Wireless Transport Layer Security

WTLS Sessions and Connections WTLS, Protocol Architecture, Cryptographic Algorithms.

WIRELESS TRANSPORT LAYER SECURITY

WTLS provides security services between the mobile device (client) and the WAP gateway. WTLS is based on the industry-standard Transport Layer Security (TLS) Protocol,[3] which is a refinement of the Secure Sockets Layer (SSL) protocol. TLS is the standard security protocol used between Web browsers and Web servers. WTLS is more efficient that TLS, requiring fewer message exchanges. To provide end-to- end security, WTLS is used between the client and the gateway, and TLS is used between the gateway and the target server (Figure 17.14). WAP systems translate between WTLS and TLS within the WAP gateway. Thus, the gateway is a point of vulnerability and must be given a high level of security from external attacks.

WTLS provides the following features.

- **Data integrity:** Uses message authentication to ensure that data sent between the client and the gateway are not modified.
- **Privacy:** Uses encryption to ensure that the data cannot be read by a third party.
- **Authentication:** Uses digital certificates to authenticate the two parties.
- **Denial-of-service protection:** Detects and rejects messages that are replayed or not successfully verified.

WTLS Sessions and Connections

Two important WTLS concepts are the secure session and the secure connection, which are defined in the specification as:

- **Secure connection:** A connection is a transport (in the OSI layering model def- inition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Secure session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryp- tographic security parameters, which can be shared among multiple connec- tions. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

There are a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

A session state is defined by the following parameters:

- **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Protocol version:** WTLS protocol version number.
- **Peer certificate:** Certificate of the peer. This element of the state may be null.
- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null, RC5, DES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- **Master secret:** A 20-byte secret shared between the client and server.
- **Sequence number:** Which sequence numbering scheme (off, implicit, or explicit) is used in this secure connection.
- **Key refresh:** Defines how often some connection state values (encryption key, MAC secret, and IV) calculations are performed.
- **Is resumable:** A flag indicating whether the session can be used to initiate new connections.

The connection state is the operating environment of the record protocol. It includes all parameters that are needed for the cryptographic operations (encryp- tion/decryption and MAC calculation/verification). Each secure connection has a connection state, which is defined by the following parameters.

- **Connection end:** Whether this entity is considered a client or a server in this secure session.
- **Bulk cipher algorithm:** Includes the key size of this algorithm, how much of that key is secret, whether it is a block or stream cipher, and the block size of the cipher (if appropriate).
- **MAC algorithm:** Includes the size of the key used for MAC calculation and the size of the hash which is returned by the MAC algorithm.
- **Compression algorithm:** Includes all information the algorithm requires to do compression.
- **Master secret:** A 20-byte secret shared between the client and server.
- **Client random:** A 16-byte value provided by the client.
- **Server random:** A 16-byte value provided by the server.
- **Sequence number mode:** Which scheme is used to communicate sequence numbers in this secure connection.
- **Key refresh:** Defines how often some connection state parameters (encryption key, MAC secret, and IV) are updated. New keys are calculated at every $n = 2^{key\_refresh}$ messages, that is, when the sequence number is $0, 2^n, 3^n$, etc.

WTLS Protocol Architecture

WTLS is not a single protocol but rather two layers of protocols, as illustrated in Figure 17.15. The WTLS Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of WTLS. Three higher-layer protocols are defined as part of WTLS: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. These WTLS-specific protocols are used in the management of WTLS exchanges and are examined subsequently in this section.

WTLS Record Protocol The WTLS Record Protocol takes user data from the next higher layer (WTP, WTLS Handshake Protocol, WTLS Alert Protocol, and WTLS Change Cipher Spec Protocol) and encapsulates these data in a PDU. The following steps occur (Figure 17.16).

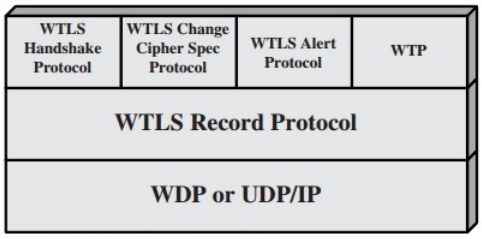

Figure 17.15 WTLS Protocol Stack

Step 1. The payload is compressed using a lossless compression algorithm.

Step 2. A message authentication code (MAC) is computed over the compressed data, using HMAC. One of several hash algorithms can be used with HMAC, including MD-5 and SHA-1. The length of the hash code is 0, 5, or 10 bytes. The MAC is added after the compressed data.

The Record Protocol header consists of the following fields (Figure 17.17).

- **Record type** (8 bits): Consisting of the subfields:

–**Record length field indicator** (1 bit): Indicates whether a record length field is present.

–**Sequence number field indicator** (1 bit): Indicates whether a sequence number field is present.

–**Cipher spec indicator** (1 bit): If this bit is zero, it indicates that no compression, MAC protection, or encryption is used.

–**Content type** (4 bits): The higher-layer protocol above the WTLS Record Protocol.

- **Sequence number** (16 bits): A sequence number associated with this record. This provides reliability over an unreliable transport service.
- **Record length** (16 bits): The length in bytes of the plaintext data (or compressed data if compression is used).

Change Cipher Spec Protocol Associated with the current transaction is a cipher spec, which specifies the encryption algorithm, the hash algorithm used as

part of HMAC, and cryptographic attributes, such as MAC code size. There are two states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created.

The Change Cipher Spec Protocol is one of the three WTLS-specific protocols that use the WTLS Record Protocol, and it is the simplest. This protocol consists of a single message, which consists of a single byte with the value 1. The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection. Thus, when the Change Cipher Spec message arrives, the sender of the message sets the current write state to the pending state and the receiver sets the current read state to the pending state.

Alert Protocol The Alert Protocol is used to convey WTLS-related alerts to the peer entity. As with other applications that use WTLS, alert messages are compressed and encrypted, as specified by the current state.

Each message in this protocol consists of 2 bytes. The first byte takes the value warning(1), critical(2), or fatal(3) to convey the severity of the message. The second byte contains a code that indicates the specific alert. If the level is fatal, WTLS immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established. A critical alert message results in termination of the current secure connection. Other connections using the secure session may continue and the secure identifier may also be used for establishing new secure connections.

The connection is closed using the alert messages. Either party may initiate the exchange of the closing messages. If a closing message is received, then any data after this message is ignored. It is also required that the notified party verifies termination of the session by responding to the closing message.

Error handling in the WTLS is based on the alert messages. When an error is detected, the detecting party sends an alert message containing the occurred error. Further procedures depend on the level of the error that occurred.

Examples of fatal alerts:

- **session_close_notify:**notifies the recipient that the sender will not send any more messages using this connection state or the secure session.
- **unexpected_message:**An inappropriate message was received.
- **bad_record_mac:**An incorrect MAC was received.
- **decompression_failure:** The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure:**Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal_parameter:**A field in a handshake message was out of range or inconsistent with other fields.

Examples of nonfatal alerts:

- **connection_close_notify:**Notifies the recipient that the sender will not send any more messages using this connection state.
- **bad_certificate:**A received certificate was corrupt (e.g., contained a sig- nature that did not verify).
- **unsupported_certificate:**The type of the received certificate is not supported.
- **certificate_revoked:**A certificate has been revoked by its signer.
- **certificate_expired:**A certificate has expired.
- **certificate_unknown:**Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

Handshake Protocol The most complex part of WTLS is the Handshake Protocol. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithms and cryptographic keys to be used to protect data sent in a WTLS record. The Handshake Protocol is used before any application data are transmitted. An important function of the Handshake Protocol is the generation of a pre-master secret, which in turn is used to generate a master secret. The master secret is then used to generate various cryptographic keys.

The Handshake Protocol consists of a series of messages exchanged by client and server. Figure 17.18 shows the initial exchange

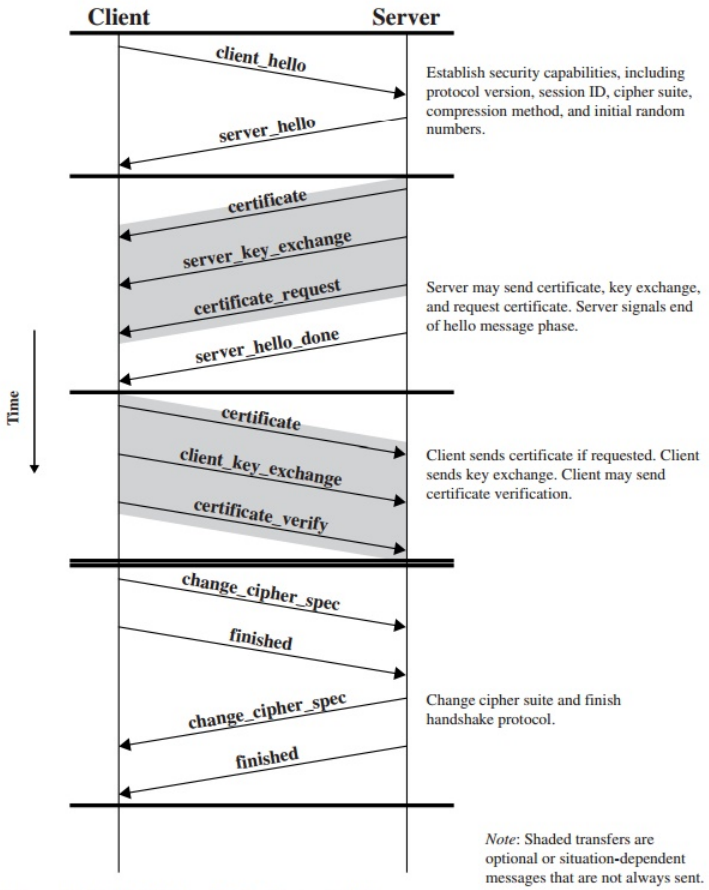

Figure 17.18 WTLS Handshake Protocol Action

connection between client and server. The exchange can be viewed as having four phases.

The **first phase** is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client. The client sends a client_hellomessage that includes a session ID and a list of cryptographic and compression algorithms supported by the client (in decreasing order of preference for each algorithm type). After sending the client_hellomessage, the client waits for the server_hellomessage. This message indicates which cryptographic and compression algorithms will be used for the exchange.

The **second phase** is used for server authentication and key exchange. The server begins this phase by sending its public-key certificate if it needs to be authenticated. Next, a server_key_exchange message may be sent if it is required. This message is needed for certain public-key algorithms used for symmetric key exchange. Next, the server can request a public-key certificate from the client, using the certificate_requestmessage. The final message in phase 2 (and one that is always required) is the server_hello_donemessage, which is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response. This message has no parameters.

The **third phase** is used for client authentication and key exchange. Upon receipt of the server_hello_done message, the client should verify that the server provided a valid certificate if required and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more mes- sages back to the server. If the server has requested a certificate, the client sends a certificate message. Next is the client_key_exchangemessage, which must be sent in this phase. The content of the message depends on the type of key exchange. Finally, in this phase, the client may send a certificate_verifymessage to pro- vide explicit verification of a client certificate.

The **fourth phase** completes the setting up of a secure connection. The client sends a change_cipher_specmessage and copies the pending CipherSpec into the current CipherSpec. Note that this message is not considered part of the Handshake Protocol but is sent using the Change Cipher Spec Protocol. The client then immediately sends the finished message under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful. In response to these two messages, the server sends its own change_cipher_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point, the hand- shake is complete, and the client and server may begin to exchange application layer data.

Cryptographic Algorithms

Authentication Authentication in the WTLS is carried out with certificates. Authentication can occur either between the client and the server or when the client only authenticates the server. The latter procedure can happen only if the server allows it to occur. The server can require the client to authenticate itself to the server.

- **Certificate_version:** Version of the certificate.
- **Signature_algorithm:** Algorithm used to sign the certificate.
- **Issuer:** Defines the party who has signed the certificate, usually some CA.
- **Valid_not_before:** The beginning of validity period of the certificate.
- **Valid_not_after:** The point of time after the certificate is no longer valid.
- **Subject:** Owner of the key, associated with the public key being certified.
- **Public_key_type:** Type (algorithm) of the public key.
- **Parameter_specifier:** Specifies parameter relevant for the public key.
- **Public key:** The public key being certified.
- **Signature:** Signed with the CA's private key.

KEY EXCHANGE The purpose of the WTLS protocol is for the client and server to generate a mutually shared pre-master key. This key is then used to generate as master key, as explained subsequently. A number of key exchange protocols are supported by WTLS. They can be grouped into those protocols that include a server_key_exchange message as part of the Handshake Protocol (Figure 17.18) and those that don't.

The server_key_exchangemessage is sent by the server only when the server certificate message (if sent) does not contain enough data to allow the client to exchange a pre-master secret. The following three methods require the use of the server_key_exchange message.

- **DH_anon:** The conventional Diffie-Hellman computation is performed anonymously (without authentication). The negotiated key (Z) is used as the pre_master_secret.
- **ECDH_anon:** The elliptic curve Diffie-Hellman computation is performed. The negotiated key (Z) is used as the pre_master_secret.
- **RSA_anon:** This is an RSA key exchange without authentication. The server sends its RSA public key. In this method, a 20-byte secret value is generated by the client, encrypted under the server's public key, and sent to the server. The server uses its private key to decrypt the secret value. The pre_master_secret is the secret value appended with the server's public key.

The server key exchange message is not sent for the following key exchange methods.

ECDH_ECDSA: Elliptic curve Diffie-Hellman key exchange with ECDSA- based certificates. The server sends a certificate that contains its ECDH public key. The server certificate is signed with ECDSA by a third party trusted by the client. Depending whether the client is to be authenticated or not, it sends its certificate containing its ECDH public key signed with ECDSA by a third party trusted by the server or just its (temporary) ECDH public key. Each party calculates the pre-master secret based on one's own private key and counterpart's public key received as such or contained in a certificate.

- **RSA:** RSA key exchange with RSA-based certificates. The server sends a certificate that contains its RSA public key. The server certificate is signed with RSA by a third party trusted by the client. The client extracts the server's public key from the received certificate, generates a secret value, encrypts it with the server's public key, and sends it to the server. The pre- master secret is the secret value appended with the server's public key. If the client is to be authenticated, it signs some data (messages sent during the handshake) with its RSA private key and sends its certificate and the signed data.

PSEUDORANDOM FUNCTION(PRF) The PRF is used for a number of purposes in WTLS. The PRF takes as input a secret value, a seed, and an identifying label and produces an output of arbitrary length. In the TLS standard, two hash algorithms were used in order to make the PRF as secure as possible. In order to save resources, WTLS can be implemented using only one hash algorithm. Which hash algorithm is actually used is agreed on during the handshake as a part of the cipher spec.

The PRF is based on the data expansion function

MASTER KEY GENERATION The shared master secret is a one-time 20-byte value (160 bits) generated for this session by means of secure key exchange.The creation is in two stages. First, a pre_master_secretis exchanged. Second, the master_secret is calculated by both parties, using the function

master_secret = PRF(pre_master_secret, "master secret", ClientHello.random |ServerHello.random)

where ClientHello.randomand ServerHello.randomare the random numbers exchanged during the first phase of the handshake protocol.

The MAC and encryption keys are then derived from the master key. The MAC calculation uses the HMAC algorithm (Chapter 12) and encompasses the fields indicated in the expression

HMAC_hash (MAC_secret, seq_number|WTLSCompressed. record_type|WTLSCompressed.length |WTLS Compressed.fragment)

where WTLSCompressed.fragment refers to the (optionally) compressed plain- text data field.

Study Material, Lecturing Notes, Assignment, Reference, Wiki description explanation, brief detail
: Wireless Transport Layer Security |

Prev Page (https://www.brainkart.com/article/Wireless-Application-Protocol-Overview_8487/) Next Page (https://www.brainkart.com/article/WAP-End-to-End-Security_8489/)

Related Topics

Cryptography and Network Security (https://www.brainkart.com/subject/Cryptography-and-Network-Security_137/)

Anna University - All Subjects (https://www.brainkart.com/menu/anna-university/)

Anna University ECE - All Subjects (https://www.brainkart.com/menu/anna-university-ece/)

Anna University CSE - All Subjects (https://www.brainkart.com/menu/anna-university-cse/)

Anna University IT - All Subjects (https://www.brainkart.com/menu/anna-university-it/)

Engineering - All Subjects (https://www.brainkart.com/menu/engineering/)

Electron - All Subjects (https://www.brainkart.com/menu/electron/)