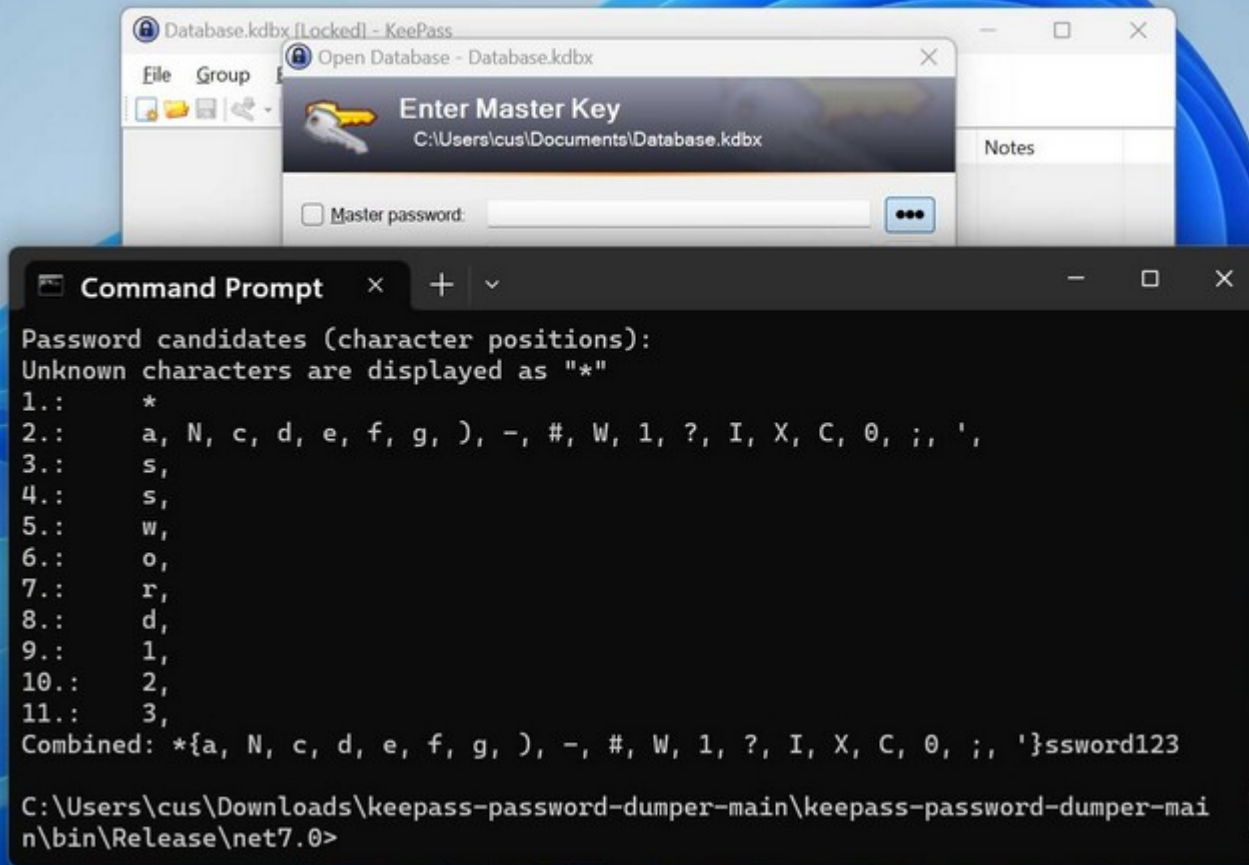


Hashing e Criptografia

Informações e Práticas





Fonte: <https://www.bleepingcomputer.com/news/security/keepass-exploit-helps-retrieve-clear-text-master-password-fix-coming-soon/> (20230518)

CVE-ID

CVE-2023-32784

[Learn more at National Vulnerability Database \(NVD\)](#)

• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

In KeePass 2.x before 2.54, it is possible to recover the cleartext master password from a memory dump, even when a workspace is locked or no longer running. The memory dump can be a KeePass process dump, swap file (pagefile.sys), hibernation file (hiberfil.sys), or RAM dump of the entire system. The first character cannot be recovered. In 2.54, there is different API usage and/or random string insertion for mitigation.

References



Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness



New to C
[Start here](#)

Home > CWE List > CWE- Individual Dictionary Definition (4.12)

ID Lookup:

[Home](#)[About](#)[CWE List](#)[Mapping](#)[Top-N Lists](#)[Community](#)[News](#)[Search](#)

CWE-256: Plaintext Storage of a Password

Weakness ID: 256

Abstraction: Base

Structure: Simple

View customized information:

[Conceptual](#)[Operational](#)[Mapping
Friendly](#)[Complete](#)[Custom](#)

▼ Description

Storing a password in plaintext may result in a system compromise.

▼ Extended Description

Password management issues occur when a password is stored in plaintext in an application's properties, configuration file, or memory. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource. In some contexts, even storage of a plaintext password in memory is considered a security risk if the password is not cleared immediately after it is used.

Aleatoriedade

O que é Aleatoriedade (*Randomness*)

É o processo de obter números (ou bits) que devem possuir duas importante propriedades: **distribuição uniforme** (todos os valores devem ter mesma probabilidade de ocorrência) e devem ser **gerados de forma não predizível** (*unpredictable*). Esta última diz que um valor gerado não pode ter qualquer relação com números obtidos anteriormente.

Como obter uma sequência de valores aleatórios (pseudo-aleatórios)

Usando Java

```
jshell> Random random = new Random();  
Jshell> int randomWithNextInt =  
random.nextInt();  
Jshell> randomWithNextInt = random.nextInt();
```

Em Python

```
>>> import random  
>>> random.randint(0,1000)
```


Como obter uma sequência de valores aleatórios (pseudo-aleatórios)

Usando PHP

```
<?php  
$bad_random_number = rand(0, 10);  
$secret_key = random_bytes(16);  
?>
```

Como obter uma sequência de valores aleatórios (pseudo-aleatórios)

Usando Java

```
jshell> Random random = new Random();  
Jshell> int randomWithNextInt = random.nextInt();  
Jshell> randomWithNextInt = random.nextInt();
```

Em Python

```
>>> import random  
>>> random.randint(0,1000)
```

Em PHP

```
<?php  
$bad_random_number = rand(0,  
$secret_key = random_bytes(16);  
?>
```

Não seguros para uso em
aplicações seguras
(criptográficas)

Como obter uma sequência de valores aleatórios (geradores **números criptograficamente fortes**)

Warning: The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the **secrets** module.

See also: M. Matsumoto and T. Nishimura, “Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator”, ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3–30 1998.

Complementary-Multiply-with-Carry recipe for a compatible alternative random number generator with a long period and comparatively simple update operations.

Como obter uma sequência de valores aleatórios (geradores **números criptograficamente fortes**)

Java: classe **SecureRandom**

<https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html>

```
jshell> import java.security.SecureRandom
```

```
jshell> SecureRandom secureRandom = new SecureRandom();
```

```
secureRandom ==> NativePRNG
```

```
jshell> byte bytes[] = new byte[20];
```

```
bytes ==> byte[20] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
```

```
jshell> random.nextBytes(bytes);
```

```
jshell> bytes
```

```
bytes ==> byte[20] { -21, -81, 89, -125, -81, 93, -78, -25, -119, -122, 64, 41, 62, 24, -46, 95, 2, 12, 124, -87 }
```

Funções de *Hash*

Códigos e Cifragem

12c8dfa285f14e1af8c5254e7092d0d3
5f65604c3d3ee629f6b32f6c2192c6d2
4032af8d61035123906e58e067140cc5
6652775dc3647f13c2d944922a5120c8
b01084209f8271f6e1f668713ffa2bd5
b0d7a5e71a397561c878a1cd28c0dfdd
5645F13F500882B21AC3884B83324540
73E03866F7FFB239F1298F76A705BE7F
a8f72257e2de676ff643cc2cc57f3ddb
012345678901234567890123456789a
MDEyMzQ1Njc4OWFiY2RlZg==

Códigos e Cifragem

Dado de Entrada: 0123456789abcdef (16 bytes)

MD2:	12c8dfa285f14e1af8c5254e7092d0d3
MD4:	5f65604c3d3ee629f6b32f6c2192c6d2
MD5:	4032af8d61035123906e58e067140cc5
RIPEMD-128:	6652775dc3647f13c2d944922a5120c8
BLAKE2b-128:	b01084209f8271f6e1f668713ffa2bd5
BLAKE2s-128:	b0d7a5e71a397561c878a1cd28c0dfdd
LM Hash:	5645F13F500882B21AC3884B83324540
NT Hash:	73E03866F7FFB239F1298F76A705BE7F

Códigos e Cifragem

Dado de Entrada: 0123456789abcdef HEX (64 bits)

AES-128: a8f72257e2de676ff643cc2cc57f3ddb

ECB Key: 012345678901234567890123456789a

Base64: MDEyMzQ1Njc4OWFiY2RlZg==

O que é Hashing

Hashing: função determinística unidirecional que toma uma entrada de tamanho variável e produz uma saída de tamanho fixo.

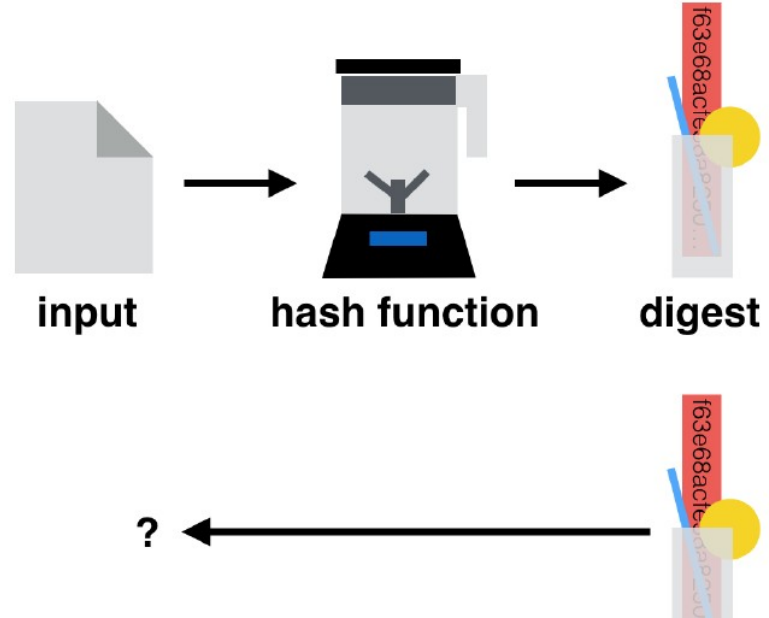
Determinística: sempre será produzido o mesmo resultado se a entrada for a mesma.

Unidirecional: é inviável computacionalmente obter o texto de entrada dado o texto de saída.

O que é Hashing

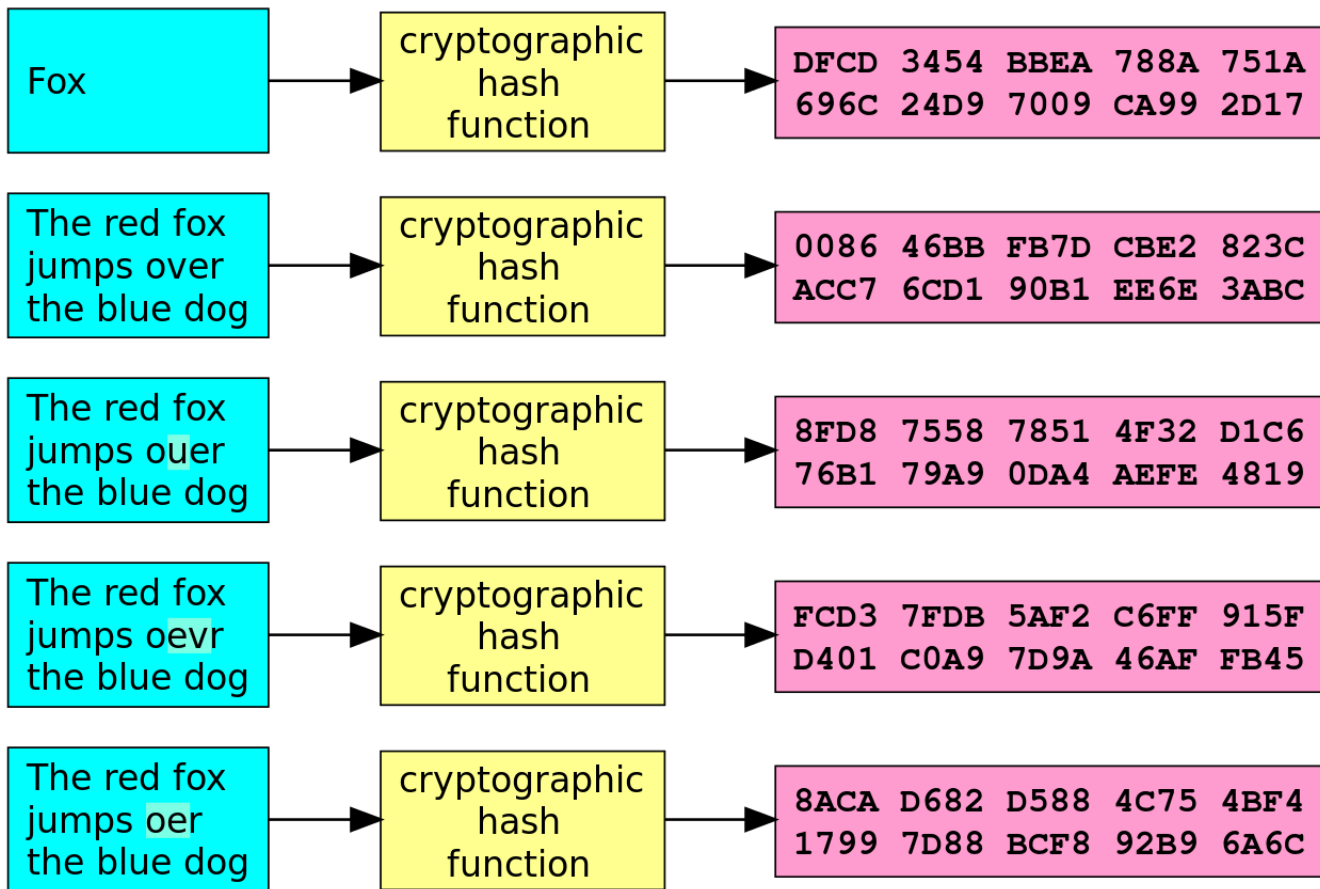
Entrada: texto ou binário (arquivos executáveis, imagens, etc).

Saída: um stream de bits de tamanho determinado, normalmente referido como **hash**, **digest** ou **fingerprint**.



Input

Digest



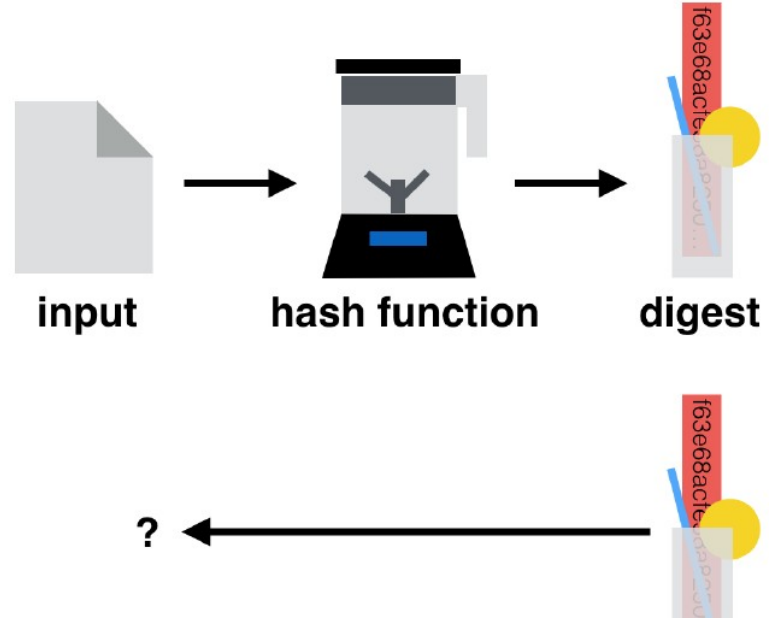

```
jshell> import java.security.MessageDigest
jshell> MessageDigest md = MessageDigest.getInstance("SHA-256");
md ==> SHA-256 Message Digest from SUN, <initialized>
jshell> byte[] encHash = md.digest("a".getBytes());
encHash ==> byte[32] { -54, -105, -127, 18, -54, 27, -67, -54 ... -
123, -81, -18, 72, -69 }
jshell> encHash.toString()
$5 ==> "[B@3d012ddd"
```

```
>>> import hashlib
>>> md = hashlib.sha256()
>>> md.update(b"a")
>>> md.digest()
b'\xca\x97\x81\x12\xca\x1b\xbd\xca\xfa\xc2\x1b3\x9a#\xdcM\xa7\x86\xef\xf8\x14|Nr\xb9\x80w\x85\xaf\xeeH\xbb'
>>> md.hexdigest()
'ca978112ca1bbdcafacc231b39a23dc4da786eff8147c4e72b9807785afee48bb'
```

O que é Hashing

Entrada: texto ou binário (arquivos executáveis, imagens, etc).

Saída: um stream de bits de tamanho determinado, normalmente referido como **hash**, **digest** ou **fingerprint**.



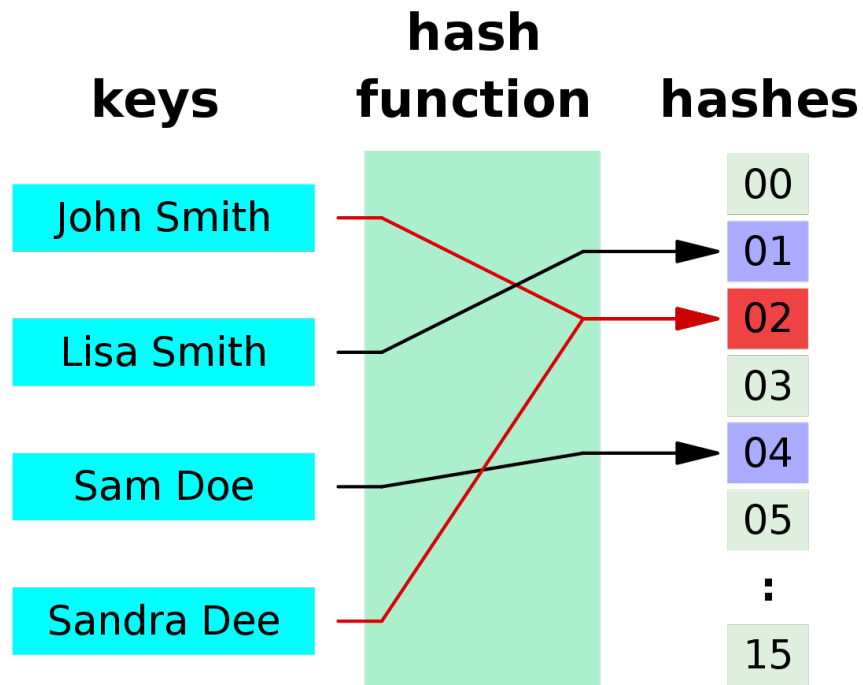
Hashing: propriedades

- Aplicar função hashing sobre uma mesma entrada, irá produzir sempre o mesmo resultado.
- Uma pequena mudança na entrada deve alterar completamente a saída.
- Saída sempre de mesmo tamanho (dependente do algoritmo de hashing), não importando o tamanho da entrada. Permite **ofuscar o tamanho e tipo de informação original**.

Hashing: problema

Colisão: ela é mais provável quanto menor o tamanho em bits do bloco de saída.

Exemplo com tamanho de somente 16 valores possíveis (4 bits).



Funções de *Hash*

Aplicações e Usos

- 1. Identificação exclusiva; e**
- 2. Checagem de integridade**

Hashing geram um fingerprint exclusivo de qualquer informação, logo, é possível fazer checagem de integridade. Como?

Refazendo o hashing e comparando os resultados.

1. Identificação exclusiva; e

2. Checagem de integridade

Hashing geram um fingerprint exclusivo de qualquer informação, logo, é possível fazer checagem de integridade.


Verify your download

Verify your download for security and integrity using the proper checksum file. If there is a good signature from one of the Fedora keys, and the SHA256 checksum matches, then the download is valid.

1. Download the [checksum file](#) into the same directory as the image you downloaded.

2. Import Fedora's GPG key(s)

```
curl -O https://fedoraproject.org/fedora.gpg
```

 You can verify the details of the GPG key(s) [here](#).

3. Verify the checksum file is valid

```
gpgv --keyring ./fedora.gpg Fedora-Workstation-38-1.6-x86_64-CHECKSUM
```

4. Verify the checksum matches

```
sha256sum -c Fedora-Workstation-38-1.6-x86_64-CHECKSUM
```

If the output states that the file is valid, then it's ready to use!

Conteúdo arquivo: **Fedora-Workstation-38-1.6-x86_64-CHECKSUM**

-----BEGIN PGP **SIGNED** MESSAGE-----

Hash: SHA256

Fedora-Workstation-Live-x86_64-38-1.6.iso: 2099451904 bytes

SHA256 (Fedora-Workstation-Live-x86_64-38-1.6.iso) =

7a444a2e19012023bf0b015ae30135bafc5fd20f4f333310d42b118745093992

-----BEGIN PGP **SIGNATURE**-----

iQIzBAEBCAAdFiEEaLg7q7o9VGe2FxIhgJqNf0sQtGQFamQ5B+cACgkQgJqNf0sQ
tGSm0BAAnQVdrztVVw+SPkJnY2bM8icmCZvEQxsN7zglhc6IubA710Htb003vrsr
1p7V/DoS00segXd9KIH618Li/602zx6tELzhZVaH0KvT/xlM7jh/ZqcUVhop65Jy
sXiCiIdKabfyxkHoq0GBzsGGmU3n3GULQmsNfvUXoghawUNK0E1+VgV4RLGEuUNrT
5IViT4Ct60jq+Sk9Gj9b7gheprZQZ0ZpZJ19ms8pK2CEPHSEnKWOMGFp7Ho0iEzG
9u+DLy20De1GV8cdxQ+vCGcc8KL3wFHKZvZku5TrlHODua/+NvihdCzLtNuRM4u4
ckJo9WitN4FpySlv0WKR2jC3WTi1Zsw/lvR2uXv4DSsa9hdu2DpU0YCvCCIMtoXw
j5lE4/2fNLlahsgD8NACtI3ulomM/VkIHtGR7dT43jTCsrnkPSbTeGcwUSgGTjM
vZ24gJHHb3y84jF6o3VbfNfHRAVZX3H02MQ0lRlresle0gVXwWwYroxGtdYSSqK
p/bC0oa0lmFcrG7rLqR0SG1IqBhdW5egT/U/Et+7xPztbxR3SmRd8CShLYD2VTFp
UAtn9w/qGAo/BSuS+5XPpAiX9Kx0hhK01bB+Hc26tzAeEYp406382DVdTDmuvv/v
IbQa9yao7yboowsKbe3Cv6axMLVqNcZsulawmQ2r8YVEzBPurgQ=
=7dH0

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

1. Identificação exclusiva; e
2. Checagem de integridade

Arquivos distribuídos via CDN (Content Delivery Networks) podem conter informação de integridade (**subresource integrity**) para evitar incorporação de bibliotecas com scripts maliciosos:

```
<script src="https://code.jquery.com/jquery-  
2.1.4.min.js"  
  integrity="sha256-
```

```
8WqyJLuWKRBVhxXIL1jBDD7SDxU936oZkCnxQbWwJVw="></script>
```

Atividade de verificação de integridade

Acessar a página **<http://20.226.9.204:8888/>** e conferir alguns dos hashes. No Windows:

<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash?view=powershell-7.3>

```
Get-FileHash img-a-low.png
```

```
-Algorithm SHA256 |
```

```
Format-List
```

1. Identificação exclusiva; e

2. Checagem de integridade

Trust On First Use (TOFU): mecanismo de **autenticação** no qual a confiança do primeiro acesso de uma entidade. Suas premissas são:

- Um atacante nem sempre estará presente, durante o momento da conexão; e
- A janela de vulnerabilidade deste processo é pequena.

Protocolo SSH: exemplo do primeiro acesso a um servidor SSH

```
ssh user@localhost
```

```
The authenticity of host '[localhost]:22 ([127.0.0.1]:22)' can't be established.
```

```
ED25519 key fingerprint is SHA256:wRkXU8YWLdXBD018iIuap+wAwBhfU/+2U8cl0FUo2tM.
```

```
This key is not known by any other names
```

```
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

3. Armazenamento e Verificação de Senhas

É um **risco** armazenar senhas em qualquer sistema.

Usar esquemas criptográficos traz problemas de gerenciamento de chaves.

Uma forma mais direta, com baixo overhead e com bom nível de segurança é pelo uso de **hashing**.

3. Armazenamento e Verificação de Senhas

```
<?php

$username = $_REQUEST['username'];
$password = $_REQUEST['password'];

$sql = "INSERT INTO users (username, password)
VALUES ('" . $username . "', '" . $password . "')";

$conn = new mysqli("localhost", "root", "", "myDB");
$conn->query($sql)
$conn->close();
?>
```

3. Armazenamento e Verificação de Senhas

```
Properties prop = new Properties();  
prop.load(new FileInputStream("config.properties"));  
String password = prop.getProperty("password");  
DriverManager.getConnection(url,usr, password);  
...
```



KeePass exploit helps retrieve cleartext master password, fix coming soon

By **Bill Toulas**

May 18, 2023

04:26 PM

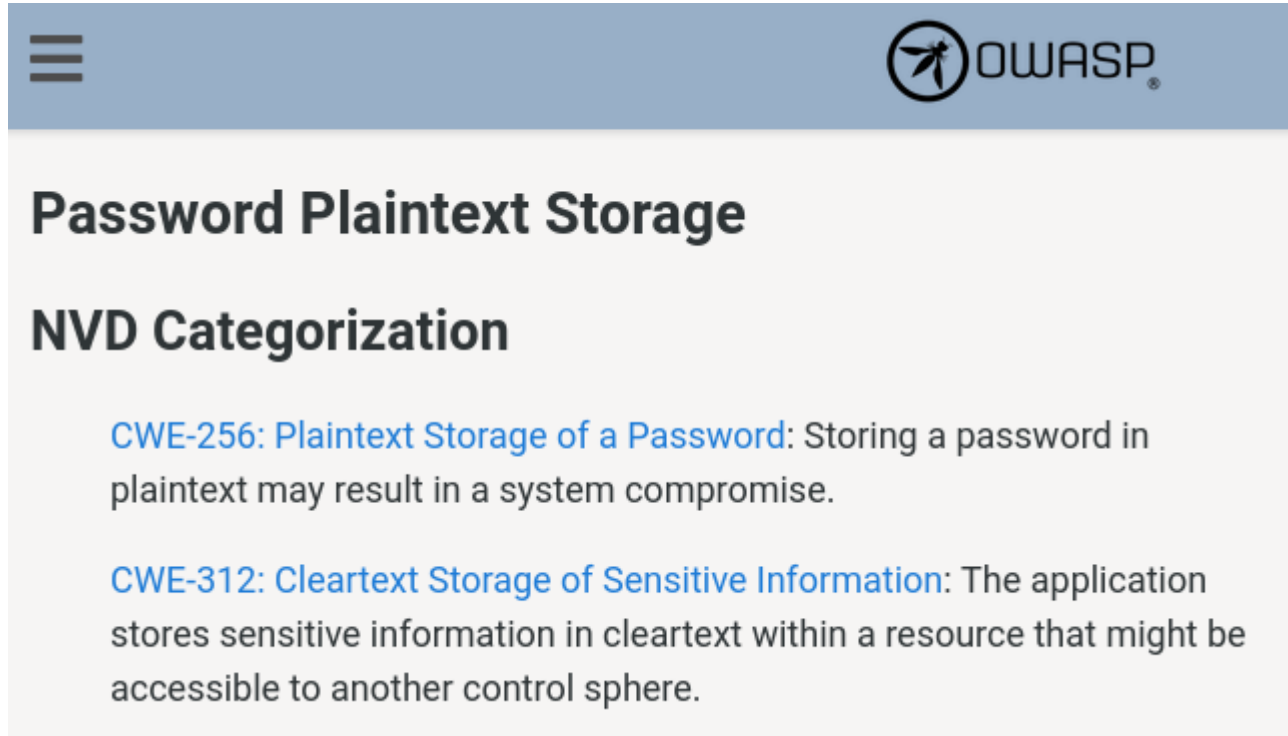
8

```
1.: a, N, c, d, e, f, g, ), -, #, W, 1, ?, I, X, C, 0, ;, ',
2.: s,
3.: s,
4.: w,
5.: o,
6.: r,
7.: d,
8.: 1,
9.: 2,
10.: 3,
11.: Combined: *{a, N, c, d, e, f, g, ), -, #, W, 1, ?, I, X, C, 0, ;, '}ssword123

C:\Users\cus\Downloads\keepass-password-dumper-main\keepass-password-dumper-main\bin\Release\net7.0>
```

Fonte: <https://www.bleepingcomputer.com/news/security/keepass-exploit-helps-retrieve-cleartext-master-password-fix-coming-soon/> (20230518)

3. Armazenamento e Verificação de Senhas



The image is a screenshot of a web page from OWASP. At the top, there is a blue header bar containing a hamburger menu icon on the left and the OWASP logo on the right. Below the header, the main content area has a light gray background. The title "Password Plaintext Storage" is prominently displayed in a large, bold, black font. Underneath the title, the section "NVD Categorization" is also in bold black font. This section lists two categories: "CWE-256: Plaintext Storage of a Password" and "CWE-312: Cleartext Storage of Sensitive Information". Each category is followed by a brief description of the vulnerability.

Password Plaintext Storage

NVD Categorization

CWE-256: Plaintext Storage of a Password: Storing a password in plaintext may result in a system compromise.

CWE-312: Cleartext Storage of Sensitive Information: The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere.

Fonte: obtido de https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage

3. Armazenamento e Verificação de Senhas

Mas, e quais soluções? **Criptografia Simétrica**

```
<?php

$username    = $_REQUEST['username'];

$secret_key  = $_ENV['MASTER_KEY'];
$nonce       = random_bytes(SODIUM_CCRYPTO_SECRETBOX_NONCEBYTES);
$ciphertext  = sodium_crypto_secretbox($_REQUEST['password'],
                                       $nonce, $secret_key);
$password    = base64_encode($ciphertext);

$sql = "INSERT INTO users (username, password, nonce)
VALUES ('" . $username . "', '" . $password . "', '" . $nonce . "')";

$conn = new mysqli("localhost", "root", "", "myDB");
$conn->query($sql)
$conn->close();
?>
```

3. Armazenamento e Verificação de Senhas
Mas, e quais soluções? Hashing de Senha

Atividade de verificação de integridade

3. Armazenamento e Verificação de Senhas
Mas, e quais soluções? ~~Hashing de Senha~~

<http://project-rainbowcrack.com/table.htm>

3. Armazenamento e Verificação de Senhas

Mas, e quais soluções?

Hashing de Senha+Salt


```
jshell> import java.security.SecureRandom
jshell> SecureRandom r = new SecureRandom();
r ==> Hash_DRBG,SHA-256,128,reseed_only
jshell> byte[] salt = new byte[16];
salt ==> byte[16] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
jshell> r.nextBytes(salt);
jshell> r
r ==> Hash_DRBG,SHA-256,128,reseed_only
jshell> r.toString()
$11 ==> "Hash_DRBG,SHA-256,128,reseed_only"
jshell> salt
salt ==> byte[16] { 23, -92, 61, 101, 13, 81, -99, 66, 54, -40, -53, -93, -96, 1, -45,
17 }
jshell> MessageDigest md = MessageDigest.getInstance("SHA-256");
md ==> SHA-256 Message Digest from SUN, <initialized>
jshell> md.update(salt);
jshell> salt
salt ==> byte[16] { 23, -92, 61, 101, 13, 81, -99, 66, 54, -40, -53, -93, -96, 1, -45,
17 }
jshell> md
md ==> SHA-256 Message Digest from SUN, <in progress>
jshell> byte[] hashedPass = md.digest("senha".getBytes());
hashedPass ==> byte[32] { -38, 88, 56, 47, -110, 51, -104, -127, ... 68, 34, 38, -53,
-56, 96 }
jshell> hashedPass.toString()
$18 ==> "[B@7c53a9eb"
```

3. Armazenamento e Verificação de Senhas

Mas, e quais soluções?

Hashing de **Senha+Salt*Número_Iterações**

Algoritmos/funções de Derivação de Senhas (*key derivation functions*).

Objetivo: aumentar complexidade computacional através da repetição do processo de hashing. Visa diminuir **vulnerabilidade** por **ataque de força bruta**.

PBKDF2 (Password-Based Key Derivation Function 1 and 2)

Argon2

bcrypt: usado em algumas distribuições Linux (derivado do Blowfish)

crypt: disponível no Linux e em algumas linguagens de programação

NTLMv2

3. Armazenamento e Verificação de Senhas

Mas, e quais soluções?

Hashing por algoritmos de derivação de senha

```
<?php
//Include database connection file
include 'dbconn.php';

if (isset($_POST['submit'])){
    $username = $_POST['username'];
    // Normal Password
    $pass = $_POST['password'];
    // Securing password using password_hash
    $secure_pass = password_hash($pass, PASSWORD_BCRYPT);
    $sql = "INSERT INTO login_tb (u_username, u_password)
    VALUES('$username', '$secure_pass')";
    $result = mysqli_query($conn, $sql);
}??>
```

3. Armazenamento e Verificação de Senhas




Mas, e quais soluções?

Hashing por algoritmos de derivação de senha

```
<?php
//Include database connection file
include 'dbconn.php';
```

```
if (isset($_POST['submit'])) {
```

	u_id	u_username	u_password
--	------	------------	------------

			3	GFG	\$2y\$10\$xde.d0zcMy0/tGbTTjvEY.CY8UP9kt1gNRZgWR/SNib...
--	---	---	---	-----	--

```
    // Securing password using password_nasn
    $secure_pass = password_hash($pass, PASSWORD_BCRYPT);
    $sql = "INSERT INTO login_tb (u_username, u_password)
VALUES('$username', '$secure_pass')";
    $result = mysqli_query($conn, $sql);
}??>
```

3. Armazenamento e Verificação de Senhas

Mas, e quais soluções?

Hashing por algoritmos de derivação de senha

Module `java.base`

Package `javax.crypto.spec`

Class `PBEKeySpec`

`java.lang.Object`

`javax.crypto.spec.PBEKeySpec`

All Implemented Interfaces:

`KeySpec`

`PBEKeySpec`

```
public PBEKeySpec(char[] password,  
                  byte[] salt,  
                  int iterationCount,  
                  int keyLength)
```

```
KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 65536, 128);  
SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
```

3. Armazenamento e Verificação de Senhas

Mas, e quais soluções?

Hashing por algoritmos de derivação de senha

```
mark:$6$.n.:17736:0:99999:7:::
```

```
[--] [----] [---] - [---] ----
```

					+----->	9. Unused
					+----->	8. Expiration date
					+----->	7. Inactivity period
					+----->	6. Warning period
					+----->	5. Maximum password age
					+----->	4. Minimum password age
					+----->	3. Last password change
					+----->	2. Encrypted Password
					+----->	1. Username

4. Códigos de autenticação de mensagens (MAC – Message Authentication Code)

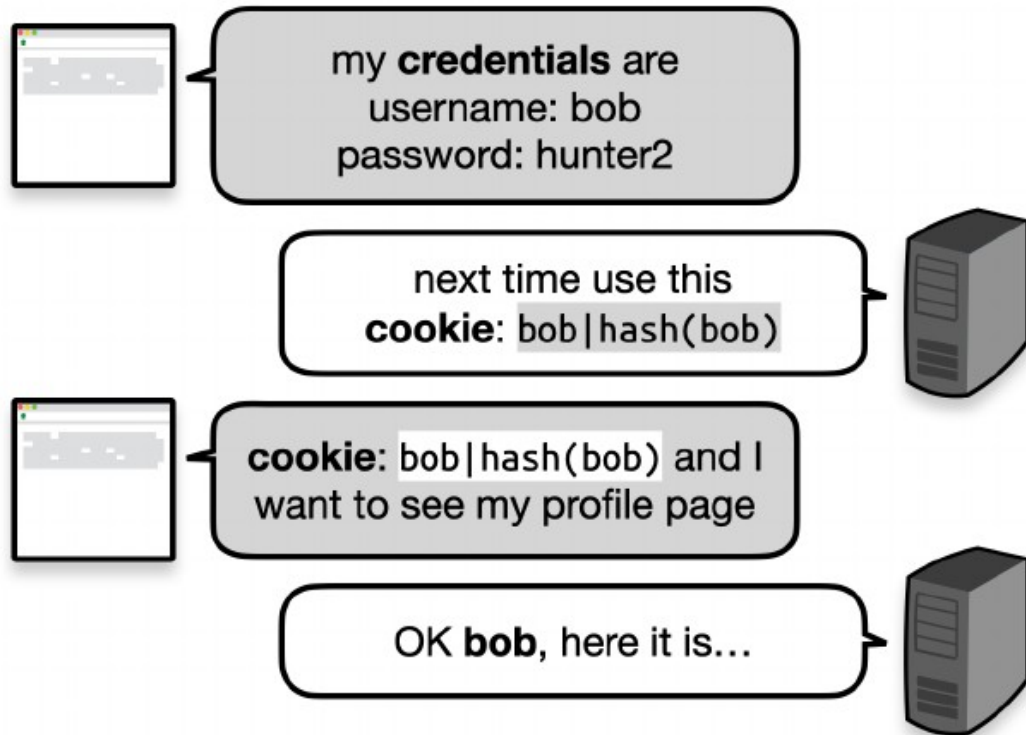
O MAC é uma função de hashing privada.



Considerações:

- MACs são resistentes à falsificação de origem;
- Tamanho do código associado com segurança;
- Problemas com ataques de **replay**.

4. Códigos de autenticação de mensagens (MAC – Message Authentication Code)





how are you?
MAC(k1, "how are you?")



fine and you?
MAC(k2, "fine and you?")



pretty good!
MAC(k1, "pretty good!")



fine and you?
MAC(k2, "fine and you?")



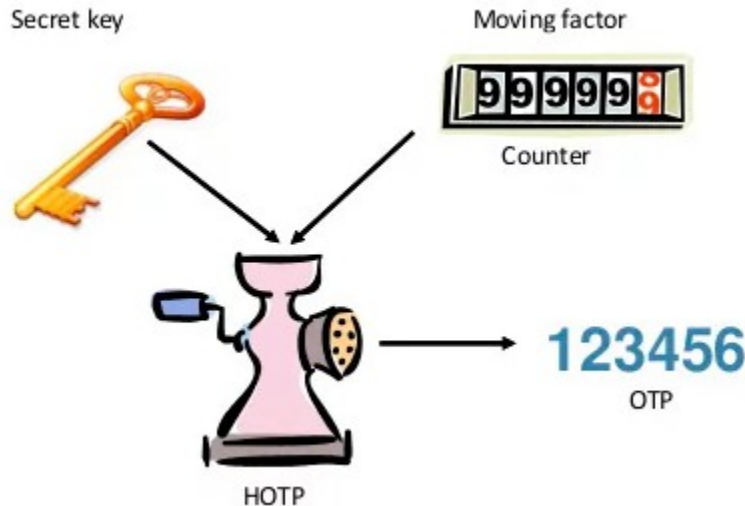
how are you?
MAC(k1, "how are you?")

Ataques de **Replay**: uma solução
é usar um
número sequencial ou
timestamp

4. Códigos de autenticação de mensagens (MAC – Message Authentication Code)

Autenticação 2FA e MFA

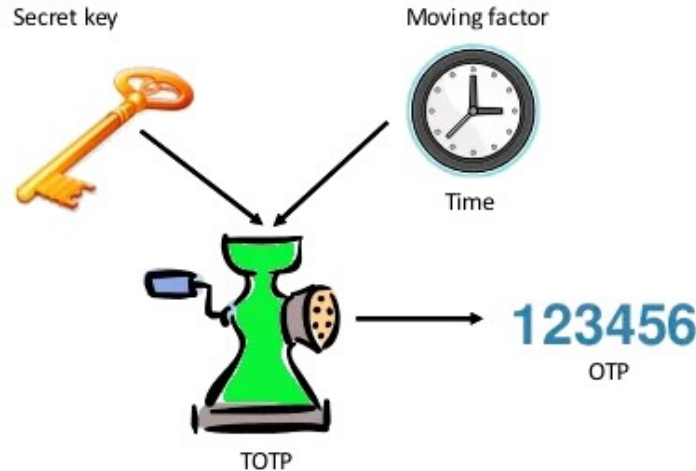
HOTP (HMAC-based One-time Password)



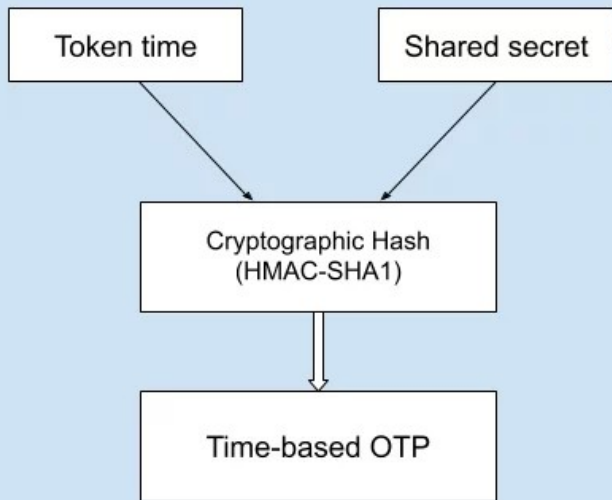
4. Códigos de autenticação de mensagens (MAC – Message Authentication Code)

Autenticação 2FA e MFA

TOTP (Time-Based One-time Password)

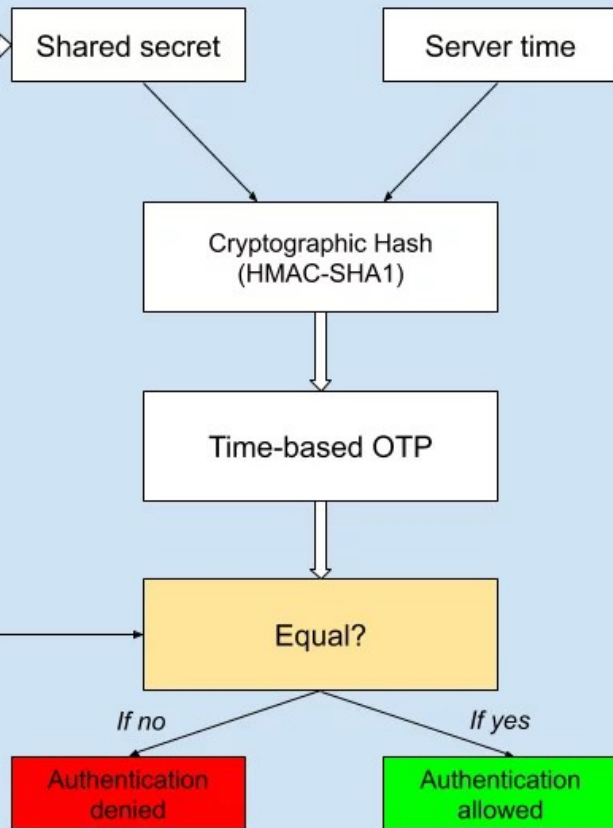


TOTP token



My OTP is 489523

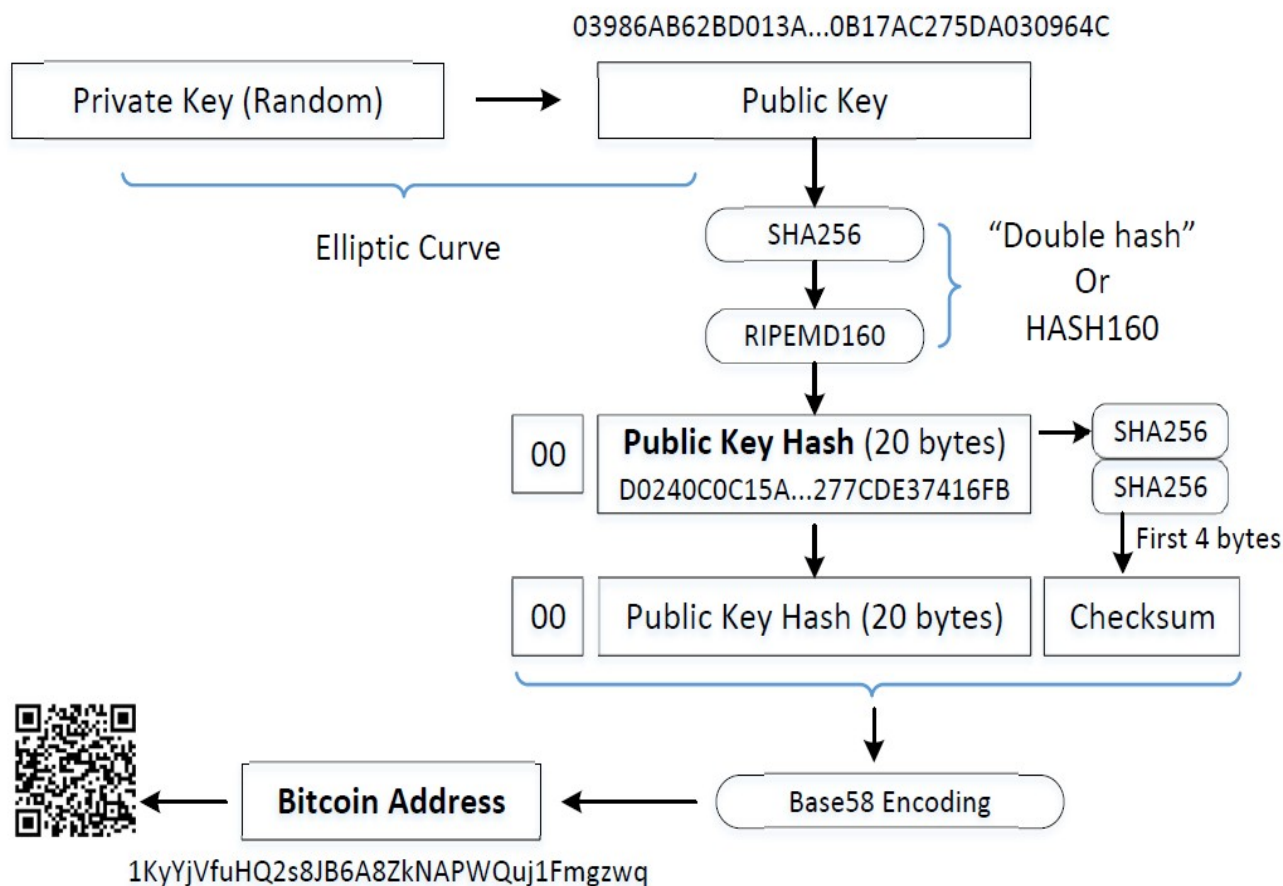
Server



5. Hashing em Bitcoin

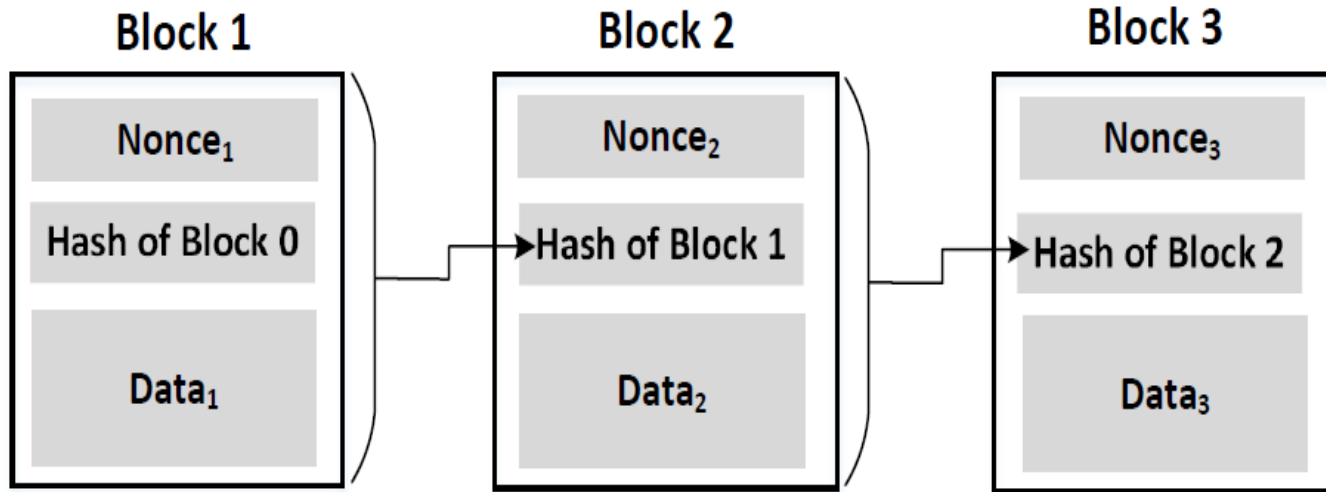
Uso do Hashing para obter um endereço Bitcoin a partir de uma chave pública.

Um usuário cria um par de chaves usando algoritmo de curva elíptica e o seu endereço de Bitcoin é gerado a partir da chave pública.



5. Hashing em Bitcoin

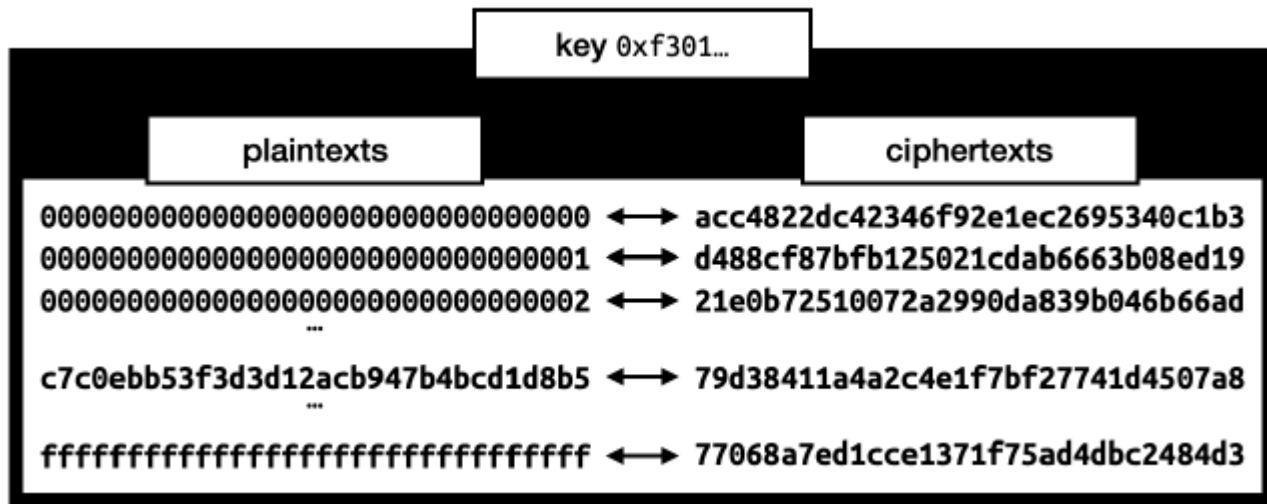
Uso do Hashing para criar blocos de transações aceitas (mineradas).



Criptografia Simétrica e Modos de Operação

Criptografia Simétrica

Algoritmo que toma um texto de tamanho variável e cifra para um texto cifrado de tamanho de bloco. As permutações aleatorizadas são controladas por uma chave (*pseudo-random permutations* - PRPs)

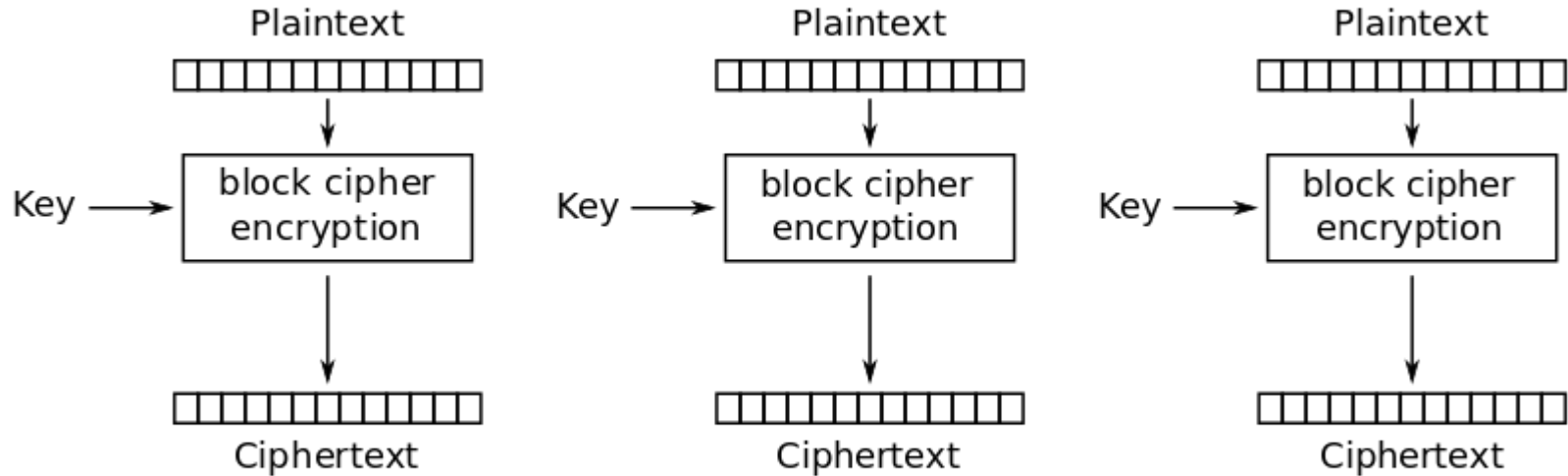


Criptografia Simétrica

Duas características de aplicação de algoritmos simétricos:

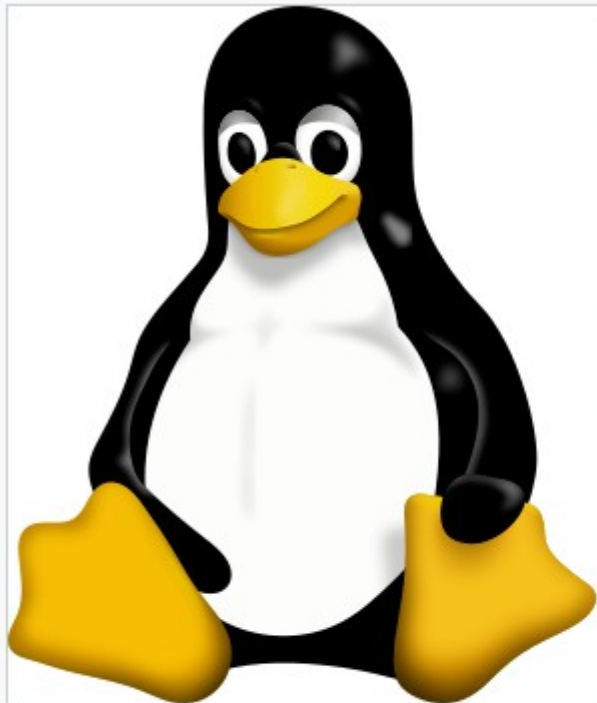
- **Modo de operação:** como tratar a cifragem de entradas com tamanho maior que o tamanho de um bloco;
- **Padding** (preenchimento): caso o número de bytes não complete o tamanho, como será preenchido o bloco.

Electronic codebook (ECB)

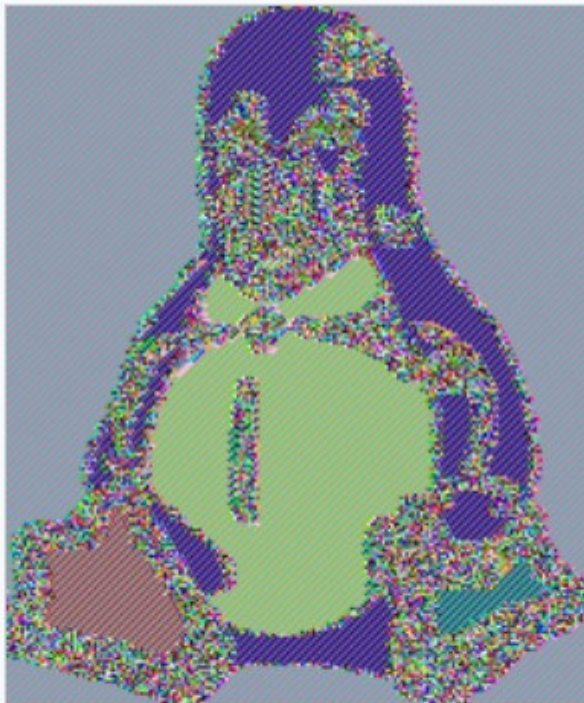


Electronic Codebook (ECB) mode encryption

Electronic codebook (ECB)



Original image

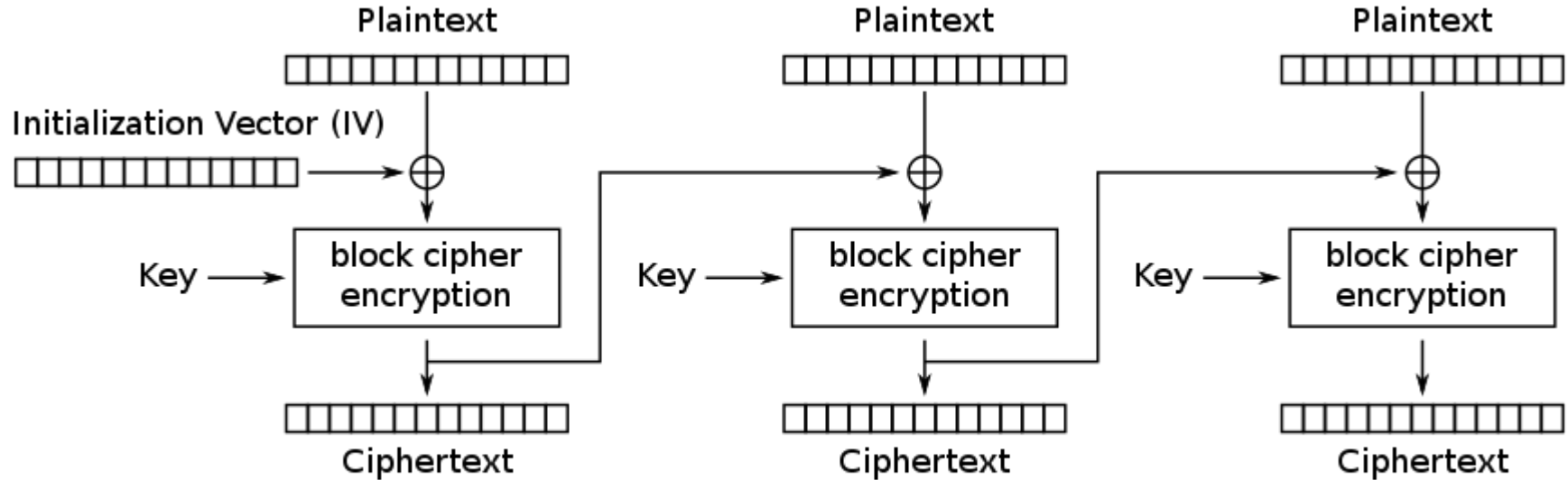


Using ECB allows patterns to be easily discerned



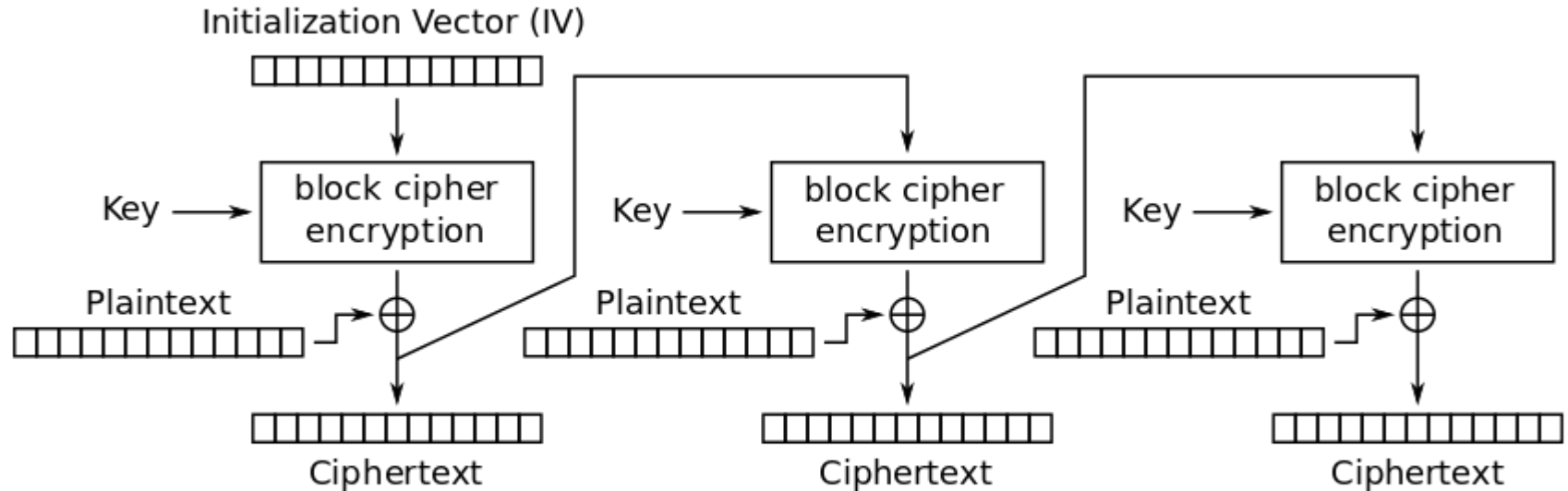
Modes other than ECB result in pseudo-randomness

Cipher block chaining (CBC)



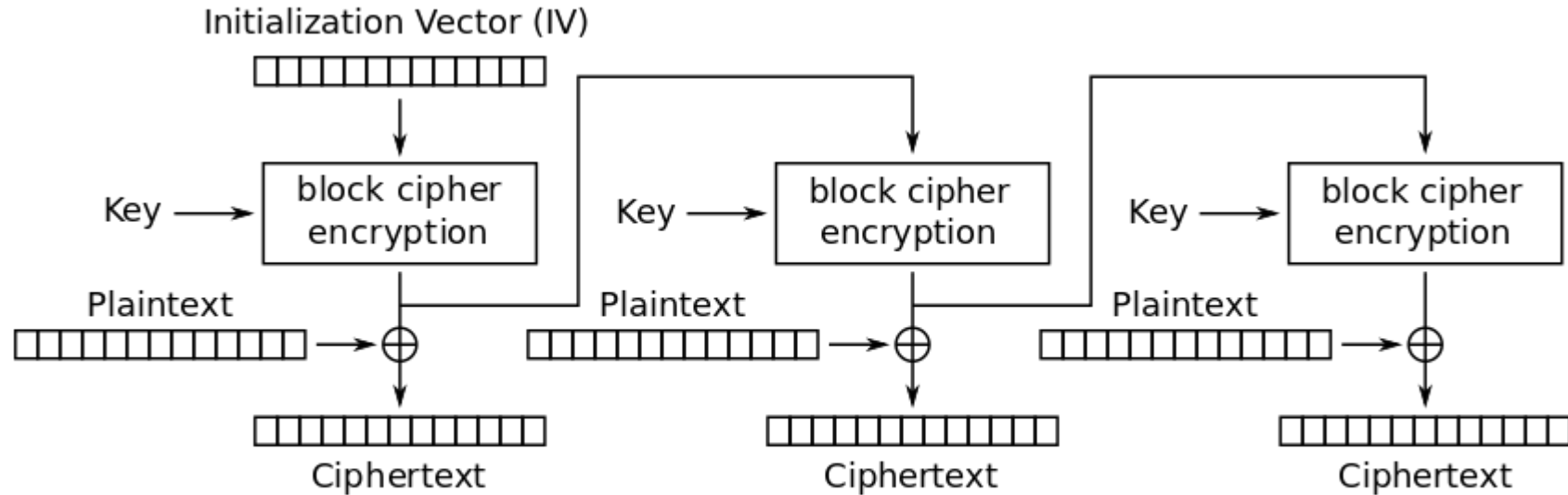
Cipher Block Chaining (CBC) mode encryption

Cipher feedback (CFB)



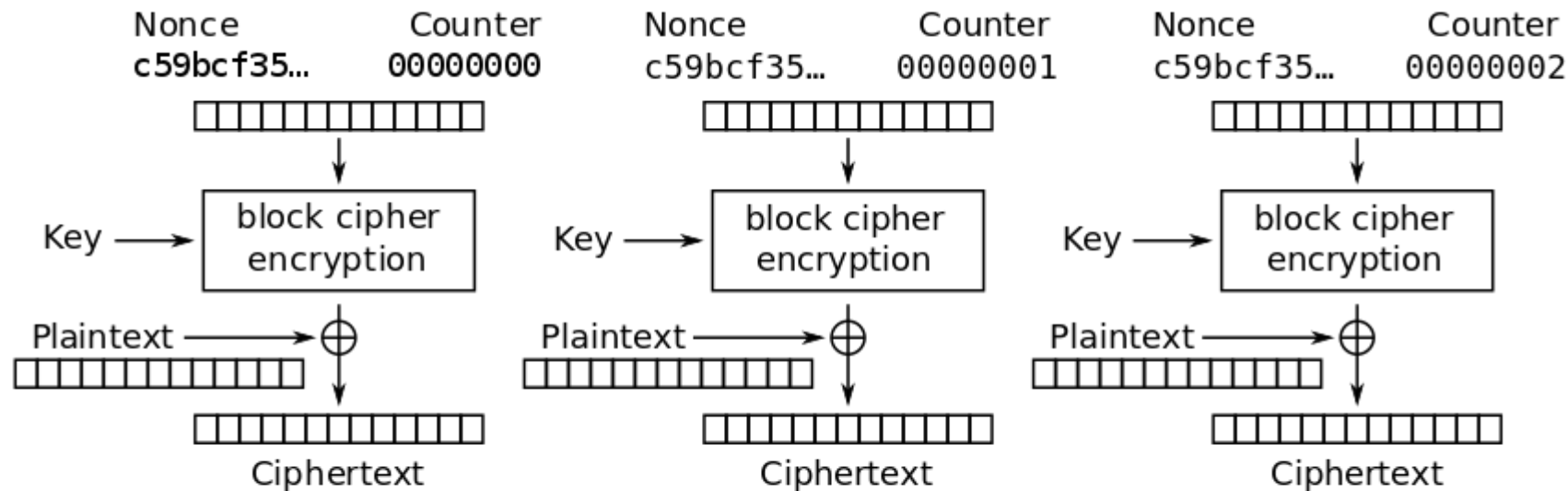
Cipher Feedback (CFB) mode encryption

Output feedback (OFB)



Output Feedback (OFB) mode encryption

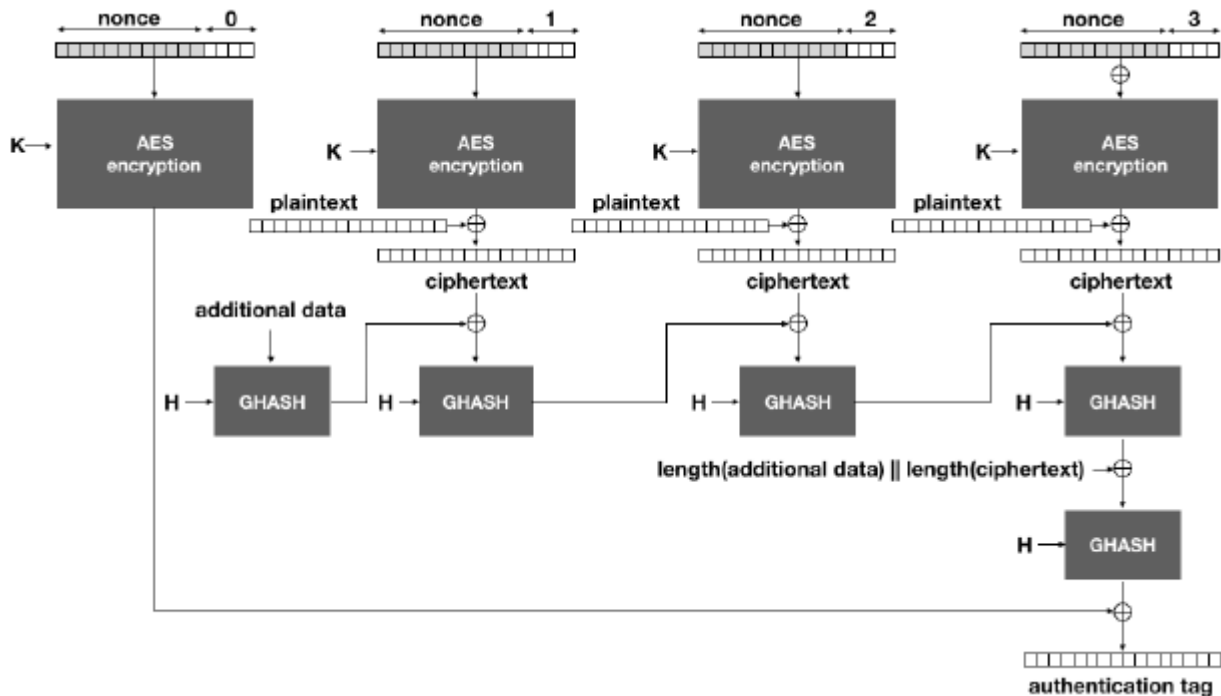
Counter (CTR)



Counter (CTR) mode encryption

AES-GCM

Usa o AES-CTR com uma **chave simétrica K** para cifrar o texto claro (plaintext) e o GMAC para autenticação de dado associado (associated data) e o texto cifrado (ciphertext) com uma **chave de autenticação H**.



AES-GCM


Exemplo de uso da autenticação cifrada com AES-GCM
(TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3) no protocolo TLS 1.3

Website Identity

Website: www.nasa.gov

Owner: This website does not supply ownership information.

Verified by: Amazon

[View Certificate](#)

Privacy & History

Have I visited this website prior to today? Yes, 10 times

Is this website storing information on my computer? No [Clear Cookies and Site Data](#)

Have I saved any passwords for this website? No [View Saved Passwords](#)

Technical Details

Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)

Atividade de verificação de cifragem simétrica usando CyberChef

Postura de segurança

Trust On First Use (TOFU): mecanismo de **autenticação** no qual a confiança do primeiro acesso de uma entidade. Suas premissas são:

- Um atacante nem sempre estará presente, durante o momento da conexão; e
- A janela de vulnerabilidade deste processo é pequena.

Protocolo SSH: exemplo do primeiro acesso a um servidor SSH

```
ssh user@localhost
The authenticity of host '[localhost]:22 ([127.0.0.1]:22)' can't be established.
ED25519 key fingerprint is SHA256:WRkXU8YWLdXBD018iIuap+wAwBhfU/+2U8cl0FUo2tM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? fingerprint
```

Artigo da Wikipedia no qual há informações sobre alguns protocolos que usam mecanismo TOFU: https://en.wikipedia.org/wiki/Trust_on_first_use

Bibliografia

- DIOGENES, Yuri; OZKAYA, Erdal. **Cybersecurity** – *Attack and Defense Strategies: Improve your security posture to mitigate risks and prevent attackers from infiltrating your system*. Third Edition. Packt Publishing Ltd, 2022.
- DU, Wenliang. *Computer Security: A Hands-on Approach*. 2nd Edition. Independent Published, 2019a.
- DU, Wenliang. *Internet Security: A Hands-on Approach*. 2nd Edition. Independent Published, 2019b.
- GOLLMANN, D. *Computer Security*. 3. ed. UK: John Wiley & Sons, 2011. 456 p.
- KUROSE, Jim e ROSS, Keith. *Computer Networking: A Top-Down Approach*. 8th edition. Pearson, 2020
- STALLINGS, William. *Criptografia e segurança de redes: princípios e práticas*. Tradução de Daniel Vieira. 6. ed. São Paulo: Pearson Education do Brasil, 2015. 558p.