CSE331 Computer Security Fundamentals
Crash course on cryptography

Nick Nikiforakis
nick@cs.stonybrook.edu
Purpose of this slide deck

• This slide deck is meant to give you a high level understanding of cryptographic ideas and the appropriate jargon
  – We need to understand the basics of symmetric and asymmetric cryptography to appreciate HTTPS (HTTP over SSL/TLS)

• This is, by no means, a comprehensive examination of cryptography
Cryptography

• Cryptography comes from the Greek words «κρυπτό» (secret) and «γραφή» (writing)
  – Mangling information in a way that allows unmangling by the intended recipients

• Common Uses of Cryptography:
  – Protect the confidentiality of information in transit (or in storage)
  – Protect the integrity of information (even if that is public)
  – Authenticate a user
  – Non-repudiation: A sender cannot send a message and then deny it
Typical use

Cleartext message, e.g. “Attack at dawn”

Mangled message, e.g. “asd234fdshixcvxc vasd”
Cryptography

• A typical cryptographic system involves an algorithm and one (or more) keys
  – If a key is compromised you just change the key, you don’t have to come up with a new algorithm

• Kerckhoffs’ Principle: “A cryptosystem should be secure even if everything about the system, except the key, is public knowledge”
  – The security does not depend on the attacker not knowing which algorithm we are using
Types

• Types of cryptographic functions:
  – Symmetric Key Cryptography (also known as Secret Key Cryptography)
  – Asymmetric Key Cryptography (also known as Public Key Cryptography)
  – Hash functions
    • We looked at some characteristics of those when we talked about authentication and storing passwords
Symmetric Key Cryptography

• In symmetric key cryptography, we only have one key. The same key is used to encrypt the message and to decrypt it
  – Analogous to real-world locks and locking/unlocking
Basic Problem

Given: both parties already know the same secret

Goal: send a message confidentially

Any communication system that aims to guarantee confidentiality must solve this problem.
Symmetric Key Cryptography

• This kind of cryptography is the oldest kind of crypto and it was the only kind we had until very recently (less than 50 years)

• One of the oldest examples of symmetric key cryptography is Caesar’s cipher
  – Caesar’s cipher belongs to the “historical ciphers” in the sense that it is interesting to study, but it is hopelessly broken
Cæsar cipher

Letter shifting cipher (A=>D, B=>E, C=>F, ...
5-tuple

\[ \mathcal{M} = \{ \text{all sequences of letters} \} \]

\[ \mathcal{K} = \{ i \mid i \text{ is an integer and } 0 \leq i \leq 25 \} \]

\[ \mathcal{E} = \{ E_k \mid k \in \mathcal{K} \text{ and for all letters } m, \]
\[ E_k(m) = (m + k) \mod 26 \} \]

\[ \mathcal{D} = \{ D_k \mid k \in \mathcal{K} \text{ and for all letters } c, \]
\[ D_k(c) = (26 + c - k) \mod 26 \} \]

\[ \mathcal{C} = \mathcal{M} \]

History: Cæsar’s key was 3.
Cæsar cipher

Plaintext is HELLO WORLD

Change each letter to the third letter following it (X goes to A, Y to B, Z to C)

– Key is 3, usually written as letter ‘D’

Ciphertext is KHOOR ZRUOG
Cryptanalysis of Cæsar Cipher

Brute force attack sufficient:
– Since the keyspace is small (26 possible keys), try all possible keys until you find the right one.

<table>
<thead>
<tr>
<th>Decryption key (26-K)</th>
<th>Candidate plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>exxegoexsrgi</td>
</tr>
<tr>
<td>1</td>
<td>dwwdfndwrqfh</td>
</tr>
<tr>
<td>2</td>
<td>cvvcemcvqpeg</td>
</tr>
<tr>
<td>3</td>
<td>buubdlbupodf</td>
</tr>
<tr>
<td>4</td>
<td><strong>attackatonce</strong></td>
</tr>
<tr>
<td>5</td>
<td>zsszbjzsnmbd</td>
</tr>
<tr>
<td>6</td>
<td>yrraiyrmlac</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>23</td>
<td>haahjrhavujl</td>
</tr>
<tr>
<td>24</td>
<td>gzzgiqgzutik</td>
</tr>
<tr>
<td>25</td>
<td>fyyfhpftyshj</td>
</tr>
</tbody>
</table>
General Simple Substitution Cipher

Key Space: All permutations of alphabet (26! keys)

Encryption:

Replace each plaintext letter \( x \) with \( K(x) \)

Decryption:

Replace each ciphertext letter \( y \) with \( K^{-1}(y) \)

Example:

\[
\begin{array}{cccccccccccccccccccccccc}
\end{array}
\]

CRYPTO \( \rightarrow \) BQCOWP
General Simple Substitution Cryptanalysis

• Exhaustive search impossible
  – Key space size is $26! \approx 4 \times 10^{26}$
  – Historically thought to be unbreakable.

• However, languages have different frequencies of
  – Letters
  – digraphs (groups of 2 letters)
  – trigraphs (groups of 3 letters)
  – etc.

Simple substitution ciphers preserve letter frequencies.
English Letter Frequencies

The chart shows the frequency of each letter in the English language. The frequencies are represented on a vertical scale, with 'a' being the most frequent and 'z' being the least frequent.
Additional Frequency Features

• Digraph frequencies
  – Common digraphs: EN, RE, ER, NT

• Vowels other than E rarely followed by another vowel.

• The letter Q is followed only by U.

• ...

Countering Frequency Analysis

Nulls
– Insert additional symbols (numbers) which have no meaning in random places.

Idiosyncratic spellings
– n0rM4L s34rCh

Homophonic substitution
– Each letter has multiple substitutions.

Techniques increase difficulty but don’t make impossible.
Countering Frequency Analysis

Primary weakness of simple substitution:

- Each ciphertext letter corresponds to only one letter of plaintext.

Solution: *polyalphabetic* substitution

- Use multiple cipher alphabets.
- Switch between cipher alphabets from character to character in the plaintext.
Vigènere Cipher

Use phrase instead of letter as key.

Example

– Message THE BOY HAS THE BALL
– Key VIG
– Encipher using Cæsar cipher for each letter:

key VIGVIGVIGVIGVIGVIGVIGVIGVIGVIGVIGVIGVIGVIGVIG
plain THEBOYHASTHEBALL
cipher OPKWWECIYOPKWIRG
Vigènere Cryptanalysis

1. Find key length (period), which we will call $n$.
2. Break message into $n$ parts, each part being enciphered using the same key letter.
3. Use frequency analysis to solve resulting $n$ simple substitution ciphers.

<table>
<thead>
<tr>
<th>key</th>
<th>VIGVIGVIGVIGVIGVIGV</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>THEBOYHASTHEBALL</td>
</tr>
<tr>
<td>cipher</td>
<td>OPKWWECIYOPKWIRG</td>
</tr>
</tbody>
</table>
One-Time Pad (Vernam Cipher)

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
\[ \text{ciphertext} = \text{plaintext} \oplus \text{key} \]

Decrypt by bitwise XOR of ciphertext and key:
\[ \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \]

Cipher achieves **perfect secrecy** if and only if there are as many possible keys as possible plaintexts, and every key is equally likely  
(Claude Shannon, 1949)
One-Time Pad

• A Vigenère cipher with a random key at least as long as the message.
• Provably unbreakable.
• Example ciphertext: DXQR.
• Equally likely to correspond to
  – plaintext DOIT (key AJIY)
  – plaintext DONT (key AJDY)
  – and any other 4 letters.
Advantages of One-Time Pad

• Easy to compute
  – Encryption and decryption are the same operation
  – Bitwise XOR is very cheap to compute

• As secure as theoretically possible
  – Given a ciphertext, all plaintexts are equally likely, regardless of attacker’s computational resources
  – ...if and only if the key sequence is truly random
    • True randomness is expensive to obtain in large quantities
  – ...if and only if each key is as long as the plaintext
    • But how do the sender and the receiver communicate the key to each other? Where do they store the key?
Problems with One-Time Pad

• Key must be as long as the plaintext
  – Impractical in most realistic scenarios
  – Was used for diplomatic and intelligence traffic

• Does not guarantee integrity
  – One-time pad only guarantees confidentiality
  – Attacker cannot recover plaintext, but can easily change it to something else

• Insecure if keys are reused
  – Attacker can obtain XOR of plaintexts
Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
ciphertext = plaintext ⊕ key

Decrypt by bitwise XOR of ciphertext and key:
ciphertext ⊕ key = (plaintext ⊕ key) ⊕ key = plaintext ⊕ (key ⊕ key) = plaintext

No Integrity
**Dangers of Reuse**

Learn relationship between plaintexts

\[ C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = \]

\[ (P_1 \oplus P_2) \oplus (K \oplus K) = P_1 \oplus P_2 \]
Types of symmetric ciphers

• **Block Ciphers**
  – Process one block at a time
  – Substitution and transposition (permutation) techniques
  – Examples: *DES*, *AES*, ...

• **Stream Ciphers**
  – Process one bit or byte at a time
  – Inspired by One-Time Pad
  – Plaintext is combined (XOR) with a *pseudorandom* keystream (*NOT the same as one-time pad*)
  – Synchronous (key is not dependent on output stream) vs. asynchronous (key is dependent on output stream)
  – Examples: *RC4*, *any block cipher in OFB or CTR mode*, ...
Stream Ciphers vs. Block Ciphers

1 bit at a time

b bits at a time

*Img source: Understanding Cryptography, Paar, C.; Pelzl, J.*
# Block ciphers

## Multiple rounds of substitution, permutation, ...

—*Confusion*: each character of the ciphertext should depend on several parts of the key

—*Diffusion*: changing a plaintext character should result in several changed ciphertext characters

<table>
<thead>
<tr>
<th></th>
<th>DES</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length</td>
<td>56 bits</td>
<td>128, 192, 256 bits</td>
</tr>
<tr>
<td>Block size</td>
<td>64 bits</td>
<td>128 bits</td>
</tr>
<tr>
<td>Rounds</td>
<td>16</td>
<td>10, 12, 14</td>
</tr>
<tr>
<td>Construction</td>
<td>Substitution, permutation</td>
<td>Substitution, permutation, mixing, addition</td>
</tr>
<tr>
<td>Developed</td>
<td>1977</td>
<td>1998</td>
</tr>
<tr>
<td>Status</td>
<td>Broken!</td>
<td>OK (for now)</td>
</tr>
</tbody>
</table>

**DES rounds**

Plaintext

<table>
<thead>
<tr>
<th>L₀</th>
<th>R₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>K₀</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td>K₁</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Kᵦ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Rₙ₊₁</td>
<td>Lₙ₊₁</td>
</tr>
</tbody>
</table>

Ciphertext
Encrypting a Large Message

• So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size

• **Electronic Code Book (ECB) mode**
  – Split plaintext into blocks, encrypt each one separately using the block cipher

• **Cipher Block Chaining (CBC) mode**
  – Split plaintext into blocks, XOR each block with the result of encrypting previous blocks

• Also various counter modes, feedback modes, etc.
ECB Mode

- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks
Information Leakage in ECB Mode

[Wikipedia]

Encrypt in ECB mode
CBC Mode: Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

<table>
<thead>
<tr>
<th>Initialization vector (random)</th>
<th>plaintext</th>
<th>block cipher</th>
<th>block cipher</th>
<th>block cipher</th>
<th>block cipher</th>
<th>ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CBC Mode: Decryption

plaintext

Initialization vector

+ key decrypt + key decrypt + key decrypt

+ key decrypt + key decrypt + key decrypt

ciphertext
ECB vs. CBC

[Picture due to Bart Preneel]

AES in ECB mode

AES in CBC mode

Similar plaintext blocks produce similar ciphertext blocks (not good!)
DIY

$ perl -e 'print "A"x200,"B"x200,"C"x200,"D"x200 ' > abcd.txt
$ openssl aes-256-ecb -in abcd.txt -out abcd.enc.ecb
$ openssl aes-256-cbc -in abcd.txt -out abcd.enc.cbc

xxd is command-line hex viewer
$ xxd abcd.enc.ecb
$ xxd abcd.enc.cbc
All modes are beautiful not you and deserve to be used
Why do we need Public Key Cryptography?

Classical cryptography session:
1. Alice and Bob agree on algorithm.
2. Alice and Bob agree on key.
3. Alice encrypts her message with agreed upon algorithm and key.
4. Alice sends ciphertext message to Bob.
5. Bob decrypts ciphertext with same algorithm and key as Alice used.

How do we transmit the key?
Public-Key Cryptography

Given: Everybody knows Bob’s public key - How is this achieved in practice?
Only Bob knows the corresponding private key

Goals: 1. Alice wants to send a message that only Bob can read
2. Bob wants to send a message that only Bob could have written
Applications of Public-Key Crypto

• Encryption for confidentiality
  – Anyone can encrypt a message
    • With symmetric crypto, must know the secret key to encrypt
  – Only someone who knows the private key can decrypt
  – Secret keys are only stored in one place

• Digital signatures for authentication
  – Only someone who knows the private key can sign
  – Everyone who knows the public key can validate the signature

• Session key establishment
  – Exchange messages to create a secret session key
  – Then switch to symmetric cryptography (why?)
Public-Key Encryption

- **Key generation**: computationally easy to generate a pair (public key PK, private key SK)
- **Encryption**: given plaintext M and public key PK, easy to compute ciphertext \( C = E_{PK}(M) \)
- **Decryption**: given ciphertext \( C = E_{PK}(M) \) and private key SK, easy to compute plaintext M
  - Infeasible to learn anything about M from C without SK
  - **Trapdoor function**: Decrypt(SK,Encrypt(PK,M))=M
    - Trapdoor function: Easy to compute in one direction, yet difficult to compute in the opposite direction (finding the inverse function) without some secret information
Popular Algorithms

• Diffie-hellman
  – Based on the difficulty of the discrete logarithm problem
    • given $g^x \mod p$, it’s hard to extract $x$

• RSA cryptosystem
  – Based on the difficulty of factoring large integer numbers
    • Given an integer $x$ that is 50 digits long what are the prime numbers $p$ and $q$ where $p*q = x$?
Advantages of Public-Key Crypto

• Confidentiality without shared secrets
  – Very useful in open environments
  – Can use this for key establishment, avoiding the “chicken-or-egg” problem
    • With symmetric crypto, two parties must share a secret before they can exchange secret messages

• Authentication without shared secrets

• Encryption keys are public, but we must be sure that Alice’s public key is really her public key
  – This is a hard problem... Often solved using public-key certificates
Disadvantages of Public-Key Crypto

- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - SSL, IPsec, most other systems based on public crypto
- Keys are longer
  - 2048 bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - Factoring and discrete logarithm problem
Digital signatures

• In contrast with symmetric crypto, asymmetric crypto allows an important property for digital signatures: **non-repudiation**
  – The signer of message (e.g. Alice) cannot later claim that she didn’t sign a message
  – In symmetric encryption, the signer can later claim that she never signed a message and that the recipient instead forged her signature (since the signature is based on a common secret key)
Digital Signatures: Basic Idea

**Given:** Everybody knows Bob’s public key. Only Bob knows the corresponding private key.

**Goal:** Bob sends a “digitally signed” message.
1. To compute a signature, must know the private key.
2. To verify a signature, only the public key is needed.
Questions?
Credits

• One-time pad slides from by Vitaly Shmatikov
• Slides on Caesar’s cipher from James Walden
• Block cipher slides from Michalis Polychronakis’ “Network Security” course