Cryptographic protocols: design and analysis

David Wagner
University of California, Berkeley
Notation

$A, B, C, S$: names of legitimate parties. (Short for: Alice, Bob, client, server.)

$M$: name of a malicious attacker. (Short for: Mallet.)
1. \( A \rightarrow B : x \)

The above means:

1. Protocol designer intended the message \( x \) to be sent by party \( A \) to party \( B \).

2. This message was intended to be sent first in a series of similar messages.
Caveats

1. $A \rightarrow B : x$

Do note:

1. $B$ only receives the message $x$, not who it came from.
   (Thus, messages should include the sender’s name if the recipient needs to know it.)

2. There is no guarantee that $A$, the network, or the adversary will behave as intended.
   (Thus, messages might be intercepted, modified, re-ordered, etc.)
More Notation

$k$ is a key; $k^{-1}$ is its inverse.

For symmetric cryptosystems, $k = k^{-1}$; for public-key cryptosystems, $k$ is the public key and $k^{-1}$ the corresponding private key.
\{x\}_k \text{ means } x \text{ encrypted under } k.

Warning: This is implicitly assumed to provide both secrecy and integrity, and is not the standard notation. For instance, \{x, y\}_k \text{ securely binds } x \text{ to } y.

(Exercise: How do you implement \{x\}_k?)

\[x\]_{k^{-1}} \text{ means } x \text{ signed under } k^{-1}.

Most authors conventionally use \{x\}_{k^{-1}} for signatures, but I don’t like the standard notation. (Exercise: Why not?)
$T_A$ is a timestamp chosen by $A$.

$N_A$ is an unpredictable random nonce (a “challenge”) chosen...
Who’s awake?

What does the following notation mean?

1. \( A \rightarrow B : \{ A, [k_{AB}, A, B]_{K_A^{-1}} \}_B \)
2. \( B \rightarrow A : \{ \text{message} \}_{k_{AB}} \)
Warmup

Establishing a secure channel with a challenge-response protocol:

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

... 

Can you spot the flaw?
Key exchange between $A, B$, with the aid of an online certification server $S$.

1. $A \rightarrow S : \quad A, B$
2. $S \rightarrow A : \quad \text{cert}_A, \text{cert}_B$
3. $A \rightarrow B : \quad \text{cert}_A, \text{cert}_B, \{[k_{AB}, T_A]_{K_A^{-1}}\}_{K_B}$

Can you spot the flaw?
Look closely:

3. \[ A \rightarrow B : \text{cert}_A, \text{cert}_B, \{ [k_{AB}, T_A]^{K_A^{-1}} \}^K_B \]

The key \( k_{AB} \) isn’t bound to the names of the endpoints \( A, B \).

Therefore, \( B \) can extract the quantity \( [k_{AB}, T_A]^{K_A^{-1}} \) and use it in a new connection to \( C \), like this:

3'. \[ B \rightarrow C : \text{cert}_A, \text{cert}_C, \{ [k_{AB}, T_A]^{K_A^{-1}} \}^K_C \]

As a result, \( C \) mistakenly concludes he is speaking with \( A \).
Moral: Be explicit. Bind all names, and all other relevant context, to every message.

Exercise: Why do so many protocols fail this way?

Credits: Abadi and Needham.
Early SSL

Key exchange with mutual authentication:

1. \( A \rightarrow B : \{k_{AB}\}^{K_B} \)
2. \( B \rightarrow A : \{N_B\}^{k_{AB}} \)
3. \( A \rightarrow B : \{\text{cert}_A, [N_B]^{K_A^{-1}}\}^{k_{AB}} \)

Can you spot the flaw?
Breaking early SSL

Look closely:

1. \( A \rightarrow B : \{k_{AB}\}K_B \)
2. \( B \rightarrow A : \{N_B\}k_{AB} \)
3. \( A \rightarrow B : \{\text{cert}_A, [N_B]_{K_A^{-1}}\}k_{AB} \)

Alice will sign \textit{anything} with her private key.
The attack on early SSL

$B$ can open a connection to $C$ and pretend to be $A$, as follows:

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

When $C$ challenges $B$ with nonce $N_C$, Bob sends $N_B = N_C$ and uses her as an oracle.

1. $A \rightarrow B : \{k_{AB}\}_{KB}$
2. $B \rightarrow A : \{N_C\}_{k_{AB}}$
3. $A \rightarrow B : \{\text{cert}_A, [N_C]_{K_A^{-1}}\}_{k_{AB}}$

$A$ will sign anything, so $B$ extracts $[N_C]_{K_A^{-1}}$ and he’s in:

3. $B \rightarrow C : \{\text{cert}_A, [N_C]_{K_A^{-1}}\}_{k_{AB}}$
Fixing early SSL

Fix: replace \( [N_B]_{K_A}^{-1} \) with \([A, B, N_A, N_B]_{K_A}^{-1}\).

1. \( A \to B : \{k_{AB}\}_{K_B} \)
2. \( B \to A : \{N_B\}_{k_{AB}} \)
3. \( A \to B : \{\text{cert}_A, [A, B, N_A, N_B]_{K_A}^{-1}\}_{k_{AB}} \)

Moral: Don’t let yourself be used as a signing oracle. Add randomness—and bind names—before signing.

Credits: Abadi and Needham.
A is cellphone handset, B is a base station.

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

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

Can you spot the weakness?
X.509 standard #1

Sending a signed, encrypted message to $B$:

1. $A \rightarrow B : A, [T_A, B, \{\text{message}\}^B_K]^A_K^{-1}$

Can you spot the flaw?
Breaking X.509 standard #1

Look again:

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

There's no reason to believe the sender was ever aware of the contents of the message.
Example: Proving yourself by sending a password.

Attacker $M$ intercepts Alice’s encrypted password:

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

Then $M$ extracts $\{\text{password}\}_{K_B}$, and sends

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

Now $M$ is in, without needing to know the password.
Example: Secure auctions.

The same attack provides an easy way for $M$ to send in a copy of $A$'s bid under his own name, without needing to know what $A$'s bid was.
Lessons

An important difference between

- Authentication as *endorsement* (i.e., taking responsibility)
- Authentication as a way of *claiming credit*.

Encrypting before signing provides a secure way of assigning responsibility, but an insecure way to establishing credit.

**Moral:** sign before encrypting.

Credits: Abadi
Pop quiz. Watch carefully.

A, B establish a shared key $k_B$ using the help of a fast server $S$:

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

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

Can you spot the flaw?
Breaking TMN

Let’s play spot the oracle!

The attack: Given \( \{k_B\}_K S \), \( M, M' \) can conspire to recover \( k_B \):

1’. \( M \rightarrow S : \{k_B\}_K S \)
2’. \( M' \rightarrow S : \{k_{M'}\}_K S \)
3’. \( S \rightarrow M : k_B \oplus k_{M'} \)

Now \( M, M' \) can recover \( k_B \) from \( \{k_B\}_K S \).

Credits: Simmons.
A and B establish an authenticated shared key $k_{AB} = r_A \oplus ?$

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

Do you see the subtle weakness?
Triangle attacks on Goss

If session keys sometimes leak, the system breaks.

\( M \) can recover \( r_A \) from \( \{r_A\}_{KB} \) by opening a session to \( B \) and replaying \( A \)'s encrypted contribution to the key:

1. \( M \rightarrow B : C, \{r_A\}_{KB} \)
2. \( B \rightarrow M : B, \{r'_B\}_{KM} \)

Now if \( M \) can learn \( k_{BM} \) somehow, he can compute \( r_A = k_{BM}r_B \).

Basically, if \( B \) lets session keys leak, \( M \) can use him as as a decryption oracle to obtain \( r_A \) from \( \{r_A\}_{KB} \).

Play the same games with \( A \) to recover \( r_B \) from \( \{r_B\}_{KA} \); you then learn \( k_{AB} \).

Credits: Burmester.
Explicitness is powerful (and cheap).

The mathematical notation

1. $B \rightarrow A : \ N_B$
2. $A \rightarrow B : \ \{N_B, k_{A,B}\}_{K_A}$

might be implemented in practice as

1. $B \rightarrow A : \ \text{"Msg 1 from } B \text{ to } A \text{ of GSM protocol v1.0 is a challenge } N_B\text{."}$
2. $A \rightarrow B : \ \{\text{"Msg 2 from } A \text{ to } B \text{ of GSM protocol v1.0 is a response to the challenge } N_B; \text{ and } A \text{ asserts that the session key } k_{A,B} \text{ is fresh and good for communication between } A \text{ and } B \text{ on the session where } N_B \text{ was seen."}\} _{K_A}$

(Can you see why each of the elements above are there?)
Implementing protocols

Any value received as cleartext should be treated as untrustworthy: you may use it as a **hint** for performance, but don’t depend on it for security.

**Minimize state;** each message should be self-explanatory.
Implementing protocols

Don’t reuse keys: for instance, signing keys and decryption keys should not be equated. Use a separate session key for each direction.

Hash everything. Each message should include the (signed?) hash of all previous messages in the interaction. This makes cut-and-paste attacks harder.

Measure twice, cut once.