Public Key Cryptography

Lecture 10

Key Establishment Protocols

Index

Generalities

Key distribution: a fundamental problem that motivated public-key cryptography.

Vocabulary

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

Generalities (cont.)

Vocabulary

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

Generalities (cont.)

Protocols

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

Key establishment protocols result in shared secrets which are typically called, or used to derive, *session keys*. Ideally, their use are restricted to a short time period such as a single session, after which any trace of it is eliminated.

Motivation for use of session keys

- to limit available ciphertext (under a fixed key) for cryptanalytic attack
- to limit exposure, with respect to both time period and quantity of data, in the event of (session) key compromise
- to avoid long-term storage of a large number of distinct secret keys, by creating keys only when actually required
- to create independence across communications sessions or applications.

It is also desirable in practice to avoid the requirement of maintaining state information across sessions.

Key transport based on symmetric encryption

Examples of such protocols

- Point-to-point key update
- Shamir's no-key protocol
- Kerberos

Point-to-point key update by symmetric encryption

- It makes use of a long-term symmetric key K shared a priori by two parties A and B.
- The key, initially distributed over a secure channel or resulting from a key pre-distribution scheme, is used repeatedly to establish new session keys W.
- Notation:
  r_A: a random number
  t_A: timestamp
  n_A: sequence number generated by A
  E: a symmetric encryption algorithm
  Optional message fields are denoted by an asterisk (*).

Point-to-point key update by symmetric encryption (cont.)

1. Key transport with one pass

$$A \rightarrow B : E_K(r_A) \tag{1}$$

The session key used is $W = r_A$ and both A and B obtain implicit key authentication.
Additional optional fields include: a timestamp, a field containing redundancy, a target identifier. Thus:

$$A \rightarrow B : E_K(r_A, t_A*, B*) \tag{2}$$

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 $f(r_A, r_B)$.

2. Key transport with challenge-response

$$A \leftarrow B \quad : \quad n_B \tag{1}$$
$$A \rightarrow B \quad : \quad E_K(r_A, n_B, B*) \tag{2}$$

The session key used is $W = r_A$.
If it is desired that both parties contribute to the session key, A may insert a nonce (*number used once*) n_A preceding n_B and a third message is added as below.

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

Key transport without a priori shared keys

Shamir's no-key protocol

- *Summary:* A and B exchange 3 messages over a public channel.
- *Result:* secret key K is transferred with privacy (but no authentication) from A to B.
- *1. One-time setup (definition and publication).*

  (a) Select and publish for common use a prime p chosen such that computation of discrete logarithms modulo p is infeasible.

  (b) A and B choose respective secret random numbers a, b, with $1 \leq a, b \leq p-2$, each relatively prime to $p-1$. They respectively compute a^{-1} and $b^{-1} \bmod p-1$.
- *2. Protocol messages.*

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

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

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

Key transport without a priori shared keys (cont.)

Shamir's no-key protocol

- *3. Protocol actions.* Perform the following steps for each shared key required.

  (a) A chooses a random key K for transport to B, $1 \leq K \leq p-1$. A computes $K^a \bmod p$ and sends B the result.

  (b) B exponentiates $(\bmod p)$ the received value by b, and sends A the result.

  (c) A exponentiates $(\bmod p)$ the received value by $a^{-1} \bmod p-1$, yielding $K^b \bmod p$. A sends the result to B.

  (d) B exponentiates $(\bmod p)$ the received value by $b^{-1} \bmod p-1$, yielding the newly shared key $K \bmod p$.

Basic Kerberos protocol

- It 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, T shares a secret with each.
- 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).

The protocol proceeds as follows.

- A requests from T appropriate credentials 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 protocol (cont.)

Basic Kerberos protocol

- *Summary:* A interacts with trusted server T and party B.
- *Result:* entity (mutual) authentication of A to B, 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 *lifetime*.
- *2. One-time setup.*
  A and T share a key K_{AT}; similarly, B and T share K_{BT}.
  Define $ticket_B = E_{K_{BT}}(k, A, L)$; $authenticator = E_k(A, T_A, A^*_{subkey})$.
- *3. Protocol messages.*

$$
\begin{aligned}
A \rightarrow T &: A, B, N_A && (1)\\
A \leftarrow T &: ticket_B, E_{K_{AT}}(k, N_A, L, B) && (2)\\
A \rightarrow B &: ticket_B, authenticator && (3)\\
A \leftarrow B &: E_k(T_A, B^*_{subkey}) && (4)
\end{aligned}
$$

Basic Kerberos protocol (cont.)

Basic Kerberos protocol

- *4. Protocol actions.* Algorithm E includes a built-in integrity mechanism, and protocol failure results if any decryption yields an integrity check failure.

  (a) A generates a nonce N_A and sends to T message (1).

  (b) T generates a new session key k, and defines a 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 to A message (2).

  (c) 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).

Basic Kerberos protocol (cont.)

Basic Kerberos protocol

- *4. Protocol actions.*

  (d) B receives message (3), decrypts the ticket using K_{BT} yielding k to allow decryption of the authenticator. B checks that: the identifier fields (A) in the ticket and authenticator match; the timestamp T_A is valid; 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.

  (e) (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.

  (f) (Optionally for mutual entity authentication)

  A decrypts message (4). If the timestamp within matches that sent in message (3), A declares authentication of B successful and saves B_{subkey} (if present) as required.

Key agreement based on symmetric techniques

Definition

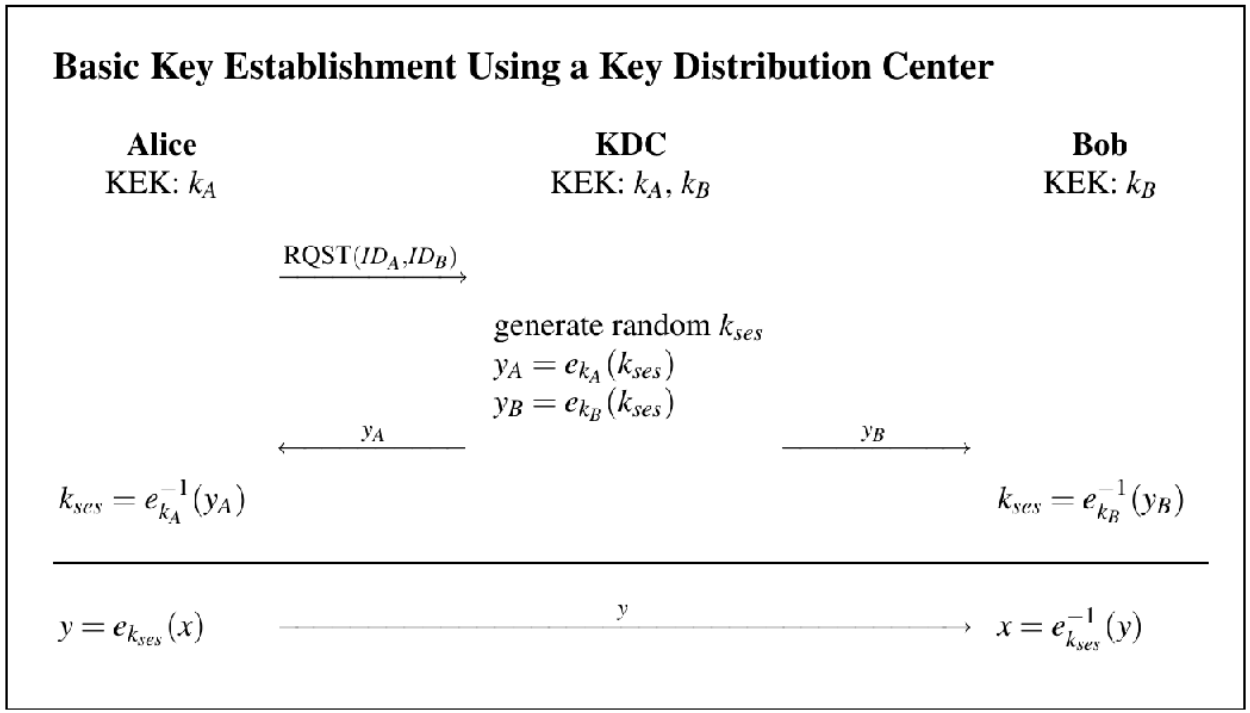
A *key distribution system* (KDS) is a method whereby, during an initialization stage, a trusted server generates and distributes secret data values (pieces) to users, such that any pair of users may subsequently compute a shared key unknown to all others (aside from the server).

For fixed pairwise keys, a KDS is a key pre-distribution scheme.

Example. A trivial KDS is as follows:

- The trusted server chooses distinct keys for each pair among the n users, and by some secure means initially distributes to each user its $n - 1$ keys appropriately labelled.
- This theoretically provides perfect security, but due to the large amount of storage required, alternate methods are sought.

Key establishment using a Key Distribution Center



Security of a Key Distribution Center

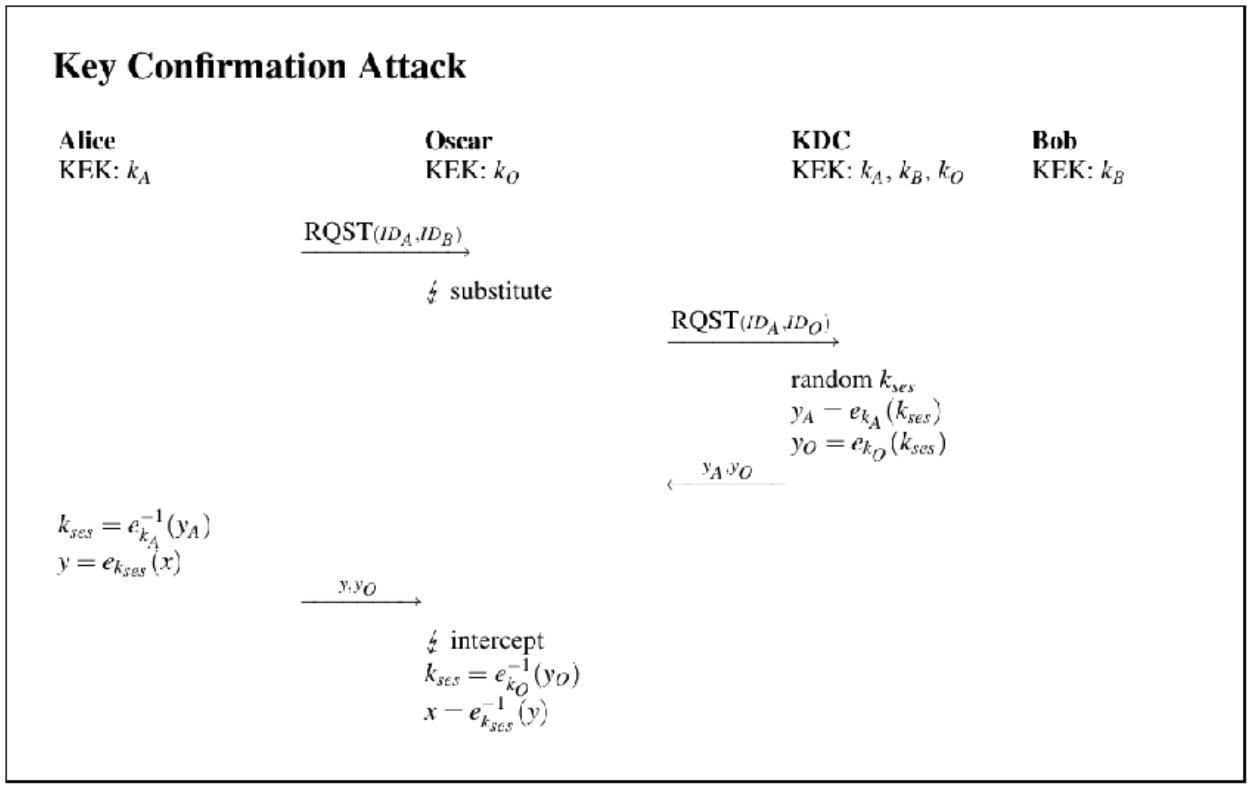
Replay Attack

- Neither Alice nor Bob know whether the encrypted session key they receive is actually a new one.
- This can be a particularly serious issue if an old session key has become compromised.
- If Oscar gets a previous session key, he can impersonate the KDC and resend old messages y_A and y_B to Alice and Bob. Since Oscar knows the session key, he can decipher the plaintext that will be encrypted by Alice or Bob.

Key Confirmation Attack

- Alice is not assured that the key she receives from the KDC is actually for a session between her and Bob.
- This attack assumes that Oscar is also a legitimate (but malicious) user.
- By changing the session-request Oscar can trick the KDC and Alice to set up session between him and Alice instead of Bob.

Security of a Key Distribution Center



Key transport based on public-key encryption

Protocols without signature:

- Needham-Schroeder

Protocols with signature:

- X.509

Protocols without signature

They are appropriate for one-way communications and store-and-forward applications such as electronic mail and fax.

Needham-Schroeder public-key protocol

- *Summary:* A and B exchange 3 messages.
- *Result:* entity authentication, key authentication, and key transport (all mutual).
- *1. Notation.*

  $P_X(Y)$ denotes public-key encryption of data Y using party X's public key;

  $P_X(Y_1, Y_2)$ denotes the encryption of the concatenation of Y_1 and Y_2;

  k_1, k_2 are secret symmetric session keys chosen by A and B respectively.
- *2. One-time setup.* Assume A, B possess each other's authentic public-key.

Protocols without signature (cont.)

Needham-Schroeder public-key protocol

- *3. Protocol messages.*

$$A \to B \quad : \quad P_B(k_1, A) \tag{1}$$
$$A \leftarrow B \quad : \quad P_A(k_1, k_2) \tag{2}$$
$$A \to B \quad : \quad P_B(k_2) \tag{3}$$

- *4. Protocol actions.*

  (a) A sends B message (1).

  (b) B recovers k_1 upon receiving message (1), and returns to A message (2).

  (c) Upon decrypting message (2), A checks the key k_1 recovered agrees with that sent in message (1). A sends B message (3).

  (d) Upon decrypting message (3), B checks the key k_2 recovered agrees with that sent in message (2). The session key may be computed as $f(k_1, k_2)$ using an appropriate publicly known non-reversible function f.

Protocols with signature

Types:

- sign the key, then public-key encrypt the signed key
- sign the key, and separately public-key encrypt the (unsigned) key
- public-key encrypt the key, then sign the encrypted key

Examples for the last type are $X.509$ strong authentication protocols. Both of them were designed to provide the following to the responder B.

- the identity of A, and that the token (i.e. cryptographically protected data) received by B was constructed by A (and not thereafter altered)
- the token received by B was specifically intended for B
- the token received by B has "freshness"
- the mutual secrecy of the transferred key

Protocols with signature (cont.)

X.509 strong two-way authentication (two-pass)

- *Summary:* A sends B one message, and B responds with one message.
- *Result:* mutual entity authentication and key transport with key authentication.
- *1. Notation.*

  $P_X(y)$ denotes the result of applying X's encryption public key to data y.
  $S_X(y)$ denotes the result of applying X's signature private key to y.
  r_A, r_B are never re-used numbers (to detect replay and impersonation).
  $cert_X$ is a certificate binding party X to a public key suitable for both encryption and signature verification.
- *2. System setup.*

  (a) Each party has its public key pair for signatures and encryption.
  (b) A must acquire (and authenticate) the encryption public key of B.
- *3. Protocol messages.* (An asterisk denotes items that are optional.)

  Let $D_A = (t_A, r_A, B, data_1^*, P_B(k_1)^*)$, $D_B = (t_B, r_B, A, r_A, data_2^*, P_A(k_2)^*)$.

$$A \rightarrow B \quad : \quad cert_A, D_A, S_A(D_A) \tag{1}$$

$$A \leftarrow B \quad : \quad cert_B, D_B, S_B(D_B) \tag{2}$$

Protocols with signature (cont.)

X.509 strong two-way authentication (two-pass)

- *4. Protocol actions.*

  (a) A obtains a timestamp t_A indicating an expiry time, generates r_A, optionally obtains a symmetric key k_1 and sends to B message (1). ($data_1$ is optional for data origin authentication.)

  (b) B verifies the authenticity of $cert_A$, extracts A's signature public key, and verifies A's signature on the data block D_A. B then checks that the identifier in message (1) specifies itself as intended recipient, that the timestamp is valid, and checks that r_A has not been replayed.

  (c) If all checks succeed, B declares the authentication of A successful, decrypts k_1 using its private decryption key, and saves this now-shared key. (This terminates the protocol if only unilateral authentication is desired.) B then obtains timestamp t_B, generates r_B, and sends A message (2). ($data_2$ is optional data, and k_2 is an optional symmetric key provided for A.)

  (d) A carries out actions analogous to those of B. If all checks succeed, A declares the authentication of B successful, and saves key k_2 for subsequent use. A and B share mutual secrets k_1 and k_2.

Protocols with signature (cont.)

X.509 strong three-way authentication (three-pass)

- *Summary:* A and B exchange 3 messages.
- *Result:* as in the previous protocol, without requiring timestamps.
- The protocol differs from the previous one as follows:

  1. Timestamps t_A and t_B may be set to zero, and need not be checked.
  2. Upon receiving (2), A checks the received r_A matches that in message (1).
  3. A third message is sent from A to B:

$$A \rightarrow B \quad : \quad (r_B, B), S_A(r_B, B) \tag{3}$$

  4. Upon receiving (3), B verifies the signature matches the received plaintext, that plaintext identifier B is correct, and that plaintext r_B received matches that in (2).

Key agreement based on asymmetric techniques

Diffie-Hellman and related key agreement protocols:

- Diffie-Hellman
- ElGamal

Key agreement based on asymmetric techniques (cont.)

Diffie-Hellman key agreement (basic version)

- *Summary:* A and B each sends one message over an open channel.
- *Result:* shared secret K known to both parties A and B.
- *1. One-time setup.* An appropriate prime p and generator α of $\mathbb{Z}_p^*$ ($2 \leq \alpha \leq p-2$) are selected and published.
- *2. Protocol messages.*

$$A \rightarrow B \quad : \quad \alpha^x \bmod p \tag{1}$$

$$A \leftarrow B \quad : \quad \alpha^y \bmod p \tag{2}$$

- *3. Protocol actions.* Perform the following steps each time a shared key is required.

  (a) A chooses a random secret x, $1 \leq x \leq p-2$, and sends B message (1).

  (b) B chooses a random secret y, $1 \leq y \leq p-2$, and sends A message (2).

  (c) B receives α^x and computes the shared key as $K = (\alpha^x)^y \bmod p$.

  (d) A receives α^y and computes the shared key as $K = (\alpha^y)^x \bmod p$.

Key agreement based on asymmetric techniques (cont.)

ElGamal key agreement (half-certified Diffie-Hellman)

- *Summary:* A sends to B one message for one-pass key agreement.
- *Result:* shared secret K known to both parties A and B.
- *1. One-time setup.* Each user B does the following:
  Pick an appropriate prime p and generator α of $\mathbb{Z}_p^*$.
  Select a random integer b, $1 \leq b \leq p-2$, and compute $\alpha^b \bmod p$.
  B publishes its public key (p, α, α^b), keeping private key b secret.
- *2. Protocol messages.*

$$A \rightarrow B \quad : \quad \alpha^x \bmod p \tag{1}$$

- *3. Protocol actions.* Perform the following steps each time a shared key is required.
  (a) A obtains an authentic copy of B's public key (p, α, α^b).
  A chooses a random integer x, $1 \leq x \leq p-2$, and sends B message (1).
  A computes the key as $K = (\alpha^b)^x \bmod p$.
  (b) B computes the same key on receipt of message (1) as $K = (\alpha^x)^b \bmod p$.

Man-in-the-Middle Attack Against the DHKE

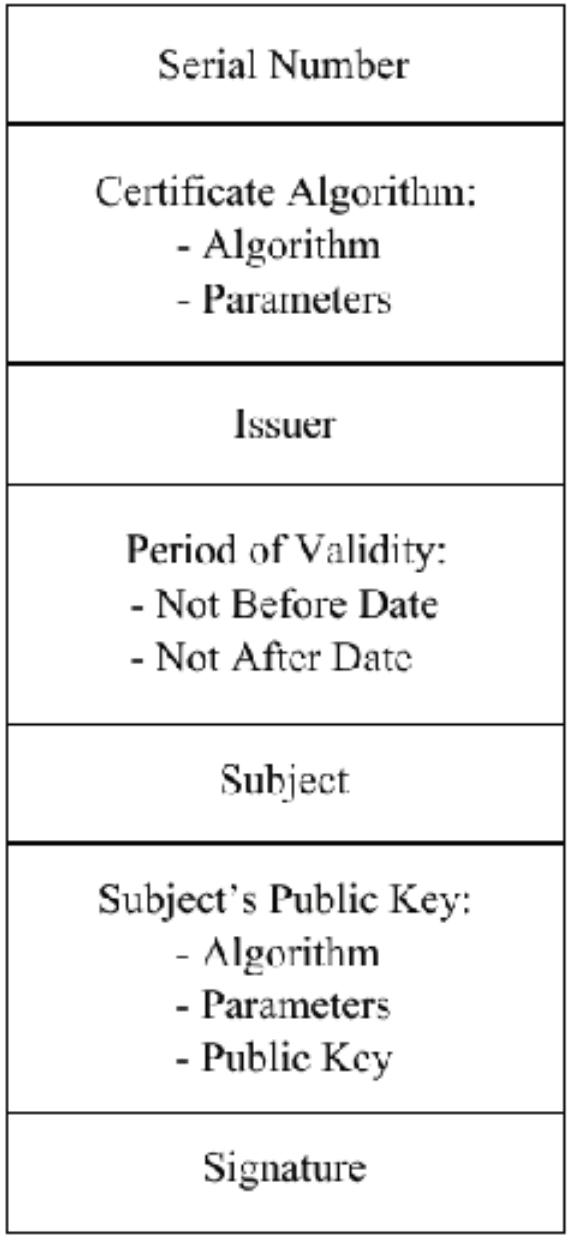
Man-in-the-Middle Attack Against the DHKE

Alice	**Oscar**	**Bob**
choose $a = k_{pr,A}$		choose $b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$		$B = k_{pub,B} \equiv \alpha^b \bmod p$
$\xrightarrow{A}$	substitute $\tilde{A} \equiv \alpha^o$	$\xrightarrow{\tilde{A}}$
$\xleftarrow{\tilde{B}}$	substitute $\tilde{B} \equiv \alpha^o$	$\xleftarrow{B}$
$k_{AO} \equiv (\tilde{B})^a \bmod p$	$k_{AO} \equiv A^o \bmod p$ $k_{BO} \equiv B^o \bmod p$	$k_{BO} \equiv (\tilde{A})^b \bmod p$

Diffie–Hellman Key Exchange with Certificates

Alice		**Bob**
$a = k_{pr,A}$		$b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$		$B = k_{pub,B} \equiv \alpha^B \bmod p$
$\text{Cert}_A = [(A, ID_A), s_A]$		$\text{Cert}_B = [(B, ID_B), s_B]$
	$\xrightarrow{\text{Cert}_A}$	
	$\xleftarrow{\text{Cert}_B}$	
verify certificate:		verify certificate:
$\text{ver}_{k_{pub,CA}}(\text{Cert}_B)$		$\text{ver}_{k_{pub,CA}}(\text{Cert}_A)$
compute session key:		compute session key:
$k_{AB} \equiv B^a \bmod p$		$k_{AB} \equiv A^b \bmod p$

X.509 Certificates



Quantum bit generator

- Bennett and Brassard (1984)
- it works for limited distances (tens of kilometers)
- Outline:

  Alice transmits to Bob a string of binary digits. Alice sends 0 and 1 using photons with different polarisations.

  Alice has 2 available schemes to associate polarisation of photons with 0 and 1. In the *rectilinear scheme* (or scheme $+$) she sends $|$ for 1 and $-$ for 0. In the *diagonal scheme* (or scheme $\times$) she sends $/$ for 1 and $\backslash$ for 0. For each photon/bit Alice changes randomly the two schemes.

  Bob measures the polarisation of photons with one of his detectors $+$ or $\times$ to get the bits. Acording to quantum physics, at a certain moment Bob can measure only one of the polarisations, using one of his detectors.

Quantum bit generator (cont.)

Algorithm:

1. Alice starts to transmit a string of photons/bits using randomly one of her two polarisation schemes (rectilinear [R] or diagonal [D]).
2. Bob measures the polarisation of these photons to obtain the correct bits. Since he does not know the polarisation scheme chosen by Alice, he chooses randomly between his detectors $+$ and $\times$. Sometimes he chooses right, and sometimes wrong.
3. In order to clarify the correct bits, Alice and Bob talk to each other on an open communication line. Alice tells Bob which polarisation scheme she used for each bit (but not the polarisation way!), and Bob tells her in which cases he used the correct detector. The random string consists of the resulting bits when Bob used a correct detector.

Quantum bit generator (cont.)

The table of possibilities of choice for the polarisation schemes

Alice's scheme	R	R	R	R	R	R	D	D	D	D	D	D
Alice's bit	1	1	1	0	0	0	1	1	1	0	0	0
Alice transmits	|	|	|	–	–	–	⟋	⟋	⟋	⟍	⟍	⟍
Bob's detector	+	×	×	+	×	×	+	+	×	+	+	×
correct detector?	yes	no	no	yes	no	no	no	no	yes	no	no	yes
Bob detects	|	⟋	⟍	–	⟋	⟍	|	–	⟋	|	–	⟍
Bob's bit	1	1	0	0	1	0	1	0	1	1	0	0
correct bit?	yes	yes	no	yes	no	yes	yes	no	yes	no	yes	yes
resulting bit	1			0					1			0

Selective Bibliography


A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
[http://www.cacr.math.uwaterloo.ca/hac]


C. Paar, J. Pelzl, *Understanding Cryptography*, Springer, 2009.


B. Schneier, *Applied Cryptography*, John Wiley and Sons, 1996.


S. Singh, *The Code Book on CD-ROM*.
[http://www.simonsingh.net/The_CDROM.html]