Cryptographic protocols

Valtteri Niemi

(text also from Timo Karvi and Ari Renvall)

Department of Computer Science
University of Helsinki, Finland

Fall 2018
Building a Key Establishment Protocol

- We discuss an attempt to design a good security protocol based on cryptography.
  The scenario is as follows:

- We assume that there is a set of users, any two of whom may wish to establish a new key for use in securing their subsequent communications through cryptography. Such a key is known as a session key.

- In order to achieve their aim the users interact with an entity called the server. All users trust the server to execute the protocol faithfully and not to engage in any other activity against them. Furthermore, the server is trusted to generate the new key and to do so in such a way that it is sufficiently random.

- Consider two users, $A$ and $B$, and the trusted server $S$. The aim of the protocol is to establish a new secret key $K_{AB}$ between $A$ and $B$. The role of $S$ is to generate the key and transport it to $A$ and $B$. 
Our first attempt is the following na"ıve protocol with three messages:

1. \( A \rightarrow S: \ A, B \)
2. \( S \rightarrow A: \ K_{AB} \)
3. \( A \rightarrow B: \ K_{AB}, A \)

Usually we make the following

Security assumption 1: The adversary is able to eavesdrop on all messages sent in a cryptographic protocol.

With this assumption, we see that the protocol is vulnerable, because the adversary can simply take the secret key \( K_{AB} \). To overcome this we assume that \( A \) and \( S \) as well as \( B \) and \( S \) already share a secret key.
Second Attempt

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: \{K_{AB}\}_{K_{AS}}, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B: \{K_{AB}\}_{K_{BS}}, A$

This protocol is as insecure in an open environment as our first attempt, but for a completely different reason.

**Security Assumption 2:**
The adversary is able to modify all messages sent in a cryptographic protocol. In addition the adversary can re-route any message to any other principal. This includes the ability to generate and insert completely new messages.

By applying the second security assumption it is possible to break the protocol without breaking the cipher: The attack proceeds as follows:
1. \( A \rightarrow S: A, B \)
2. \( S \rightarrow A: \{K_{AB}\}_{K_{AS}}, \{K_{AB}\}_{K_{BS}} \)
3. \( A \rightarrow C: \{K_{AB}\}_{K_{BS}}, A \)
4. \( C \rightarrow B: \{K_{AB}\}_{K_{BS}}, D \)

The adversary \( C \) simply intercepts the message from \( A \) to \( B \) and substitutes \( D \)'s identity for \( A \)'s. The consequence is that \( B \) believes he is sharing the key with \( D \) whereas he is in fact sharing it with \( A \). Maybe \( B \) will give \( D \)'s confidential information to \( A \).
Security Assumption 3:
The adversary may be a legitimate protocol participant (an insider), or an external party (an outsider), or a combination of both.

This assumption leads to an alternative attack against the second protocol:

1. $A \rightarrow C$: $A, B$
2. $C \rightarrow S$: $A, C$
3. $S \rightarrow C$: $\{K_{AC}\}K_{AS}, \{K_{AC}\}K_{CS}$
4. $C \rightarrow A$: $\{K_{AC}\}K_{AS}, \{K_{AC}\}K_{CS}$
5. $A \rightarrow C$: $\{K_{AC}\}K_{CS}, A$

Now $A$ thinks he is communicating with $B$, but in reality he communicates with $C$ who is able to read all the messages sent by $A$. 
The previous attacks show that we should add the identities of the participants into the messages in a secure way. This leads to the following attempt:

1. $A \rightarrow S$: $A, B$
2. $S \rightarrow A$: $\{K_{AB}, B\}_{K_{AS}}, \{K_{AB}, A\}_{K_{BS}}$
3. $A \rightarrow B$: $\{K_{AB}, A\}_{K_{BS}}$

However, even this version is not completely satisfactory.

**Security Assumption 4:**

*An adversary is able to obtain the value of the session key used in any sufficiently old run of the protocol.*
The attack based on this assumption as follows:

1. $A \rightarrow C$: $A, B$
2. $C \rightarrow A$: $\{K'_{AB}, B\}_{K_{AS}}, \{K'_{AB}, A\}_{K_{BS}}$
3. $A \rightarrow B$: $\{K'_{AB}, A\}_{K_{BS}}$

This time $C$ intercepts the message from $A$ to $S$. The key $K'_{AB}$ is an old session key used by $A$ and $B$ in a previous session. Because $K'_{AB}$ is old, $C$ has maybe succeeded to break it. Even if $K'_{AB}$ has not been broken, $C$ could replay old messages in this new session, causing problems.
Fourth Attempt

In order to prevent replays, we must add timestamps or nonces into the protocol. A nonce is a random value generated by one party and returned to that party to show that a message is newly generated.

The fourth version uses nonces:

1. $A \rightarrow S$: $A, B, N_A$
2. $S \rightarrow A$: $\{K_{AB}, B, N_A, \{K_{AB}, A\}K_{BS}\}K_{AS}$
3. $A \rightarrow B$: $\{K_{AB}, A\}K_{BS}$
4. $B \rightarrow A$: $\{N_B\}K_{AB}$
5. $A \rightarrow B$: $\{N_B - 1\}K_{AB}$

The above protocol is the famous protocol of Needham and Schroeder, from 1978. Unfortunately, it still has a flaw. If an attacker knows old session keys, he can use such in the last three messages. Then $B$ thinks he is communicating with $A$ using a new session key while actually he is communicating with the attacker using an old session key.
By adding a third nonce we achieve finally a secure protocol:

1. $A \rightarrow B$: $A$
2. $B \rightarrow A$: $\{A, N'_B\}_{K_{BS}}$
3. $A \rightarrow S$: $A, B, N_A, \{A, N'_B\}_{K_{BS}}$
4. $S \rightarrow A$: $\{K_{AB}, B, N_A, \{K_{AB}, A, N'_B\}_{K_{BS}}\}_{K_{AS}}$
5. $A \rightarrow B$: $\{K_{AB}, A, N'_B\}_{K_{BS}}$
6. $B \rightarrow A$: $\{N_B\}_{K_{AB}}$
7. $A \rightarrow B$: $\{N_B - 1\}_{K_{AB}}$
### Concepts Related to Key Establishment Protocols

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A key transport protocol</strong> is a key establishment protocol in which one of the principals generates the key and this key is then transferred to all protocol users.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A key agreement protocol</strong> is a key establishment protocol in which the session key is a function of inputs by all protocol users.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A key establishment protocol provides <strong>forward secrecy</strong> if compromise of the long-term keys of a set of principals does not compromise the session keys established in previous protocols runs involving those principals.</td>
</tr>
</tbody>
</table>
Types of Attacks

- Our list of attacks is not exhaustive; we just list typical ways of attacking security protocols.
- Different protocols have different objectives. For example, some protocols may have no need confidentiality, being concerned solely with real-time authentication.
- Similarly, some protocols may use only light measures against adversaries, because efficiency requirements prevent heavy cryptography or too many round-trips.
- Moreover, one should always clearly define the goals a security protocol is aiming to satisfy. Without such definitions, security proof and analyses are difficult to carry out.
Eavesdropping is perhaps the most basic attack on a protocol.

It is obvious that encryption must be used to protect confidential information such as session keys. In certain protocols there may be other information that also needs to remain confidential.

As an example, protocols for key establishment in mobile communications usually demand that the identity of the mobile station remains confidential.

Eavesdropping is sometimes distinguished as being a passive attack. Our other attacks require the adversary to be active.
- If a protocol message field is not redundant then modification of it is a potential way to make an attack.
- Use of cryptographic integrity mechanisms is therefore helpful in protocols for authentication and key establishment.
- Many attacks do not alter any known message fields at all, but split and re-assemble fields from different messages.
- This implies the integrity measures should cover all parts of the message that must be kept together; encryption is not enough.
Replay attacks happen when the adversary interferes with a protocol run by inserting a message, or part of a message, that has been sent previously in a valid protocol run.

This is often used in combination with other attack elements.

Replay attacks are usually avoided by using enumeration of packets, nonces or timestamps.

Note that replay as such is an essential tool in making communication reliable.
Reflection could be seen as a special case of replay.

A typical example is a scenario where two principals engage in a shared key protocol and one simply returns a challenge that is intended for itself.

This attack is more effective if parallel runs of the same protocol are allowed but this is indeed often the case. That is why we pose

**Security Assumption 5:**

*The adversary may start any number of parallel protocol runs between any principals and these principals may take same or different roles in different runs.*
Suppose $A$ and $B$ already share a secret key $K$ and choose respective nonces $N_A$ and $N_B$ for use in the protocol. The protocol is intended to mutually authenticate both parties by demonstrating knowledge of $K$.

1. $A \rightarrow B$: \{ $N_A$ \}_K
2. $B \rightarrow A$: \{ $N_B$ \}_K, N_A
3. $A \rightarrow B$: $N_B$

On receipt of message 2, $A$ deduces that it must have been sent by $B$ since only $B$ has $K$. 
However, if $A$ is willing to engage in parallel protocol runs then there is another possibility, namely that message 2 was originally formed by $A$. An adversary $C$ can masquerade as $B$ and successfully complete two runs of the protocol:

1. $A \rightarrow C_B$: $\{N_A\}_K$
1'. $C_B \rightarrow A$: $\{N_A\}_K$
2. $A \rightarrow C_B$: $\{N'_A\}_K, N_A$
2'. $C_B \rightarrow A$: $\{N'_A\}_K, N_A$
3. $A \rightarrow C_B$: $N'_A$
3'. $C_B \rightarrow A$: $N'_A$
Certificate Manipulation

- Principals, who make use of a certificate are trusting that a certificate authority has correctly identified the owner of the public key at the time that the certificate was issued.

- However, it is not necessarily expected that the authority is provided with evidence that the corresponding private key is actually held by the principal claiming ownership of the key pair. This leads to potential attacks in which the adversary gains a certificate for a public key, even though it does not know the corresponding private key.

- Consider a key agreement protocol of Matsumoto et al.: Principals $A$ and $B$ possess public keys $g^a$ and $g^b$, respectively, and corresponding private keys $a$ and $b$.

- Here $g$ generates a suitable group in which the discrete logarithm problem is hard.
Each public key is certified and so $A$ and $B$ possess certificates $\text{Cert}(A)$ and $\text{Cert}(B)$ which contain copies of their public keys ($g^a$ and $g^b$).

1. $A \rightarrow B$: $g^x$, $\text{Cert}(A)$
2. $B \rightarrow A$: $g^y$, $\text{Cert}(B)$

The shared key is $K_{AB} = g^{ay+bx}$, calculated by $A$ as $(g^y)^a(g^b)^x$ and by $B$ as $(g^a)^y(g^x)^b$. 
The adversary $C$ engineers an attack by choosing a random value $c$, claiming that $g^{ac}$ is its public key, and obtaining a certificate for this public key.

Notice that $C$ cannot obtain the corresponding private key $ac$.

$C$ then masquerades as $B$, and completes two runs of the protocol, one with $A$ and one with $B$, as shown in the attack below.

1. $A \rightarrow C_B$: $g^x$, $Cert(A)$
1'. $C \rightarrow B$: $g^x$, $Cert(C)$
2'. $B \rightarrow C$: $g^y$, $Cert(B)$
2. $C_B \rightarrow A$: $g^{yc}$, $Cert(B)$
After the attacking run is complete, $A$ will calculate the key 
\[ K_{AB} = (g^{yc})^a(g^x)^b = g^{acy+bx} \] 
and $B$ will calculate the key 
\[ K_{CB} = (g^{ac})^y(g^x)^b = g^{acy+bx}. \]

Thus $A$ and $B$ have found the same key, but $A$ believes that this key 
is known only to $A$ and $B$ while $B$ believes it is known only to $C$ and $B$. This misunderstanding can lead to problems in the subsequent use of the session key.

Attacks of this sort can be avoided by demanding that every principal demonstrates knowledge of the private key before a certificate is issued for any public key. For instance, the private key owner has to sign a specific message or challenge.
Adadi and Needham have proposed a set of principles intended to act as 'rules of thumb' for protocol designers. They were derived from observation of the most common errors that have been found in published protocols. Of course, new kind of errors or attacks will be found, but the following list helps at least to avoid old mistakes.

1. Every message should say what it means: the interpretation of the message should depend only on its content.

2. The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.

3. If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal’s name explicitly in the message.

4. Be clear about why encryption is being done.
5. When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message.

6. Be clear about what properties you are assuming about nonces.

7. If a predictable quantity (e.g. a counter) is to be effective as a freshness guarantee, it should be protected so that an intruder cannot simulate a challenge and later replay a response.

8. If timestamps are used as freshness guarantees, then the difference between local clocks at various machines must be much less than the allowable age of a message.

9. A key may have been used recently, for example to encrypt a nonce, and yet be old and possibly compromised.
10. It should be possible to deduce which protocol, and which run of that protocol, a message belongs to, and to know its number in the protocol.

11. The trust relations in a protocol should be explicit and there should be good reasons for the necessity of these relations.