

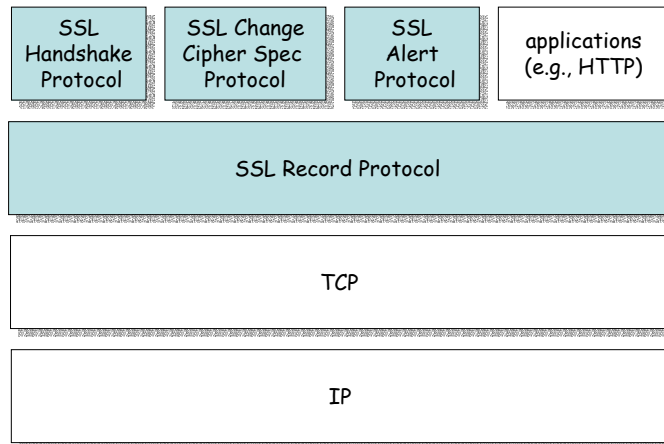
## SSL - Secure Socket Layer

- architecture and services
- sessions and connections
- SSL Record Protocol
- SSL Handshake Protocol
- key exchange alternatives
- analysis of the SSL Record and Handshake Protocols
- SSL vs. TLS

## What is SSL?

- SSL - Secure Socket Layer
- it provides a secure transport connection between applications (e.g., a web server and a browser)
- SSL was developed by Netscape
- SSL version 3.0 has been implemented in many web browsers (e.g., Netscape Navigator and MS Internet Explorer) and web servers and widely used on the Internet
- SSL v3.0 was specified in an Internet Draft (1996)
- it evolved into RFC 2246 and was renamed to TLS (Transport Layer Security)
- TLS can be viewed as SSL v3.1

## SSL architecture



© Levente Buttyán

3

## SSL components

- SSL Handshake Protocol
  - negotiation of security algorithms and parameters
  - key exchange
  - server authentication and optionally client authentication
- SSL Record Protocol
  - fragmentation
  - compression
  - message authentication and integrity protection
  - encryption
- SSL Alert Protocol
  - error messages (fatal alerts and warnings)
- SSL Change Cipher Spec Protocol
  - a single message that indicates the end of the SSL handshake

© Levente Buttyán

4

## Sessions and connections

- an SSL session is an association between a client and a server
- sessions are stateful; the session state includes security algorithms and parameters
- a session may include multiple secure connections between the same client and server
- connections of the same session share the session state
- sessions are used to avoid expensive negotiation of new security parameters for each connection
- there may be multiple simultaneous sessions between the same two parties, but this feature is not used in practice

## Session and connection states

- session state
  - session identifier
    - arbitrary byte sequence chosen by the server to identify the session
  - peer certificate
    - X509 certificate of the peer
    - may be null
  - compression method
  - cipher spec
    - bulk data encryption algorithm (e.g., null, DES, 3DES, ...)
    - MAC algorithm (e.g., MD5, SHA-1)
    - cryptographic attributes (e.g., hash size, IV size, ...)
  - master secret
    - 48-byte secret shared between the client and the server
  - is resumable
    - a flag indicating whether the session can be used to initiate new connections
  - connection states

## Session and connection states cont'd

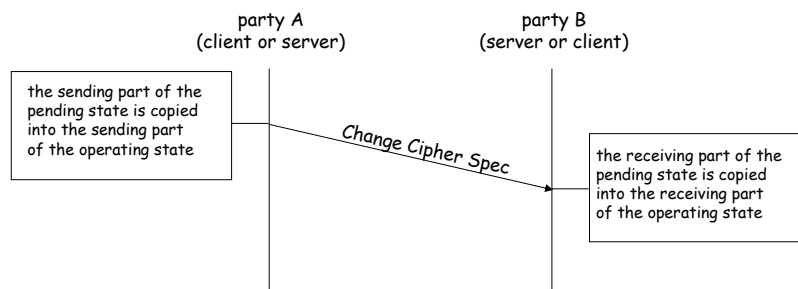
- connection state
  - server and client random
    - random byte sequences chosen by the server and the client for every connection
  - server write MAC secret
    - secret key used in MAC operations on data sent by the server
  - client write MAC secret
    - secret key used in MAC operations on data sent by the client
  - server write key
    - secret encryption key for data encrypted by the server
  - client write key
    - secret encryption key for data encrypted by the client
  - initialization vectors
    - an IV is maintained for each encryption key if CBC mode is used
    - initialized by the SSL Handshake Protocol
    - final ciphertext block from each record is used as IV with the following record
  - sending and receiving sequence numbers
    - sequence numbers are 64 bits long
    - reset to zero after each Change Cipher Spec message

© Levente Buttyán

7

## State changes

- operating state
  - currently used state
- pending state
  - state to be used
  - built using the current state
- operating state  $\leftarrow$  pending state
  - at the transmission and reception of a Change Cipher Spec message

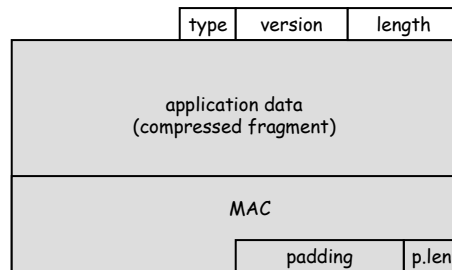


© Levente Buttyán

8

## SSL Record Protocol – processing overview

- fragmentation
  - compression
  - MAC computation
  - padding
  - encryption
- SSL Record Protocol message:



## Header

- type
  - the higher level protocol used to process the enclosed fragment
  - possible types:
    - change\_cipher\_spec
    - alert
    - handshake
    - application\_data
- version
  - SSL version, currently 3.0
- length
  - length (in bytes) of the enclosed fragment or compressed fragment
  - max value is  $2^{14} + 2048$

## MAC

$MAC = \text{hash}( MAC\_wr\_sec \mid pad\_2 \mid \text{hash}( MAC\_wr\_sec \mid pad\_1 \mid seq\_num \mid type \mid length \mid frag ))$

- similar to HMAC but the pads are concatenated
- supported hash functions:
  - MD5
  - SHA-1
- pad\_1 is 0x36 repeated 48 times (MD5) or 40 times (SHA-1)
- pad\_2 is 0x5C repeated 48 times (MD5) or 40 times (SHA-1)

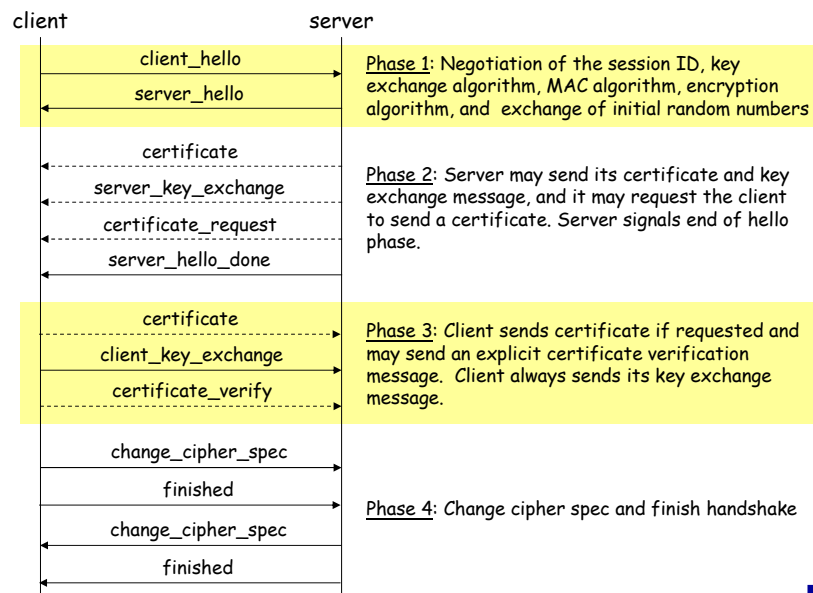
## Encryption

- supported algorithms
  - block ciphers (in CBC mode)
    - RC2\_40
    - DES\_40
    - DES\_56
    - 3DES\_168
    - IDEA\_128
    - Fortezza\_80
  - stream ciphers
    - RC4\_40
    - RC4\_128
- if a block cipher is used, than padding is applied
  - last byte of the padding is the padding length

## SSL Alert Protocol

- each alert message consists of 2 fields (bytes)
- first field (byte): "warning" or "fatal"
- second field (byte):
  - fatal
    - unexpected\_message
    - bad\_record\_MAC
    - decompression\_failure
    - handshake\_failure
    - illegal\_parameter
  - warning
    - close\_notify
    - no\_certificate
    - bad\_certificate
    - unsupported\_certificate
    - certificate\_revoked
    - certificate\_expired
    - certificate\_unknown
- in case of a fatal alert
  - connection is terminated
  - session ID is invalidated → no new connection can be established within this session

## SSL Handshake Protocol - overview



## Hello messages

- client\_hello
  - client\_version
    - the highest version supported by the client
  - client\_random
    - current time (4 bytes) + pseudo random bytes (28 bytes)
  - session\_id
    - empty if the client wants to create a new session, or
    - the session ID of an old session within which the client wants to create the new connection
  - cipher\_suites
    - list of cryptographic options supported by the client ordered by preference
    - a cipher suite contains the specification of the
      - key exchange method, the encryption and the MAC algorithm
      - the algorithms implicitly specify the hash\_size, IV\_size, and key\_material parameters (part of the Cipher Spec of the session state)
    - example: SSL\_RSA\_with\_3DES\_EDE\_CBC\_SHA
  - compression\_methods
    - list of compression methods supported by the client

© Levente Buttyán

15

## Hello messages cont'd

- server\_hello
  - server\_version
    - $\min(\text{highest version supported by client, highest version supported by server})$
  - server\_random
    - current time + random bytes
    - random bytes must be independent of the client random
  - session\_id
    - session ID chosen by the server
    - if the client wanted to resume an old session:
      - server checks if the session is resumable
      - if so, it responds with the session ID and the parties proceed to the finished messages
    - if the client wanted a new session
      - server generates a new session ID
  - cipher\_suite
    - single cipher suite selected by the server from the list given by the client
  - compression\_method
    - single compression method selected by the server

© Levente Buttyán

16



## Supported key exchange methods

- RSA based (SSL\_RSA\_with...)
  - the secret key (pre-master secret) is encrypted with the server's public RSA key
  - the server's public key is made available to the client during the exchange
- fixed Diffie-Hellman (SSL\_DH\_RSA\_with... or SSL\_DH\_DSS\_with...)
  - the server has fix DH parameters contained in a certificate signed by a CA
  - the client may have fix DH parameters certified by a CA or it may send an unauthenticated one-time DH public value in the client\_key\_exchange message
- ephemeral Diffie-Hellman (SSL\_DHE\_RSA\_with... or SSL\_DHE\_DSS\_with...)
  - both the server and the client generate one-time DH parameters
  - the server signs its DH parameters with its private RSA or DSS key
  - the client may authenticate itself (if requested by the server) by signing the hash of the handshake messages with its private RSA or DSS key
- anonymous Diffie-Hellman (SSL\_DH\_anon\_with...)
  - both the server and the client generate one-time DH parameters
  - they send their parameters to the peer without authentication
- Fortezza
  - Fortezza proprietary key exchange scheme

© Levente Buttyán

17

## Server certificate and key exchange msgs

- certificate
  - required for every key exchange method except for anonymous DH
  - contains one or a chain of X.509 certificates (up to a known root CA)
  - may contain
    - public RSA key suitable for encryption, or
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters
- server\_key\_exchange
  - sent only if the certificate does not contain enough information to complete the key exchange (e.g., the certificate contains an RSA signing key only)
  - may contain
    - public RSA key (exponent and modulus), or
    - DH parameters (p, g, public DH value), or
    - Fortezza parameters
  - digitally signed
    - if DSS: SHA-1 hash of (client\_random | server\_random | server\_params) is signed
    - if RSA: MD5 hash and SHA-1 hash of (client\_random | server\_random | server\_params) are concatenated and encrypted with the private RSA key

© Levente Buttyán

18

## Cert request and server hello done msgs

- `certificate_request`
  - sent if the client needs to authenticate itself
  - specifies which type of certificate is requested (`rsa_sign`, `dss_sign`, `rsa_fixed_dh`, `dss_fixed_dh`, ...)
- `server_hello_done`
  - sent to indicate that the server is finished its part of the key exchange
  - after sending this message the server waits for client response
  - the client should verify that the server provided a valid certificate and the server parameters are acceptable

## Client authentication and key exchange

- `certificate`
  - sent only if requested by the server
  - may contain
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters
- `client_key_exchange`
  - always sent (but it is empty if the key exchange method is fix DH)
  - may contain
    - RSA encrypted pre-master secret, or
    - client one-time public DH value, or
    - Fortezza key exchange parameters
- `certificate_verify`
  - sent only if the client sent a certificate
  - provides client authentication
  - contains signed hash of all the previous handshake messages
    - if DSS: SHA-1 hash is signed
    - if RSA: MD5 and SHA-1 hash is concatenated and encrypted with the private key  
 $MD5(\text{master\_secret} \mid \text{pad\_2} \mid MD5(\text{handshake\_messages} \mid \text{master\_secret} \mid \text{pad\_1}))$   
 $SHA(\text{master\_secret} \mid \text{pad\_2} \mid SHA(\text{handshake\_messages} \mid \text{master\_secret} \mid \text{pad\_1}))$

## Finished messages

- finished
  - sent immediately after the `change_cipher_spec` message
  - used to authenticate all previous handshake messages
  - first message that uses the newly negotiated algorithms, keys, IVs, etc.
  - contains the MD5 and SHA-1 hash of all the previous handshake messages:
 

```
MD5( master_secret | pad_2 | MD5( handshake_messages | sender | master_secret | pad_1 ) ) |
SHA( master_secret | pad_2 | SHA( handshake_messages | sender | master_secret | pad_1 ) )
```

 where "sender" is a code that identifies that the sender is the client or the server (client: 0x434C4E54; server: 0x53525652)

## Cryptographic computations

- pre-master secret
  - if key exchange is RSA based:
    - generated by the client
    - sent to the server encrypted with the server's public RSA key
  - if key exchange is Diffie-Hellman based:
    - $\text{pre\_master\_secret} = g^{xy} \text{ mod } p$
- master secret (48 bytes)
 

```
master_secret = MD5( pre_master_sec | SHA( "A" | pre_master_sec | client_random | server_random ) ) |
MD5( pre_master_sec | SHA( "BB" | pre_master_sec | client_random | server_random ) ) |
MD5( pre_master_sec | SHA( "CCC" | pre_master_sec | client_random | server_random ) )
```
- keys, MAC secrets, IVs
 

```
MD5( master_secret | SHA( "A" | master_secret | client_random | server_random ) ) |
MD5( master_secret | SHA( "BB" | master_secret | client_random | server_random ) ) |
MD5( master_secret | SHA( "CCC" | master_secret | client_random | server_random ) ) | ...
```



key block :

client write MAC sec	server write MAC sec	client write key	server write key	...
----------------------	----------------------	------------------	------------------	-----

## Key exchange alternatives

- RSA / no client authentication
  - server sends its encryption capable RSA public key in `server_certificate`
  - `server_key_exchange` is not sent
  - client sends encrypted pre-master secret in `client_key_exchange`
  - `client_certificate` and `certificate_verify` are not sentor
  - server sends its RSA or DSS public signature key in `server_certificate`
  - server sends a temporary RSA public key in `server_key_exchange`
  - client sends encrypted pre-master secret in `client_key_exchange`
  - `client_certificate` and `certificate_verify` are not sent

© Levente Buttyán

23

## Key exchange alternatives cont'd

- RSA / client is authenticated
  - server sends its encryption capable RSA public key in `server_certificate`
  - `server_key_exchange` is not sent
  - client sends its RSA or DSS public signature key in `client_certificate`
  - client sends encrypted pre-master secret in `client_key_exchange`
  - client sends signature on all previous handshake messages in `certificate_verify`or
  - server sends its RSA or DSS public signature key in `server_certificate`
  - server sends a one-time RSA public key in `server_key_exchange`
  - client sends its RSA or DSS public signature key in `client_certificate`
  - client sends encrypted pre-master secret in `client_key_exchange`
  - client sends signature on all previous handshake messages in `certificate_verify`

© Levente Buttyán

24

## Key exchange alternatives cont'd

- fix DH / no client authentication
  - server sends its fix DH parameters in `server_certificate`
  - `server_key_exchange` is not sent
  - client sends its one-time DH public value in `client_key_exchange`
  - `client_certificate` and `certificate_verify` are not sent
- fix DH / client is authenticated
  - server sends its fix DH parameters in `server_certificate`
  - `server_key_exchange` is not sent
  - client sends its fix DH parameters in `client_certificate`
  - `client_key_exchange` is sent but empty
  - `certificate_verify` is not sent

© Levente Buttyán

25

## Key exchange alternatives cont'd

- ephemeral DH / no client authentication
  - server sends its RSA or DSS public signature key in `server_certificate`
  - server sends signed one-time DH parameters in `server_key_exchange`
  - client sends one-time DH public value in `client_key_exchange`
  - `client_certificate` and `certificate_verify` are not sent
- ephemeral DH / client is authenticated
  - server sends its RSA or DSS public signature key in `server_certificate`
  - server sends signed one-time DH parameters in `server_key_exchange`
  - client sends its RSA or DSS public signature key in `client_certificate`
  - client sends one-time DH public value in `client_key_exchange`
  - client sends signature on all previous handshake messages in `certificate_verify`

© Levente Buttyán

26

## Key exchange alternatives cont'd

- anonymous DH / no client authentication
  - server\_certificate is not sent
  - server sends (unsigned) one-time DH parameters in server\_key\_exchange
  - client sends one-time DH public value in client\_key\_exchange
  - client\_certificate and certificate\_verify are not sent
- anonymous DH / client is authenticated
  - not allowed

## Analysis of the SSL Record and Handshake Protocols

## Eavesdropping

- + all application data is encrypted with a short term connection key
- + short term key is derived from per-connection salts (client and server randoms) and a strong shared secret (master secret) by hashing (one-way operation)
  - + even if connection keys are compromised the master secret remains intact
- + different keys are used in each connection and in each direction of the connection
- + supported encryption algorithms are strong

## Traffic analysis

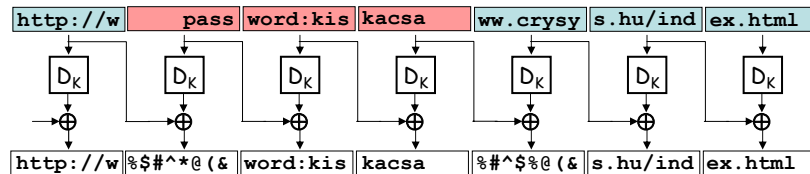
- SSL doesn't attempt to protect against traffic analysis
  - padding length is not random
  - no padding if a stream cipher is used (this is the default option)
- if SSL is used to protect HTTP traffic, then an attacker
  - can learn the length of a requested URL
  - can learn the length of the HTML data returned
  - could find which URL was requested with high probability

## Active attacks on confidentiality

- cut-and-paste attack

C → S: pass word:kis kacsa

S → C: http://w ww.crysy s.hu/ind ex.html



- + SSL prevents cut-and-paste attacks
  - + different keys are used in different directions (and connections)
  - + all encrypted packets are authenticated by a MAC

© Levente Buttyán

31

## Replay attacks

- + SSL protects against replay attacks by including an implicit sequence number in the MAC computation
  - + prevents re-order and deletion of messages
- + sequence numbers are 64 bit long
  - + practically never wraps around

© Levente Buttyán

32

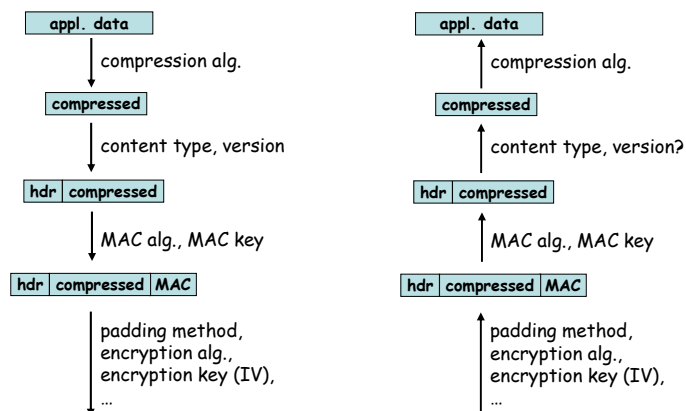


## Message authentication

- +/- SSL uses a HMAC-like MAC
  - it actually uses an obsolete version of HMAC
  - + HMAC is provably secure
- + MAC secret is 128 bits long
- + different MAC secrets are used in different directions and connections
- the MAC doesn't involve the version number (part of the message)
  - if the version number is ever used, then it should be covered by the MAC
  - if the version number is never used, then it should not be sent

## The Horton principle

- not only data should be authenticated, but all context information on which processing and interpretation of the data depend (e.g., algorithms, keys, information added to headers, etc)

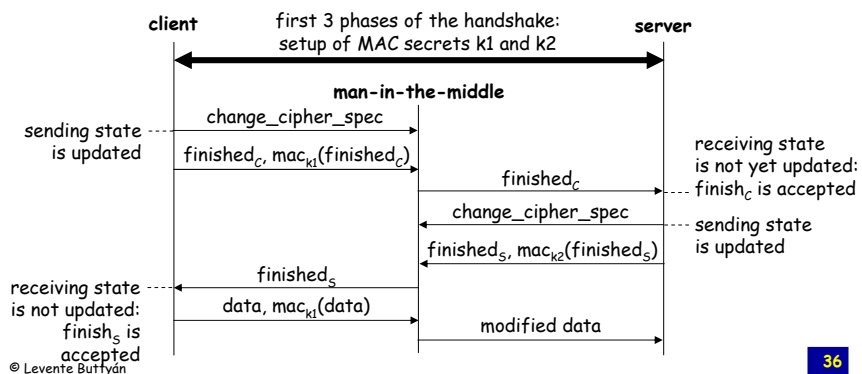


## Cipher suite rollback attack

- in SSL 2.0, an attacker could force the use of an export-weakened encryption algorithm by modifying the list of supported cipher suites in the hello messages
- this is prevented in SSL 3.0 by authenticating all handshake messages with the master secret (in the finished messages)
- the master secret itself is authenticated by other means
  - for the client:
    - implicit authentication via the server certificate
      - only the server could decrypt the RSA encrypted pre-master secret
      - only the server could compute the pre-master secret from the client's public DH value
    - explicit authentication via the server\_key\_exchange message (if sent)
      - ephemeral DH parameters are signed by the server
  - for the server:
    - explicit authentication via the certificate\_verify message (if sent)
      - certificate\_verify is signed by the client
      - it involves the master secret

## Dropping the change\_cipher\_spec msg

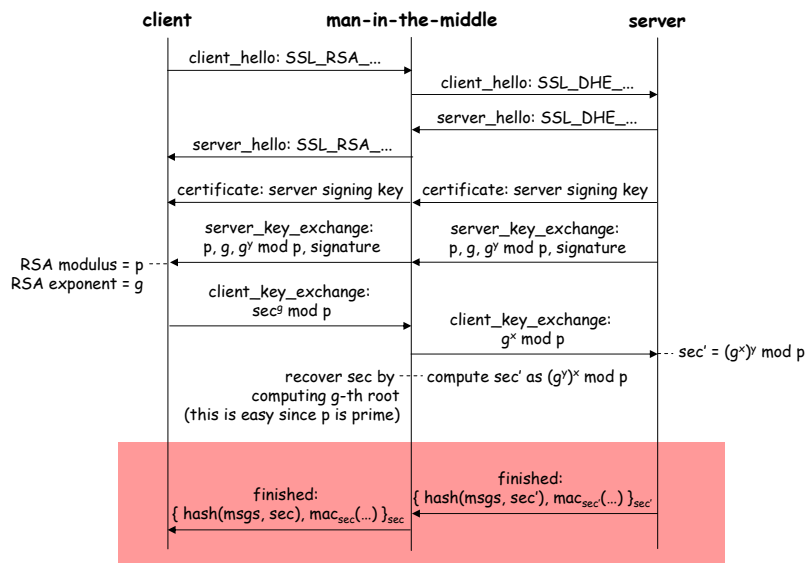
- authentication in the finished message does not protect the change\_cipher\_spec message (it is not part of the handshake protocol !)
- this may allow the following attack:
  - assume that the negotiated cipher suite includes only message authentication (no encryption)



## Dropping the change\_cipher\_spec msg

- if the negotiated cipher suite includes encryption, then the attacks doesn't work
  - client sends encrypted finished message
  - server expects clear finished message
  - the attacker cannot decrypt the encrypted finished message
- simplest fix: require reception of change\_cipher\_spec before processing the finished message
  - this seems to be obvious, but...
  - even Netscape's reference SSL implementation SSLRef 3.0b1 allows processing finished messages without checking if a change\_cipher\_spec has been received
  - SSLRef 3.0b3 contains the fix
- another fix: include the change\_cipher\_spec message in the computation of the finished message
  - this would require a more radical change in the SSL specification

## Key-exchange algorithm rollback



## Key-exchange algorithm rollback

- SSL authenticates only the server's (RSA or DH) parameters in the server\_key\_exchange message
- it doesn't authenticate the context (key exchange algorithm in use) in which those parameters should be interpreted
- this is not compliant with the Horton principle !
- a fix:
  - hash all messages exchanged before the server\_key\_exchange message
  - include the hash in the signature in server\_key\_exchange message

© Levente Buttyán

39

## Version rollback attacks

- SSL 3.0 implementations may still support SSL 2.0
- an attacker may change the client\_hello message so that it looks like an SSL 2.0 client\_hello
- as a result the client and the server will run SSL 2.0
- SSL 2.0 has serious security flaws
  - among other things, there are no finished messages to authenticate the handshake
  - the version rollback attack will go undetected
- fortunately, SSL 3.0 can detect version rollback
  - pre-master secret generated on SSL 3.0 enabled clients:
 

```
struct{
    ProtocolVersion client_version; // latest version supported by the client
    opaque random[46];           // random bytes
} PreMasterSecret;
```
  - an SSL 3.0 enabled server detects the version rollback attack, when it runs an SSL 2.0 handshake but receives a pre-master secret that includes version 3.0 as the latest version supported by the client

© Levente Buttyán

40

## MAC usage

- while the SSL Record Protocol uses HMAC (an early version), the SSL Handshake Protocol uses ad-hoc MACs at several points
  - certificate\_verify:  
`hash( master_secret | pad_2 | hash( handshake_messages | master_secret | pad_1 ) )`
  - finished:  
`hash( master_secret | pad_2 | hash( handshake_messages | sender | master_secret | pad_1 ) )`
- in addition, these ad-hoc MACs involve the master secret
- this is dangerous, and SSL should use HMAC consistently

## Analysis summary

- SSL Record Protocol
  - + good protection against passive eavesdropping and active attacks
  - should better protect against traffic analysis (e.g., apply random padding)
  - should use the latest version of HMAC
- SSL Handshake Protocol
  - + some active attacks are foiled
    - cipher suite rollback
    - version rollback
  - other active attacks could still be possible depending on how an implementation interprets the SSL specification
    - dropping change\_cipher\_spec messages
    - key-exchange algorithm rollback
  - ad-hoc MAC constructions should be replaced with HMAC
- overall: SSL 3.0 was an extremely important step toward practical communication security for Internet applications

## SSL vs. TLS

### Miscellaneous changes

- version number
  - for TLS the current version number is 3.1
- cipher suites
  - TLS doesn't support Fortezza key exchange and Fortezza encryption
- padding
  - variable length padding is allowed (max 255 padding bytes)
- MAC
  - TLS uses the latest version of HMAC
  - the MAC covers the version field of the record header too
- certificate\_verify message
  - the hash is computed only over the handshake messages
  - in SSL, the hash contained the master\_secret and pads
- more alert codes

## New pseudorandom function (PRF)

- $P\_hash(secret, seed) = \text{HMAC\_hash}(secret, A(1) \parallel seed) \parallel \text{HMAC\_hash}(secret, A(2) \parallel seed) \parallel \text{HMAC\_hash}(secret, A(3) \parallel seed) \parallel \dots$

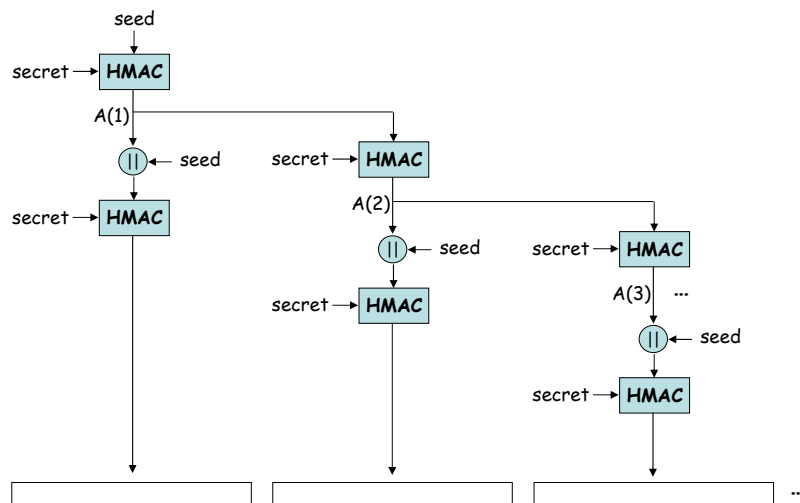
where

$$A(0) = seed$$

$$A(i) = \text{HMAC\_hash}(secret, A(i-1))$$

- $\text{PRF}(secret, label, seed) = P\_MD5(secret\_left, label \parallel seed) \oplus P\_SHA(secret\_right, label \parallel seed)$

## P\_hash illustrated



## Usage of the new PRF

- finished message  
`PRF( master_secret,  
"client finished",  
MD5(handshake_messages) | SHA(handshake_messages) )`
- cryptographic computations
  - pre-master secret is calculated in the same way as in SSL
  - master secret:  
`PRF( pre_master_secret,  
"master secret",  
client_random | server_random )`
  - key block:  
`PRF( master_secret,  
"key expansion",  
server_random | client_random )`

## Recommended readings

- SSL v3.0 specification, available on-line at <http://wp.netscape.com/eng/ssl3/index.html>
- D. Wagner, B. Schneier, *Analysis of the SSL 3.0 protocol*, 2<sup>nd</sup> USENIX Workshop on Electronic Commerce, 1996.
- The TLS protocol v1.0, available on-line as RFC 2246