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 \(^2\)...

---

**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 <\text{-K}> Q
  - share valid key K b/w only P, Q
- \text{\texttt{\textasciitilde K}} \rightarrow P
  - has valid public key K
- P <\Rightarrow= 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 indivisible (else they are multiple messages), internally redundant so to be recognizable on decryption
  - recognizability is explicit in follow on work
- Message meaning:
  \[
  P \text{ believes } Q \iff P, P \text{ sees } |X|_K
  \]
  - “See to said”
- Nonce verification:
  \[
  P \text{ believes fresh}(X), P \text{ believes } Q \text{ said } X
  \]
  - “True now”
- Jurisdiction:
  \[
  P \text{ believes } Q \text{ controls } X, P \text{ believes } Q \text{ believes } X
  \]
  - “Authority to say”

Kerberos: set up a session key

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

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

- 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

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.

Needham-Schroeder

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

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

Fig. 2. The Andrew Square RPC Handshake.

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.

Andrew

- 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

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.

Needham-Schroeder

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

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

Fig. 2. The Andrew Square RPC Handshake.

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.
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\}\textsubscript{Ki}
  • I->B: \{Na, A\}\textsubscript{Kb}
  • B->I: \{Na, Nb\}\textsubscript{Ka}
  • I->A: \{Na, Nb\}\textsubscript{Ka}
  • A->I: \{Nb\}\textsubscript{Ki}
  • I->B: \{Nb\}\textsubscript{Kb}
  – Fix: message 6 B->A: \{B,Na,Nb\}\textsubscript{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

- \( \text{fastlogin} = \text{username} + \text{crypt} (\text{username} + \text{server secret}) \)
  - 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!
    - \( \text{crypt("mynameisdave") == crypt("mynameisjoe"')} \)

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 presumably 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 own 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!