Public key authenticity
- Public keys don't need to be secret, but they must be right
- Wrong key → can't stop middleperson
- So we still have a pretty hard distribution problem

Symmetric key servers
- Users share keys with server, server distributes session keys
- Symmetric key-exchange protocols, or channels
- Standard: Kerberos
- Drawback: central point of trust

Certificates
- A name and a public key, signed by someone else
  \[ C_A = \text{Sign}_S(A; K_A) \]
- Basic unit of transitive trust
- Commonly use a complex standard “X.509”

Certificate authorities
- “CA” for short: entities who sign certificates
- Simplest model: one central CA
- Works for a single organization, not the whole world

Web of trust
- Pioneered in PGP for email encryption
- Everyone is potentially a CA: trust people you know
- Works best with security-motivated users
  - Ever attended a key signing party?

CA hierarchies
- Organize CAs in a tree
- Distributed, but centralized (like DNS)
- Check by follow a path to the root
- Best practice: sub CAs are limited in what they certify
**PKI for authorization**
- Enterprise PKI can link up with permissions
- One approach: PKI maps key to name, ACL maps name to permissions
- Often better: link key with permissions directly, name is a comment
  - More like capabilities

**The revocation problem**
- How can we make certs "go away" when needed?
- Impossible without being online somehow
  1. Short expiration times
  2. Certificate revocation lists
  3. Certificate status checking

**Outline**
- Key distribution and PKI
- Cryptographic protocols, cont'd
- Blind SQL injection (demo)
- SSH
- SSL/TLS
- More causes of crypto failure

**Abstract protocols**
- Outline of what information is communicated in messages
  - Omit most details of encoding, naming, sizes, choice of ciphers, etc.
- Describes honest operation
  - But must be secure against adversarial participants
- Seemingly simple, but many subtle problems

**Protocol notation**
- $A \rightarrow B : N_B \{T_0, B, N_B\}_K$
  - $A \rightarrow B$: message sent from Alice intended for Bob
  - $B$ (after $\rightarrow$): Bob's name
  - $(\cdot \cdot \cdot)_K$: encryption with key $K$

**Anti-pattern: "oracle"**
- Any way a legitimate protocol service can give a capability to an adversary
- Can exist whenever a party decrypts, signs, etc.
- "Padding oracle" was an instance of this at the implementation level

**Needham-Schroeder**
- Mutual authentication via nonce exchange, assuming public keys (core):
  - $A \rightarrow B : \{N_A, A\}_E_B$
  - $B \rightarrow A : \{N_A, N_B\}_E_A$
  - $A \rightarrow B : \{N_B\}_E_B$

**Needham-Schroeder MITM**
- $A \rightarrow C : \{N_A, A\}_E_C$
- $C \rightarrow B : \{N_A, A\}_E_B$
- $B \rightarrow C : \{N_A, N_B\}_E_A$
- $C \rightarrow A : \{N_A, N_B\}_E_A$
- $A \rightarrow C : \{N_B\}_E_C$
- $C \rightarrow B : \{N_B\}_E_B$
Certificates, Denning-Sacco

- A certificate signed by a trusted third-party $S$ binds an identity to a public key
  - $C_A = \text{Sign}_S(A, K_A)$

- Suppose we want to use $S$ in establishing a session
  - $A \to S : A, B$
  - $S \to A : C_A, C_B$
  - $A \to B : C_A, C_B, \text{Sign}_A(K_{AB})_{K_S}$

Attack against Denning-Sacco

- $A \to S : A, B$
- $S \to A : C_A, C_B$
- $A \to B : C_A, C_B, (\text{Sign}_A(K_{AB}))_{K_S}$
- $B \to S : B, C$
- $S \to B : C_B, C_C$
- $B \to C : C_A, C_B, (\text{Sign}_A(K_{AB}))_{K_C}$

By re-encrypting the signed key, Bob can pretend to be Alice to Charlie.

Envelopes analogy

- Encrypt then sign, or vice-versa?
- On paper, we usually sign inside an envelope, not outside. Two reasons:
  - Attacker gets letter, puts in his own envelope (c.f. attack against X.509)
  - Signer claims “didn’t know what was in the envelope” (failure of non-repudiation)

Design robustness principles

- Use timestamps or nonces for freshness
- Be explicit about the context
- Don’t trust the secrecy of others’ secrets
- Whenever you sign or decrypt, beware of being an oracle
- Distinguish runs of a protocol

Implementation principles

- Ensure unique message types and parsing
- Design for ciphers and key sizes to change
- Limit information in outbound error messages
- Be careful with out-of-order messages

Outline

Key distribution and PKI
Cryptographic protocols, cont’d
Blind SQL injection (demo)
SSH
SSL/TLS
More causes of crypto failure

Short history of SSH

- Started out as freeware by Tatu Ylönen in 1995
- Original version commercialized
- Fully open-source OpenSSH from OpenBSD
- Protocol redesigned and standardized for "SSH 2"
OpenSSH t-shirt

SSH host keys
- Every SSH server has a public/private keypair
- Ideally, never changes once SSH is installed
- Early generation a classic entropy problem
  - Especially embedded systems, VMs

Authentication methods
- Password, encrypted over channel
- .shosts: like .rhosts, but using client host key
- User-specific keypair
  - Public half on server, private on client
- Plugins for Kerberos, PAM modules, etc.

Old crypto vulnerabilities
- 1.x had only CRC for integrity
  - Worst case: when used with RC4
- Injection attacks still possible with CBC
  - CRC compensation attack
- For least-insecure 1.x-compatibility, attack detector
  - Alas, detector had integer overflow worse than original attack

Newer crypto vulnerabilities
- IV chaining: IV based on last message ciphertext
  - Allows chosen plaintext attacks
  - Better proposal: separate, random IVs
- Some tricky attacks still left
  - Send byte-by-byte, watch for errors
  - Of arguable exploitability due to abort
- Now migrating to CTR mode

SSH over SSH
- SSH to machine 1, from there to machine 2
  - Common in these days of NATs
- Better: have machine 1 forward an encrypted connection
  1. No need to trust 1 for secrecy
  2. Timing attacks against password typing

SSH (non-)PKI
- When you connect to a host freshly, a mild note
- When the host key has changed, a large warning

Outline
- Key distribution and PKI
- Cryptographic protocols, cont'd
- Blind SQL injection (demo)
- SSH
- SSL/TLS
- More causes of crypto failure
SSL/TLS
- Developed at Netscape in early days of the public web
  - Usable with other protocols too, e.g. IMAP
- SSL 1.0 pre-public, 2.0 lasted only one year, 3.0 much better
- Renamed to TLS with RFC process
  - TLS 1.0 improves SSL 3.0
  - TLS 1.1 and 1.2 in 2006 and 2008, only gradual adoption

IV chaining vulnerability
- TLS 1.0 uses previous ciphertext for CBC IV
- But, easier to attack in TLS:
  - More opportunities to control plaintext
  - Can automatically repeat connection
- "BEAST" automated attack in 2011: TLS 1.1 wakeup call

Compression oracle vuln.
- Compr($S || A$), where $S$ should be secret and $A$ is attacker-controlled
- Attacker observes ciphertext length
- If $A$ is similar to $S$, combination compresses better
- Compression exists separately in HTTP and TLS

But wait, there's more!
- Too many vulnerabilities to mention them all in lecture
- Kaloper-Meršinjak et al. have longer list
  - "Lessons learned" are variable, though
  - Meta-message: don't try this at home

HTTPS hierarchical PKI
- Browser has order of 100 root certs
  - Not same set in every browser
  - Standards for selection not always clear
- Many of these in turn have sub-CAs
- Also, "wildcard" certs for individual domains

Hierarchical trust?
- No. Any CA can sign a cert for any domain
- A couple of CA compromises recently
- Most major governments, and many companies you've never heard of, could probably make a google.com cert
- Still working on: make browser more picky, compare notes

CA vs. leaf checking bug
- Certs have a bit that says if they're a CA
- All but last entry in chain should have it set
- Browser authors repeatedly fail to check this bit
- Allows any cert to sign any other cert

MD5 certificate collisions
- MD5 collisions allow forging CA certs
- Create innocuous cert and CA cert with same hash
  - Requires some guessing what CA will do, like sequential serial numbers
  - Also 200 PS3s
- Oh, should we stop using that hash function?
CA validation standards
- CA's job to check if the buyer really is foo.com
- Race to the bottom problem:
  - CA has minimal liability for bad certs
  - Many people want cheap certs
  - Cost of validation cuts out of profit
- "Extended validation" (green bar) certs attempt to fix

HTTPS and usability
- Many HTTPS security challenges tied with user decisions
- Is this really my bank?
- Seems to be a quite tricky problem
  - Security warnings often ignored, etc.
  - We'll return to this as a major example later

Outline
- Key distribution and PKI
- Cryptographic protocols, cont'd
- Blind SQL injection (demo)
- SSH
- SSL/TLS
- More causes of crypto failure

Random numbers and entropy
- Cryptographic RNGs use cipher-like techniques to provide indistinguishability
- But rely on truly random seeding to stop brute force
  - Extreme case: no entropy → always same "randomness"
- Modern best practice: seed pool with 256 bits of entropy
  - Suitable for security levels up to $2^{256}$

Netscape RNG failure
- Early versions of Netscape SSL (1994-1995) seeded with:
  - Time of day
  - Process ID
  - Parent process ID
- Best case entropy only 64 bits
  - (Not out of step with using 40-bit encryption)
- But worse because many bits guessable

Debian/OpenSSL RNG failure (1)
- OpenSSL has pretty good scheme using /dev/urandom
- Also mixed in some uninitialized variable values
  - "Extra variation can't hurt"
- From modern perspective, this was the original sin
  - Remember undefined behavior discussion?
- But had no immediate ill effects

Debian/OpenSSL RNG failure (2)
- Debian maintainer commented out some lines to fix a Valgrind warning
  - "Potential use of uninitialized value"
- Accidentally disabled most entropy (all but 16 bits)
- Brief mailing list discussion didn't lead to understanding
- Broken library used for ~2 years before discovery

Detected RSA/DSA collisions
- 2012: around 1% of the SSL keys on the public net are breakable
  - Some sites share complete keypairs
  - RSA keys with one prime in common (detected by large-scale GCD)
- One likely culprit: insufficient entropy in key generation
  - Embedded devices, Linux /dev/urandom vs. /dev/random
- DSA signature algorithm also very vulnerable
### New factoring problem (CCS'17)
- An Infineon RSA library used primes of the form \( p = k \cdot M + (65537^a \mod M) \).
- Smaller problems: fingerprintable, less entropy.
- Major problem: can factor with a variant of Coppersmith's algorithm.
  - E.g., 3 CPU months for a 1024-bit key.

### Side-channel attacks
- **Timing analysis:**
  - Number of 1 bits in modular exponentiation.
  - Unpadding, MAC checking, error handling.
  - Probe cache state of AES table entries.
- **Power analysis:**
  - Especially useful against smartcards.
- **Fault injection:**
- **Data non-erasure:**
  - Hard disks, "cold boot" on RAM.

### WEP "privacy"
- First WiFi encryption standard: Wired Equivalent Privacy (WEP).
- F&S: designed by a committee that contained no cryptographers.
- Problem 1: note "privacy": what about integrity?
  - Nope: stream cipher + CRC = easy bit flipping.

### WEP shared key
- Single key known by all parties on network.
- Easy to compromise.
- Hard to change.
- Also often disabled by default.
- Example: a previous employer.

### WEP key size and IV size
- Original sizes: 40-bit shared key (export restrictions) plus 24-bit IV = 64-bit RC4 key.
  - Both too small.
- 128-bit upgrade kept 24-bit IV.
  - Vague about how to choose IVs.
  - Least bad: sequential, collision takes hours.
  - Worse: random or everyone starts at zero.

### WEP RC4 related key attacks
- Only true crypto weakness.
- RC4 "key schedule" vulnerable when:
  - RC4 keys very similar (e.g., same key, similar IV).
  - First stream bytes used.
- Not such a problem for other RC4 users like SSL.
  - Key from a hash, skip first output bytes.

### New problem with WPA (CCS'17)
- Session key set up in a 4-message handshake.
- Key reinstallation attack: replay #3.
  - Causes most implementations to reset nonce and replay counter.
  - In turn allowing many other attacks.
  - One especially bad case: reset key to 0.
- Protocol state machine behavior poorly described in spec.
  - Outside the scope of previous security proofs.

### Trustworthiness of primitives
- Classic worry: DES S-boxes.
- Obviously in trouble if cipher chosen by your adversary.
- In a public spec, most worrying are unexplained elements.
- Best practice: choose constants from well-known math, like digits of \( \pi \).
Dual_EC_DRBG (1)

- Pseudorandom generator in NIST standard, based on elliptic curve
- Looks like provable (slow enough) but strangely no proof
- Specification includes long unexplained constants
- Academic researchers find:
  - Some EC parts look good
  - But outputs are statistically distinguishable

Dual_EC_DRBG (2)

- Found 2007: special choice of constants allows prediction attacks
  - Big red flag for paranoid academics
- Significant adoption in products sold to US govt.
  - FIPS-140 standards
  - Semi-plausible rationale from RSA (EMC)
- NSA scenario basically confirmed by Snowden leaks
  - NIST and RSA immediately recommend withdrawal