A logic of authentication

CS380L: Mike Dahlin

November 25, 2002

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 seperate 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. checsums across entire message
2. chain encryption state across message

        All standard practise today
```

```
Correct implementation assumed the ``conjunction'' rules in BAN paper
```

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 $\rightarrow$ 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 protocl 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}
    $\rightarrow$ A has and believes K_{ab} (in fact, believes A $\overset{K_{ab}}{\leftrightarrow}$ B)
  - Step 3: A sends B $\{K_{ab}\}K_{bs}$
    $\rightarrow$ 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
- 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

- In BAN logic, we will discover that B **believes** S **once said** ($A \overset{Kab}{\leftrightarrow} B$), but we will not be able to show that B **believes** S **believes** ($A \overset{Kab}{\leftrightarrow} B$)
  - So we won't be able to take S **has jurisdiction over** ($A \overset{K}{\leftrightarrow} B$) 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:

$$\frac{\text{P } \textbf{believes} \text{ (X) is } \textbf{fresh} \text{ , P } \textbf{believes} \text{ Q } \textbf{once said} \text{ X}}{\text{P } \textbf{believes} \text{ Q } \textbf{believes} \text{ X}}$$

- In the example, after B decrypts the message, we have
  - B **believes** S **once said** $A \overset{Kab}{\leftrightarrow} B$
    e.g., "A and B can communicate using shared key Kab"
- There's no way to get from that step to a statement about belief unless you include as an initial assumption
  - B **believes** ($A \overset{Kab}{\leftrightarrow} B$) is **fresh**
- **If** you make that assumption that B accepts the key as new, then you can proceed:
  - B **believes** S **believes** $A \overset{Kab}{\leftrightarrow} B$
    (e.g., *B believes S believes A and B can communicate using* K_{ab})
  - B **believes** $A \overset{Kab}{\leftrightarrow} B$
    (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	$A \mid\!\sim X$
A **sees** X	$A \triangleleft X$
A **has jurisdiction over** X	$A \Rightarrow X$
(X) is **fresh**	$\#(X)$
K is a shared key for communicating between A and B	$A \overset{K}{\leftrightarrow} B$
V is a secret shared between A and B	$A \overset{V}{\rightleftharpoons} B$
K is B's public key	$\overset{K}{\longmapsto} B$

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

$$\frac{\text{A } \textbf{believes} \text{ (A} \overset{Kab}{\leftrightarrow} \text{B), A } \textbf{sees} \text{ } X_{K_{ab}}}{\text{A } \textbf{believes} \text{ B } \textbf{once said} \text{ X}}$$

   - To upgrade from "A **sees** X" to "A **believes** B **once said** X" X must be associated with a secret B has:
     - $A \overset{Kab}{\leftrightarrow} B$ – 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

$$\frac{\text{A } \textbf{believes} \text{ (X) is } \textbf{fresh} \text{ , A } \textbf{believes} \text{ B } \textbf{once said} \text{ X}}{\text{A } \textbf{believes} \text{ B } \textbf{believes} \text{ X}}$$

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

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

4. etc.

   - (joining, dividing, associating freshness with entire message...

4.2 Needham Shroeder protocol example

Assumptions

A $\mid\equiv$ S $\mid\Rightarrow$ A $\overset{K}{\leftrightarrow}$ B
B $\mid\equiv$ S $\mid\Rightarrow$ A $\overset{K}{\leftrightarrow}$ B
A $\mid\equiv \#(Na)$
B $\mid\equiv \#(Nb)$
A $\mid\equiv$ A $\overset{Kas}{\leftrightarrow}$ S
S $\mid\equiv$ A $\overset{Kas}{\leftrightarrow}$ S
B $\mid\equiv$ B $\overset{Kbs}{\leftrightarrow}$ S
S $\mid\equiv$ B $\overset{Kbs}{\leftrightarrow}$ S

Protocol analysis:

1. $A \rightarrow S$: A, B, N_a (Note: N_a *is a nonce*)

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

   - Apply message meaning:
     A $\mid\equiv$ S $\mid\!\sim \{N_a$, A $\overset{Kab}{\leftrightarrow}$ B$\}$
   - Apply "if one part of a formula is fresh, the entire formula must be fresh" (rule 5 in article)
     A $\mid\equiv \#(A \overset{Kab}{\leftrightarrow} B)$
   - Apply nonce verification:
     A $\mid\equiv$ S $\mid\equiv$ A $\overset{Kab}{\leftrightarrow}$ B
   - Apply jurisdiction:
     A $\mid\equiv$ A $\overset{Kab}{\leftrightarrow}$ B

3. $A \rightarrow B$: $\{$A $\overset{Kab}{\leftrightarrow}$ B$\}_{K_{bs}}$

   - Apply message meaning:
     B $\mid\equiv$ S $\mid\!\sim$ A $\overset{Kab}{\leftrightarrow}$ B
   - We're stuck!
     - No way to get to B $\mid\equiv$ S $\mid\equiv$ A $\overset{Kab}{\leftrightarrow}$ B

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

   - Apply message meaning:
     A $\mid\equiv$ B $\mid\!\sim N_b$,
     A $\mid\equiv$ B $\mid\!\sim$ A $\overset{Kab}{\leftrightarrow}$ B
   - Apply nonce verification:
     (recall A $\mid\equiv \#(A \overset{Kab}{\leftrightarrow} B)$)
     A $\mid\equiv$ B $\mid\equiv$ A $\overset{Kab}{\leftrightarrow}$ 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 $\overset{Kab}{\leftrightarrow}$ 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 $\overset{Kab}{\leftrightarrow}$ B, but not knowing that the key had successfully been transmitted to B.
   - Would this message have been better formalized as:
     4. $B \rightarrow A$: $\{A \overset{Kab}{\leftrightarrow} B, N_b\}_{K_{ab}}$

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 $\mid\equiv$ A $\mid\!\sim \{N_b - 1, A \overset{Kab}{\leftrightarrow} B\}$
     - Apply nonce verification (to get this far, B $\mid\equiv \#(Kab)$):
       B $\mid\equiv$ A $\mid\equiv$ A $\overset{Kab}{\leftrightarrow}$ 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?
- At some level what we are saying is:
  - 4) $B \rightarrow A$: $\{A \overset{Kab}{\leftrightarrow} B\}_{K_{ab}}$
  - 5) $A \rightarrow B$: $\{A \overset{Kab}{\leftrightarrow} B\}_{K_{ab}}$

  But since we need to ignore the "$C \rightarrow D$" part of the message, how would these things be distinguishable? (e.g., how to avoid replay attacks?)
- Proposal:
  - 4) $B \rightarrow A$: $\{B, A \overset{Kab}{\leftrightarrow} B\}_{K_{ab}}$
  - 5) $A \rightarrow B$: $\{A, A \overset{Kab}{\leftrightarrow} B\}_{K_{ab}}$
  - This, I think, follows the "prudent practice" of "Explicit communication...interpretation of the message should depend only on its content."
  - English translation: "After receiving Kab, B says that Kab is a key that A and B can use to communicate." and similarly for msg 5.