Security: Cryptography

Lecture 38
Comparison: Stream vs Block

Stream Cipher
- Encrypts bit-by-bit
- $|P| = |Q| = 2$
- Few choices for E (roughly 2)
- Message can have any length

Block Cipher
- Encrypts a fixed-length ($k$-bit) sequence
- $|P| = |Q| = 2^k$
- Many choices for E (roughly $2^k!$)
- Padding added s.t. $|m| \mod k = 0$
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
  - Known as a *symmetric key* cipher
- 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: Dominating Set

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

A Map of the Town of Iceberg
Example: Dominating Set

- Easy direction: Create a graph with a dominating set of size 6 from this forest...

The Secret Solution
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 ($B \ll |\text{messages}|$), 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 those 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**

- `22c3683b094136c339b391ae71b20f04`
- `bd18d50263b01456f22e3ff0d003bf56`
- `dd7ed8f6dacc48eeac348bade70d39ee`
## Crypto. Hash as Fingerprint

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th><strong>Digest</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>DFCD 3454 BEEA 788A 751A 696C 24D9 7009 CA99 2D17</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>0066 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>6FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45</td>
</tr>
<tr>
<td>The red fox jumps over the blue dog</td>
<td>8ACA D682 D588 4C75 4BF4 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 now reject (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 the stored value
  - If server is compromised, intruder finds hashes but not passwords

- Problem:
  - See [md5decrypt.net/en/Sha256/](https://md5decrypt.net/en/Sha256/)
c023d5796452ad1d80263a05d11dc2a42b8c19c5d7c88c0e84ae3731b73a3d34
Role of Salt

- **Danger:**
  - Intruder pre-computes hashes for many (common) passwords: aka a *rainbow table*
  - Scan stolen hashes for matches

- **Solution:** *salt*
  - Server prepends text to password before hashing
  - Text must be *unique* to user
  - Text does not *need* to be secret
    - Ok: Deterministic value based on user name
    - Better: Random value, stored in the table

- Protects the fingerprint, by making it not mass pre-computable
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
  - Hello Bob!
  - Encrypt
  - 7AG7680191B02FN3
  - Decrypt
  - Hello Bob!

- Bob
  - Bob’s public key
  - 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:
  \[ D(E(m)) = (m^e)^d = m^{ed} = \mod m \]

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

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

- \( D(m) \) serves as a digital **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: $k_{private}$, $k_{public}$
  - Anyone can use public key to encrypt
  - Only holder of private key can decrypt
  - Use private key to create a digital signature