Cryptography

Various protocols

Professor: Marius Zimand

Key Distribution

We have seen the Diffie-Hellman key exchange protocol.

It is vulnerable to the **man-in-the-middle attack.**

Eve sits in the middle and impersonates Bob when talking to Alice and impersonates Alice when talking to Alice.

Solution: in the protocol Alice and Bob need to authenticate themselves.

Station-to-Station Protocol

- uses a Trusted Authority (TA)
- uses some encryption scheme (E_k, D_k) (ex: DES, RSA, ...)
- uses a signature scheme

Protocol:

1. p - prime, α - primitive root of p. these are public.
2. Alice chooses random x_A, Bob chooses random x_B.
3. Alice computes $\alpha^{X_A} (\mod p)$, Bob computes $\alpha^{X_B} (\mod p)$.
4. Alice sends Bob: $\alpha^{X_A} (\mod p)$.
5. Bob computes $K = (\alpha^{X_A})^{X_B} (\mod p)$.
6. Bob sends Alice: $\alpha^{X_B} (\mod p)$ and $E_K(sign_B(\alpha^{X_B}, \alpha^{X_A}))$.
7. Alice computes $K = (\alpha^{X_B})^{X_A} (\mod p)$.
8. Alice decrypts $E_K(sign_B(\alpha^{X_B}, \alpha^{X_A}))$ and gets $sign_B(\alpha^{X_B}, \alpha^{X_A})$.
9. Alice requests from TA ver_B (verification algorithm for $sign_B$).
10. Alices uses ver_B to validate Bob's signature.
11. Alice sends Bob: $E_K(sign_A(\alpha^{X_A}, \alpha^{X_B}))$.
12. Bob does as Alice to validate Alice's signature.

Commmunication with the TA has to be secure (no man-in-the-middle). This is easier to ensure, because the TAs are few and specialized for this (as opposed to Alice and Bob).

Identification protocols

- to log in to a computer
- ATM
- smart door, etc.

General approach: challenge-response.

Assume Alice wants to identify herself.

- Alice has a secret key K_A.
- Bob (server, ATM, door, ...) gives her a challenge that can be solved only by someone who knows the secret K_A.

(1) K_A is a password.

(a) Server stores a table of passwords.

Alice	password Alice
Bob	password Bob
Charles	password Charles
...	...

This schema is not too good: whoever sees the table, gets the secret(s).

(b) Better approach: use a one-way function h.

Alice	h(password Alice)
Bob	h(password Bob)
Charles	h(password Charles)
...	...

Alice sends to the server: h(password Alice).

This schema is vulnerable to the replay attack: Eve also sends h(password Alice).

The replay attack is more general. Here is another variant of the replay attack.

Alice $\xrightarrow{E_K(\text{Give Eve \$ 100})}$ Bob.

Eve replays this several times.

(c) **ssh identification protocol** (widely used)

Preparations:

- user (Alice) generates an RSA (or DSA) key pair (public key, private key).
- she stores the public key on the remote Host
- she keeps the secret key secret

Protocol

1. Alice $\longrightarrow$ Host: I am Alice and I want to login in.
2. Host $\longrightarrow$ Alice: Host generates a random challenge r and sends it to Alice.
3. Alice $\longrightarrow$ Host: Alice responds by signing r, i.e., she sends $sign_{Alice}(r)$ to Host.

4. Host: validates $sign_{Alice}(r)$.

This prevents the replay attack.

Weak point: Alice signs r chosen by the Host.

$sign_{Alice}(r) = D_{K_A}(r)$, where K_A is Alice's secret key.

So now the Host knows: r and $D_{K_A}(r)$.

This opens the possibility of a chosen plaintext attack.

Principle: "Don't use a secret key on something that is entirely not yours."

Schnorr's identification protocol

Parameters:

- p - prime.
- q - prime, q divides $p - 1$.
- g - number such that $g^q = 1 (\mod p)$.
- $x \in \{0, 1, \ldots, q - 1\}$, randomly chosen.
- $X = g^x (\mod p)$.
- p, q, g - public
- x -secret key, X - public key.

Idea: Alice identifies herself by showing that she knows x.

Protocol:

1. Alice generates random $r \in \{0, 1, \ldots, q-1\}$. She calculates $R = g^r (\mod p)$ and sends R to Host.

2. Host generates $c \in \{0, 1, \ldots, q-1\}$. Sends c to Alice. (c is the challenge).

3. Alice calculates $z = cx + r (\mod q)$ and sends z to the Host (z is the response to the challenge).

   Note: $g^z = g^{cx+r} (\mod p)$.

   Note: z depends on c (chosen by Host) and r and x (chosen by Alice).

4. Host validates z by checking that

$$g^z = R \cdot X^c (\mod p).$$

   Justification:

$$g^z = g^{cx+r} = (g^x)^c \cdot g^r = X^c \cdot R (\mod p).$$

Suppose Eve wants to impersonate Alice.

Eve needs (z, r) so that

$$z = cx + r (\mod q)$$

which means that

$$z - r = c \cdot x.$$

If x is not known, then $c \cdot x$ can be anything in the range $\{0, \ldots, q-1\}$ equally likely.

So the probability of guessing correctly is $1/q$.

Remarkable property of Schnorr's protocol: it is **ZERO-KNOWLEDGE**. This means that Alice does not leak any information about her secret.

Definition 1 *(INFORMAL) A zero-knowldege proof of knowledge is a protocol by which Alice convinces Host that she knows a secret x without the Host learning anything about x that he did not know before the execution of the protocol.*

In the **ssh** protocol, the Host was getting pairs

$$(r, sign_{Alice}(r))$$

He did not have this before and could not produce such pairs. So ssh is not zero-knowledge.

Example: Zero-knowledge proof for SUDOKU:

`http://blog.computationalcomplexity.org/2006/08/zero-knowledge-sudoku.html`

In the Schnorr protocol, let's see what the Host is seeing

1. A $\longrightarrow$ H: R
2. H $\longrightarrow$ A: c
3. A $\longrightarrow$ H: z

and the protocol succeeds if $g^z = R \cdot X^c (\mod p)$.

Note that R, z, c, taken separately are just random numbers.

Host can produce random c and z and computes $R = g^z \cdot X^{-c} (\mod p)$.

Thus he can mimic the protocol without Alice's help.

So Host does not learn anything new during the protocol.

Fiat-Shamir protocol

Parameters:

- p, q - large prime numbers.
- $n = p \cdot q$.
- Alice's private key: random $x \in \{0, 1, \ldots, n-1\}$.

  Alice's public key: $X = x^2 (\mod n)$.

Recall: finding sqrt mod n is hard- as hard as factoring n.

Protocol (Alice proves that she knows x without leaking anything about x):

1. Alice: generates random r, computes $R = r^2 (\mod n)$, sends R to Host.
2. Host generates random bit b and sends it to Alice. b is the challenge.
3. if $b = 0$, Alice responds with $z = r$ (which is $\sqrt{R}$).

   if $b = 1$, Alice responds with $z = r \cdot x$ (which is $\sqrt{RX}$).
4. if $b = 0$, Host Checks that $z^2 = R$.

   if $b = 1$, Host Checks that $z^2 = RX$.

Why does this convince Host that Alice knows x?

The transcript of the protocol is:

1. A $\longrightarrow$ H: R
2. H $\longrightarrow$ A: b
3. A $\longrightarrow$ H: z

Test

if $b = 0$, then check if $z^2 = R \pmod{n}$.

if $b = 1$, then check if $z^2 = RX \pmod{n}$.

Let's take Eve. Eve does not know x.

Eve can produce good z's, if she knows the challenge b.

1. if $b = 0$, Eve chooses random r, sends $R = r^2$ and $z = r$.
2. if $b = 1$, Eve chooses random r, sends $R = r^2 \cdot X^{-1}$ and $z = r$.

But Eve can predict b only with prob. 1/2. So with prob. 1/2, Host will see that she is not Alice. If the protocol is repeated k times, the prob. that Eve fools the Host every time is $1/2^k$.

On the other hand, the Host can predict the challenge. So he can produce the transcript of a successful run of a protocol without knowing x.

This means that the Fiat-Shamir protocol is **zero-knowledge**.

Symmetric-key protocols

Symmetric-key crypto is still very important (faster than PK crypto).

1. identification

2. key establishment

We use some crypto primitives: encryption, signature, hashing, etc.

We assume that the primitives are secure.

When we combine them into protocols: the protocols are not necessarily secure.

Concept: nonce (number used once): typically a random number chosen from a huge domain.

Notation:

- N_A: Alice's nonce.

- N_B: Bob's nonce.

- S: trusted third authority

- KAB: symmetric key shared by Alice and Bob.

Example 1. Mutual identification protocol.

1. $A \longrightarrow B$: A, N_A (N_A acts like a challenge for B).

2. $B \longrightarrow A$: $E_{KAB}(N_A), N_B$ (response to the challenge and challenge for A).

3. $A \longrightarrow B$: $E_{KAB}(N_B)$.

Looks ok, but is flawed, because it is vulnerable to the reflection attack.

Reflection attack E will be identified by B as being A.

1. $E(A) \longrightarrow B$: A, N_E.
2. $B \longrightarrow E(A)$: $E_{KAB}(N_E), N_B$
3. $E(A) \longrightarrow B$: A, N_B. (initiates a new protocol)
4. $B \longrightarrow E(A)$: $E_{KAB}(N_B), N'_B$
5. $E(A) \longrightarrow B$: $E_{KAB}(N_B)$.

Prevention of the reflection attack

1. $A \longrightarrow B$: A, N_A
2. $B \longrightarrow A$: $E_{KAB}(A, N_A), N_B$
3. $A \longrightarrow B$: $E_{KAB}(B, N_B)$.

Note: Eve needs $E_{KAB}(B, N_B)$ and she will not get this in a rflection attack. There is still a problem: B encrypts a message chosen by A $\longrightarrow$ possibility of a chosen-plaintext attack.

Final version: ISO 9798-2 protocol

1. $A \longrightarrow B$: A, N_A
2. $B \longrightarrow A$: $E_{KAB}(A, N_A, N_B)$
3. $A \longrightarrow B$: $E_{KAB}(N_B, N_A)$.

Key Establishment Protocols using symmetric crypto

- use a trusted authority, also called sometimes KDC (Key distribution center)

- each user share with the KDC a key

- if n users, there are n shared keys (instead of $n(n-1)/2$ if each pair of users share a key).

Protocol 1. (S is the KDC, and T_S - time stamp issued by S)

1. $A \longrightarrow S$: A, B

2. $S \longrightarrow A$: $E_{KAS}(K_{AB}, T_S)$, $E_{KBS}(K_{AB}, T_S)$

3. $A \longrightarrow B$: $E_{KBS}(K_{AB}, T_S)$.

Notes:

K_{AB} is created by S (sometimes called a "ticket.")

Both A and B check that the time stamp is recent.

This protocol does not provide identification.

Protocol 2 (we add identification)

1. $A \longrightarrow S$: A, B

2. $S \longrightarrow A$: $E_{KAS}(A, B, K_{AB}, T_S)$, $E_{KBS}(A, B, K_{AB}, T_S)$

3. $A \longrightarrow B$: $E_{KBS}(A, B, K_{AB}, T_S)$.

Motes:

Now A and B know explicitely with whom they share the key.

This follows the Abadi-Needham Principle 1.

Every message in a protocol should say explicitely what it means. The naming should not depend on the context (history) of the protocol.

So message 2 says: S has created K_{AB} at time T_S to be shared by A and B.

Needham-Schroeder protocol

-early protocol (1978), forerunner of Kerberos.

1. $A \longrightarrow S$: A, B, N_A
2. $S \longrightarrow A$: $E_{KAS}(N_A, B, K_{AB}, E_{KBS}(K_{AB}, A))$.
3. $A \longrightarrow B$: $E_{KBS}(K_{AB}, A)$.
4. $B \longrightarrow A$: $E_{KAB}(N_B)$.
5. $A \longrightarrow B$: $E_{KAB}(N_B + 1)$.

Looks ok, but are we sure that A and B are authenticated to each other?

For example, if E has an old-K_{AB} and $E_{KBS}(\text{old} - K_{AB}, A)$ then E can replace (3) and (5) and impersonate A to B.

Kerberos is using timestamps.

Analysis is complicated.

People have designed a special logical system to analyze such protocols (BAN logic - Burrows, Abadi, Needham).