5 Security Analysis of Real World Protocols

Abstract Several de facto or industrial standards are widely used in many real world applications are discussed and analyzed via the trusted freshness approach. The typical cryptographic protocols include the Secure Socket Layer Protocol (SSL) and its variant, Transport Layer Security Protocol (TLS), the Internet Key Exchange Protocol (IKE) and the Kerberos Authentication Protocol. From the discussion and the security analysis of these protocols, we will see that it is very challenging to achieve strong security properties of the cryptographic protocols fit for application.

Our study of cryptographic protocols in the preceding chapters has focused on the academic protocols, while in this chapter, we will touch some widely used real world cryptographic protocols and analyze them using the trusted freshness method. This will help the readers to understand the trusted freshness method deeply and evaluate the security strength of a cryptographic protocol they may use in practice.

The Internet is an enormous open network of computers and devices called "nodes". To deal with the complicated network well, the ISO (the International Organization for Standardization) presents the Open System Interconnection Reference Model (OSI Reference Model or OSI Model) which is an abstract description for layered communications and computer network protocol design. In its most basic form, it divides network architecture into seven layers which, from top to bottom, are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layers, and they are also called the seventh layer, the sixth layer, and so on and so forth. A layer is a collection of conceptually similar functions that provide services to the layer above it and receive service from the layer below it.

Figure 5.1 illustrates a simplified ISO Open System Interconnection (OSI) architecture considering placement of key distribution protocols[1–12].

Internet is an open network environment, and each node in the network trusts each other from the Internet's original design intention, hence insecure systems are already in wide use. To keep backward compatibility, secure solutions should be added in with the least interruption to the insecure systems which are already in operation[13]. E.g. the SSL avoids modifying "TCP

L. Dong et al., *Cryptographic Protocol*

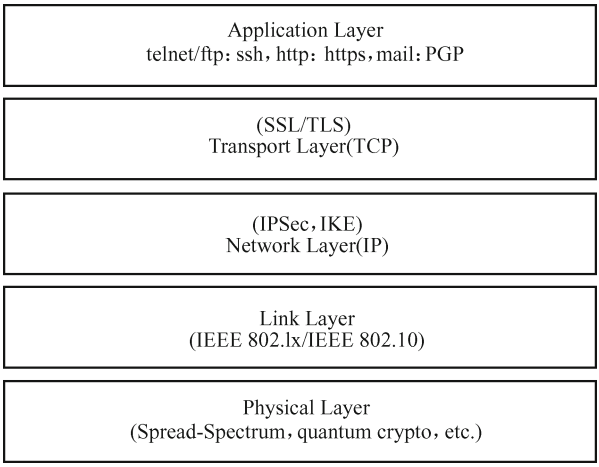


Fig. 5.1 The ISO Open System Interconnection.

stack" and requires minimum changes to the application, mostly used to authenticate servers; IPSec is transparent to the application, but it requires modification of the network stack and establishes a secure channel between nodes.

In this chapter, we shall introduce and discuss several cryptographic protocols which are de facto or industrial standards, and they are already widely used in many real world applications. The real world protocols we shall study include the Secure Socket Layer Protocol (SSL)[2], and its variant, Transport Layer Security Protocol (TLS)[3]; the Internet Key Exchange Protocol (IKE)[4, 5] and the Kerberos Authentication Protocol[6, 7]. From the discussion and the security analysis of these protocols, we will see that it is very challenging to achieve strong security properties of the cryptographic protocols fit for application.

5.1 Secure Socket Layer and Transport Layer Security

Secure Socket Layer (SSL), and its variant, Transport Layer Security (TLS), are the de facto standards used to end-to-end encrypt, and they are mainly for WorldWideWeb (Web for short) security[2, 3]. This includes specifically credit card purchases and bank sites, but it may also be used on any site requesting a password or dealing with personal information. SSL and TLS use public-key encryption.

The most recent draft of the SSL 3.0 specification was published in November 1996 by Netscape and SSL 3.0 was the basis for the TLS 1.0 (RFC 2246) specification published by the Internet Engineering Task force (IETF) in 1999. The IETF made some small changes and clarifications and published RFC4346 in 2006 detailing TLS 1.1. The most recent draft of the TLS 1.2

(RFC 5246) specification was published in August 2008 by IETF.

5.1.1 SSL and TLS overview

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. Except cryptographic security, the goals also include interoperability, extensibility, and relative efficiency. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The SSL Handshake Protocol can be considered as a stateful process running on the client and server machines. A stateful connection is called a "session", and a session can be renegotiated.

The SSL protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., Transmission Control Protocol, that is TCP protocol) is the SSL Record Protocol. The SSL Record Protocol provides secure encapsulation of the communication channel for use by higher layer application protocols. The higher level protocols include Handshake, Change Cipher Spec, and Alert protocols as well as application data. The SSL handshake protocol is a key exchange protocol which initializes and synchronizes cryptographic state at the two endpoints. After the key-exchange protocol completes, sensitive application data can be sent via the SSL record layer. One advantage of SSL is that a higher level protocol can layer on top of the SSL Protocol transparently. The technique used to encrypt and verify the integrity of SSL records is specified by the currently active Cipher Spec. A typical example would be to encrypt data using DES and generate authentication codes using MD5.

SSL/TLS has 4 underlying protocols: Handshake, Record, Change Cipher Spec, and Alert. This is laid out as:

8 bit	8 bit	8 bit	16 bit	16384 bytes
Type	Major version	Minor Version	Record Length	Record Data

In decimal, the types are as follows:
20 Change Cipher Spec
21 Alert
22 Handshake
23 Application (data)

The version would be 3 and then 0 for SSL 3.0. Since TLS is a "minor modification to the SSL 3.0 protocol," TLS 1.0 is defined as SSL major version 3, minor version 1, TLS 1.1 is 3 and then 2, and the upcoming TLS 1.2 will be major version 3, then minor version 3.

The record length is written in terms of bytes and can not exceed 2^{14} (16, 384). Compression allows for the length to be extended by up to 1024 bytes,

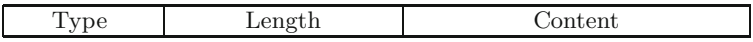
to a new max of 17,408 bytes in the TLS compressed length field.

SSL connections begin with a 4-way handshake. The keys for symmetric encryption and for HMAC are generated uniquely for each session connection and are based on a secret negotiated by the SSL Handshake Protocol.

Alert messages with a level of fatal result in the immediate termination of the connection. In this case, other connections corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections.

5.1.2 The SSL handshake protocol

The SSL Handshake Protocol is one of the defined higher level clients of the SSL Record Protocol. The SSL Handshake Protocol allows the server and client to authenticate each other and to negotiate a encryption algorithm and cryptographic keys for symmetric encryption and for HMAC uniquely for each session connection, and thereby to establish a secure session connection with the SSL Record Protocol to process secure communications with higher level application protocols. The handshake protocol structure is:



The allowed values for type are indicated in Table 5.1.

Table 5.1 Allowed values for type in SSL handshake protocol

Type Value	Type	Remark
0	HelloRequest	
1	ClientHello	
2	ServerHello	
11	Certificate	Optional
12	ServerKeyExchange	Optional
13	CertificateRequest	Optional
14	ServerHelloDone	
15	CertificateVerify	
16	ClientKeyExchange	Optional
20	Finished	

The handshake protocol messages are presented in the order in which they must be sent; sending handshake messages in an unexpected order results in a fatal error.

The data handshake process performs the following steps, as shown in Fig. 5.2.

Message 1 $C \rightarrow S$: Client Hello

Message 2 $S \rightarrow C$: ServerHello
ServerCertificate*
ServerKeyExchange*
CertificateRequest*
ServerHelloDone
Message 3 $C \rightarrow S$: ClientCertificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished
Message 4 $S \rightarrow C$: [ChangeCipherSpec]
Finished

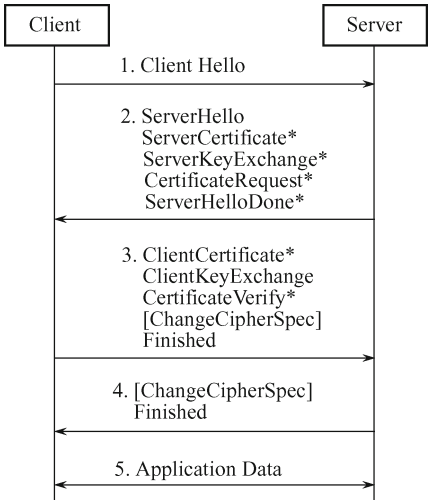


Fig. 5.2 Message flow for a full handshake of SSL.

Notation

C denotes a client (the client-side web browser), S denotes the web server, and * indicates optional or situation-dependent messages that are not always sent.

Premise

ClientCertificate (ServerCertificate) is a certification of C's (or S's) identity and corresponding public-key K_C (or K_S) signed by a trusted certification authority center CA.

Protocol actions

1) In Message 1 (*ClientHello message*:)

The client starts the session connection by sending a message to which the server must respond with a ServerHello message, or else the connection will fail. The client may resume an existing session. The *ClientHello* message will have the following data:

— The protocol version is in the client hello which is for backward compatibility use.
— A 32-bit Unix format timestamp and a 28-byte random number ClientHello.random are generated by the client.
— The session identifier: When the client wishes to start a new session connection, this field should be empty. The client may specify a session identifier of a current or previous session. Doing this allows for multiple secure connections without going through the entire handshake process each time, although both Hello, the Change Cipher Spec, and both Finished messages must still be exchanged and be valid.
— The cipher suite, a list of the cryptographic options supported in the client side machine, sorted with the client's first preference first. Each cipher suite defines the algorithm for key exchange, the bulk encryption algorithm with secret key and length, and the message authentication code (MAC). A wide range of public-key and symmetric cryptographic algorithms, digital signature schemes, MAC schemes and hash functions can be proposed by the client.

A cipher suite identifies a Cipher Spec. These structures are part of the SSL session state. The Cipher Spec includes:

```
enum {stream, block} CipherType;
enum {true, false} IsExportable;
enum {null, rc4, rc2, des, 3des, des40, fortezza}
      BulkCipherAlgorithm;
enum {null, md5, sha} MACAlgorithm;
struct {
      BulkCipherAlgorithm bulk_cipher_algorithm;
      MACAlgorithm mac_algorithm;
      CipherType cipher_type;
      IsExportable is_exportable;
      uint8 hash_size;
      uint8 key_material;
      uint8 IV_size;
     } CipherSpec;
```

Here, unit8 is an unsigned byte.

— The compression method supported in the client side machine.

2) In Message 2 (*ServerHello message*:)

The server responds to Client Hello message with the ServerHello message. The *ServerHello* message will have the following data:

— The version number being used: the lowering of the server's highest supported version and the version in the client hello.

— A 32-bit Unix format timestamp and a 28-byte random number ServerHello.random are generated by the server.
— The session identifier: if the session ID is recognized, then a short handshake is used and the following fields are filled in with the values from the previous connection. Otherwise, the ServerHello generates a new session ID, uses this new value in this field, and caches the session ID in its local memory. The server may return an empty session ID to indicate that the session will not be cached and therefore cannot be resumed.
— The cipher suite chosen by the server, where the server selects a single scheme for each necessary cryptographic operation, informs the client in this field.
— The compression method chosen by the server.

If the server can not find an acceptable cipher suite and compression method, it will respond with a handshake failure alert.

(1) ServerCertificate message: Unless the key exchange method is anonymous, if the server is to be authenticated (which is generally the case), the server will send out a certificate immediately after sending the ServerHello. The certificate is generally an X.509 v3 certificate public-key and unless otherwise specified uses the same key exchange method and signing algorithm previously decided on. An X.509 certificate contains sufficient information about the name and the public-key of the certificate owner and that about the issuing certification authority. Sending a list of certificates permits the client to choose one with the public-key algorithm supported in the client's machine. That is, certificates from all the up line servers are necessary to get to the one that the client trusts must be included. The order of these should be so that each certificate validates the one before it.

(2) ServerKeyExchange message: If the ServerCertificate does not contain enough data for a pre-master secret, then a ServerKeyExchange is sent with either an RSA public, or a Diffie-Hellman public-key (This is the case for DHE_DSS, DHE_RSA, and DH_anon; but not for RSA, DH_DSS, and DH_RSA key exchange methods). ServerKeyExchange contains the server's public-key material matching the certificate list in ServerCertificate. The material for Diffie-Hellman key agreement will be included here which is the tuple (p, g, g^y) where p is a prime modulus, g is a generator modulo p of a large group and y is an integer cached in the server's local memory.

(3) CertificateRequest message: If it is appropriate, the server may request a certificate from the client with a CertificateRequest. This would immediately follow the ServerCertificate, or present the ServerKeyExchange. The CertificateRequest would specify the types of certificates the server will accept and the Certificate Authorities the server trusts.

(4) ServerHelloDone message: The ServerHelloDone indicates to the client that server is done sending data and the client should now verify the certificates and whatnot it has received.

3) In Message 3 (*ClientCertificate message*:)

After receiving the ServerHelloDone the client would respond with a message identical in format to the ServerCertificate if a CertificateRequest was received before.

(1) ClientKeyExchange message: If RSA is used, the Client Key Exchange message includes an encrypted pre-master secret which consists of a 48-bit number that is encrypted with the server's public-key.

If Diffie-Hellman is used, but not Fixed Diffie-Hellman, then the public-key parameters are sent here.

(2) CertificateVerify message: If the client sent a certificate, then it would send a CertificateVerify message at this point, in most cases. This would include a signature in the same format as defined for the ServerKeyMessage as well as an MD5 sum of all of the previous messages and a SHA hash of all of the previous messages.

(3) ChangeCipherSpec message: The Client sends the ChangeCipherSpec message indicating that all future traffic will be computed with the master secret. The random numbers and the pre-master secret are used by both systems in a pseudorandom function to calculate the master secret.

The change cipher spec protocol is a single byte that will always have a value of 1. It is encrypted and compressed under the current cipher (the pre-master secret) and with compression method.

(4) Finished message: Up to now, the client and server have negotiated the shared secret information known only to themselves. This value is a 48-byte quantity called the master secret.

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)).

The client now sends the Finished message. This consists of the master secret, the finished label, an MD5 of all previous messages and an SHA of all previous messages. All of this is encrypted with the master secret. If the server can read all of this, then the server knows that the key generation was successful. The Finished message is the first protected with the just-negotiated algorithms, keys, and secrets.

4) In Message 4 (*ChangeCipherSpec & Finished message*:)

The server responds with its own ChangeCipherSpec and Finished messages which verify to the client that the key generation was successful.

If any warning or fatal errors occur, an alert is sent. Alerts consist of a byte that defines whether it's a warning (1) or a fatal (2) alert, and a byte

that indicates the specific alert. The possible values for alerts are indicated in Table 5.2.

Table 5.2 Alerts in SSL

Fatal alerts	Not fatal alerts
unexpected_message (10)	close_notify (0)
bad_record_mac (20)	no_certificate_RESERVED (41) – this is SSL 3.0 only
decryption_failed (21)	bad_certificate (42)
record_overflow (22)	unsupported_certificate (43)
decompression_failure (30)	certificate_revoked (44)
handshake_failure (40)	certificate_expired (45)
illegal_parameter (47)	certificate_unknown (46)
unknown_ca (48)	decrypt_error (51)
access_denied (49)	no_renegotiation (100)
decode_error (50)	
export_restriction_RESERVED (60)	
protocol_version (70)	
insufficient_security (71)	
internal_error (80)	
user_canceled (90)	

Application data: The master secret is used to generate keys and secrets for encryption and MAC computations. To generate the key material, compute key_block until enough output has been generated, where

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)) + [...].

Then the key_block is partitioned as follows.

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] /* non-export ciphers */
server_write_IV[CipherSpec.IV_size] /* non-export ciphers */

Any extra key_block material is discarded.

Now that the keys and secrets are computed, data may be sent encapsulated inside record protocol. This data will be encrypted and compressed in the agreed upon methods and can be reliably read by the other end but not

likely anyone in-between.

5.1.3 Security analysis of SSL based on trusted freshness

SSL is helpful for enhancing the security of communications, however it is not as secure as it intends to. In this subsection, we give the security analysis of SSL based on trusted freshness, and some attacks are given from the absence of the protocol security properties. Moreover, attacks related to TLS renegotiation implementation are also briefly introduced.

5.1.3.1 Security analysis of SSL negotiation based on trusted freshness

Example 5.1 (A full SSL handshake protocol with both side certificates) When a new session begins, the CipherSpec encryption, hash, and compression algorithms are initialized to null. Figure 5.3 illustrates the message exchanges in SSL handshake protocol related to authentication and key establishment with the certification verifying both server side and client side.

Message 1 $C \rightarrow S : Ver_C, T_C, N_C, \ NULL$

Message 2 $S \rightarrow C : Ver_S, T_S, N_S, SID_S, Cert_S$

Message 3 $C \rightarrow S : \{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}}$

Message 4 $S \rightarrow C : \{S, N_S, N_C\}_{k_{CS}}$

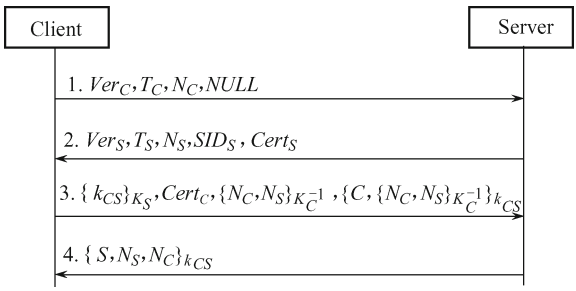


Fig. 5.3 Message in a full handshake of SSL with certifications of both sides.

Notation

C denotes a client (the client-side web browser), S denotes the web server. Ver_C is the protocol version in the client hello, and Ver_S is the version number being used (the lowering of the server's highest supported version and the version in the client hello). T_C and T_S are timestamps generated by C and S referring to an absolute time, where clocks are not required to be set correctly by the basic SSL Protocol, but higher level or application protocols may define additional requirements. N_C and N_S are nonces randomly chosen by C and S respectively. k_{CS} is a new session key between C and S to

be established in this authentication protocol (Note: we do not distinguish pre_master_secret, master_secret and other keys and secrets for encryption and MAC computations since they do not effect the security analysis of SSL). SID_S is a new session ID generated and cached by the Server. $Cert_C$ ($Cert_S$) is a certificate of C (or S) and corresponding public-key K_C (or K_S) signed by a trusted certificate authority CA. K_S and K_S^{-1} are public-key and private key of S, while K_C and K_C^{-1} are public-key and private key of C.

Premise

Both C and S know the public-key of the trusted certificate authority CA to get K_C and K_S. Each principal knows the key pair of himself, that is, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S.

Protocol actions

1) In Message 1, the client C starts the negotiation by sending the client SSL version Ver_C, the timestamp T_C, a randomly chosen new nonce N_C by C for this protocol run, a $NULL$ session identifier and the intended cryptographic algorithm for key exchange, the bulk encryption algorithm with secret key and length, and MAC (the chosen cipher suite does not effect the security analysis of SSL, hence omitted.).

2) Upon receiving Message 1, since the session ID is $NULL$, a full handshake is launched.

3) In Message 2, S generates a new session ID SID_S, and sends it to C with the version number Ver_S being used, the timestamp T_S, the randomly chosen new nonce N_S by S, and the certificate of S containing the name and the public-key.

4) Upon receiving Message 2, C randomly chooses a new session key k_{CS} for this protocol run, then encrypts it under K_S and sends it to C; C encrypts the identity of himself, the randomly chosen nonce N_C and N_S, and sends $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$ to S.

5) Upon receiving Message 3, S gets the new session key k_{CS} using S's private key K_S^{-1}, then decrypts $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$ using k_{CS} also K_C and verifies the correction of N_C and N_S.

6) In Message 4, S encrypts the identity of S, the randomly chosen nonce N_C and N_S, and sends $\{S, N_S, N_C\}_{k_{CS}}$ to C.

7) Upon receiving Message 4, C decrypts $\{S, N_S, N_C\}_{k_{CS}}$ using k_{CS} and verifies the correction of N_C and N_S.

Successful execution should convince C and S that k_{CS} is a new session key between C and S. Actually, this protocol has not achieved the key exchange and authentication security objects as it intends to.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_C, and C also believes that N_C is open.

2) Upon receiving Message 1, from Lemma 4.2, S believes that N_C is open.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, S has the freshness assurance of the randomly chosen nonce N_S, and S also believes that N_S is open.

4) Upon receiving Message 2, from Lemma 4.2, C believes that N_S is open.

5) In Message 3, from Lemma 4.2 and Lemma 4.3, C has the confidentiality and the freshness assurances of the new session key k_{CS}.

6) Upon receiving Message 3, from Lemma 4.3, S has the freshness assurance of the new session key k_{CS}. From Lemma 4.4, S has the association assurance of k_{CS} with C, since only C could sign $\{N_C, N_S\}_{K_C^{-1}}$ using its private key, and the identity of C is explicitly indicated in Message 3. Upon receiving $\{N_C, N_S\}_{K_C^{-1}}$, from Lemma 4.1, S has the liveness assurance of C based on the trusted freshness N_S.

7) Upon receiving Message 4, from Lemma 4.1, C has the liveness assurance of S. From Lemma 4.3, C has the freshness assurance of N_S. From Lemma 4.4, S has the association assurances of N_C, N_S and k_{CS} with S, since only S could get k_{CS} from the encryption $\{k_{CS}\}_{K_S}$ using its private key, and the identity of S is explicitly indicated in Message 4.

Upon termination of the protocol run, as indicated in Table 5.3, C believes that S is present, and the new session key k_{CS} is confidential, fresh, and associated with S, while S believes that C is present, and the new session key k_{CS} is confidential, fresh, and associated with C.

Table 5.3 Security analysis of the full handshake of SSL protocol

	C				S			
	S	N_C	N_S	k_{CS}	C	N_C	N_S	k_{CS}
Message 1		01#				0?#		
Message 2			0?#				01#	
Message 3				11#	1		01C	11C
Message 4	1	01S	01S	11S				
End of run	1			11S	1			11C

Example 5.2 (Attack-1 on a full SSL handshake protocol with both side certificates) From the absence of the association of k_{CS} with S in the point of view of S, we can construct an attack as shown in Fig. 5.4.

$$
\begin{aligned}
&\text{Message 1 } C \to I: && Ver_C, T_C, N_C, NULL\\
&\text{Message 1}' \; I(C) \to S: && Ver_C, T_C, N_C, NULL\\
&\text{Message 2}' \; S \to I(C): && Ver_S, T_S, N_S, SID_S, Cert_S\\
&\text{Message 2 } I \to C: && Ver_S, T_S, N_S, SID_S, Cert_I\\
&\text{Message 3 } C \to I: && \{k_{CS}\}_{K_I}, Cert_C, \{N_C, N_S\}_{K_C^{-1}},\\
& && \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}}
\end{aligned}
$$

$$\begin{aligned}
&\text{Message } 3' \quad I(C) \to S : \{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \\
&\qquad\qquad\qquad\qquad \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}} \\
&\text{Message } 4 \quad I \to C : \quad \{I, N_S, N_C\}_{k_{CS}} \\
&\text{Message } 4' \quad S \to I(C) : \{S, N_S, N_C\}_{k_{CS}}
\end{aligned}$$

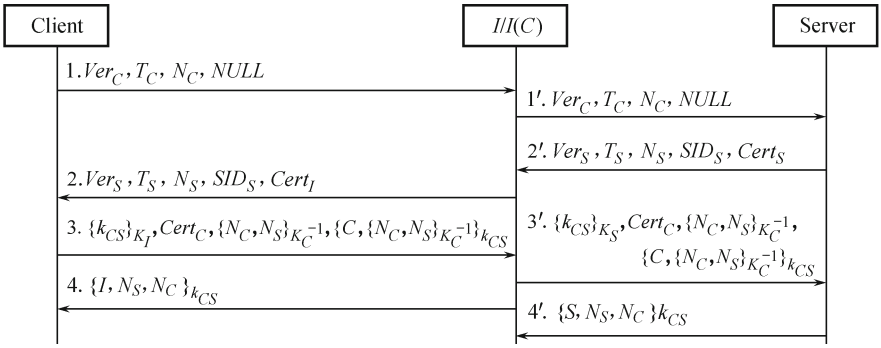


Fig. 5.4 An attack on a full handshake of SSL with certifications of both sides.

Notation

I denotes the adversary, and $I(C)$ is an adversary I impersonating C. $Cert_I$ is a certificate of I and corresponding public-key K_I signed by CA. K_I and K_I^{-1} are public-key and private key of I. Other notations are the same as the original SSL handshake protocol.

Premise

Each principal knows the public-key of the trusted certification authority center CA to get K_C, K_S and K_I. Each principal knows the key pair of himself, that is, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S, K_I and K_I^{-1} for I.

Protocol actions

1) In Message 1, the client C starts a protocol run with I. In Message $1'$, the adversary $I(C)$ replays the Message 1 to S to start a fake protocol run between C and S by impersonating C.

2) Upon receiving Message $1'$, S makes response to $I(C)$ with Message $2'$ including the certificate of S. I substitutes $Cert_S$ with $Cert_I$ in Message $2'$, then forwards Message 2 $\{\text{Ver}_S, T_S, N_S, SID_S, Cert_I\}$ to C.

3) Upon receiving Message 2, C randomly chooses a new session key k_{CS} for this protocol run between C and I. In Message 3, C generates $\{N_C, N_S\}_{K_C^{-1}}$ using C's private key K_C^{-1} to show that it is really C who has sent this message $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$.

4) Upon receiving Message 3, I gets the new session key k_{CS} using I's private key K_I^{-1}. Then, I encrypts k_{CS} with S's public-key K_S, generates Message $3'$ $\{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}$, and $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$, then

forwards it to S.

5) Upon receiving Message $3'$, S gets k_{CS} and verifies $\{N_C, N_S\}_{K_C^{-1}}$, then S believes that it must be C who is sharing the new session key k_{CS} with S. At the same time, I will complete his protocol run with C normally.

Upon termination of the attack on the full SSL handshake protocol with both side certificates, the adversary I causes S to have false beliefs: S has completed a successful protocol run with C, and is sharing a new session key k_{CS} with C, whereas in fact, C knows nothing about the key establishment with S, and actually C shares the key k_{CS} with I. Furthermore, S concludes that subsequently messages could be encrypted using k_{CS} and safely transmitted to C (actually known by I).

Example 5.3 (Attack-2 on a full SSL handshake protocol with both side certificates) From the absence of the association of k_{CS} with C in the point of view of C, we can construct an attack as shown in Fig. 5.5.

$$
\begin{array}{lll}
\text{Message } 1 & C \to I(S): & Ver_C, T_C, N_C, NULL \\
\text{Message } 1' & I(C) \to S: & Ver_C, T_C, N_C, NULL \\
\text{Message } 2' & S \to I(C): & Ver_S, T_S, N_S, SID_S, Cert_S \\
\text{Message } 2 & I(S) \to C: & Ver_S, T_S, N_S, SID_S, Cert_I \\
\text{Message } 3 & C \to I(S): & \{k_{CS}\}_{K_I}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \\
 & & \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}} \\
\text{Message } 3' & I(C) \to S: & \{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \\
 & & \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}} \\
\text{Message } 4 & I(S) \to C: & \{S, N_S, N_C\}_{k_{CS}} \\
\text{Message } 4' & S \to I(C): & \{S, N_S, N_C\}_{k_{CS}}
\end{array}
$$

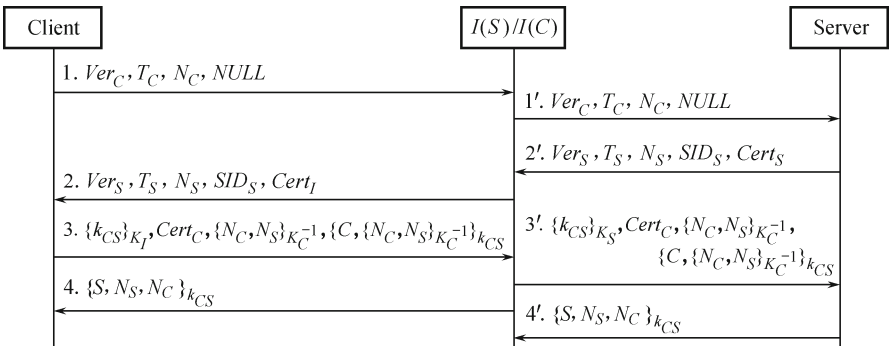


Fig. 5.5 Another attack on a full handshake of SSL with certifications of both sides.

Notation

I denotes the adversary, and $I(C)/I(S)$ denotes the adversary I impersonating C/S respectively. $Cert_I$ is a certificate of I and K_I signed by CA. K_I and K_I^{-1} are public-key and private key of I. Other notations are the same as the original SSL handshake protocol.

Premise

Each principal knows the public-key of the trusted certificate authority CA to get K_C, K_S and K_I. Each principal knows the key pair of himself, that is, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S, K_I and K_I^{-1} for I.

Protocol actions

1) In Message 1, the client C starts a protocol run with S. I intercepts Message 1, then forwards it to S as Message $1'$.

2) Upon receiving Message $1'$, S makes response to $I(C)$ with Message $2'$ including the certificate of S. I substitutes $Cert_S$ with $Cert_I$ in Message $2'$, then forwards Message 2 $\{Ver_S, T_S, N_S, SID_S, Cert_I\}$ to C.

3) Upon receiving Message 2, C randomly chooses a new session key k_{CS} for this protocol run between C and S (Actually, it is the adversary I impersonating S). In Message 3, C encrypts k_{CS} with S's public-key K_S (Actually, the public-key is deduced from $Cert_I$ in Message $2'$, hence it is the adversary I's public-key K_I), and generates Message 3 $\{\{k_{CS}\}_{K_I}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}\}$.

4) Upon receiving Message 3, I gets the new session key k_{CS} using I's private key K_I^{-1}. I encrypts k_{CS} with S's public-key K_S in Message $3'$, then forwards Message $3'$ to S.

5) In Message 4, I will complete his protocol run with C normally by impersonating S.

6) Upon receiving Message $4'$, S will complete his protocol run with C normally (Actually, it is the adversary I impersonating C).

Upon termination of the attack on the full SSL handshake protocol with both side certificates, the adversary I causes both sides to have false beliefs: each side has completed a successful protocol run with its opponent, and is sharing a new session key k_{CS} with the opponent, whereas in fact, the shared key k_{CS} between C and S is also known by I.

In an execution of the SSL Handshake Protocol, the client may be chosen to be anonymous and so is not authenticated to the server. As a result, the server unilaterally authenticates itself to the client. The output from the execution is a unilaterally authenticated channel from the server to the client. This is a typical example of using the SSL Protocol in a Web-based electronic commerce application, for example, buying a book from an online bookseller. The output channel assures the client that only the authenticated server will receive its instructions on book purchase which may include confidential information such as its user's bankcard details, the book title, and the delivery

address[13].

5.1.3.2 Attacks related to renegotiation

This protocol can be executed with all the optional messages and the ClientKeyExchange message omitted. This is the case when the client wants to resume an existing session. Session resumption can save the server time and CPU by obviating the need to do a full basic SSL negotiation. The ostensible reason for renegotiation is to allow either end to decide that it would like to refresh its cryptographic keys, increase the level of authentication, increase the strength of the cipher suite in use, and so forth. If the session ID in the ClientHello message is non-empty, the server will look in its session cache for a match. If a match is found and the server is willing to establish the new connection using the specified session state, the server will respond with the same session ID as supplied by the client. This indicates a resumed session and dictates that the parties must proceed directly to the finished messages, omitting all the optional messages and the ClientKeyExchange message. The renegotiation SSL handshake protocol structure is illustrated in Fig. 5.6.

$$
\begin{aligned}
&\text{Message 1} \quad C \rightarrow S: \ Ver_C, T_C, N_C, SID_{Existing} \\
&\text{Message 2} \quad S \rightarrow C: \ Ver_S, T_S, N_S, SID_{Existing} \\
&\text{Message 3} \quad C \rightarrow S: \{k_{CS}\}_{K_S}, \{C, N_C, N_S\}_{k_{CS}} \\
&\text{Message 4} \quad S \rightarrow C: \{S, N_S, N_C\}_{k_{CS}}
\end{aligned}
$$

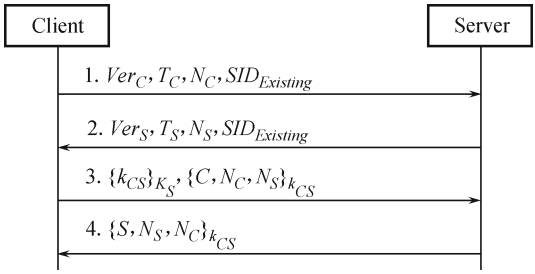


Fig. 5.6 Message exchanges in a renegotiation handshake of SSL.

Notation

$SID_{Existing}$ is the ID of an existing session stored in the server's session cache. Other notations are the same as the original SSL handshake protocol.

Premise

$SID_{Existing}$ is stored in the server's session cache and the server is willing to establish the new connection using the specified session state.

The premises are the same as the original SSL handshake protocol.

Protocol actions

1) In Message 1, if the client C wants to resume an existing session, then the client sends a ClientHello message using the Session ID of the session to be resumed.

2) Upon receiving Message 1, if the Session ID of the ClientHello Message is non-empty, the server will check its session cache for a match. If a match is found, and the server is willing to reestablish the connection under the specified session state, it will send a ServerHello Message with the same Session ID value. Otherwise this field will contain a different value identifying the new session.

3) In Message 3, C sends a randomly chosen new session key k_{CS} under K_S for this resumption, and also an encryption $\{C, N_C, N_S\}_{k_{CS}}$ of previously sent handshake messages under k_{CS}.

4) Upon receiving Message 3, S gets the new session key k_{CS} using S's private key K_S^{-1}, then decrypts $\{C, N_C, N_S\}_{k_{CS}}$ using k_{CS} and verifies the correction of N_C and N_S.

5) In Message 4, S encrypts the identity of himself, the randomly chosen nonce N_C and N_S, and sends $\{S, N_S, N_C\}_{k_{CS}}$ to C.

6) Upon receiving Message 4, C decrypts $\{S, N_S, N_C\}_{k_{CS}}$ using k_{CS} and verifies the correction of N_C and N_S.

Successful execution should convince C and S that k_{CS} is a new session key for this resumption. Actually, the security of the renegotiation process is even worse than the original full SSL handshake protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_C, and C also believes that N_C is open.

2) Upon receiving Message 1, from Lemma 4.2, S believes that N_C is open.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, S has the freshness assurance of the randomly chosen nonce N_S, and S also believes that N_S is open.

4) Upon receiving Message 2, from Lemma 4.2, C believes that N_S is open.

5) In Message 3, from Lemma 4.2 and Lemma 4.3, C has the confidentiality and the freshness assurances of the new session key k_{CS}.

6) Upon receiving Message 3, from Lemma 4.2 and Lemma 4.3, S has the confidentiality and the freshness assurances of the new session key k_{CS}.

7) Upon receiving Message 4, from Lemma 4.1, C has the liveness assurance of S. From Lemma 4.3, C has the freshness assurance of k_{CS}. From Lemma 4.4, S has the association assurances of N_C, N_S and k_{CS} with S, since only S could get k_{CS} from the encryption $\{k_{CS}\}_{K_S}$ using its private key, and the identity of S is explicitly indicated in Message 4.

Upon termination of the protocol run, we get analyzing result from Table 5.4, it means C believes that S is present, and the new session key k_{CS} is confidential, fresh, and associated with S, while S only believes that the new session key k_{CS} is confidential, fresh, but S does not know whether k_{CS} is associated with C and/or S or not.

Table 5.4 Security analysis of the full handshake of SSL protocol

	C				S			
	S	N_C	N_S	k_{CS}	C	N_C	N_S	k_{CS}
Message 1		01#				0?#		
Message 2			0?#				01#	
Message 3				11#				11#
Message 4	1	01S	01S	11S				
End of run	1			11S				11#

Example 5.4 (Attack-1 on SSL renegotiation) From the absence of the liveness of C in the point of view of S, we can construct an attack as shown in Fig. 5.7.

$$\begin{aligned}
&\text{Message 1} \quad I(C) \rightarrow S : Ver_C, T_C, N_C, SID_{Existing}\\
&\text{Message 2} \quad S \rightarrow I(C) : Ver_S, T_S, N_S, SID_{Existing}\\
&\text{Message 3} \quad I(C) \rightarrow S : \{k_{CS}\}_{K_S}, \{C, N_C, N_S\}_{k_{CS}}\\
&\text{Message 4} \quad S \rightarrow I(C) : \{S, N_S, N_C\}_{k_{CS}}
\end{aligned}$$

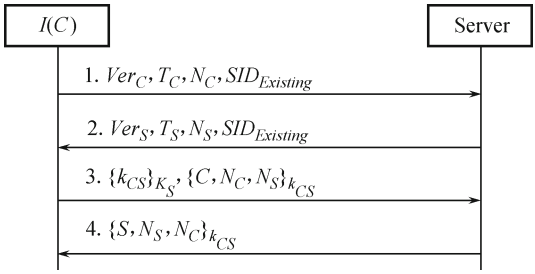


Fig. 5.7 An attack on the SSL renegotiation to cheat S.

Notation

$I(C)$ is an adversary I impersonating C. Other notations are the same as the original renegotiation SSL handshake protocol.

Premise

Premises are the same as the original renegotiation SSL handshake protocol.

Protocol actions

1) In Message 1, the adversary I starts SSL renegotiation to resume an existing session by impersonating C.

2) Upon receiving Message 1, S checks its session cache for a match and makes response to $I(C)$ with Message 2.

3) Upon receiving Message 2, $I(C)$ randomly chooses a new session key k_{CS} for this resumption between C (actually the adversary) and S, then encrypts k_{CS} under K_S and sends the encryption to S. $I(C)$ also sends the encryption of the previously sent handshake messages under k_{CS} to S.

4) Upon receiving Message 3, S gets the new session key k_{CS} using his private key K_S^{-1} and verifies $\{C, N_C, N_S\}_{k_{CS}}$. Then S believes that it must be C who is sharing the new session key k_{CS} with S.

Upon termination of the attack on the SSL renegotiation, the adversary I causes S to have false beliefs: S has completed a successful protocol run with C, and is sharing a new session key k_{CS} with C, whereas in fact, C knows nothing about the key establishment with S, and actually S shares the key k_{CS} with I. Furthermore, S concludes that subsequently messages could be encrypted using k_{CS} and safely transmitted to C (actually I).

The above attack can be launched directly by an adversary I impersonating a legitimate client C even without the liveness of C. This case is perhaps more attractive to the attacker because this characteristic permits the adversary to apply an attack on an intended victim at any time, and no particular client-side or server-side configuration is required for this attack to succeed.

5.1.3.3 Attacks related to TLS renegotiation implementation[14]

TLS (including RFC 5246 and previous ones, SSL v3 and previous ones) is subject to a number of serious man-in-the-middle (MITM) attacks related to renegotiation in real world. The TLS standard permits either end to request renegotiation of the TLS session at any time. The MITM attacks related to renegotiation are expected to generalize well to not only HTTPS but also other protocols layered on TLS. There are three general attacks related to renegotiation against HTTPS, each with slightly different characteristics, all of which yield the same result: the attacker is able to execute an HTTP transaction of his choice, authenticated by a legitimate user (the victim of the MITM attack).

Example 5.5 (Client certificate authentication renegotiation attack) The server cannot insist that the client provide a valid client certificate until it has received the certificate request from the client and filtered it through its authentication rules. For requests that are found to require client certificate authentication, the HTTPS server must then renegotiate the TLS channel to obtain and validate the certificate from the client. Unfortunately, because HTTP lacks a specific response code to instruct the client to resubmit the request within the newly authenticated channel, the server must apply the authentication retroactively to the original request. This "authentication gap" is the central weakness exploited by these attacks. Most existing installations which currently rely on client certificates for authentication appear to be vulnerable to this client certificate authentication renegotiation attack.

Example 5.6 (Differing server cryptographic requirements renegotiation attack) HTTPS servers that host resources with varying cipher suite requirements (this is often the case since web servers often host "secure" or "anonymous" content with varying certificate authentication requirements) may be vulnerable to another renegotiation attack. Because of the variations in the level of cipher suite strength, the web server has to be willing to negotiate TLS at the most basic encryption level supported on the server. Only after having seen the URL requested by the client can the server accurately determine which cipher suites will be acceptable. If the current cipher suite is not one of the required cipher suites, the server must request a renegotiation and agree on new parameters. The act of soliciting client renegotiation triggers the same weakness as in the case of client certificates: the server is forced to replay the buffered request, which in this case includes the chosen plaintext of the attacker. This has the effect of authorizing the transaction requested by the MITM.

Example 5.7 (Client-initiated renegotiation attack) TLS equally allows the client side of the connection to initiate a renegotiation. The MITM splices an initial request with an un-terminated HTTP "ignore" header onto the beginning of the client's intended request, again steals whatever authentication or authorization information provided. Note that this does not require pipelining or HTTP keep-alive. In all other respects, the server sees the same sort of request buffer as above. Most or all server applications built on TLS implementations which honor client-initiated renegotiation are vulnerable.

5.2 Internet Protocol Security

The Internet is an enormous open network of computers and devices, which are called "network nodes", each with unique Internet Protocol (IP) address. IPSec (Internet Protocol Security) is a suite of protocol designed by IETF[15, 16] to provide security for IPv4 and IPv6. The security services include confidentiality, data authentication, data integrity, and key management. IPSec provides securing communications over the Internet in key layer – the network layer of TCP/IP, hence the protection which covers the addressing information as well as the content can be very effective. In general, security at the IP layer can provide a wide protection on all applications at higher layers. Due to scalability and practical implementation considerations automatic key management seems a natural choice in significantly large virtual private networks (VPNs), IPSec has become standard by default of the most of the IP VPN technology in the world.

5.2.1 IPSec overview

IPSec has two modes: Transport mode (between two hosts) and Tunnel mode (between hosts/firewalls) and three sub protocols: Authentication Header (AH, RFC 2402[17]), Encapsulating Security Payload (ESP, RFC 2406[18]) and Internet Key Exchange Protocol (IKE, RFC 2409[19]). AH assures integrity protection, ESP provides encryption services and optional integrity protection while IKE allows communicating entities to derive session keys for secure communication via a series of messages exchange. IKE is the current IETF standard of authenticated key exchange protocol for IPSec, hence it is the concern of this book and it will be discussed in detail in Subsection 5.2.2.

5.2.1.1 Authentication Header (AH)

The Internet Protocol (IP) has evolved from version 4 (IPv4) to version 6 (IPv6). The data structure for IPv6 is a multiple of 32-bit data blocks called datagrams. In IPv6 with IPSec protection, an IP packet has an additional field called "Authentication Header" (AH)[17, 20]. Authentication protection (in fact, data integrity with origin identification) is a mandatory service for IPSec. The position for the AH in an IP packet (see Fig. 5.8, Fig. 5.9) is between "IP header" and the "TCP field". AH can have a variant length but must be a multiple of 32-bit datagrams which are organized into several subfields which contain data for providing cryptographic protection on the IP packet.

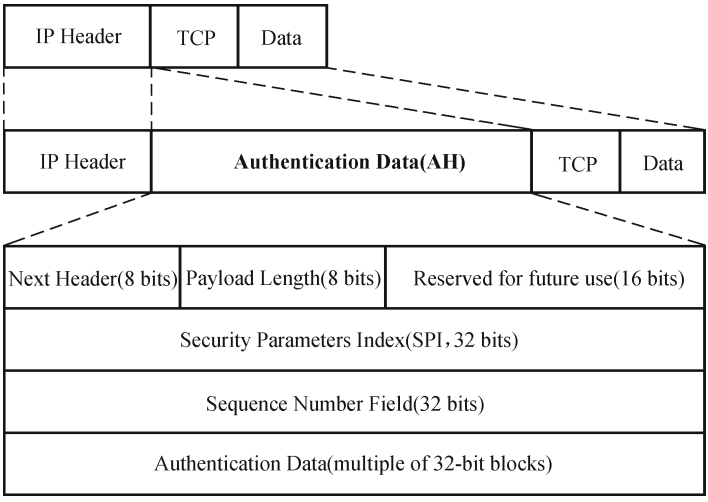


Fig. 5.8 Structure of an AH and its Position in an IP Packet in transport mode.

The subfield named "Security Parameters Index" (SPI) in an AH is an arbitrary 32-bit value which specifies (uniquely identifies) the cryptographic algorithms used for the authentication service for this IP packet. The sub-

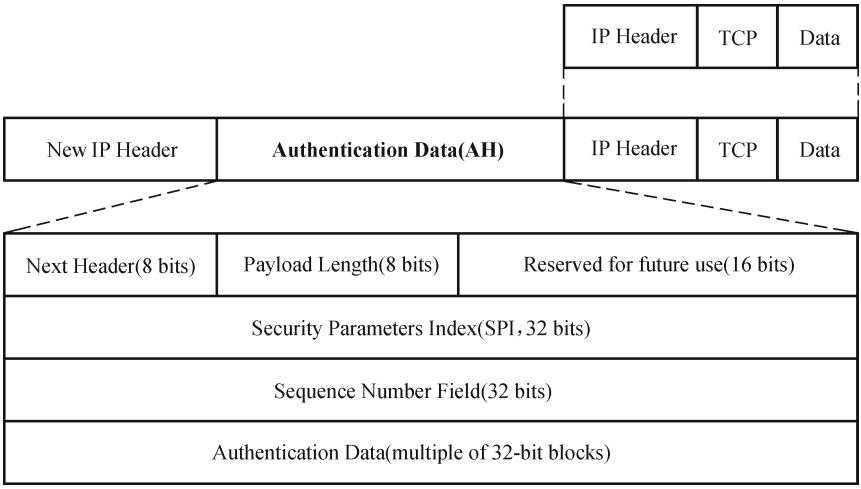


Fig. 5.9 The Structure of an AH and its Position in an IP Packet in tunnel mode.

field named "Sequence Number" can be used against replay of IP packets. The other subfield named "Authentication Data" (also called Integrity Check Value, ICV) in an AH contains the authentication data generated by the message sender for the message receiver to conduct data integrity verification. The receiver of the IP packet can use the algorithm uniquely identified in SPI and a secret key to regenerate "Authentication Data" and to compare with that received. End nodes have already established a shared secret session key manually or by IKE.

5.2.1.2 Encapsulating Security Payload (ESP)

Confidentiality (encryption) protection is an optional service for IPSec. To achieve this, a multiple of 32-bit datagrams named "Encapsulating Security Payload" (ESP)[18] is specified and allocated in an IP packet. The position for the ESP in an IP packet (see Fig. 5.10, Fig. 5.11) is between "IP header" (note that an ESP can follow an AH too) and the "TCP field". The format of an ESP is shown in Fig. 5.12.

The subfield named "Security Parameters Index" (SPI) in an ESP is an arbitrary 32-bit value which specifies (uniquely identifies) the encryption algorithm used for the confidentiality service for this IP packet.

The second subfield "Sequence Number" can be used against replay of IP packets.

The third subfield "Payload Data" has a variable length which is the ciphertext of the confidential data. Since an IP (v6) packet must have a length as a multiple of 32 bits, the plaintext "Payload Data" of variable length must be padded, and the paddings are given in "Padding". The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding

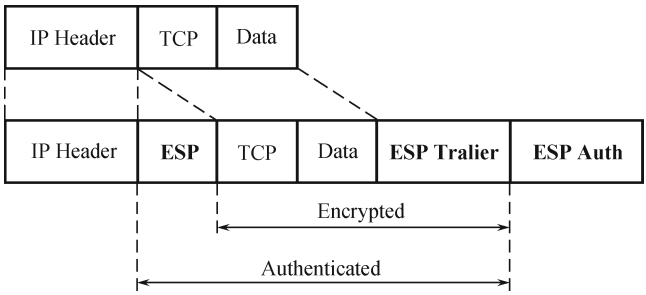


Fig. 5.10 An ESP and its Position in an IP Packet in transport mode.

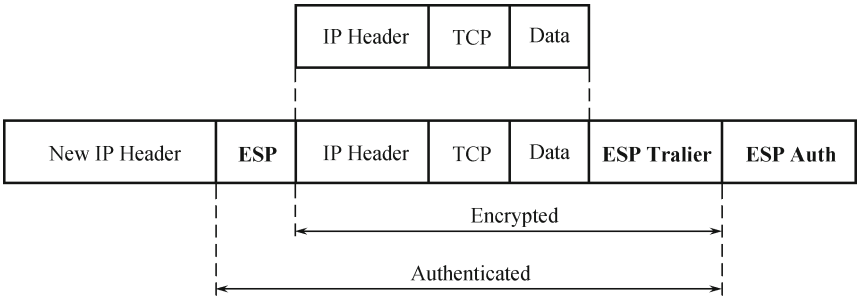


Fig. 5.11 An ESP and its Position in an IP Packet in tunnel mode.

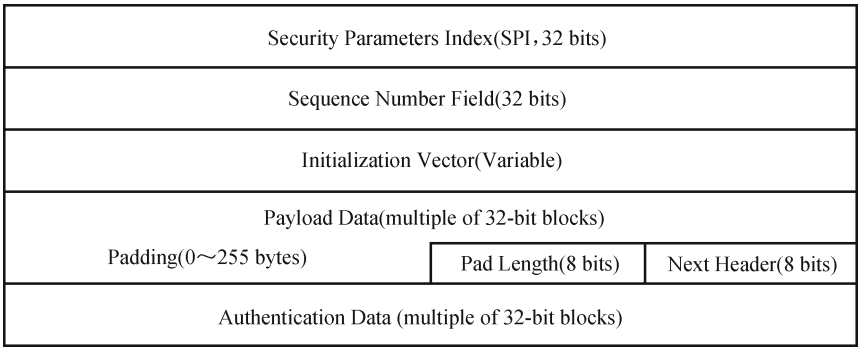


Fig. 5.12 The Structure of an Encapsulating Security Payload.

byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: $'01'||'02'||\cdots||'xy'$, where $'xy'$ is the hexadecimal value so that $'01' < 'xy' < 'FF'$. Therefore, the maximum number of the padding bytes is $'FF' = 255_{(10)}$. The length of the padding bytes is stated in "Pad Length".

The fourth subfield "Authentication Data" has the same meaning as that in an AH. However, "Authentication Data" in an ESP and that in an AH are

different. In an ESP, "Authentication Data" is for providing a data integrity protection on the ciphertext in the ESP packet, while in an AH, "Authentication Data" is for providing a data integrity protection on an IP packet.

5.2.2 Internet Key Exchange

Internet Key Exchange (IKE) is a set of protocols and mechanisms designed to perform two functions, creation of a protected environment (which includes authenticated peers that are unknown to each other in advance) and to establish and manage Security Associations (SA) between the authenticated peers, called the initiator and responder (or the receiver)[4, 19]. Both of the parties need to provide a digital signature in the key exchange protocol that the other party will verify. A successful verification means that the other party is authenticated. In order to be able to verify the signature also the public-key (certificate) needs to be trusted, and verified. SA defines how the traffic between the two hosts is to be protected.

Notation 5.1 (Security Associations (SA)) Each secure connection is called a security association (SA). A SA is simply a contract between two entities to provide a minimum set of services. It can be bi-directional or unidirectional. In case of unidirectional SA, which is often the case, we shall need two unidirectional SAs to complete one communication. With the view point of a programmer an SA can be considered as a data structure containing the information on Security Policy Index (SPI), its state (alive or expired), authentication algorithm, sequence number and SA life time. Considering globally, an SA is a set of proposals. A proposal can be thought of as a set of protocols and a protocol is, in turn, a set of transforms. A transform is a set of algorithms.

IKE is heart of the IPSec because it not only controls the services to be offered to secure the traffic but also manages the whole range of different transform options available at different levels and at different granularity. IKE architecture is based on three other protocols, namely, the Internet Security Association and Key Management Protocol (ISAKMP) [RFC 2408][21], the Oakley Key Determination Protocol (OAKLEY) [RFC 2412][22] and the Versatile Secure Key Exchange Mechanism for Internet (SKEME)[23].

ISAKMP provides a common framework for two communication parties to establish SA and cryptographic keys in an Internet environment. ISAKMP defines the procedures for authenticating a communicating peer, for creation and management of Security Associations, for key generation techniques, and for threat mitigation (e.g., denial of service and replay attacks). However ISAKMP does not define any specific key exchange technique so that it can support many different key exchange techniques.

OAKLEY describes a protocol by which two authenticated parties can agree on secure and secret keying material. The OAKLEY protocol supports Perfect Forward Secrecy, compatibility with the ISAKMP protocol for managing Security Associations, user-defined abstract group structures for use with the Diffie-Hellman algorithm, key updates, and incorporation of keys distributed via out-of-band mechanisms. The basic mechanism is the Diffie-Hellman key exchange algorithm.

SKEME describes an authenticated key exchange technique which supports deniability of connections between communication partners and quick key refreshment.

As a hybrid protocol of these work, IKE can be thought of as a suite of two-party protocols, featuring authenticated session key exchange, most of which in the suite use the Diffie-Hellman key exchange mechanism. IKE has many options for the two participants to negotiate and agree upon in an on-line fashion.

5.2.2.1 Two phases of IKE

IKE operates in two phases, namely, Phase 1 and Phase 2.

1. IKE Phase 1

The purpose of Phase 1 is to authenticate the communicating peers and generate the shared secret from which other keys will be computed. For IKE Phase 1, IKE has several modes, Main Mode, Aggressive Mode, Base Mode, New Group Mode, etc.

Phase 1 assumes that each of the two parties involved in a key exchange can verify the cryptographic capability of the other party, where capability might be enabled by a pre-shared secret key for a symmetric cryptosystem, or by a private key matching a reliable copy of a public-key for a public-key cryptosystem. Phase 1 attempts to achieve mutual authentication based on showing that cryptographic capability and establishes a shared session key from which other keys will be computed as an output from the IKE phases of exchanges.

2. IKE Phase 2

Phase 2 intends to create an IPSec security association and to generate new keys quickly, which relies on the shared session key agreed in Phase 1. All messages in this phase are made secure due to the algorithms and keys negotiated in Phase 1. This Create-Child-SA request may be launched by any party once Phase 1 is completed.

A multiple number of Phase 2 exchanges may take place between the same pair of entities involved in Phase 1. The reason for having a multiple number of Phase 2 exchanges is that they allow the users to set up multiple connections with different security properties, such as “integrity-only,” “confidentiality-only”, “encryption with a short key” or “encryption with a strong key”.

Notation 5.2 (IKE key material) SKEYID is a string derived from secret material known only to the active parties in the exchanges. The value SKEYID is computed separately for each authentication method and SKEYID is also a key seed of other keys.

For signature public-keys: $SKEYID = prf(nonces, g^{xy} \bmod p)$;

For encryption public-keys: $SKEYID = prf(hash(nonces), cookies)$;

For pre-shared secret key: $SKEYID = prf(pre\text{-}shared\ secret\ key,\ nonces)$.

The result of either Main Mode or Aggressive Mode is three groups of authenticated keying material:

For secret to generate other keys: $SKEYID\_d = prf(SKEYID, g^{xy}|cookies|0)$;

For integrity key: $SKEYID\_a = prf(SKEYID, SKEYID\_d|g^{xy}|cookies|1)$;

For encryption key: $SKEYID\_e = prf(SKEYID, SKEYID\_a|g^{xy}|cookies|2)$.

Here, $prf(key, msg)$ is the keyed pseudo-random function, while nonces and cookies are from the IKE exchanges between the initiator and the responder, and g^{xy} is the Diffie-Hellman shared secret.

5.2.2.2 IKE Modes

For IKE Phase 1, IKE has several modes, Main Mode, Aggressive Mode, Base Mode, New Group Mode, etc., which define how the actual key exchange procedure is to be done. Main Mode and Aggressive Mode are two of the most common modes. Main Mode (MM) has six message exchanges, and it should be run first in Phase 1, that is, two parties cannot run an aggressive mode without running a main mode first. Aggressive Mode (AM) has only three messages, and it is optional in Phase 1, that is, it can be omitted. There are four types of keys: pre-shared secret key, public encryption key (fields are separately encrypted using the public-key), optimized public encryption key (used to encrypt a random symmetric key, and then data is encrypted using the symmetric key) and public signature key (used only for signature purpose). For each key type there are two types of Phase 1 exchanges: a "main mode" and an "aggressive mode", hence there are 8 variants of IKE Phase 1.

For IKE Phase 2, IKE supports only one mode: the Quick Mode. The Quick Mode takes 3 packets to complete.

1. IKE Phase-1 Mode

The IKE Phase-1 is used to perform the mutual authentication, to exchange proposals, specific information and certificates. The result of the Phase-1 can be called in many ways: *Phase-1 SA*, *ISAKMP SA*, *IKE SA*, etc. They all mean the same thing. The Phase-1 SA is also used to protect the actual Phase-2 negotiation and other informational notifications that may be sent in IKE.

1) Phase-1 Main Mode The IKE negotiation always starts by executing the Main Mode in Phase-1 of the protocol. The Phase-1 Main Mode is

performed as shown in Fig. 5.13.

1. *Initiator* → *Responder* : *HDR, SA*
2. *Responder* → *Initiator* : *HDR, SA*
3. *Initiator* → *Responder* : *KE, NONCE, CR*
4. *Responder* → *Initiator* : *KE, NONCE, CR*
5. *Initiator* → *Responder* : *ID, *CERT, SIGR*
6. *Responder* → *Initiator* : *ID, *CERT, SIG*

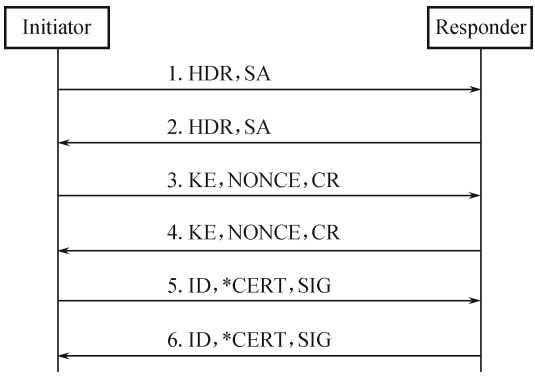


Fig. 5.13 Message flow for IKE Phase-1 Main Mode.

Notation

HDR is an ISAKMP header whose exchange type defines the IKE mode. SA is used to list the security properties supported by the initiator and the responder respectively. Key Exchange (KE) payload is the Diffie-Hellman public value, and is used to exchange the Diffie-Hellman public-keys. NONCE payload by the initiator or the responder respectively is used (with other information) in the IKE to compute the secret data for the Phase-1 SA. CR payload is certification request, and it includes the name of the CA (certification authority). ID payload is used to tell the other party who the sender or the responder is. The CERT payload includes the sender's or the responder's end entity certificate, and is also possible to send Certificate Revocation List (CRL). The signature payload (SIG) is the digital signature that the other party must verify. * indicates optional or situation-dependent messages that are not always sent.

Premise

CERT is a certification of the initiator or the responder and corresponding public-key signed by a trusted certificate authority *CA*.

Protocol actions

The first two messages negotiate policy; the next two exchange Diffie-

Hellman public values and ancillary data (e.g., nonces) necessary for the exchange, and the last two messages authenticate the Diffie-Hellman Exchange.

In Message 1: The initiator sends an HDR including an initiator cookie. The HDR is an ISAKMP header whose exchange type defines the IKE mode. An ISAKMP Header fields includes Initiator Cookie, Responder Cookie, Message ID etc. During Phase 1 negotiation, the initiator and responder cookies determine the ISAKMP SA. Message ID is a unique message identifier randomly generated by the initiator in Phase 2, which is used to identify protocol state during Phase 2 negotiation. This Message ID and the initiator's SPI(s) to be associated with each protocol in the proposal are sent to the responder. The SPI(s) will be used by the security protocols once the Phase 2 negotiation is completed. During Phase 1 negotiation, Message ID must be set to 0.

The SA payload is mandatory and it is used to list the security properties the initiator supports. It includes the ciphers, hash algorithms, key lengths, life time, and other information. It is possible to send only one SA payload in Phase-1.

In Message 2: The responder sends back an HDR including a responder cookie to the initiator. The responder must also include SA payload in its reply. The SA payload the responder sent includes the security properties it selected from the initiator's security property list (the SA payload).

Note that Message 1 and Message 2 are not encrypted, since there are no key to encrypt them with.

In Message 3: The IKE protocol is based on the Diffie-Hellman key exchange algorithm, which was the first ever invented algorithm that uses public-key cryptography (in 1976). Message 3 is used to exchange the Diffie-Hellman public-keys inside a Key Exchange (KE) payload. The Diffie-Hellman public-keys are created automatically every time the Phase-1 negotiation is performed, and they are destroyed automatically after the Phase-1 SA is destroyed.

There is also a NONCE payload that is generated by the sender and sent in Message 3 which is used (with other information) in the IKE to compute the secret data for the Phase-1 SA. The NONCE payload includes random data from random number generator.

The CR payload is a Certificate Request payload and is used to request for certificates by a specific CA. The CR payload includes the name of the CA for which it would like to receive the remote's end entity certificate (peer certificate). If empty CR payload is received, it means that it requests any certificate from any CA.

In Message 4: The responder also sends its Diffie-Hellman public-key, NONCE and CR payload in Message 4 to the initiator.

The CR payload is usually sent in the third and fourth packets, but it can be sent also in the first and second packets. Message 3 and Message 4 are not encrypted, since there is no key to encrypt them with.

In Message 5: The ID payload is used to tell the other party who the

sender is, and it also can be used to make policy decisions and to find the certificate of the remote end. The ID may be IP address, Fully Qualified Domain Name (FQDN), email address, or something similar.

The initiator may send zero (0) or more certificate payloads (CERT), with each including one certificate (or CRL). The CERT payload is optional payload, but usually if it is not sent, the result of the IKE is "Authentication Failed" error. The CERT payload includes the sender's end entity certificate, but it is also possible to send CRL inside a CERT payload. The CERT payload is optional because it is possible that the remote end has cached the public-key locally, and does not need to receive the CERT payload in the negotiation. Usually implementations do not cache it locally and in this case failing to send CERT payload also causes failure of the IKE negotiation.

The signature payload (SIG) is the digital signature that the other party must verify. The SIG payload includes the digital signature computed with the private key of the corresponding public-key (usually sent inside the CERT payload), and provides the authentication to the other party. When both of the parties successfully verify each other's SIG payloads, they are then mutually authenticated. They use the public-key found in the certificate (usually received in CERT payload, or some other means (cached locally, fetched from Lightweight Directory Access Protocol (LDAP), etc.)) to verify the signature. Before verifying the signature they also verify the certificate of the remote end. They check whether they trust the issuer (Certificate Authority, CA) of the certificate, and they also check whether the certificate is valid (not revoked, etc.) or not.

In Message 6: The responder also sends its ID, CERT and SIG payload in Message 6 to the initiator.

Note that Message 5 and Message 6 are fully encrypted, since the key was computed after Message 3 and Message 4 (where the Phase-1 SA is created and Diffie-Hellman public values is computed). When communication is protected, all payloads following the ISAKMP header MUST be encrypted.

After these packets are sent and the digital signatures are successfully verified the result of this Phase-1 negotiation is the Phase-1 SA, which can be used to protect other packets sent in the IKE, such as the packets of the Phase-2 negotiation. This also completes the Phase-1 negotiation successfully.

2) The Phase-1 Aggressive Mode The Phase-1 Aggressive Mode is performed as shown in Fig. 5.14.

1. Initiator → Responder: *VID, SA, KE, NONCE, ID*
2. Responder → Initiator: *VID, SA, KE, NONCE, ID, *CERT, SIG*
3. Initiator → Responder: **CERT, SIG*

Notation

VID means Vendor ID. Other notations are the same as in the IKE Phase-1 Aggressive Mode

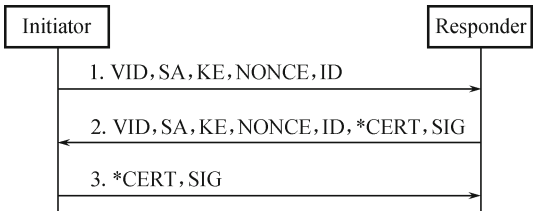


Fig. 5.14 Message flow for IKE Phase-1 Aggressive Mode.

Premise

CERT is a certificate of the initiator or the responder and its corresponding public-key signed by a trusted certificate authority *CA*.

Protocol actions

1) In Message 1: The initiator sends the crypto proposal supported, exchanges Diffie-Hellman public values and ancillary data necessary for the exchange, and identities.

2) In Message 2: The responder selects the crypto supported from the initiator's security property list (the SA payload), exchanges Diffie-Hellman public values and ancillary data necessary for the exchange, and identities. In addition Message 2 authenticates the responder.

3) In Message 3: Message 3 authenticates the initiator and provides a proof of participation in the exchange.

2. The IKE Phase-2 Mode

After the Phase-1 is successfully completed the Phase-2 negotiation can proceed. The purpose of the Phase-2 exchange is to provide and to refresh the key material that is used to create the Security Associations (SAs) to protect the actual IP traffic with IPSEC. The Phase-2 exchange is protected by encrypting the Phase-2 packets with the key material derived from the Phase-1. The Phase-2 also provides proposal list which defines the actual ciphers, HMACs, hash algorithms and other security properties that are used in the protection of the IP traffic. The proposal proposed in the Phase-1 is merely for protection of traffic under the Phase-1 SA (like the packets of the Phase-2), and not for the actual IP traffic. The Phase-2, also called the Quick Mode, is for the protection of the actual IP traffic. Since the ISAKMP SA is bi-directional, either communication party may initiate Quick Mode. The Phase-2 Quick Mode is performed as shown in Fig. 5.15.

Notation

HASH payload is the keying material exchanged. The SA payload is the Phase-2 proposal list which indicates the security properties. The NONCE payload includes always random data. The KE payload is from the ephemeral Diffie-Hellman exchange of Phase-1 Main Mode, and is confidential in Quick Mode. The ID payload is the participant's ID, usually IP address. * indicates

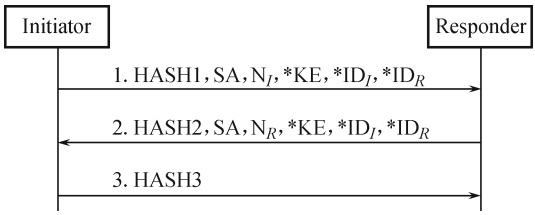


Fig. 5.15 Message flow for IKE Phase-2 Quick Mode.

optional or situation-dependent messages that are not always sent.

Premise

Diffie-Hellman public values have already exchanged in Phase 1.

Protocol actions

1) In Message 1: The initiator sends SA, HASH, NONCE and ID payloads to the responder to generate a new session key for IP traffic.

The SA payload is the Phase-2 proposal list which includes the ciphers, HMACs, hash algorithms, life times, key lengths, the IPSEC encapsulation mode (ESP, AH etc.) and other security properties. Note that it is possible to send more than one SA payloads in Phase-2, although usually only one is sent.

The HASH and NONCE payloads (marked here as N_I) are the keying material which are exchanged, and then are used to create the new key pair. The NONCE payload includes always random data.

The ID payloads, marked as ID_I and ID_R, for initiator's ID and responder's ID, respectively, are optional in Phase-2. Usually IKE implementations do send the ID payloads in Phase-2 since they can be easily used to make local policy decisions. However, as noted, they are not mandatory and can be omitted. The ID_I is the initiator's ID, usually IP address or similar, and the ID_R is the responder's ID, usually IP address, IP range or IP subnet. Both of the initiator and responder usually use the ID payloads to search the local policy for matching connection. The ID payloads in the Phase-2 are also called "proxy IDs", "pseudo IDs" or similar, since they do not necessarily represent the actual negotiator (for example when Security Gateway (SGW) negotiates on behalf of some client).

2) In Message 2: The responder selects the crypto from the Phase-2 proposal list, and sends HASH, NONCE (marked here as N_R) and ID payloads.

3) In Message 3: The initiator sends a HASH payload.

After the Phase 2 has been completed by sending the last packet, the result of the Phase-2 is two Security Associations (SAs). One is for inbound traffic, the other is for outbound traffic. This also completes the IKE key exchange for basic key exchange.

Note that all messages in Phase 2 are encrypted using $SKEYID\_e$, and integrity protected using $SKEYID\_a$.

5.2.3 Security analysis of IKE based on trusted freshness

The examples in this subsection assume that digital signatures are used in authentication and key establishment of IKE.

5.2.3.1 Security analysis of main mode

Example 5.8 (Public Signature Keys, IKE Phase-1 Main Mode) Figure 5.16 illustrates the message exchanges related to authentication and key establishment in public signature key-based IKE Phase-1 Main Mode, and some minute details are omitted.

Message 1 $A \rightarrow B:\ HDR_A, SA_A$

Message 2 $B \rightarrow A:\ HDR_B, SA_B$

Message 3 $A \rightarrow B:\ HDR_A, g^x, N_A$

Message 4 $B \rightarrow A:\ HDR_B, g^y, N_B$

Message 5 $A \rightarrow B:\ HDR_A, \left\{ ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}} \right\}_{SKEYID}$

Message 6 $B \rightarrow A:\ HDR_B, \left\{ ID_B, Cert_B, \{HASH_B\}_{K_B^{-1}} \right\}_{SKEYID}$

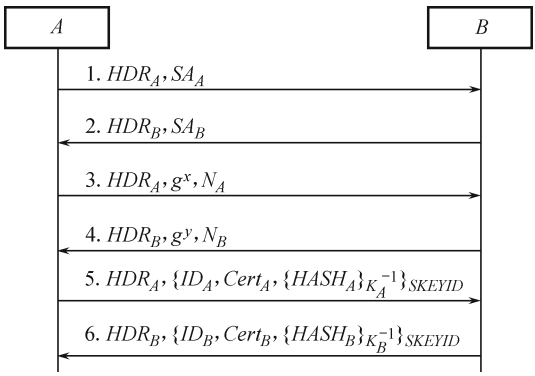


Fig. 5.16 Message exchanges in signature-based IKE phase-1 main mode.

Notation

A denotes the initiator, B denotes the responder.

HDR_A and HDR_B, ISAKMP headers of A and B, respectively are for keeping the session state information for these two entities.

SA_A, SA_B are the Security Associations (SA) of A and B, respectively. A and B use SA_A, SA_B to negotiate parameters to be used in the current run of the protocol: encryption algorithms, signature algorithms, pseudo-random functions for hashing messages to be signed, etc. A may propose a set of proposals, whereas B must reply with only one choice.

x and y are the private values randomly chosen by the initiator A and the responder B respectively. g^x, g^y are the Diffie-Hellman public values of A and B respectively. The g^{xy} can be computed via $(g^y)^x$ or $(g^x)^y$ by A and B, respectively.

N_A and N_B are nonces randomly chosen by A and B, respectively.

ID_A and ID_B are endpoint identities of A and B, respectively.

$Cert_A$ and $Cert_B$ are certifications of A and B issued by a trusted certificate authority CA, respectively.

$HASH_A$ and $HASH_B$ are hash values computed by A and B, respectively. The entire ID payload (including ID type, port, and protocol but excluding the generic header HDR) is hashed into both $HASH_A$ and $HASH_B$.

$$HASH_A = prf_1\,(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$$
$$HASH_B = prf_1\,(SKEYID|g^y|g^x|C_B|C_A|SA_B|ID_B)$$

where $SKEYID$ is the new session key between A and B, which can be computed as $SKEYID = prf_2(N_A|N_B|g^{xy})$. C_A and C_B are the initiator's and the responder's cookie respectively. prf_1 and prf_2 are pseudo-random functions agreed in SAs. For A and B to authenticate each other, the mutually obtainable hash values $HASH_A$ and $HASH_B$ will be signed by A and B respectively via the negotiated digital signature algorithm.

Note that we do not distinguish $SKEYID$ from $SKEYID\_d$, $SKEYID\_a$, $SKEYID\_e$, etc., since all these keys can be derived from $SKEYID$ and some open key materials.

Premise

Both A and B trust the issuer CA of the certificates $Cert_A$ and $Cert_B$, and they could verify that the certificate is valid (not revoked, etc.). That is, they know the public-key of CA to get K_A of A and K_B of B. Each principal knows the key pair of himself, that is, K_A and K_A^{-1} for A, K_B and K_B^{-1} for B.

Protocol actions

1) In Message 1, the initiator A starts the negotiation by sending an HDR_A including a null Message ID, an initiator cookie C_A, and an SA_A including encryption algorithms, signature algorithms, pseudo-random functions for hashing messages to be signed, key lengths, life time, and other information.

2) Upon receiving Message 1, since the Message ID is NULL, then a Phase-1 Main Mode is applied.

3) In Message 2, the responder B sends back an HDR_B including a responder cookie C_B and also the initiator cookie C_A to A. The responder also includes an SA_B in its reply to indicate the security properties B selects.

4) In Message 3, A randomly chooses a Diffie-Hellman private value x and a nonce N_A for this run. Then A computes the Diffie-Hellman public-key g^x and sends it to the responder B with N_A.

5) In Message 4, B also randomly chooses a Diffie-Hellman private value y and a nonce N_B for this run. Then B computes the Diffie-Hellman public value g^y and sends it to the initiator A with N_B.

Up to now, they have exchanged the Diffie-Hellman public values and ancillary data (e.g., nonces) necessary, and both A and B could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$.

6) In Message 5, A computes the hash of the entire ID payload $HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$, signs $HASH_A$ using A's private key K_A^{-1} to show his identity of A, and then encrypts all payloads following the ISAKMP header $\{ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}}\}$ using the negotiated new session key $SKEYID$.

7) Upon receiving Message 5, B decrypts $\{ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}}\}_{SKEYID}$ to get $\{ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}}\}$ using $SKEYID$, verifies $Cert_A$ to get K_A, and then checks the correction of the hash value:

$$\left\{\{HASH_A\}_{K_A^{-1}}\right\}_{K_A} = HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A).$$

8) In Message 6, B does similar things as A has done in Message 5. Upon receiving Message 6, A does similar things as B has done upon receiving Message 5.

Successful execution should authenticate the identities of the communication parties A and B, and establish a new session key $SKEYID$ between A and B. Actually, Phase-1 Main Mode l suffers from a flaw which has been proved by Lowe, Meadows and Mao respectively[5, 24, 25].

Protocol security analysis

1) In Message 1 and Message 2, neither A nor B could draw any useful assurance from it since there is not any trusted freshness identifier from the point of view of the two parties.

2) In Message 3, from Lemma 4.2 and Lemma 4.3, A has the freshness assurance of the randomly chosen Diffie-Hellman private value x and the nonce N_A, and A also believes that x is confidential and N_A is open.

3) Upon receiving Message 3, from Lemma 4.2, B believes that N_A is open and x is confidential.

4) Similarly, in Message 4, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the randomly chosen Diffie-Hellman private value y, and the nonce N_B is fresh and open. Further, B could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$, and from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of $SKEYID$.

5) Upon receiving Message 4, from Lemma 4.2, A believes that N_B is open and y is confidential. Further, A could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$, and from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of $SKEYID$.

6) In Message 5, A could not get any new assurance.

7) Upon receiving Message 5, from Lemma 4.1, B has the liveness assurance of A based on the trusted freshness N_B and $SKEYID = prf_2(N_A|N_B|g^{xy})$, since it must be A who has signed the fresh hash value $HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$ using A's private key. From Lemma 4.4, B has the association assurance of $SKEYID$ with A, since only A could sign $HASH_A$ using A's private key, and the identity of A is explicitly indicated in Message 5.

8) Similarly, in Message 6, B could not get any new assurance. Upon receiving Message 6, A has the liveness assurance of B and the association assurance of $SKEYID$ with B.

Upon termination of the protocol run, the analyzing result from Table 5.5shows that A believes that B is present, and the new session key $SKEYID$ is confidential, fresh, and associated with B, while B believes that A is present, and the new session key $SKEYID$ is confidential, fresh, and associated with A.

Table 5.5 Security analysis of the IKE phase-1 main mode

	A						B					
	B	N_A	N_B	x	y	$SKEYID$	A	N_A	N_B	x	y	$SKEYID$
Message 1												
Message 2												
Message 3		01#		11#				0?#		1?#		
Message 4			0?#		1?#	11#			01#		11#	11#
Message 5							1					11A
Message 6	1					11B						
End of run	1					11B	1					11A

Example 5.9 (Attack-1 on Signature-based, IKE Phase-1 Main Mode) From the absence of the association of $SKEYID$ with B in the point of view of B, there exists an attack[5, 24, 25], as shown in Fig. 5.17.

$$
\begin{aligned}
&\text{Message } 1 \quad A \to I: && HDR_A, SA_A\\
&\text{Message } 1' \quad I(A) \to B: && HDR_A, SA_A\\
&\text{Message } 2' \quad B \to I(A): && HDR_B, SA_B\\
&\text{Message } 2 \quad I \to A: && HDR_B, SA_B\\
&\text{Message } 3 \quad A \to I: && HDR_A, g^x, N_A\\
&\text{Message } 3' \quad I(A) \to B: && HDR_A, g^x, N_A\\
&\text{Message } 4' \quad B \to I(A): && HDR_B, g^y, N_B\\
&\text{Message } 4 \quad I \to A: && HDR_B, g^y, N_B\\
&\text{Message } 5 \quad A \to I: && HDR_A, \left\{ ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}} \right\}_{SKEYID}\\
&\text{Message } 5' \quad I(A) \to B: && HDR_A, \left\{ ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}} \right\}_{SKEYID}
\end{aligned}
$$

Message 6′ $B \rightarrow I(A): \ HDR_B, \left\{ ID_B, Cert_B, \{HASH_B\}_{K_B^{-1}} \right\}_{SKEYID}$

Message 6 $I \rightarrow A:$ Dropped

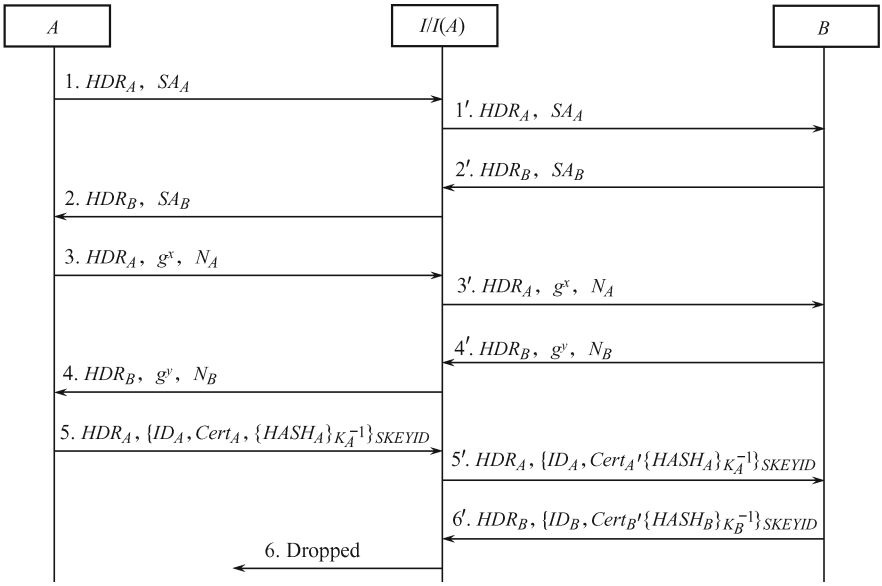


Fig. 5.17 An attack on the signature-based IKE phase-1 main mode.

Notation

I denotes the adversary, and $I(A)$ is an adversary I impersonating the initiator A. Other notations are the same as the original IKE phase-1 main mode.

Premise

The premises are the same as the original IKE phase-1 main mode.

Protocol actions

1) In Message 1, the initiator A starts a protocol run with I. In Message $1'$, the adversary $I(A)$ replays the Message 1 to B to start a fake protocol run between A and B by impersonating B. B responds to Message $2'$.

2) Upon receiving Message $2'$ and so on, the adversary I just replays all the follwing messages to the initiator A or the responder B respectively until receiving Message $6'$.

3) Upon receiving Message $6'$, the adversary I just drops this message.

Upon termination of the attack on the IKE Phase-1 Main Mode with both side certificates, the adversary I causes B to have false beliefs: B has completed a successful protocol run with A, and is sharing a new session key *SKEYID* with A, whereas in fact, A knows nothing about the key establish-

ment with B, while A thinks that A has been talking with I in an incomplete run. Furthermore, B will never be notified of any abnormality and B will keep the session state information for these two entities A (Actually the adversary I) and B. This is effective for the adversary I to make a denial of service attacks[24].

Note that the adversary I could not launch an attack similar to Example 5.3, since the $HASH_A$ includes the confidential and fresh seed of *SKEYID*.

5.2.3.2 Security analysis of aggressive mode

Example 5.10 (Public signature keys, IKE phase-1 aggressive mode) Aggressive Mode is a cut-down simplification from IKE Phase-1 Main Mode, and Fig. 5.18 shows the message exchanges with some minute details omitted. The first two messages negotiate policy, exchange Diffie-Hellman public values and ancillary data necessary for the exchange, and identities. In addition, the second message authenticates the responder. The third message authenticates the initiator and provides a proof of participation in the exchange.

Aggressive Mode can be used to reduce round trips even further.

Message 1 $A \rightarrow B:\ HDR_A, SA_A, g^x, N_A, ID_A$

Message 2 $B \rightarrow A:\ HDR_B, SA_B, g^y, N_B, ID_B, [Cert_B], \{HASH_B\}_{K_B^{-1}}$

Message 3 $A \rightarrow B:\ [Cert_A], \{HASH_A\}_{K_A^{-1}}$

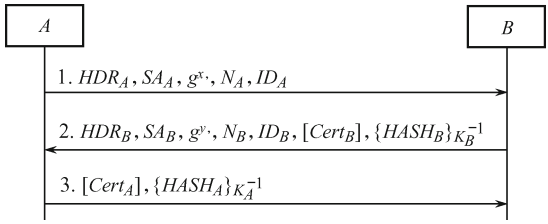


Fig. 5.18 Message exchanges in signature-based IKE phase-1 aggressive mode.

[x] indicates that x is optional, other notations, premise and protocol actions in this mode are the same as those in Main Mode (Example 5.8), hence omitted.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the freshness assurance of the private value x and the nonce N_A, and A also believes that x is confidential and N_A is open.

2) Upon receiving Message 1, from Lemma 4.2, B believes that N_A is open and x is confidential.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the private value y, and the nonce N_B

is fresh and open. Further, B could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$, and from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of $SKEYID$.

4) Upon receiving Message 2, from Lemma 4.1, A has the liveness assurance of B based on the trusted freshness N_A and $SKEYID = prf_2(N_A|N_B|g^{xy})$, since it must be B who has signed the fresh hash value $HASH_B = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_B|ID_B)$ using B's private key. From Lemma 4.4, A has the association assurance of $SKEYID$ with B, since only B could sign $HASH_B$ using B's private key, and the identity of B is explicitly indicated in Message 2.

5) Similarly, upon receiving Message 3, from Lemma 4.2 and Lemma 4.3, B has the liveness assurance of A based on the trusted freshness N_B and $SKEYID = prf_2(N_A|N_B|g^{xy})$, since it must be A who has signed the fresh hash value $HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$ using A's private key. From Lemma 4.4, B has the association assurance of $SKEYID$ with A, since only A could sign $HASH_A$ using A's private key, and the identity of A is explicitly indicated in Message 3.

Upon termination of the protocol run, the analyzing result is indicated in Table 5.6, it means A believes that B is present, and the new session key $SKEYID$ is confidential, fresh, and associated with B, while B believes that A is present, and the new session key $SKEYID$ is confidential, fresh, and associated with A.

Table 5.6 Security analysis of the IKE phase-1 aggressive mode

	A						B					
	B	N_A	N_B	x	y	$SKEYID$	A	N_A	N_B	x	y	$SKEYID$
Message 1		01#		11#				0?#		1?#		
Message 2	1		0?#		1?#	11B			01#		11#	11#
Message 3							1					11A
End of run	1					11B	1					11A

Note that $HASH_A$ and $HASH_B$ take the new session key $SKEYID$ as its seed input to pseudo-random function, hence the signature of these two hash values could not be faked, and the signatures are exclusively verifiable by the principals who hold the agreed session key. However, the attack (Example 5.9 Attack-1) on IKE main mode is still effect on the IKE aggressive mode.

5.2.3.3 Security analysis of quick mode

Example 5.11 (Public signature keys, IKE phase-2 quick mode) Each instance of a Quick Mode uses a unique initialization vector (e.g., Message ID), hence it is possible to have multiple simultaneous Quick Modes, based on a single ISAKMP SA, in progress at any one time. Quick Mode is essentially an SA negotiation and exchanges of nonces that provides replay protection. The nonces are used to generate fresh key material and to prevent replay attacks from generating bogus security associations. Base Quick Mode (without the

KE payload) refreshes the keying material derived from the exponentiation in Phase-1. Figure 5.19 illustrates the Quick Mode exchanges related to security.

Message 1 $A \rightarrow B:\ HDR_A, \{HASH_1, SA_A, N_A, [g^x], [ID_A, ID_B]\}_{SKEYID}$

Message 2 $B \rightarrow A:\ HDR_B, \{HASH_2, SA_B, N_B, [g^y], [ID_A, ID_B]\}_{SKEYID}$

Message 3 $A \rightarrow B:\ HDR_A, \{HASH_3\}_{SKEYID}$

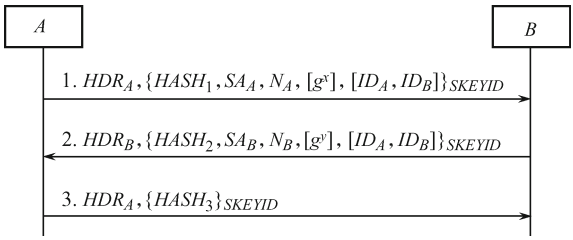


Fig. 5.19 Message exchanges in signature-based IKE phase-2 quick mode.

Notation

[x] indicates that x is optional, other notations in Quick Mode are the same as those in Main Mode except the following:

k is a new session key generated from this Quick Mode run.

$$k = prf(SKEYID\_d, g_{qm}^{xy} | protocol | SPI | N_A | N_B)$$

where g_{qm}^{xy} is the shared secret from the ephemeral Diffie-Hellman exchange of this Quick Mode. "*protocol*" and "*SPI*" are from the ISAKMP proposal payload that contains the negotiated transform.

$HASH_1$ and $HASH_3$ are hash values computed by A, while $HASH_2$ by B:

$$HASH_1 = prf(SKEYID\_a, ID_{AB} | SA_A | N_A [| g^x][| ID_A | ID_B])$$
$$HASH_2 = prf(SKEYID\_a, ID_{AB} | N_A | SA_B | N_B [| g^y][| ID_A | ID_B])$$
$$HASH_3 = prf(SKEYID\_a, 0 | ID_{AB} | N_A | N_B)$$

ID_{AB} is a message ID to indicate a Quick Mode in progress for a particular ISAKMP SA, and this particular SA is identified by the cookies in the ISAKMP header.

Premise

g^{xy} is from the ephemeral Diffie-Hellman exchange of Phase-1 Main Mode, and is confidential in Quick Mode. Suppose the key $SKEYID\_a$ from Main Mode is confidential and only known by A and B. Other premises are the same as those in Main Mode.

Protocol actions

Omitted for interest of concision.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the freshness assurance of the nonce N_A, and the confidentiality and freshness assurances of the value x. From Lemma 4.4, A has the association assurance of N_A and x with A and B, since the key $SKEYID\_a$ from Main Mode is confidential and only known by A and B.

2) Upon receiving Message 1, B could not draw any useful assurance from Message 1 since there is not any trusted freshness identifier from the point of view of B.

3) Similarly, in Message 2, from Lemma 4.2 and Lemma 4.3, B has the freshness assurance of the nonce N_B, and the confidentiality and freshness assurances of both y and k. From Lemma 4.4, B has the association assurance of N_B, y and k with A and B.

4) Upon receiving Message 2, from Lemma 4.2, Lemma 4.3, and Lemma 4.4, A believes that k is confidential, fresh and associated with A and B based on the trusted freshness N_A. From Lemma 4.1, A has the liveness assurance of B based on the trusted freshness N_A, since it must be B who has generated the fresh hash value $HASH_2 = prf(SKEYID\_a, ID_{AB}|N_A|SA_B|N_B[|g^y][|ID_A|ID_B])$ using the shared key $SKEYID\_a$.

5) Similarly, upon receiving Message 3, from Lemma 4.1, B has the liveness assurance of A from the hash value $HASH_3 = prf(SKEYID\_a, 0|ID_{AB}|N_A|N_B)$ including the trusted freshness N_B.

Table 5.7 shows the analyzing result of the IKE phase-2 quick mode. Upon termination of the protocol run, with the premise that the g^{xy} from Main Mode is confidential in Quick Mode, the protocol achieves that A believes that B is present, and the new session key k is confidential, fresh, and associated with both A and B, while B believes that A is present, and the new session key k is confidential, fresh, and associated with A. Hence, IKE Phase-2 Quick Mode has achieved the key exchange and authentication security objects as it intends to.

Table 5.7 Security analysis of the IKE phase-2 quick mode

	A						B					
	B	N_A	N_B	x	y	k	A	N_A	N_B	x	y	k
Message 1		$01AB$		$11AB$				$0?\#$		$1?\#$		
Message 2	1		$0?\#$		$1?\#$	$11AB$			$01AB$		$11AB$	$11AB$
Message 3							1					
End of run	1					$11AB$	1					$11AB$

However, we should note that if g^{xy} from Main Mode was not confidential in Quick Mode, that is, if g^{xy} was known by the adversary, then the adversary could initiate a Quick Mode instance at any one time by impersonating A or

B, and shared a new session key with the victim.

5.3 Kerberos — the network authentication protocol

In open network computing environments, a user (an employee, a subscriber or a customer) may be provided with various kinds of information resources and services: remote hosts, file servers, printers, and many other networked services. When a user requests use of a network service, the service provider wants an assurance that the user is who he says he is in a physically insecure network. However, it would be unrealistic and uneconomic to require a user to maintain several different cryptographic assurances, no matter whether in terms of memorizing various passwords, or in terms of holding a number of smartcards. Furthermore, unencrypted passwords sent over the network may suffer from the "sniff" attack. Hence, Kerberos[6, 7] is introduced by Massachusetts Institute of Technology (MIT) as a solution to these network security problems: a trusted third party mediates between a user and a resource server by issuing a shared session key between the two entities. The Kerberos protocol allows a legitimate user to log onto his terminal once a day (typically) and then transparently access all the networked resources he needs for the rest of that day. Each time the legitimate user wants to access an information resource, to retrieve a file from a remote server for example, Kerberos will securely handle the required authentication behind the scene without any user intervention.

5.3.1 Kerberos overview

Kerberos is developed as part of the MIT Athena project. The Kerberos protocol is designed to provide strong authentication so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and a server have proved their identities via Kerberos, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business. Furthermore, Kerberos is being used as a building block for higher-level protocols. Kerberized applications are those that use the Kerberos authentication protocol to provide authentication, and to provide encryption and signing for subsequent messages. Kerberos relies on conventional encryption rather than public-key encryption, that is, it uses private-key cryptography. PKINIT[26], which adds public-key authentication and a fair amount of complexity to the basic protocol, is an extension of Kerberos 5.

In basic Kerberos, a session generally starts with a user logging onto a system. This triggers the creation of a client process that will transparently

handle all his authentication requests. The client process — usually acting for a human user – interacts with three other types of principals when using Kerberos 5. The initial authentication between the client and the Kerberos administrative principal is traditionally based on a shared key derived from a password chosen by the user. The client's goal is to be able to authenticate himself/herself to various application servers (e.g., email, file, and print servers). This is done by obtaining a "Ticket-granting ticket" (TGT) from a "Kerberos Authentication Server" (AS) and then presenting this TGT to a "Ticket-Granting Server" (TGS) in order to obtain a "Service ticket" (ST). ST is the credential that the client uses to authenticate himself/herself to the application server. The AS and the TGS together are known as the "Key Distribution Center" (KDC)

A TGT might be valid for a day, and may be used to obtain several STs for many different application servers from the TGS, while a single ST is valid for a few minutes (although it may be used repeatedly) and is used for a single application server. That is, Authentication Service Exchange occurs once for every logon session, the user doesn't need to login every time he starts an application that uses Kerberos. *Delegation* refers to the facility for a service to impersonate an authenticated client in order to relieve the user of the additional burden of authenticating to multiple services. To the latter services, it will look as if they are communicating directly with the user, whereas in reality another service will sit between them and the user.

Kerberos can provide authentication, authorization and accounting security properties. Authentication is the confirmation that a user who is requesting services is a valid user of the network services requested. Authorization is the granting of specific type of service to a user based on their authentication, and what services they are requesting and what the current system state is. Accounting is the tracking of the consumption of network resources by users.

Kerberos is more secure than LAN Manager(LM) authentication and NTLM authentication, since user's passwords are never sent across the network encrypted or in plain text; session keys are only passed across the network in encrypted form; client and server systems are mutually authenticated, and Kerberos limits the duration of their users' authentication.

There are two major versions of Kerberos in common use. Version 4 is the most widely used version and it uses DES encryption algorithm, which has been shown to be vulnerable to brute-force-attacks with little computing power. Version 5 is a draft Internet Standard (RFC 1510[6]) which has corrected some of the security deficiencies of Version 4. For example, Kerberos Version 5 has indicated the notion realm of the user, added a random nonce to assure the response fresh, improved the ticket lifetime to enhance the security, etc.

Here are the related ports, protocols and functions of Kerberos:

Port	Protocol	Function
88	UDP TCP	Kerberos V5
750	UDP TCP	Kerberos V4 Authentication
751	UDP TCP	Kerberos V4 Authentication
752	UDP	Kerberos password server
753	UDP	Kerberos user registration server
754	TCP	Kerberos slave propagation
1109	TCP	POP with Kerberos
2053	TCP	Kerberos demultiplexer
2105	TCP	Kerberos encrypted rlogin

In Windows 2000, 56bit DES and 128bit RC4 are the most commonly used ones in Kerberos; in Windows Server 2003, RC4-HMAC, DES-CBC-CRC and DES-CBC-MD5 are commonly used; in Windows XP, RC4 is commonly used while others are allowed, and DES is notably used; in Windows Vista, 256bit AES, 3DES, SHA2 are commonly used.

5.3.1.1 Terms

Term 5.1 Kerberos uses "*realm*" to group user accounts. A Kerberos realm means a single Kerberos administrative domain, and it includes at lease a Kerberos server, a number of Clients and several Application servers.

Kerberos supports inter-realm authentication, but the Kerberos server in each realm should shared a secret key with those servers in other realms, and the Kerberos server in one realm should trust the one in other realm to authenticate its users. Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child.

A Kerberos server, which is a trusted third party, or, in Kerberos terminology, the Key Distribution Center (KDC). The KDC itself is made of two subservices: the Authentication Service and the Ticket Granting Service. In Windows 2000 and Windows Server 2003, both services run on the KDC server. While in other Kerberos implementations these two subservices can run on different machines such as an Authentication Server (AS) and a Ticket Granting Server (TGS). This is for the scenario where application servers belong to different TGS's in different domains.

A number of clients, all should have registered with the Kerberos server KDC to use the information resources and services.

Several application servers, which are target servers that provide information resources and services, and they all have shared long-term keys with the Kerberos server KDC.

Notation 5.3 Key Distribution Center (KDC) is composed of an Authentication Server (AS) and a Ticket Granting Server (TGS). It has a database that houses all principals, including severs and clients, and their keys for a given realm. For example, the Kerberos KDC runs on every Windows 2000 domain controller and Windows 2003 server.

Notation 5.4 Authentication Server (AS) authenticates a client logon and issues a Ticket Granting Ticket (TGT) for future authentication.

Notation 5.5 Ticket Granting Server (TGS) is responsible for accepting and verifying TGT from the AS, and grants application service tickets to clients holding this TGT. The existence of TGS allows the clients only to have to authenticate themselves once to the AS to get TGT, which can then be presented to the TGS.

Notation 5.6 Application Server (S) is responsible for accepting and verifying service tickets from the TGS, and grants information resources and services to a network client.

Notation 5.7 Client (C)Client is a Client (a user process) which makes use of a network service on behalf of a user. The user credential is given to the client C as C prompts the user to key-in his password. Note that in some cases a server may itself be a client of some other servers. Kerberos assumes that the workstations or machines are secure, i.e., there is no way for an attacker to intercept communication between a user and a client.

Notation 5.8 Ticket or Kerberos ticket is encrypted protocol messages used to confirm identities of the end participants and to establish a new session key that both parties will shared for secure communication. Kerberos uses two types of tickets in its process of authentication: TGTs and Service Tickets.

Notation 5.9 Authenticator consists of timestamps that are encrypted with the secret session key shared between the client and the AS, or between the client and the application server. Note that the timestamp cannot exceed the expiration time. The authenticator has a very short life time to prevent replay attacks, and the authenticator can only be used once. One authenticator is typically built in per session of use of a service.

Notation 5.10 Ticket Granting Ticket (TGT) is issued by the Authentication Server (SA) that contains the client's Privilege Attribute Certificate (PAC).

Notation 5.11 Privilege Attribute Certificate (PAC) is strictly used in Windows 2000 Kerberos authentication, which contains information such as the user's Security ID (SID), group membership SIDs, and users' rights on the domain.

Notation 5.12 Service Ticket is issued by the Ticket Granting Server, which provides authentication for a specific application server or resource.

Notation 5.13 Session key is a derived value used strictly for the immediate session between a client and a resource server.

5.3.1.2 Kerberos exchanges

The Kerberos protocol consists of several sub-protocols (or phases, or exchanges): the authentication service exchange, ticket granting service exchange, client/server exchange. The Kerberos authentication process is illustrated in Fig. 5.20.

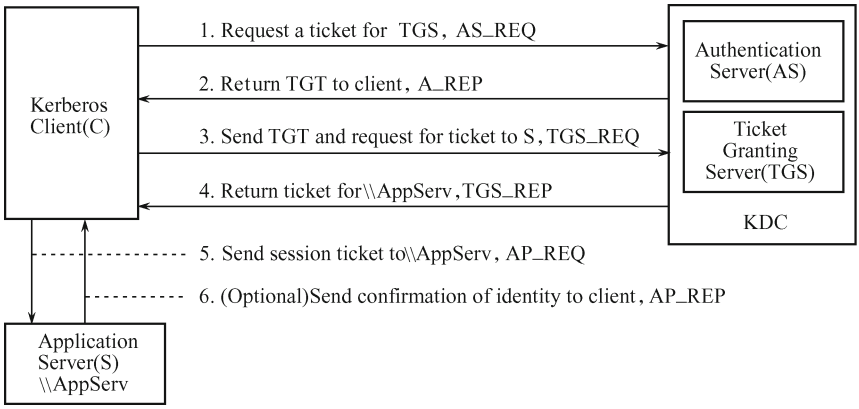


Fig. 5.20 The Kerberos protocol message exchanges.

Phase 1 Authentication service exchange:

— Message 1: The client sends a request AS_REQ to the authentication server (AS) requesting a Ticket Granting Ticket (TGT) to the Ticket Granting Server (TGS).

— Message 2: AS looks up the client and server principals named in the AS_REQ in its database, extracting their respective keys, then generates a "random" session key ($k_{c,tgs}$) for use between the client and the TGS. AS creates and sends the client a TGT which includes the client part and the TGS part. The part of TGT for the client including $k_{c,tgs}$ is encrypted under the client's secret key, and the part of TGT for TGS including $k_{c,tgs}$ is encrypted under the long-term secret key between the AS and the TGS.

Phase 2 Ticket granting service exchange:

— Message 3: The client decrypts the encrypted part using its secret key, verifies client's sending nonce (to detect replays) and recovers the session key $k_{c,tgs}$, then uses $k_{c,tgs}$ to create an authenticator containing the user's name, IP address and a timestamp. The client sends this authenticator, along with the TGT, to the TGS, requesting access to the application server S.

— Message 4: The TGS decrypts the TGT, then uses $k_{c,tgs}$ inside the TGT to verify the user's name, IP address and the timestamp in the authenticator. If everything matches, then the TGS generates a "random" new session key ($k_{c,s}$) for the client and the application server. The Kerberos

database is queried to retrieve the record for the requested server, and the TGS creates and sends the client a new ticket. This ticket is also made of the client part and the S part. The part for the client encrypts the $k_{c,s}$ using $k_{c,tgs}$, and the part for S encrypts the $k_{c,s}$ with the network address, the client's name, the server's name, the time of initial authentication and an expiration time using the long-term secret key $K_{s,tgs}$ between the S and the TGS.

Phase 3 Client/server exchange:

— Message 5: The client decrypts the encrypted part using $k_{c,tgs}$, verifies client's sending nonce and recovers the session key $k_{c,s}$, then uses $k_{c,s}$ to create a new authenticator containing the user's name, IP address and a timestamp. The client sends the received encrypted session ticket and the encrypted new authenticator to the application server S.

— Message 6: The application server decrypts and checks the ticket, authenticator, client address and timestamp. For applications that require two-way authentication, the target server returns a message consisting of the timestamp plus 1, encrypted with $k_{c,s}$.

Up to now, the application server **S** and the client **C** have authenticated the opponent party is who he claims to be, and the two now shared an encryption key $k_{c,s}$ for future secure communications. The user ID and password are secure since they are never sent over the network.

5.3.2 Basic Kerberos network authentication service

Kerberos 5, the most recent version, is introduced in the early 1990s[6, 27], and it is available for all major operating systems: Microsoft Windows 2000, Microsoft Windows server 2003, and many UNIX and UNIX-like operating systems, including FreeBSD, Apple's Mac OS X, Red Hat Enterprise Linux 4, Sun's Solaris, IBM's AIX, HP's OpenVMSLinux[28]. Especially, Microsoft bases its Kerberos implementation on the standard defined in RFC 1510. Fig. 5.21 shows the Message exchanges of Kerberos V:

Message 1 $C \rightarrow AS:$ $\text{Options}||\text{ID}_c||\text{Realm}_c||\text{ID}_{tgs}||\text{Times}||\text{Nonce}_1$

Message 2 $AS \rightarrow C:$ $\text{ID}_c||\text{Realm}_c||\text{Ticket}_{tgs}||\{\text{ID}_{tgs}||\text{Realm}_{tgs}||k_{c,tgs}||\text{Times}||\text{Nonce}_1\}_{K_c}$

Message 3 $C \rightarrow TGS:$ $\text{Options}||\text{ID}_s||\text{Realm}_s||\text{Times}||\text{Nonce}_2||\text{Ticket}_{tgs}||\text{Authenticator}_{c_1}$

Message 4 $TGS \rightarrow C:$ $\text{Realm}_c||\text{ID}_c||\text{Ticket}_s||\{k_{c,s}||\text{Times}||\text{Nonce}_2||\text{ID}_s||\text{Realm}_s\}_{k_{c,tgs}}$

Message 5 $C \rightarrow S:$ $\text{Options}||\text{Ticket}_s||\text{Authenticator}_{c_2}$

Message 6 $S \rightarrow C:$ $\{C_{time}||[\text{Subkey}]||[\text{Seq}]\}_{k_{c,s}}$

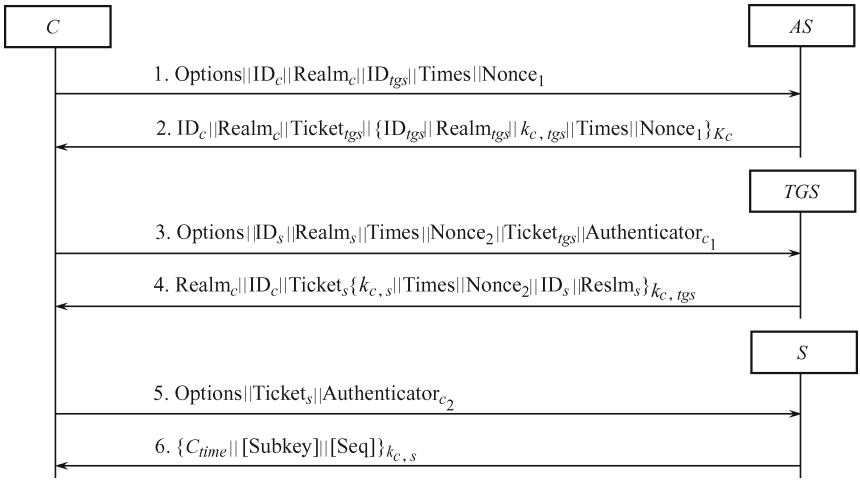


Fig. 5.21 Message exchanges of domain authentication based on Kerberos.

where

$\text{Ticket}_{tgs} = \{\text{ID}_{tgs}||\text{Realm}_{tgs}||\text{Flags}||k_{c,tgs}||\text{Realm}_c||\text{ID}_c||\text{Times}\}_{K_{as,tgs}}$,

$\text{Authenticator}_{c_1} = \{\text{ID}_c||\text{Realm}_c||\text{C}_{time}\}_{k_{c,tgs}}$,

$\text{Ticket}_s = \{\text{ID}_s||\text{Realm}_s||\text{Flags}||k_{c,s}||\text{Realm}_c||\text{ID}_c||\text{Times}\}_{K_{s,tgs}}$,

$\text{Authenticator}_{c_2} = \{\text{ID}_c||\text{Realm}_c||\text{C}_{time}||[\text{Subkey}]||[\text{Seq}]\}_{k_{c,s}}$

The fields in the above messages are:

Options: the client may specify a number of options in the initial request. Among these options are whether preauthentication is to be performed; whether the requested ticket is to be renewable, proxiable, or forwardable; whether it should be postdated or allow postdating of derivative tickets, etc.

ID_c, ID_{tgs} or ID_s: it is the identity of the client user, the ticket granting server or the application server. It is a uniquely named client or server instance that participates in a network communication.

Realm_c, Realm_{tgs} or Realm_s: it indicates the realm of the client user, the *TGS* server or the application server respectively.

Time consists of three parts:

from: the desired start time for the ticket;

till: the requested expiration time;

rtime: requested renew-till time.

Nonce_1 or Nonce_2: it is a random value generated by the client to assure the response of freshness. If the same number is included in the encrypted response from the *KDC*, it provides evidence that the response is fresh and has not been replayed by an attacker.

Ticket_{tgs}: it is a ticket granting ticket to obtain service-granting ticket, which is received from *AS*.

K_c: it is a long-term key between the client and the authentication server, which is traditionally derived from a password chosen by the user.

$k_{c,tgs}$: a temporary key for secure communication between the client and the ticket granting server.

Authenticator$_{c_1}$: it is the authenticator that contains plaintext encrypted under $k_{c,tgs}$, hence it proves that the client knows the temporary key $k_{c,tgs}$.

$k_{c,s}$: a temporary session key for secure communication between the client and the application server.

Authenticator$_{c_2}$: it is the authenticator that contains plaintext encrypted under $k_{c,s}$, hence it proves that the client knows the session key $k_{c,s}$.

[...]: it indicates an optional field.

Subkey: it contains the client's choice for an encryption key which is to be used to protect this specific application session. If this field is left out the session key from the ticket will be used.

Seq: it is a sequence number used to detect replays. The initial sequence number should be random and uniformly distributed across the full space of possible sequence numbers, so that it cannot be guessed by an attacker.

C$_{time}$: it contains the current time on the client's host.

5.3.3 Security analysis of Kerberos based on trusted freshness

Example 5.12 Figure 5.22 illustrates the security analysis of Kerberos V5 based on trusted freshness. For ease of exposition of the mutual authentication and key establishment idea in the Kerberos Authentication Protocol, only some mandatory protocol messages will be presented and some minute details are omitted to avoid obscuration.

Message 1	$C \rightarrow AS:$	C, TGS, T, N_1
Message 2	$AS \rightarrow C:$	$C, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_{K_c}$
Message 3	$C \rightarrow TGS:$	$S, T, N_2, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}},$ $\{C, Client\_time\}_{k_{c,tgs}}$
Message 4	$TGS \rightarrow C:$	$C, \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$
Message 5	$C \rightarrow S:$	$\{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{C, Client\_time\}_{k_{c,s}}$
Message 6	$S \rightarrow C:$	$\{Client\_time\}_{k_{c,s}}$

Notation

C denotes the client, AS denotes the authentication server, TGS denotes the ticket granting server and S denotes the application server.

T and *Client_time* are the timestamps chosen by the client for the ticket.

N_1 and N_2 are nonces randomly chosen by the client for AS_REQ and TGS_REQ respectively.

K_c is a long-term key between the client C and the AS, which is usually

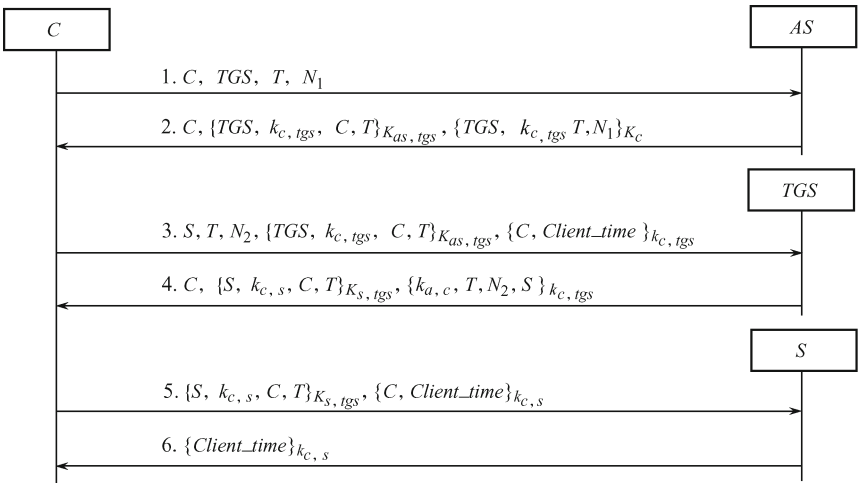


Fig. 5.22 Messages of the Kerberos protocol.

derived from the user's password.

$K_{as,tgs}$ and $K_{s,tgs}$ are the long-term keys between AS and TGS, between S and TGS respectively.

$k_{c,tgs}$ is a temporary key randomly chosen by the authentication server AS for the temporary session between C and TGS.

$k_{c,s}$ is a new session key randomly chosen by the ticket granting server TGS for this protocol run between C and the application server S.

Premise

The authentication server AS and the client C know their shared key K_c and also K_c^{-1}. $K_{as,tgs}$ is confidential and only known by the authentication server AS and the ticket granting server TGS; $K_{s,tgs}$ is confidential and only known by the authentication server S and the ticket granting server TGS. N_1 and N_2 are randomly chosen nonces. $k_{c,tgs}$ and $k_{c,s}$ are randomly chosen temporary keys for this protocol run.

Protocol actions

1) In Message 1, the client C starts the protocol run by sending the identities of the client and the ticket granting server (from whom C desires a TGT), a timestamp T and a randomly chosen new nonce N_1 by C.

2) In Message 2, the authentication server AS randomly chooses a temporary key $k_{c,tgs}$ for the subsequent communication between C and TGS, then generates a granting ticket $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ to TGS using the long-term key $K_{as,tgs}$ between AS and TGS to show that it is AS who has sent the ticket. AS also sends C the temporary key $k_{c,tgs}$ using the long-term key K_c to keep $k_{c,tgs}$ secret. K_c is usually derived from the user's password. This is the only time that this long-term key is used in a standard Kerberos

run because later exchanges use freshly generated keys.

3) Upon receiving Message 2, the client C may undertake the Ticket-Granting exchange. It decrypts $\{TGS, k_{c,tgs}, T, N_1\}_{K_c}$ using the key K_c, verifies the correction of N_1, and gets the temporary key $k_{c,tgs}$.

4) In Message 3, the client C generates the authenticator $\{C, Client\_time\}_{k_{c,tgs}}$ to prove that the client knows the key $k_{c,tgs}$. The encrypted timestamp prevents an eavesdropper from recording both the ticket and the authenticator to replay them later. C also forwards the ticket $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ received in Message 2. Encrypting the authenticator in the session key $k_{c,tgs}$ proves that it is generated by a party possessing the session key. Since no one except C and the server TGS knows the session key (it is never sent over the network in the clear), this guarantees the identity of the client C.

5) Upon receiving Message 3, the ticket granting server TGS decrypts $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ using the long-term key $K_{as,tgs}$, verifies the correction of the timestamp T, and gets the temporary key $k_{c,tgs}$. TGS also verifies the authenticator $\{C, Client\_time\}_{k_{c,tgs}}$.

6) In Message 4, TGS randomly chooses a new session key $k_{c,s}$ for the application between S and C, then generates a service ticket $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ to S using the long-term key $K_{s,tgs}$ to show that it is TGS who has sent the ticket. TGS also sends C the session key $k_{c,s}$ using the negotiated temporary key $k_{c,tgs}$ to keep $k_{c,s}$ secret.

7) Upon receiving Message 4, C decrypts $\{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$ using $k_{c,tgs}$, verifies the correction of the nonce N_2, and gets the temporary key $k_{c,s}$.

8) In Message 5, the client C generates the authenticator $\{C, Client\_time\}_{k_{c,s}}$ to prove that the client knows the key $k_{c,s}$. C also forwards the service ticket $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ received in Message 4 to the application server S.

9) Upon receiving Message 5, the application server S decrypts $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ using the long-term key $K_{s,tgs}$, verifies the correction of the timestamp T, and gets the new session key $k_{c,s}$. S also verifies the authenticator $\{C, Client\_time\}_{k_{c,s}}$.

10) In Message 6, S sends the encryption $\{Client\_time\}_{k_{c,s}}$ to show the ownership of $k_{c,s}$ by the identity S.

Successful execution should achieve mutual authentication and convince both C and S that $k_{c,s}$ is a secure new session key between C and S.

Protocol security analysis

The security properties related to mutual authentication and key establishment idea will be indicated in detail, and other security properties, such as the security of $k_{c,tgs}$ from the point of view of S, and the security of $k_{c,s}$ from the point of view of TGS, will be omitted.

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_1, and C also believes that N_1 is open. AS could not draw any useful assurance from Message 1 since there is

not any trusted freshness identifier from the point of view of AS.

2) Upon receiving Message 2, from Lemma 4.2, C has the confidential assurance of $k_{c,tgs}$. From Lemma 4.3, C has the freshness assurance of the temporary key $k_{c,tgs}$ since $k_{c,tgs}$ is sent to C together with C's trusted freshness N_1. From Lemma 4.4, C has the association assurance of N_1 and $k_{c,tgs}$ with C, since only C could get $k_{c,tgs}$ from the encryption $\{TGS, k_{c,tgs}, T, N_1\}_{K_c}$ using the long-term key K_c between C and AS. From Lemma 4.4, C also has the association assurance of N_1 and $k_{c,tgs}$ with TGS since the identity of TGS is explicitly indicated in $\{TGS, k_{c,tgs}, T, N_1\}_{K_c}$ of Message 2.

3) In Message 3, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_2, and C also believes that N_2 is open.

4) Upon receiving Message 3, from Lemma 4.2 and Lemma 4.3, TGS has the confidential and freshness assurances of the temporary key $k_{c,tgs}$ based on the timestamp T. From Lemma 4.4, TGS has the association assurance of $k_{c,tgs}$ with both TGS and C since the identities of both TGS and C are explicitly indicated in Message 3 and the encryption $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ could only be generated by the authentication server AS using the shared long-term key $K_{as,tgs}$. From Lemma 4.1, TGS has the liveness assurance of C based on the timestamp $Client\_time$, since it must be C who has just generated the authenticator $\{C, Client\_time\}_{k_{c,tgs}}$ using the shared temporary key $k_{c,tgs}$.

5) Upon receiving Message 4, from Lemma 4.2, C has the confidential assurance of $k_{c,s}$. From Lemma 4.3, C has the freshness assurance of the temporary key $k_{c,s}$ since $k_{c,s}$ is sent to C together with C's trusted freshness N_2. From Lemma 4.4, C has the association assurance of $k_{c,s}$ with C and S since the identity of S is explicitly indicated in Message 4 and the encryption $\{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$ is generated under the shared temporary key $k_{c,tgs}$ between C and TGS. From Lemma 4.1, C has the liveness assurance of TGS based on the trusted freshness N_2, since it must be TGS who has just generated the encryption $\{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$ using $k_{c,tgs}$.

6) Upon receiving Message 5, from Lemma 4.2 and Lemma 4.3, S has the confidential and freshness assurances of the temporary key $k_{c,s}$ based on the timestamp T. From Lemma 4.4, S has the association assurance of $k_{c,s}$ with both C and S since the identities of both C and S are explicitly indicated in Message 5 and the encryption $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ could only be generated by TGS using the shared long-term key $K_{s,tgs}$. From Lemma 4.1, S has the liveness assurance of C based on the timestamp $Client\_time$, since it must be C who has just generated the authenticator $\{C, Client\_time\}_{k_{c,s}}$ using the shared session key $k_{c,s}$.

7) Upon receiving Message 6, from Lemma 4.1, C has the liveness assurance of S based on the timestamp $Client\_time$, since it must be S who has just generated the authenticator $\{Client\_time\}_{k_{c,s}}$ using the shared session key $k_{c,s}$.

Table 5.8 indicates the analyzing result of Kerberos protocol. Upon termi-

nation of the protocol run, S believes that C is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both C and S, while C believes that S is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both C and S. That is, the analyzed Kerberos authentication protocol has achieved the security objects of mutual authentication and secure key establishment.

Table 5.8 Security analysis of the Kerberos protocol

	C							TGS		S	
	AS	TGS	S	N_1	$k_{c,tgs}$	N_2	$k_{c,s}$	C	$k_{c,tgs}$	C	$k_{c,s}$
Message 1				$01\#$							
Message 2	1			$01C\,TGS$	$11C\,TGS$						
Message 3						$01\#$		1	$11C\,TGS$		
Message 4		1				$01CS$	$11CS$				
Message 5										1	$11CS$
Message 6			1								
End of run			1				$11CS$			1	$11CS$

5.3.4 Public-key Kerberos

PKINIT introduces a new trust model in which the KDC is not the first entity to identify the users (as is the case for classical Kerberos). Before KDC authentication, users are identified by the certification authority CA in order to obtain a certificate. In this new model the users and the KDC obviously both need to trust the same CA.

Public-Key Kerberos PKINIT[26], which is included in Windows 2000 and Windows Server 2003, is an extension to Kerberos 5 that uses public-key cryptography for initial authentication. That is, PKINIT modifies the authentication service exchange but not other parts of the basic Kerberos 5 protocol to avoid shared secrets between a client and an authentication server. PKINIT enables the smart card logon process to a Windows 2000 or later domain. PKINIT allows a client's master key to be replaced with its public-key credentials in the Kerberos Authentication[27].

In traditional Kerberos 5 protocol, the long-term shared key in the authentication service exchange is typically derived from a password, which limits the strength of the authentication to the user's ability to choose and remember good passwords, while PKINIT uses public-key cryptography and thus avoids this problem. Furthermore, if a public-key infrastructure (PKI) is already in place, PKINIT allows network administrators to use it rather than to expend additional effort to manage users' long-term keys needed for traditional Kerberos. However, this protocol extension adds complexity to Kerberos as it retains symmetric encryption in the later exchanges but relies on asymmetric encryption, digital signatures, and corresponding certificates

in the first exchange. PKINIT is intended to add flexibility, security and administrative convenience by introducing public-key cryptography.

In PKINIT, the client C and the authentication server AS each possesses an independent public/secret key pair, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S, respectively. Certificate sets $Cert_C$ and $Cert_S$ issued by a PKI independent of Kerberos are used to testify the binding between each principal and his purported public-key. C and AS need only maintain the public-keys of a few known certification authorities CA within the PKI. Hence, AS need not maintain keys individually shared with each client, and dictionary attacks are defeated as user-chosen passwords are replaced with automatically generated asymmetric keys. Since very few users would be able to remember a random public/private key pair, PKINIT authentication is typically used with smartcard, where the keys and certificate chains are stored in a smartcard that the user swipes in a reader at login time or in the user's hard drive.

PKINIT is supported by Kerberized versions of Microsoft Windows, including Windows 2000 Professional and Server, Windows XP, and Windows Server 2003[29]; it has also been included in Heimdal since 2002[30]. PKINIT is not yet supported in the MIT reference implementation.

The manner in which PKINIT works depends on both the protocol version and the mode invoked. "PKINIT-n" is used to refer to the protocol as specified in the nth draft revision and "PKINIT" for the generic protocol[31]. PKINIT can operate in two modes: Diffie-Hellman (DH) mode and public-key encryption mode.

5.3.4.1 PKINIT public-key encryption mode

In public-key encryption mode, the key pairs are used for both signature and encryption. The latter is designed to (indirectly) protect the confidentiality of AK, while the former ensures its integrity.

Phase 1 Authentication service exchange

The abstract structure of the authentication service exchange in public-key encryption PKINIT – 26 is given:

$$\begin{aligned}
&\text{Message 1} \quad C \rightarrow AS: \; Cert_C, \{t_c, n_2\}_{K_c^{-1}}, C, TGS, T, N_1 \\
&\text{Message 2} \quad AS \rightarrow C: \; \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C, \\
&\qquad\qquad \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k
\end{aligned}$$

The last part of Message 1, "C, TGS, T, N_1", is exactly as in basic Kerberos 5, containing the client's name, the name of the TGS from which the client wants to get a TGT, a timestamp and a nonce. $Cert_C, \{t_c, n_2\}_{K_c^{-1}}$ is added by PKINIT and contains the client's certificates $Cert_C$ and client's signature $\{t_c, n_2\}_{K_c^{-1}}$ over a timestamp t_c and another nonce n_2. The nonces n_2 and timestamp t_c are generated by C specifically for this request.

Message 2 is more complex than in basic Kerberos. The last part of Message 2 "$C, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$, $\{TGS, k_{c,tgs}, T, N_1\}_k$" is very similar to AS's reply in basic Kerberos; the difference is that the symmetric key k

which is used to protect $k_{c,tgs}$ is now freshly generated by AS and not a long-term shared key that is usually derived from the client's password. The ticket-granting ticket TGT $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ is encrypted with a long-term key $K_{as,tgs}$ shared between AS and the ticket granting server TGS; the TGT contains TGS's name, $k_{c,tgs}$, C's name, and AS's local time T. The message part encrypted under the freshly generated symmetric key k includes TGS's name, $k_{c,tgs}$, AS's local time T, and the nonce N_1 from the request. To ensure the ability to learn k of C, PKINIT adds the message part $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ in Message 2. This encryption is encrypted under K_c, and it contains AS's certificate $Cert_{AS}$ and the signature $\{k, n_2\}_{K_{as}^{-1}}$ by AS, hence only C can get the freshly generated key k.

PKINIT leaves the subsequent exchanges of Kerberos unchanged.

Example 5.13 Here is the security analysis of the Kerberos public-key encryption PKINIT based on trusted freshness. The whole message exchanges of PKINIT mode are illustrated in Fig. 5.23.

Message 1 $C \rightarrow AS: \; Cert_C, \{t_c, n_2\}_{K_c^{-1}}, C, TGS, T, N_1$

Message 2 $AS \rightarrow C: \; \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C,$
$\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k$

Message 3 $C \rightarrow TGS: S, T, N_2, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}},$
$\{C, Client\_time\}_{k_{c,tgs}}$

Message 4 $TGS \rightarrow C: C, \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$

Message 5 $C \rightarrow S: \; \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{C, Client\_time\}_{k_{c,s}}$

Message 6 $S \rightarrow C: \; \{Client\_time\}_{k_{c,s}}$

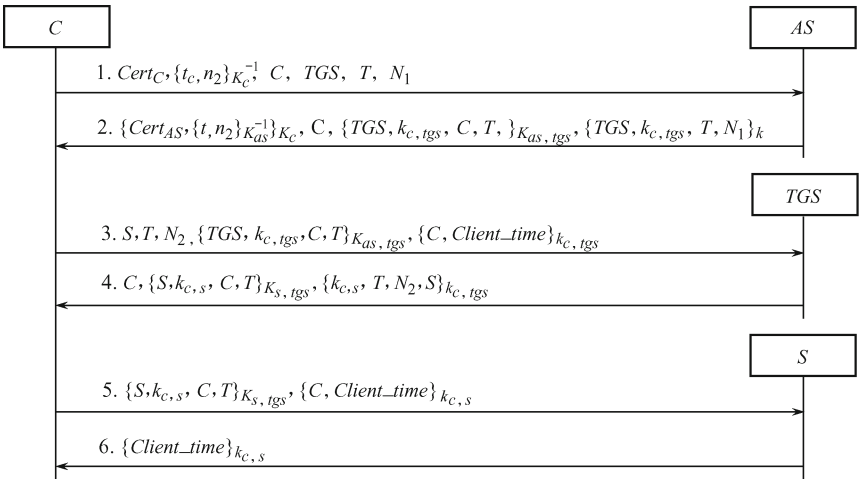


Fig. 5.23 Kerberos message exchanges in Public-key encryption PKINIT mode.

Notation

K_c and K_c^{-1} are the public-key and the private key for C, and K_{as} and K_{as}^{-1} are for AS.

$Cert_C$ and $Cert_{AS}$ denote the client's certificate and the authentication server's certificate respectively.

t_c is a timestamp from the client C and n_2 is a nonce randomly chosen by C.

k is a symmetric key randomly chosen by AS to protect the temporary key $k_{c,tgs}$.

Other notations are the same as in the basic Kerberos protocol, hence omitted.

Premise

Each principal knows the public-key of the trusted certificate authority CA to get K_c or K_{as} from $Cert_C$ or $Cert_{AS}$. Each principal knows the key pair of himself, that is, K_c and K_c^{-1} for C, K_{as} and K_{as}^{-1} for AS.

Protocol actions

1) In Message 1, the client C randomly chooses the nonces n_2 and N_1, and signs n_2 and timestamp t_c using C's private key K_c^{-1}, then C starts the protocol run by sending the certificate of C, the signature $\{t_c, n_2\}_{K_c^{-1}}$, the identities of C and TGS, a timestamp T and N_1.

2) Upon receiving Message 1, AS gets the public-key of C from $Cert_C$, decrypts $\{t_c, n_2\}_{K_c^{-1}}$ using C's public-key K_c, verifies the correction of timestamp t_c, and gets the nonce n_2. The KDC will then query the Active Directory for a mapping between the certificate $Cert_C$ and a Windows account. If it finds a mapping, it will issue a TGT to the corresponding account.

3) In Message 2, the authentication server AS randomly chooses a symmetric key k and a temporary key $k_{c,tgs}$ for the subsequent communication between TGS and C, then AS signs k and n_2 using AS's private key K_{as}^{-1}. AS makes response to C with $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ to keep the freshly generated key k confidential by encrypting under C's public-key K_c. The last part of Message 2 is very similar to AS's reply in basic Kerberos, hence omitted.

Other messages are the same as the basic Kerberos protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_1 and n_2, and C also believes that N_1 and n_2 are open. TGS could not draw any useful assurance from Message 1.

2) Upon receiving Message 2, from Lemma 4.2, C has the confidential assurance of k and $k_{c,tgs}$ since only C could decrypt $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ using C's private key K_c^{-1} to get k and then $k_{c,tgs}$. From Lemma 4.3, C has the freshness assurance of the temporary key k since k is sent to C together

with C's trusted freshness n_2. Similarly, C has the freshness assurance of the temporary key $k_{c,tgs}$ since $k_{c,tgs}$ is sent to C together with C's trusted freshness N_1. From Lemma 4.4, C has the association assurance of k and n_2 with AS, since AS has signed k and n_2 using AS's private key K_{as}^{-1}. However, from the point of view of C, k and n_2 are not associated with TGS or C since if an adversary is a legal user, then the adversary could impersonate C and generate the encryption $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ using the public K_c. Note that $k_{c,tgs}$ is not associated with TGS since k is not associated with C, although the identity of TGS is explicitly indicated in Message 2.

3) In Message 3, C has the same security beliefs as those in the basic Kerberos.

4) Upon receiving Message 3, TGS has the same security beliefs as those in the basic Kerberos.

5) Upon receiving Message 4, from Lemma 4.2, C has the confidential assurance of the temporary key $k_{c,s}$ since it is encrypted under the temporary key $k_{c,tgs}$. From Lemma 4.3, C has the freshness assurance of $k_{c,s}$ since $k_{c,s}$ is sent to C together with C's trusted freshness N_2.

6) Upon receiving Message 5, S has the same security beliefs as those in the basic Kerberos.

7) Upon receiving Message 6, C could not authenticate the liveness of S since C is not sure whether $k_{c,s}$ is between C and S, hence C is not sure whether the fresh message $\{Client\_time\}_{k_{c,s}}$ is from S or not.

Table 5.9 indicates the analyzing result of Public-key Kerberos protocol. Upon termination of the protocol run, S believes that C is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both S and C, while C only believes that $k_{c,s}$ is confidential and fresh, but C is not sure whether $k_{c,s}$ is between C and S or not.

Table 5.9 Security analysis of the PKINIT Public-key Kerberos protocol

	C									TGS		S	
	AS	TGS	S	k	n_2	N_1	$k_{c,tgs}$	N_2	$k_{c,s}$	C	$k_{c,tgs}$	C	$k_{c,s}$
Message 1					01#	01#							
Message 2	1			11AS	01AS	01#	11#						
Message 3								01#		1	11C TGS		
Message 4									11#				
Message 5												1	11CS
Message 6													
End of run									11#			1	11CS

Example 5.14 From the absence of the association of $k_{c,s}$ with C and S in the point of view of C, there exists an attack[31] as illustrated in Fig. 5.24.

Message 1 $\quad C \rightarrow I(AS): \; Cert_C, \{t_c, n_2\}_{K_c^{-1}}, C, TGS, T, N_1$

Message 1′ $\quad I \rightarrow AS: \; Cert_I, \{t_c, n_2\}_{K_i^{-1}}, I, TGS, T, N_1$

Message 2′	$AS \rightarrow I:$	$\left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_i}, I,$ $\{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k$
Message 2	$I(AS) \rightarrow C:$	$\left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C,$ $\{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k$
Message 3	$C \rightarrow I(TGS):$	$S, T, N_2, \{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}},$ $\{C, Client\_time\}_{k_{c,tgs}}$
Message 3′	$I \rightarrow TGS:$	$S, T, N_2, \{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}},$ $\{I, Client\_time\}_{k_{c,tgs}}$
Message 4′	$TGS \rightarrow I:$	$I, \{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$
Message 4	$I(TGS) \rightarrow C:$	$C, \{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$
Message 5	$C \rightarrow I(S):$	$\{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{C, Client\_time\}_{k_{c,s}}$
Message 5′	$I \rightarrow S:$	$\{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{I, Client\_time\}_{k_{c,s}}$
Message 6	$S \rightarrow I:$	$\{Client\_time\}_{k_{c,s}}$
Message 6′	$I(S) \rightarrow C:$	$\{Client\_time\}_{k_{c,s}}$

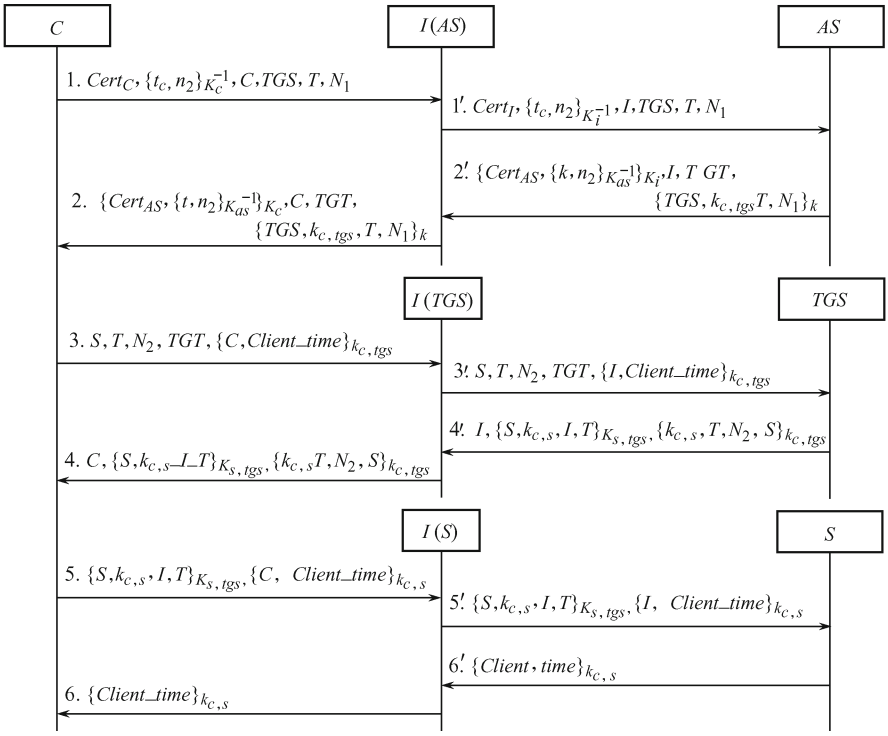


Fig. 5.24 An attack on the Kerberos PKINIT Public-key Encryption mode.

Notation

I denotes the adversary, K_i and K_i^{-1} are the public-key and the private key for I, and $Cert_I$ denotes the adversary's certificate.

$I(TGS)$ or $I(S)$ is the adversary I impersonating TGS or S.

n_2 is a nonce randomly chosen by the client for AS_REQ in PKINIT public-key encryption mode.

$TGT = \{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}}$ is a ticket granting ticket.

Other notations are the same as the original Keberos PKINIT protocol.

Premise

The adversary is a legal user, Other premises are the same as the original Keberos protocol.

Protocol actions

1) In Message 1, the client C starts a new protocol run. I intercepts Message 1, replaces $Cert_C$ with $Cert_I$, gets $\{t_c, n_2\}$ using C's public-key and encrypts it using I's private key, then sends Message $1'$ to AS.

2) Upon receiving Message $1'$, AS responds to I just as AS does in basic Keberos.

3) Upon receiving Message $2'$, I gets $\{k, n_2\}_{K_{as}^{-1}}$ and k using I's private key and AS's public-key, hence I can get $k_{c,tgs}$ using k.

4) In Message 2, I constructs $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ from $\{k, n_2\}_{K_{as}^{-1}}$ using C's public-key, replaces the identity of I with C.

5) In Message 3, C responds as usual. I intercepts Message 3, replaces the identity of C with I in $\{C, Client\_time\}$, and then sends $\{I, Client\_time\}_{k_{c,tgs}}$ to TGS.

6) Upon receiving Message $3'$, TGS responds to I just as TGS does in basic Keberos.

7) Upon receiving Message $4'$, I gets $k_{c,s}$ using $k_{c,tgs}$.

8) In Message 4, I replaces the identity of I with C in Message $4'$, and then forwards Message 4.

9) Up to now, both C and I know the session key $k_{c,s}$, hence they can complete the subsequence protocol run as usual.

Upon termination of the attack on the Kerberos PKINIT public-key encryption, the adversary I causes C to have false beliefs: C has completed a successful protocol run with S, and is sharing a new session key $k_{c,s}$ with S, whereas in fact, S knows nothing about the key establishment with C, and thinks that it has been talking with the adversary I, and sharing $k_{c,s}$ with I. From now on, C will send subsequent sensitive data encrypted under $k_{c,s}$ which is also known by I.

5.3.4.2 PKINIT Diffie-Hellman mode

In Diffie-Hellman (DH) mode, the key pairs (K_c and K_c^{-1} for C, K_{as} and K_{as}^{-1} for AS) are used to provide digital signature support for an authenticated

Diffie-Hellman key agreement which is used to protect the temporary key $k_{c,tgs}$ between the client and the TGS. A variant of this mode allows the reuse of previously generated shared secrets $k_{c,tgs}$.

The following abstract description leaves out a number of fields which are of no significance with respect to our analysis. We invite the interested reader to consult the specifications[26]. The simplified authentication service exchange in Diffie-Hellman PKINIT-26 is:

$$\begin{array}{lll}
\text{Message 1} & C \rightarrow AS: & \left\{Cert_C, \{t_c, n_2\}_{K_c^{-1}}\right\}_{K_{as}}, C, TGS, T, N_1 \\
\text{Message 2} & AS \rightarrow C: & \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C, \\
 & & \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k
\end{array}$$

Example 5.15 The whole message exchanges of PKINIT Diffie-Hellman mode are illustrated in Fig. 5.25.

$$\begin{array}{lll}
\text{Message 1} & C \rightarrow AS: & \left\{Cert_C, \{t_c, n_2\}_{K_c^{-1}}\right\}_{K_{as}}, C, TGS, T, N_1 \\
\text{Message 2} & AS \rightarrow C: & \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C, \\
 & & \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k \\
\text{Message 3} & C \rightarrow TGS: & S, T, N_2, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \\
 & & \{C, Client_time\}_{k_{c,tgs}} \\
\text{Message 4} & TGS \rightarrow C: & C, \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}} \\
\text{Message 5} & C \rightarrow S: & \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{C, Client_time\}_{k_{c,s}} \\
\text{Message 6} & S \rightarrow C: & \{Client_time\}_{k_{c,s}}
\end{array}$$

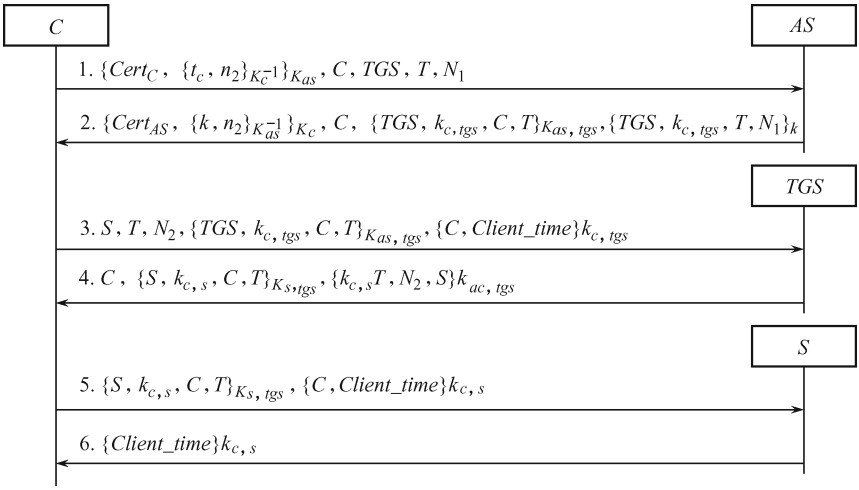


Fig. 5.25 Kerberos message exchanges in PKINIT Diffie-Hellman mode.

Table 5.10 indicates the security analysis result of PKINIT Diffie-Hellman

mode. The security analysis details are left to the interested reader.

Table 5.10 Security analysis of the Kerberos PKINIT Diffie-Hellman protocol

	C									TGS		S	
	AS	TGS	S	k	n_2	N_1	$k_{c,tgs}$	N_2	$k_{c,s}$	C	$k_{c,tgs}$	C	$k_{c,s}$
Message 1					01#	01#							
Message 2	1			11 C AS	01 C AS	01 C TGS	11 C TGS						
Message 3								01#		1	11 C TGS		
Message 4		1							11 C S				
Message 5												1	11 C S
Message 6			1										
End of run			1						11 C S			1	11 C S

Upon termination of the protocol run, S believes that C is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both S and C; at the same time, C believes that S is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both S and C.

References

[1] Tanenbaum AS (2001) Computer Networks, 3rd edn. Prentice Hall, New Jersey.

[2] Freier AO, Karlton P, Kocher PC (1996) The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt. Accessed 29 Apr 2007

[3] Dierks T, Allen C (1999) the TLS Protocol Version 1.0, RFC 2246. http://tools.ietf.org/html/rfc2246. Accessed 21 May 2011

[4] Kaufman C (2005) Internet Key Exchange (IKEv2) Protocol, RFC 4306. http://tools.ietf.org/html/rfc4306. Accessed Dec 2005

[5] Meadows C (1999) Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In: Proceedings of 1999 IEEE Symposium on Security and Privacy, Oakland, 9 – 12 May 1999

[6] Neuman C (1993) The Kerberos Network Authentication Service (V5), RFC 1510. http://www.ietf.org/rfc/rfc1510.txt. Accessed 5 May 2011

[7] Neuman BC, Ts'o T (1994) Kerberos: an Authentication Service for Computer Networks. IEEE Communications Magazine 32(9): 33 – 38

[8] Ylonen T (1995) The SSH (secure shell) Remote Login Protocol, Internet-Draft. http://www.free.lp.se/fish/rfc.txt. Accessed 15 Nov 1995

[9] Ylonen T (2002) SSH Authentication Protocol, RFC4252. http://www.ietf.org/rfc/rfc4252.txt. Accessed 5 May 2011

[10] Ylonen T (2002) SSH Connection Protocol, RFC4254. http://www.ietf.org/rfc/rfc4254.txt. Accessed 5 May 2011

[11] Ylonen T (2002) SSH Protocol Architecture, RFC4251. http://www.ietf.org/rfc/rfc4251.txt. Accessed 5 May 2011

[12] Ylonen T (2002) SSH Transport Layer Protocol, RFC4253. http://www.ietf.org/rfc/rfc4253.txt. Accessed 5 May 2011

[13] Stallings W (2006) Cryptography and Network Security: Principles and Practice, 4th edn. Prentice Hall, New Jersey

[14] Ray M, Dispensa S (2009) Renegotiating TLS. http://www.phonefactor.com/sslgapdocs/Renegotia- ting_TLS.pdf. Accessed 5 May 2011

[15] Thayer R, Doraswamy N, Glenn R (1998) IP Security Document Roadmap, RFC2411. http://tools.ietf.org/html/rfc2411. Accessed Nov 1998

[16] Hoffman P (2005) Cryptographic Suites for IPsec, RFC4308. http://tools.ietf.org/html/rfc4308. Accessed 5 May 2011

[17] Kent S, Atkinson R (1998) IP Authentication Header, RFC2402. http://tools.ietf.org/html/rfc2402. Accessed 5 May 2011

[18] Kent S, Atkinson R (1998) IP Encapsulating Security Payload (ESP), RFC2406. http://tools.ietf.org/html/rfc2406. Accessed Nov 1998

[19] Harkins D, Carrel D (1998) The Internet Key Exchange Protocol (IKE), RFC 2409. http://www.ietf.org/rfc/rfc2409.txt. Accessed Dec 2005

[20] Kent S (2005) IP Authentication Header, RFC4302. http://tools.ietf.org/html/rfc4302. Accessed Dec 2005

[21] Maughan D, Schertler M, Schneider M, Turner J (1998) Internet Security Association and Key Management Protocol (ISAKMP). IETF RFC 2408. http://www.ietf.org/rfc/rfc2408. Accessed November 1998

[22] Orman H (1998) The OAKLEY Key Determination Protocol. IETF RFC 2412. http://www.ietf.org/rfc/rfc2412. Accessed November 1998

[23] Krawczyk H (1996) SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In: Proceedings of Symposium on Network and Distributed System Security (SNDSS '96), San Diego, 22 – 23 Feb 1996

[24] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[25] Lowe G (1996) Some new Attacks Upon Security Protocols. In: Proceedings of the 9th IEEE Computer Security Foundations Workshop, Kenmare, 10 – 12 Mar 1996

[26] Zhu L, Tung B (2006) Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), RFC4556. http://www.ietf.org/rfc/rfc4556.txt. Accessed 10 June 2010

[27] Neuman C, Yu T, Hartman S, Raeburn K (2005) The Kerberos Network Authentication Service (V5). http://www.ietf.org/rfc/rfc4120

[28] Wikipedia. Kerberos (protocol). http://en.wikipedia.org/wiki/Kerberos_(protocol). Accessed 5 Dec 2011

[29] Microsoft Security Bulletin MS05-042. http://www.microsoft.com/technet/security/bulletin/ms05-042.mspx. Accessed 9 Aug 2010

[30] Strasser M, Steffen A (2002) Kerberos PKINIT Implementation for Unix Clients. Technical Report, Zurich University of Applied Sciences Winterthur, Nov 2010

[31] Cervesato I, Jaggard AD, Scedrov A, Tsay JK, Walstad C (2008) Breaking and Fixing Public-key Kerberos. Journal Information and Computation 206(2 – 4): 402 – 424