Network Security: TLS/SSL

Tuomas Aura, Microsoft Research, UK

Outline

1. More building blocks
2. Authenticated key exchange
3. Diffie-Hellman
4. Key exchange using public-key encryption
5. TLS/SSL
6. TLS handshake
7. TLS record protocol
8. TLS trust model

2

More building blocks

Sequence numbers

- Sequence numbers in messages allow the recipient to check for lost, reordered or duplicated messages
- Sequence numbers in authenticated messages allow the recipient to detect intentional message deletion, reordering and duplication
- Notation: i, SN, seq num

4

Nonces

- Timestamps require accurate clocks and don't prevent rapid replays:
  $A \rightarrow B$: T_A, M, $S_A(T_A, M)$ // S_A("Transfer £100")
- Checking freshness with B's nonce:
  $B \rightarrow A$: N_B
  $A \rightarrow B$: N_B, M, $S_A(N_B, M)$
- Alice's nonce is a bit string selected by Alice, which is never reused and (usually) unpredictable
- Nonce implementations:
  - 128-bit random number (unlikely to repeat)
  - timestamp concatenated with a random number (protects against errors in RNG initialization and/or clock
  - hash of a timestamp and random number
- Problematic nonces: sequence number, deterministic PRNG output, timestamp
- Nonce notations: N_A, R_A

5

Message notation

- The goal of TLS and many other security protocols is to protect opaque upper-layer data
  - Notation: M, data, payload
- Messages may be composed by concatenating byte or bit strings
  - Notation: $M_1 \| M_2 \| M_3$ or M_1, M_2, M_3
- Messages must have unambiguous decoding and meaning:
  - E.g. "Send £100 to account 2322323." vs. "100"‖"7244244" vs. "1007"‖"244244" vs. "£100 a/c 2322323"
  - Simple concatenation of fixed-length bit fields
  - Self-delimiting, such as ASN.1 DER and other type-length-value (TLV) encodings

6

Authenticated key exchange

Basic goals for key exchange

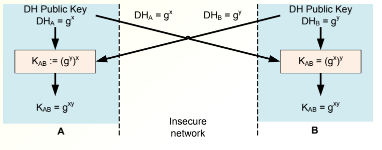
- Create a good session key:
  - Secret i.e. known only to the intended participants
  - Fresh i.e. never used before
- Authentication:
  - Mutual i.e. bidirectional authentication: each party knows who it shares the key with (sometimes also unidirectional authentication)
- Optional properties:
  - Entity authentication: each participant know that the other is online and participated in the protocol
  - Key confirmation: each participant knows that the other knows the session key
  - Protection of long-term secrets: long term secrets such as private keys or shared master keys are not compromised even if session keys are
  - Forward secrecy (or perfect forward secrecy): compromise of current secrets should not compromise past session keys
  - Contributory: both parties contribute to the session key; neither can decide the session-key value alone
  - Non-repudiation: a party cannot deny taking part in the protocol
  - Integrity of version and algorithm negotiation: increase difficulty of fall-back attacks

Advanced goals

- Identity protection:
  - Passive or active attackers cannot learn the identities of the protocol participants
- Denial-of-service resistance:
  - The protocol cannot be used to exhaust memory or CPU of the participants
  - The protocol cannot be used to flood third parties with data
  - It is not easy to prevent the participants from completing the protocol

Diffie-Hellman

Diffie-Hellman

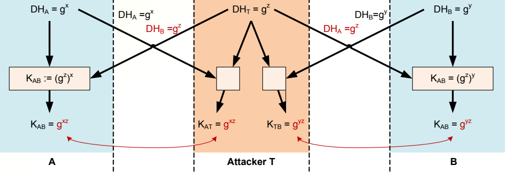


- Key exchange based on commutative public-key operations
  - Each party has its own secret exponent x, y
  - Each party sends or publishes its own public DH key
  - Both compute the same shared secret or key material
- Public-key notations: g^x, g^y, DH_A, DH_B, DH-A, DH-B, PK_B, PK_B
- Shared secret notations: g^{xy}, SK, K_{AB}, K_{DH}
- Needs authentication!

11

Man-in-the-middle (MitM) attack

- Diffie-Hellman is secure against passive attackers
  - Not possible to discover the shared secret by sniffing the network
- Vulnerable to an active attack:
  - To A, the attacker pretends to be A
  - To B, the attacker pretends to be B



12

Alice and Bob

- Common informal notation for cryptographic protocols
- Alice A, Bob B, Carol C, Trent T,
  Client C, Server S,
  Initiator I, Responder R, etc.
- Diffie-Hellman:
  $A \rightarrow B$: A, g^x
  $B \rightarrow A$: B, g^y
  SK = $h(g^{xy})$
- Man-in-the-middle attack:
  $A \rightarrow T(B)$: A, g^x // Trent intercepts the message
  $T(A) \rightarrow B$: A, g^z // Trent spoofs the message
  $B \rightarrow T(A)$: B, g^y // Trent intercepts the message
  $T(B) \rightarrow A$: B, g^z // Trent spoofs the message

13

Authenticating Diffie-Hellman

- Certified Diffie-Hellman public keys:
  $A \rightarrow B$: A, g^x, $Cert_A$
  $B \rightarrow A$: B, g^y, $Cert_B$
  - $Cert_A$ is a standard public-key certificate, e.g. X.509, where the subject key is A's Diffie-Hellman public key
- Signed Diffie-Hellman (more common):
  $A \rightarrow B$: A, g^x, $S_A(A, g^x)$, $Cert_A$
  $B \rightarrow A$: B, g^y, $S_B(B, g^y)$, $Cert_B$
  - $Cert_A$ is a standard public-key certificate, e.g. X.509, where the subject key is A's public signature key
- MitM attack prevented
- Still missing freshness!

14

Diffie-Hellman with nonces (1)

- Signed Diffie-Hellman with nonces:
  $A \rightarrow B$: A, N_A, g^x, $S_A(A, N_A, g^x)$, $Cert_A$
  $B \rightarrow A$: B, N_B, g^y, $S_B(B, N_B, g^y)$, $Cert_B$
  SK = $h(N_A, N_B, g^{xy})$

Secret session key?
Fresh session key?
Mutual authentication?
Entity authentication?
Key confirmation?
Protection of long-term secrets?
Forward secrecy?
Contributory?
Non-repudiation?
Integrity of negotiation?
DoS protection?
Identity protection?

15

Diffie-Hellman with nonces (2)

- Signed Diffie-Hellman with nonces:
  $A \rightarrow B$: A, N_A, g^x, $S_A(A, N_A, g^x)$, $Cert_A$
  $B \rightarrow A$: B, N_B, g^y, $S_B(B, N_B, g^y)$, $Cert_B$
  SK = $h(N_A, N_B, g^{xy})$
- Properties:
  - Secret, fresh session key
    → Both know that SK cannot be known to anyone other than A and B
  - Mutual authentication
  - Protection of long-term secrets
  - Contributory
  - Non-repudiation or participation, but not of completion
- Missing properties:
  - No entity authentication or key confirmation:
    → Neither party knows that the other really took part in the protocol or that the other computer the same key
  - Not clear from the above spec whether it gives forward secrecy
  - No negotiation, so can't say anything about that
  - No DoS or identity protection

16

Variation with key confirmation

- Signed Diffie-Hellman with nonces:
  $A \rightarrow B$: A, B, N_A, g^x, $S_A(A, B, N_A, g^x)$, $Cert_A$
  $B \rightarrow A$: A, B, N_A, N_B, g^x, g^y, $S_B(A, B, N_A, N_B, g^x, g^y)$, $Cert_B$
  $A \rightarrow B$: A, B, MAC_{SK}(A, B, "Done.")
  SK = $h(N_A, N_B, g^{xy})$
- Real protocols are more complex and have even more variations
  - Version and algorithm negotiation
  - DoS protection
  - Identity protection

Secret session key?
Fresh session key?
Mutual authentication?
Entity authentication?
Key confirmation?
Protection of long-term secrets?
Forward secrecy?
Contributory?
Non-repudiation?
Integrity of negotiation?
DoS protection?
Identity protection?

17

Ephemeral Diffie-Hellman

- Diffie-Hellman exponents can be reused
  - Nonces guarantee a fresh session key
- Forward secrecy is achieved by using ephemeral Diffie-Hellman exponents
  - Pick a fresh exponent and forget previous ones
- Cost of forward secrecy:
  - Random-number generation for new exponents
  - Computation of new public keys
- No changes to the protocol messages → each party can choose how often it wants to replace its Diffie-Hellman keys
  - Exponents typically replaced every day or every hour, regardless of how many exchanges performed

18

Key exchange using public-key encryption

PK encryption of session key

- Public-key encryption of the session key:
  - A → B: A, PK_A
  - B → A: B, E_A(SK)
  - SK = session key
  - E_A(…) = encryption with A's public key
- Man-in-the-middle attack:
  - A → T(B): A, PK_A // Trent intercepts the message
  - T(A) → B: A, PK_T // Trent spoofs the message
  - B → T(A): B, E_T(SK) // Trent intercepts the message
  - T(B) → A: B, E_A(SK) // Trent spoofs the message

20

Authenticated key exchange

- Public-key encryption of the session key:
  - A → B: A,B, N_A, $Cert_A$
  - B → A: A,B, N_A,N_B, E_A(SK), S_B(A,B, N_A,N_B, E_A(SK)), $Cert_B$
  - A → B: A,B, MAC_{SK}(A,B, "Done.")
  - SK = session key
  - $Cert_A$ = certificate for A's public e
  - E_A(…) = encryption with A's publi
  - $Cert_B$ = certificate for B's public s
  - S_B(…) = B's signature

Secret session key?
Fresh session key?
Mutual authentication?
Entity authentication?
Key confirmation?
Protection of long-term secrets?
Forward secrecy?
Contributory?
Non-repudiation?
Integrity of negotiation?
DoS protection?
Identity protection?

TLS/SSL

TLS/SSL

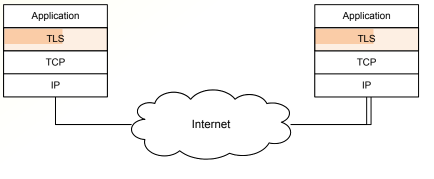
- Originally Secure Sockets Layer (SSLv3) by Netscape in 1995
- Originally intended to facilitate web commerce:
  - Fast adoption because built into web browsers
  - Encrypt credit card numbers and passwords on the web
- Early attitudes, especially in the IETF:
  - IPSec will eventually replace TLS/SSL
  - TLS/SSL is bad because it slows the adoption of IPSec

  Now SSL/TLS is the dominant encryption standard
- Standardized as Transport-Layer Security (TLSv1) by IETF [RFC2246]
  - Minimal changes to SSLv3 implementations but not interoperable

23

TLS/SSL architecture (1)

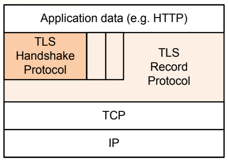
- Encryption and authentication layer added to the protocol stack between TCP and applications.
- End-to-end security between client and server, usually web browser and server.



24

TLS/SSL architecture (2)

- TLS Handshake Protocol — authenticated key exchange
- TLS Record Protocol — block data delivery
- Minor protocols:
  - Alert — error messages
  - Change Cipher Spec — turn on encryption or update keys



- General architecture of security protocols: authenticated key exchange + session protocol

25

Cryptography in TLS

- Many key-exchange mechanisms and algorithm suites defined
- Most widely deployed cipher suite, default in TLS 1.1: TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - RSA = handshake: RSA-based key exchange
  - Key-exchange uses its own MAC composed of SHA-1 and MD5
  - 3DES_EDE_CBC = data encryption with 3DES block cipher in EDE mode and CBC
  - SHA = data authentication with HMAC-SHA-1
- Default cipher suite in TLS 1.0: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
  - DHE_DSS = handshake: ephemeral Diffie-Hellman key exchange authenticated with DSS* signatures
- Examples of other cipher suites:
  TLS_NULL_WITH_NULL_NULL
  TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
  TLS_DHE_DSS_WITH_AES_256_CBC_SHA [RFC3269]

TLS handshake

27

TLS handshake protocol

- Runs on top of TLS record protocol
- Negotiates protocol version and cipher suite (i.e. cryptographic algorithms)
  - Protocol versions: 3.0 = SSLv3, 3.1 = TLSv1
  - Cipher suite e.g. DHE_RSA_WITH_3DES_EDE_CBC_SHA
- Performs authenticated key exchange
  - Often only server authenticated

28

TLS Handshake

1. Negotiation
2. Authentication
3. Key exchange
4. Start session

Protocol versions, client nonce, cipher suites

Client
ClientHello →

Server
Protocol version, server nonce, cipher suite

ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
ServerHelloDone ←

Server certificate

Server Diffie-Hellman key and signature

Optional, client typically unauthenticated.

Certificate*
ClientKeyExchange
CertificateVerify*
ChangeCipherSpec
Finished →

Client Diffie-Hellman key

Signature

ChangeCipherSpec
Finished ←

Encrypted and MAC'ed session data

Application data ↔ Application data

TLS handshake

1. C → S: ClientHello
2. S → C: ServerHello, Certificate, [ServerKeyExchange], [CertificateRequest], ServerHelloDone
3. C → S: [Certificate], ClientKeyExchange, [CertificateVerify], ChangeCipherSpec, Finished
4. S → C: ChangeCipherSpec, Finished

- [Brackets] indicate fields for bidirectional authentication

30

TLS_DHE_DSS handshake

1. C → S: Versions, N_C, SessionId, CipherSuites
2. S → C: Version, N_S, SessionId, CipherSuite
   CertChain$_S$
   g, n, g^y, Sign$_S$(N_C, N_S, g, n, g^y)
   [Root CAs]
3. C → S: [CertChain$_C$]
   g^x
   [Sign$_C$(all previous messages including N_C, N_S, g, n, g^y, g^x)]
   ChangeCipherSpec
   MAC$_{SK}$ ("client finished", all previous messages)
4. S → C: ChangeCipherSpec
   MAC$_{SK}$("server finished", all previous messages)

1. Negotiation
2. Ephemeral Diffie-Hellman
3. Nonces
4. Signature
5. Certificates
6. Key confirmation and negotiation integrity

- pre_master_secret = g^{xy}
- master_secret = SK = h(g^{xy}, "master secret", N_C, N_S)
- Finished messages are already protected by the new session keys

31

TLS_DHE_DSS handshake

Secret session key?
Fresh session key?
Mutual authentication?
Entity authentication?
Key confirmation?
Protection of long-term secrets?
Forward secrecy?
Contributory?
Non-repudiation?
Integrity of negotiation?
DoS protection?
Identity protection?

1. C → S: Versions, N_C, SessionId, CipherSuites
2. S → C: Version, N_S, SessionId, CipherSuite
   CertChain$_S$
   g, n, g^y, Sign$_S$(N_C, N_S, g, n, g^y)
   [Root CAs]
3. C → S: [CertChain$_C$]
   g^x
   [Sign$_C$(all previous messages including N_C, N_S, g, n, g^y, g^x)]
   ChangeCipherSpec
   MAC$_{SK}$ ("client finished", all previous messages)
4. S → C: ChangeCipherSpec
   MAC$_{SK}$("server finished", all previous messages)

- pre_master_secret = g^{xy}
- master_secret = SK = h(pre_master_secret, "master secret", N_C, N_S)
- *Finished* messages are already protected by the new session keys

32

TLS_RSA handshake

1. C → S: Versions, N_C, SessionId, CipherSuites
2. S → C: Version, N_S, SessionId, CipherSuite
   CertChain$_S$
   [Root CAs]
3. C → S: [CertChain$_C$]
   E$_S$(pre_master_secret),
   [Sign$_C$(all previous messages including N_C, N_S, E$_S$(…))]
   ChangeCipherSpec
   MAC$_{SK}$ ("client finished", all previous messages)
4. S → C: ChangeCipherSpec
   MAC$_{SK}$("server finished", all previous messages)

1. Negotiation
2. RSA
3. Nonces
4. Signature
5. Certificates
6. Key confirmation and negotiation integrity

- E$_S$ = RSA encryption (PKCS #1 v1.5) with S's public key from CertChain$_S$
- pre_master_secret = random number chosen by C
- master_secret = SK = h(pre_master_secret, "master secret", N_C, N_S)
- *Finished* messages are already protected by the new session keys

33

TLS_RSA handshake

Secret session key?
Fresh session key?
Mutual authentication?
Entity authentication?
Key confirmation?
Protection of long-term secrets?
Forward secrecy?
Contributory?
Non-repudiation?
Integrity of negotiation?
DoS protection?
Identity protection?

1. C → S: Versions, N_C, SessionId, CipherSuites
2. S → C: Version, N_S, SessionId, CipherSuite
   CertChain$_S$
   [Root CAs]
3. C → S: [CertChain$_C$]
   E$_S$(pre_master_secret),
   [Sign$_C$(all previous messages including N_C, N_S, g, n, g^y, g^x)]
   ChangeCipherSpec
   MAC$_{SK}$ ("client finished", all previous messages)
4. S → C: ChangeCipherSpec
   MAC$_{SK}$("server finished", all previous messages)

- E$_S$ = RSA encryption (PKCS #1 v1.5) with S's public key from CertChain$_S$
- pre_master_secret = random number chosen by C
- master_secret = SK = h(g^{xy}, "master secret", N_C, N_S)
- *Finished* messages are already protected by the new session keys

34

Nonces in TLS

- Client and Server Random are nonces
- Concatenation of a real-time clock value and random number:

```
struct {
        uint32 gmt_unix_time;
        opaque random_bytes[28];
} Random;
```

35

Session vs. connection

- TLS session can span multiple connections
  - Client and server cache the session state and key
  - Client sends the SessionId of a cached session in Client Hello, otherwise zero
  - Server responds with the same SessionId if found in cache, otherwise with a fresh value
- New master_secret calculated with new nonces for each connection
- Change of IP address does not invalidate cached sessions

36

TLS record protocol

TLS record protocol

- For write (sending):
  1. Take arbitrary-length data blocks from upper layer
  2. Fragment to blocks of ≤ 4096 bytes
  3. Compress the data (optional)
  4. Apply a MAC
  5. Encrypt
  6. Add fragment header (SN, content type, length)
  7. Transmit over TCP server port 443 (https)
- For read (receiving):
  - Receive, decrypt, verify MAC, decompress, defragment, deliver to upper layer

38

TLS record protocol - abstraction

- Abstract view: E_{K1} (data, $HMAC_{K2}$(SN, content type, length, data))
- Different encryption and MAC keys in each direction
  - All keys and IVs are derived from the master_secret
- TLS record protocol uses 64-bit unsigned integers starting from zero for each connection
  - TLS works over TCP, which is reliable and preserves order. Thus, sequence numbers must be received in exact order

39

TSL trust model

40

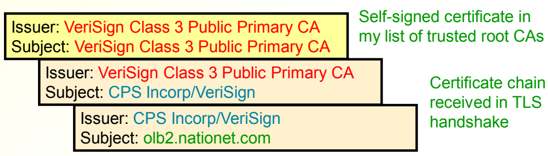
Typical TLS Trust Model

- Trust root: web browsers come with a pre-configured list of root CAs (e.g. Verisign)
  - Users can add or remove root CAs — which do you accept?
- Root-CA public keys are stored in self-signed certificates
  - Not really a certificate; just a way of storing the CA public keys
- Users usually do not have client certificates
  - Businesses pay a top-level CA to issue a server certificate. Client users do not want to pay
  - Typically, password authentication of the user over the server-authenticated HTTPS channel (web form or HTTP basic access authentication)

41

TLS Certificate Example

- Example of a TLS certificate chain: Nationwide (a building society in the UK)



- But how do I know that olb2.nationet.com is the Nationwide online banking site?

42

TLS Applications

- Originally designed for web browsing
- New applications:
  - Any TCP connection can be protected with TLS
  - The SOAP remote procedure call (SOAP RPC) protocol uses HTTP as its transport protocol. Thus, SOAP can be protected with TLS
  - TLS-based VPNs
  - EAP-TLS authentication and key exchange in wireless LANs and elsewhere
- The web-browser trust model is usually not suitable for the new applications!

Exercises

- Password-based protocols are generally vulnerable to offline guessing attacks (apart from a new class of special protocols). Is TLS server authentication + HTTP digest vulnerable to offline guessing?
- Use a network sniffer (e.g. Netmon, Ethereal) to look at TLS/SSL handshakes. Can you spot a full handshake and session reuse? Can you see the lack of identity protection?
- What factors mitigate the lack of identity protection in TLS?
- In what ways do web browsers and bank web sites try to ensure that the user knows they are connected to their bank with HTTPS, not to a phishing site and not with unprotected HTTP?
- Why is the front page of a web site often insecure (HTTP) even if the password entry and/or later data access are secure (HTTPS)? What security problems can this cause?
- How to set up multiple secure (HTTPS) web sites behind a NAT or on a virtual server that has only one IP address? (Try this in practice.)
- How would you modify the TLS handshake to improve identity protection? Remember that SessionId is also a traceable identifier.