CS 261 Computer Security
Crypto Protocols - Design & Analysis

Lecture by Prof. David Wagner
Scribe: Thomas Kho

November 20, 2007

1 Certificates vs. Public Keys

In public key crypto, you need to know principals' public keys to make sure that you're not talking to an impostor. The goal of a public key infrastructure (PKI) is to ensure that the name-to-public key binding is secure. For example, one way you might achieve this is if the telephone company publishes the name-public key associations in the phone book. This implies trust in the telephone company.

Certificates are of the form

$$[DavidWagner, 0xDEF8...]_{CA}$$

where 0xDEF8... is the public key of David Wagner and the message (certificate) is signed by a certificate authority (CA).

1.1 Kerberos vs. PKI

We can contrast this to Kerberos: instead of a placing trust in a KDC, we instead trust a certificate authority.

In Kerberos, revocation is simple–you delete an account and all access will be revoked once any outstanding keys expire. Key revocation is not so simple with certificates. In a PKI, revocation can be done by publishing a certificate revocation list of all certificates that have been revoked. Another way of revocation in a PKI is an online system like OSCP where one queries a CA to see if a key has been revoked. Yet another alternative is to have expiration dates on certificates.

One advantage of a PKI over Kerberos is that PKI allows multiple co-existing certificate authorities and hierarchical chains of trust. For example, one might have the arrangement:

```
       CA
     /    \
   UCB   Stanford
  /   \    /   \
DAW
```

The above is a certificate chain in which the CA asserts UCB's identity and UCB asserts user DAW's identity.

An unstrutured form of the above is the Web of Trust approach where peers sign each others' keys to extend trust.

PKI allows for a public key to be signed by multiple CAs, and someone who receives a certificate signed by multiple CAs can check if he trusts any of the CAs that have signed the certificate.

2 Discussion of paper

Most of class was spent looking at proposed protocols and noting their flaws, and the discussion follows the protocols examined in the paper. Most of this content can be found in the slides accompanying the lecture.

2.1 Notation

1. $A \rightarrow B : X$ means that the protocol designer intended for the message X to be sent by A to B, with no assurances to secrecy or reliability of the communications channel.

2.2 Warmup

We're given the following protocol:

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : [N_B]_{K_A^{-1}}$
4. $A \rightarrow B : \{message\}_{K_B}$
5. $A \rightarrow B : \{message'\}_{K_B}$

The flaw in this protocol is that there is no binding of the message sent in (3) to messages (4) and (5). There's no authentication in either messages (4) or (5) that they came from A.

2.3 Denning-Sacco #1, key exchange

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : cert_A, cert_B$
3. $A \rightarrow B : cert_A, cert_B, \{[k_{AB}, T_A]_{K_A^{-1}}\}_{K_B}$

The flaw in this protocol is that B can re-encrypt message (3) and send it to C, and C will think that it was sent to C from A.

The **moral** is to be explicit and to bind names with messages. Signatures serve to bind together parts of a message's contents so that one cannot change any field of the message.

2.4 SSL version 1, key exchange with mutual authentication

There was a flaw in version 1 of the SSL protocol:

1. $A \rightarrow B : \{k_{AB}\}_{K_B}$
2. $B \rightarrow A : \{N_B\}_{k_{AB}}$
3. $A \rightarrow B : \{cert_A, [N_B]_{K_A^{-1}}\}_{k_{AB}}$

A hint to finding the flaw in this protocol (SSL version 1 was never submitted for standards) is that A will sign anything with her public key ($[N_B]_{K_A^{-1}}$ in message (3)).

The obvious flaw given the hint is that Alice becomes a signing oracle and will sign arbitrary messages. This leads to the following attack: an impostor B can connect to C and pretend to be A.

1'. $B \rightarrow C : \{k_{BC}\}_{K_C}$
2'. $C \rightarrow A : \{N_C\}_{k_{BC}}$

1. $A \rightarrow B : \{k_{AB}\}_{K_B}$
2. $B \rightarrow A : \{N_C\}_{k_{AB}}$
3. $A \rightarrow B : \{cert_A, [N_C]_{K_A^{-1}}\}_{k_{AB}}$

3'. $B \rightarrow C : \{cert_A, [N_C]_{K_A^{-1}}\}_{k_{AB}}$

B initiates a connection with C, which authenticates via a signed nonce N_C. B turns around and challenges A with this same nonce N_C in a different SSL session (where B is the server) and has A sign N_C. B can then extract the signed N_C from (3) and use it in his session with C (3') to gain A's credentials.

2.5 "How could I have found this?

To discover something like this SSL flaw, it might be instructure to go through the goals of the protocol in your head. For example, the main goals might be for A and B to know they're talking with each other and not with an impostor, and that the key remains a secret. With a set of listed goals, it's easier to see if any are violated.

The fix to the protocol is to introduce more information. The message $[N_B]_{K_A^{-1}}$ should be replaced with $[A, B, N_A, N_B]_{K_A^{-1}}$.

The **moral** here is that you shouldn't let yourself be used as a signing oracle by naming principals and adding randomness to anything you sign.

2.6 GSM challenge-response

GSM is the communications protocol used by some cell networks. A handset A and a base station B communicate with a long-lived symmetric key K_{AB}:

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : A, [N_B]_{K_{AB}^{-1}}, \{data\}_k$

where $k = f(K_{AB}, N_B)$ is the voice privacy key.

One weakness in this protocol is that the base station never authenticates itself to the handset. While a fake base station could not decode the data, it could mount a denial of service to the handset.

2.7 X.509 Standard #1

The protocol is for sending a signed, encrypted message from A to B as follows:

1. $A \rightarrow B : A, [T_A, B, \{message\}_{K_B}]_{K_A^{-1}}$

The subtle issue in this protocol is that there is no indication that A knew the contents of the message. For example, take a system where you prove your identity by sending a password. Alice proves she has access to a system by sending a password:

1. $A \rightarrow B : A, [T_A, B, \{password\}_{K_B}]_{K_A^{-1}}$

An attacker could extract $\{password\}_{K_B}$ and sends:

1'. $M \rightarrow B : M, [T_M, B, \{password\}_{K_B}]_{K_M^{-1}}$

Another example where this can introduce issues is in secure auctions, where an eavesdropper can duplicate another bidder's bid without knowing the bid amount.

The takeaway from this example is the role that authentication can play in systems:

1. Authentication as endorsement (i.e. taking responsibility)
2. Authentication as a way of claiming credit (i.e. stating authorship)

Signatures serve purpose (1) but not (2), and the **moral** is to sign before encrypting.

2.8 An aside on timestamping services

Timestamping services serve to provide unforgeable signatures that bind a document to a time. One proposal is to use hash chains, which have the following form:

$$\begin{tikzcd}
 & M_1 \arrow[d] & M_2 \arrow[d] & M_3 \arrow[d] & M_4 \arrow[d] & \text{hash} \\
 & H_1 \arrow[d] & H_2 \arrow[d] & H_3 \arrow[d] & H_4 \arrow[d] & \\
0 \arrow[r] & X_1 \arrow[r] & X_2 \arrow[r] & X_3 \arrow[r] & X_4 &
\end{tikzcd}$$

In this system, the X_i's and H_i's are published. It's not possible back-date messages because the published values commit you to the contents of each message.

2.9 TMN

A and B establish a shared key k_B with the help of a fast server S.

1. $A \rightarrow S : \{k_A\}_{K_S}$
2. $B \rightarrow S : \{k_B\}_{K_S}$
3. $S \rightarrow A : k_A \oplus k_B$

The assumptions in this protocol are that messages (1) and (2) are signed by A and B, respectively, and that the server response (3) says that part of the shared key is from B.

A recovers k_B as $k_A \oplus (k_A \oplus k_B)$

The flaw in this protocol is that it provides an oracle for an attacker. Given $\{k_B\}_{K_S}$, nodes M and M' can conspire to recover k_B:

1'. $M \rightarrow S : \{k_B\}_{K_S}$
2'. $M' \rightarrow S : \{k_{M'}\}_{K_S}$
3'. $S \rightarrow M : k_B \oplus k_{M'}$

M receives $k_B \oplus k_{M'}$, but can collude with M' which generated $k_{M'}$ to determine k_B.

2.10 Goss railway protocol

In this protocol, A and B establish a shared key $k_{AB} = r_A \oplus r_B$:

1. $A \rightarrow B : A, \{r_A\}_{K_B}$
2. $B \rightarrow A : B, \{r_B\}_{K_A}$

The goal for establishing session keys is that the key used for one session cannot be used to learn anything about previous or future sessions.

This protocol is susceptible to a triangle attack in which attacker M can recover r_A from $\{r_A\}_{K_B}$:

1. $M \rightarrow B : M, \{r_A\}_{K_B}$
2. $B \rightarrow M : B, \{r'_B\}_{K_M}$

If M learns k_{BM} in the course of further interaction with B (e.g. the key somehow leaks out or is found out through malicious means), M can compute $r_A = K_{BM} \oplus r'_B$. Doing the same with A, M can learn r_B and can then determine the key k_{AB}.

Given that M has a mechanism to learn k_{BM}, M can use both A and B as a decryption oracle.

3 Summary

There were a number of principles to take away:

- You should write out your protocols in plain English to aide in evaluating correctness.

- Your protocols should be very explicit and should including everything (including versioning your protocol).
- Not discussed in the paper is the idea that each message should contain a hash of all messages before the message, so that the context (session history) of the message is very obvious and that it is also obvious if there is a man-in-the-middle attack.
- Don't reuse keys. This applies both across roles (signing, decrypting, ...) and time (session keys).

Lastly, protocol errors are very subtle. It is important to look everywhere (take a system view) for potential security flaws. An example of this is that signed email commonly signs just the contents of an e-mail message and none of its headers.