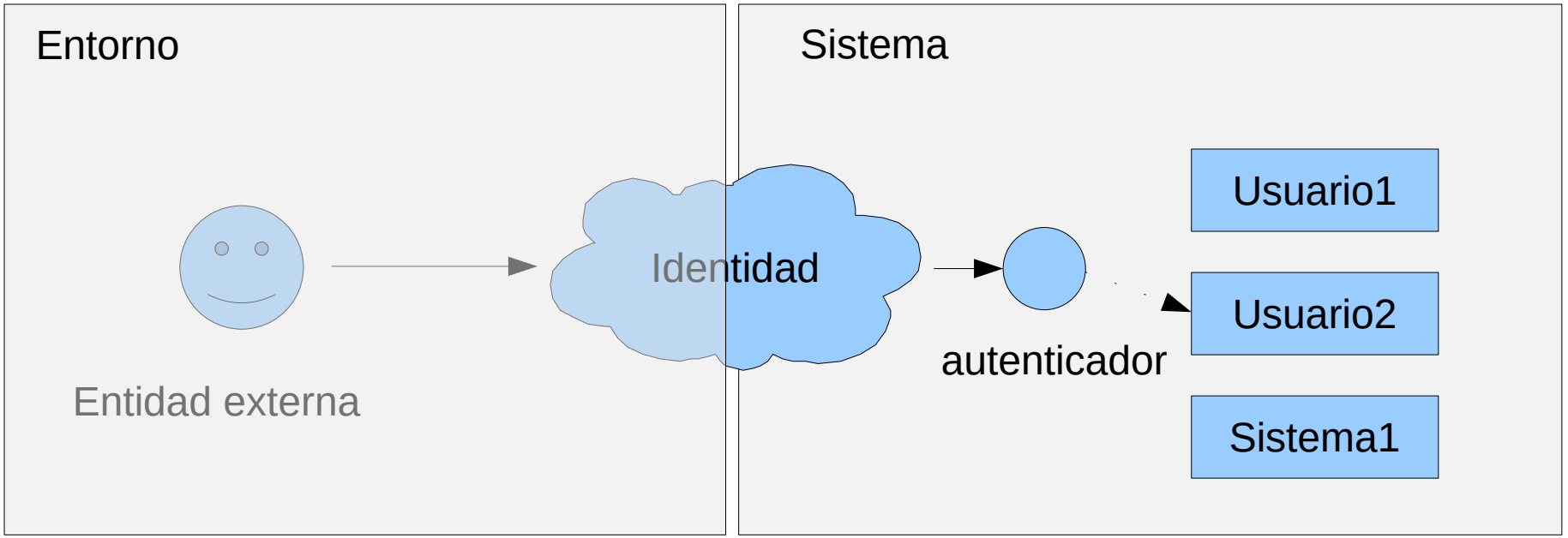
Criptografía y Seguridad

Seguridad en aplicaciones: Autenticación

Autenticación

- Asociar una identidad a un principal
  - La identidad corresponde a una entidad externa
  - El principal es una representación interna del sistema



Podriamos definir la seguridad en un sistema definiendo objetos y las acciones que se pueden ejecutar sobre el. Sin embargo, en los sistemas reales, nuestros sujetos suelen ser externos a nuestro sistema. Por lo tanto, tenemos que poder asociar una entidad externa a una entidad interna (usuario o principal). Aparece un proceso nuevo que se llama proceso de autenticacion.

Confirmación de identidad

- Pasos:
  - Obtener información de identidad
  - Analizarla
  - Determinar si corresponden a la entidad
- Consecuencias:
  - Hay que almacenar información de cada entidad
  - Hay que contar con mecanismos para procesar la información

Sistema de Autenticación

Esta asociado a la parte de "integridad" en lo que es seguridad en sistemas

Formalmente un sist. de autenticacion basico son dos conjuntos de informacion y tres funciones

- Componentes
  - A = { a } - Información de autenticación (puede ser provista de forma externa)
    - La designación de la entidad e información provista
  - C = { c } - Información complementaria (asociada a principales, y esta almacenada en el sistema)
    - Contiene la especificación de un principal e información almacenada por el sistema
  - F = { F: A -> C } - Funciones de complementación
    - Deriva información complementaria (a partir de informacion externa, permite llegar a informacion complementaria permanente)
  - L = { L: AxC -> { 0, 1} } - Funciones de autenticación
    - Determina si un par A,C son una asociación válida (recibe a y c, y devuelve si es valida esa asociacion)
  - S = { s } - funciones de selección (le da dinamismo al sistema)
    - Funciones que permiten crear y actualizar A y C

Información de Autenticación

- Provista por una entidad externa
- Posee diferentes grados de confianza
- Fuentes
  - Algo que conozco
  - Algo que tengo
  - Algo que soy
  - Donde estoy (contexto)

Algo que conozco

- Ejemplos
  - Clave
  - Frase (passphrase)
  - Pregunta secreta
- Basan la identificación en secretos compartidos
  - Requiere almacenar de manera segura el secreto
  - Depende de la confidencialidad del secreto
  - Uno de los mecanismos más empleados
  - Sirve de base a otros mecanismos

Algo que tengo

- Ejemplos La idea es que todos estos tienen un chip interno con capacidad de computo propia.
  - Smart card/USB token
  - Generador de claves (por ejemplo RSA-ID)
  - Celular
  - Tarjeta de coordenadas (Token Grid)
- Basan la identificación en algún elemento físico
  - Requiere el cuidado del elemento
  - El elemento puede ser copiado / clonado
  - La información en transito puede ser suplantada

Algo que soy

- Ejemplos
  - Huellas digitales
  - Retina
  - Voz
  - Cara

Las caracteristicas son intrinsecas, entonces:
- Son dificiles que se pierdan
- No son transferibles (como si lo es una clave)
- No se pueden cambiar -> si alguien tiene nuestra huella, estamos en problemas

- Basan la identificación en características intrínsecas de la entidad
  - Las características no son fácilmente modificables
  - Los métodos actuales no son 100% eficaces
  - Asumen que el dispositivo de lectura no puede ser manipulado

El analisis de este tipo de factores es probabilistico. Si yo tengo una contraseña, el sistema puede analizar si es o no. Los metodos como el de la huella digital o de voz devuelven un x% de probabilidad de que la huella que se esta mostrando es la del usuario. Hay muchos casos de hermanos desbloqueando con huella digital el telefono del otro.

Contexto

Tiene que ver con cuestiones espacio-temporales

- Ejemplos
  - Red de origen/ Pais y ciudad / Geolocalización
  - Host / Terminal / Device de origen
  - Dia y hora de acceso, Velocity
- Basan la identificación en el contexto de la información que se recibe
  - Pueden actuar como factores positivos
    - Un operador debe conectarse desde la red privada del centro de computos
  - O como factores negativos
    - Si un usuario accede desde otro pais hay MENOS chances de que sea quien dice ser

Ejemplo de sistema de autenticación

- Autenticación con claves, guardadas como texto plano
- A = { x / x es clave }
- C = A
- F = { identidad }
- L = { igual }
- S = { adduser(), removeuser(), passwd() }

Claves

- Secuencias de caracteres

  Google hace poco inicio una campaña para eliminar el uso de claves para siempre. Pero vamos a tardar para llegar hasta ahi.

  - Con restricciones. Ej: 8 caracteres, 10 dígitos, etc
  - Generadas aleatoriamente, por el usuario o de manera asistida

    Las generadas aleatoriamente convienen mas por un tema de entropia

- Secuencias de palabras

  Una frase es mas memorable que una secuencia de caracteres. Se busca que sean mas largas, para aumentar la entropia

  - Llamadas frases clave (pass-phrases)
- Algorítmicas
  - Pregunta-respuesta (challenge-response)
  - Claves de uso único (OTP – One time passwords)

Almacenamiento

- Como texto en claro

  Esta es la forma mas simple de almacenar claves. Obviamente es la mas insegura, porque un administrador del sistema puede ver las claves de los usuarios.

  - Puede ser en un archivo o en una base de datos
  - Puede ser accedido por fuera del sistema
  - No puede garantizarse confidencialidad de las claves

Almacenamiento

- Archivo cifrado
  - Requiere una clave para acceder a las contraseñas
    El problema con esto es que hacemos con esta nueva clave
  - La clave puede estar
    - En un archivo de configuración
    - En el mismo ejecutable del sistema
    - Ser ingresada cada vez que inicia el sistema (es dificil de usar en cloud)
    - En un dispositivo criptográfico especializado

    Hoy en dia hay muchas herramientas que uno les pasa un binario o apk y nos devuelve todas las posibles claves que encuentre
  - Solo recomendado si es requisito accesar a la clave original
    - Por ejemplo, para reenviarla a un tercer sistema

Mas adelante veremos la forma correcta de (no) almacenar contraseñas: PBKDF2

Ejemplo: Sistema unix (legacy)

- Se almacena por cada usuario el resultado de una de 4096 funciones de hash
- A = { secuencias de hasta 8 caracteres }
- C = { xxHHHHHHHHHHH }
  - xx = funcion de hash (0-4095) – 2 caracteres
  - HH...H = Resultado de la función – 11 caracteres
- F = { 4096 versiones modificadas de DES }
- L = { login, su, sudo, ... }
- S = { passwd, adduser, ... }

Unix fue el primero que empezo a utilizar hashes de claves.
Guardaba la info de cada usuario en lineas en /etc/passwd
Al no almacenar la clave, el archivo /etc/passwd podia estar disponible para todo el mundo.
Con el tiempo, se decidio mover la info de autenticacion a el archivo /etc/shadow, que solo es legible por root. Esto tiene que ver con el principio de acceso minimo (y seguridad en capas)

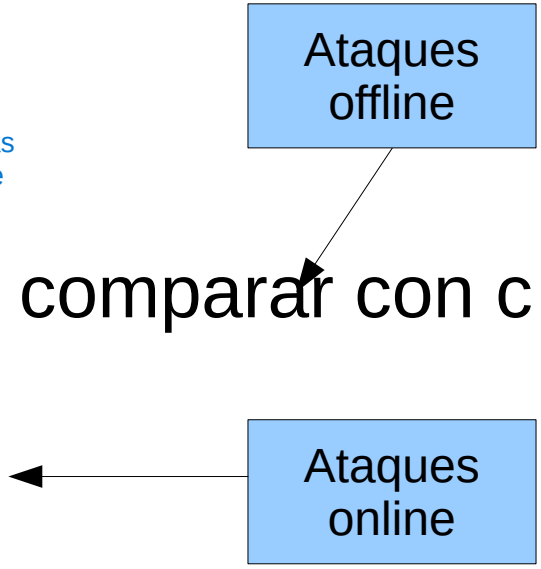
La funcion de hash tiene resistencia a las preimagenes. Esto nos permite probarle a una entidad que conocemos algo, sin revelarle ese algo.
Ademas, estas no requieren clave, por lo que tenemos un problema menos.
El cliente le pasa la clave en texto plano al servidor, y el servidor calcula el hash y lo verifica.
Es un error conceptual que el cliente calcule el hash y lo envie, porque entonces cualquiera puede leer el hash en la base de datos y enviar eso como cliente, haciendose pasar por nosotros.

Ataques a un Sistema de autenticación

- Objetivo del sistema: identificar correctamente a entidades
- Objetivo de un atacante: ser identificado como una entidad (impersonarla)
- Mecanismo: Encontrar a $\in$ A tal que:
  - Para algún f $\in$ F, f(a) = c
  - c está asociado a una entidad
- Verificación:
  - Probar varios a, computar f(a) y comparar con c
  - Intentar autenticar via l(a) $\in$ L

El mecanismo formal para atacar un sistema, es encontrar un conjunto de informacion, tal que para alguna de las funciones que hace autenticacion, nos matchee con la entidad que queremos impersonar

Ejemplo: calculo un monton de hashes de claves y las comparo con el hash del que quiero atacar. Es offline porque no requiere que el sistema este funcionando.



Es como el ataque offline, pero en vez de compararlo con un hash, se llama a la funcion "login" del sistema

Adivinando claves

- Prueba y error de una lista de claves potenciales
- Offline: se conoce f y c y se prueban para diferentes a si f(a) = c

  JohnTheRipper tiene un autogenerador de claves, y funciona para distintos SO como Linux y Windows

  - Ejemplo, obteniendo el archivo /etc/shadow de Linux, con programas como crack y john the ripper
  - Ejemplo, obteniendo el archivo SAM de windows, con programas como ophcrack
- Online: Haciendo uso de la función L, probando con diferentes a, hasta pasar la autenticación
  - Ejemplo, probado la función de login con una cuenta conocida (root, administrator, guest)

La ventaja de un ataque offline es que podemos paralelizarlo y hacerlo mas rapido. Los ataques online suelen ser muy lentos.

Prevenciones (general)

- Esconder información para que a, c o f no sean conocidas simultáneamente

  (nosotros no buscamos ocultar la funcion de hash o lo que sea, sino que buscamos ocultar la info complementaria, como /etc/shadow)

  - Por lo general f es conocida o puede conocerse
  - Es más fácil proteger c que a, que está en manos de la entidad externa
  - Ejemplo: /etc/shadow en unix
- Prevenir el uso a la función de autenticación

  Todas estas de abajo hacen que los ataques online sean mas dificiles

  - Tiempos crecientes ante fallas en la autenticación

    (por ejemplo, cuando pongo mal la clave, me contesta en 1ms, si la pongo mal de vuelta, me contesta en 100ms y asi hasta que no tenga sentido hacer un ataque online)
  - Deshabilitar principales (despues de n intentos fallidos, deshabilito el usuario por X tiempo)

    La inhabilitacion de cuentas sirve para sistemas internos, pero puede ser un problema para sistemas externos. Hay un caso que paso en ebay, en el cual un usuario desactivaba las cuentas de otros usuarios que estaban en las mismas subastas que el
  - Jailing / Honeypot (identificamos que nos estan atacando, y hacemos pensar al atacante que lo logro, llevandolo a un sitio falso. esto no se usa tanto, solo en sistemas ultraseguros)
  - Captchas

    (previene automatizaciones)

Prevenciones (general)

- Subir la complejidad de las claves
- Formula de Anderson: P = TG / N
  - P = probabilidad de adivinar una clave
  - G = número de pruebas realizables por segundo
  - T = tiempo dedicado a las pruebas
  - N = tamaño del espacio de claves
- Se puede utilizar para estimar el tiempo de un ataque

Ejercicio

- Considerar:
  - Claves numéricas de 10 dígitos
  - Se pueden probar 10.000 claves por segundo
- ¿Cual es el tiempo de búsqueda estimado para tener una probabilidad > 0.5?
  - $P \geq TG / N \Rightarrow T \leq PN / G = 0.5 * 10^{10} / 10.000$
  - $T \leq 500.000s \sim 6$ dias

Ejercicio (2)

- Considerar:
  - Claves de 8 letras (26 posibles caracteres)
  - Se pueden probar 10.000 claves por segundo
- ¿Cual es el tiempo de búsqueda estimado para tener una probabilidad > 0.5?
  - $P \geq TG / N \Rightarrow T \leq PN / G = 0.5 * 26^8 / 10.000$
  - $T \leq 1.044.135s \sim 121$ días

Ejercicio (3)

- Considerar:
  - Claves de alfanuméricas (36 posibles caracteres)
  - Se pueden probar 100.000 claves por segundo
- ¿Qué longitud mínima se debe requerir para que la probabilidad de que un atacante encuentra la clave en menos de 1 año sea menor a 0.5?
- $P \geq TG / N \Rightarrow$
  - $N \geq TG / P = (365*24*60*60)* 100.000 / 0.5$
  - $N=36^L \geq 6,3 * 10^{12}$
  - $L \geq \log_{36} (6,3 * 10^{12}) \sim 8.22 \Rightarrow L \geq 9$

Mejores ataques

- Hasta ahora se asume una búsqueda aleatoria

Entonces la formula de anderson nos da una cuota superior, pero hay ataques que son mas eficientes

- Mejoras:
  - Utilizar diccionarios de palabras
  - Utilizar diccionarios de claves descubiertas
  - Utilizar transformaciones simples de palabras
    - Sufijos
    - Prefijos
    - Reemplazar L -> 1, o -> 0, vocales por números
    - Palabras espejadas
    - Dos palabras combinadas
  - Utilizar información de contexto
    - Nombre, usuario, dni, fecha de nacimiento, etc

Selección de claves - alternativas

- Selección aleatoria
  - Condición ideal – cada clave es equiprobable
  - Difícil de memorizar (ej: fL3K&8%j”)

Se choca con el principio de diseño de " aceptacion psicologica", que busca no irrumpir mucho con la usabilidad

- Claves pronunciables
  - Palabras sin sentido, pero formadas por fonemas
  - Ejemplos: helgoret, mipoterjo, jusacila
  - Fáciles de memorizar
  - Problemas: el espacio de claves se ve muy reducido
- Permitir que el usuario elija
  - Problema: las claves tienden a ser fáciles de adivinar

Hay iniciativas a nivel global para eliminar las claves, porque ninguno de los 3 sistemas es muy bueno

Revisión proactiva de claves

- Consiste en el análisis de una clave al momento de crearla
  - Permite detectar y rechazar claves 'fáciles'
  - Debería permitir aplicar reglas sobre palabras
  - Debería permitir realizar búsquedas en diccionarios
  - Debería permitir realizar búsquedas de patrones
  - Debería utilizar información sobre el usuario y el contexto

La idea es ponerse en la posicion de hacker, e intentar hackear las claves de nuestros propios usuarios (con diccionarios y eso) todo el tiempo. Si lo logramos, le avisamos al usuario que la cambie y listo.

Expiración de claves

- Forzar a los usuarios a cambiar de clave luego de algún tiempo o evento
  - Limita el daño producido por una clave comprometida
  - Requisitos
    - Evitar el reuso de claves (recordar y bloquear las N últimas)
    - Evitar que se pueda cambiar de clave varias veces en un periodo de tiempo
    - Dar a los usuarios tiempo para pensar una nueva clave
      - No forzarlos a cambiar clave al momento de login
      - Avisar con anticipación

Salting

- Escenario:
  - Un atacante consigue c_1, c_2, ... c_n
  - Cada prueba a_i --> $f(a_i)$ puede probarse en paralelo contra todos los c_j
- Objetivo: aumentar el tiempo requerido para adivinar claves
- Método: introducir una perturbación
  - f(a) = x || f'(a, x)

Si nosotros tuviesemos n usuarios con la misma clave, le agregamos un salt aleatorio a la informacion complementaria. Esto hace que los usuarios con la misma clave terminen almacenando valores de hash distintos. El salting previene contra ataques masivos a un monton de cuentas, pero si el ataque es una cuenta en particular, el salting es irrelevante.

  - Cada c se calcula con una perturbación diferente
  - El atacante no puede reutilizar f(a) para buscar varias cuentas en paralelo

PBKDF2 (PKCS 5 v2.0)

- Idea (password base key derivation function)
  - Sea *pass* una contraseña
  - Calcular un Salt: S
  - Utilizar una función pseudoaleatoria (criptosistema simétrico o MAC):
    - u_1 = PRF(pass, S)
  - Aplicar la función iterativamente:
    - u_2 = PRF(pass, u_1)
    - …
    - u_j = PRF(pass, u_{j-1})
  - Resultado: $u_1 \oplus u_2 \oplus \ldots \oplus u_j$

La idea es que las funciones de hash son extremadamente rapidas, y nuestras claves son muy cortas. Lo que se hace es reemplazar las funciones de hash por esta construccion, que recibe dos parametros que controla la dificultad computacional para su ejecucion.

Primero se procesa el salt (estado 1), despues procesamos esa salida, etc. El resultado final es un XOR de todos los estados intermedios. La idea es que se tenga que ejecutar la funcion j veces (con j siendo un parametro). Si yo por ejemplo le digo que tiene que cifrar 25mil veces para obtener la clave, entonces los ataques offline se convierten mucho mas lentos.

PBKDF2 (PKCS 5 v2.0)

- Algoritmo:
  - K = PBKDF2(prf, pass, salt, c, len)
  - $K = T_1 \| T_2 \| \ldots \| T_n$ (n = len / |prf|)
  - $T_i = u_1 \oplus u_2 \oplus \ldots \oplus u_c$
    - u_1 = PRF(Pass, Salt || INT32_BE(i))
    - u_k = PRF(Pass, u_{k-1})

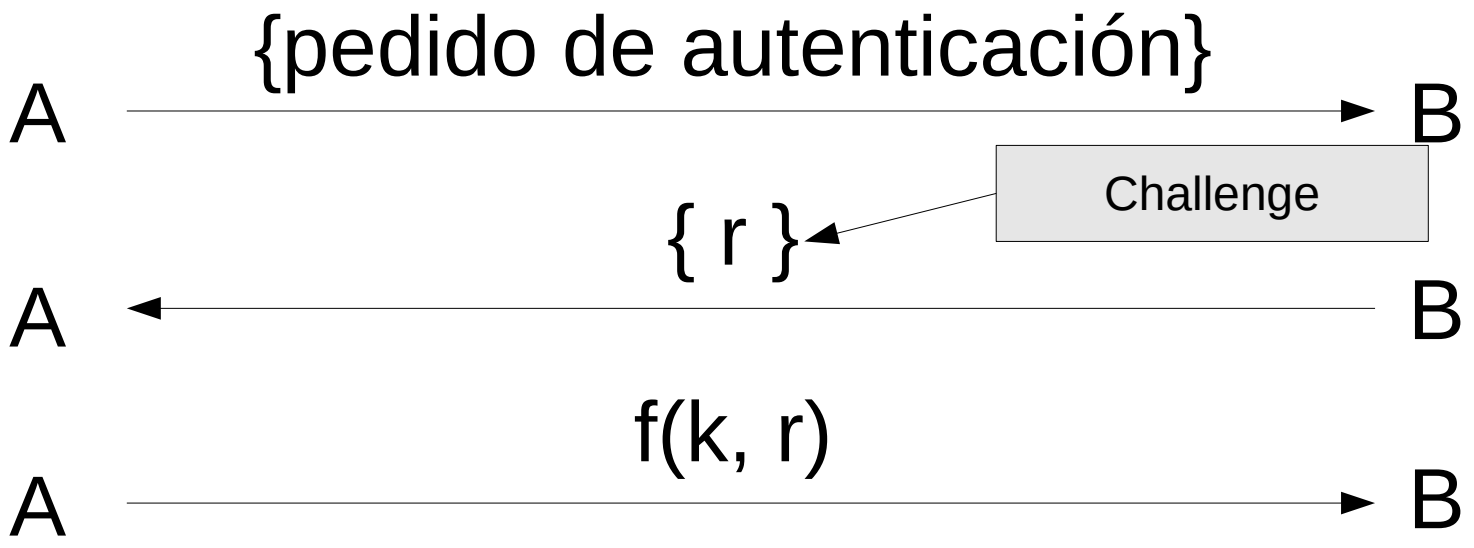
TIP: En java, JCE implementa PBKDF2:

```
PBEKeySpec spec = new PBEKeySpec(password, salt, iterations, bytes * 8);
SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
byte[] key = skf.generateSecret(spec).getEncoded();
```

Challenge - Response

- Problema: autenticar a un usuario remoto requiere que se envíen las claves
  - Requiere un canal seguro
  - Descubrir una comunicación antigua puede comprometer la clave
- Solución: no enviar la clave

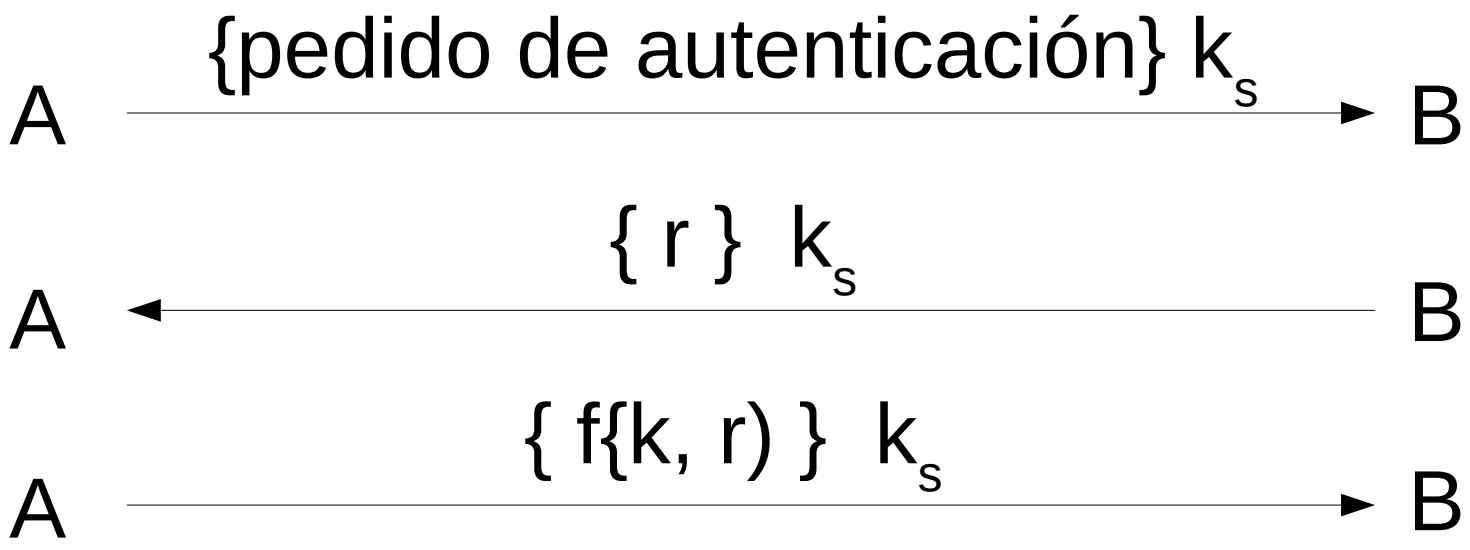
La autenticacion generalmente surge cuando todavia no tenemos un canal seguro. Los challenge-response envian al servidor un challenge y devuelven al cliente una transformacion del challenge, que utilizan al mismo tiempo autenticacion.



EKE – Encripted Key Exchange

- Objetivo: impedir que un atacante pueda adivinar claves capturadas en la red
- Mecanismo: realizar el dialogo en un canal encriptado
  - El atacante no tiene forma de saber si una clave es correcta

EKE es un RFC de un sistema Challenge-Response



OTP – Claves de uso único

- Objetivo: Generar claves que sirven una única vez

La idea es que sea computacionalmente imposible para alguien que a partir de una secuencia de claves anteriores pueda seguir generando numeros de la secuencia

¡No confundir con One Time Pad!

A y B comparten una clave K y un contador C

Concepto:

(K, C) → OTP → (otp, C')

Dos variantes estandarizadas:

- HOTP → hmac based OTP
- TOTP → time based OTP

HOTP – RFC 4226

- OTP basado en hmac

HOTP(K,C) = Truncate(HMAC-SHA1(K, C))

Truncate: extrae 32 bits de los 160 del mac

- Offset = mac[19] & 0xf ← 4 bits del ultimo byte
- bin = mac[offset … offset+3] & 0x7fffffff
- Result = bin % 10^n (n = cantidad de dígitos)

La idea es que esta funcion tiene una clave y un contador. Entonces como cambia el contenido del MAC, la salida es totalmente distinta.
Este es el algoritmo que usan los generadores de claves OTP.

HOTP – RFC 4226

- OTP basado en hmac

Tiene un problema, que es que se puede pre-generar claves sin que la persona sepa, y usarlas mas adelante sin su conocimiento.

HOTP(K,C) = Truncate(HMAC-SHA1(K, C))

- El contador se incrementa en cada uso
- Requiere sincronización (look-ahead)
- Se aconseja usar throtling

(porque un atacante puede empezar a probar todas las claves posibles)

- Genera códigos de 1 a 9 dígitos
  - Se recomienda usar al menos 6

Nada impide que alguien use el dispositivo y genere secuencias sin usarlas. Entonces el servidor va a terminar con la version vieja del contador.
La solucion a esto es que el servidor asuma que no siempre al servidor le va a llegar la clave del contador actual, sino que hay un parametro configurable para evaluar tambien los n siguientes. Esto se llama sincronizacion.

TOTP – RFC 6238

- OTP basado en timestamps

Soluciona el problema de HOTP, ya que no permite que se pre-generen claves.
Asumiendo que tanto el generador como el servidor tienen el tiempo sincronizado, entonces las claves siempre coincidiran.
Sin embargo, es dificil mantener el tiempo exacto en un dispositivo que nunca se conecta a internet, entonces se puede definir un intervalo de confianza que tome los K codigos anteriores y siguientes.
Google authenticator y otros usan TOTP.

TOTP(K) = HOTP(K, T)

```
HOTP(K,C) = Truncate(HMAC-SHA-1(K,C))
```

T= (*current unix time* – T_0) / X

Y nos quedamos con la parte entera de esto.
Cuando se pasan los X segundos, la clave ya es incorrecta.

T_0: tiempo de referencia

- Usualmente, T_0=0

X: cantidad de segundos de vida de cada codigo

- Usualmente X=30

TOTP – RFC 6238

- OTP basado en timestamps

  TOTP(K) = HOTP(K, T)

- No requiere un contador
- Se aconseja utilizar un drift-window
  - Cantidad de ticks desfasados aceptados
- Se aconseja usar throtling
- Genera códigos de 1 a 9 dígitos
  - Se recomienda usar al menos 6

Multi factor authentication

- Consiste en requerir dos factores de diferente tipo

Hoy en dia NO se recomienda que el servidor envie al cliente un OTP por SMS. Se hicieron varios estudios y no es tan dificil spoofear un SMS.

- Racional: Cada tipo de factor requiere comprometer distintos assets

- Combinaciones comunes:
  - Algo que conozco + algo que tengo
  - Algo que conozco + algo que soy
  - Algo que tengo + algo que soy
  - Algo que soy + contexto

Autenticación remota (SSO)

- Consiste en la delegación de la autenticación en un sistema externo

Hoy en dia esta muy bien visto no tener un sistema de autenticacion propio, sino utilizar otros mejores (como Auth0 o Okta)

- Implica una relación de confianza
- Productos:

La idea de estos productos es que empleados de empresas masivas puedan tener una sola cuenta para todos los servicios de la empresa

Estos 3 son propietarios, entonces no prosperaron en internet

  - Active Directory Federation Services
  - CAS
  - Siteminder
- Cada uno utiliza una tecnologia diferente
- Tecnologias abiertas
  - OpenID 1 y 2 → Obsoletos
  - OpenID Connect → Basado en Oauth2

Es el protocolo con el cual las aplicaciones permiten que la gente se registre con su cuenta de Google, etc

Lectura Recomendada

Capítulo 12-13
Computer Security Art and Science

Matt Bishop