

A logic of authentication

CS380L: Mike Dahlin

May 4, 2006

1 Preliminaries

1.1 Review

1.2 Outline

1.3 Preview

2 Background

2.1 Scope

- This paper is a tool you need to have in your toolbag
- But, it alone won't make you a security expert
- As earlier reading indicates: sophisticated crypto/protocol attacks are what designers (and academics) obsess about, but most real attacks come from “social engineering” or insider attacks
- Examples from “Why cryptosystems fail”
 - Insider arranges for all accounts to have same password
 - Insider steals money – knows customer will be blamed
 - * Broad conclusion: make the entity responsible for verifying security the one at risk if security authorization fails
 - * In US – burdon of proof on bank (for ATM)
 - * In Britain – burdon of proof on end-user
 - Test-sequence causes machine to disgorge money
 - Fake ATM machine scam
 - ...
- Example from “reflections on trusting trust”
 - Do you trust the compiler, editor, ...
 - Once your system is cracked, you are doomed
 - Principle: minimize “trusted computing base”

2.1.1 Types of Attacks

- Needham (one of the authors) elsewhere classifies 4 types of attacks on crypto systems
 - This paper only looks at one of them.

1. Attacks on the cryptographic algorithm

- brute force
 - Aside: You might be wondering: how does brute force computer decide it got the right message?
 - unstated assumption: message has redundancy to indicate it is "well formed"
 - examples: (1) ASCII text, english words (exceedingly unlikely random bits turn into ascii or english), (2) checksum, (3) magic number
 - BTW DES (56 bits) isn't enough any more
 - * Michael Wiener 1993: build a search machine (CMOS chips)
 - \$1 million → 3.5 hours
 - \$10 million → 21 minutes

key idea: easy to parallelize/build hardware: no per-key I/O just load each chip with "starting key" "encrypted message" "plaintext message"

 - * 2002 – assume (conservatively) halve cost every 2 years
 - * a \$1M machine can crack 1 password every 600 seconds or so
 - * about 100 passwords/day → 30,000/year → 100K passwords during a 3-year lifetime → \$10/password
 - * Don't use DES-56 for secrets worth more than \$1.00
 - * Question: How much did it cost NSA to crack a password when they approved DES in the mid 1970s? *about \$1000?*

2. Attacks on the message

e.g. DES codes messages as separate 64 bit blocks

```
S-->A
<AAAAAAAA><BBBBBBBB><CCCCCCCC>...
"You can trust" "machine bar" "to act as badguy"
```

```
S-->A
<DDDDDDDD><EEEEEEEE><FFFFFFFF>...
"You can trust" "machine foo" "to act as dahlin"
```

adversary could munge second message
<DDDDDDDD><BBBBBBBB><FFFFFFFF>...

--> solution:

1. checksums across entire message
2. chain encryption state across message

All standard practise today

3. Attacks on keys based on guessing

- Humans can't remember 56-bit DES keys (let alone 511-bit RSA keys)
- These keys generated from something humans can remember: passwords
- Humans generate really bad passwords
 - (e.g. the space of all likely passwords is a small subset of 2^{56} or 2^{511})
 - * common words (english or other languages)
 - * names (TV, movies, music, famous people, nicknames, brand names...)
 - * easily obtained information (birthday, licens #, userid..)
 - * keyboard batterns "qwerty"
 - * simple permutations (eg. backwards)
 - * systematic substitution ($o \rightarrow 0, l \rightarrow 1$)
 - * passwords on other systems
 - e.g. Internet work (nov 1988)
 - * no password
 - * user name
 - * user name appended to itself
 - * nickname for user name
 - * last name
 - * last name backwards
 - * 432 word dictionary
 - * dictionary of english words

4. attacks on the protocol by a set of messages

- adversary replays and misues my messages
- e.g. consider how often I say "Hello, mike dahlin here"
 - here I am on machine redhook",
 - "here I am on redhook running telnet to senna",
 - "here I am on machine senna", ...)
- solution: timestamps and nonces
- adversary uses message from one part of protocol in another part of a different conversation
- (e.g. the CCITTT example in paper)

2.2 motivation

- Background:
- Needham and Schroeder built a distributed authentication protocol published 1978
- They were pretty famous and their protocol did (just about) exactly what you want it to do → so this protocol became famous and was actually used pretty widely

- When it got to be widely used, people found a bug
- This upset Needham and Schroeder
 - they had thought long and hard about the protocols and didn't realize they were making a much stronger assumption about one message than they wanted to
 - Cryptography has this tradition of naming protocols after their inventors – the flawed protocol was called the "Needham Schroeder protocol"
- They are pretty smart guys and they made this mistake conclude: need a better way to design protocols
- I'm not sure of the exact timing, but I think the following is not too far off. The original protocol was published in 1978; the correction to the protocol was published in 1987 This bug hung around for a long time!
- Also, CCITT protocol made it a long ways through international standards process before this paper blew it out of the water.

2.2.1 The needham schroeder protocol

1. $A \rightarrow S: A, B, N_a$ (Note: N_a is a nonce)
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. $A \rightarrow B: \{N_b - 1\}_{K_{ab}}$

- Notice this looks pretty much like kerberos
 - (Actually, kerberos looks pretty much like this!)
- Intuition:
 - Step 2: S sends A K_{ab} and $\{K_{ab}\}_{K_{bs}}$ – all encrypted by K_{as}
 - A has and believes K_{ab} (in fact, believes $A \stackrel{K_{ab}}{\leftrightarrow} B$)
 - Step 3: A sends B $\{K_{ab}\}_{K_{bs}}$
 - B has and believes (?) K_{ab}
 - Step 4-5: A and B handshake nonces to make sure they're both currently talking to each other
- Claim: At the end of the protocol, A knows it is talking to B and vice versa.
 - **More precisely:** the holder of K_{as} knows it is talking to the holder of K_{bs} and vice versa
- QUESTION: What's the problem with this?
- ANSWER: Message 3 is not protected by nonces
 - There's no way for B to conclude that the K_{ab} it receives is a current key
 - Example: Intruder has unlimited time to crack an old session key and reuse it as if it were fresh

- Example: Suppose A’s private key were compromised Intruder uses K_a to get K_{as} for many services $s \rightarrow$ intruder can continue to use these session keys even after K_a ’s private key is changed
- Example: I am a disgruntled employee. Before I get fired, I run the first few steps of the protocol a bunch of times, gathering up a bunch of tickets $\{K_{ab}, A\}_{K_{bs}}$ for all of the different servers B in our system (mail server, file server, database, ...). After I get fired, I can continue to log into all of the company’s servers
- This is easy to see if we follow the “prudent practice” of covering up “ $X \rightarrow Y$ ” and all unencrypted stuff. If you don’t have the implicit assumption that the last messages are between A and B but instead imagine that all messages are anonomously broadcast, then it is apparant that an adversary that has an arbitrarily old key K_{ab} and arbitrarily old certificate $\{K_{ab}, A\}_{K_{bs}}$ can run steps 3, 4, and 5 and convince B to use that key any time it wants.
- But this still requires intuition (and is easier in hindsight than w/o); how to find this mechanically and w/o the benefits of hindsight?
- In BAN logic, we will discover that B **believes** S **once said** ($A \stackrel{K_{ab}}{\leftrightarrow} B$), but we will not be able to show that B **believes** S **believes** ($A \stackrel{K_{ab}}{\leftrightarrow} B$)
 - So we won’t be able to take S **has jurisdiction over** ($A \stackrel{K}{\leftrightarrow} B$) and upgrade to B believing that it has a good key.

2.2.2 Timestamps and nonces

- goal: avoid being confused by replays
- (x) is **fresh** == this message has never gone over the network before
- you do this by including a nonce
 1. a timestamp in the message
 2. challenge-response:
 - If I issue a new challenge and get a new response then the message is fresh
- In the logic, the only way to upgrade from *principle once said* X to *principle believes* X is the nonce verification rule:

$$\frac{P \text{ believes } (X) \text{ is fresh , } P \text{ believes } Q \text{ once said } X}{P \text{ believes } Q \text{ believes } X}$$
- In the example, after B decrypts the message, we have
 - B **believes** S **once said** $A \stackrel{K_{ab}}{\leftrightarrow} B$
e.g., "A and B can communicate using shared key K_{ab} "
- There’s no way to get from that step to a statement about belief unless you include as an initial assumption
 - B **believes** ($A \stackrel{K_{ab}}{\leftrightarrow} B$) is **fresh**
- **If** you make that assumption that B accepts the key as new, then you can proceed:

- B **believes** S **believes** A $\stackrel{K_{ab}}{\leftrightarrow}$ B
(e.g., B believes S believes A and B can communicate using K_{ab})
- B **believes** A $\stackrel{K_{ab}}{\leftrightarrow}$ B
(e.g., B believes A and B can communicate using K_{ab} – **authority**)
- ...

3 Admin

4 BAN Logic

4.1 Definitions and notation

Verbose	Blackboard
A believes X	A \models X
A once said X	A \sim X
A sees X	A \triangleleft X
A has jurisdiction over X	A \models X
(X) is fresh	$\#(X)$
K is a shared key for communicating between A and B	A $\stackrel{K}{\leftrightarrow}$ B
V is a secret shared between A and B	A $\stackrel{V}{\rightleftharpoons}$ B
K is B's public key	$\stackrel{K}{\mapsto}$ B

4.1.1 Key postulates in the logic

1. message meaning

- This is the rule that lets you upgrade from
P **sees** X
to
P **believes** Q **once said** X
- e.g., for shared keys
$$\frac{A \text{ believes } (A \stackrel{K_{ab}}{\leftrightarrow} B), A \text{ sees } X_{K_{ab}}}{A \text{ believes } B \text{ once said } X}$$
- To upgrade from "A **sees** X" to "A **believes** B **once said** X" X must be associated with a secret B has:
 - A $\stackrel{K_{ab}}{\leftrightarrow}$ B – A and B's shared key
 - B's private key
 - A and B's shared secret

2. Nonce verification

- lets you upgrade from
Q **once said** X
to
Q **believes** X
- discussed above
$$\frac{A \text{ believes } (X) \text{ is fresh}, A \text{ believes } B \text{ once said } X}{A \text{ believes } B \text{ believes } X}$$

3. jurisdiction

- Lets you upgrade from A **believes** X to B **believes** X
- e.g., lets you transfer beliefs from authority to someone who trusts authority

$$\frac{A \text{ believes } B \text{ has jurisdiction over } X, A \text{ believes } B \text{ believes } X}{A \text{ believes } X}$$

4. etc.

- (joining, dividing, associating freshness with entire message...
- Paper is fairly technical and detailed. But, 99% of time you end up applying the same 3 rules in the same order in the same way:
 message meaning, nonce verification, jurisdiction
 message meaning, nonce verification, jurisdiction
 message meaning, nonce verification, jurisdiction
 message meaning, nonce verification, jurisdiction
 message meaning, nonce verification, jurisdiction
 message meaning, nonce verification, jurisdiction
 message meaning, nonce verification, jurisdiction
 ...
- Technical details are important for avoiding silly blunders like wrongly associating two parts of a message that are not encrypted together.
- One criticism of BAN: Not all silly blunders are formalized
 E.g., doesn't formalize things like "secrets need to be secret", so a protocol that sends the key on the wire in the clear can slip through.
 E.g., translation from "formalization" to "concrete encoding" leaves room for errors where ambiguity allows messages to be substituted from prior rounds of protocol

4.2 Needham Shroeder protocol example

Assumptions

- $A \models S \Rightarrow A \stackrel{K}{\leftrightarrow} B$
- $B \models S \Rightarrow A \stackrel{K}{\leftrightarrow} B$
- $A \models \#(Na)$
- $B \models \#(Nb)$
- $A \models A \stackrel{K_{as}}{\leftrightarrow} S$
- $S \models A \stackrel{K_{as}}{\leftrightarrow} S$
- $B \models B \stackrel{K_{bs}}{\leftrightarrow} S$
- $S \models B \stackrel{K_{bs}}{\leftrightarrow} S$

Protocol analysis:

1. $A \rightarrow S: A, B, N_a$ (Note: N_a is a nonce)
2. $S \rightarrow A: \{N_a, A \stackrel{K_{ab}}{\leftrightarrow} B, \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
 - Apply message meaning:
 $A \models S \sim \{N_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}$

- Apply “if one part of a formula is fresh, the entire formula must be fresh” (rule 5 in article)

$$A \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$$
 - Apply nonce verification:

$$A \models S \models A \stackrel{K_{ab}}{\leftrightarrow} B$$
 - Apply jurisdiction:

$$A \models A \stackrel{K_{ab}}{\leftrightarrow} B$$
3. $A \rightarrow B: \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$
- Apply message meaning:

$$B \models S \vdash A \stackrel{K_{ab}}{\leftrightarrow} B$$
 - We’re stuck!
 - No way to get to $B \models S \models A \stackrel{K_{ab}}{\leftrightarrow} B$
4. $B \rightarrow A: \{N_b\}_{K_{ab}}$
- Apply message meaning:

$$A \models B \vdash N_b,$$

$$A \models B \vdash A \stackrel{K_{ab}}{\leftrightarrow} B$$
 - Apply nonce verification:
 (recall $A \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$)

$$A \models B \models A \stackrel{K_{ab}}{\leftrightarrow} B$$
 - Notice, this message is a bit subtle. The nonce is not here to prove freshness to A. Ignore it in interpreting this message. A already knows that K_{ab} is fresh, and already knows that $A \stackrel{K_{ab}}{\leftrightarrow} B$, so this message proves to A that B believes in the shared key.
 - From A’s point of view, this message could just be

$$B \rightarrow A: \{0_{K_{ab}}\}$$
 - Without this message, A would end the protocol knowing that $A \stackrel{K_{ab}}{\leftrightarrow} B$, but not knowing that the key had successfully been transmitted to B.
 - Would this message have been better formalized as:

$$4. B \rightarrow A: \{A \stackrel{K_{ab}}{\leftrightarrow} B, N_b\}_{K_{ab}}$$
5. $A \rightarrow B: \{N_b - 1\}_{K_{ab}}$
- Without a shared key, no way to apply **message meaning** and no way for B to know who said this. We’re stuck.
 - What was desired:
 - Apply (erroneously, it turns out) message meaning

$$B \models A \vdash \{N_b - 1, A \stackrel{K_{ab}}{\leftrightarrow} B\}$$
 - Apply nonce verification (to get this far, $B \models \#(Kab)$):

$$B \models A \models A \stackrel{K_{ab}}{\leftrightarrow} B$$

One last point

- The construction of messages 4 and 5 suggests (I think incorrectly) that N_b has something to do with proving freshness. E.g., that it is a challenge/response for A to prove it has the key. All we really need to do is send 2 different messages encrypted by the shared key. E.g.,

- 4) $B \rightarrow A: \{Message4\}_{K_{ab}}$
- 5) $A \rightarrow B: \{Message5\}_{K_{ab}}$

- How should this be formalized?

- 4) $B \rightarrow A: \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}} from B$
- 5) $A \rightarrow B: \{A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}} from A$

Note that “from X” is similar to Kerberos formalization in Section 4 of the paper – it distinguishes the messages to prevent replay.

- A concrete realization might be

- 4) $B \rightarrow A: \{B, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$
- 5) $A \rightarrow B: \{A, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$
- This, I think, follows the “prudent practice” of “Explicit communication...interpretation of the message should depend only on its content.”
- English translation: “After receiving K_{ab} , B says that K_{ab} is a key that A and B can use to communicate.” and similarly for msg 5.

4.3 Kerberos

Kerberos is a follow-on to Needham Schroeder. Widely deployed.

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: \{T_s, B, K_{ab}, \{T_s, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B: \{T_s, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
4. $B \rightarrow A: \{T_a + 1\}_{K_{ab}}$

Formalized as

2. $S \rightarrow A: \{T_s, A \stackrel{K_{ab}}{\leftrightarrow} B, \{T_s, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B: \{T_s, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}, \{A, T_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$
4. $B \rightarrow A: \{B, T_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$

Analysis

Assumptions

- $A \equiv S \vdash A \stackrel{K}{\leftrightarrow} B$
- $B \equiv S \vdash A \stackrel{K}{\leftrightarrow} B$
- $A \equiv \#(T_s)$ (**Note:** Time service part of TCB!)
- $B \equiv \#(T_s)$
- $B \equiv \#(T_a)$
- $A \equiv A \stackrel{K_{as}}{\leftrightarrow} S$
- $S \equiv A \stackrel{K_{as}}{\leftrightarrow} S$
- $B \equiv B \stackrel{K_{bs}}{\leftrightarrow} S$
- $S \equiv B \stackrel{K_{bs}}{\leftrightarrow} S$

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

- message meaning: $A \models S \sim \{T_s, A \stackrel{K_{ab}}{\leftrightarrow} B\}$
- “if one part of formula is fresh, entire formula is fresh”: $A \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$
- nonce verification: $A \models S \models A \stackrel{K_{ab}}{\leftrightarrow} B$
- jurisdiction: $A \models A \stackrel{K_{ab}}{\leftrightarrow} B$

3. $A \rightarrow B: \{T_s, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}, \{from A, T_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$

For the first half of the message:

- message meaning: $B \models S \sim \{T_s, A \stackrel{K_{ab}}{\leftrightarrow} B\}$
- “if one part of formula is fresh, entire formula is fresh”: $B \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$
- nonce verification: $B \models S \models A \stackrel{K_{ab}}{\leftrightarrow} B$
- jurisdiction: $B \models A \stackrel{K_{ab}}{\leftrightarrow} B$

For the second half of the message:

- message meaning: $B \models A \sim \{A, T_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}$
- “if one part of formula is fresh, entire formula is fresh”: $B \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$
- nonce verification: $B \models A \models A \stackrel{K_{ab}}{\leftrightarrow} B$

What does the second half of the message mean?

4. $B \rightarrow A: \{from B, T_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$

- message meaning: $A \models B \sim \{B, T_a, A \stackrel{K_{ab}}{\leftrightarrow} B\}$
- “if one part of formula is fresh, entire formula is fresh”: $A \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)$
- nonce verification: $A \models B \models A \stackrel{K_{ab}}{\leftrightarrow} B$

NB: This last “handshake” convinces A that B exists and has the key. Essentially, A knows the protocol terminated successfully.