Machine Translated by Google

Agencia para
Ciberseguridad Nacional

GPDP | GARANTE PER LA PROTEZIONE DEI DATI PERSONALI

PAUTAS FUNCIONES CRIPTOGRÁFICO

Almacenamiento de contraseñas

DICIEMBRE 2023

Machine Translated by Google

Este documento, que forma parte de los “Lineamientos de Funciones Criptográficas”, desarrollado por la Agencia Nacional de Ciberseguridad, en convenio con el Garante para la Protección de Datos Personales, contiene recomendaciones respecto a la conservación de contraseñas.
El documento tiene en cuenta las amenazas presentes el día de su publicación. Dada la diferente naturaleza de los sistemas de información objetivo, no es posible garantizar que estas recomendaciones puedan utilizarse sin adaptaciones específicas.

En todo caso, la pertinencia de la implementación de las soluciones propuestas deberá ser sometida, previamente, a evaluación y validación por parte de los responsables de la seguridad de los sistemas de información del destino.

El documento ha sido editado, en particular, por Simone Dutto, Sergio Polese y Giordano Santilli, criptógrafos expertos que trabajan en la División de Escrutinio Tecnológico, Criptografía y Nuevas Tecnologías del Servicio de Certificación y Supervisión de ACN.

Queremos agradecer, en particular, a Dorotea Alessandra De Marco y Marco Coppotelli, funcionarios de la Autoridad Garante para la Protección de Datos Personales, por su colaboración.

Versión	Fecha de publicación	Nota
1.0	12/07/2023	Primera publicación

Machine Translated by Google

Resumen

página

1. Introducción 5
2. Hashing de contraseñas 6
   2.1. El salto 7
   2.2. El pimiento 8
   2.3. Actualización de la función hash de contraseñas 8
3. Algoritmos principales 9
   3.1. PBKDF2 9
   3.2. cifrado 11
   3.3. bcrypt 12
   3.4. Argón2 13
4. Conclusiones dieciséis

Bibliografía 18

Índice de figuras

Figura 1 - Algoritmo PBKDF2 10

Figura 2: Algoritmo BlockMix interno de scrypt 11

Figura 3 - Función de expansión de bcrypt 13

Figura 4 - Función de compresión de Argon2 14

Índice de tablas

Tabla 1: Algoritmos de hash de contraseñas recomendados con sus parámetros mínimos 17

Machine Translated by Google

Lista de símbolos matemáticos utilizados

Símbolo	Descripción	Símbolo	Descripción
{0, 1}	Campo binario de valores que puede asumir un solo bit		Operación XOR, es decir, suma bit a bit de cadenas binarias
{0, 1}n	Espacio vectorial de cadenas binarias de longitud n	entero(s)	Convertir el número entero i en una cadena binaria de 32 bits
{0, 1}	Conjunto de cadenas binarias de longitud arbitraria.	x	Parte entera inferior de x, es decir, el entero más grande menor o igual a x
	Concatenación de cadenas	mi,j	El elemento de la matriz M ubicado en la fila i y la columna j

Machine Translated by Google

1 Introducción

Hoy en día, el acceso a la mayoría de los sistemas y servicios de TI requiere pasar uno o más procedimientos de autenticación de TI que a menudo incluyen el uso de una palabra clave (contraseña).

La gestión de contraseñas es un aspecto fundamental de la seguridad informática y la protección de datos personales. Por lo tanto, los administradores de sistemas y servicios deben proporcionar medidas técnicas y organizativas efectivas para el almacenamiento, retención y uso de contraseñas.

Los archivos en los que se almacenan las contraseñas terminan cada vez más en manos de terceros después de los ciberataques y luego se publican en línea o se utilizan para llevar a cabo otros ataques.

Por estos motivos, se recomienda encarecidamente el uso de funciones hash de contraseñas criptográficas seguras. Este documento tiene como objetivo proporcionar orientación y recomendaciones sobre las funciones que actualmente se consideran más seguras.

El documento presenta la siguiente estructura: en el capítulo 2 se introduce el concepto de hash de contraseñas, centrando la atención en las propiedades que deben satisfacer las funciones y en los posibles ataques a los que pueden estar sujetos los archivos de contraseñas; en el capítulo 3 Se presentan en detalle los algoritmos más comunes utilizados para el hash de contraseñas. Finalmente, en el capítulo 4 Se proporcionan indicaciones sobre qué algoritmos se recomiendan y sobre sus respectivos parámetros.

Machine Translated by Google

2 hash de contraseña

Una función hash toma como entrada una cadena de bits de longitud arbitraria y devuelve una cadena de bits de longitud fija, llamada resumen. Los detalles y recomendaciones para funciones hash se pueden encontrar en el documento dedicado.

Una de las propiedades de las funciones hash criptográficas es la de ser unidireccionales, es decir, no invertibles: dado un resumen, es computacionalmente difícil encontrar una entrada que permita obtenerla a través de la función hash.
Esta característica hace que las funciones hash sean adecuadas para Conservación de contraseñas: en lugar de almacenar contraseñas en texto plano en el archivo, se guarda su resumen para que cualquier persona malintencionada que pueda acceder a ellas no pueda acceder directamente a las contraseñas, sino solo a su resumen. Por tanto, en este contexto hablamos de hash de contraseñas [1]. La extrema complejidad de revertir las funciones hash hace imposible que el atacante recupere las contraseñas originales de los usuarios. Por el contrario, es sencillo verificar la exactitud de la contraseña del usuario en el momento de la solicitud de acceso, ya que basta con calcular el resumen utilizando la función hash utilizada en fase de guardado de contraseña y comparar el valor obtenido con el presente en el archivo.
Aunque el hash de contraseñas mejora la seguridad del almacenamiento de contraseñas, es importante prestar atención a algunos aspectos delicados que pueden dejar lugar a vulnerabilidades. El principal escenario a tener en cuenta es una violación de datos, es decir, el incidente en el que se filtra toda o parte de la lista que contiene resúmenes de contraseñas guardados en un servidor. Una vez que el atacante ha obtenido la lista de pares usuario-digest, puede proceder por fuerza bruta, es decir, calculando el hash de contraseñas aleatorias hasta encontrar una coincidencia, o puede llevar a cabo un ataque al diccionario [2]. Este tipo de ataque aprovecha el hecho de que los usuarios suelen elegir contraseñas triviales o palabras que tienen sentido, por lo que un atacante podría calcular el hash de las contraseñas más comunes hasta encontrar una coincidencia dentro de la lista filtrada.

Es importante tener en cuenta que cualquiera que esté en posesión del archivo también puede identificar a todos los usuarios que utilizan la misma contraseña, ya que, en este caso, los resúmenes también son los mismos.

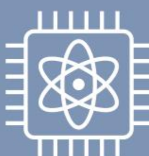
Es importante no utilizar directamente funciones hash criptográficas comunes, ya que estas últimas están optimizadas para realizar cálculos muy rápidamente y existen métodos para acelerar la generación de resúmenes que harían que cualquier ataque sea mucho más rápido.

Por lo tanto, es esencial que los algoritmos de hash de contraseñas tengan los siguientes requisitos: • una

complejidad computacional tal que sea rápido calcular un único resumen, pero sea excesivamente costoso calcular un gran número de ellos, para disuadir a los atacantes de buscar al usuario. contraseñas mediante intentos de procedimiento;

• una capacidad de memoria requerida que satura la RAM cuando se calculan muchos resúmenes simultáneamente.

Por estas razones, cuando hablamos de hash de contraseñas, necesitamos algoritmos ad hoc que tengan como objetivo ralentizar las capacidades ofensivas del atacante.
Una solución alternativa, que cumpla con los requisitos descritos anteriormente, podría incluir el cifrado de contraseña.
Sin embargo, esta opción no se recomienda, principalmente debido a la complejidad de administrar las claves criptográficas utilizadas al cifrar las contraseñas.



Seguridad cuántica

Como la mayoría de la criptografía simétrica, las funciones de hash de contraseñas son susceptibles a ataques informáticos cuánticos utilizando el algoritmo Grover [3], que, sin embargo, sólo proporciona un aumento cuadrático en la velocidad de los ataques de fuerza bruta. Por lo tanto, duplicar el tamaño del resumen nos permite garantizar el mismo nivel de seguridad que los estándares actuales, haciendo inútiles las mejoras cuán

2.1. El salto

Como se señaló anteriormente, en caso de que personas malintencionadas adquieran el archivo de contraseñas, el ataque de diccionario es una alternativa válida a la simple fuerza bruta.
Además, para reducir el esfuerzo computacional y, por tanto, los tiempos de ataque a expensas de la memoria utilizada, se suelen utilizar tablas arcoíris.
o "tablas arcoíris" [4], es decir, tablas precompiladas que contienen resúmenes de una gran cantidad de contraseñas comunes. Estos resúmenes se pueden comparar directamente con los del archivo, lo que hace que el ataque sea extremadamente rápido.

Para defenderse de este ataque, se recomienda utilizar algoritmos de hash de contraseñas que implican agregar un salt [5] a cada contraseña. La sal es una cadena de bits aleatorios que se concatena con la contraseña antes de calcular el resumen y luego se guarda en texto sin cifrar junto con el resumen de la contraseña del usuario. En caso de que se desee un mayor nivel de seguridad, el salt también se puede guardar en un archivo diferente al que contiene las contraseñas.

La sal no obstaculiza el procedimiento de autenticación informática del usuario: como se guarda en texto claro, basta con concatenarla con la contraseña introducida por el usuario durante el inicio de sesión antes de aplicar el algoritmo de hash de contraseña utilizado. Aunque no está cifrado, salt proporciona protección en varios frentes:

• en un ataque de diccionario, cuando el atacante calcula el resumen de una contraseña, para cada usuario debe concatenar el salt correspondiente, que será válido sólo para ese único usuario. Por tanto, el número de aplicaciones de la función hash que el atacante tendrá que realizar

intentar identificar a los usuarios utilizando una contraseña particular crece a medida que aumenta el tamaño del almacén de contraseñas de los usuarios;

- dos usuarios que utilicen la misma contraseña tendrán sal diferente y por lo tanto se les asociarán dos digestiones diferentes. Por lo tanto, un atacante no podrá identificar a varios usuarios utilizando la misma contraseña con un solo cálculo;
- las tablas rainbow quedan inutilizables, ya que incluso las contraseñas más comunes son modificadas por el salt y por lo tanto es necesario recalcular la tabla sumando todas las sales posibles para cada contraseña, haciendo claramente el proceso más oneroso.

Entonces, en general, salt no aumenta el nivel de seguridad al almacenar la contraseña de un usuario específico, pero permite ralentizar las capacidades ofensivas de un atacante en todo el archivo en proporción directa a su tamaño.

Para que sea eficaz, la sal debe:

- generarse aleatoriamente para cada contraseña;
- tener una longitud adecuada; de lo contrario, un atacante aún tendría la capacidad de precalcular una tabla de arco iris en particular, concatenando todas las sales posibles con las contraseñas más comunes.

2.2. El chile

Para agregar una capa adicional de seguridad, a veces se utiliza otra herramienta criptográfica llamada pepper [6]. El pepper es una cadena de bits aleatoria que, a diferencia del salt, puede ser la misma para todas las contraseñas del archivo, pero debe mantenerse en secreto ya que se utiliza como clave de un HMAC o un mecanismo de cifrado simétrico aplicado al resumen de contraseñas. .

Entonces, en el caso de la pimienta, el archivo contendrá los HMAC o el cifrado de resúmenes de contraseñas, que se compararán con los obtenidos a partir de las contraseñas introducidas por los usuarios.

2.3 Función hash de contraseña actualizada

En caso de que desee aumentar el nivel de seguridad de un sistema de autenticación que utiliza uno antiguo

Función de hash de contraseña h , al adoptar un algoritmo H más seguro , es necesario actualizar los resúmenes de contraseñas ya guardados en el archivo. Una posible estrategia se muestra a continuación:

1. la nueva función H se aplica a los resúmenes antiguos obtenidos a través de h, reemplazando los valores antiguos en el archivo con los obtenidos, es decir, para cada contraseña P, se guarda H(h(P)) , en lugar de h (PAG). De hecho, calcular el resumen de un resumen no supone un riesgo de seguridad si se utilizan funciones de hash de contraseña adecuadas
2. correspondiente a cada elemento contenido en el archivo, se inserta una bandera para realizar un seguimiento de la necesidad de actualizar el resumen;
3. en el primer inicio de sesión de cada usuario, si es necesario actualizar el resumen, después de verificar la correspondencia entre H(h(P)) y el valor guardado en el archivo, se calcula H(P) , que luego se guarda. como nuevo valor del resumen, y se desactiva el flag mencionado anteriormente. De esta forma, más adelante sólo se utilizará el algoritmo H.

En ciertos contextos, otros métodos que requieren que los usuarios actualicen sus contraseñas guardando directamente el resumen obtenido con el nuevo algoritmo de hash de contraseñas podrían ser una solución más rápida.

Machine Translated by Google

3 algoritmos principales

Como se señaló anteriormente, se desaconseja encarecidamente calcular un resumen para almacenar contraseñas mediante la simple aplicación única de una función hash criptográfica como las indicadas en el documento dedicado. En este contexto, una práctica común es adoptar funciones de derivación de claves [1], diseñadas para obtener una o más claves secretas a partir de un valor secreto inicial, como una clave maestra.

A pesar del diferente propósito de construcción, estas funciones satisfacen de manera excelente las propiedades requeridas para el hash de contraseñas y, por lo tanto, se encuentran entre los algoritmos más utilizados. Las funciones de derivación de claves suelen ser configurables según sea necesario con respecto a todas o algunas de las siguientes características:

- el coste computacional, normalmente representado por el número de iteraciones de una función del algoritmo que, si aumenta, ralentiza su ejecución;
- la memoria utilizada, es decir, la memoria máxima necesario para ejecutar el algoritmo. En este contexto, es importante resaltar que la memoria requerida por todas las operaciones no debe desviarse mucho del valor máximo, de lo contrario esta distribución diferente podría proporcionar información al atacante (ataques de canal lateral [1]);
- la posibilidad de paralelización, como el algoritmo se podría dividir en varias partes independientes por el otro y por tanto ejecutable simultáneamente. Si existe esta posibilidad, el usuario normalmente puede elegir el número de procesos simultáneos.

Estas características están definidas por parámetros personalizables. Si, por un lado, se hace posible la aplicación en diferentes contextos, por otro, esta libertad exige responsabilidad en la elección de parámetros que no afecten a la seguridad del algoritmo.
A continuación se presentan algunas recomendaciones sobre algoritmos de hash de contraseñas, incluidas indicaciones sobre los valores mínimos de estos parámetros.

3.1 PBKDF2

La función de derivación de clave basada en contraseña 2 (PBKDF2) es una función de derivación de clave basada en contraseña diseñada en 1999 y publicada en 2000 como parte de la serie "Estándares de criptografía de clave pública" (PKCS). [7]. Estas publicaciones forman parte de los estándares creados y publicados por los laboratorios RSA, empresa especializada en seguridad informática propietaria, entre otras, de la patente del famoso sistema criptográfico de clave pública RSA. El funcionamiento de PBKDF2 se ilustra en la Figura 1: la función toma como entrada una contraseña P, una sal S y, mediante la aplicación de un HMAC repetido durante un número preestablecido de iteraciones c, calcula una cadena DK de longitud len. En fórmulas,

$$DK = PBKDF2(P, S, c, HMAC, len).$$

Para obtener información más detallada sobre los algoritmos para generar HMAC, consulte el documento dedicado a los códigos de autenticación de mensajes.

El número de bloques que ℓ conformarán DK es el mínimo que permite alcanzar la longitud len ℓ > 232 deseado. En el caso de , PBKDF2 regresa un error y no continúa con el cálculo. Esto evita crear un resumen excesivamente largo para memorizar.

El estado inicial para el i-ésimo bloque se forma concatenando la sal con el contador convertido i en binario de 32 bits (tipo entero), es decir, S int(i). Este valor se actualiza realizando para c el algoritmo HMAC veces consecutivas utilizando siempre la contraseña P como clave. El valor final Ti del bloque es el XOR ( ) entre todos los estados parciales obtenidos. Es importante observar como las operaciones realizadas sobre los bloques individuales son completamente independientes entre sí y por tanto los valores finales de Ti se pueden obtener en paralelo.

Finalmente, el valor DK es la concatenación de todos los estados de Ti . es decir

$$DK = T1 \quad ... \quad Tℓ,$$

en el que el último bloque se trunca para obtener el longitud deseada .

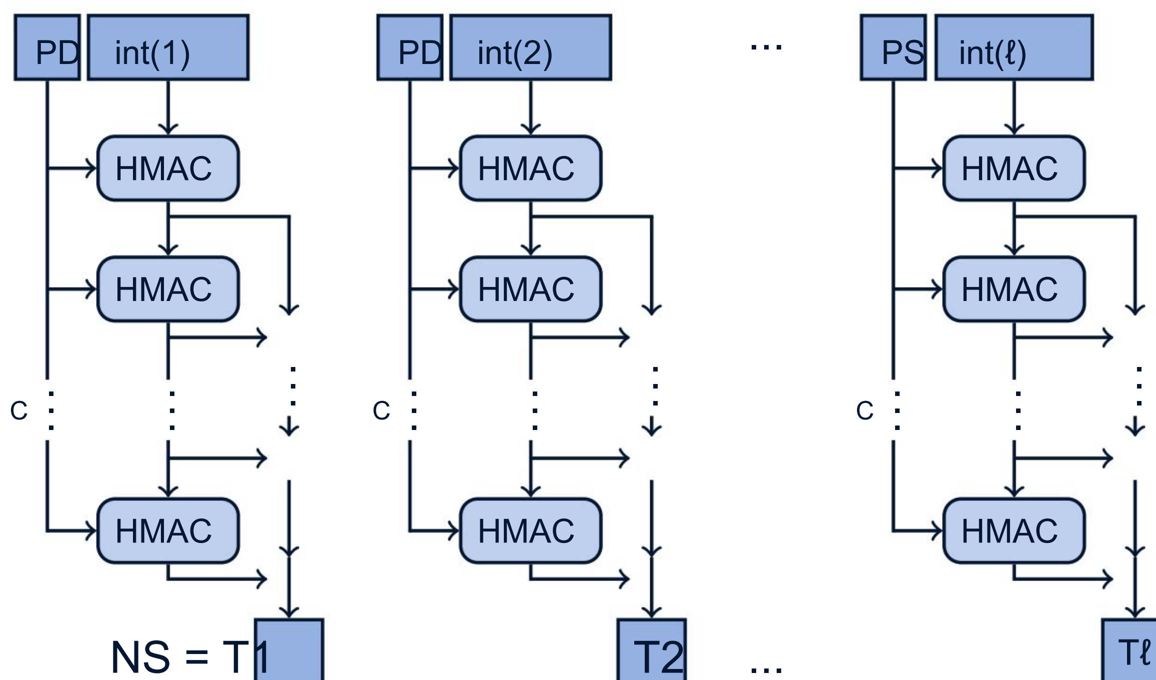


Figura 1 - Algoritmo PBKDF2

Según las especificaciones NIST de 2010 [8] y el RFC de 2017 [9], la sal S debe tener al menos 128 bits de longitud obtenida mediante un generador de números aleatorios aprobado en las recomendaciones indicadas en [10], [11], [12]. , el número de iteraciones c debe ser adecuado a la capacidad computacional del sistema utilizado y superior a 1000, y len debe tener al menos 112 bits de longitud. Sin embargo, cabe señalar que los valores mínimos sugeridos actualmente para estos parámetros son demasiado bajos para permitir una seguridad adecuada, teniendo en cuenta las capacidades informáticas cada vez mayores disponibles para los atacantes a través de procesadores de última generación (CPU, GPU, ASIC, FPGA). Por lo tanto, se planean actualizaciones que aumentarán el número de iteraciones para garantizar una seguridad adecuada y excluirán el uso de HMAC-SHA-1, que actualmente todavía recomienda el NIST como función interna. Por el momento, los estudios más recientes [13] recomiendan utilizar al menos 600.000 iteraciones con HMAC-SHA256 o al menos 210.000 iteraciones con HMAC-SHA512.

3.2 guión

scrypt (léase “esse”-crypt en italiano o “es”-crypt en inglés) es una función de derivación de claves basada en contraseña creada en 2009 por Colin Percival [14] [15] con el objetivo de hacer menos efectivos los ataques basados en implementaciones de hardware especializadas. En comparación con sus predecesores, scrypt proporciona un nuevo parámetro que se puede personalizar para hacer que el algoritmo sea más seguro cuando se utiliza en el contexto del hash de contraseñas, es decir, la memoria necesaria para un cálculo completo del algoritmo. Maximizar la memoria requerida tiene el efecto de ralentizar las implementaciones de hardware especializadas, sin afectar el rendimiento de las implementaciones de software.

La función toma como entrada una contraseña P, un salt S, un parámetro n que representa el requisito de memoria y capacidad computacional (en términos de mebibytes o MiB, es decir, 220 bytes de CPU y RAM necesarios), un parámetro r que determina el tamaño de los bloques, un parámetro p para paralelización y la longitud len de la salida:

$$DK = scrypt(P, S, n, r, p, len).$$

La inicialización, procesamiento y finalización de scrypt ocurren de la siguiente manera:

1. aplicar una única iteración de PBKDF2 con HMAC-SHA256 para obtener p bloques de longitud 1024·r bits, eso es

$$P1 \quad ... \quad Pp = PBKDF2(P, S, 1, HMAC\text{-}SHA256, p \cdot 1024 \cdot r);$$

2. Se procesan los bloques individuales en paralelo, aplicando a cada uno la función Mix descrita a continuación, obteniendo para cada uno 1≤i≤p

$$Pi = Mezclar(r, Pi, n);$$

3. finalmente se aplica nuevamente una única iteración de PBKDF2 con HMAC-SHA256, pero usando bloques obtenido en forma de sal, para obtener

$$DK = PBKDF2(P, P1 \quad ... \quad Pp, 1, HMAC\text{-}SHA256, len).$$

La función Mix utilizada dentro de scrypt introduce los requisitos de memoria y capacidad computacional al trabajar en un único bloque Pi a través de una función interna adicional llamada BlockMix que se describe más adelante.

La salida X de Mix(Pi,n,r) se obtiene mediante el siguiente procedimiento:

1. copie Pi en el bloque X;

2. construir n bloques V1 = X, Vj = BlockMix(Vj–1,r) por cada 2≤j ≤n y actualizar X = Vn;

3. n veces, calcule un nuevo índice 1≤j ≤n dependiente de X y actualización X = MezcladeBloques(X Vj , r)·

El algoritmo BlockMix de scrypt, a su vez, utiliza un cifrado de flujo llamado Salsa20, diseñado por Daniel J. Bernstein en 2005 [16]. Como se describe en la Figura 2, la entrada X se divide inicialmente en 2·r bloques de 512 bits y se aplica Salsa20 al XOR entre el primer y el último bloque.

Posteriormente se actualiza el estado alternando el XOR con el bloque posterior a la aplicación de la función Salsa20. Los estados intermedios alcanzados después de cada aplicación de Salsa20 constituyen la salida final del algoritmo, pero se reorganizan de tal manera que los primeros r bloques de la salida son los de índice impar y los r restantes son los de índice par, es decir.

$$MezcladeBloques(X, r) = Y1 \quad Y3 \quad ... \quad Y2r-1 \quad Y2 \quad Y4 \quad ... \quad Y2r.$$

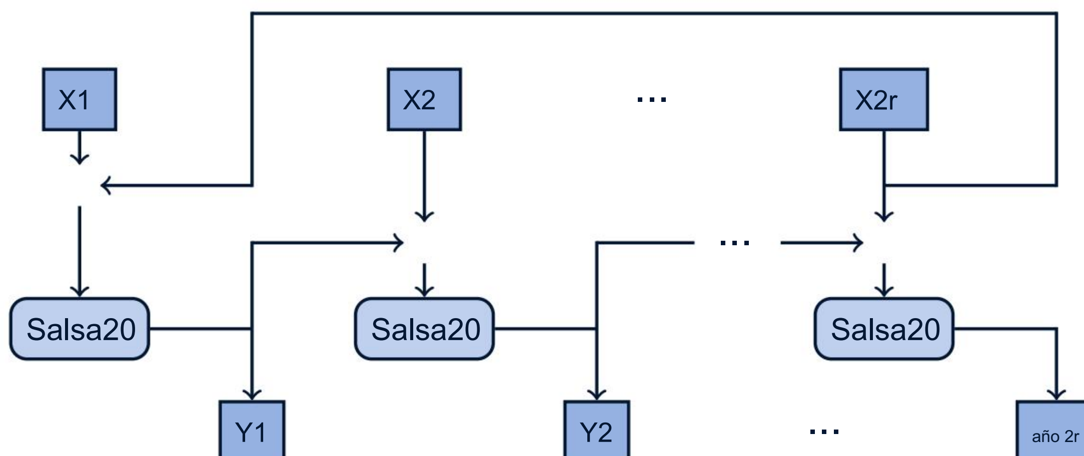


Figura 2: Algoritmo BlockMix interno de scrypt

Machine Translated by Google

3.3 bcripta

A diferencia de las otras funciones descritas en este documento, bcrypt ("bi"-crypt) fue diseñada específicamente para ser una función de hash de contraseñas y no una función de derivación de claves. Fue introducido por Provos y Mazières en 1999 [17] y utiliza un cifrado de bloques, específicamente Blowfish, implementado en modo ECB [18]. La idea es crear un algoritmo de programación de claves para el cifrado de bloques que tenga altos requisitos computacionales para aumentar el tiempo necesario para ejecutar el cifrado y luego aplicar el cifrado varias veces.

La función toma como entrada la contraseña P, el salt S, de 128 bits de longitud, y un parámetro c que identifica el número de ciclos a realizar, determinando el coste computacional necesario para el hash de contraseña. En fórmulas,

$$H = \text{bcripta}(P, S, c).$$

Los pasos principales son:

1. el cifrado de bloque se inicializa mediante un

   Función de configuración con costo computacional dependiente de c. Específicamente, Blowfish requiere 18 claves redondas $K_1, \dots, K_{18}$ de 32 bits y 4 S-boxes (cajas de sustitución) $SB_1, \dots, SB_4$, que asocian salidas de 32 bits con entradas de 8 bits, guardadas como una tabla de consulta de 1 KB (256 entradas de 32 bits) cada una. Estos datos constituyen el estado interno.

   $$\sigma = (K_1,...,K_{18}, SB_1,..., SB_4).$$

   Específicamente, la función Configuración (P,S,c) realiza los siguientes pasos:

   a. las cifras de la constante π se copian como valor inicial en las teclas redondas $K_1, \dots, K_{18}$ y en las entradas individuales de las casillas S $SB_1, \dots, SB_4$;

   b. la función de expansión clave que se describe a continuación se utiliza para actualizar el estado σ aprovechando la contraseña y la sal, en fórmulas

   $$\sigma = \text{Expandir}(\sigma, P, S);$$

   c. durante 2c veces, se alternan las expansiones σ que usan solo la sal y solo la contraseña, para aumentar el costo computacional de todo el algoritmo, es decir

   $$\sigma = \text{Expandir}(\sigma, S, 0),$$

   $$\sigma = \text{Expandir}(\sigma, P, 0);$$

2. Se crea una cadena constante de 192 bits, definida como

   T = "OrpheanBeholderScryDoubt",

   que se cifra iterativamente 64 veces con Blowfish en modo y estado del BCE σ, actualizando T de vez en cuando con el cifrado obtenido, es decir

   $$T = \text{pez globoECB}_\sigma(T);$$

3. el resultado final es la concatenación de c con la sal S y la T obtenido, en fórmulas

   $$\text{bcrypt}(P, S, c) = c \quad S \quad T.$$

La función Expandir(σ, P, S) toma como entrada el σ estado que consta de las claves redondas $K_1, \dots, K_{18}$ y las cajas S $SB_1, \dots, SB_4$, una cadena P de longitud arbitraria y una cadena S de 128 bits, y lleva a cabo operaciones de actualización de claves redondas y S- cajas realizando cifrado con Blowfish.

Machine Translated by Google

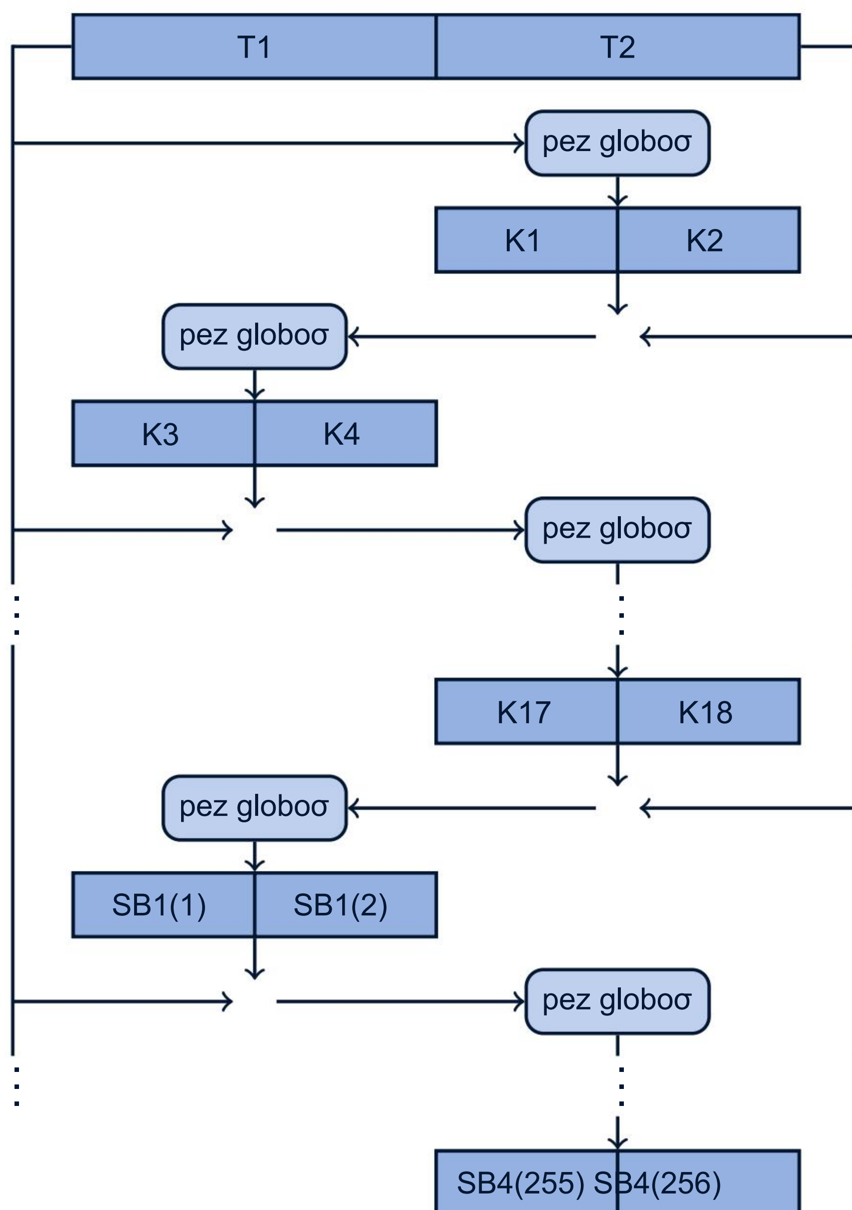


Figura 3 - Función de expansión de bcrypt

En particular, las claves redondas K1,…,K18 del estado se σ actualizan inicialmente calculando el XOR entre sus valores iniciales y la cadena de 576 bits obtenida repitiendo varias veces P. Posteriormente, S se divide en dos partes de 64 bits. 'uno, S = S2 S1, que se agregará vía XOR alternativamente a los cifrados parciales obtenidos.

Luego, S1 se cifra con Blowfish utilizando el estado. El resultado σ corresponde al primer par de claves K1,K2, con las que se actualiza el estado y cuyo XOR con S2 se vuelve σ a cifrar con Blowfish para obtener K3,K4.

Luego se repite el proceso hasta obtener todas las claves redondas. Posteriormente, procedemos a

Modifique también las S-boxes, generando dos salidas de S-boxes en cada paso, según el diagrama de la figura. Esto se repite 4·128 veces para que todas las entradas de todos los S-box se actualicen una vez. El resultado de

La función es, por tanto, el nuevo estado σ.

3.4 Argón2

En 2013, un grupo de criptógrafos abrió una convocatoria de algoritmos de hash de contraseñas [19] para seleccionar nuevos estándares criptográficos para almacenar contraseñas.

Hasta la fecha límite para la presentación de algoritmos, se habían presentado 24 candidatos y comenzó la contraseña.

Machine Translated by Google

Competencia de hash (PHC) [20]. Al final del concurso, en julio de 2015, solo uno de estos algoritmos fue seleccionado como ganador: Argon2. Este concurso, sin embargo, no forma parte de las clásicas selecciones oficiales del NIST, sino que forma parte de un esfuerzo privado de la comunidad científica para proponer nuevas soluciones para la protección con contraseña. Tras este resultado y la ausencia de ataques conocidos en la literatura, algunos organismos internacionales [13] ya han reconocido a Argon2 como la mejor opción en general para el hash de contraseñas.

Argon2 es una función de derivación clave diseñada por Alex Biryukov, Daniel Dinu y Dmitry Khovratovich [21].
El algoritmo está diseñado para maximizar el coste de los ataques a contraseñas realizados con máquinas ASIC (capaces de obtener numerosos resúmenes simultáneamente), gracias a la cantidad personalizable de memoria necesaria durante su cálculo.
Los autores propusieron diferentes versiones del algoritmo:

- Argon2d, que utiliza acceso a memoria dependiendo de los datos que esté procesando. De esta forma, el algoritmo es más resistente a los ataques perpetrados con herramientas dedicadas, como GPU y ASIC, pero más sensible a los ataques de canal lateral. Por estos motivos, esta versión está dedicada a aplicaciones blockchain y relacionadas con prueba de trabajo;
- Argon2i, que no proporciona acceso a la memoria dependiente de los datos, por lo que es más resistente a los ataques de canales laterales. Esta versión se utiliza principalmente como función de derivación de claves y como función de hash de contraseñas;
- Argon2id, versión mixta de los anteriores, que implica el uso de Argon2i en la primera mitad del proceso y Argon2d en la segunda mitad, para garantizar la protección tanto de ataques de canal lateral como de ataques dedicados de fuerza bruta.

La función toma como entrada, además de la contraseña P, el salt S y la longitud len de la salida, también parámetros para aumentar la complejidad computacional, es decir, c para el número de iteraciones, m para la cantidad de memoria y p para la grado de paralelización personalizable. En fórmulas,

$$DK = \text{Argón2}(P, S, c, m, p, len).$$

En esencia, Argon2 utiliza una función de compresión f, que transforma dos entradas de 1024 bytes cada una en una única salida de 1024 bytes, y una función hash h. En particular, la documentación oficial obtiene h como XOF (función de salida extensible) a partir de Blake2b [22], una versión mejorada del candidato de competencia SHA-3 Blake [23].

Como se describe en la Figura 4, la función f calcula el XOR entre las dos entradas y organiza el resultado en una matriz de 8×8 con entradas de 16 bytes cada una. Posteriormente, esta matriz se modifica aplicando π, es decir, la función redonda de Blake2b, primero a cada fila y luego a cada columna de la matriz. El resultado final se obtiene calculando el XOR entre la matriz obtenida y la matriz inicial.

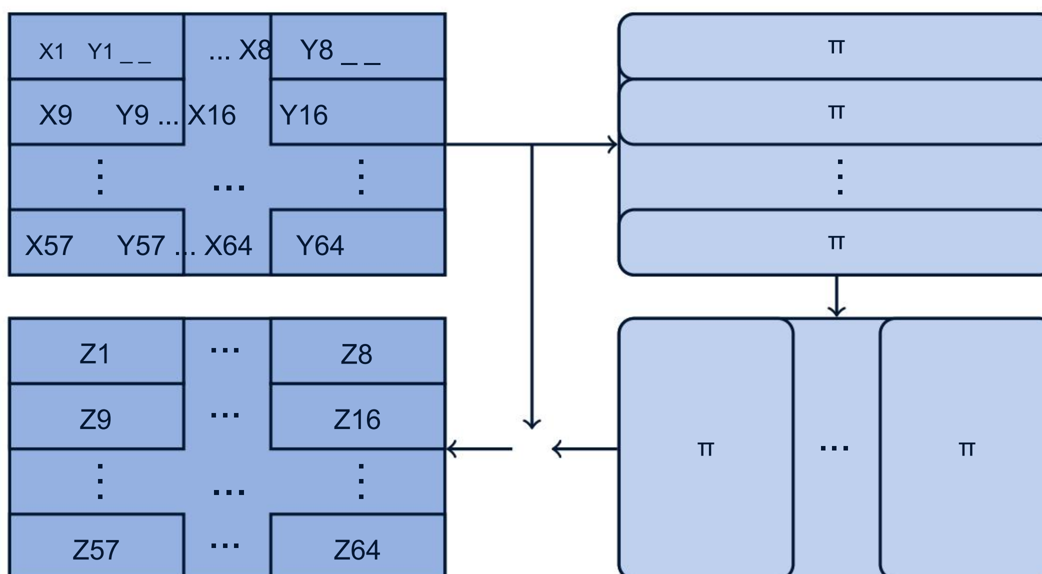


Figura 4 - Función de compresión de Argon2

Argon2 procesa las entradas calculando un resumen de la contraseña, la sal y una serie de otros parámetros iniciales con la función hash h y genera el estado inicial H. A continuación, se genera una matriz B con p filas y q = 4 · m /4p columnas, compuestas por bloques de 1024 bytes obtenidos como

$$B_{i,1} = h(H \quad 0 \quad i), 1 \leq i \leq p,$$

$$B_{i,2} = h(H \quad 1 \quad i), 1 \leq i \leq p,$$

$$B_{i,j} = f(B_{i,j-1}, B_{i',j'}) \quad 1 \leq yo \leq p, 3 \leq j \leq q,$$

donde los índices i y j se determinan de manera diferente en las diferentes versiones de Argon2, determinando las características descritas anteriormente.

Si el costo c es estrictamente mayor que 1, la matriz B se actualiza c –1 veces de la siguiente manera:

$$B_{i,1} = B_{i,1} \quad f(B_{i,q}, B_{i',j'}), \quad 1 \leq yo \leq p,$$

$$B_{i,j} = B_{i,j} \quad f(B_{i,j-1}, B_{i',j'}), 1 \leq i \leq p, 2 \leq j \leq q.$$

La elección de los índices i y j se realiza de manera que en cada paso sea posible realizar p operaciones en paralelo con respecto al índice de fila.

Finalmente, el resultado final se obtiene aplicando la función hash h al XOR entre los elementos de la última columna de la matriz B, es decir:

$$DK = h(B_{1,q} \quad B_{2,q} \quad ... \quad B_{p,q}).$$

Machine Translated by Google

4 Conclusiones

Al concluir los análisis realizados en este documento, recomendamos utilizar los algoritmos de hash de contraseñas indicados en la Tabla 1, donde también se muestran los parámetros mínimos recomendados para los diferentes algoritmos.

En general, el uso de salt es una condición obligatoria para cualquier algoritmo de hash de contraseñas y no se recomienda el uso de soluciones personalizadas.
El algoritmo PBKDF2 debe garantizar una seguridad adecuada, por lo que se recomienda utilizar únicamente las funciones hash indicadas en el documento dedicado. maximizando el número c de iteraciones.

En cuanto a bcrypt, está bastante anticuado y no es recomendable, también dada la cantidad de avances y mejoras obtenidas por otros algoritmos.
Finalmente, scrypt y Argon2id son los algoritmos más robustos y eficientes y deben priorizarse sobre cualquier otra opción.

En cuanto a los parámetros mínimos de Argon2id, se subraya que aumentar el grado de paralelización p requiere necesariamente el aumento de al menos uno de los otros dos parámetros, es decir, el número de iteraciones cy la memoria requerida m. En cualquier caso, se recomienda evaluar un tamaño adecuado de los parámetros de Argon2id en función de su capacidad computacional y de memoria, para garantizar una ejecución rápida con una sola contraseña, pero que haga factible un gran número de ejecuciones.

<table>
<tr><th>Algoritmo</th><th colspan="5">Parámetros</th></tr>
<tr><td rowspan="3">PBKDF2</td><td>Longitud mínima de sal</td><td>Longitud del resumen (len)</td><td>Número de iteraciones (c)</td><td colspan="2">algoritmo HMAC</td></tr>
<tr><td rowspan="2">128 bits</td><td rowspan="2">128 bits</td><td>600.000</td><td colspan="2">HMAC-SHA256</td></tr>
<tr><td>210.000</td><td colspan="2">HMAC-SHA512</td></tr>
<tr><td rowspan="6">cifrar</td><td>Longitud mínima de sal</td><td>Longitud del resumen (len)</td><td>Tamaño de bloque (r)</td><td>Requerimiento de memoria y capacidad computacional (n)</td><td>Grado de paralelización (p)</td></tr>
<tr><td rowspan="5">128 bits</td><td rowspan="5">128 bits</td><td rowspan="5">8</td><td>128 MB</td><td>1</td></tr>
<tr><td>64 MB</td><td>2</td></tr>
<tr><td>32 MB</td><td>3</td></tr>
<tr><td>16 MB</td><td>5</td></tr>
<tr><td>8 MB</td><td>10</td></tr>
<tr><td rowspan="8">Argón2id</td><td>Longitud mínima de sal</td><td>Longitud del resumen (len)</td><td>Número de iteraciones (c)</td><td>Requisito de memoria (m)</td><td>Grado de paralelización (p)</td></tr>
<tr><td rowspan="7">128 bits</td><td rowspan="5">128 bits</td><td>1</td><td>46 MB</td><td rowspan="5">1</td></tr>
<tr><td>2</td><td>19 MB</td></tr>
<tr><td>3</td><td>12 MB</td></tr>
<tr><td>4</td><td>9 MB</td></tr>
<tr><td>5</td><td>7 MB</td></tr>
<tr><td rowspan="2">256 bits</td><td>1</td><td>2048 MB</td><td rowspan="2">4</td></tr>
<tr><td>3</td><td>64 MB</td></tr>
</table>

Tabla 1: Algoritmos de hash de contraseñas recomendados con sus parámetros mínimos

Machine Translated by Google

Bibliografía

[1] D. Stinson y M. Paterson, Criptografía: teoría y práctica, CRC press, 2018.

[2] R. Shirey, «RFC 4949 - Glosario de seguridad de Internet, versión 2», 2007.

[3] L. Grover, «Un algoritmo mecánico cuántico rápido para la búsqueda de bases de datos», en Actas del vigésimo octavo simposio anual ACM sobre teoría de la computación, 1996.

[4] P. Oechslin, «Hacer un intercambio de memoria y tiempo criptoanalítico más rápido», en Advances in Cryptology - CRYPTO, 2003.

[5] NIST, «SP 800-63-3 - Directrices de identidad digital», 2020.

[6] U. Manber, «Un esquema simple para hacer que las contraseñas basadas en funciones unidireccionales sean mucho más difíciles de descifrar», Computers & Security, vol. 15, núm. 2, págs. 171-176, 1996.

[7] B. Kaliski, «RFC 2898 - PKCS #5: Especificación de criptografía basada en contraseñas», 2000.

[8] NIST, «SP 800-132 - Recomendación para la derivación de claves basada en contraseñas, Parte 1: Aplicaciones de almacenamiento», 2010.

[9] K. Moriarty, B. Kaliski y A. Rusch, «RFC 8018 - PKCS #5: Especificación de criptografía basada en contraseñas, versión 2.1», 2017.

[10] Organización Internacional de Normalización, «ISO/IEC 20543:2019: Tecnología de la información - Técnicas de seguridad - Métodos de prueba y análisis para generadores de bits aleatorios dentro de ISO/IEC 19790 e ISO/IEC 15408», 2019.

[11] NIST, «SP 800-90A - Recomendación para la generación de números aleatorios utilizando generadores deterministas de bits aleatorios», 2015.

[12] NIST, «SP 800-90B - Recomendación para las fuentes de entropía utilizadas para la generación aleatoria de bits», 2018.

[13] OWASP, «Hoja de referencia para el almacenamiento de contraseñas», 2023. [En línea]. Disponible: https://cheatsheetseries.owasp.org/cheatsheets/Contraseña_Almacenamiento_Cheat_Sheet.html. [Consultado el 28 de junio de 2023].

[14] C. Percival, «Derivación de claves más sólida mediante funciones secuenciales de memoria dura», 2009.

Machine Translated by Google

Bibliografía

[15] C. Percival, «RFC 7914 - La función de derivación de clave basada en contraseña de scrypt», 2016.

[16] DJ Bernstein, «La familia Salsa20 de Stream Ciphers», en New Stream Cipher Designs: The eSTREAM Finalists, 2008.

[17] N. Provos y D. Mazières, «A Future-Adaptable Password Scheme», en la Conferencia Técnica Anual de USENIX, 1999.

[18] B. Schneier, «Descripción de una nueva clave de longitud variable, cifrado de bloque de 64 bits (Blowfish)» en Fast Software Encryption (FSE), 1993.

[19] «Convocatoria de presentaciones de APS», 2013. [En línea]. Disponible: https://www.password-hashing.net/cfh.html. [Consultado el 03 de julio de 2023].

[20] «Concurso de hash de contraseñas», 2015. [En línea]. Disponible: https://www.password-hashing.net/. [Consultado el 03 de julio de 2023].

[21] A. Biryukov, D. Dinu y D. Khovratovich, «RFC 9106 - Función de memoria dura Argon2 para aplicaciones de prueba de trabajo y hash de contraseñas», 2021.

[22] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn y W. Christian, «BLAKE2: Simpler, Smaller, Fast as MD5», en Criptografía aplicada y seguridad de redes, 2013.

[23] J.-P. Aumasson, W. Meier, R. Phan y L. Henzen, The Hash Function BLAKE, Berlín: Springer, 2014.