Security: Cryptography

Lecture 38
Some High-Level Goals

☐ Confidentiality
   ■ Non-authorized users have limited access

☐ Integrity
   ■ Accuracy/correctness/validity of data

☐ Availability
   ■ No down-time or disruptions
   ■ (We’re looking at you, Carmen!)

☐ Authenticity
   ■ Agents are who they claim to be

☐ Non-repudiation
   ■ A party to a transaction can not later deny their participation
Methods of Attack

- **Target people** (“social engineering”)
  - Phishing: email, phone, surveys, ...
  - Baiting: click & install, physical media, ...

- **Target software** (“exploits”)
  - Unpatched OS, browser, programs
  - Buffer overflow
  - Code injection and cross-site scripting

- **Target channel** (“man-in-the-middle”)
  - Eavesdropping
  - Masquerading, tampering, replay
Cryptography

- Etymology (Greek)
  - kryptos: hidden or secret
  - grapho: write

- Basic problem:
  - 2 agents (traditionally “Alice” and “Bob”)
  - A & B want to exchange private messages
  - Channel between A & B is not secure (“Eve” is eavesdropping)

- Solution has other applications too
  - Protect stored data (e.g. on disk, or in cloud)
  - Digital signatures for non-repudiation
  - Secure passwords for authentication
Core Idea: The Secret

- Alice & Bob share some secret
  - Secret can not be the message itself
  - Secret used to protect arbitrary messages

- Crude analogy: a padlock
  - Copies of the physical key are the secret
  - Alice puts message in box and locks it
  - Bob unlocks box and reads message

- But real channels are bit streams
  - Eve can see the bits!
  - Message must be garbled in some way
  - Secret is strategy for garbling/degarbling
Protecting Messages

- Alice garbles (encrypts) the message
- Sends the encrypted cipher-text
- Bob knows how to degarble (decrypt) cipher-text back into plain-text

Image: www.devx.com
Encryption/Decryption Function

E: P → Q
D: Q → P

E(m) = c
D(c) = m
i.e. D = E⁻¹

"hello"
m
Secret
E
D
c
“fwspdaad”

Plaintext messages (P)

Ciphertext Messages (Q)
Families of Encryption Functions

- Each pair of agents needs their own E
  - Many E’s (& corresponding D’s) needed
- But good E’s are hard to invent
- Solution: design one (good) E, which is parameterized by a number
  - That is, have a huge family of E’s: \( E_0, E_1, E_2, \ldots, E_K \)
  - Secret: which \( E_i \) is used (\( i \) is the “key”)

Computer Science and Engineering ▪ The Ohio State University
Classic Example: Caesar Cipher

- Shift letter by x positions (in alphabet)
  - For m=“hello world”, E₃(m)=“khoor zruog”
  - 26 possible ciphers

- Generalization: arbitrary mapping
  - E.g. a→s, b→n, c→v, d→f, e→r, ...
  - For m=“hello world”, E(m) = “jraap eptaf”
  - 26! possible ciphers... that’s a lot!

- Weakness: Frequency Analysis
  - In English text, letters appear in predictable ratios
  - From enough cipher-text, can infer E
Frequency Fingerprint

<table>
<thead>
<tr>
<th>Frequency</th>
<th>E</th>
<th>T</th>
<th>A</th>
<th>O</th>
<th>I</th>
<th>N</th>
<th>S</th>
<th>R</th>
<th>H</th>
<th>D</th>
<th>L</th>
<th>U</th>
<th>C</th>
<th>M</th>
<th>F</th>
<th>Y</th>
<th>W</th>
<th>G</th>
<th>P</th>
<th>B</th>
<th>V</th>
<th>K</th>
<th>X</th>
<th>Q</th>
<th>J</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12.02</td>
<td>9.10</td>
<td>8.12</td>
<td>7.68</td>
<td>7.31</td>
<td>6.95</td>
<td>6.28</td>
<td>6.02</td>
<td>5.92</td>
<td>4.32</td>
<td>3.98</td>
<td>2.88</td>
<td>2.71</td>
<td>2.61</td>
<td>2.30</td>
<td>2.11</td>
<td>2.09</td>
<td>2.03</td>
<td>1.82</td>
<td>1.49</td>
<td>1.11</td>
<td>0.69</td>
<td>0.17</td>
<td>0.11</td>
<td>0.10</td>
<td>0.07</td>
</tr>
</tbody>
</table>
Polyalphabetic Cipher

- Alberti’s idea: Use different $E_i$’s within the same message
  - $E(“hello world”) = E_a(“h”)E_b(“e”)E_c(“l”)E_d(“l”)E_e(“o”)…$

- Alice & Bob need to agree on the sequence of E’s to use

- Claude Shannon proved that this method is perfectly secure (1949)
  - Precise information-theoretic meaning
  - A “one-time pad”
One-Time Pad

- Message is a sequence of bits
  \[ m_0 \ m_1 \ m_2 \ m_3 \ m_4 \ m_5 \ m_6 \ldots \]

- One-time pad is *random* bit sequence
  \[ x_0 \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ldots \]

- E is XOR operation, bit-wise

- Cipher text is
  \[ m_0 + x_0 \ m_1 + x_1 \ m_2 + x_2 \ m_3 + x_3 \ m_4 + x_4 \ m_5 + x_5 \ m_6 + x_6 \ldots \]

- Problem: Pad is long and cannot be re-used (hence cumbersome to share)

- “Solution”: pseudo-random sequence, generated from a seed (the key)
# Comparison: Stream vs Block

<table>
<thead>
<tr>
<th>Stream Cipher</th>
<th>Block Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø Encrypts bit-by-bit</td>
<td>Ø Encrypts a fixed-length ((k\text{-bit})) sequence</td>
</tr>
<tr>
<td>Ø (</td>
<td>P</td>
</tr>
<tr>
<td>Ø Few choices for (E) (roughly 2)</td>
<td>Ø Many choices for (E) (roughly (2^k!))</td>
</tr>
<tr>
<td>Ø Message can have any length</td>
<td>Ø Padding added s.t. (</td>
</tr>
</tbody>
</table>
AES

- Advanced Encryption Standard (2001)
  - Replaced DES (1977)
- Block size always 128 bits (4x4 bytes)
- Key size is 128, 192, or 256 bits
- Multi-step algorithm, many rounds
Symmetric Key

- For ciphers (so far): Knowing E is enough to figure out D (its inverse)
  - If you know how to encrypt, you can decrypt too
  - Called “symmetric key”
- Example: Caesar cipher
  - If \( E(m) = m + 3 \), \( D(m) = m - 3 \)
- Example: One-time pad
  - Use same pad and same operation (xor)
- Example: AES
  - Use same key, reverse rounds and steps
One-Way Functions

- For some functions, the inverse is hard to calculate
  - One direction (P→Q) is easy, but opposite direction (Q→P) is hard/expensive/slow

- Intuition:
  - Given a puzzle solution, easy to design a puzzle with that solution (the “forward” direction)
  - Given the puzzle, hard to come up with the solution (the “inverse” direction)
Example: Vertex Cover

- Hard direction: Find a vertex cover of size at most 6 in the following graph.

A Map of the Town of Iceberg
Example: Vertex Cover

- Easy direction: Create a graph with a 6-vertex cover from this forest...
Example: Factoring

- Multiplying numbers is easy (i.e. fast)
  - Can multiply 2 $n$-bit numbers in $n^2$ steps

- Factoring a number is hard (i.e. slow)
  - To factor an $n$-bit number, need $2^n$ steps (approximately the number’s value)

- Aside:
  - Primality testing is fast (recall lab activity in Software I and Fermat’s Little Theorem)
  - But this fast test doesn’t reveal the factors of a composite number
Cryptographic Hash Functions

- A hash function maps values to $\mathbb{Z}_B$
  - Every message, regardless of its length, maps to a number in the range $0..B - 1$
  - Result called a digest (constant-length, $\lg B$)
  - Good hashes give uniform distribution: small diff in message $\rightarrow$ big diff in digest

- Cryptographic hash func’s are one-way
  - Given a digest, computationally infeasible to find any message that hashes to it
  - Collisions must still exist ($B < |\text{messages}|$), but are infeasible to find for large enough $B$
  - Digest = a (small, fixed-size) fingerprint
Fixed-Length Digests

**cleartext**

- hello, world

- this is cleartext that anybody can easily read without the key used by encryption. It's also bigger than the box of text above.

- This is some really long text that we mean to encrypt, and to keep these pearls of wisdom out of the reach of the bad guy.

- We don't really know how anybody could ever break our rot13 encryption, but if the NSA puts its mind to it, perhaps they will manage.

- It's not an easy job making up random text for examples.

**MD5 digest**

- hash function
  - 22c3683b094136c3390391ae71b20f04

- hash function
  - bd18d50263b01456f22e3ff0d003bf66

- hash function
  - 267ed3138dacc48e3ac340bade7003d3e
# Crypto. Hash as Fingerprint

<table>
<thead>
<tr>
<th>Input</th>
<th>Digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>DFCD 3454 BBEA 788A 751A</td>
</tr>
<tr>
<td></td>
<td>696C 24D9 7009 CA99 2D17</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>0066 46BB FB7D CBE2 823C</td>
</tr>
<tr>
<td></td>
<td>ACC7 6CD1 90B1 EE6E 3ABC</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>6FD8 7558 7851 4F32 D1C6</td>
</tr>
<tr>
<td></td>
<td>76B1 79A9 0DA4 AEFE 4819</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>FCD3 7FDB 5AF2 C6FF 915F</td>
</tr>
<tr>
<td></td>
<td>D401 C0A9 7D9A 46AF FB45</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>8ACA D682 D588 4C75 4BF4</td>
</tr>
<tr>
<td></td>
<td>1799 7D88 BCF8 92B9 6A6C</td>
</tr>
</tbody>
</table>
Common Cryptographic Hashes

- **MD5**
  - Flaws discovered, now considered “cryptographically broken”
  - Do not use!

- **SHA-1**
  - Common but deprecated: Windows, Chrome, Firefox will reject in 2017
  - 160-bit digests (i.e. 40 hex digits)

- **Replaced by SHA-2**
  - A family of 6 different hash functions
  - Digest sizes: 224, 256, 384, or 512 bits
Utility of Crypto. Hashes

- **Integrity verification (super-checksum)**
  - File download, check digest matches

- **Password protection**
  - Server stores the *hash* of user’s password
  - Check an entered password by computing its hash and comparing to stored value
  - If server is compromised, intruder finds hashes but not passwords

**Note:** Always use “salt”

- Unique-to-user text added to password before hashing
- Prevents intruder from using table of pre-computing hashes for common passwords:
  - See [7df4aec92e54330cb7179a7e315af92f](#)
One-Way Function with Trapdoor

- Function *appears* to be one-way
  - But, in reality, the inverse is easy if one knows a secret (the “trapdoor”)
- There are now 2 different secrets:
  - The one-way-seeming function, E
  - The trapdoor for its inverse, D
- Knowing E is *not enough* to infer D
- Creates an asymmetry:
  - Alice knows only E
  - Bob knows both E and D
Public-Key Encryption

- Algorithms for E and D known by all
  - But parameterized by matched keys
- Asymmetry
  - Key for Bob’s E is *public*
  - Key for Bob’s D is *private*
- Anyone can encrypt messages for Bob
- Only Bob can decrypt these messages
- Important consequences
  - Each agent needs only 1 public key
  - No pre-existing shared secret needed
Public and Private Keys

Alice

Bob

Hello Bob!

Encrypt

7AG7680191B02FN3

Bob’s public key

Decrypt

Hello Bob!

Bob’s private key
RSA

- E and D are actually the same function $m^k \mod n$
  - Parameterized by pair $(k, n)$, i.e. the key
- Private key: $(d, n)$
  - $D(m) = m^d \mod n$
- Public key: $(e, n)$
  - $E(m) = m^e \mod n$
- Choice of $e$ & $d$ is based on factoring
  - Choose 2 large prime numbers, $p$ and $q$
  - Calculate their product, $n = pq$
  - Pick any $d$ relatively prime with $(p-1)(q-1)$
  - Find an $e$ s.t. $ed = 1 \mod (p-1)(q-1)$
Digital Signature

- Usual direction for encryption:
  \[ D(E(m)) = (m^e)^d \mod n = m^{ed} \mod n = m \]

- One-to-one, so backwards works too!
  \[ E(D(m)) = (m^d)^e \mod n = m^{de} \mod n = m \]

- Consider:
  - Bob “encrypts” \( m \) using his \textbf{private} key, \( d \)
  - Bob sends \textbf{both} \( m \) and \( D(m) \)
  - Anyone can undo the “encrypted” part using Bob’s \textbf{public} key, \( e \)
  - Result will be \( m \)

- \( D(m) \) serves as a digital \textbf{signature} of \( m \)
  - Only Bob could have created this signature
  - Use: non-repudiation
Performance Considerations

- Symmetric key algorithms are faster than public key algorithms

- Optimization for encryption (RSA)
  - Create a fresh symmetric key, $k$
  - Use symmetric algorithm to encrypt $m$
  - Use recipient’s public key to encrypt $k$

- Optimization for digital signatures
  - Calculate the digest for $m$ (always short)
  - Use sender’s private key to encrypt digest
Summary

- **Symmetric-key encryption**
  - Sender and receive share secret key
  - Stream ciphers work one bit at a time (e.g., one-time pad)
  - Block ciphers work on larger blocks of bits (e.g., SHA-2)

- **One-way functions: Hard to invert**
  - Cryptographic hash produces fixed-size digest
  - Digest serves as a fingerprint

- **Public key encryption**
  - Matching keys: $k_{\text{private}}$, $k_{\text{public}}$
  - Anyone can use public key to encrypt
  - Only holder of private key can decrypt
  - Use private key to create a digital signature