CSci 5271
Introduction to Computer Security
Day 17: Crypto protocols and “S” protocols
Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline
- Public-key crypto basics, cont'd
- Public key encryption and signatures
- Announcements
- Cryptographic protocols, pt. 1
- Key distribution and PKI
- SSH
- SSL/TLS, DNSSEC

Public key primitives
- Public-key encryption (generalizes block cipher)
  - Separate encryption key EK (public) and decryption key DK (secret)
- Signature scheme (generalizes MAC)
  - Separate signing key SK (secret) and verification key VK (public)

Modular arithmetic
- Fix modulus $n$, keep only remainders mod $n$
  - mod 12: clock face; mod $2^{32}$: unsigned int
- $+, -, \times$ work mostly the same
- Division: see Exercise Set 1
- Exponentiation: efficient by square and multiply

Generators and discrete log
- Modulo a prime $p$, non-zero values and $\times$ have a nice (“group”) structure
- $g$ is a generator if $g^0, g, g^2, g^3, \ldots$ cover all elements
- Easy to compute $x \mapsto g^x$
- Inverse, discrete logarithm, hard for large $p$

Diffie-Hellman key exchange
- Goal: anonymous key exchange
- Public parameters $p, g$; Alice and Bob have resp. secrets $a, b$
  - Alice$\rightarrow$Bob: $A = g^a \pmod p$
  - Bob$\rightarrow$Alice: $B = g^b \pmod p$
  - Alice computes $B^a = g^{ab} = k$
  - Bob computes $A^b = g^{ab} = k$

Relationship to a hard problem
- We’re not sure discrete log is hard (likely not even NP-complete), but it’s been unsolved for a long time
- If discrete log is easy (e.g., in P), DH is insecure
- Converse might not be true: DH might have other problems

Categorizing assumptions
- Math assumptions unavoidable, but can categorize
- E.g., build more complex scheme, shows it’s “as secure” as DH because it has the same underlying assumption
- Commonly “decisional” (DDH) and “computational” (CDH) variants
Key size, elliptic curves
- Need key sizes ~10 times larger than security level
  - Attacks shown up to about 768 bits
- Elliptic curves: objects from higher math with analogous group structure
  - (Only tenuously connected to ellipses)
- Elliptic curve algorithms have smaller keys, about \( 2 \times \) security level

Outline
- Public-key crypto basics, cont'd
- Public key encryption and signatures
- Announcements
- Cryptographic protocols, pt. 1
- Key distribution and PKI
- SSH
- SSL/TLS, DNSSEC

General description
- Public-key encryption (generalizes block cipher)
  - Separate encryption key \( EK \) (public) and decryption key \( DK \) (secret)
- Signature scheme (generalizes MAC)
  - Separate signing key \( SK \) (secret) and verification key \( VK \) (public)

RSA setup
- Choose \( n = pq \), product of two large primes, as modulus
  - \( n \) is public, but \( p \) and \( q \) are secret
- Compute encryption and decryption exponents \( e \) and \( d \) such that
  \[
  M^ed = M \pmod{n}
  \]

RSA encryption
- Public key is \( (n, e) \)
- Encryption of \( M \) is \( C = M^e \pmod{n} \)
- Private key is \( (n, d) \)
- Decryption of \( C \) is \( C^d = M^{ed} = M \pmod{n} \)

RSA signature
- Signing key is \( (n, d) \)
- Signature of \( M \) is \( S = M^d \pmod{n} \)
- Verification key is \( (n, e) \)
- Check signature by \( S^e = M^{de} = M \pmod{n} \)
  - Note: symmetry is a nice feature of RSA, not shared by other systems

RSA and factoring
- We're not sure factoring is hard (likely not even \( NP \)-complete), but it's been unsolved for a long time
- If factoring is easy (e.g., in \( P \)), RSA is insecure
- Converse might not be true: RSA might have other problems

Homomorphism
- Multiply RSA ciphertexts \( \Rightarrow \) multiply plaintexts
- This homomorphism is useful for some interesting applications
- Even more powerful: fully homomorphic encryption (e.g., both \( + \) and \( \times \))
  - First demonstrated in 2009; still very inefficient
Problems with vanilla RSA

- Homomorphism leads to chosen-ciphertext attacks
- If message and $e$ are both small compared to $n$, can compute $M^{1/e}$ over the integers
- Many more complex attacks too

Hybrid encryption

- Public-key operations are slow
- In practice, use them just to set up symmetric session keys
  + Only pay RSA costs at setup time
  - Breaks at either level are fatal

Padding, try #1

- Need to expand message (e.g., AES key) size to match modulus
- PKCS#1 v. 1.5 scheme: prepend 00 01 FF FF .. FF
- Surprising discovery (Bleichenbacher'98): allows adaptive chosen ciphertext attacks on SSL
  - Variants recurred later (c.f. "ROBOT" 2018)

Modern “padding”

- Much more complicated encoding schemes using hashing, random salts, Feistel-like structures, etc.
- Common examples: OAEP for encryption, PSS for signing
- Progress driven largely by improvement in random oracle proofs

Simpler padding alternative

- “Key encapsulation mechanism” (KEM)
- For common case of public-key crypto used for symmetric-key setup
  - Also applies to DH
- Choose RSA message $r$ at random mod $n$, symmetric key is $H(r)$
  - Hard to retrofit, RSA-KEM insecure if $e$ and $r$ reused with different $n$

Post-quantum cryptography

- One thing quantum computers would be good for is breaking crypto
- Square root speedup of general search
  - Countermeasure: double symmetric security level
- Factoring and discrete log become poly-time
  - DH, RSA, DSA, elliptic curves totally broken
  - Totally new primitives needed (lattices, etc.)
- Not a problem yet, but getting ready

Box and locks revisited

- Alice and Bob’s box scheme fails if an intermediary can set up two sets of boxes
  - Compare middleperson attack
- Real world analogue: challenges of protocol design and public key distribution

Outline

Public-key crypto basics, cont’d
Public key encryption and signatures
Announcements
Cryptographic protocols, pt. 1
Key distribution and PKI
SSH
SSL/TLS, DNSSEC
Grading progress

Exercise set 2 grades now available
- Regrade request deadline will be next Wednesday 11/10
- Midterm grades and solutions not ready yet, sorry

Research project meetings

All groups should now have a Google Calendar invitation (for Zoom) or an email invitation

Outline

Public-key crypto basics, cont’d
Public key encryption and signatures
Announcements
Cryptographic protocols, pt. 1
Key distribution and PKI
SSH
SSL/TLS, DNSSEC

A couple more security goals

- Non-repudiation: principal cannot later deny having made a commitment
  - i.e., consider proving fact to a third party
- Forward secrecy: recovering later information does not reveal past information
  - Motivates using Diffie-Hellman to generate fresh keys for each session

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

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

Example: simple authentication

\[ A \rightarrow B : A, \{A, N\}_K_A \]
- E.g., Alice is key fob, Bob is garage door
- Alice proves she possesses the pre-shared key \(K_A\)
  - Without revealing it directly
- Using encryption for authenticity and binding, not secrecy

Nonce

\[ A \rightarrow B : A, \{A, N\}_K_A \]
- N is a nonce: a value chosen to make a message unique
- Best practice: pseudorandom
- In constrained systems, might be a counter or device-unique serial number
Replay attacks

- A nonce is needed to prevent a verbatim replay of a previous message
- Garage door difficulty: remembering previous nonces
  - Particularly: lunchtime/roommate/valet scenario
- Or, door chooses the nonce: challenge-response authentication

Middleperson attacks

- Older name: man-in-the-middle attack, MITM
- Adversary impersonates Alice to Bob and vice-versa, relays messages
- Powerful position for both eavesdropping and modification
- No easy fix if Alice and Bob aren't already related

Chess grandmaster problem

- Variant or dual of middleperson
- Adversary forwards messages to simulate capabilities with his own identity
- How to win at correspondence chess
- Anderson's MiG-in-the-middle

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

Public key authenticity

- Public keys don't need to be secret, but they must be right
- Wrong key can't stop MITM
- 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
  - \( C_A = \text{Sign}_{K_B}(A, K_A) \)
- 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

Public-key crypto basics, cont’d
Public key encryption and signatures
Announcements
Cryptographic protocols, pt. 1
Key distribution and PKI
SSH
SSL/TLS, DNSSEC

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”
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 (cf. HA1)
  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
- Public-key crypto basics, cont'd
- Public key encryption and signatures
- Announcements
- Cryptographic protocols, pt. 1
- Key distribution and PKI
- SSH
- SSL/TLS, DNSSEC

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 11 and 12 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($S \parallel A$), 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
- 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
- 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

DNS: trusted but vulnerable

- Almost every higher-level service interacts with DNS
- UDP protocol with no authentication or crypto
  - Lots of attacks possible
- Problems known for a long time, but challenge to fix compatibly

DNSSEC goals and non-goals

- Authenticity of positive replies
- Authenticity of negative replies
- Integrity
  - Confidentiality
  - Availability

First cut: signatures and certificates

- Each resource record gets an RRSIG signature
  - E.g., A record for one name—address mapping
  - Observe: signature often larger than data
- Signature validation keys in DNSKEY RRs
- Recursive chain up to the root (or other “anchor”)

Add more indirection

- DNS needs to scale to very large flat domains like .com
- Facilitated by having single DS RR in parent indicating delegation
- Chain to root now includes DSes as well

Negative answers

- Also don’t want attackers to spoof non-existence
  - Gratuitous denial of service, force fallback, etc.
- But don’t want to sign “x does not exist” for all x
- Solution 1, NSEC: “there is no name between acacia and baobab”

Preventing zone enumeration

- Many domains would not like people enumerating all their entries
- DNS is public, but “not that public”
- Unfortunately NSEC makes this trivial
- Compromise: NSEC3 uses password-like salt and repeated hash, allows opt-out

DANE: linking TLS to DNSSEC

- “DNS-based Authentication of Named Entities”
- DNS contains hash of TLS cert, don’t need CAs
- How is DNSSEC’s tree of certs better than TLS’s?
Signing the root
- Political problem: many already distrust US-centered nature of DNS infrastructure
- Practical problem: must be very secure with no single point of failure
- Finally accomplished in 2010
  - Solution involves ‘key ceremonies’, international committees, smart cards, safe deposit boxes, etc.

Deployment
- Standard deployment problem: all cost and no benefit to being first mover
- Servers working on it, mostly top-down
- Clients: still less than 20%
- Will probably be common for a while: insecure connection to secure resolver

What about privacy?
- Users increasingly want privacy for their DNS queries as well
- Older DNSCurve and DNSCrypt protocols were not standardized
- More recent “DNS over TLS” and “DNS over HTTPS” are RFCs
- DNS over HTTPS in major browsers might have serious centralization effects