CS640 Lecture 24: 12/4/2014

Outline
- Crypto algorithms
- Session keys
- Public key infrastructure
- Secure protocols

Basic Cryptography
- Encrypt/decrypt algorithm should be
  - Public -- inventing algorithms is hard, so we don’t want to have to develop a new one if something is leaked
  - Easy to compute with the key -- efficient in software and hardware, and on mobile devices which have fewer resources
  - Hard to compute without the key -- computers keep getting more powerful
- Key should be
  - Secret
  - Long -- length of key often determines “level” of security
- Types of functions
  - Cryptographic hash -- no keys
  - Symmetric/secret key -- one shared key
  - Asymmetric/public key -- pair of keys: one public & one private

Cryptographic Hash
- Also known as “cryptographic checksum” -- used to detect if message has been tampered
- Take message $m$ of any length, and produce smaller message $h(m)$
- Properties
  - Pre-image resistance -- hard to find $m$ given $h(m)$; “one-way” function
  - Second pre-image resistance -- given a message $m$, it is hard to find a message $m'$ that hashes to the same $h(m)$
  - Collision resistance -- hard to find any two messages $m$ and $m'$ such that $h(m) = h(m')$
- How many bits for the hash?
  - If hash is $n$ bits, it takes $2^n/2$ tries to find a collision
  - MD5 -- uses 128 bits; weakness known for a while
  - SHA-1 -- uses 160 bits; also not recommended for use
  - SHA-2 -- collection of six different hash functions; use 224, 256, 384 or 512 bits
  - SHA-3 -- emerging standard
- Example: self-certifying names
  - File-sharing software (e.g., BitTorrent) names files with $h($file-data$)$
  - Verify $h($downloaded-data$)$ = name of file

Symmetric/Secret Key
- Sender and receiver share a common key
- None of the original structure of the plaintext should exist in the ciphertext
- Otherwise, attackers could look for patterns -- e.g., commonly used letters in the English language; HTTP request starts with method (GET, POST, etc.)

- **Variants**
  - Data Encryption Standard (DES) -- 64-bit keys (8-bits are parity); easy to recover a key given today's processing power
  - Triple DES (3DES) -- 168-bit keys; encrypt using first 56-bits, decrypt using middle 56-bits, encrypt using last 56-bits (inverse for decryption); slow to implement in software
  - Advanced Encryption Standard (AES) -- 128, 192, or 256-bit keys; fast implementations in hardware or software and low memory footprint

- **Challenge: key distribution**
  - Physically deliver key -- not practical
  - Use existing key to deliver new key -- need unique key pair for each pair of endpoints \((n*(n-1)/2\) total keys for \(n\) endpoints)
  - Use key distribution center (KDC) -- KDC generates session keys and distributes them to pairs of endpoints that wants to communicate; need \(n\) master keys if you have \(n\) endpoints

**Asymmetric/Public Key**

- **Pair of keys**
  - Public key given to many senders
  - Private key kept a secret by the receiver (i.e., owner)
  - The private key cannot be derived from the public key

- Data is encrypted using the public key, but can only be decrypted using the private key

- Data can also be encrypted using the private key and decrypted using the public key
  - Obviously not useful for sending confidential data
  - But, can be used to verify a message came from a specific entity, since only one entity should have the private key and the ability to encrypt a message using that key

- **RSA (named after inventors Rivest Shamir, and Adleman)**
  - Relies on the high computational cost of factoring large numbers
    - Factorization takes \(O(e^{\log n \log \log n})\) operations
    - Exponentiation takes \(O((\log n)^3)\) operations
    - 2048 or 4096-bit keys
  - Key generation
    - Choose two large prime numbers \(p\) and \(q\) (each 1024 or 2048-bits)
    - \(n = p * q\)
    - Choose the encryption key \(e\), such that \(e\) and \((p - 1) * (q - 1)\) are relatively prime -- i.e., they have no common factor greater than 1
    - Compute decryption key: \(d = e^{-1} \mod ((p - 1) * (q - 1))\)
- Construct public key as $<e, n>$ and private key as $<d, n>$
- Discard (do not disclose) original primes $p$ and $q$
  - Basic algorithm
    - public key = $<e, n>$
    - private key = $<d, n>$
    - plaintext = $m$
    - ciphertext = $c$
    - encryption: $c = m^e \mod n$
    - decryption: $m = c^d \mod n$
- Much slower than symmetric key, but key distribution is easier

**Session Keys**
- Symmetric key used for a single session
- Generated and shared between communicating parties at runtime
- Benefits
  - Allows for faster (symmetric key) encryption without needing to pre-distribute a symmetric key
  - Amount of data encrypted with key is limited, so it’s harder for an attacker to crack the encryption
- Two options for establishing sessions keys:
  - Use asymmetric key encryption to create a secure channel over which a session key can be sent; after both parties know the key, then switch to symmetric key encryption
  - Diffie-Hellman key agreement
    - Two parameters which are public and can be used by everyone
      - $p$ = prime number
      - $g$ = primitive root of $p$ (for every number $1 \leq n \leq p - 1$, there must be some value $k$ s.t. $n = g^k \mod p$)
    - Both parties generate their own private random number ($a$ and $b$, respectively)
    - First party sends $g^a \mod p$ to second party; second party can compute $g^{ab} \mod p$
    - Second party sends $g^b \mod p$ to first party; first party can compute $g^{ba} \mod p$
    - $g^{ab} \mod p = g^{ba} \mod p$, so this is the session key
    - It is hard to determine $a$ or $b$ from $g^a \mod p$ or $g^b \mod p$, so the key is secret

**Public Key Infrastructure**
- Mechanism for certifying bindings between public keys and identities
  - Identities could be email address, domain name, etc.
  - Binding ensured using a digital signature
- Public key certificate
  - Identity of entity being certified
  - Public key of the entity being certified
  - Identity of the signer
• Digital signature
  ■ Compute digest of identifies and public key using cryptographic hash
  ■ Encrypt digest using signer’s private key
  ■ Anyone with signer’s public key can decrypt the digest and compare it against a digest they compute
  ○ Identifier for digital signature algorithm -- Elliptic Curve Digital Signature Algorithm (ECDSA) is the current standard
  ○ Expiration timestamp
• Chain of trust -- e.g., X certifies public key for Y, and Y certifies public key for Z, then there is a chain of certificates from X to Z
• Certificate Authority (CA)
  ○ Verifies identities and issues public key certificates
  ○ Trusted organization (e.g., VeriSign)
  ○ Web browsers preconfigured with certificates for common CAs
  ○ Publish certificate revocation list -- digitally signed list of certificates that have been voided

Secure Protocols
• Pretty Good Privacy (PGP)
  ○ Provides email security
  ○ Only intended recipient can read the email
    ■ Sender encrypts message with receiver’s public key; only receiver can decrypt with their private key
  ○ Sender and receiver are who they say they are
    ■ Obtain certificate for sender or receiver’s public key
• Secure Shell (SSH)
  ○ Runs atop TCP
  ○ Client authenticates the server using RSA -- accept server’s public key the first time you connect, or obtain server’s public key out of band
  ○ Establish session key
  ○ Authenticate user
    ■ Password
    ■ Public-key -- user provides public key to server a priori
    ■ Host-based -- authenticate client host, and assume user on client is the same user on the server
• Transport Layer Security (TLS)
  ○ Runs between TCP and application protocol (e.g., HTTP)
○ TLS handshake

- New random value $p$; supported ciphers
- New random value $q$; digital certificate with public key
- Pre-secret, encrypted using servers’ public key

- Compute shared secret key, using pre-secret and $p$
- Switch to symmetric key; use shared secret key

- Compute shared secret key, using pre-secret and $q$
- Switch to symmetric key; use shared secret key

○ Messages from application layer are:
  - Fragmented or coalesced into blocks
  - Integrity-protected by computing cryptographic hash of message + secret value + sequence number, and appending hash to message
    - Including sequence number in hash prevents replay and reordering attacks
  - Encrypt message and hash using symmetric key encryption with shared secret key
  - Pass to the transport layer