Password

Complexity password:

* 10-12 characters.
* Majuscule/miniscule.
* Special characters.

Use a function for store passwords: calcul\_hash (plaintext\_pdw, hash, salt)

Possibility to add a parameter: pepper => store outside bdd.

|  |
| --- |
| Table password |
| uid |
| Salt\_2 |
| IV |
| Cipher Key AES |
| Cipher (passwd) |

Security Layer => password:

Hashed\_Password

Bdd

CypherSym\_3:

AES - GCM

4

Salt\_2:

Random\_Generator

Hash\_2:

ARGON2D

3

Step of encrypting the plaintext password before storing it in the bdd.

NETWORK

2

Hash\_1:

SHA-512

Salt\_1:

smartnotary

1

Plaintext pwd

---------------------------------------------------------------------------------------------------------------------------

EXTRAIRE :

get on html page

Password

+ common salt : ‘SmartNotary’

SHA2-256

256-bits

Client Side :

---------------------------------------------------------------------------------------------------------------------------

Insecure Network :

---------------------------------------------------------------------------------------------------------------------------

256-bits

Server Side :

+ random salt

Argon-2D

256-bits

256-bits

96-bits

IV

KAES

AES\_GCM\_256

Structure

KAES

IV

Cipher of hash

File :

=> KAES

---------------------------------------------------------------------------------------------------------------------------

DB User

* KRSA\_PR

[KAES].KRSA\_PUB

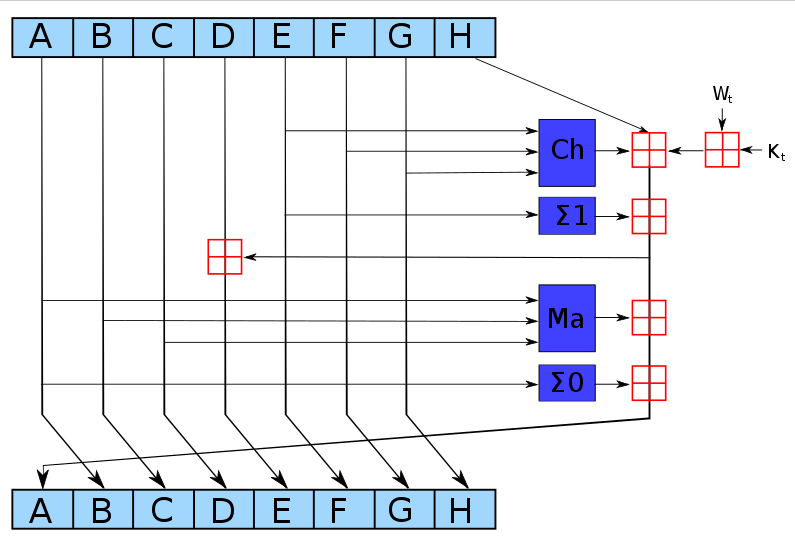
File : cipher code

* KAES
* KRSA\_PUB

SHA-512 :

SHA-2 :

* family hash functions .
* created NSA.
* Inspired function MD4 by Riverst.
* Operate on word size 64bits = SHA-512).
* message size max : 2128 bits.
* blocks size : 1024 bits.
* words size : 64 bits.
* hash size : 512 bits.
* rounds (function of compression): 80.
* Addition => modulo 264.



Algorithm => 2 phases :

Pre-processing:

1. padding : complete message => message size is multiple of 1024 bits.
2. Cut block: message into blocks of 1024 bits.
3. Initialize : values of hash (H(0)).

Hash Compute:

* Table of 64 words (Wt, t= {0,63})
* 8 variables (a, b, c, d, e, f, g, h) sur 64 bits
* 8 variables including hash values (H0(i)) sur 64 bits
* 2 variables T1 and T2, words of 64bits.

=> treat successively N blocks of M :

Security against :

* Collision attacks = 256 bits.
* Length extension attacks = 0 bits.

ARGON2 : function of key derivation => 2 versions:

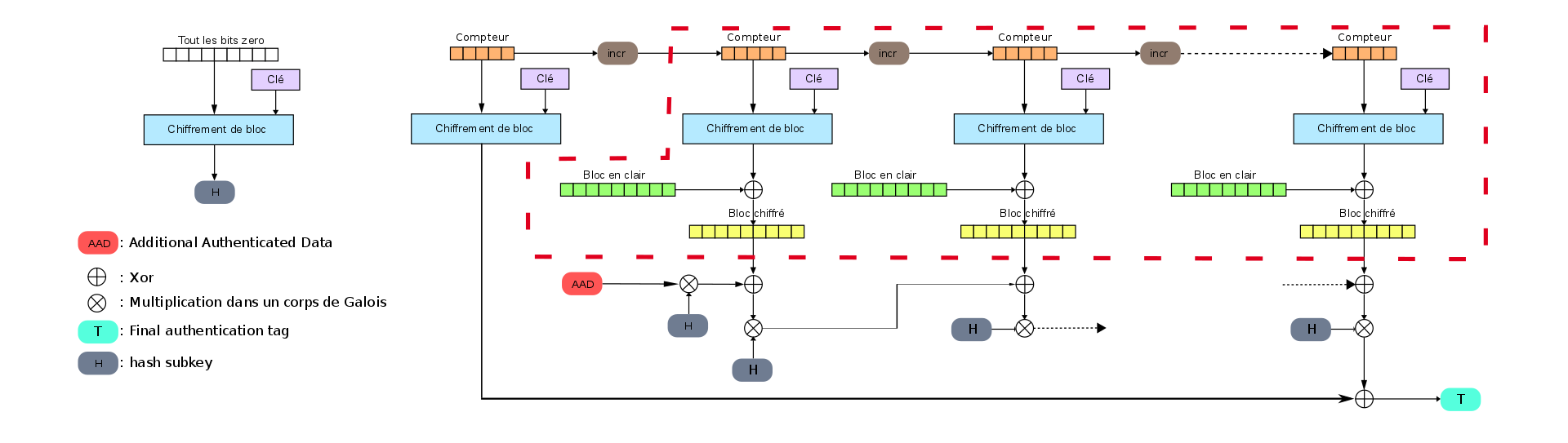
* Argon2D : designed to resist against attacks by GPU.
* Argon2i : optimize against attacks by auxiliary channel.

AES-GCM :

Block cipher operation mode => symmetric cryptography.

designed for authenticity, integrity, and confidentiality.

Based on counter (mode CTR) in which the multiplication between the counter and the 128-bit key is performed in the Galois field.

There are four inputs for authenticated encryption: the secret key, initialization vector (IV) (sometimes called a nonce†), the plaintext itself, and optional additional authentication data (AAD). The nonce and AAD are passed in the clear. There are two outputs: the ciphertext, which is the same length as the plaintext, and an authentication tag (the "tag"). The tag is sometimes called the message authentication code (MAC) or integrity check value (ICV).

