Introduction to Cryptography
The Basics
Modern cryptography is a branch of applied mathematics.

About 100 years ago, cryptanalysts were using group theory and permutation theory—and the amount of math used has increased dramatically since then.

(One of the reasons for the British success during World War II at Bletchley Park (and the Poles before them): they hired mathematicians, rather than the linguists employed as cryptanalysts during World War I)

Consequently, this unit will have far more math than the rest of the course.
What is “Cryptography”? 

- Literally: “secret writing” (from Latin roots)
- (Purists call the subject “cryptology”, which includes cryptography, cryptanalysis, etc.)
- Historically, cryptography was used for confidentiality: keeping messages secret
- Encryption goes back thousands of years
- Today, we do much more, including authentication, integrity-checking, “signing” documents, playing games of chance over the Internet, cryptocurrencies like Bitcoin, etc.
The Caesar Cipher

We’ll start by using historical, pen-and-paper ciphers on ordinary letters—it’s easier to see what’s happening, and the principles are the same.

Replace each letter with the one three down in the alphabet:

\[
\begin{array}{cccccccc}
A & B & C & \cdots & X & Y & Z \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
D & E & F & \cdots & A & B & C
\end{array}
\]

- According to Suetonius, this cipher was used by Julius Caesar
- But why did he shift by 3? Could he have used a different number?
In modern terminology, encryption is done with and \textit{algorithm} and a \textit{key}.

For Caesar, the algorithm was

\[(P_i + K) \mod 23 \rightarrow C_i\]

where \(P_i\) is a character of the \textit{plaintext}, \(C_i\) is the corresponding \textit{ciphertext} character, and \(K\) is the key.

(The classical Latin alphabet had 23 letters: no J, U, or W...)

The key \(K\) was fixed at 3.

(We sometimes say it used a key of ‘D’, because it mapped A→D).
General Flow

plaintext $\xrightarrow{\text{encryption}}$ ciphertext $\xrightarrow{\text{decryption}}$ plaintext

- plaintext, cleartext — original message
- ciphertext — mangled information
- key — additional information used for encryption and decryption
Codes and Ciphers

- **Ciphers** operate on syntactic elements, e.g., letters or groups of letters
- **Codes** operate on semantic elements, e.g., words, phrases, and sentences
- The output of a codebook can be **superenciphered** for greater security
- Commercial codebooks were used for compression—telegraph companies charged by the word—but provided some security from casual eavesdroppers
- Through World War II, though, most governments and militaries used stronger codes for confidentiality
- (Codes are no longer used seriously)

(from the 1896 Atlas Universal Travelers’ and Business Telegraphic Cipher Code)
A cryptosystem is a pair of functions, $E$ and $D$, such that:

\[
\begin{align*}
\text{C} & \leftarrow E(\text{P}, \text{K}) \\
\text{P} & \leftarrow D(\text{C}, \text{K}') \\
\text{P} & = D(E(\text{P}, \text{K}), \text{K}')
\end{align*}
\]

where $P$ is the plaintext, $C$ is the ciphertext, $E$ is the encryption function, $D$ is the decryption function, and $K$ and $K'$ are keys. Often, $K' = K$; sometimes, $K' = F(K)$.

Today, $K \in \{0, 1\}^l$, $P \in \{0, 1\}^m$, and $C \in \{0, 1\}^n$. The key length is $l$, the input blocksize is $m$, and the output blocksize is $n$. Generally, $m = n$. 

A cryptosystem is pair of algorithms that take a key and under control of that key convert plaintext to ciphertext and back.

Plaintext is what you want to protect; ciphertext should appear to be random gibberish.

The design and analysis of today’s cryptographic algorithms is highly mathematical. Do not try to design your own algorithms.

I repeat: do NOT try to design your own algorithms; with rare exceptions, avoid any product that brags of their own, “highly secure” encryption algorithms.
Without knowledge of $K$, inverting $E$ should be infeasible. That is, cryptanalysis should be infeasible.

That should be true even if the enemy has many $\langle P, C \rangle$ pairs, or even if the enemy can supply many $P$s to be encrypted.

The keylength $l$ should be large enough that the enemy can not perform a brute force (also known as exhaustive search) attack on the key space.

Assume that the enemy knows all details of your algorithm, but does not know the key (Kerckhoffs, 1883).

Later, we’ll discuss other necessary security properties for cryptosystems.
Keys

- Must be strongly protected
- Ideally, should be a random set of bits of the appropriate length
- Ideally, each key should be used for a limited time only
- Ensuring that these properties hold is a major goal of cryptographic research and engineering
Brute-Force Attacks

- Build massively parallel machine
- Can be distributed across the Internet
- Give each processor a set of keys and a plaintext/ciphertext pair
- If no known plaintext, look for probable plaintext (i.e., length fields, high-order bits of ASCII text, etc.)
- On probable hit, check another block and/or do more expensive tests
Adding one bit to the key doubles the work factor for brute force attacks
The effect on encryption time is often negligible or even free
It costs *nothing* to use a longer RC4 key
Going from 128-bit AES (Advanced Encryption Standard) to 256-bit AES takes (at most) 40% longer, but increases the attacker’s effort by a factor of $2^{128}$
Using triple DES (Data Encryption Standard) costs 3× more than DES to encrypt, but increases the attacker’s effort by a factor of $2^{112}$
Moore’s Law favors the defender
Julius Caesar (supposedly) never changed his key; if the key was ever recovered, there would be no more secrecy.

A brute force attack on 22 possible keys wouldn’t take long, even by hand.

But: were his enemies even literate? Pompey and his officers almost certainly could read, but what about Caesar’s enemies in Gaul?

(Al-Kindi published a ground-breaking work on cryptanalysis circa 850 in Baghdad—but that implies that he encountered sophisticated methods of encryption. Al-Khalil wrote a (now-lost) book on cryptography in Basra about 75 years earlier.)
A *monoalphabetic cipher* always maps the same plaintext symbol (letter) to the same ciphertext symbol.

- Many characteristics, e.g., letter frequencies, show through.
- Solving these by hand is easy.
- In a good cipher, the output frequency is *flat*.
- In other words, we want high entropy: it should be indistinguishable from true-random.
English Letter Frequencies

(Data taken from *Alice’s Adventures in Wonderland*)
**Entropy (Shannon, 1948)**

- *Entropy* measures the amount of disorder—randomness—in a set
- Entropy is usually measured in bits: $\log_2 26 \approx 4.70$ if all letters were equally probable
- Lack of randomness aids cryptanalysis
- Simple Caesar (or other substitution) ciphers do not increase entropy:
  - `$ entropy alice.txt`  
    4.161  
  - `$ caesar alice.txt | entropy`  
    4.161  
  - `$ substitute -k VKQCTDRSBJUNPEOFAGIZGWXYLH alice.txt | entropy`  
    4.161
During World War II, Claude Shannon worked on SIGSALY, a voice encryptor. This led him to invent information theory, and publish mathematical formalizations of communications and secrecy.

A fundamental issue: how much information does a message contain?

Example: if you flip a coin 100 times and get “heads” 99 times, an “H” carries very little information. But a “T” carries a lot.

English is redundant—you don’t need every letter.

Start of an old subway ad: “F U CN RD THS. . . ”

\[-\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)\]
In a secrecy system there are two statistical choices involved, that of the message and of the key. We may measure the amount of information produced when a message is chosen by $H(M)$:

$$H(M) = - \sum P(M) \log P(M),$$

the summation being over all possible messages. Similarly, there is an uncertainty associated with the choice of key given by:

$$H(K) = - \sum P(K) \log P(K).$$

Information and uncertainty are maximized when the distributions are flat.
Increasing Entropy: Longer Keys

- Use a different key for different letters of the plaintext.
- Example: Vigenère cipher—repeat a keyword as necessary; encrypt each letter with the appropriate letter of the keyword.

```
J U D G E J U D G E J U D D ...
A L I C E S A D V E N T U ...
J F L I I B U G B I W N X ...
```

- The two As are encrypted differently—but the two Es happen to be encrypted with the same key letter.

```
$ vigenere -k JUDGE alice.txt | entropy
4.626
```
Increasing Entropy: Longer Encryption Blocks

Longer encryption blocks flatten frequency distribution ($\log_2 26^2 \approx 9.4$)

Caesar Digraph Frequency

Playfair Digraph Frequency

```bash
$ caesar alice.txt | entropy -n 2
7.693

$ playfair -k MYVOW alice.txt | \ entropy -n 2
8.599
```
Increasing Entropy: Transposition

Transposition cipher: rearrange the order of the letters by transposing a matrix, to break up common letter sequences, e.g., THE and ING

Example: $k = 5 \times 4$

```
<table>
<thead>
<tr>
<th>A</th>
<th>L</th>
<th>I</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>S</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>V</td>
<td>E</td>
<td>N</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>I</td>
<td>N</td>
<td>W</td>
<td>O</td>
</tr>
</tbody>
</table>
```

$\rightarrow$ AEVUI ... LSERN ... IANEW ... CDTSO

$\$ transpose -k 4x17 alice.txt | entropy
4.162
$\$ transpose -k 4x17 alice.txt | entropy -n 2
8.320
$\$ transpose -k 4x17 alice.txt | entropy -n 3
12.386
Maximum Entropy

- We get maximum security if the key is truly random and is as long as the plaintext.
- This is called the one-time pad, and is used by some spies, the Washington-Moscow hotline, etc.
- One-time pads are guaranteed 100% secure—if the key is really random, i.e., not generated by an algorithm, and if it is never reused.
- (Invented in 1882 by Frank Miller, a Sacramento banker; reinvented in 1919 by Gilbert Vernam and Joseph Mauborgne. It is unclear if Mauborgne knew of Miller’s work.)

```
$ random -c 150000 >/tmp/random-key; entropy /tmp/random-key
4.700
$ otp -K /tmp/random-key alice.txt | entropy
4.700
$ otp -K /tmp/random-key alice.txt | entropy -n 2
9.396
```
During World War II, the Soviets had trouble producing enough true-random one-time pads. Some of their spies in the U.S. had duplicate pages. American cryptanalysts detected this and were able to read traffic that should have been perfectly secure.
Principles of Cipher Design

- Substitution
- Permutation
- Longer keys
- Encrypting blocks of plaintext

Most modern ciphers—which operate on bits and bytes, not ASCII letters—use combinations of these principles, and generally repeated combinations.

```
$ vigenere -k JUDGE alice.txt | entropy
4.626
$ vigenere -k JUDGE alice.txt | entropy -n 2
8.918
$ transpose -k 4x17 alice.txt | entropy
4.162
$ transpose -k 4x17 alice.txt | entropy -n 2
8.320
$ vigenere -k JUDGE alice.txt | transpose -k 4x17 | entropy -n 2
9.211
```
Did We Get It Right?

Let’s combine all three ciphers and see what we get

```
$ playfair -k SPHINX alice.txt | vigenere -k JUDGE | transpose -k 4x17
```

The frequency plot still isn’t flat—and is that the only metric?
Once More, But Iterated and With More Key Material

$ \text{playfair -k SPHINX alice.txt | vigenere -k JUDGE | transpose -k 4x17 | playfair -k BLACKQUARTZ | vigenere -k MYVOW | transpose -k 13x5}$

Noticeably better!
A cipher is no stronger than its key length: if there are too few keys, an attacker can enumerate all possible keys.

The old DES cipher has 56 bit keys — arguably too few in 1976; far too few today. (*Deep Crack* was built in 1996 by the EFF.)

Strength of cipher depends on how long it needs to resist attack.

No good reason to use less than 128 bits.

NSA rates 128-bit AES as good enough for SECRET traffic; 256-bit AES is good enough for TOP-SECRET traffic.

But a cipher can be considerably weaker! (A monoalphabetic cipher over all possible byte values has $2^{256}$ keys — a length of 1684 bits — but is trivially solvable.)

Let’s look at some modern designs.
Until the 1970s, ciphers were designed by governments and amateurs—there was almost no private sector or academic interest (or competence)

In 1972, the US and the Soviet Union negotiated a huge grain deal—and US intelligence later learned that the Soviets were eavesdropping on the American grain negotiators

Protecting unclassified civilian traffic was now seen as a matter of national security

The National Bureau of Standards (NBS) (now called NIST, the National Institute of Standards and Technology) put out a call for a modern, public encryption algorithm
There was only one submission: Lucifer, from IBM
Lucifer encrypted 64-bit blocks under the control of a 112-bit key; there were 16 rounds (iterations)
The basic structure was a Feistel network
The design went to NBS, which shared it with NSA
It came back changed: the S-boxes were different, and the key size was cut to 56 bits...
The result became DES: the Data Encryption Standard
DES: The Data Encryption Standard

- DES was the first modern, public design
- DES is a block cipher—it operates on blocks of fixed length: 64 bits
- Stream ciphers operate on continuous sequences of bits or bytes (or, less commonly other lengths)
- A tremendous amount of academic cryptographic knowledge came from studying DES—it was, after all, an NSA-approved design
- Even though it’s now obsolete, it’s still worth studying, to understand how a modern cipher can be built
The Feistel Network

- An iterated design; for DES, there are 16 rounds
- The key is converted to a *key schedule*; there is a separate *subkey* for each round
- Each round contains a keyed substitution/permutation
- In each round, the two halves are swapped; one half is combined with the subkey in the $F$ function, the output of which is in turn exclusive-ORed with the other half
- To decrypt, use the same key schedule, but go through the rounds in reverse-order
- (Spend some time understanding why that works. Briefly, exclusive-OR is its own inverse, and the swap/pass-through means that the correct data is XORed each time.)
DES Round Structure
The $F$ Function

- Feistel networks are general-purpose constructs—the security of the cipher depends on the $F$ function
- The heart of the DES $F$ function is the set of $S$-boxes, which are a set of 64-element arrays that implement a non-linear function of $R_i$ and $K_i$
- ($E$ expands $R$ by replicating some bits; $P$ is an unkeyed permutation to shuffle things around)
- The $S$-boxes are crucial to DES’ security—but where did they come from? It was long known that they did not appear to be random. Was there an NSA backdoor in the $S$-boxes?
DES Round Function

\[ R \text{ (32 bits)} \]

\[ E \]

48 bits

\[ + \]

\[ K_i \text{ (48 bits)} \]

\[ S0 \quad S1 \quad S2 \quad S3 \quad S4 \quad S5 \quad S6 \quad S7 \]

\[ P \]

32 bits
Confusion and Diffusion

Confusion  Each bit of ciphertext should depend on several—preferably many—bits of the key.

Diffusion  Each bit of the output should depend on many bits of the input. When a single bit of plaintext is changed, statistically half the bits of the ciphertext should change.

Rounds  No single round can achieve perfect confusion or diffusion. One reason for multiple rounds is to achieve this goal.
Was DES Secure?

A lot of people distrusted DES—had the NSA sabotaged it?
After all, the NSA has two roles, protecting US communications—and reading other countries’ messages
Would the NSA permit deployment of a truly strong cipher?
Whit Diffie and Martin Hellman published an analysis showing that the NSA could afford to build a brute-force DES-cracking engine, but almost no one else could
And what about the S-boxes?
A Senate committee concluded that the NSA had not sabotaged the design of DES
Was that the whole story?
The decision to get involved with NBS was hardly unanimous. This argued the opposite case – that, as Frank Rowlett had contended since World War II, in the long run it was more important to secure one’s own communications than to exploit those of the enemy.\textsuperscript{121}

Once that decision had been made, the debate turned to the issue of minimizing the damage. Narrowing the encryption problem to a single, influential algorithm might drive out competitors, and that would reduce the field that NSA had to be concerned about.

\textsuperscript{122} They compromised on a 56-bit key.
In 1991, two Israeli academics, Eli Biham and Adi Shamir, came up with differential cryptanalysis, a very powerful, general-purpose cryptanalysis technique.

They showed that the S-boxes were in fact extremely strong—the maximum strength of DES is the key size limit, $2^{56}$; the S-boxes provide $2^{54}$ strength.

This clearly could not have happened by accident. IBM and/or the NSA knew of differential cryptanalysis (IBM called it Attack T) 17 years earlier and designed DES to resist it.

The 56-bit key size limit remained—and we now know that that was intentional.
An Alleged Dialog . . .

**DES designer:** “You know, we knew about what you call differential cryptanalysis way back when we were designing DES.”

**Shamir:** “Yes, that’s quite obvious; congratulations.”

**DES designer:** “Thank you.”

**Shamir:** “Tell me, have you discovered any stronger attacks on DES since then?”

**DES designer:** Crickets!
The key size issue remained—civilian academics remained convinced that (a) the NSA had shortened the key size, and (b) the NSA was able to read DES.

The US government suggested “escrowed encryption”—there would be an overt backdoor to let the government read encrypted traffic—but no one was interested...

But they kept insisting that DES was secure enough.

In 1998, the Electronic Frontier Foundation settled the issue decisively—by building an open source DES-cracking engine for about $250,000.

And now factor in Moore’s Law.

As I said, DES is obsolete.

NIST started a new competition, for AES: the Advanced Encryption Standard.
AES was needed for several reasons:

1. The first, obviously, is that DES was clearly insecure.
2. It was possible to iterate DES with three separate keys, in so-called “EDE”—encrypt, decrypt, encrypt—mode, but there were other issues.
3. DES itself was too slow; 3DES was slower still.
4. (DES was designed to be implemented in hardware—why? it wasn’t an NSA plot—and things like bit permutations are fast in hardware: just route wires a bit differently. Today, we need speed in software, too, including on 8-bit processors.)
5. The 64-bit blocksize was too small (stay tuned... )
The AES Competition

- NIST announced an open competition; there were 15 submissions from all over the world
- Requirements: 128-bit blocksize, 128-, 192-, and 256-bit keys, secure, simple, efficient
- There were NIST-sponsored public conferences to discuss the candidates
- The NSA evaluated the submissions but did not change any part of any submission
- The process got high marks for openness, transparency, and fairness—and there were no accusations of NSA tampering
Rijndael, by Joan Daemon and Vincent Rijmen, was the choice of most outside experts as well as of NIST.

It’s very fast, on low-end and high-end processors.

Internal operations are on bytes, not bits, which contributes to its speed.

It uses 10 rounds for 128-bit keys, 12 rounds for 192 bits, and 14 rounds for 256-bit keys.

It uses a substitution/permutation network, rather than a Feistel design.

AES was formally approved in 2002; as of now, there are no attacks known that are even close to being feasible.
Convert a Block to a Matrix

<table>
<thead>
<tr>
<th>$p_0$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$p_5$</th>
<th>$p_6$</th>
<th>$p_7$</th>
<th>$p_8$</th>
<th>$p_9$</th>
<th>$p_{10}$</th>
<th>$p_{11}$</th>
<th>$p_{12}$</th>
<th>$p_{13}$</th>
<th>$p_{14}$</th>
<th>$p_{15}$</th>
</tr>
</thead>
</table>

$\downarrow$

<table>
<thead>
<tr>
<th>$p_0$</th>
<th>$p_4$</th>
<th>$p_8$</th>
<th>$p_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_5$</td>
<td>$p_9$</td>
<td>$p_{13}$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$p_6$</td>
<td>$p_{10}$</td>
<td>$p_{14}$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$p_7$</td>
<td>$p_{11}$</td>
<td>$p_{15}$</td>
</tr>
</tbody>
</table>
Round Operations

**SubBytes** A non-linear “substitute bytes” operation—each byte is replaced by a different one, using a single S-box for all 16 positions.

**ShiftRows** Rotate the bytes in each row by 0, 1, 2, or 3 positions, depending on the row.

**MixColumns** Apply a linear transform—specified as a matrix multiplication over the proper field—to all four bytes in each column. (Yes, “field”, as in algebra: $GF(2^8)$.

**AddRoundKey** Bitwise exclusive-OR of the matrix with the subkey for this round.

There are special operations before the first round and for the last round.
Stream Ciphers

- DES and AES are block ciphers—they can’t operate until enough bytes have arrived. In some situations, where bytes arrive slowly and asynchronously—think keystrokes on a Teletype—we need something that can encrypt a byte at a time.
- For security, though, we want a different key or a different *something* so that the same plaintext letter isn’t always mapped to the same ciphertext.
- The answer is a *stream cipher*, which stores internal state and hence can change behavior with each encryption.
- Typical modern stream ciphers generate a *keystream* which is exclusive-ORed with the plaintext to encrypt or the ciphertext to decrypt.

(Photo taken August 12, 2012, at the Connections Museum, Seattle.)
Teleprinters—“Teletype” is a brand name—in their modern form go back about 120 years.

The keyboard and the printer were connected directly to the communications line—if you typed “A”, 0100 0001 would be sent immediately with no protocol; if 0101 1000 was received, the “X” would be printed immediately (actually, it was slightly more complicated, but only slightly).

The encryptor and decryptor had to work instantly, on one byte at a time.

(Note also the paper tape reader/punch.)
The best-known public stream cipher is RC4, devised by Ron Rivest in 1987 (RC4 stands for “Ron’s Cipher 4”)

A primary goal: really fast encryption—remember how much slower computers were back then, and how DES was inherently slow

It was originally a trade secret of a company he’d founded—but in 1994, someone leaked or reverse-engineered the code . . .

RC4 is now considered obsolete—its output is distinguishable from random, and in many situations there are feasible attacks

Note well: stream ciphers are very hard to use securely—more on this in a few days
The 256-byte array state is initialized from the key, and is changed for each byte generated. Note the XOR in the last line of code, encrypting or decrypting a byte at a time in a variable-length buffer.

```c
for(counter = 0; counter < buffer_len; counter++) {
    x = (x + 1) % 256;
    y = (state[x] + y) % 256;
    swap_byte(&state[x], &state[y]);
    
xorIndex = (state[x] + state[y]) % 256;
    buffer_ptr[counter] ^= state[xorIndex];
}
```

(From the original messages to the Cypherpunks Archive at http://cypherpunks.venona.com/archive/1994/09/msg00304.html)
Encryption and Decryption are the Same Operation

Remember that XOR is its own inverse

\[ P_i \oplus K_i = C_i \]
\[ C_i \oplus K_i = (P_i \oplus K_i) \oplus K_i \]
\[ = P_i \oplus (K_i \oplus K_i) \]
\[ = P_i \oplus 0 \]
\[ = P_i \]

The use of XOR for encryption and decryption of teleprinter traffic goes back to the 1920s.
Vernam’s One-Time Pad

- Vernam (of AT&T) and Mauborgne (US Army Signal Corps) (re)invented the one-time pad around 1919–20.
- It used a paper tape of true-random characters for keying material; when the operator typed a letter, the paper tape would advance to encrypt that character.
- But that much random tape was hard to handle, so Morehouse (AT&T) suggested using two tapes of relatively prime lengths:
  \[ C_i = P_i \oplus K_{1,i} \oplus K_{2,j} \]
- This is not a one-time pad, it’s a stream cipher: the actual key, \( K_i \), is derived algorithmically from \( K_{1,i} \) and \( K_{2,j} \)—and Friedman cracked it.
- Morehouse had invented a *stream cipher*.
Questions?

(Northern mockingbird eating a wasp, Morningside Park at W 113th St., September 9, 2020)