Introduction to Computer Security

Cryptographic Protocols

Dr. Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: October 25, 2019 at 14:41

Thought Experiment

Imagine you have a friend who lives in some foreign country where the secret police spy on everyone and everything, and you want to send a valuable object to this friend. You have a strongbox big enough to hold the object. The box has a lock ring that can accommodate several locks. But your friend does not have the key to any lock that you have.

You can't send a key in the mail because the secret police will intercept and copy it. You can't leave the box unlocked, because the object is too valuable. Ideally, you'd lock the object in the box and mail it to your friend, so that he can open it, but the secret police can't. How could you do it?

One Possible Answer

You might take the following sequence of steps:

1. You put your treasure into the box, attach your lock to the clasp, and mail the box to your friend.
2. He adds his own lock, for which he has the key, and mails the box back to you.
3. You remove your lock and mail the box back to him. He now removes his lock and opens the box.

What's This Got to do with Computing?

The procedure just described could be regarded as a *protocol* – a structured dialog intended to accomplish some communication-related goal.

What goal: To send some content secretly in the context of a hostile or untrustworthy environment, when the two parties don't already share a secret/key

In fact, you could implement the "same" protocol to send a message confidentially across the Internet. Here,

- the *valuable thing* is the contents of a secret message;
- the *locks* are applications of a cryptographic algorithm (cipher) with appropriate *cryptographic keys.*

But for this to work in the computing world there's a particular feature that the ciphers have to satisfy. Can you see what it is?

What is the Property?

Imagine that instead of putting on another lock, your friend puts your lockbox inside another locked box. Our protocol wouldn't work because you couldn't reach inside his box to take off your lock in step 3.

You have to be able to "reach inside" his encryption to undo yours. One way this would be true is if the ciphers *commute*.

$$Cipher_1(k_1, Cipher_2(k_2, msg)) = Cipher_2(k_2, Cipher_1(k1, msg))$$

Most encryption algorithms don't have this property. But one that does is: exclusive or (XOR) your message with a randomly generated string (key) of the same length.

So Here's the Protocol

Let K_a be a random string generated by A, and K_b be a random string generated by B.

1. $A \rightarrow B : M \oplus K_a$
2. $B \rightarrow A : (M \oplus K_a) \oplus K_b$
3. $A \rightarrow B : ((M \oplus K_a) \oplus K_b) \oplus K_a)$

In step 3, the two applications of K_a "cancel out," leaving $(M \oplus K_b)$, which B can easily decrypt with his own key K_b.

Whoops!

This is effectively using the one-time pad, so should be very strong. Right?

Even though the one-time pad is a theoretically unbreakable cipher, there's a reason it's called "one-time." Our protocol is fundamentally flawed. Can you see why?

Whoops!

This is effectively using the one-time pad, so should be very strong. Right?

Even though the one-time pad is a theoretically unbreakable cipher, there's a reason it's called "one-time." Our protocol is fundamentally flawed. Can you see why?

An evesdropper who records the three messages can XOR combinations of them to extract any of M, K_a, and K_b. Verify this for yourself.

Cryptographic Protocols

Sometimes a security-related interaction can be handled in a single message from one subject to another. Sometimes however, it is necessary to conduct a structured dialogue. This is called a *protocol*. A protocol involving cryptographic primitives is called a *cryptographic protocol*.

A cryptographic protocol may involve two, three, or more players or *principals*. Some principals may have specific roles and certain assumed properties. For example, some protocols involve a trusted *authentication server* or *certification authority*.

Definition

Here is one way to define *cryptographic protocol*.

Definition: A *protocol* is a structured dialogue among two or more parties in a distributed context controlling the syntax, semantics, and synchronization of communication, and designed to accomplish a communication-related function.

Definition: A *cryptographic protocol* is a protocol using cryptographic mechanisms to accomplish some security-related function.

Possible Goals

Among the goals of a cryptographic protocol may be one or more of the following:

- Key agreement or establishment
- Entity authentication
- Symmetric encryption and message authentication material construction
- Secured application-level data transport
- Non-repudiation methods

A Protocol Example

Consider the following simple protocol:

1. $A \rightarrow B : \{\{K\}_{K_a^{-1}}\}_{K_b}$
2. $B \rightarrow A : \{\{K\}_{K_b^{-1}}\}_{K_a}$

Informal goal: A shares with B a secret key K, and each party is authenticated to the other.

What are the assumptions? Precisely what are the goals? Are they satisfied? How can you be sure?

A Protocol Example

Consider the following simple protocol:

1. $A \rightarrow B : \{\{K\}_{K_a^{-1}}\}_{K_b}$
2. $B \rightarrow A : \{\{K\}_{K_b^{-1}}\}_{K_a}$

Informal goal: A shares with B a secret key K, and each party is authenticated to the other.

What are the assumptions? Precisely what are the goals? Are they satisfied? How can you be sure?

This protocol is fatally flawed. Can you see how?

Crypto. Protocols (Cont.)

The typical assumption of cryptographic protocols is that several principals in a distributed setting are attempting to establish a secure communication in the face of a hostile environment. The protocol must be robust and reliable in the face of a determined attacker.

A protocol involves a sequence of steps of message exchange. A *step* in the protocol is of the form:

$$A \rightarrow B : M$$

meaning that principal A sends to principal B the message M.

Because of the distributed nature of the system and the possibility of malicious actors, there is typically no guarantee that B receives the message, *or is even expecting the message.*

Taking an Abstract View

There is much detail involved in making a protocol work, particularly at the lower levels of the implementation hierarchy.

We will usually ignore issues like:

- What are the mechanisms of message transmission?
- How does a principal know that a decryption has succeeded?
- How can you reliably parse a message of multiple components?
- If a message contains the name of a principal, what is the form of that name?
- How are public keys maintained and distributed?

Those are all important issues in protocol implementation, but typically swept under the rug in abstract analysis.

Crypto. Protocols (Cont.)

Often the message M is highly structured and may contain one or more encrypted portions.

The protocol functions in a specific cryptographic context. For example, the principals might (or might not) be assumed to operate within a public key infrastructure (PKI) in which each has a private key and generally known public key. It is important to understand the implied context.

A message may contain various parts: $M_1, \ldots M_n$. We assume that these are concatenated or otherwise packaged into a single message in such a way that the receiver can recognize and extract the various components.

Crypto. Protocols (Cont.)

Descriptions of protocols often use the notation $\{M\}_k$ to denote what we have sometimes designated by $E(k, M)$, i.e., the encryption of the message M using key k. Notice that decryption and encryption are essentially identical algorithms. Hence, a decryption step would be similarly denoted.

One consequence of this is that successive layers of encryption may actually cancel others. For example,

$$\{\{M\}_{K_R^{-1}}\}_{K_R} = M.$$

and the decryption of a symmetrically encrypted message $\{M\}_K$ is actually $\{\{M\}_K\}_K = M$.

Crypto. Protocols (Cont.)

An analysis of any protocol attempts to answer the following types of questions:

- What are the goals of the protocol?
- What does the protocol actually achieve?
- In particular, does it achieve its stated objective?
- Does this protocol include unnecessary steps or messages?
- Does this protocol need more assumptions than another might?
- Does it encrypt items that could be sent in the clear?
- Is it susceptible to attack? What would an attack look like?

Attacks

Protocols have been published and in use for years before someone notes a significant vulnerability. A difficult aspect of analyzing cryptographic protocols is answering the question: What constitutes an attack?

- Are both authentication and secrecy assured?
- Is it possible to impersonate one or more of the parties?
- Is it possible to interject messages from an earlier exchange (replay attack)?
- What tools can an attacker deploy?
- *If any private key is compromised, what are the consequences?
- *If an earlier session key is compromised, what are the consequences in the current context?

Are the last two fair questions?

Attacks on Protocols

This is a partial list of attacks on protocols:

Known-key attack: attacker gains some keys used previously and then uses this info to attack the protocol and possibly determine new keys.

Replay: attacker records a communication session and replays some or all of it at a later time.

Impersonation: attacker assumes the identity of one of the legitimate parties in a network.

Man-in-the-Middle: attacker interposes himself between two parties and pretends to each to be the other.

Interleaving attack: attacker injects spurious messages into a protocol run to disrupt or subvert it.

Attackers

The designer of a protocol should assume that an attacker can access *all of the traffic* and interject his own messages into the flow. Can the attackers messages be arbitrary? Why not? What restrictions do we impose on the attacker?

The protocol should be robust in the face of such a determined and resourceful attacker.

Important Point About Protocols

Due to the distributed nature of the system, protocols are typically highly asynchronous.

A party to a protocol probably will not know anything about the current run of the protocol except the messages it has received and sent.

Consequently, except for the initiator of the protocol, other parties to the protocol *will not even know that they are participating* until they receive their first message. That message must be of a form that they can identify and respond to.

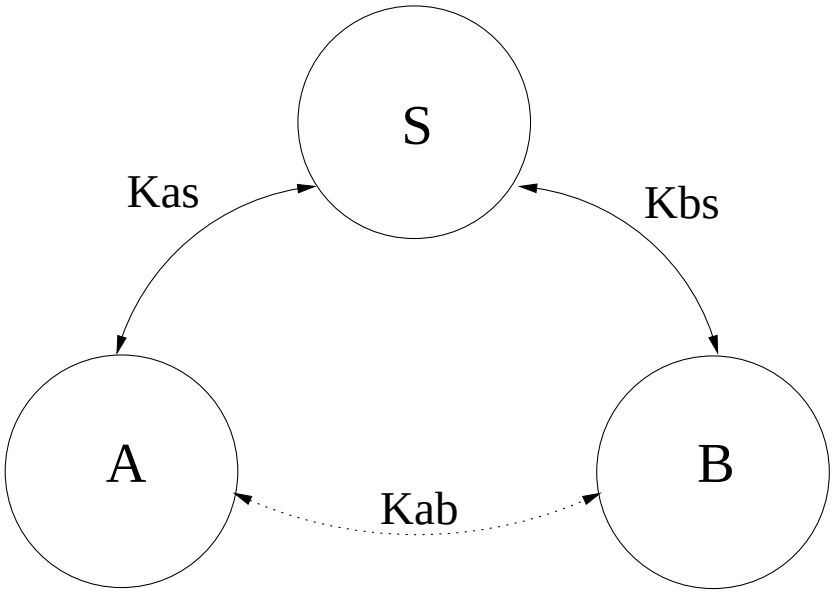
Needham-Schroeder Protocol

Many existing protocols are derived from one proposed (1978) by Needham and Schroeder, including the widely used Kerberos authentication protocol suite.

The protocol is a *shared-key authentication protocol* designed to generate and propagate a *session key*, i.e., a shared key for subsequent symmetrically encrypted communication.

Needham-Schroeder

There are three principals: A and B, two principals desiring mutual communication, and S, a trusted key server.



It is assumed that A and B have already established secure symmetric communication with S using shared keys K_{as} and K_{bs}, respectively.

Nonces and Timestamps

The protocol also uses *nonces* (short for "number used once"), randomly generated values included in messages. If a nonce is generated and sent by A in one step and returned by B in a later step, A knows that B's message is *fresh* and not a replay from an earlier exchange.

Note that a nonce *is not a timestamp*. It is designed to prevent replay attacks and show that a message is fresh. The only assumption is that it has not been used (with high probability) in any earlier interchange.

If you want to show that a message is *recent*, as opposed to fresh, use a timestamp. A timestamp can be used in a context in which messages have a limited useful lifetime. However, this introduces a range of issues regarding clock synchronization.

Needham-Schroeder (Cont.)

The protocol is as follows:

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Here N_a and N_b are nonces.

Two questions to ask of any step in any protocol:

- What is the sender trying to say in her message?
- What is the receiver entitled to believe after receiving the message?

Needham-Schroeder Step 1

The first step of the protocol is:

1. $A \rightarrow S : A, B, N_a$
   - What is the sender trying to say in her message?

Needham-Schroeder Step 1

The first step of the protocol is:

1. $A \rightarrow S : A, B, N_a$

   - What is the sender trying to say in her message?
     Hey, S! I'm A and I want to talk to B, so generate a new key for us. And by the way, here's a nonce that you can use in subsequent messages so we'll be sure that you're responding to this request.
   - What is the receiver entitled to believe after receiving the message?

Needham-Schroeder Step 1

The first step of the protocol is:

1. $A \rightarrow S : A, B, N_a$

   - What is the sender trying to say in her message?
     Hey, S! I'm A and I want to talk to B, so generate a new key for us. And by the way, here's a nonce that you can use in subsequent messages so we'll be sure that you're responding to this request.
   - What is the receiver entitled to believe after receiving the message?
     A wants to talk to B, so I need to generate a new session key and get it to them. I should use N_a in the response so that they'll know it's fresh.

Needham-Schroeder Step 2

The second step of the protocol is:

2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

Answer the two questions for this step.

Needham-Schroeder Step 2

The second step of the protocol is:

2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

Answer the two questions for this step.

Meaning of Step 2: S generates an appropriate session key K_{ab} for use by A and B and sends it to A in a message encrypted with their shared key K_{as}. Why is the other information included? What can A infer upon receipt of this message?

Needham-Schroeder Step 3

The third step of the protocol is:

3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$

Meaning of Step 3: A relays the new session key to B. What does B know after this step?

Needham-Schroeder Step 4

The fourth step of the protocol is:

4. $B \rightarrow A : \{N_b\}_{K_{ab}}$

Meaning of Step 4: B sends an acknowledgement to A. Why is this necessary? What can A infer from the receipt of this message?

Needham-Schroeder Step 5

The fifth and final step of the protocol is:

5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Meaning of Step 5?: What is the reason for this step? What do A and B now believe is true? What has been accomplished?

Needham-Schroeder

Here's the complete protocol again. Is it "good"? What does that mean?

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Attacks on N-S

Recall our earlier list of some things to ask about a protocol.

- Are both authentication and secrecy assured?
- Is it possible to impersonate one or more of the parties?
- Is it possible to interject messages from an earlier exchange (replay attack)?
- What tools can an attacker deploy?
- If any private key is compromised, what are the consequences?
- If an earlier session key is compromised, what are the consequences in the current context?

Flaws in Needham-Schroeder

Denning and Sacco pointed out that the compromise of a session key has bad consequences. An intruder can reuse an old session key and pass it off as a new one as though it were fresh. What would such an attack look like?

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Flaws in Needham-Schroeder

Claim: At the end of the protocol, A knows it is talking to B and vice versa. Actually, the holder of K_{as} knows it is talking to the holder of K_{bs}.

Problem: Message 3 is not protected by nonces. There is no way for B to know if the K_{ab} it receives is current. An intruder has unlimited time to crack an old session key and reuse it as if it were fresh. What does that attack look like?

Example: (from Mike Dahlin) a disgruntled employee runs the first few steps of the protocol multiple times, gathering up a bunch of tickets $\{K_{ab}, A\}_{K_{bs}}$ for all the different servers B in the system. After he is fired, he can still log onto all of the company's servers.

Flaws in Needham-Schroeder

Bauer, et al. pointed out that if key K_{as} were compromised, anyone could impersonate A and establish communication with any other party. This is true, even if A's key is later changed. Describe this attack.

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

These flaws persisted for almost 10 years before they were discovered.

Is it Fair?

The "attacks" discovered by Denning and Sacco and by Bauer, et al. ask what happens if a key is broken.

Is it fair to ask that question? Isn't a presumption of any cryptographic protocol that the encryption is strong?

How might you address this question if you're a protocol designer?

NS (Public Key)

Needham and Schroeder also suggested a protocol based on public key cryptography. The goal is for A and B to establish communication and exchange two independent, secret numbers.

As with their shared key protocol, a third party S is involved. Here the assumption is that public keys are not universally known, so a server is necessary to manage the keys. S acts as a certification authority and disseminates public keys.

NS Public Key (Cont.)

Here K_x refers to X's public key, and K_x^{-1} to the corresponding private key.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_b, B\}_{K_S^{-1}}$
3. $A \rightarrow B : \{N_a, A\}_{K_b}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_a, A\}_{K_S^{-1}}$
6. $B \rightarrow A : \{N_a, N_b\}_{K_a}$
7. $A \rightarrow B : \{N_b\}_{K_b}$

Here N_a and N_b are the two numbers that they wish to exchange.

What does each party know after each step?

NS Public Key

The Needham-Schroeder Public Key protocol is susceptible to a *man in the middle* attack. If an attacker C can persuade A to begin a communication with I, then C can pass on the traffic to B and convince B that he is actually talking to A. What does that attack look like?

Note also that this protocol does not make assumptions about PKI that are typically made today. That is, there is usually not a special key server in the system.

Otway-Rees

Another very important and much studied protocol is the Otway-Rees protocol. Below is one of several variants.

1. $A \rightarrow B : M, A, B, \{N_a, M, A, B\}_{K_{as}}$
2. $B \rightarrow S : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{M, N_b, A, B\}_{K_{bs}}$
3. $S \rightarrow B : M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. $B \rightarrow A : M, \{N_a, K_{ab}\}_{K_{as}}$

What is the goal of the protocol? What purpose do the parts of each message play? What do the principals believe after each step?

From Wikipedia

One problem with this protocol is that a malicious intruder can arrange for A and B to end up with different keys. Here is how: after A and B execute the first three messages, B has received the key K_{ab}. The intruder then intercepts the fourth message. S/he resends message 2, which results in S generating a new key K'_{ab}, subsequently sent to B. The intruder intercepts this message too, but sends to A the part of it that B would have sent to A. So now A has finally received the expected fourth message, but with K'_{ab} instead of K_{ab}.

Another problem is that although the server tells B that A used a nonce, B doesn't know if this was a replay of an old message.

A Flawed Protocol

The following is a simple protocol that we introduced previously. The idea is for A to communicate a session key K securely to B, and receive an acknowledgement such that A knows that B has the key.

Can you find the flaw?

1. $A \rightarrow B : \{\{K\}_{K_a^{-1}}\}K_b$
2. $B \rightarrow A : \{\{K\}_{K_b^{-1}}\}K_a$

How would you correct it?

Verification of Cryptographic Protocols

Protocols can be notoriously difficult to get correct. Flaws have been discovered in protocols published many years before. Hence, it would be nice to be able to reason formally about protocol correctness.

One major difficulty is that you'd like to ensure that no "spy" can obtain keys or other information intended to remain secret. However, it is tricky to add the spy into the protocol description and delimit what capabilities the spy has.

Verification

There are several major approaches to the verification problem:

1. *Belief logics* such as the BAN logic (Burrows, Abadi, and Needham) and the GNY logic (Gong, Needham, Yahalom) allow short abstract proofs, but may miss some important flaws.
2. *State exploration methods* model the protocol as a finite state system and conduct an exhaustive search checking that all reachable states are safe. The general purpose model checker FDR has been used in this way, and a specialized tool, the Interrogator (Millen, et al.).
3. *General-purpose theorem proving* using induction over potential traces of protocol execution. This was pioneered by Lawrence Paulson of Cambridge University using the Isabelle Theorem Prover.

Belief Logics

A belief logic is a formal system for reasoning about beliefs. Any logic consists of a set of logical operators and rules of inference.

The trick is getting from a protocol as a sequence of message exchanges to a collection of *belief statements*. It is analogous to the problem in Hoare logic of getting from programming fragments to logical implications.

You have to postulate some reasonable initial assumptions about the state of knowledge/belief of the principals.

Belief Logics: BAN

The BAN logic is a modal logic of belief. It has 10 primitive modal operators including:

- $P|\equiv X$: *(P believes X)* P is entitled to act as though X is true.
- $A \triangleleft X$: *(A sees X)* someone has sent a message to A containing X so that he can read X and repeat it.
- $A|\sim K$: *(A once said K)* at some time, A used key K.
- $A|\sim X$: *(A once said X)* at some time, A uttered a message containing X.
- $A \Longrightarrow X$: *(A has jurisdiction over X)* A is an authority on X and can be trusted on X.
- $A \overset{K}{\longleftrightarrow} B$: *(A and B share key K)* A and B can use key K to communicate. The key is unknown to anyone else.

BAN Operators (Cont.)

$\#(X)$: *(X is fresh)* meaning that X has not been sent before in any run of the protocol.

$\xrightarrow{K} B$: *(B has_public_key K)* B has a published public key K and corresponding private key K^{-1}.

$A \overset{X}{\Longleftrightarrow} B$: *(A and B share secret X)* X is a secret known only to A, B and possibly some trusted associates.

Belief Logics: Rules

There are numerous rules of inference for manipulating the protocol to generate a set of beliefs. For example,

Message meaning: If A believes(A share(K) B) and A sees $\{X\}_K$ then A believes(B said X).

$$\frac{A|\equiv (A \overset{K}{\longleftrightarrow} B), A \triangleleft \{X\}_K}{A|\equiv (B|\sim X)}$$

Belief Logics: Rules

Nonce verification: If A believes X is fresh and A believes B once said X, then A believes B believes X.

$$\frac{A|\equiv (\#(X)), A|\equiv (B|\sim X)}{A|\equiv (B|\equiv X)}$$

Jurisdiction: If A believes B has jurisdiction over X and A believes B believes X, then A believes X.

$$\frac{A|\equiv (B \Longrightarrow X), A|\equiv (B|\equiv X)}{A|\equiv X}$$

BAN Logic: Idealization

To get from protocol steps to logical inferences, we have a process called *idealization*. This attempts to turn the message sent into its intended semantics. For example, given the protocol step:

$$A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}$$

If B knows the key K_{bs}, this tells us that K_{ab} is a key to communicate with A. An idealized version is:

$$A \rightarrow B : \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}$$

One purpose of idealization is to omit parts of the message that do not contribute to the beliefs of the recipients. *In BAN all plaintext is omitted since it can be forged.*

Needham-Schroeder: Idealization

Here's the familiar Needham-Schroeder protocol.

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Needham-Schroeder is idealized as follows:

1. omitted since all components are plaintext
2. $S \rightarrow A : \{N_a, (A \overset{K_{ab}}{\longleftrightarrow} B), \#(A \overset{K_{ab}}{\longleftrightarrow} B), \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b, (A \overset{K_{ab}}{\longleftrightarrow} B)\}_{K_{ab}}$ *from B*
5. $A \rightarrow B : \{N_b, (A \overset{K_{ab}}{\longleftrightarrow} B)\}_{K_{ab}}$ *from A*

BAN Logic: Assumptions

The following initial assumptions are given for Needham-Schroeder:

$A|\equiv A \overset{K_{as}}{\longleftrightarrow} S \qquad B|\equiv B \overset{K_{bs}}{\longleftrightarrow} S \qquad S|\equiv A \overset{K_{as}}{\longleftrightarrow} S$

$S|\equiv B \overset{K_{bs}}{\longleftrightarrow} S$

$S|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$

$A|\equiv (S \Longrightarrow A \overset{K}{\longleftrightarrow} B) \qquad B|\equiv (S \Longrightarrow A \overset{K}{\longleftrightarrow} B)$

$A|\equiv (S \Longrightarrow \#(A \overset{K}{\longleftrightarrow} B))$

$A|\equiv \#(N_a) \qquad B|\equiv \#(N_b) \qquad S|\equiv \#(A \overset{K_{ab}}{\longleftrightarrow} B)$

$B|\equiv \#(A \overset{K}{\longleftrightarrow} B)$

The very last of these is pretty strong. Needham and Schroeder did not realize they were making it, and were criticized for it.

BAN Logic: Analyzing the Protocol

From step 2 of the protocol:

$$A \triangleleft \{N_a, (A \overset{K_{ab}}{\longleftrightarrow} B), \#(A \overset{K_{ab}}{\longleftrightarrow} B), \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

The *Nonce Verification Rule* says:

$$\frac{A|\equiv (\#(X)), A|\equiv (B|\sim X)}{A|\equiv (B|\equiv X)}$$

Since A believes N_a to be fresh, we get:

$$A|\equiv (S|\equiv A \overset{K_{ab}}{\longleftrightarrow} B)$$

BAN Logic: Analyzing the Protocol

The *Jurisdiction Rule* says that:

$$\frac{A|\equiv (B \Longrightarrow X), A|\equiv (B|\equiv X)}{A|\equiv X}$$

From this we obtain:

$$A|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

$$A|\equiv \#(A \overset{K_{ab}}{\longleftrightarrow} B)$$

BAN Logic: Analyzing the Protocol

Since A has also seen the part of the message encrypted under B's key, he can send it to B. B decrypts the message and obtains:

$$B|\equiv (S|\sim A \overset{K_{ab}}{\longleftrightarrow} B)$$

meaning that B believes that S once sent the key.

At this point, we need the final dubious assumption:

$$B|\equiv \#(A \overset{K}{\longleftrightarrow} B)$$

With it, we can get:

$$B|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

BAN Logic: Analyzing the Protocol

From the last two messages, we can infer the following. How?

$$A|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

$$B|\equiv A \overset{K_{ab}}{\longleftrightarrow} B$$

$$A|\equiv (B|\equiv A \overset{K_{ab}}{\longleftrightarrow} B)$$

$$B|\equiv (A|\equiv A \overset{K_{ab}}{\longleftrightarrow} B)$$

These are the point of the protocol. The proof exhibits some assumptions that were not apparent.

Problems with BAN

Idealization of the protocol is not defined unambiguously. It depends on the interpretation of the meaning of some steps.

If the step contains information about the receiving party, this is often lost in the idealization. Therefore, the idealization step isn't reversable.

BAN doesn't formalize some potential errors. E.g., "secret things need to stay secret." So you can send a key in the clear and BAN will be perfectly happy.

Using a Belief Logic

Using a logic such as BAN or GNY to analyze a protocol requires:

1. Converting the protocol into a formalized version suitable for manipulation in the logic system.
2. Parsing the protocol to extract specialized logical predicates. E.g., in GNY the protocol is preprocessed to yield certain "not-originate-here" messages.
3. Analyzing the resulting protocol to extract beliefs that can be derived from it.
4. Comparing the set of beliefs to the desired properties/goals of the protocol.

Some of these steps are extremely subtle.

Other Approaches

Some researchers have used model checkers to prove properties of protocols via a state search approach.

Larry Paulson and others have made tremendous strides in proving the correctness of protocols using a general purpose logic and theorem prover. The idea is to model how each step in the protocol advances the state of knowledge of each principle. Then we prove using induction that certain items are never known by various parties.

Both of these are specialized topics within automated reasoning and require sophisticated tools.

Prudent Engineering

Martín Abadi and Roger Needham in "Prudent Engineering Practice for Cryptographic Protocols" give advice for constructing secure protocols:

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

Principle 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.

Principle 3: Mention the principal's name explicitly in the protocol, if the identity is essential to the message meaning.

Principle 4: Be clear about why encryption is being done—secrecy, authentication, binding, etc.

Prudent Engineering (cont.)

Principle 5: A signature on encrypted material doesn't imply that the signer knows the contents.

Principle 6: Be clear about what a nonce means—temporal succession, association, etc.

Principle 7: When using a predictable value, protect it from a replay attack.

Principle 8: If timestamps are used for freshness, watch out for clock skew.

Principle 9: Recent use of a key does not mean the key is new.

Principle 10: It should be possible to tell what encoding is used, to what protocol it belongs, and which step in the protocol.

Principle 11: The designer should understand the trust relationships his protocol depends on.