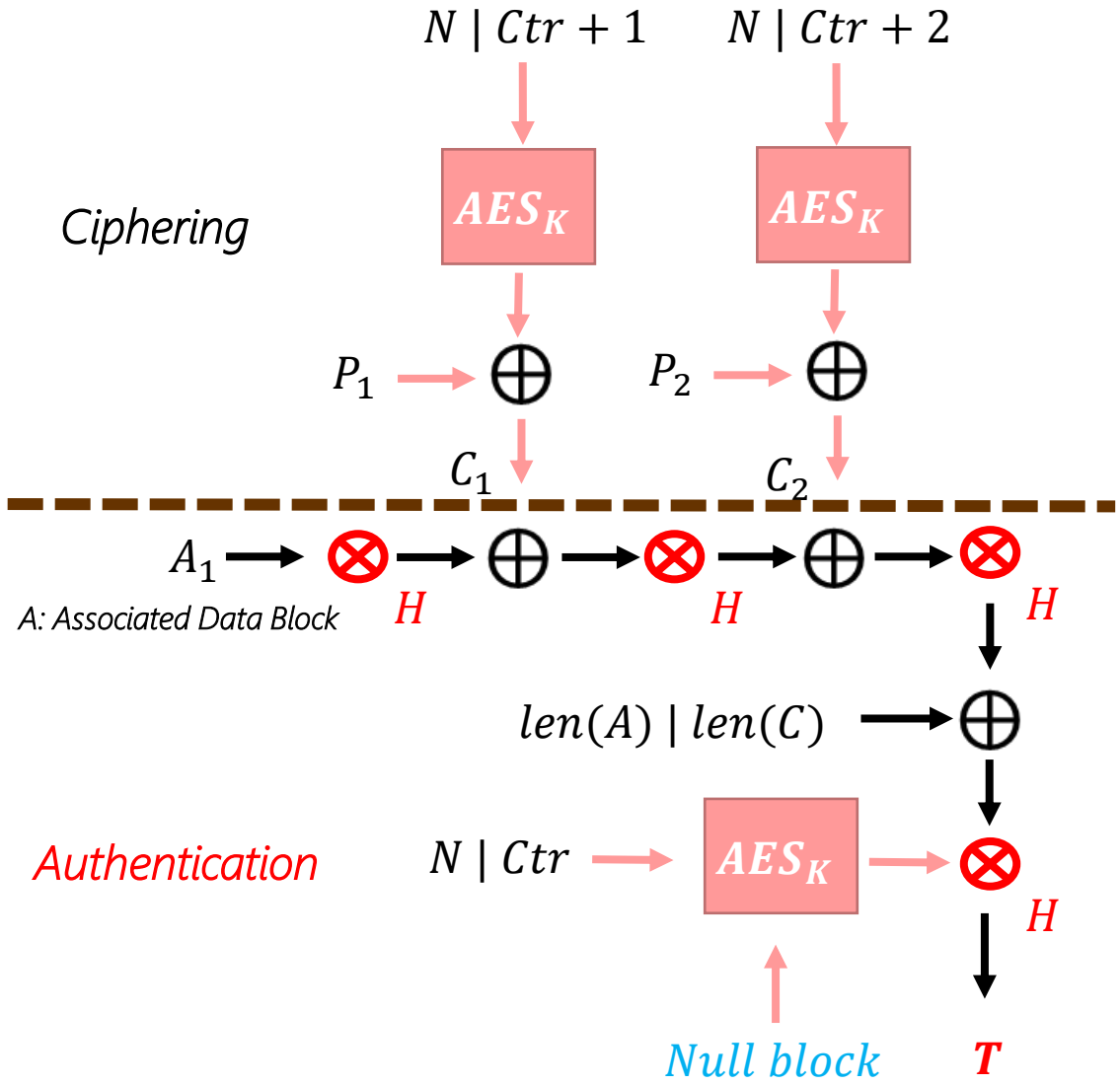
- We already saw how CTR works. What's new is the authentication tag

$$T = AES(K, N \mid 0) \oplus GHASH(H, A, C)$$

- It uses $GHASH$, a universal hash function.

- $H = AES(K, 0)$, *is an authentication key,* derived from K, *the encryption key*

- *As can be seen in the flow-diagram, a series of polynomial multiplications by* H *are carried out along the chain* $\otimes H$



AES – Galois Counter Mode (GCM)

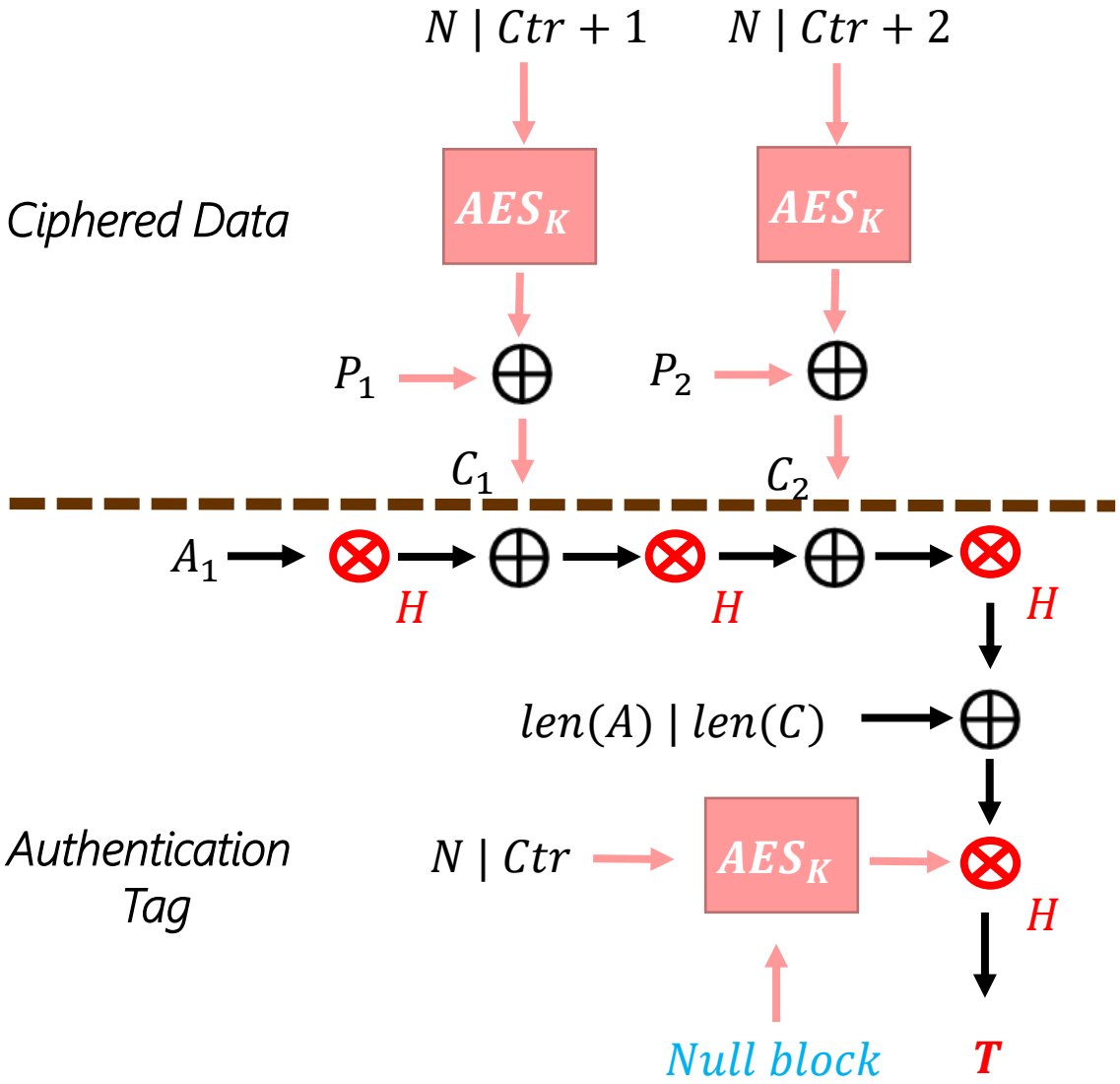
Advantages:

- Provides confidentiality, authenticity, and integrity, and a high level of security, *if the counter is never reused for the same key*.
- Efficient (parallelizable, as CTR)
- Streamable (ciphering of one stage can happen while the other is authenticating)

Disadvantages:

- Complicated to implement
- Authentication part blows up if a nonce is reused

$$T_1 \oplus T_2 = AES(K, N \mid 0) \oplus GHASH_1 \oplus AES(K, N \mid 0) \oplus GHASH_2 = GHASH_1 \oplus GHASH_2$$



So far, everything works with a shared key.

But where did the key come from?

Key Agreement Protocols

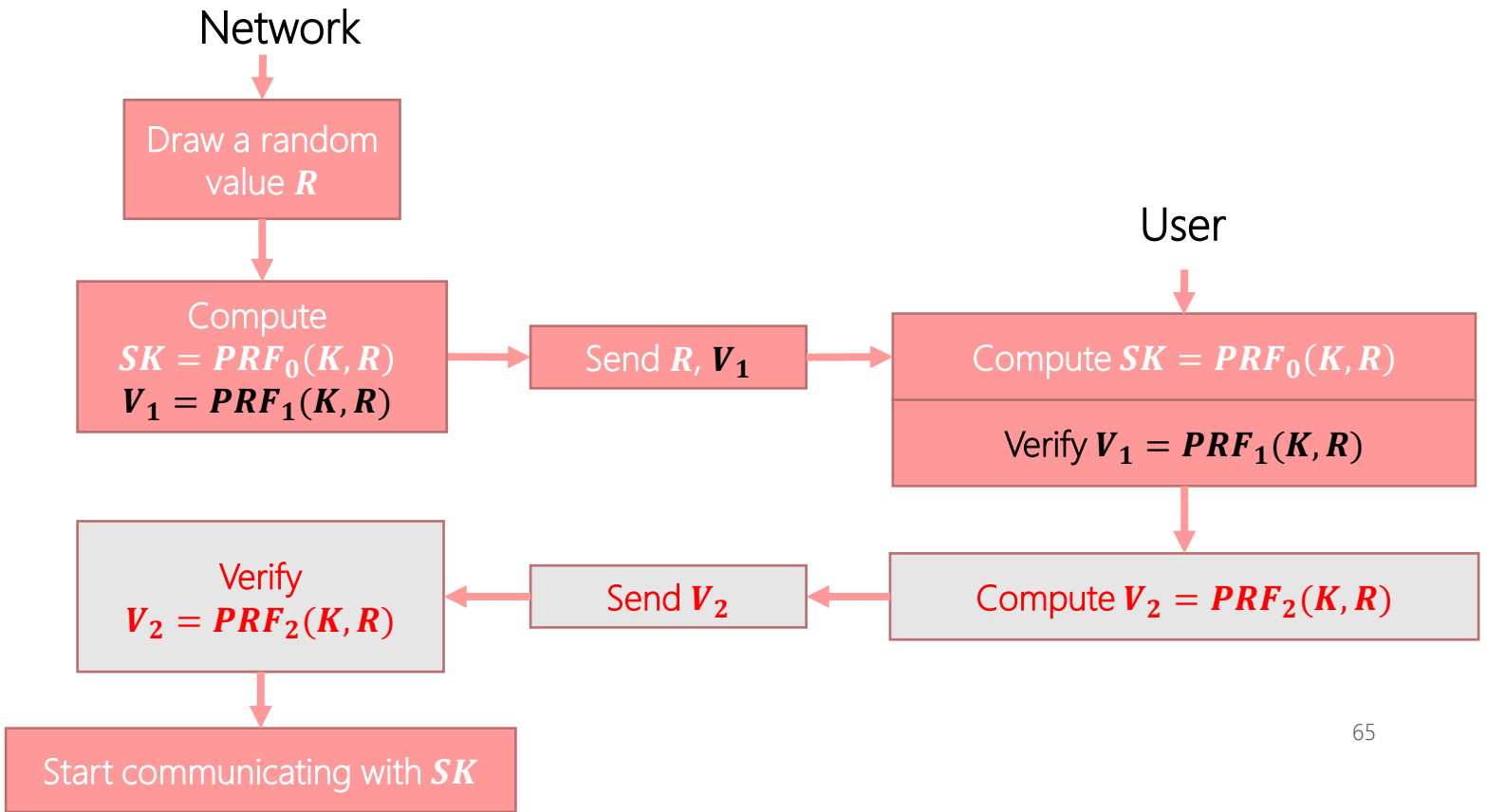
Authenticated Key Agreement (AKA) using PRFs

Example: Subscriber-Identity Modules (SIM) Cards

- *User has a key $\boldsymbol{K}$ permanently written to the SIM Card*
- *Network knows the key of the user, as it provided the SIM Card*

Three PRFs

If key is lost, all comms are compromised



Politecnico di Torino

Instead of storing a shared key, can we come-up with a new key to use?

Mathematical Preliminaries – Part II

Let $\mathbb{Z}_p^*$ be a group of non-zero integers between 1 and $p-1$ modulo p where p is a prime number.
It has the following properties:

- *Closure:* $\forall x, y \in \mathbb{Z}_p^*, \exists\, z = x * y \in \mathbb{Z}_p^*$
- *Associativity:* $(x * y) * z = x * (y * z)$
- *There is a neutral element* e: $e * x = x * e = x$
- *There is an inverse:* $\forall x \in \mathbb{Z}_p^*, \exists\, y \in \mathbb{Z}_p^*$ *such that* $x * y = y * x = e$
- *Commutativity:* $(x * y) = (y * x)$
- *All operations are executed modulo* p

A group is called **cyclic** *if there is at least one element* g*, called a* **generator,** *such that the set*

$$< g > = \{g^k \mid k \in \mathbb{Z}\} \, mod \, p$$

Spans all distinct group elements.

$\mathbb{Z}_p^*$ *is finite, containing on the order of* 2^p *elements, and p is a very large number (thousands of bits long)*

The Discrete Logarithm Problem (DLP)

Given $g, x \in \mathbb{Z}_p^*$, *find* y *such that* $x = g^y$

Hard!

Diffie-Hellman Key Exchange

- *Is a way to establish a shared secret over an unsecure channel.*
- *That shared secret may then be used to derive a key (i.e., an AES key) to transform the unsecure channel into a secure channel.*
- It relies on the computational hardness of the *Discrete Logarithm Problem (DLP)*

Two public parameters that must be ***carefully chosen:***

- g, the base number (generator)
- p, a prime defining $\mathbb{Z}_p^*$

(Anonymous) Diffie-Hellman Key Exchange

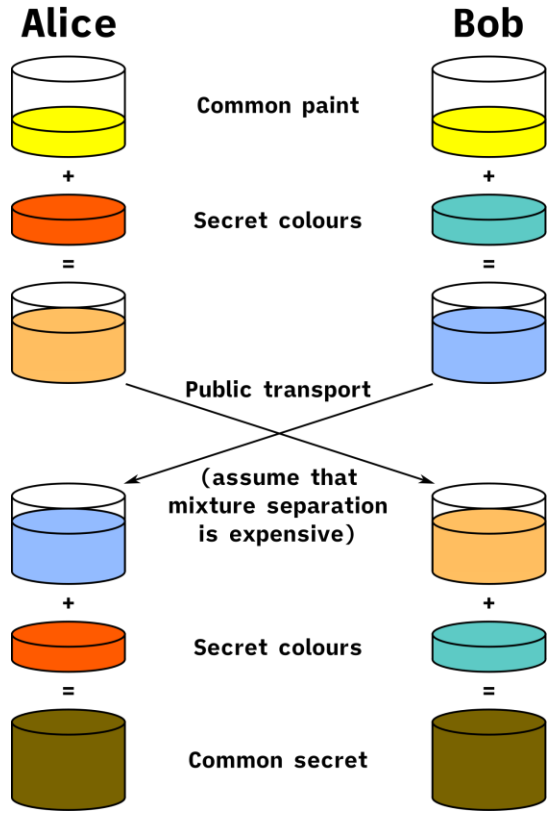
Alice wants to send a message to Bob.

- *Alice randomly picks a number* $\boldsymbol{a} \in \mathbb{Z}_p^*$
- *Bob randomly picks a number* $\boldsymbol{b} \in \mathbb{Z}_p^*$

- *Alice computes* $\boldsymbol{A = g^a \bmod p}$ *and sends it to Bob over an unsecure channel*
- *Bob computes* $\boldsymbol{B = g^b \bmod p}$ *and sends sit to Alice over an unsecure channel*

- *Alice computes* $\boldsymbol{B^a = \left(g^b\right)^a \bmod p = g^{ba} \bmod p = A^b}$
- *Bob computes* $\boldsymbol{A^b = (g^a)^b \bmod p = g^{ab} \bmod p = B^a}$

Shared secret!



Source: University of Essen

- *From this shared secret, we can use a key derivation function to generate symmetric keys!*

Authenticated Diffie-Hellman Key Exchange

- *But how do I know if Alice is really Alice?*
- *Here's where public-key cryptography (asymmetric) comes into play*
- *Each user has their own private and public key (i.e., RSA, ECC).*
- *The public key is known and verified by both parties in advance*
- *Now we can make use of **digital signatures!***

Alice wants to send a message to Bob.

- *Alice randomly picks a number* $\boldsymbol{a} \in \mathbb{Z}_p^*$ *and signs it with her private key, generating* $\boldsymbol{sig_a}$
- *Bob randomly picks a number* $\boldsymbol{b} \in \mathbb{Z}_p^*$ *and signs it with his private key, generating* $\boldsymbol{sig_b}$

- *Alice computes* $\boldsymbol{A = g^a \bmod p}$ *and sends* $\boldsymbol{(A, sig_a)}$ *it to Bob over an unsecure channel, together with*
- *Bob computes* $\boldsymbol{B = g^b \bmod p}$ *and sends* $\boldsymbol{(B, sig_b)}$ *sit to Alice over an unsecure channel*

- *Alice verifies* $\boldsymbol{sig_b}$ *with Bob's public key*
- *Bob verifies* $\boldsymbol{sig_a}$ *with Alice's public key*

- *If signatures match, both proceed. If not, abort.*

- *Alice computes* $\boldsymbol{B^a = (g^b)^a \bmod p = g^{ba} \bmod p = A^b}$
- *Bob computes* $\boldsymbol{A^b = (g^a)^b \bmod p = g^{ab} \bmod p = B^a}$

If you like crypto…

High demand for cryptographers, both in offense and defence.

If you want to learn more about secure real-world systems using crypto, check out:

- Signal (X3DH + Double Racket) Protocol, which secures Signal and Whatsapp. Did you know you can verify the person you're chatting to with a QR Code?

- Transport Layer Encryption (TLS): secures all WWW communications, non-encrypted traffic and invalid certificates are refused by modern browsers.

- Apple Secure Enclave: protects user and company data in devices sold to billions of users. Present in all modern iPhones

Secure Enclave Capabilities:
- Asymmetric Encryption
- Symmetric Encryption
- Hashing
- Key Generation
- A dedicate silicon where (per-device, manufacturer) keys can be permanently written to during fabrication.

If you want to learn *breaking* crypto…

Set 1: Basics

Set 2: Block crypto

Set 3: Block & stream crypto

Set 4: Stream crypto and randomness

Set 5: Diffie-Hellman and friends

Set 6: RSA and DSA

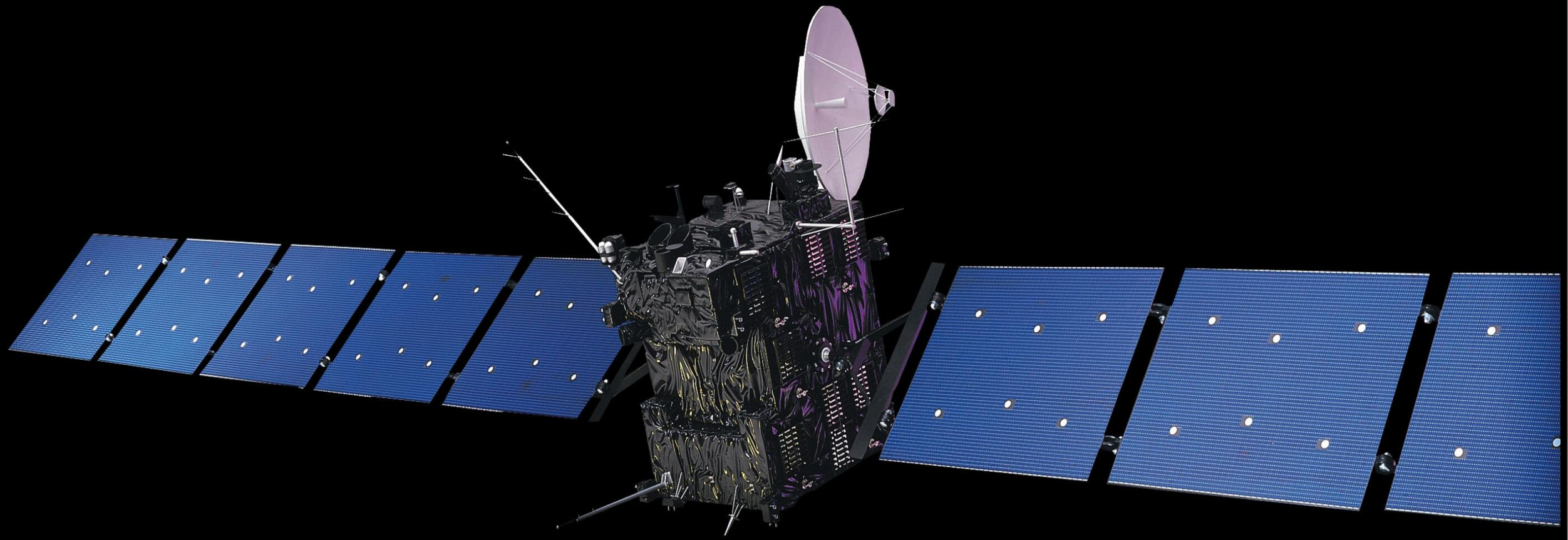
Set 7: Hashes

Set 8: Abstract Algebra

The Cryptopals Crypto Challenges

https://www.cryptopals.com/

Questions?



Contact:

gabriel.maiolini@polito.it

Source: ESA