Tema 8: Comunicación TLS

Docente : Héctor Xavier Limón Riaño
email: xavier120@hotmail.com

Certificados X.509

Introducción

- TLS es una de las piezas claves de la seguridad moderna de Internet
- HTTPS básicamente encapsula tráfico HTTP dentro de TLS
- Al igual que muchas tecnologías vistas en el curso, TLS es de gran complejidad y tiene muchos detalles de configuración

Introducción

- TLS le permite a un cliente verificar la identidad de un servidor para poder establecer una sesión segura y mantener llaves compartidas de cifrado
- Las llaves se mantienen secretas incluso si un atacante intercepta el tráfico

Certificados X.509

- Son una parte esencial en la autenticación TLS
- TLS usa PKI (public key infrastructure) para establecer identidades y asegurar las comunicaciones
- Típicamente las entidades que quieren tener una entidad bajo TLS requieren de un certificado X.509
- Estos certificados son los más ampliamente utilizados en Internet hoy en día

Campos X.509

- En temas pasados se vieron certificados, sin embargo, se utilizó una representación muy sencilla basada en diccionarios serializados mediante json
- Los campos de un certificado X.509 son una serie de pares llave/valor
- Los campos pueden tener sub-campos jerárquicos
- Básicamente un certificado X.509 tiene 3 posibles campos, los cuales pueden tener varios niveles de sub-campos
- A continuación se muestra la estructura un certificado X.509 versión 3, la versión 1 y 2 son un subconjunto de la versión 3

Campos X.509

- 1. Certificate
  - (a) Version Number
  - (b) Serial Number
  - (c) Signature Algorithm ID
  - (d) Issuer Name
  - (e) Validity Period
    - i. Not Before
    - ii. Not After
  - (f) Subject Name
  - (g) Subject Public Key Info
    - i. Public Key Algorithm
    - ii. Subject Public Key
  - (h) Issuer Unique Identifier (optional)
  - (i) Subject Unique Identifier (optional)
  - (j) Extensions (optional)
- 2. Certificate Signature Algorithm
- 3. Certificate Signature

Campos X.509

- La adición más importante de la versión 3 es el apartado de extensiones, que permite agregar más comprobaciones de seguridad
- La versión 1 y 2 se siguen utilizando y se consideran también seguros
- Recuerden que el propósito principal de un certificado es ligar la identidad de un subject a una llave pública bajo la firma de un issuer (entidad de confianza)
- Los campos que identifican al subject, la llave pública y el issuer son los más críticos, los demás campos proveen de contexto necesario para entender e interpretar los datos

Campos X.509

- Los campos que identifican los algoritmos usados son especialmente importantes
- En la práctica, una gran variedad de algoritmos criptográficos son utilizados y soportados por los sistemas criptográficos reales, los certificados deben ser flexibles para soportar todos
- Otros campos importantes son los referentes al tiempo de vida del certificado
- Así mismo, el número serial del certificado es único por issuer, e identifica al certificado, esto es útil por ejemplo para le revocación de certificados

Campos X.509

- Los campos de Issuer Name y Subject Name describen las identidades pretendidas
- Estos campos tienen sub-componentes estructurados, llamados nombres distinguidos:
  1. CN: CommonName
  2. OU: OrganizationalUnit
  3. O: Organization
  4. L: Locality
  5. S: StateOrProvinceName
  6. C: CountryName

Campos X.509

- Por ejemplo, un issuer o subject puede ser representado de esta forma:
- CN=Charlie, OU=Espionage, O=EA, L=Room 110, S=HQ, C=EA
- No todos los sub-campos son obligatorios pero CN es usualmente el campo crítico
- El CN es el identificador principal en la validación de un certificado
- En certificados modernos (versión 3) también se suele incluir un campo adicional: Subject Alternative Name
- En realidad el CN suele ser un nombre de dominio como google.com

Campos X.509

- Una vez identificados el issuer y el subject, los campos restantes son la llave pública del subject y la firma del contenido del certificado
- La firma se calcula sobre el binario del certificado bajo una codificación llamada DER (Distinguished Encoding Rules)
- La firma prueba tanto que el certificado no ha sido modificado como que fue aprobado por el issuer verdadero

Peticiones de firma de certificados

- Se requieren para crear certificados
- Una entidad interesada en tener un certificado, crea un CSR (Certificate Signing Request) y lo manda a un CA (Certificate Authority)
- Un CSR tiene casi los mismos campos que un certificado X.509 pero no contiene los campos referentes al issuer, dado que esos campos los debe establecer el CA
- Una vez el CA tiene el CSR, usa su propio certificado y llave privada asociada para generar el certificado con los campos faltantes

Peticiones de firma de certificados

- El campo issuer de un certificado debe ser igual al campo subject del certificado de quien firma
- Una vez todos los campos son rellenados el CA firma el certificado con su llave privada
- Obviamente la entidad que manda el CSR nunca manda su llave privada al CA, el CSR sólo tiene su llave pública, nadie, ni siquiera el CA debe tener la llave privada

Codificación de certificados

- Como ya se mencionó antes, los certificados X.509 son usualmente codificados en un formato binario llamado DER
- Sin embargo, la mayoría de representaciones en disco de certificados utiliza el formato PEM (visto en temas anteriores), también se utiliza PEM para llaves privadas y CSR
- PEM es un formato de texto, lo que facilita su transferencia a través de protocolos basados en texto como los propios del correo electrónico

Generación de certificados

- Un método común es utilizando openssl desde la línea de comandos
- cryptography utiliza openssl como back-end, por lo que ya deberías tenerlo instalado
- Primero se necesita generar una llave privada y luego un CSR
- Muchos de los comandos que se verán a continuación se pueden combinar, pero se presentarán por separado

Generar una llave privada

- En este caso se usa una llave RSA
- Esto lo hemos hecho muchas veces con python, pero no con openssl
- openssl genpkey -algorithm RSA -out domain_key.pem -pkeyopt rsa_keygen_bits:2048
- Se debe tener cuidado con estos comandos y los "tutoriales en internet", muchas veces se recomienda usar el comando genrsa, se debe preferir usar siempre genpkey que es más general
- También es muy importante, como ya se ha visto en el curso, que el tamaño de la llave sea de al menos 2048 bits

Generar CSR a través de una llave

- El proceso de generación de CSR extrae la llave pública de la llave privada y la agrega a la petición
- openssl req -new -key domain_key.pem -out domain_request.csr
- El comando anterior es interactivo, pero el único dato obligatorio para TLS es el referente al nombre de subject, sin embargo, algunos CA requieren de los demás campos
- El certificado se genera en formato PEM, si se desea ver todo de una forma legible para los humanos:
- openssl req -in domain_request.csr -text

Generar CSR a través de una llave

- La versión generada es la 1, y no la 3, OpenSSL siempre asigna esta versión a menos que se usen los campos de extensión
- En el certificado final, el issuer podría agregar campos de extensión para tener un certificado versión 3
- Otros campos como "Serial Number" no están presentes, ya que son agregados por el CA
- Otra cosa interesante es que el CSR está firmado, en este caso por la llave privada del subject, para demostrar que posee dicha llave
- Cualquiera podría agregar cualquier llave pública a un CSR si no se tuviera la comprobación de firma, a esta comprobación se le llama **prueba de posesión**

Firmar un CSR para producir un certificado

- Recordemos que los certificados necesitan estar firmados por el CA, para que el subject pueda establecer la validez de su identidad
- El CA es responsable de cierto nivel de verificación
- Si alguien trata de reclamar una identidad que no posee, el CA debería realizar un proceso de validación y denegar la petición
- Un atacante (y cualquiera en realidad) tiene también la posibilidad de crear un certificado auto-firmado, osea firmado con la misma llave privada del subject
- Recordemos que los certificados auto-firmados son utilizados por los certificados raíz de los CA raíz, ya que la cadena de dependencias de CAs debe terminar en algún punto
- En un certificado auto-firmado el subject e issuer son iguales

Firmar un CSR para producir un certificado

- Dado que cualquiera puede generar un certificado auto-firmado, es necesario tener además CAs raíz de confianza
- En una verificación de certificados, se necesita seguir la cadena de issuers, hasta llegar al CA raíz de confianza, si no es de confianza se debe rechazar el certificado
- Toda la seguridad del sistema (de internet en gran medida también) depende de el requisito anterior
- Los navegadores web por ejemplo, incluyen por defecto CAs raíz de confianza, de tal forma que el usuario no sea engañados para entrar, por ejemplo, a un servicio de facebook falso, claro, esto sólo funciona para https

Certificados auto-firmado

- openssl permite generar certificados auto-firmados
- Estos certificados pueden ser usados para testing, por ejemplo para probar sistemas web bajo https
- También pueden ser mal utilizados por atacantes
- openssl x509 -req -days 30 -in domain_request.csr -signkey domain_key.pem -out domain_cert.crt
- Para ver el contenido: openssl x509 -in domain_cert.crt -text

Creación de certificados

- Una vez se tiene un certificado, se puede utilizar para firmar certificados nuevos
- El siguiente ejemplo muestra cómo hacerlo, usando para ejemplificar llaves EC
- Primero se debe generar el CSR, y para eso se necesita un par de llaves:
- openssl genpkey -algorithm EC -out localhost_key.pem -pkeyopt ec_paramgen_curve:P-256
- En este caso se usó la cura P-256 de NIST, otra opción popular es 25519 pero no es soportada por cryptography

Creación de certificados

- Se genera ahora el CSR para el CN 127.0.0.1 (más adelante se verá porqué)
- openssl req -new -key localhost_key.pem -out localhost_request.csr
- Con el certificado auto-firmado generado antes, es posible generar un certificado a asociado al CSR, para esto se necesita la llave privada del CA (la llave privada asociada a la llave pública del certificado auto-firmado)
- En el ejemplo se utilizarán algunas opciones de la versión 3 para limitar el uso del certificado generado
- En este caso no se permitirá que se puedan crear certificados nuevos con el certificado generado

Creación de certificados

- La limitación introducida es para evitar abuso si es que de alguna forma un atacante logra robar o engañar a un CA para tener un certificado con su llave privada correspondiente
- Lo anterior refuerza la cadena de autoridad de los certificados, no cualquiera debería poder ser un CA
- También es útil limitar el uso que se le puede dar a la llave pública contenida en el certificado, para que por ejemplo sólo se pueda usar para firmas digitales (otro uso puede ser firmar listas de revocación de certificados CRLs

Creación de certificados

- Para agregar las opciones de la versión 3, es puede crear un archivo de texto, en este caso llamado v3.ext, que contiene dos líneas:
  - keyUsage=digitalSignature
    basicConstraints=CA:FALSE
- Para firmar el CSR:
  - openssl x509 -req -days 365 -in localhost_request.csr -CAkey domain_key.pem -CA domain_cert.crt -out localhost_cert.crt -set_serial 123456789 -extfile v3.ext

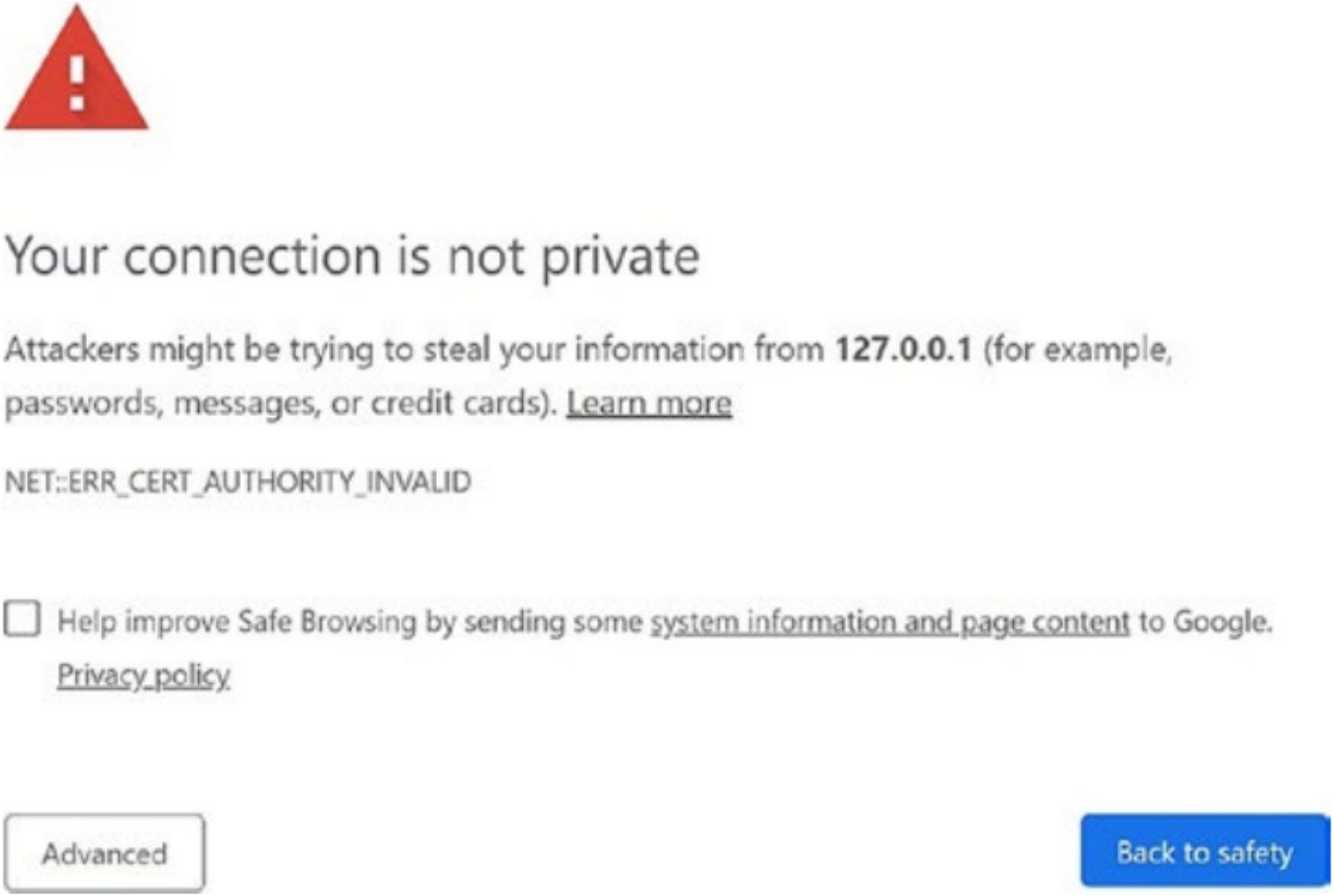
Creación de certificados

- En el caso del comando anterior, es necesario especificar el número de serie, cosa que no es obligatoria en un certificado auto-firmado
- Un CA real nunca debe reusar números de serie y debe mantener un registro de los mismos por si un certificado necesita ser revocado
- Una vez más, para poder validar la validez de la cadena de autoridad, el issuer del nuevo certificado debe ser igual al subject del certificado del CA

Creación de certificados

- El CN del nuevo certificado es 127.0.0.1, esto se hizo adrede para ver el comportamiento del navegador web bajo este dominio
- Con la opción s_server de openssl se puede levantar un servidor web con certificados asociados, en este caso la cadena de autoridad empieza con domain_cert.crt y se habilita dicha cadena con -build_chain (se manda toda la cadena de certificados al cliente)
- openssl s_server -accept 8888 -www -cert localhost_cert.pem -key localhost_key.pem -cert_chain domain_cert.crt -build_chain

Creación de certificados



Creación de certificados

- Si se activan las opciones de desarrollador, es posible ver más información del problema
- El problema es que el certificado raíz no es parte de los certificados raíz de confianza del navegador
- Ojo: esto no quiere decir que los datos en la conexión viajen sin protección criptográfica, el problema de seguridad es un posible spoofing en los certificados

Confianza en CA

- Existen formas de agregar un certificado raíz a los certificados raíz de confianza del navegador
- Si se hace esto, cualquier certificado firmado directa o indirectamente por el certificado raíz se determina válido y de confianza
- **NUNCA** debe hacerse esto, no debe ser promocionado como algo que se debe hacer en una empresa u organización, lo cual en la práctica a veces sucede
- Sobre todo no debe se debe confiar en alguien que quiera proponer este tipo de prácticas, pues puede ser parte de un ataque de ingeniería social

Confianza en CA

- Una organización sin las medidas adecuadas de seguridad no debe ser un CA, ya que la posibilidad de un robo de la llave privada del CA puede causar graves problemas
- Un atacante puede convencer a cualquiera que confíe en el CA raíz de que una página falsa es auténtica

Confianza en CA

- Pero, ¿Cómo es que los navegadores confían en alguna organización?
- La respuesta desgraciadamente es más arbitraria de lo que debería
- Un número pequeño de "autoridades" se han establecido a si mismas como autoridades raíz confiables
- Estas organizaciones y compañías tienen sus certificados raíz instalados por defecto en SOs populares y en navegadores web
- Toda la demás confianza se deriva de esas autoridades arbitrarias

Tarea

- Realiza una investigación sobre las autoridades de confianza raíz en Internet, cubriendo los siguientes puntos:
- ¿Aproximadamente cuántas son y como se distribuyen?
- ¿Cómo se puede llegar a ser una entidad de este tipo?
- Implicaciones de seguridad

Conseguir un certificado

- Necesario para el uso de HTTPS
- Si deseas tener un certificado para tu sitio en internet (prácticamente obligatorio hoy en día) es usualmente necesario probar que tienes control sobre el dominio del sitio
- Esto se logra con protocolos como ACME (ver documentación de let's encrypt)
- También es común contar con este servicio a través de un webhosting, aunque suele ser de pago

Certificados en cryptography

- Varias de las operaciones vistas con openssl pueden ser usadas en cryptography
- Se pueden generar llaves privadas, como se ha visto antes en el curso
- Se pueden crear CSRs
- Se pueden crear certificados autofirmados
- Pero una limitante importante es que no se puede firmar un CSR para generar un certificado nuevo
- Por lo que tampoco se tiene soporte a la verificación de cadenas de autoridad
- A continuación se muestra el código de estas operaciones

TLS 1.2 y 1.3

Introducción

- El objetivo de TLS es brindar seguridad de transporte (de ahí las siglas)
- La conectividad en Internet esta basada en los protocolos de la familia TCP/IP
- Los protocolos base de la familia de TCP/IP no tienen seguridad, esto es, no hay garantías de autenticación, integridad ni privacidad
- TLS está diseñado para brindar seguridad a TCP/IP

Introducción

- TLS se originó con el protocolo SSL (Secure Socker Layer) creado por Netscape a mediados de los 90's
- Sin embargo, SSL cayó en desuso debido a diversas vulnerabilidades conocidas, aunque en la actualidad a veces se utiliza como sinónimo de TLS
- La primera versión liberada de TLS fue la 2, seguida rápidamente por la 3
- La siguiente versión fue renombrada 1.0
- A partir de entonces las versiones subsecuentes arreglan problemas del protocolo y agrega o actualiza métodos criptográficos

Introducción

- La versión 1.2 de TLS ya lleva varios años en uso y aun se considera actual
- Recientemente se ha liberado la versión 1.3, pero no se describe actualmente como un remplazo de la versión 1.2, las dos versiones se consideran actuales y están en uso

TLS handshake

- TLS inicia con un handshake, el cual es en extremo crítico
- TLS tiene dos objetivos principales:
  - Establecer identidad
  - Derivar llaves de sesión para el transporte seguro
- Estos dos objetivos son logrados típicamente mediante el handshake

TLS handshake

- También es en el handshake donde se encuentran las diferencias más importantes entre la versión 1.2 y 1.3
- Primero se revisará el handshake de la versión 1.2 y luego de la versión 1.3

Hellos

- TLS 1.2 comienza con un cliente enviando un mensaje **hello** al servidor
- Dicho mensaje contiene información de configuración TLS del cliente, así como un nonce
- Uno de los detalles de configuración es la lista de **cipher suites**
- Algo que suele confundir a los que son nuevos con TLS, es que el protocolo TLS es en realidad una combinación de protocolos que trabajan en conjunto, y soporta un número de combinaciones de algoritmos y protocolos

Hellos

- El mensaje hello debe lograr que el cliente y el servidor se pongan de acuerdo en cuanto a los algoritmos y protocolos criptográficos a usar
- La lista de cipher suite, le indica al servidor todas las formas en que el cliente puede comunicarse
- El servidor selecciona una de esas formas en respuesta, si es que hay al menos una coincidencia en los cipher suites que soportan

TLS chiper suites

- Este suite típicamente incluye algoritmos para:
  - Intercambio de llaves
  - Firma digital
  - Cifrado en masa
  - Hashing

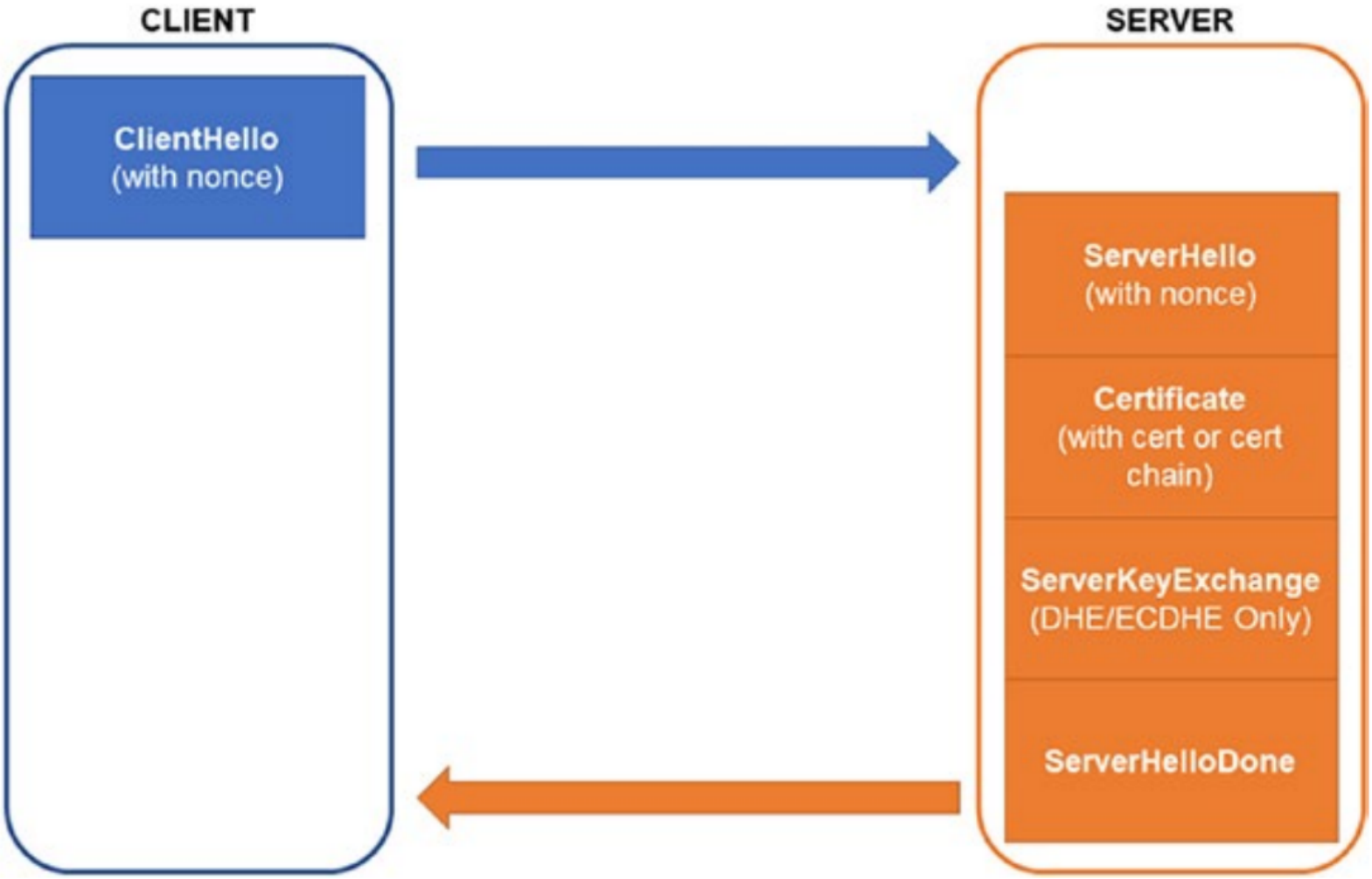
TLS chiper suites

- Por ejemplo, en TLS 1.2 un ejemplo de suite puede ser:
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
  - TLS: protocolo base del cipher suite
  - ECDHE: Diffie-Hellman con curvas elípticas y llaves efímeras
  - ECDSA: Firmas digitales con curvas elípticas, usado para autenticar ECDHE
  - AES_256_CBC: Algoritmo de cifrado en masa
  - SHA384: Algoritmo de hashing. Se utiliza en dos partes, en el algorimo de derivación de llaves, usado para producir la llave AES a partir de ECDHE, y para autenticar los mensajes de AES CBC con HMAC-SHA-384

TLS chiper suites

- En gran parte la seguridad de TLC depende de el cipher suite usado
- Aunque dos servidores utilicen por ejemplo TLS 1.2, podrían tener diferentes cipher suites
- Cuando el servidor recibe un hello de cliente, determina si utilizará alguno de los cipher suites propuestos por el cliente, si es así regresa una respuesta como sigue:
  - Hello: este incluye un nonce aleatorio del servidor
  - Certificado: el certificado o cadena de certificados del servidor
  - Intercambio de llaves: si se usa DHE o ECDHE, el servidor envía su parte pública como parte del hello, para transporte de llaves RSA no aplica este campo
  - Fin: un marcador de fin de mensaje

Hellos


CLIENT
SERVER
ClientHello
(with nonce)
ServerHello
(with nonce)
Certificate
(with cert or cert chain)
ServerKeyExchange
(DHE/ECDHE Only)
ServerHelloDone

Autenticación de cliente

- La configuración más popular de TLS actualmente autentica sólo al servidor cuando el certificado del servidor se envía con el ServerHello
- A menos que el servidor lo especifique de forma explicita, el cliente no envía un certificado para autenticarse
- Para muchas de las aplicaciones en Internet, esto es suficiente
- Cuando ambas partes se autentican con un certificado, se le llama MTLS (Mutual TLS)

Autenticación de cliente

- En Internet los clientes suelen ser libres de entrar a los servicios que quieran
- Los servidores deben asegurarle a los clientes que son el servicio verdadero que espera el cliente
- La identidad del cliente es algo menos claro
- La identidad de un servidor está usualmente ligada a un dominio o dirección IP
- Lo anterior no es posible hacerlo con un cliente, además el cliente es una entidad más dinámica que puede utilizar una variedad de dispositivos y medios de comunicación

Autenticación de cliente

- Para circunstancias donde se necesita autenticar al cliente, como en una transacción bancaria, lo que importa es la identidad del usuario, más que la identidad de la máquina
- Para este fin se utilizan mecanismos como los clásicos usuario-contraseña u otros a nivel aplicación
- Después de haber autenticado al servidor y creado una sesión segura con una llave compartida, el usuario puede enviar de forma segura sus datos de identidad

Derivación de llaves de sesión

- Como ya se vio en el tema 6, existen dos formas de conseguir llaves simétricas a través de métodos asimétricos: transporte de llaves (con RSA) y acuerdo de llaves (con DH)
- En el handshake de TLS 1.2 se necesita que el cliente y el servidor compartan una llave simétrica
- En realidad, en el handshake no se obtienen la llave final, sino un PMS (pre-master-secret), junto con otros datos no secretos que se usarán para generar el **master secret**, como son los nonce del hello del cliente y servidor

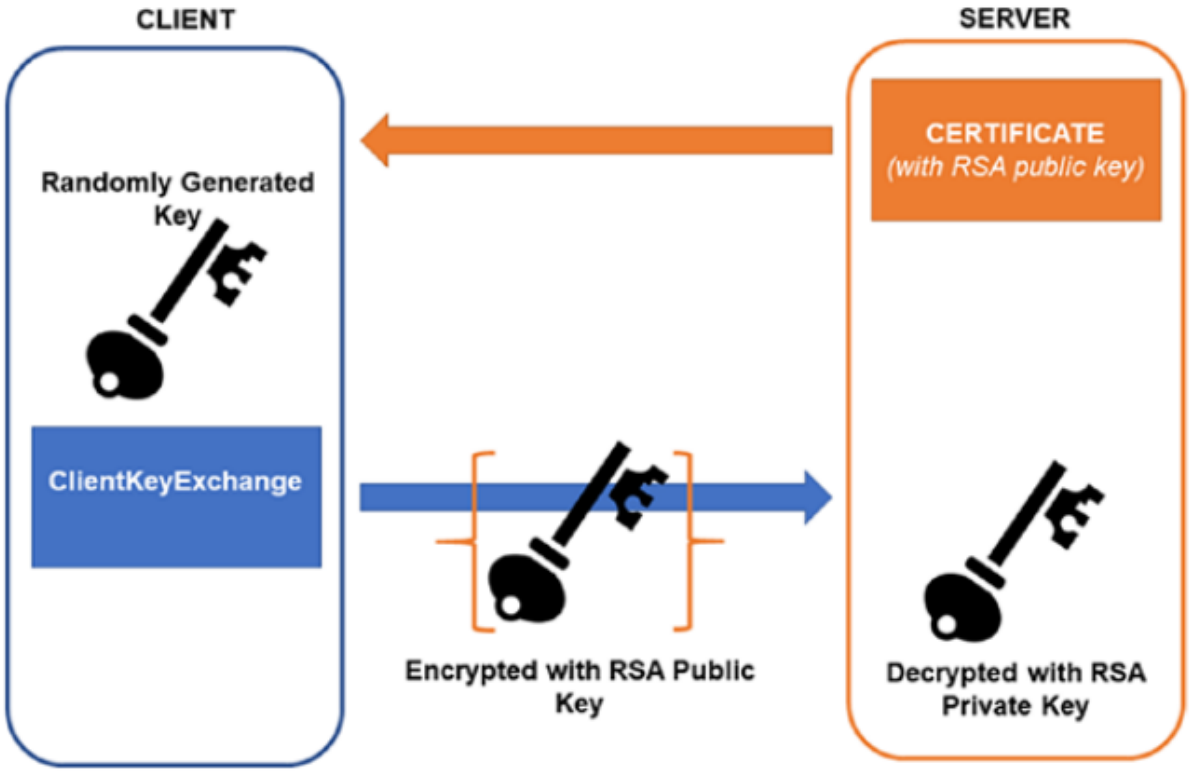
Derivación de llaves de sesión

- El master secret se utiliza para generar todas las llaves de sesión necesarias para la comunicación en masa
- TLS 1.2 a través de sus varios cipher suits, brinda tanto transporte de llaves como acuerdo de llaves para el PMS
- Los cipher suites que empiezan con TLS_RSA utilizan transporte de llaves RSA, por ejemplo TLS_RSA_WITH_3DES_EDE_CBC_SHA (este cipher suite es inseguro)

Derivación de llaves de sesión

- Notar que con transporte de llaves RSA no se necesita algoritmo de firma digital, pues no hay información que se necesite autenticar
- En este tipo de transporte el cliente recibe el certificado del servidor que fue enviado con el hello del servidor, extrae la llave pública y cifra el PMS con dicha llave para enviarla al servidor, sólo el servidor podrá descifrar el PMS (por esto no se requiere de firmas)
- De esta forma tanto cliente como servidor comparten el mismo PMS

Derivación de llaves de sesión



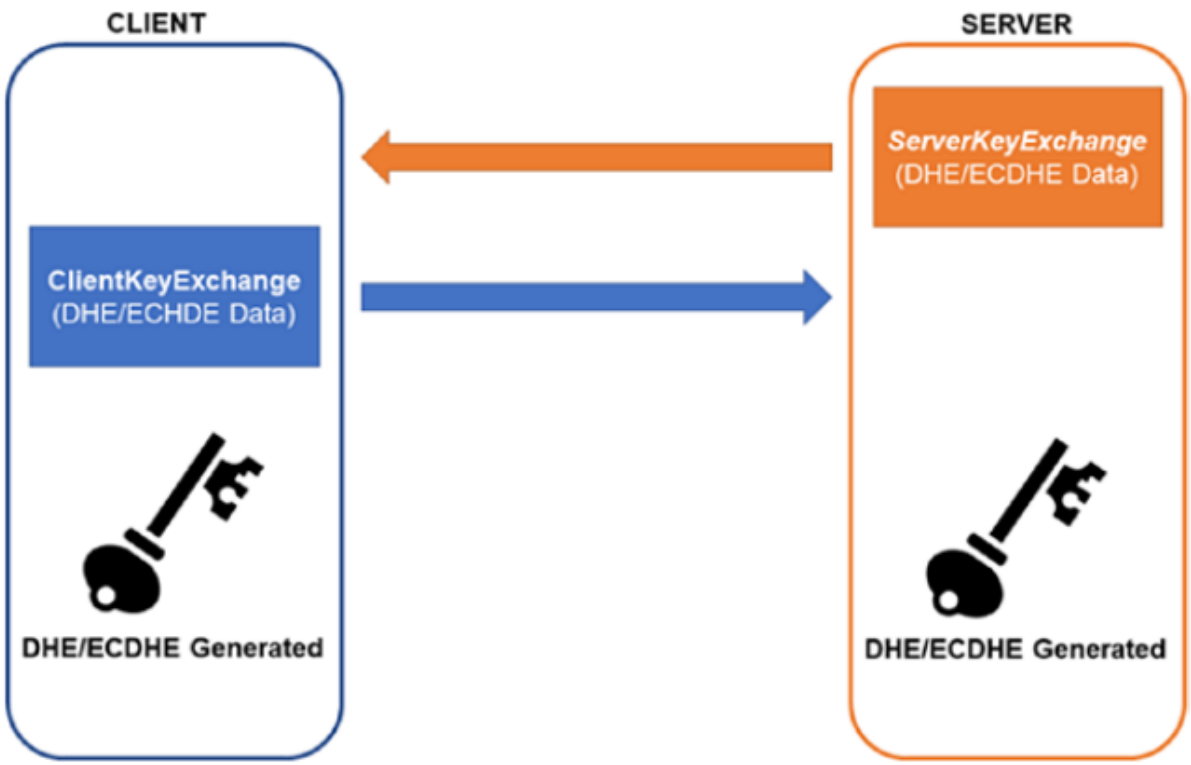
Derivación de llaves de sesión

- DHE y ECDHE se comportan diferente
- El PMS no se transmite, sino que se deriva de forma simultanea
- Las llaves pública y privada de DH/ECDH se pueden generar de forma efímera para usarse una sola vez

Derivación de llaves de sesión

- Recordemos que la llave pública DH/ECHD debe ser autenticada (junto con información de parámetros, como el tipo de curva), al menos del lado del servidor
- Para esto se usa una llave privada del servidor de larga duración, ya sea RSA o ECDSA
- Cuando el cliente recibe los datos DH/ECHD firmados, usa la llave pública en el certificado del servidor para verificar la firma

Derivación de llaves de sesión



Derivación de llaves de sesión

- La seguridad de los dos métodos vistos varía mucho
- El transporte de llaves RSA no brinda confidencialidad hacía adelante, así mismo la generación del PMS lo realiza el cliente y el servidor debe confiar en que el cliente no reusa el mismo PMS o lo genera de forma pobre
- Además, el transporte RSA podría usar un padding PKCS 1.5 que contiene vulnerabilidades
- Los problemas anteriores no los tiene DH/ECHD y debe ser la opción elegida en todos los casos (excepto si no está disponible)

Cambio a cifrado

- Una vez se finalizó el intercambio de llaves simétricas ya no es necesario ni recomendable mandar información plana
- Toda la información a partir de este punto debe estar cifrada y autenticada
- Para señalar esto, el cliente envía al servidor un mensaje llamado ChangeCipherSpec
- Esto indica que todos los mensajes subsecuentes utilizarán el cipher suite negociado
- El servidor contesta con su propio ChangeCipherSpec

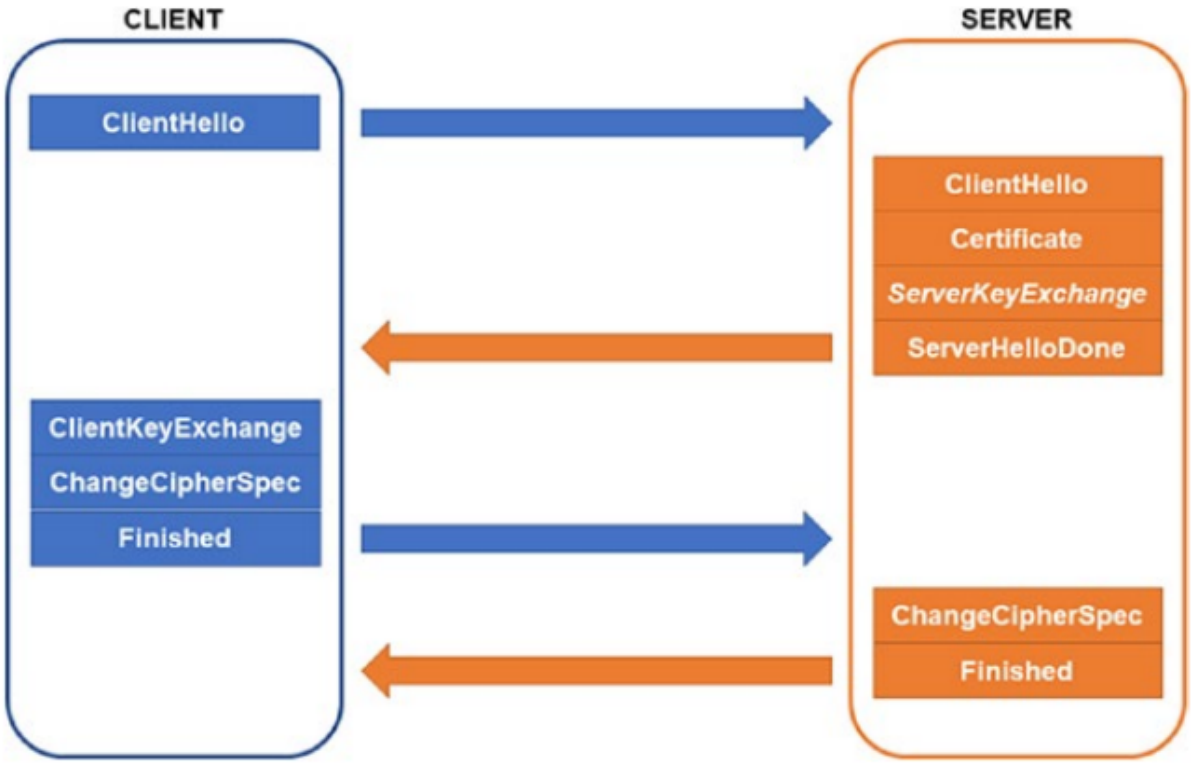
Cambio a cifrado

- Cada parte envía entonces un mensaje (ya cifrado) Finished para completar el handshake
- El mensaje Finished contiene un hash de todos los mensajes del handshake enviados
- El propósito del hash es asegurarse de que ningún atacante alteró alguna parte del handshake

Cambio a cifrado

- Por ejemplo, un atacante podría haber alterado el mensaje de hello del cliente para dejar sólo cipher suits débiles
- Tanto cliente como servidor llevan un registro de todos los mensajes enviados, así la otra parte puede comprobar integridad con el hash de los mensajes recibidos contra el hash del mensaje Finished
- De no pasarse la comprobación se considera que el canal está comprometido y se cierra de inmediato

Cambio a cifrado



Derivación de llaves y transferencia en masa

- Después del handshake, el cliente ya verificó la identidad del servidor y ambas partes comparten un PMS
- Con el PMS y datos adicionales (como los nonce de cliente y servidor) derivan llaves simétricas
- Estas llaves simétricas tienen el objetivo de brindar confidencialidad, integridad y autenticación en cada mensaje

Derivación de llaves y transferencia en masa

- En TLS 1.2 el PMS se expande al master secret usando lo que se conoce como una función pseudo-aleatoria (PRF)
- La expansión de una llave es simplemente tomar un secreto y expandirlo a más bytes
- Por defecto el PRF utiliza un mecanismo basado en HMAC-SHA256 para expandir bytes de entrada a un tamaño arbitrario
- El cipher suite puede especificar otro método

Derivación de llaves y transferencia en masa

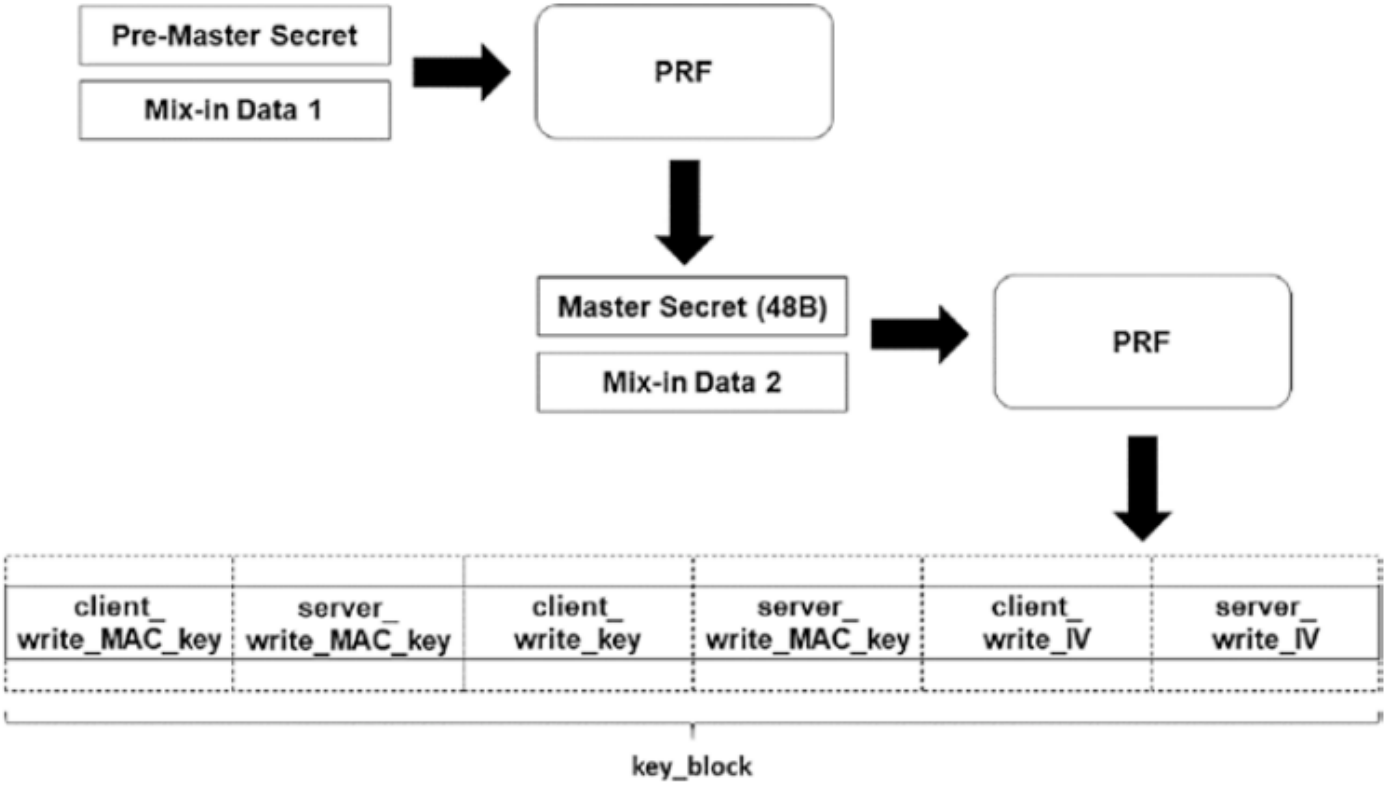
- En el caso de TLS el PMS se expande (cual sea su longitud) a 48 bytes para generar el master secret
- El master secret es a su vez expandido a tantos bytes sea necesario para obtener todas las llaves de sesión e IVs requeridos por el cipher suite
- Con todas las expansiones del master secret se obtiene el key_block que varía en tamaño de acuerdo al cipher suite

Derivación de llaves y transferencia en masa

El key_block tiene a lo mucho 6 parámetros:

- client write MAC key
- server write MAC key
- client write key
- server write key
- client write IV
- server write IV

Derivación de llaves y transferencia en masa



Derivación de llaves y transferencia en masa

- No todos los cipher suits usan todos los campos, por ejemplo, si se utiliza AES-GCM los campos de MAC no son necesarios
- Sin embargo, todos los chiper suits proveen integridad, confidencialidad y autenticación
- En el pasado se han encontrado vulnerabilidades en algunos cipher suites, por ejemplo en TLS 1.0 AES-CBC tenía una vulnerabilidad de ataque de padding oráculo derivado de aplicar MAC y luego cifrar (recuerden que lo correcto es cifrar luego MAC)

Derivación de llaves y transferencia en masa

- Una vez se finalizó el handshake, se puede iniciar la transferencia en masa mediante métodos simétricos
- TLS segmenta las transmisiones en paquetes, recordando que esto es importante para AES-GCM y para la aplicación de MACs ya que no debería esperarse al final de una transmisión larga para verificar si hay algún problema de integridad
- Cada paquete de TLS está cifrado y autenticado (ya sea con MAC o AE)
- Estos paquetes están estructurados en un formato llamado TLSCipherText
- Estos paquetes son de alrededor de 16 kilobytes
- La estructura se especifica como un struct de C

Derivación de llaves y transferencia en masa

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        select (SecurityParameters.cipher_type) {
            case stream: GenericStreamCipher;
            case block: GenericBlockCipher;
            case aead: GenericAEADCipher;
        } fragment;
} TLSCiphertext;
```

Derivación de llaves y transferencia en masa

- En general los campos de la estructura anterior son fáciles de interpretar, con excepción del select
- En dicho apartado se listan 3 posibles opciones que representan 3 posibles cifradores: GenericStreamCipher, GenericBlockStream, y GenericAEADCipher
- Los primeros dos tipos incluyen una MAC
- Cuando TLS manda datos, se construye un flujo binario con las piezas de la estructura en el orden mostrado
- A continuación se muestra la estructura de cada subtipo de cifrador

Derivación de llaves y transferencia en masa

```
stream-ciphered struct {
      opaque content[TLSCompressed.length];
      opaque MAC[SecurityParameters.mac_length];
} GenericStreamCipher ;

struct {
      opaque IV[SecurityParameters.record_iv_length];
      block-ciphered struct {
          opaque content[TLSCompressed.length];
          opaque MAC[SecurityParameters.mac_length];
          uint8 padding[GenericBlockCipher.padding_length];
          uint8 padding_length;
      };
} GenericBlockCipher;
```

Derivación de llaves y transferencia en masa

```
struct {
    opaque nonce_explicit[SecurityParameters.record_iv_length];
    aead-ciphered struct {
        opaque content[TLSCompressed.length];
    };
} GenericAEADCipher ;
```

Derivación de llaves y transferencia en masa

- En el caso de AEAD, tanto el content type, version y length del mensaje están autenticados
- Cada paquete tiene su propia MAC o tag AEAD asociada, por lo que se pueden manejar transmisiones largas revisando posibles problemas sobre la marcha

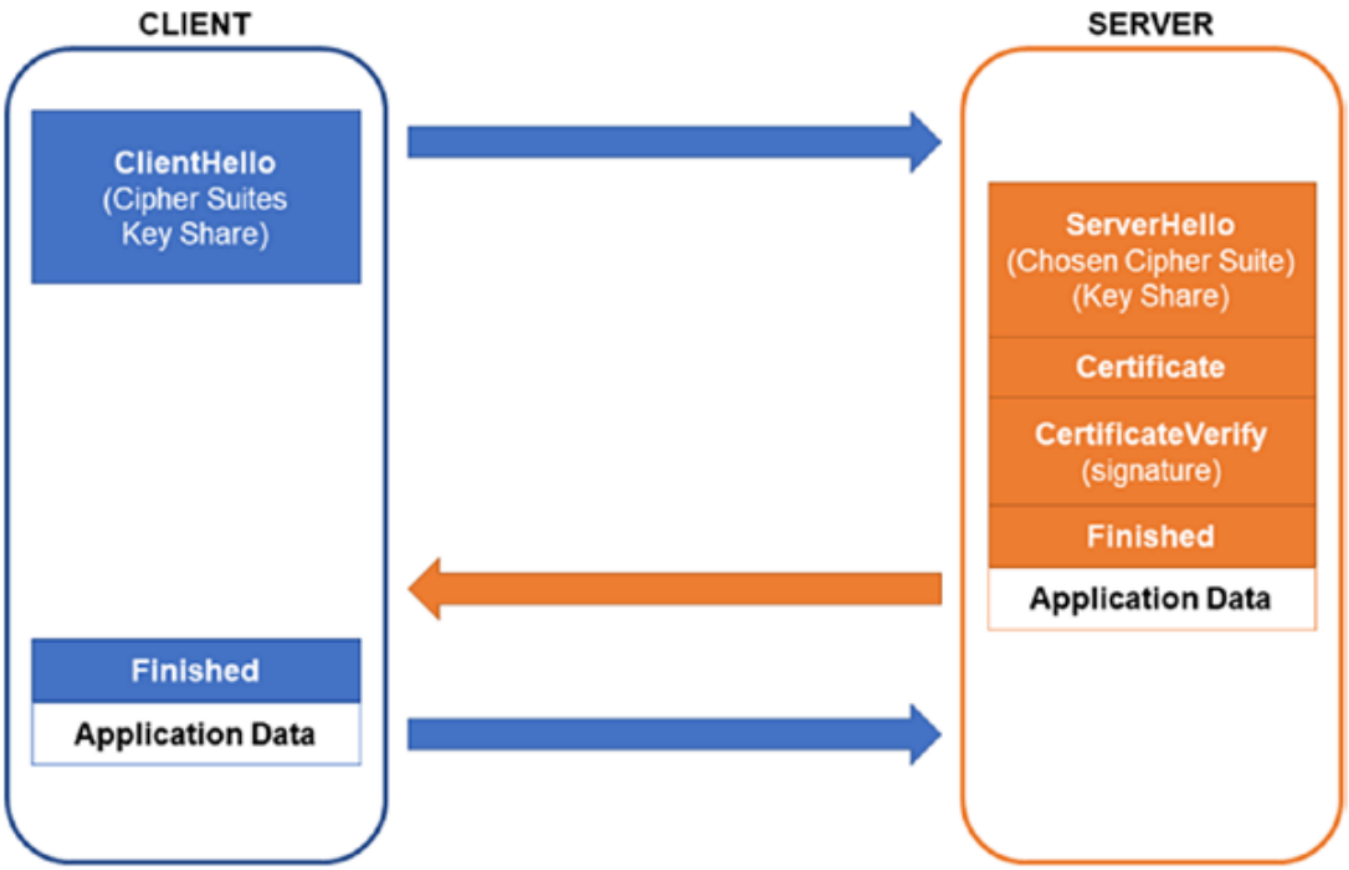
TLS 1.3

- Esta versión hace grandes cambios al proceso de handshake
- TLS 1.3 desecha la mayoría de cipher suits presentes en TLS 1.2, permitiendo sólo 5, todos basados en AEAD:
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_GCM_SHA256
  - TLS_AES_128_CCM_8_SHA256
  - TLS_AES_128_CCM_SHA256

TLS 1.3

- Básicamente se da soporte a AES-GCM, AES-CCM y a ChaCha20-Poly1305
- Con estas reducciones se vuelve más difícil equivocarse en la configuración permitiendo chipers débiles
- También se vuelve más difícil para los atacantes engañar para que se use un cipher suite débil (ya no hay ninguno débil)
- RSA ya no se puede utilizar como transporte de llaves, sólo DHE y ECDHE
- Un cambio importante en el handshake es que ahora todo el proceso se hace en una sola ronda de mensajes

TLS 1.3


CLIENT
SERVER
ClientHello
(Cipher Suites
Key Share)
ServerHello
(Chosen Cipher Suite)
(Key Share)
Certificate
CertificateVerify
(signature)
Finished
Application Data
Finished
Application Data

TLS 1.3

- Esta simplificación del handshake ayuda mucho en protocolos sin estado, como HTTP, ya que en una sola petición-respuesta se puede realizar toda la transmisión segura
- En TLS 1.2 es más complejo y costoso lograr lo mismo ya que se tiene que crear primero un túnel TLS para cada comunicación entre cliente y servidor, para que luego pueda realizarse la petición-respuesta
- TLS 1.3 elimina gran parte de la latencia de transmisión con respecto a TLS 1.2, lo cual representa una gran diferencia en comunicaciones web

TLS 1.3

- Al no haber transporte de llaves RSA, la confidencialidad hacia adelante se vuelve obligatoria
- Limitar a sólo AEAD es también una mejora importante
- **CUIDADO**: existe una "variante" de TLS 1.3 conocida como eTLS, la cual no es oficial y remueve mucha de la seguridad importante de TLS, como la confidencialidad hacia adelante
- Supuestamente esta variante es más eficiente y fácil de usar, pero debilita la seguridad, nunca debes de utilizarla, ni deberías utilizar navegadores web que le dan soporte, en el futuro a esta tecnología pretenden llamarla Enterprise Transport Security (ETS)

Verificación de certificados y confianza

Introducción

Algunos posibles ataques a TLS:

- Ataques de padding oráculo contra cifrado RSA en algunas versiones e implementaciones
- Ataques de padding oráculo contra AES-CBC en algunas versiones e implementaciones
- Intentar convencer al cliente de que utilice cipher suits débiles

Introducción

- Todos los anteriores son ataques que se han realizado en el pasado, pero uno que no se menciona es tratar de explotar la revisión de certificados
- Cuando un cliente recibe un certificado, debe asegurarse de que es válido y de confianza
- El certificado del cliente suele depender de una cadena de CAs
- El proceso de verificación consiste en seguir el camino de certificados, hasta llegar a un CA raíz de confianza

Proceso de verificación de certificados

De forma más detallada, el proceso seguido para verificar un certificado es:

- En el certificado del cliente, el nombre de subject debe corresponder al nombre del host (dominio) en la URL:
  - El nombre de host puede corresponderse con el CN del subject
  - El nombre de host puede corresponderse con el nombre alternativo del subject (en una extensión V3)
- Ninguno de los certificados en la cadena puede estar expirado
- Ninguno de los certificados en la cadena puede estar revocado
- El issuer de un certificado debe ser el subject del certificado en el siguiente elemento de la cadena, hasta llegar a la raíz
- Limitaciones de certificado (como KeyUsage y BasicConstraints) son reforzados
- Políticas adicionales son reforzadas, como longitud máxima de la cadena, nombrado, etc.

Proceso de verificación de certificados

- El proceso de verificación es complejo y tiene muchos elementos
- Cualquier error en uno de esos elementos puede generar un exploit
- Muchos de los exploits en TLS tienen poco que ver con el protocolo en si, aprovechan en cambio errores de programadores o de usuarios

Proceso de verificación de certificados

- Gran parte de la seguridad de TLS depende de que los certificados sean entregados a entidades autorizadas
- Los atacantes pueden intentar robar llaves privadas para suplantar identidades y que un CA les de un certificado correspondiente, pueden tratar de convencer al cliente que su certificado es de confianza, o pueden convencer al cliente de que instale un nuevo CA raíz
- Existen muchas posibilidades para que los atacantes hagan mal uso de los certificados, por eso es importante tener mecanismos de defensa, como la revocación de certificados

Revocación de certificados

- En el tema 5 se mencionó que los certificados tienen una debilidad importante en cuanto a la revocación de certificados
- La revocación es un problema complejo en muchos niveles
- Existen dos aproximaciones generales para revocar certificados: Listas de revocación de certificados (CRLs) y el protocolo OCSP (Online Certificate Status Protocol)

Revocación de certificados: CRLs

- Registro estático de certificados que han sido revocados (mediante su identificador)
- Los CRLs son a menudo específicos para un CA y son firmados por el CA
- Es importante que el CA lleve un registro de los certificados generados y que no repita identificadores
- Los CRLs se suelen publicar de forma periódica (una vez al día por ejemplo)
- Los sistemas de verificación de certificados, como los usados en TLS, deben mantener una lista de todos los certificados revocados, para poder detectar e invalidar certificados revocados

Revocación de certificados: OCSP

- Se utiliza para revisar la validez de los certificados a partir de su número de serie
- A diferencia de CRLs, el protocolo se utiliza con un servidor online en tiempo real, al cual se le puede pedir que revise la revocación en el momento que se hace la verificación
- Usualmente el CA tiene el control de su propio servidor OCSP para la verificación de sus certificados generados

Revocación de certificados: OCSP

- Obviamente OCSP tiene información más actualizada que los CRLs
- Pero introduce latencia adicional en el handshake de TLS
- Además introduce otro problema: ¿Qué debe hacer el cliente si el servidor OCSP no responde?
- La mayoría de navegadores asumen que el certificado no ha sido revocado
- Lo anterior es una posible explotación, que aproveche un ataque DOS al servidor OCSP

Revocación de certificados

- Por los problemas antes mencionados, CRLs y OCSP se consideran en la actualidad obsoletos
- Algunos navegadores como Chrome o Firefox ni siquiera utilizan o dejan utilizar estos mecanismos, sino que mantienen sus propias listas
- La revocación de certificados es todavía un problema abierto
- En la actualidad se exploran nuevos métodos como OCSP stapling

Raíces no confiables

- Un problema fundamental de TLS es que la seguridad depende de tener entidades de confianza
- Pero como dijo el poeta Romano Juvenal: "¿Quis custodiet ipsos custodes?" (¿Quién custodia a los custodios?)
- Un problema grave es que si se compromete la llave privada de un CA, un atacante puede generar certificados nuevos para cualquier dominio, haciendo que los clientes confíen en dicho certificado

Raíces no confiables

- El anterior no es un problema teórico, ha sucedido en el pasado
- Por ejemplo, en 2011 atacantes lograron robar una llave privada del CA DigiNotar, generando certificados con subdominios aparentes de Google, Yahoo, Wordpress, Mozzila y TOR
- El CA tuvo que ser removido de los CAs de confianza de todos los navegadores y OS, lo que provocó también que la organización cerrara
- En realidad nunca se supo el alcance completo del ataque

Raíces no confiables

- No sólo un CA puede ser atacado y comprometido, sino que algunos CAs llevan a cabo prácticas dudosas
- Por ejemplo, la empresa Trustico que se dedica a revender certificados (genera los certificados a nombre de otros) le pidió a un CA DigiCert que revocara más de 20 mil certificados, esto debido a que se perdió la confianza en dichos certificados
- Lo que pasó es que Trustico guardaba una copia de todas las llaves privadas de los certificados que revendía, lo cual de por si es una práctica deshonesta que puede provocar que un empleado robe las llaves o que se comprometan de otra forma
- Además Trustico en algún momento envío las llaves privadas de los 20 mil certificados a través de email, razón por la cual se pidió la revocación

Raíces no confiables

- Un cliente nunca debe confiarle a un CA su llave privada, de hacerlo siempre corre un riesgo
- Toda la seguridad criptográfica depende de que los secretos se mantengan secretos
- El problema de los certificados fraudulentos o usados de forma incorrecta es muy común y serio
- Existen algunos mecanismos para mitigar estos problemas, los cuales se discuten a continuación

Pinning de certificados

- La idea básica es que el cliente, antes de recibir el certificado, ya tiene una expectativa de lo que el certificado debe tener
- Cuando se recibe el certificado se compara contra la versión esperada (versión pinned)
- Y se aplican políticas cuando se encuentran discrepancias, bajo el supuesto de que esas discrepancias representan un posible ataque
- Esta aproximación también se le suele llamar HTTP Public Key Pinning (HPKP)

Pinning de certificados

- En algún punto la tecnología fue apoyada por Google, pero en la actualidad se considera insuficiente
- Sin embargo, se sigue utilizando en ámbitos como las aplicaciones móviles, donde el certificado del autor de la app se agrega al empaquetado de la propia app, este certificado pinned, siempre se compara con el certificado recibido en una comunicación TLS que hace la app
- Si se necesita un nuevo certificado, se saca una nueva versión de la app
- Google y Firefox hacen algo similar en sus navegadores, por ejemplo, Google notó el problema con DigiNotar mediante este mecanismo

Pinning de certificados

- HPKP utiliza el principio TOFU (trust on first use), donde básicamente la primera vez que visitas un sitio el cliente hace un pinning del certificado asociado y lo mantiene un tiempo
- HPKP es interesante pero introduce muchos problemas adicionales y puede tener explotaciones, por lo que ha ido quedando en desuso

Transparencia de certificados

- Este es un mecanismo que está ganando fuerza en los últimos tiempos
- La idea es similar a blockchain, donde cada vez que se genera un certificado se agrega también a un log público
- El log lo mantiene un tercero, pero como con blockchain, no es necesario confiar en el tercero porque todo lo generado esta fuertemente atado por verificaciones de hash

Transparencia de certificados

- El propósito del log es transparencia: los CAs pueden ser auditados por los certificados que producen
- La meta es tener todos los certificados generados de forma pública, para que puedan ser criptográficamente verificados
- Es probable que pronto los navegadores sólo confíen en certificados que aparecen en el log

Transparencia de certificados

- Esto limitará mucho a los atacantes, pues certificados falsos tendrán que hacerse públicos, lo que les permitirá a los CAs detectarlos y revocarlos rápidamente de forma automática
- Incluso si un atacante logra salirse con la suya, se tendrá un registro público de todo lo que ha hecho con los certificados, pudiendo establecer el alcance y daño del ataque
- En el caso de DigiNotar, el hecho de no conocer cuántos certificados falsos se habían generado, provocó que se tuviera que eliminar DigiNotar por completo
- Esta tecnología es todavía nueva y está evolucionando, no cuenta aun por ejemplo con mecanismos de revocación, pero es muy probable que en el futuro sea el estándar a seguir

Ataques conocidos a TLS

Introducción

- Un atacante tiene varias opciones además de los que el manejo de certificados presentan
- En la actualidad existen otros tipos de ataques con los que hay que tener cuidad
- A continuación se mencionarán varios ataques y la forma de prevenirlos

POODLE

- Padding Oracle On Downgraded Legacy Encryption
- TLS 1.0 puede ser explotado cuando se usa CBC ya sea con AES o con un cifrador más antiguo llamado DES
- TLS 1.1 y 1.2 supuestamente arregla el problema cambiando la forma en que se hace el padding
- Sin embargo, muchos servidores pueden ser atacados para re-negociar el protocolo a 1.0

POODLE

- Además, se descubrió que algunas implementaciones de TLS 1.1 y 1.2 usaban el mismo padding de TLS 1.0
- Las defensas incluyen:
  - Deshabilitar TLS 1.0 y 1.1
  - Verificar servicio TLS 1.2 mediante una herramienta de auditoría (scanner de vulnerabilidad por ejemplo)
  - Migrar a TLS 1.3 y sólo permitir esta versión ya que no soporta CBC

FREAK y Logjam

- Logjam depende de forzar al servidor a bajar de versión, en este caso para tener versiones débiles de cipher suites
- En los 90's el gobierno de EU tenía una política de no permitir que se exportara criptografía fuerte a otros países, el gobierno trataba a los algoritmos fuertes como si fueran armas
- La seguridad informática todavía lleva cicatrices debido a esa política

FREAK y Logjam

- A partir de la política surgieron cipher suites específicos para TLS, conocidos como algoritmos EXPORT, los cuales eran muy débiles
- En Logjam el hello del cliente es interceptado, cambiando la lista de cipher suits a versiones EXPORT de Diffie-Hellman, con llaves fácilmente rompibles
- Con el hash del mensaje Finished debería notarse esta manipulación, sin embargo, como el Finished va cifrado de forma débil, un atacante puede interceptarlo y cambiarlo

FREAK y Logjam

- Freak es muy parecido a Logjam, pero en vez de atacar DH ataca el transporte de llaves RSA a partir de versiones EXPORT
- Defensas contra ambos ataques incluyen:
  - Deshabilitar cipher suits débiles, especialmente versiones EXPORT
  - Usar clientes que se reusen a aceptar parámetros débiles de DH/ECDH o RSA

Sweet32

- Está pensado específicamente para cifradores de bloque que usan bloques de 64 bits
- En la mayoría de instalaciones de TLS 1.2 sólo hay un cifrador con esas características: 3DES
- 3DES utiliza por debajo DES, un cifrador simétrico con debilidades
- El tamaño de bloque impacta en que tantos datos se pueden cifrar con una misma llave antes de que se empiecen a repetir parámetros en diferentes bloques
- Para bloques de 64 bits el límite es 32 GB, lo cual es fácilmente generado por computadoras modernas

Sweet32

- Muchas implementaciones de TLS no refuerzan un límite máximo de datos cifrados con la misma llave
- Sweet32 aprovecha lo anterior para mandar suficientes datos para forzar la aparición de colisiones y recuperar datos
- Las defensas incluyen:
  - Deshabilitar cualquier cipher suite que utilice 3DES, o cualquier otro que use bloques de 64 bits

ROBOT

- Return Of Bleichenbacher's Oracle Threat
- Este ataque aprovecha los defectos del padding PKCS 1.5 para RSA, para realizar un ataque de padding oráculo
- Todas las versiones de TLS hasta la 1.2 tienen esta versión de padding, a pesar de que los defectos se descubrieron desde 1999
- Los diseñadores de TLS decidieron dejar este padding por razones de compatibilidad, y trataron de reforzarlo de otras formas

ROBOT

- En teoría lo que basta es que el servidor le deje de servir de oráculo al atacante, para no revelar información sobre el padding
- Sin embargo, estas medidas no siempre funcionan, y se descubrieron otras formas para que el servidor dé información de padding
- Defensas incluyen:
  - Deshabilitar todos los cipher suites que usen RSA para intercambio de llaves

CRIME, TIME y BREACH

- TSL 1.2 permite comprimir datos antes de cifrar, esto fue deshabilitado en 1.3
- El problema con la compresión es que revela información a los atacantes, la cual puede ser utilizada para recuperar información cifrada
- CRIME (Compression Ratio Info-leak Made Easy), se demostró primero en 2012
- La compresión sólo funciona si hay datos repetidos, si tu sólo tienes un texto cifrado que contiene un texto comprimido y si puedes insertar o parcialmente insertar mensajes para ver el efecto de la compresión y notas que la compresión a mejorado con la inserción del mensaje, eso quiere decir que agregaste algo repetido en el mensaje original

- Esta información puede ser utilizada para recuperar un número pequeño de bytes
- Cualquier pérdida de confidencialidad, por pequeña que sea, es inaceptable
- Por ejemplo, los bytes pueden representar una cookie web con información de autenticación
- CRIME fue seguido por TIME, el cual es más efectivo
- TIME inspiró a BREACH, que es un ataque diferente pero que también ataca a la compresión para revelar información
- Defensas incluyen:
  - Deshabilitar compresión

Heartbleed

- Esta no es propiamente una vulnerabilidad de TLS, sino un bug en la implementación de OpenSSL
- Este bug se daba en una extensión que permitía detectar conexiones muertas a partir de heartbeats
- Un heartbeat es un mensaje que se manda de forma periódica para que un servicio sepa que un cliente sigue activo
- Una petición típica de heartbeat incluye algunos datos que se mandan de ida y vuelta (eco) con su correspondiente longitud

Heartbleed

- Con el bug no se revisaba que la información devuelta fuera correcta
- Si la longitud del mensaje excedía el tamaño real, la implementación leía contenido de la memoria
- Aunque no hubiera garantías de que se podía leer de la memoria, podría incluir por ejemplo una llave cifrado u otros secretos
- Defensas incluyen:
  - Mantener las bibliotecas y aplicaciones actualizadas

Sockets TLS con python

- A continuación se muestra el uso de la biblioteca ssl de python, la cual tiene soporte para TLS a través de OpenSSL
- Sólo se presentará la creación de sockets cliente y servidor, los procesos de generación de certificados pueden realizarse con OpenSSL directamente