Algorithms of Information Security: Key establishement and management

Faculty of Information Technology
Czech Technical University in Prague

October 20, 2020

General classification and fundamental concepts

- Recall that a *protocol* is a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective.

Definition

Key establishment is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.

General classification and fundamental concepts

Definition

A *key transport* protocol or mechanism is a key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s).

Definition

A *key agreement* protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value.

General classification and fundamental concepts

- Key establishment protocols involving authentication typically require a set-up phase whereby authentic and possibly secret initial keying material is distributed.

Definition

Key pre-distribution schemes are key establishment protocols whereby the resulting established keys are completely determined a priori by initial keying material. In contrast, *dynamic key establishment* schemes are those whereby the key established by a fixed pair (or group) of users varies on subsequent executions.

General classification and fundamental concepts

Definition

Key authentication is the property whereby one party is assured that no other party aside from a specifically identified second party (and possibly additional identified trusted parties) may gain access to a particular secret key.

- Key authentication is independent of the actual possession of such key by the second party, or knowledge of such actual possession by the first party; in fact, it need not involve any action whatsoever by the second party.
- For this reason, it is sometimes referred to more precisely as *(implicit) key authentication.*

General classification and fundamental concepts

Definition

Key confirmation is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

Definition

Explicit key authentication is the property obtained when both (implicit) key authentication and key confirmation hold.

Definition

An *authenticated key establishment* protocol is a key establishment protocol which provides key authentication.

General classification and fundamental concepts

Cryptographic protocols involving message exchanges require precise definition of both the messages to be exchanged and the actions to be taken by each party. The following typesof protocols may be distinguished, based on objectives as indicated:

- *authentication protocol* – to provide to one party some degree of assurance regarding the identity of another with which it is purportedly communicating;
- *key establishment protocol* – to establish a shared secret;
- *authenticated key establishment protocol* – to establish a shared secret with a party whose identity has been (or can be) corroborated.

Point-to-point key update using symmetric encryption

- Server-less key transport based on symmetric techniques may either require that the two parties in the protocol initially share a long-term pairwise secret or not, respectively illustrated below by point-to-point key update techniques and Shamir's no-key algorithm.

Point-to-point key update using symmetric encryption:

- *Point-to-point key update* techniques based on symmetric encryption make use of a long-term symmetric key K shared *a priori* by two parties A and B.
- This key, initially distributed over a secure channel or resulting from a key pre-distribution scheme, is used repeatedly to establish new session keys W. Representative examples of point-to-point key transport techniques follow.

Point-to-point key update using symmetric encryption

- *Notation:* r_A, t_A, and n_A, respectively, denote a random number, timestamp, and sequence number generated by A. E denotes a symmetric encryption algorithm. Optional message fields are denoted by an asterisk $(*)$.

  1. *key transport with one pass:*

     $$A \rightarrow B : E_K(r_A)$$

     The session key used is $W = r_A$, and both A and B obtain implicit key authentication. Additional optional fields which might be transferred in the encrypted portion include: a timestamp or sequence number to provide a freshness guarantee to B; a field containing redundancy, to provide explicit key authentication to B or facilitate message modification detection; and a target identifier to prevent undetectable message replay back on A immediately. Thus:

     $$A \rightarrow B : E_K(r_A, t_{A^*}, B^*)$$

Point-to-point key update using symmetric encryption

- If it is desired that both parties contribute to the session key, B may send A an analogous message, with the session key computed as $f(r_A; r_B)$.
- Choosing f to be a one-way function precludes control of the final key value by either party, or an adversary who acquires one of r_A, r_B.

2. *key transport with challenge-response:*

$$\begin{aligned} A &\leftarrow B : n_B \\ A &\rightarrow B : E_K(r_A, n_B, B^*) \end{aligned}$$

If a freshness guarantee is desired but reliance on timestamps is not, a random number or sequence number, denoted n_B here, may be used to replace the timestamp in the one-pass technique; the cost is an additional message. The session key is again $W = r_A$.

Point-to-point key update using symmetric encryption

- If it is required that the session key W be a function of inputs from both parties, A may insert a nonce n_A preceding n_B in (2), and a third message may be added as below.
- Here r_A, r_B are random numbers serving as keying material, while n_A, n_B are nonces for freshness.

$$A \leftarrow B : n_B \tag{1}$$
$$A \rightarrow B : E_K(r_A, n_A, n_B, B^*) \tag{2}$$
$$A \leftarrow B : E_K(r_B, n_B, n_A, A^*) \tag{3}$$

Point-to-point key update by key derivation and non-reversible functions

- Key update may be achieved by key transport as above, or by key derivation wherein the derived session key is based on per-session random input provided by one party.
- In this case, there is also a single message:

$$A \rightarrow B : r_A \tag{1}$$

- The session key is computed as $W = E_K(r_A)$. The technique provides to both A and B implicit key authentication.
- It is, however, susceptible to known-key attacks. The random number r_A here may be replaced by other time-variant parameters
- For example, a timestamp t_A validated by the recipient by comparison to its local clock provides an implicit key freshness property, provided the long-term key is not compromised.

Point-to-point key update by key derivation and non-reversible functions

- Here A could control the value of W, forcing it to be x by choosing $r_A = D_K(x)$.
- Since the technique itself does not require decryption, E may be replaced by an appropriate keyed pseudorandom function h_K, in which case the session key may be computed as $W = h_K(r_A)$, with r_A a time-variant parameter as noted above.
- A key derivation protocol which entirely avoids the use of an encryption function may offer potential advantages with respect to export restrictions.
- Next protocol is such a technique, which also provides authentication guarantees as stated. It uses two distinct functions h and h'(generating outputs of different bitlengths), respectively, for message authentication and key derivation.

Authenticated Key Exchange Protocol 2 (AKEP2)

Algorithm 1 Authenticated Key Exchange Protocol 2 (AKEP2)

SUMMARY: A and B exchange 3 messages to derive a session key W.

RESULT: mutual entity authentication, and implicit key authentication of W.

1. *Setup:* A and B share long-term symmetric keys K, K' (these should differ but need not be independent). h_K is a MAC (keyed hash function) used for entity authentication. $h'_{K'}$ is a pseudorandom permutation or keyed one-way function used for key derivation.

Authenticated Key Exchange Protocol 2 (AKEP2)

```
Algorithm 1 Authenticated Key Exchange Protocol 2 (AKEP2)
  2. Protocol messages: Define $T = (B, A, r_A, r_B)$.
        $A \to B : r_A$                              (1)
        $A \leftarrow B : T, h_K(T)$                 (2)
        $A \to B : (A, r_B), h_K(A, r_B)$            (3)
        $W = {h'}_{K'}(r_B)$
```

Authenticated Key Exchange Protocol 2 (AKEP2)

Algorithm 1 Authenticated Key Exchange Protocol 2 (AKEP2)

3. *Protocol actions:* Perform the following steps for each shared key required.
    1. A selects and sends to B a random number r_A.
    2. B selects a random number r_B and sends to A the values (B, A, r_A, r_B), along with a MAC over these quantities generated using h with key K.
    3. Upon receiving message (2), A checks the identities are proper, that the r_A received matches that in (1), and verifies the MAC.
    4. A then sends to B the values (A, r_B), along with a MAC thereon.
    5. Upon receiving (3), B verifies that the MAC is correct, and that the received value r_B matches that sent earlier.
    6. Both A and B compute the session key as $W = h'_{K'}(r_B)$.

Key transport without a priori shared keys

- Shamir's no-key algorithm is a key transport protocol which, using only symmetric techniques (although involving modular exponentiation), allows key establishment over an open channel without requiring either shared or public keys.
- Each party has only its own local symmetric key.
- The protocol provides protection from passive adversaries only; it does not provide authentication. It thus solves the same problem as basic Diffie-Hellman protocol – two parties sharing no a priori keying material end up with a shared secret key, secure against passive adversaries – although differences include that it uses three messages rather than two, and provides key transport

Shamir's no-key protocol

Algorithm 2 Shamir's no-key protocol

SUMMARY: users A and B exchange 3 messages over a public channel.

RESULT: secret K is transferred with privacy (but no authentication) from A to B.

1. *One-time setup (definition and publication of system parameters).*
   1. Select and publish for common use a prime p chosen such that computation of discrete logarithms modulo p is infeasible.
   2. A and B choose respective secret random numbers a, b, with $1 \leq a, b \leq p-2$, each coprime to $p-1$. They respectively compute a^{-1} and b^{-1} mod $p-1$.

Shamir's no-key protocol

Algorithm 2 Shamir's no-key protocol

2. *Protocol messages.*

$$A \rightarrow B : K^a \bmod p \tag{1}$$

$$A \leftarrow B : (K^a)^b \bmod p \tag{2}$$

$$A \rightarrow B : (K^{ab})^{a^{-1}} \bmod p \tag{3}$$

Shamir's no-key protocol

Algorithm 2 Shamir's no-key protocol

3. *Protocol actions.* Perform the following steps for each shared key required.
   1. A chooses a random key K for transport to $B, 1 \leq K \leq p-1$. A computes K^a mod p and sends B message (1).
   2. B exponentiates (mod p) the received value by b, and sends A message (2).
   3. A exponentiates (mod p) the received value by a^{-1} mod $p-1$, effectively "undoing" its previous exponentiation and yielding K^b mod p. A sends the result to B as message (3).
   4. B exponentiates (mod p) the received value by b^{-1} mod $p-1$, yielding the newly shared key K mod p.

Kerberos and related server-based protocols

- The key transport protocols are based on symmetric encryption, and involve two communicating parties, A and B, and a trusted server with which they share long-term pairwise secret keys a *priori.*
- In such protocols, the server either plays one of the following roles:
  - *key distribution center* (KDC) supplies the session key
  - *key translation center* (KTC) makes a key chosen by one party available to the other, by re-encrypting (translating) it under a key shared with the latter.

Kerberos authentication protocol

Kerberos is the name given to all of the following:

- the distributed authentication service originating from MIT's Project Athena, which includes specifications for data integrity and encryption
- the software which implements it, and the processes executing such software
- and the specific authentication protocol used therein.

Focus here, and use of the term "Kerberos", is restricted to the protocol itself, which supports both entity authentication and key establishment using symmetric techniques and a third party.

Kerberos authentication protocol

- The basic Kerberos protocol involves A (the *client*), B (the *server* and *verifier*), and a trusted server T (the Kerberos *authentication server*).
- At the outset A and B share no secret, while T shares a secret with each (e.g., a user password, transformed into a cryptographic key by an appropriate function).
- The primary objective is for B to verify A's identity; the establishment of a shared key is a side effect.
- Options include a final message providing mutual entity authentication and establishment of an additional secret shared by A and B (a *subsession key* not chosen by T).

Kerberos authentication protocol

The protocol proceeds as follows.

- A requests from T appropriate *credentials* (data items) to allow it to authenticate itself to B.
- T plays the role of a KDC, returning to A a session key encrypted for A and a *ticket* encrypted for B.
- The ticket, which A forwards on to B, contains the session key and A's identity; this allows authentication of A to B when accompanied by an appropriate message (the *authenticator*) created by A containing a timestamp recently encrypted under that session key.

Basic Kerberos authentication protocol (simplified)

Algorithm 3 Basic Kerberos authentication protocol (simplified)

SUMMARY: A interacts with trusted server T and party B.
RESULT: entity authentication of A to B (optionally mutual), with key establishment.

1. *Notation.* Optional items are denoted by an asterisk ($*$). E is a symmetric encryption algorithm.
   N_A is a nonce chosen by A; T_A is a timestamp from A's local clock.
   k is the session-key chosen by T, to be shared by A and B.
   L indicates a validity period (called the "lifetime").
2. *One-time setup.* A and T share a key K_{AT}; similarly, B and T share K_{BT}. Define $ticket_B \overset{\text{def}}{=} E_{K_{BT}}(k, A, L)$;
   authenticator$\overset{\text{def}}{=} E_k(A, T_A, A^*_{subkey})$.

Basic Kerberos authentication protocol (simplified)

Algorithm 3 Basic Kerberos authentication protocol (simplified)

3. *Protocol messages.*

$$A \rightarrow T : A, B, N_A \tag{1}$$

$$A \leftarrow T : ticket_B, E_{K_{AT}}(k, N_A, L, B) \tag{2}$$

$$A \rightarrow B : ticket_B, authenticator \tag{3}$$

$$A \leftarrow B : E_k(T_A, B^*_{subkey}) \tag{4}$$

Basic Kerberos authentication protocol (simplified)

Algorithm 3 Basic Kerberos authentication protocol (simplified)

4. *Protocol actions.* Algorithm E includes a built-in integrity mechanism, and protocol failure results if any decryption yields an integrity check failure.
    4.1. A generates a nonce N_A and sends to T message (1).
    4.2. T generates a new session key k, and defines a validity period (lifetime L) for the ticket, consisting of an ending time and optional starting time. T encrypts k, the received nonce, lifetime, and received identifier (B) using A's key. T also creates a ticket secured using B' s key containing k, received identifier (A), and lifetime. T sends A message (2).

Basic Kerberos authentication protocol (simplified)

Algorithm 3 Basic Kerberos authentication protocol (simplified)

4.3. A decrypts the non-ticket part of message (2) using K_{AT} to recover: k, N_A, lifetime L, and the identifier of the party for which the ticket was actually created. A verifies that this identifier and N_A match those sent in message (1),and saves L for reference. A takes its own identifier and fresh timestamp T_A, optionally generates a secret A_{subkey}, and encrypts these using k to form the authenticator. A sends to B message (3).

4.4. B receives message (3), decrypts the ticket using K_{BT} yielding k to allow decryption of the authenticator. B checks that:

1. the identifier fields (A) in the ticket and authenticator match;
2. the timestamp T_A in the authenticator is valid; and
3. B's local time is within the lifetime L specified in the ticket.

If all checks pass, B declares authentication of A successful, and saves A_{subkey} (if present) as required.

Basic Kerberos authentication protocol (simplified)

Algorithm 3 Basic Kerberos authentication protocol (simplified)

```
4.5. (Optionally for mutual entity authentication:) $B$ constructs
     and sends to $A$ message (4) containing $A'$s timestamp from
     the authenticator (specifically excluding the identifier $A$, to
     distinguish it from the authenticator), encrypted using $k$. $B$
     optionally includes a subkey to allow negotiation of a
     subsession key.
4.6. (Optionally for mutual entity authentication:) $A$ decrypts
     message (4). If thetimestamp within matches that sent in
     message (3), $A$ declares authentication of $B$ successful and
     saves $B_{subkey}$ (if present) as required.
```

Basic Kerberos authentication protocol (simplified)

- Since timestamps are used, the hosts on which this protocol runs must provide both secure and synchronized clocks.
- If, as is the case in actual implementations, the initial shared keys are password-derived, then the protocol is no more secure than the secrecy of such passwords or their resistance to password-guessing attacks.
- Optional parameters A_{subkey} and B_{subkey} allow transfer of a key (other than k) from A to B or vice-versa, or the computation of a combined key using some function $f(A_{subkey}, B_{subkey})$.
- The lifetime within the ticket is intended to allow A to re-use the ticket over a limited time period for multiple authentications to B without additional interaction with T, thus eliminating messages (1) and (2). For each such re-use, A creates a new authenticator with a fresh timestamp and the same session key k, the optional subkey field is of greater use in this case.

Otway-Rees protocol

- The Otway-Rees protocol is a server-based protocol providing authenticated key transport (with key authentication and key freshness assurances) in only 4 messages – the same as Kerberos, but here without the requirement of timestamps.
- It does not, however, provide entity authentication or key confirmation.

Otway-Rees protocol

Algorithm 4 Otway-Rees protocol

SUMMARY: B interacts with trusted server T and party A.
RESULT: establishment of fresh shared secret k between A and B.

1. *Notation:*
   E is a symmetric encryption algorithm.
   k is the session-key T generates for A and B to share.
   N_A and N_B are nonces chosen by A and B, respectively, to allow verification of key freshness.
   M is a second nonce chosen by A which serves as a transaction identifier.
2. *One-time setup.* T shares symmetric keys K_{AT} and K_{BT} with A, B, respectively.

Otway-Rees protocol

Algorithm 4 Otway-Rees protocol

3. *Protocol messages.*

$$A \to B : M, A, B, E_{K_{AT}}(N_A, M, A, B) \tag{1}$$
$$B \to T : M, A, B, E_{K_{AT}}(N_A, M, A, B), E_{K_{BT}}(N_B, M, A, B) \tag{2}$$
$$B \leftarrow T : E_{K_{AT}}(N_A, k), E_{K_{BT}}(N_B, k) \tag{3}$$
$$A \leftarrow B : E_{K_{AT}}(N_A, k) \tag{4}$$

Otway-Rees protocol

Algorithm 4 Otway-Rees protocol

4. *Protocol actions.* Perform the following steps each time a shared key is required.
   4.1. A encrypts data for the server containing two nonces, N_A and M, and the identities of itself and the party B to whom it wishes the server to distribute a key. A sends this and some plaintext to B in message (1).
   4.2. B creates its own nonce N_B and an analogous encrypted message (with the same M), and sends this along with A's message to T in message (2).

Otway-Rees protocol

Algorithm 4 Otway-Rees protocol

4.3. T uses the clear text identifiers in message (2) to retrieve K_{AT} and K_{BT}, then verifies the cleartext (M, A, B) matches that recovered upon decrypting both parts of message (2). (Verifying M in particular confirms the encrypted parts are linked.) If so, T inserts a new key k and the respective nonces into distinct messages encrypted for A and B, and sends both to B in message (3).

4.4. B decrypts the second part of message (3), checks N_B matches that sent in message (2), and if so passes the first part on to A in message (4).

4.5. A decrypts message (4) and checks N_A matches that sent in message (1).

Otway-Rees protocol

- If all checks pass, each of A and B are assured that k is fresh (due to their respective nonces), and trust that the other party T shared k with is the party bound to their nonce in message (2). A knows that B is active as verification of message (4) implies B sent message (2) recently; B however has no assurance that A is active until subsequent use of k by A, since B cannot determine if message (1) is fresh.
- (nonces in Otway-Rees protocol) The use of two nonces generated by A is redundant (N_A could be eliminated in messages (1) and (2), and replaced by M in (3) and (4)),but nonetheless allows M to serve solely as an administrative transaction identifier, while keeping the format of the encrypted messages of each party identical. (The latter is generally considered desirable from an implementation view point, but dubious from a security viewpoint.)

Otway-Rees protocol

- (extension of Otway-Rees protocol) Otway-Rees protocol may be extended to provide both key confirmation and entity authentication in 5 messages. Message (4) could be augmented to both demonstrate B's timely knowledge of k and transfer a nonce to A (e.g., appending $E_k(N_A, N_B)$), with a new fifth message $(A \to B : E_k(N_B))$ providing B reciprocal assurances.