Secure Sockets Layer

Tomáš Majer

Fakulta riadenia a informatiky
Žilinská univerzita v Žiline

tomas.majer@fri.uniza.sk

14. december 2010

História SSL

Secure Sockets Layer (SSL):

- Kryptografický protokol vytvorený firmou Netscape.
- SSLv1 nikdy nebola zverejnená.
- SSLv2 bola zverejnená v roku 1995, ale obsahovala bezpečnostné diery.
- SSLv3 bola zverejnená v roku 1996.

Transport Layer Security (TLS)

- Vytvorený spoločnosťou The Internet Society (http://www.isoc.org).
- TLSv1.0 bol zverejnený v RFC 2246 v januári 1999.
- TLSv1.1 bol zverejnený v RFC 4346 v apríli 2006.
- TLSv1.2 bol zverejnený v RFC 5246 v auguste 2008.

Ciele SSL

- SSL je bezpečnostný protokol, ktorý umožňuje privátnu komunikáciu cez internet. Protokol umožňuje klient/server aplikáciám komunikovať tak, aby prenášané údaje nemohli byť odpočúvané, pozmeňované a falšované.[1]
- SSL pomocou kryptografie zabezpečuje:
  1. utajenie,
  2. autentifikáciu,
  3. integritu,
  4. neopakovateľnosť prenášaných údajov.

[1] This document specifies Version 3.0 of the Secure Sockets Layer (SSL V3.0) protocol, a security protocol that provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Fragmentácia prenášaných dát

- SSL funguje nad TCP/IP[2] protokolom, t.j. ochranu prenášaných údajov proti (náhodným, neúmyselným) chybám pri prenose zabezpečuje TCP/IP vrstva.
- Údaje medzi komunikujúcimi stranami sa vymieňajú vo fragmentoch, pričom jeden fragment nemôže obsahovať viac ako 16 KB dát.
- Vytvorený fragment sa postupne komprimuje, doplní o MAC[3] a zašifruje.
- Fragment môže obsahovať údaje:
  - okamžitá zmena spôsobu šifrovania (change_cipher_spec),
  - upozornenie (alert),
  - dohodnutie pravidiel (handshake),
  - údaje aplikácie (application_data).

[2] Transport Control Protocol/Internet Protocol

[3] Message Authentification Code

SSLPlainText

```
struct {
  ContentType type;
  ProtocolVersion version;
  uint16 length;
  opaque fragment[SSLPlaintext.length];
} SSLPlaintext;
```

SSLCompressed

```
struct {
  ContentType type;
  ProtocolVersion version;
  uint16 length;
  opaque fragment[SSLCompressed.length];
} SSLCompressed;
```

SSLCiphertext

```
struct {
  ContentType type;
  ProtocolVersion version;
  uint16 length;
  select (CipherSpec.cipher_type) {
    case stream:  GenericStreamCipher;
    case block:  GenericBlockCipher;
  } fragment;
} SSLCiphertext;
```

GenericStreamCipher

```
stream-ciphered struct {
  opaque content[SSLCompressed.length];
  opaque MAC[CipherSpec.hash_size];
} GenericStreamCipher;
```

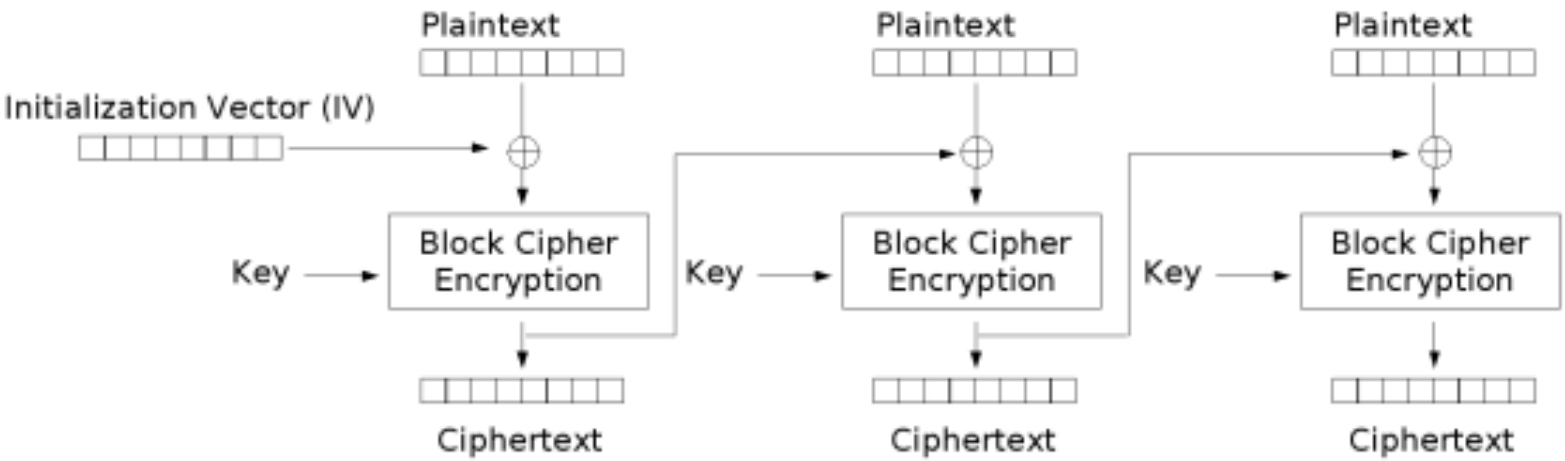
GenericBlockCipher

```
block-ciphered struct {
  opaque content[SSLCompressed.length];
  opaque MAC[CipherSpec.hash_size];
  uint8 padding[GenericBlockCipher.padding_length];
  uint8 padding_length;
} GenericBlockCipher;
```

Blokové šifry sa používajú v CBC[4] móde.

[4] Cipher-Block Chaining

Cipher-Block Chaining



Cipher Block Chaining (CBC) mode encryption

Výpočet MAC

```
hash(MAC_write_secret + pad_2 +
  hash(MAC_write_secret + pad_1 + seq_num +
    SSLCompressed.type + SSLCompressed.length +
    SSLCompressed.fragment));
```

`pad_1` Znak '6' (ASCII 0x36) opakovaný 48–krát pre MD5 alebo 40–krát pre SHA.

`pad_2` Znak '\' (ASCII 0x5C) opakovaný 48–krát pre MD5 alebo 40–krát pre SHA.

Parametre sedenia

session identifier
: identifikátor sedenia

peer certificate
: certifikát druhej strany

compression method
: dohodnutý komprimačný algoritmus

cipher spec
: dohodnutý spôsob šifrovania a overovania správ

master secret
: dohodnuté zdieľané tajomstvo

is resumable
: príznak, či je možné použiť dohodnuté parametre sedenia pre ďalšie pripojenia

Parametre pripojenia

server random

client random

: náhodné čísla generované serverom a klientom pri nadviazaní spojenia

server write MAC

client write MAC

: tajomstvá pre vytváranie MAC

server write key

client write key

: tajné kľúče symetrického šifrovacieho algoritmu

Parametre pripojenia

server initialization vector

client initialization vector

: inicializačné vektory pre blokovú šifru (v CBC móde)

server sequence number

client sequence number

: počty odoslaných správ

Generovanie tajných kľúčov

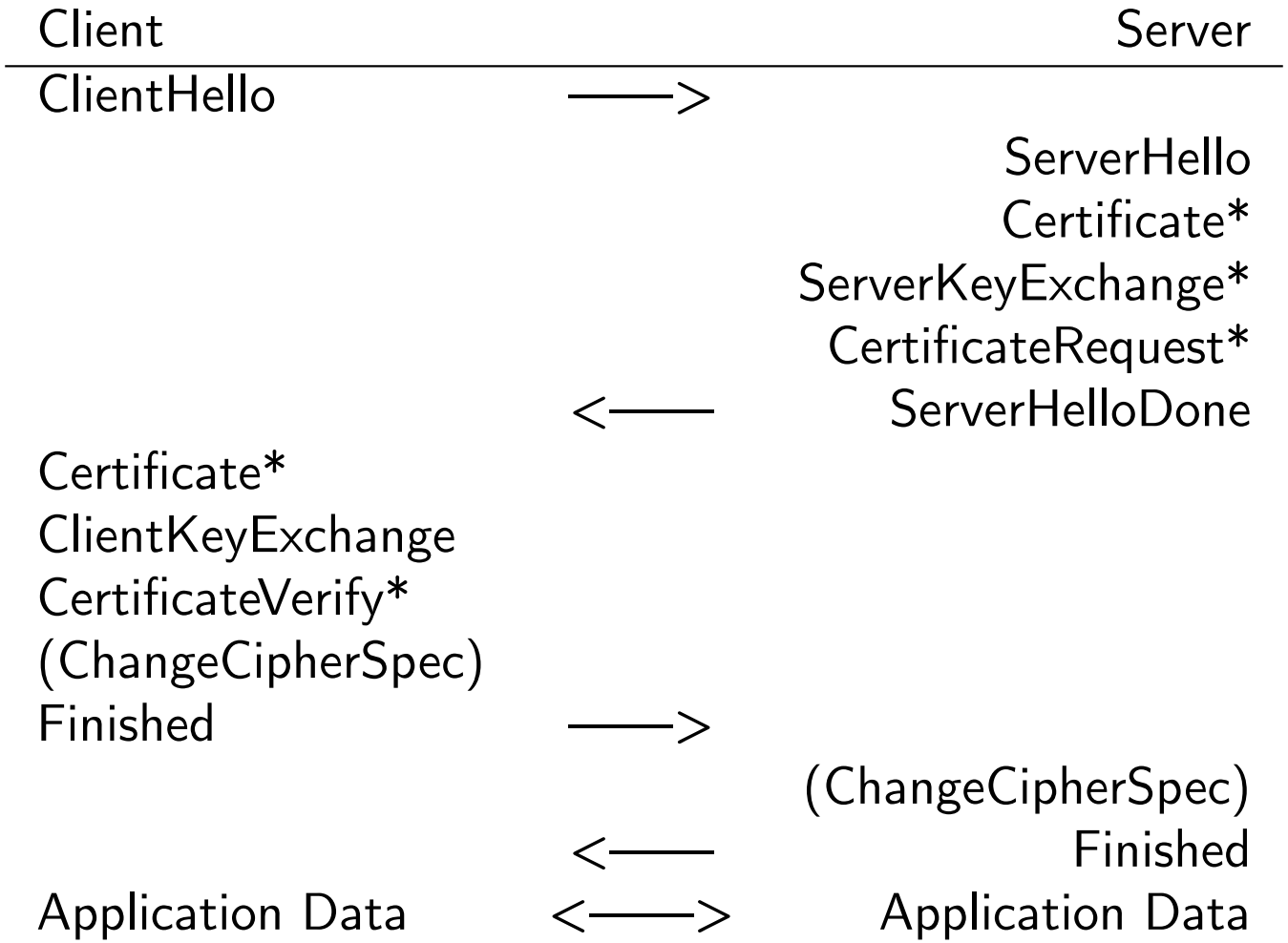
```
master_secret =
  MD5(pre_master_secret + SHA('A' + pre_master_secret +
    ClientHello.random + ServerHello.random)) +
  MD5(pre_master_secret + SHA('BB' + pre_master_secret +
    ClientHello.random + ServerHello.random)) +
  MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
    ClientHello.random + ServerHello.random));
```

Generovanie tajných kľúčov

```
key_block =
  MD5(master_secret + SHA('A' + master_secret +
    ServerHello.random + ClientHello.random)) +
  MD5(master_secret + SHA('BB' + master_secret +
    ServerHello.random + ClientHello.random)) +
  MD5(master_secret + SHA('CCC' + master_secret +
    ServerHello.random + ClientHello.random)) +
  MD5(master_secret + SHA('DDDD' + master_secret +
    ServerHello.random + ClientHello.random)) +
  MD5(...)

  client_write_MAC_secret[CipherSpec.hash_size]
  server_write_MAC_secret[CipherSpec.hash_size]
  client_write_key[CipherSpec.key_material]
  server_write_key[CipherSpec.key_material]
  client_write_IV[CipherSpec.IV_size]
  server_write_IV[CipherSpec.IV_size]
```

Handshake protokol



Handshake protokol

```
struct {
  HandshakeType msg_type;
  uint24 length;
  select (HandshakeType) {
    case hello_request:  HelloRequest;
    case client_hello:  ClientHello;
    case server_hello:  ServerHello;
    case certificate:  Certificate;
    case server_key_exchange:  ServerKeyExchange;
    case certificate_request:  CertificateRequest;
    case server_hello_done:  ServerHelloDone;
    case certificate_verify:  CertificateVerify;
    case client_key_exchange:  ClientKeyExchange;
    case finished:  Finished;
  } body;
} Handshake;
```

ClientHello

```
struct {
  ProtocolVersion client_version;
  Random random;
  SessionID session_id;
  CipherSuite cipher_suites<2..2^16-1>;
  CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;

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

opaque SessionID<0..32>;
```

ServerHello

```
struct {
  ProtocolVersion server_version;
  Random random;
  SessionID session_id;
  CipherSuite cipher_suite;
  CompressionMethod compression_method;
} ServerHello;
```

Certificate

```
opaque ASN.1Cert<1..2^24-1>;
struct {
  ASN.1Cert certificate_list<1..2^24-1>;
} Certificate;
```

ServerKeyExchange

```
struct {
  select (KeyExchangeAlgorithm) {
    case diffie_hellman:
      ServerDHParams params;
      Signature signed_params;
    case rsa:
      ServerRSAParams params;
      Signature signed_params;
    case fortezza_kea:
      ServerFortezzaParams params;
  };
} ServerKeyExchange;

enum { rsa, diffie_hellman, fortezza_kea }
  KeyExchangeAlgorithm;
```

ServerKeyExchange

```
struct {
  opaque rsa_modulus<1..2^16-1>;
  opaque rsa_exponent<1..2^16-1>;
} ServerRSAParams;

struct {
  opaque dh_p<1..2^16-1>;
  opaque dh_g<1..2^16-1>;
  opaque dh_Ys<1..2^16-1>;
} ServerDHParams; /* Ephemeral DH parameters */
```

ServerKeyExchange

```
digitally-signed struct {
  select(SignatureAlgorithm) {
    case anonymous:  struct { };
    case rsa:
      opaque md5_hash[16];
      opaque sha_hash[20];
    case dsa:
      opaque sha_hash[20];
  };
} Signature;
```

CertificateRequest

```
struct {
  ClientCertificateType certificate_types<1..2^8-1>;
  DistinguishedName certificate_authorities<3..2^16-1>;
} CertificateRequest;
```

ServerHelloDone

```
struct { } ServerHelloDone;
```

ClientKeyExchange

```
struct {
  select (KeyExchangeAlgorithm) {
    case rsa:  EncryptedPreMasterSecret;
    case diffie_hellman:  ClientDiffieHellmanPublic;
    case fortezza_kea:  FortezzaKeys;
  } exchange_keys;
} ClientKeyExchange;

struct {   ProtocolVersion client_version;   opaque
random[46]; } PreMasterSecret;

struct {
  select (PublicValueEncoding) {
    case implicit:  struct  ;
    case explicit:  opaque dh_Yc<1..2^16-1>;
  } dh_public;
} ClientDiffieHellmanPublic;
```

CertificateVerify

```
struct {
  Signature signature;
} CertificateVerify;

CertificateVerify.signature.md5_hash
  MD5(master_secret + pad_2 +
    MD5(handshake_messages + master_secret + pad_1));
Certificate.signature.sha_hash
  SHA(master_secret + pad_2 +
    SHA(handshake_messages + master_secret + pad_1));
```

Finished

```
enum { client(0x434C4E54), server(0x53525652) } Sender;

struct {
  opaque md5_hash[16];
  opaque sha_hash[20];
} Finished;

md5_hash = MD5(master_secret + pad2 +
  MD5(handshake_messages + Sender +
    master_secret + pad1));

sha_hash = SHA(master_secret + pad2 +
  SHA(handshake_messages + Sender +
    master_secret + pad1));
```

Koniec

Ďakujem za pozornosť.