Never explore vulnerabilities in someone's else's system without their permission!

- ... even if they are easy/obvious
- ... even if you mean no harm.
- At best it is rude; Usually it is harmful.
- It is almost always illegal. Trouble with the University as well.

Trying out a vulnerability on your VM is okay.
Brief Pause: Computer Security Ethics

- If you do find a novel vulnerability, do not make it public!
  - ... even if it is easy/obvious
  - ... even if you mean no harm.
  - It is almost always illegal.
  - Legal gray area: Selling it... please don't.

"Responsible disclosure" is the term of art for

- Privately notifying the vendor and possibly victims,
- Filing for a CVE,
- Waiting until it is patched to discuss your finding.

- Sometimes conflicts arise (e.g. vendor won’t fix).
The Wi-Fi network "Pat'swifi" requires a WPA2 password.

Password: [password field]

- Show password
- Remember this network

[Cancel] [Join]
Can you please come over asap to help me move the couch?
I need to be out of here by 3pm
I guess you forgot your phone at home or something
What is Cryptography?

Cryptography involves algorithms with security goals.

Cryptography involves using math to stop adversaries.
Common Security Goal: Secure Channel

Confidentiality: Adversary does not learn anything about messages $m_1, m_2$

Authenticity: $m'_1 = m_1$ and $m'_2 = m_2$
WPA2 (Wi-Fi Protected Access 2): Secure WiFi

pw="fourwordsuppercas"
GSM Cell Phone Encryption (A5/1, A5/3)

Secure channel

Physical medium (air)

User | Key
---|---
Alice Doe | 340934c3
Betty Lee | b9842544
Cheryl Zang | 93d94520
Pat Dobbs | 2ea0f48d
Disk Encryption

K = b9842544

macOS®

Hard Drive
Crypto in your browser: TLS (Transport Layer Security)

No pre-shared key, yet “guarantees” secret & authenticated communication with amazon.com.
Crypto in CS23200/33250

- A brief overview of major concepts and tools
- Cover (some of) big “gotchas” in crypto deployments
- Cover background for networking and authentication later

Not going to cover math, proofs, or many details. Consider taking CS284 (Cryptography)!
<table>
<thead>
<tr>
<th>Security Goal</th>
<th>Confidentiality</th>
<th>Authenticity/Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-shared key?</td>
<td>Yes (&quot;Symmetric&quot;)</td>
<td>Symmetric Encryption (aka Secret-key Encryption)</td>
</tr>
<tr>
<td></td>
<td>No (&quot;Asymmetric&quot;)</td>
<td>Public-Key Encryption</td>
</tr>
</tbody>
</table>
Rest of this lecture

- Symmetric Encryption Basics
- Stream Ciphers
- Block Ciphers
Rest of this lecture

- **Symmetric Encryption Basics**
  - Stream Ciphers
  - Block Ciphers
Ciphers (a.k.a. Symmetric Encryption)

A cipher is a pair of algorithms Encrypt, Decrypt:

```
K
m
```

Encrypt \(C\)

```
K
m/⊥
```

Require that decryption recovers the same message.
Historical Cipher: ROT13 ("Caesar cipher")

Encrypt(K,m): shift each letter of plaintext forward by K positions in alphabet (wrap from Z to A).

Plaintext:   DEFGH
Key (shift):  3
Ciphertext:  FGHKL

Plaintext:   ATTACKATDAWN
Key (shift):  13
Ciphertext:  NGGNPXNGQNJNA
Historical Cipher: Substitution Cipher

Encrypt(K,m): Parse key K as a permutation \( \pi \) on \{A,… Z\}. Apply \( \pi \) to each character of m.

P: ATTACKATDAWN
K: \( \pi \)
C: ZKKZAMZKYZGT

How many keys?
26! \( \approx 2^{88} \)
9 million years to try all keys at rate of 1 trillion/sec
Cryptanalysis of Substitution Cipher

CELEBRITY CIPHER by Luis Campos

Celebrity Cipher cryptograms are created from quotations by famous people, past and present. Each letter in the cipher stands for another.

"UPXEGTHWZHFXYLFHOSTPNFXHM
TPJSXEPOXVPAGAVGPOSLEEBXXOA
PMLNPFX,TPJ'CXZPWEEBXVBZX
EBHOS." — V.W.YXGVHO

Previous Solution: "Time is the cruelest teacher; first she gives the test, then teaches the lesson." — Leonard Bernstein

TODAY'S CLUE: rsenba N

© 2012 by NEA, Inc., dist. by Universal Uclick 9-20
Quick recall: Bitwise-XOR operation

We will use bit-wise XOR:

\[
\begin{array}{c}
0101 \\
\oplus 1100 \\
\hline \\
1001
\end{array}
\]

Some Properties:

- \( X \oplus Y = Y \oplus X \)
- \( X \oplus X = 000\ldots0 \)
- \( X \oplus Y \oplus X = Y \)
Cipher Example: One-Time Pad

Key $K$: Bitstring of length $L$

Plaintext $M$: Bitstring of length $L$

Encrypt($K, M$): Output $K \oplus M$

Decrypt($K, C$): Output $K \oplus C$

Example:

$$
\begin{array}{c}
0101 \\
\oplus 1100 \\
\hline
1001
\end{array}
$$

Correctly decrypts because

$$
K \oplus C = K \oplus (K \oplus M) = (K \oplus K) \oplus M = M
$$

Q: Is the one-time pad secure?

Bigger Q: What does “secure” even mean?
Evaluating Security of Crypto Algorithms

Kerckhoff’s Principle: Assume adversary knows your algorithms and implementation. The only thing it doesn’t know is the key.

1. Quantify adversary goals
   Learn something about plaintext? Spoof a message?
2. Quantify adversary capabilities
   View ciphertexts? Probe system with chosen inputs?
3. Quantify computational resources available to adversary
   Compute cycles? Memory?
**Ciphertext-only attack**: The adversary sees ciphertexts and attempts to recover some useful information about plaintexts.

More attack settings later.
Recovering Partial Information; Partial Knowledge

- Recovering entire messages is useful
- But recovering **partial information** is also be useful

A lot of information is missing here.

But can we say who this is?

- Attacker may know large parts of plaintext already (e.g. formatting strings or application content). The attacker tries to obtain something it doesn’t already know.

\[
M = \text{http://site.com?password=...}
\]
“Attacks” versus “Security”

An **attack** is successful as long as it recovers **some** info about plaintext that is useful to adversary.

Encryption should hide **all possible partial information** about plaintexts, since what is useful is situation-dependent.
Attacks can succeed without recovering the key

$K \xrightarrow{} m_1, \ldots, m_q \xrightarrow{} C_1, \ldots, C_q \xrightarrow{} m/\bot \xleftarrow{} K$

**Full break:** Adversary recovers $K$, decrypts all ciphertexts.

**However:** Clever attacker may compromise encryption without recovering the key.
Security of One-Time Pad

**Claim:** If adversary sees *only one* ciphertext under a random key, then any plaintext is equally likely, so it cannot recover any partial information besides plaintext length.

Ciphertext observed: 10111
Possible plaintext: 00101
⇒ Possible key: 10010

1. Adversary goal: Learn partial information from plaintext
2. Adversary capability: Observe a single ciphertext
3. Adversary compute resources: Unlimited time/memory (!)
Issues with One-Time Pad

1. Reusing a pad is insecure
2. One-Time Pad is *malleable*
3. One-Time Pad has a long key
Issue #1: Reusing a One-Time Pad is Insecure

\[
\begin{align*}
\text{HELLOALICE} \oplus \text{Pad} &= C_1 \\
\text{HELLOALICE} \oplus \text{Pad} &= \text{HELLOALICE} \\
\text{PWDHAMSTER} \oplus \text{Pad} &= C_2 \\
\text{PWDHAMSTER} \oplus \text{Pad} &= \text{PWDHAMSTER}
\end{align*}
\]
Issue #1: Reusing a One-Time Pad is Insecure

Has led to real attacks:
- Project Venona (1940s) attack by US on Soviet encryption
- MS Windows NT protocol PPTP
- WEP (old WiFi encryption protocol)
- Fortiguard routers! [link]
Issue #2: One-Time Pad is \textit{Malleable}

\[
\begin{align*}
\text{PAYALICE$1} & \oplus \text{Pad} = C \\
000\text{ALICE00} & \oplus 000\text{DAVID00} = C'
\end{align*}
\]

Decrypt(Pad, C') = \text{PAYDAVID$1}
Issue #3: One-Time Pad Needs a Long Key

**Can prove:** Any cipher as secure as the OTP must have:
\[ \text{Key-length} \geq \text{Plaintext-length} \]

**In practice:**
- Use *stream cipher*: \( \text{Encrypt}(K, m) = G(K) \oplus m \)
- Add *authentication tag*
- Use *nonces* to encrypt multiple messages
Outline

- Symmetric Encryption Basics
- **Stream Ciphers**
- Block Ciphers
Stream cipher syntax: Algorithm $G$ that takes one input and produces an very long bit-string as output.

Usually very, very large (petabytes if needed)

Key/Seed $k$: 1100..11

Typically 16 or 32 bytes.

$G(k)$: 11111010001000111010100101000101100100111100...

$\oplus$ DONUTSDONUTSDONUTSDONUTSDONUTSDONUTSDONUTSDON

Use $G(\text{seed})$ in place of pad.
Still malleable and still one-time, but key is shorter.
Security goal: When \( k \) is random and unknown, \( G(k) \) should “look” random.

… even to an adversary spending a lot of computation.

Much stronger requirement that “passes statistical tests”.

Brute force attack: Given \( y = G(k) \), try all possible \( k \) and see if you get the string \( y \).

Clarified goal: When \( k \) is random and unknown, \( G(k) \) should “look” random to anyone with less computational power needed for a brute force attack.

(keystrength = 256 is considered strong now)
Aside: Fundamental Physical Property of the Universe*

There exist (1-to-1) functions (say on bitstrings) that are:
1) Very fast to evaluate
2) Computationally infeasible to reverse

The disparity can be almost arbitrarily large!

Evaluating $y = f(x)$ may only take a few cycles....

... and finding $x$ from $y$ within the lifetime of the universe may not be possible, even with a computer made up of every particle in the universe.

*conjectured, but unproven property
## Computational Strength

<table>
<thead>
<tr>
<th># Steps</th>
<th>Who can do that many?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{56}$</td>
<td>Strong computer with GPUs</td>
</tr>
<tr>
<td>$2^{80}$</td>
<td>All computers on Bitcoin network in 4.5 hours</td>
</tr>
<tr>
<td>$2^{128}$</td>
<td>Very large quantum computer? (Ask Fred+Bill)*</td>
</tr>
<tr>
<td>$2^{192}$</td>
<td>Nobody?</td>
</tr>
<tr>
<td>$2^{256}$</td>
<td>Nobody?</td>
</tr>
</tbody>
</table>

*Not directly comparable but this is an estimate of equivalent power. Quantum computers are most effective against public-key crypto, but they also speed up attacks on symmetric-key crypto. (More next week.*)
Practical Stream Ciphers


Pad reuse can still happen with stream ciphers.

$m_1 \oplus G(k) \rightarrow \text{ciphertext}$

$m_2 \oplus G(k) \rightarrow \text{ciphertext}$
Addressing pad reuse: Stream cipher with a nonce

Stream cipher with a nonce: Algorithm $G$ that takes two inputs and produces a very long bit-string as output.

- “nonce” = “number once”.
- Usually denoted $IV = “initialization vector”$

Security goal: When $k$ is random and unknown, $G(IV, k)$ should “look” random and independent for each value of $IV$. 
Solution 1: Stream cipher with a nonce

- If nonce repeats, then pad repeats
Example of Pad Re-use: WEP

IEEE 802.11b WEP: WiFi security standard ’97-‘03

IV

IV is 24-bit wide counter

- Repeats after $2^{24}$ frames ($\approx 16$ million)
- IV is often set to zero on power cycle

Solutions: (WPA2 replacement)
- Larger IV space, or force rekeying more often
- Set IV to combination of packet number, address, etc
Example of Pad Re-use: WEP

IEEE 802.11b WEP: WiFi security standard ’97-‘03

- Repeats after $2^{24}$ frames ($\approx 16$ million)
- Often set to zero on reset

Solutions: (WPA2 replacement)
- Larger IV space
- Set IV to combination of packet number, address, etc

Warning: Broken
Issues with One-Time Pad

1. Reusing a pad is insecure

2. One-Time Pad is *malleable*

3. One-Time Pad has a long key

More difficult to address; We will return to this later.

- Use unique nonces
- Use stream cipher with short key
Rest of this lecture

- Symmetric Encryption Basics
- Stream Ciphers
- **Block Ciphers**
Next Up: Blockciphers

Blockciphers are a ubiquitous crypto tool applied to many different problems.

**Informal definition:** A blockcipher is essentially a substitution cipher with a very large alphabet and a very compact key. Require that efficient algorithms for forward and backward directions.

**Typical parameters:**
Alphabet = \{0,1\}^{128}
Key length = 16 bytes.

Plan: Build many higher-level protocols from a good blockcipher.

Now: Two example blockciphers, DES and AES.
Data Encryption Standard (DES)

- Originally designed by IBM
- Parameters adjusted by NSA
- NIST Standard in 1976
  - Block length $n = 64$
  - Key length $k = 56$

Parses input block into 32-bit chunks and applies 16 rounds of a “Feistel Network”
DES is Broken

<table>
<thead>
<tr>
<th>Attack</th>
<th>Complexity</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biham&amp;Shamir</td>
<td>$2^{47}$ encrypted blocks</td>
<td>1992</td>
</tr>
<tr>
<td>DESCHALL</td>
<td>41 days</td>
<td>1997</td>
</tr>
<tr>
<td>EFF Deepcrack</td>
<td>4.5 days</td>
<td>1998</td>
</tr>
<tr>
<td>EFF Deepcrack</td>
<td>22 hours</td>
<td>1999</td>
</tr>
</tbody>
</table>

- 3DES ("Triple DES") is still used by banks
- 3DES encrypts three times (so key length is 118)
- 3DES is not known to be broken but should be avoided
GET CRACKING

These are the types of DES cracking jobs that we support:

Windows LM/NTLMv1 Authentication
PPTP VPNs  WPA-Enterprise
des_crypt() Hashes
DES Kerberos5  Known Plaintext DES

NOTE: There are currently extremely high wait times.
We're in the process of adding capacity to speed things up.

QUEUE WAIT TIME:
Standard 46.2 Days, ASAP 1.0 Days

WARNING: Charges will show up on your credit card statement as from “crack.sh” and processed through Stripe. We've experienced a high number of our charges being reported as fraudulent, so we'll be blacklisting any accounts that contest charges for jobs submitted. If you wish to cancel a job or have any issues, please email david@toorcon.org and we'll be happy to cancel and refund any charges.
Advanced Encryption Standard (AES)

- NIST ran competition to replace DES starting in 1997
- Several submissions, *Rijndael* chosen and standardized
- AES is now the gold standard blockcipher
- Very fast; Intel chips even have AES instructions
Due to Rijmen and Daemen
- Block length \( n = 128 \)
- Key length \( k = 128, 192, 256 \)

Different structure from DES.
- 10 rounds of “substitution-permutation network”
AES is not (know to be) broken

<table>
<thead>
<tr>
<th>Attack</th>
<th>Complexity</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bogdanov et al.</td>
<td>$2^{126.1}$</td>
<td>2011</td>
</tr>
</tbody>
</table>

- Compare to trying all keys: $2^{126.1} \approx 2^{128}/4$
- Always prefer AES for a blockcipher if setting can support it (i.e. everything except low-power hardware)
- AES is thought to be a good "Pseudorandom Permutation"

- Outputs all look random and independent, even when inputs are maliciously controlled.
- Formal definition in CS284.
Example - AES Input/Outputs

- Keys and inputs are 16 bytes = 128 bits

- $K_1$: 9500924ad9d1b7a28391887d95fcfbd5
- $K_2$: 9500924ad9d1b7a28391887d95fcfbd6

$AES_{K_1}(00..00)=$8b805ddb39f3eee72b43bf95c9ce410f
$AES_{K_1}(00..01)=$9918e60f2a20b1b81674646dceebdb51
$AES_{K_2}(00..00)=$1303270be48ce8b8dd8316fdba38eb04
$AES_{K_2}(00..01)=$96ba598a55873ec1286af646073e36f6
So we have a blockcipher...

- Now what?

It only processes 16 bytes at a time, and I have a whole lot more data than that.

This next step is where everything flies off the rails in implementations...
Encrypting large files: ECB

- ECB = “Electronic Code Book”

\[
\text{AES-ECB}_k(M)
\]

- Parse \( M \) into blocks \( M_1, M_2, \ldots, M_t \)
  \( \text{// all blocks except } M_t \text{ are 16 bytes} \)
- Pad \( M_t \) up to 16 bytes
- For \( i = 1 \ldots t \):
  - \( C_i \leftarrow \text{AES}_k(M_i) \)
- Return \( C_1, \ldots, C_t \)
The ECB Penguin

- 16 byte chunks are consecutive pixels

Plaintext

ECB Ciphertext

- It gets even worse…
Encrypting large files, Attempt #2: CTR

- CTR = “Counter Mode”
- Idea: Build a nonce-based stream cipher from AES

\[
\text{AES-CTR}_k(\text{IV}, M)
\]
- Parse \( M \) into blocks \( M_1, M_2, \ldots, M_t \)
  // all blocks except \( M_t \) are 16 bytes
- For \( i = 1 \ldots t \):
  - \( C_i \leftarrow M_i \oplus \text{AES}_k(\text{IV}+i) \)
- Return \( \text{IV}, C_1, \ldots, C_t \)

Notes:
- No need to pad last block
- Must avoid reusing part of stream

When combined with authentication, CTR is a good cipher.
Penguin Sanity Check

Plaintext  ECB Ciphertext  CTR Ciphertext

Looks random
Encrypting large files, Attempt #3: CBC

- CBC = “Cipher Block Chaining”
- Nonce-based, but not a stream cipher
- Historical option (sometimes used without nonce)

AES-CBC\(_k(IV,M)\)

- Parse \(M\) into blocks \(M_1, M_2, \ldots, M_t\)
  // all blocks except \(M_t\) are 16 bytes
- Pad \(M_t\) up to 16 bytes
- \(C_0 \leftarrow IV\)
- For \(i=1\ldots t:\)
  - \(C_i \leftarrow AES_k(M_i \oplus C_{i-1})\)
- Return \(C_0, C_1, \ldots, C_t\)

Decryption

- \(M_1, M_2, \ldots, M_t\)
- \(C_t \leftarrow AES_{-1_k}(C_{t-1} \oplus C_t)\)
- \(M_1 = AES_{-1_k}(IV \oplus C_0)\)
- \(M_2 = AES_{-1_k}(C_1)\)
Encrypting large files, Attempt #3: CBC

- CBC = “Cipher Block Chaining”
- Nonce-based, but not a stream cipher
- Historical option (sometimes used without nonce)

\[
\text{AES-CBC}_k (IV, M)
\]

- Parse \( M \) into blocks \( M_1, M_2, \ldots, M_t \)
  // all blocks except \( M_t \) are 16 bytes
- Pad \( M_t \) up to 16 bytes
- \( C_0 \leftarrow IV \)
- For \( i=1\ldots t \) :
  - \( C_i \leftarrow \text{AES}_k (M_i \oplus C_{i-1}) \)
- Return \( C_0, C_1, \ldots, C_t \)

When combined with authentication, CBC is a good cipher.

Warning: Padding creates havoc with authentication. Very difficult to implement.
Blockcipher Encryption Summary

- AES is unbroken
- AES-CTR is most robust construction for confidentiality
- AES-CTR/AES-CBC do not provide authenticity/integrity and should almost never be used alone.
The End