A Logic of Authentication.

Burrows90: Michael Burrows, Martin Abadi, Roger Needham, ACM Trans. on Computer Systems (TOCS), vol 8, no 1, February 1990.

Big Picture Redux

- Large systems very complex
  - Lucid, clear reasoning & definitions (e.g., Saltzer)
  - "the bridge approach" - redundancy, defense-in-depth
- Subsystems amenable to more formal reasoning
  - Cryptographic protocols
  - Transactional protocols/etc.
  - Crypto itself (out of scope for 712)
  - Algorithmic correctness
  - Correctness of smaller chunks of code
- In keeping with philosophy: have to do everything...

Take-home lesson

- Cryptography itself: Leave it to the experts
  - Even theirs gets broken. :)
- Cryptographic protocols:
  - When possible, use off-the-shelf (see CRC)
  - BUT: most real systems / dist systems need them
    - Should understand well enough to evaluate/adapt to system...
    - New technologies -> new (mis)uses
    - e.g., cookies and Web authentication

Understanding Authentication

- Explicit logic to help understanding/belief, assumptions, unnecessary transfers
- Focus on beliefs of trustworthy principals
- Not for finding code bugs, deadlocks, explicit release of inappropriate information, untrustworthy principals
  - Follow on work will beat on these assumptions
- Core tool: freshness; evidence against a message having been replayed

Logic Basics

- P believes X
- P sees X
  - Received X
- P said X
  - Believed & sent X once
    - follow on work separates these
- P controls X
  - Jurisdiction/believable
- Fresh (X)
  - X not said "before now"
  - X is a Nonce, usually timestamped or sequence numbered
- P <-K-> Q
  - share valid key K b/w only P, Q
- |-K-> P
  - has valid public key K
- P <=X=> Q
  - X is shared secret b/w only P, Q
- {X}K
  - X encrypted by K from P
    - assume P can recognize & ignore its own msgs
- <X>Y
  - X signed by Y, i.e. X,H(X,Y)

Basic logic rules

- Encrypted messages are indivisable (else they are multiple messages), internally redundant so to be recognizable on decryption
  - recognizability is explicit in follow on work
- Message meaning: $\dfrac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P,\quad P \textbf{ sees } \{X\}_K}{P \textbf{ believes } Q \textbf{ said } X}$
  - "See to said"
- Nonce verification: $\dfrac{P \textbf{ believes fresh}(X),\quad P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}$
  - "True now"
- Jurisdiction: $\dfrac{P \textbf{ believes } Q \textbf{ controls } X,\quad P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}$
  - "Authority to say"

Basic logic rules

- You can decompose messages, but not consider two messages as one
- Freshness is transitive: $\dfrac{P \textbf{ believes fresh}(X)}{P \textbf{ believes fresh}(X,\ Y)}$
  - Which is why messages can't be merged
- Cleartext is ignored in logic as it is forgeable (useful as a hint or for performance)
  - follow on work finds fault with this and keeps clear text around so consistency can be checked

Kerberos: set up a session key

- The protocol:
  - server responds to A with a ticket
  - A forwards ticket and authenticator

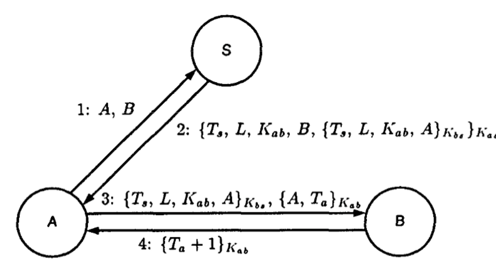


Fig. 1. The Kerberos Protocol.

- Idealized:
  - no lifetime L
  - Ta + 1 gone given msg detectably not same sender

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

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

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

Message 2. $S \to A$: $\{T_s, A \overset{K_{ab}}{\leftrightarrow} B, \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$.

Message 3. $A \to B$: $\{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}, \{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$ from A.

Message 4. $B \to A$: $\{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$ from B.

- Start with assumptions:

A **believes** $A \overset{K_{as}}{\leftrightarrow} S$,
S **believes** $A \overset{K_{as}}{\leftrightarrow} S$,
S **believes** $A \overset{K_{ab}}{\leftrightarrow} B$,
A **believes** (S **controls** $A \overset{K}{\leftrightarrow} B$),
A **believes fresh**(T_s),
B **believes** $B \overset{K_{bs}}{\leftrightarrow} S$,
S **believes** $B \overset{K_{bs}}{\leftrightarrow} S$,
B **believes** (S **controls** $A \overset{K}{\leftrightarrow} B$),
B **believes fresh**(T_s),
B **believes fresh**(T_a).

- Dependence on synch'd clocks for timestamp freshness
  - for known skew, retain all msgs in skew window & verify no replays in window

A receives Message 2. The annotation rules yield that

A **sees** $\{T_s, (A \overset{K_{ab}}{\leftrightarrow} B), \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$

holds afterward. Since we have the hypothesis

A **believes** $A \overset{K_{as}}{\leftrightarrow} S$

the message-meaning rule for shared keys applies and yields the following:

A **believes** S **said** $(T_s, (A \overset{K_{ab}}{\leftrightarrow} B), \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})$

One of our rules to break conjunctions (omitted here) then produces

A **believes** S **said** $(T_s, (A \overset{K_{ab}}{\leftrightarrow} B))$

Moreover, we have the following hypothesis:

A **believes fresh**(T_s)

The nonce-verification rule applies and yields

A **believes** S **believes** $(T_s, A \overset{K_{ab}}{\leftrightarrow} B)$

Again, we break a conjunction, to obtain the following:

A **believes** S **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

Then, we instantiate K to K_{ab} in the hypothesis

A **believes** S **controls** $A \overset{K}{\leftrightarrow} B$

deriving the more concrete

A **believes** S **controls** $A \overset{K_{ab}}{\leftrightarrow} B$

Finally, the jurisdiction rule applies, and yields the following:

A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

This concludes the analysis of Message 2.

A passes the ticket on to B, together with another message containing a timestamp. Initially, B can decrypt only the ticket:

B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

Logically, this result is obtained in the same way as that for Message 2, via the message-meaning, nonce-verification, and jurisdiction postulates.

Knowledge of the new key allows B to decrypt the rest of Message 3. Through the message-meaning and the nonce-verification postulates, we deduce the following:

B **believes** A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

The fourth message simply assures A that B believes in the key and has received A's last message. After new applications of the message-meaning and nonce-verification postulates to the fourth message, the final result is as follows:

A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$ $\quad$ B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$
A **believes** B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$ $\quad$ B **believes** A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$

Andrew

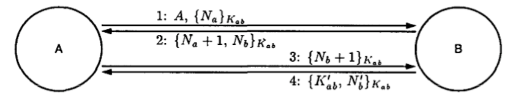


Fig. 2. The Andrew Square RPC Handshake.

- Used to establish "extra" session key subservient to a long running session
- Nonces only known to be fresh by originator
- Need to add nonce Na into message 4 so A sees something fresh
  - otherwise, old msg 4 replayable to revert to compromised K'ab

Message 1. $A \to B$: $\{N_a\}_{K_{ab}}$.
Message 2. $B \to A$: $\{N_a, N_b\}_{K_{ab}}$.
Message 3. $A \to B$: $\{N_b\}_{K_{ab}}$.
Message 4. $B \to A$: $\{A \overset{K'_{ab}}{\leftrightarrow} B, N'_b\}_{K_{ab}}$.

A **believes** $A \overset{K_{ab}}{\leftrightarrow} B$
A **believes** (B **controls** $A \overset{K}{\leftrightarrow} B$)
A **believes fresh**(N_a)

B **believes** $A \overset{K_{ab}}{\leftrightarrow} B$
B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$
B **believes fresh**(N_b)
B **believes fresh**(N'_b)

B **believes** $A \overset{K'_{ab}}{\leftrightarrow} B$
A **believes** B **said** $(A \overset{K'_{ab}}{\leftrightarrow} B, N'_b)$
B **believes** A **believes** N_b
A **believes** B **believes** (N_a, N_b)

Needham-Schroeder

- Issue: Freshness of certification
  - need to add timestamps to public key certifications from S & re-obtain periodically

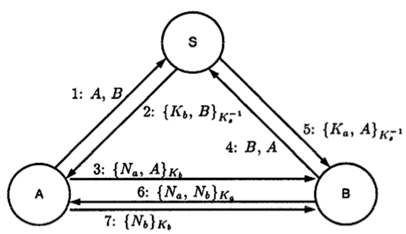


Fig. 3. The Needham-Schroeder Public-Key Protocol.

- Note that idealization sees Na and Nb as shared secrets as well as nonces
  - creating the idealization requires detailed understanding of protocol's later uses

Why was this missed?

- Likely a question of *threat model*: Original designers didn't consider compromise of *Ka, Kb* and need to change the keys associated with those principals
- Common cause of vulnerability
  - In everything -- physical, systems (insider threats? trojaned 3rd-party code? etc.)
  - even crypto. e.g., Binham & Shamir differential analysis - late 1980s
    - In the early 70s, the NSA requested a seemingly innocuous change to some of the constants in DES
      - That change made DES very resilient to Diff. crypt...
    - Side-channel attacks (timing of algo - different instructions take different amounts of time). Watching heat of processor. Heating processor. etc.

Attacks against

- That attack was known previously (Dorothy Denning & Giovanni Maria Sacco, 1981)
- But the protocol is actually more dangerously broken than that!
  - Man-in-the-middle attack
  - This paper didn't catch it. Oops.

Lowe MitM attack

- Imposter I convinces A to talk to him:
- A->I : {Na, A}Ki
- I->B: {Na, A}Kb
- B->I: {Na, Nb}Ka
- I->A: {Na, Nb}Ka
- A->I: {Nb}Ki
- I->B: {Nb}Kb
  - Fix: message 6 B->A: {B,Na,Nb}Ka

Defense-in-depth

- Thought Q: How to protect such a system?
  - Given: Not 100% confident in crypto
  - Not 100% confident in protocols
  - Not 100% confident in impl...
- Physical isolation
- Needham-Schroeder attack is MitM
  - Link encryption?
  - Secure the routers (ISPs today "cloak" routers)
- Contain effects of compromise (one user, one server, etc.)

Eval

- Mostly a "by omitted proofs" paper :-)
- Power from important existing protocols
  - Shows logic flaw Andrew had used & repaired
  - Shows logic flaw in author's important protocol
  - Shows logic flaw in an international standard
- Follow ons relax/explicit assumptions
  - GNY90: recognizability, repeat w/o belief
  - Nessbett90: bad use of keys, disclosure
  - Boyd93: hold onto cleartext for consistency
  - Rubin94: non-monotomic, ie., temporal logic

Authentication in 2000+

- The web: millions of new distributed systems
  - Authored by millions of new programmers. :)
  - SSL/TLS provides one standard, but
    - Many web sites don't like SSL (speed)
    - Hardly any use SSL certs for authentication (browser support, etc.)
    - Many don't use HTTP authentication (not very secure over unencrypted connection)
    - Many like persistent login cookies for user convenience

Threat model

- Interrogative adversary
  - Can make a reasonable # of queries to a web server (e.g., 1/second)
    - Adaptive chosen message attacks
  - Can't sniff
    - Almost any user can mount w/out special access to network
  - Can use info publicly available on web server
    - User lists if available, etc.
  - This is a pretty basic threat model...

HTTP cookies

- Recall that HTTP is stateless
- Any state must be sent to client and have client send it back (cookie)
  - Can set cookie value
  - Set duration (some time or immediate discard)
  - Control which servers cookie is sent to
    - Host, domain, port, SSL required or not

Balancing concerns

- This slide again???
- Performance: SSL, encryption speed
  - Even with today's machines, SSL is *not* cheap
  - 100s of reqs/sec vs. 1000s or 10,000s
- User acceptability / convenience
- Security (against what threats??)

Auth models

- Coarse-grained: Verify authorization, but not necessarily identity
  - e.g., “valid subscriber to Wall Street Journal”
  - For services w/no accounting/customization
- Fine-grained: Verify user identity as well

Confidentiality

- Some sites use SSL for everything (etrade), but
- many protect only login sessions (passwords) and confidential email (actually placing an order/CC#/etc)
- (Again: Cost/performance/security trade)

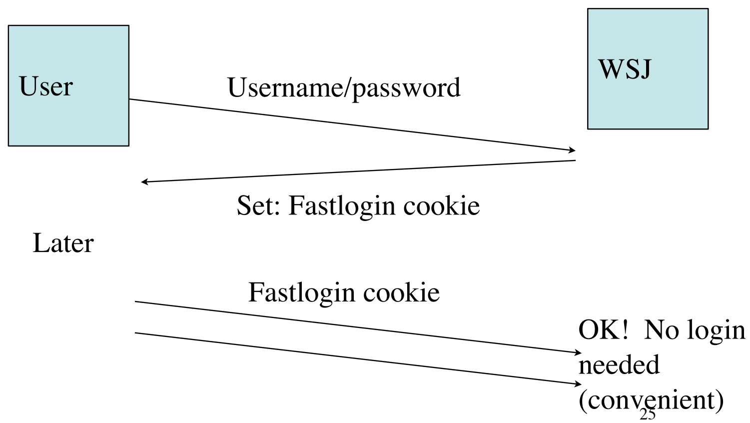
Threats

- Existential forgery:
  - Become {some unspecified} valid user
  - Gain access to content, but can’t target person
- Selective forgery:
  - “Login as Joe”
- Total break:
  - Compromise the authenticator minting mechanism
  - Can off-line construct valid auths for any user

Examples

- We broke a bunch of them...
- All had home-brewed authentication schemes (bad programmer! no cookie!)
-

Wall Street Journal



The cookie

- fastlogin = username + crypt(username + server secret)
  - Crypt is a one-way hash function (same one used to secure UNIX passwords). Can't be inverted.
  - BUT:
    - Crypt only uses first 8 characters of input!
    - crypt("mynameisdave") == crypt("mynameisjoe")

The attack

- fastlogin(8 character username) == crypt(username)
  - That's not very strong. :-)
- fastlogin(7 char username) ==
  crypt(username + 1 secret character)
  - Can brute-force in 128 tries
- fastlogin(6 char username) ==
  crypt (username + char from above + 1 secret character)
  ...
- Can discover "secret" in 128*8 steps

(It gets worse)

- Secret: "March20" (day WSJ.com went online)
- The site used the secret as the salt ("Ma") -- leaked even more information
- They didn't change the salt
  - Didn't seem to hurt things; already insecure
- No per-user revocation
  - Only way to revoke was changing secret key for entire site (which they never did)
- No lifetime/freshness...
- Even allows *invalid* accounts
  - WSJ presumbaly didn't want DB lookup on access (reasonable)
  - Could make up a username, generate cookie...

Other systems

- Fatbrain.com used a sequence number as a validator
  - The sequence # was global and monotonically incremented...
  - Could login as any user
  - And then change email address w/out needing to authenticate
  - And then click “mail me my password”
  - and then 0wn user’s account...

Doing it right

- Don’t reinvent the wheel
- Understand the crypto and protocols enough to apply them (e.g., crypt == 8 bytes...)
- Don’t rely on protocol secrecy
  - A gaggle of grad students broke 8 websites in a few weeks...
- Re-authenticate before changing security-sensitive things {email, passwords, etc.}

Make auths unforgeable

- Good way:
- cookie = {
  expiration = time
  data=s
  digest=MAC_k(expiration=t,data=s)
  }
  - Use an existing MAC, like HMAC-SHA1!