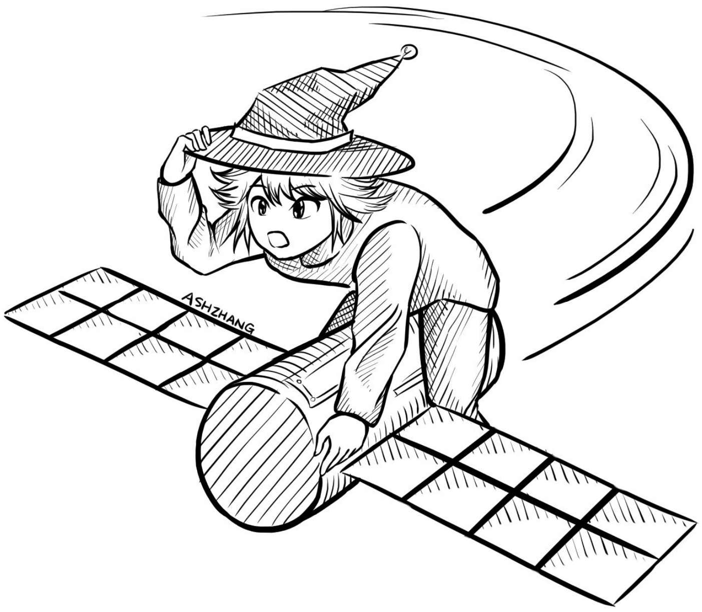
Symmetric-Key Cryptography

CS 161 Spring 2022 - Lecture 7

Announcements

- Project 1 is released
  - Checkpoint is due Friday, February 4th, 11:59 PM PT
  - Final submission is due Friday, February 18th, 11:59 PM PT
- Project party today!
  - 2-5pm in the Woz
- In person office hours & project parties really are better!
  - Much better student/unit-time support from TAs from both the student and TA viewpoint



Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	● One-time pads ● Block ciphers with chaining modes (e.g. AES-CBC) ● Stream ciphers	● RSA encryption ● ElGamal encryption
Integrity, Authentication	● MACs (e.g. HMAC)	● Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)
- Password management

Symmetric-Key Encryption

Textbook Chapter 6.1

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

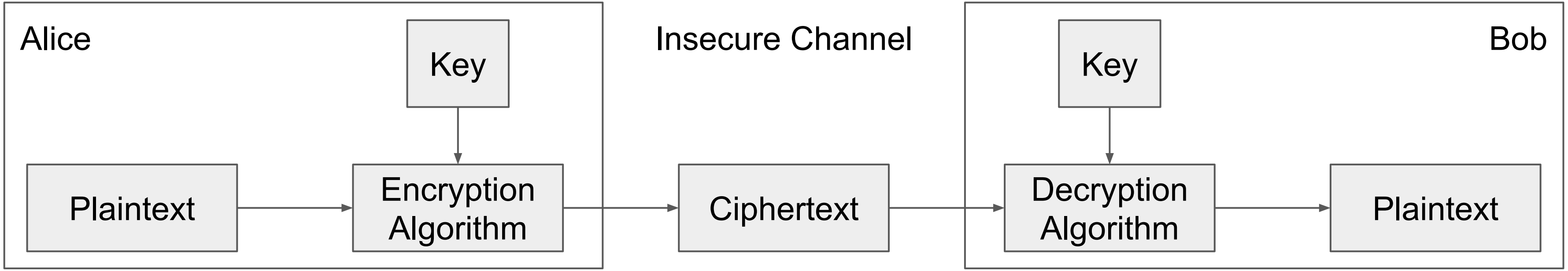
- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)
- Password management

Symmetric-Key Encryption

- The next few schemes are symmetric-key encryption schemes
  - **Encryption schemes** aim to provide *confidentiality* (but not integrity or authentication)
  - **Symmetric-key** means Alice and Bob share the same secret key that the attacker doesn't know
    - Don't worry about how Alice and Bob share the key for now
- For modern schemes, we're going to assume that messages are *bitstrings*
  - **Bitstring**: A sequence of bits (0 or 1), e.g. `1101010101001001010`
  - Text, images, etc. can usually be converted into bitstrings before encryption, so bitstrings are a useful abstraction. After all, everything in a computer is just a sequence of bits!

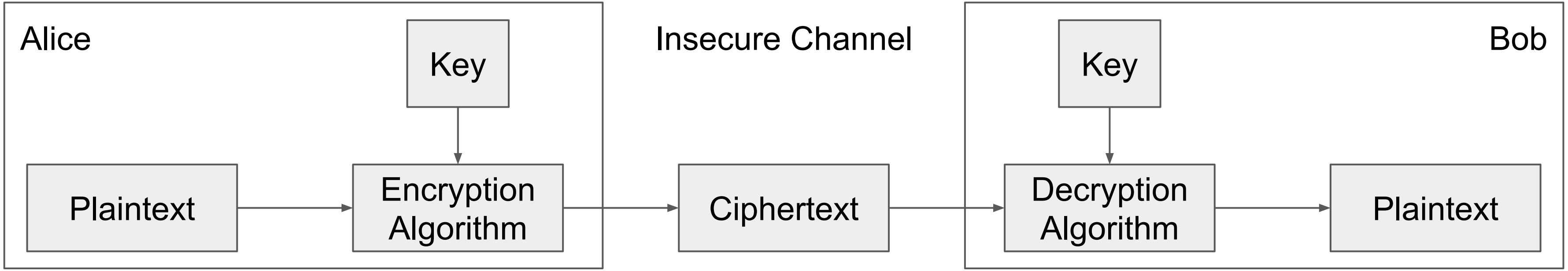
Symmetric-Key Encryption: Definition

- A symmetric-key encryption scheme has three algorithms:
  - KeyGen() → K: Generate a key K
  - Enc(K, M) → C: Encrypt a **plaintext** M using the key K to produce **ciphertext** C
  - Dec(K, C) → M: Decrypt a ciphertext C using the key K



Symmetric-Key Encryption: Definition

- What properties do we want from a symmetric encryption scheme?
  - **Correctness**: Decrypting a ciphertext should result in the message that was originally encrypted
    - Dec(K, Enc(K, M)) = M for all $K \leftarrow$ KeyGen() and M
  - **Efficiency**: Encryption/decryption algorithms should be fast: >1 Gbps on a standard computer
  - **Security**: Confidentiality



Defining Confidentiality

- Recall our definition of confidentiality from earlier: “An adversary cannot read our messages”
  - This definition isn’t very specific
    - What if Eve can read the first half of Alice’s message, but not the second half?
    - What if Eve figures out that Alice’s message starts with “Dear Bob”?
  - This definition doesn’t account for prior knowledge
    - What if Eve already knew that Alice’s message ends in “Sincerely, Alice”?
    - What if Eve knows that Alice’s message is “BUY!” or “SELL” but doesn't know which?

Defining Confidentiality

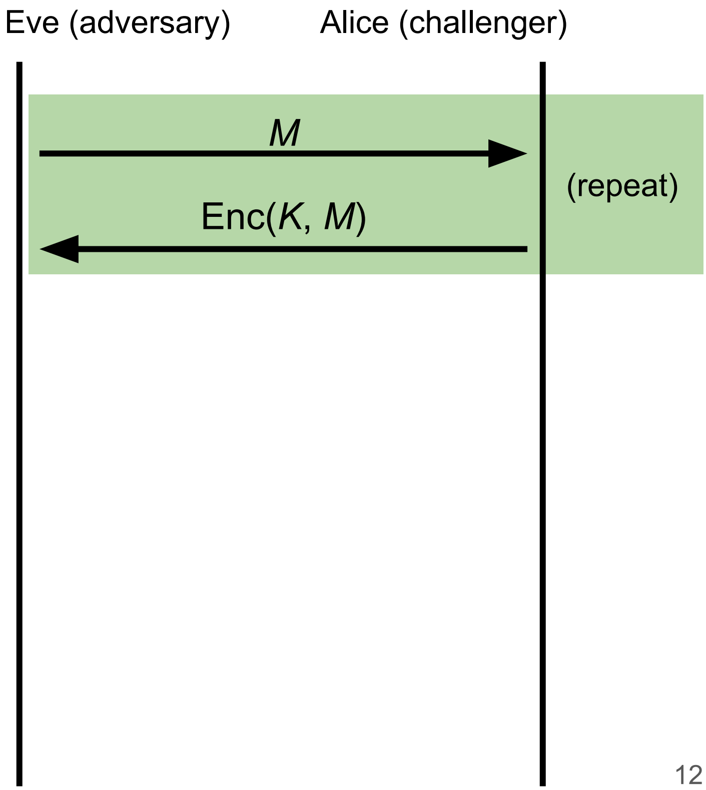
- A better definition of confidentiality: The ciphertext should not give the attacker *any additional information* about the plaintext.
- Let's design an experiment/ security game to test our definition:
  - Eve chooses two messages M_0 and M_1 of the same length
  - Alice chooses one message at random M_b, encrypts it, and sends the ciphertext
  - Eve knows either M_0 or M_1 was sent, but doesn't know which
  - Eve reads the ciphertext and tries to guess which message was sent
  - If the probability that Eve correctly guesses which message was sent is 1/2, then the encryption scheme is confidential
- Intuition
  - If the scheme is confidential, Eve can only guess with probability 1/2, which is no different than if Eve hadn’t sent the ciphertext at all
  - In other words: the ciphertext gave Eve no *additional* information about which plaintext was sent!

Defining Confidentiality: IND-CPA

- Recall our threat model: Eve can also perform a **chosen plaintext attack**
  - Eve can trick Alice into encrypting arbitrary messages of Eve's choice
  - We can adapt our experiment to account for this threat model
- A better definition of confidentiality: Even if Eve is able to trick Alice into encrypting messages, Eve can still only guess what message Alice sent with probability 1/2.
  - This definition is called **IND-CPA** (indistinguishability under chosen plaintext attack)
- Cryptographic properties are often defined in terms of “games” that an adversary can either “win” or “lose”
  - We will use one to define confidentiality precisely

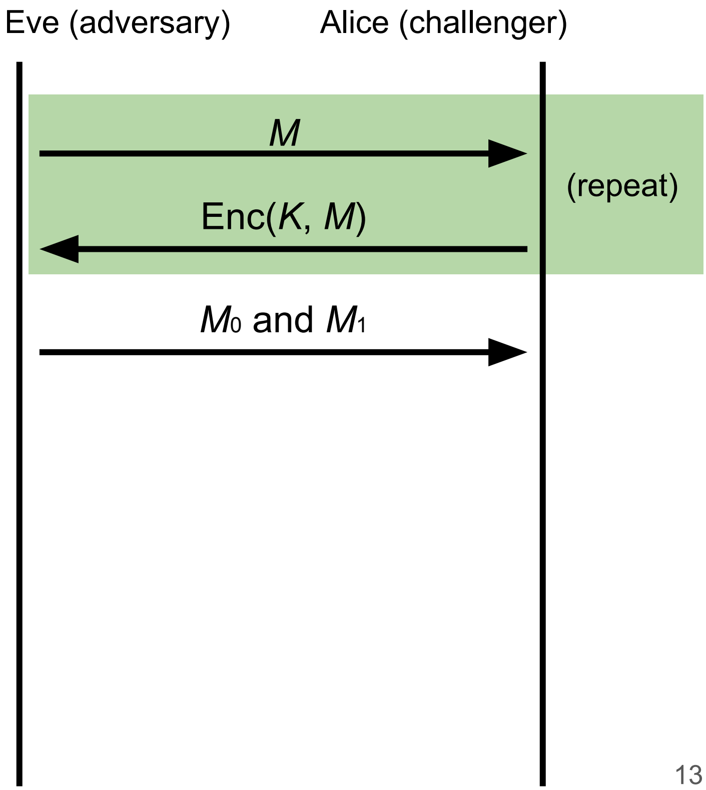
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts



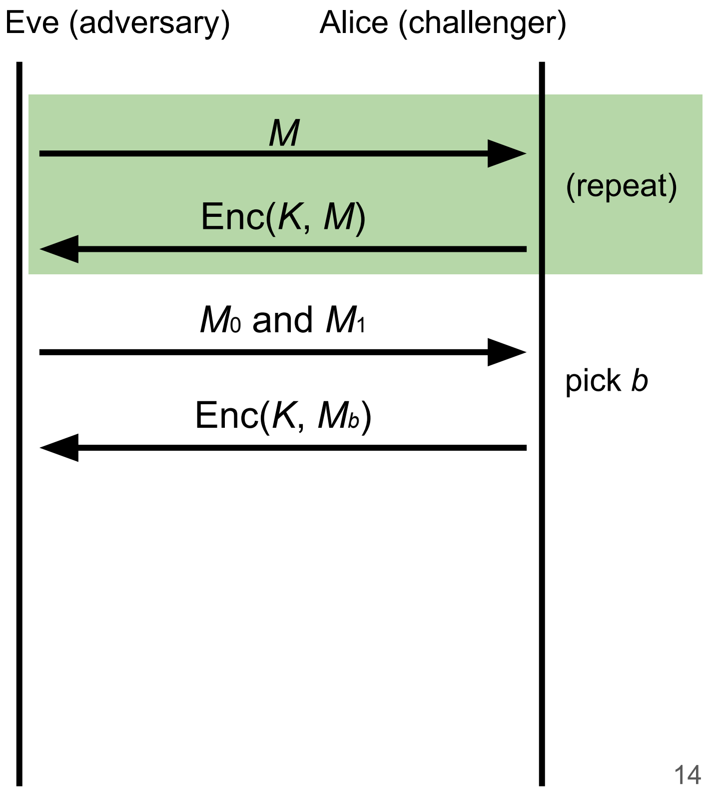
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice



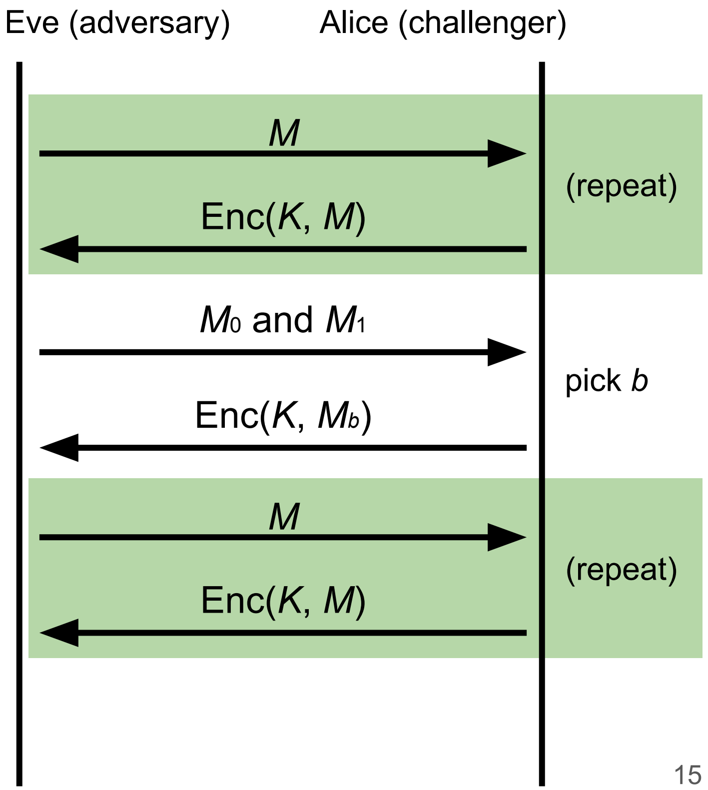
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
   a. Alice does not tell Eve which one was encrypted!



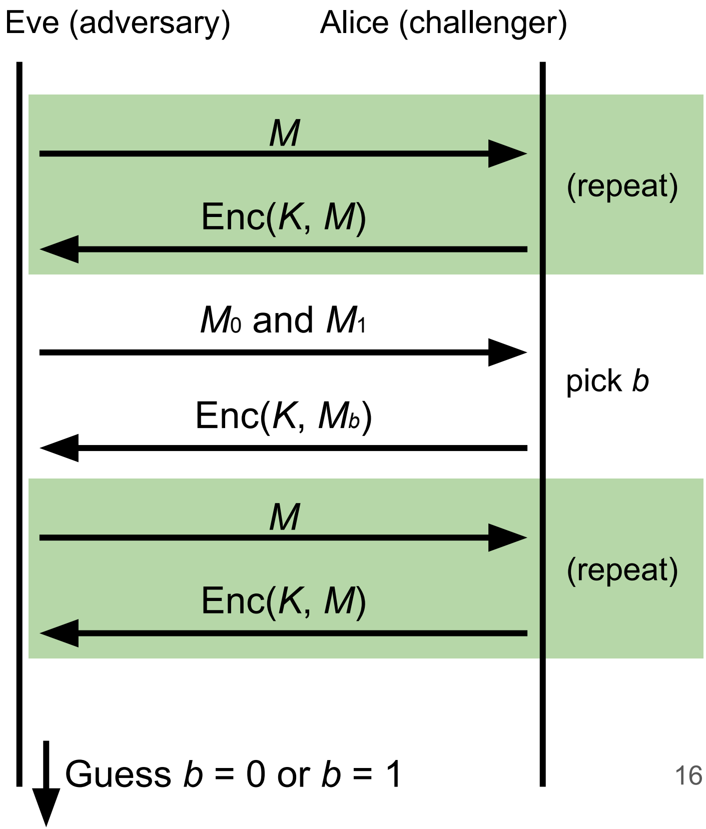
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
   a. Alice does not tell Eve which one was encrypted!
4. Eve may again choose plaintexts to send to Alice and receives their ciphertexts



Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
   a. Alice does not tell Eve which one was encrypted!
4. Eve may again choose plaintexts to send to Alice and receives their ciphertexts
5. Eventually, Eve outputs a guess as to whether Alice encrypted M_0 or M_1



Defining Confidentiality: IND-CPA

- If Eve correctly guesses which message Alice encrypted, then Eve wins. Otherwise, she loses.
- How does Eve guess whether M_0 or M_1 was encrypted? What strategy does she use?
  - We don't *assume* she uses a particular strategy; Eve represents all possible strategies
- Proving insecurity: There exists at least *one* strategy that can win the IND-CPA game with probability > 1/2
  - 1/2 is the probability of winning by random guessing
  - If you can be better than random, then the ciphertext has leaked information, and Eve is able to learn it and use it to gain an advantage!
- Proving security: For *all* attackers/Eve-s, the probability of winning the IND-CPA game is at most 1/2

Edge Cases: Length

- Cryptographic schemes are (usually) allowed to leak the length of the message
  - To hide length: All messages must always be the same length
    - 16-byte messages: We can’t encrypt large messages (images, videos, etc.)!
    - 1-GB messages: Sending small messages (text, Tweets, etc.) needs 1 GB of bandwidth!
    - This is unpractical
  - Applications that which to hide length must choose to *pad* their own messages to the maximum possible length before encrypting
- In the IND-CPA game: M_0 and M_1 must be the same length
  - To break IND-CPA, Eve must learn something other than message length

Edge Cases: Attacker Runtime

- Some schemes are theoretically vulnerable, but secure in any real-world setting
  - If an attack takes longer than the life of the solar system to complete, it probably won’t happen!
  - Or if it would require a computer made out of a literal galaxy worth of science-fiction nanotech
- In the IND-CPA game: Eve is limited to a practical runtime
  - One common practical limit: Eve is limited to polynomial runtime algorithms (no exponential-time algorithms)

Edge Cases: Negligible Advantage

- Sometimes it's possible for Eve to win with probability $1/2 + 1/2^{128}$
  - This probability is greater than 1/2, but it's so close to 1/2 that it's as good as 1/2.
  - Eve's advantage is so small that she can't use it for any practical attacks
- In the IND-CPA game: The scheme is secure even if Eve can win with probability $\leq 1/2 + \varepsilon$, where ε is *negligible*
  - The actual mathematical definition of negligible is out of scope
  - Example: $1/2 + 1/2^{128}$: Negligible advantage
  - Example: 2/3: Non-negligible advantage

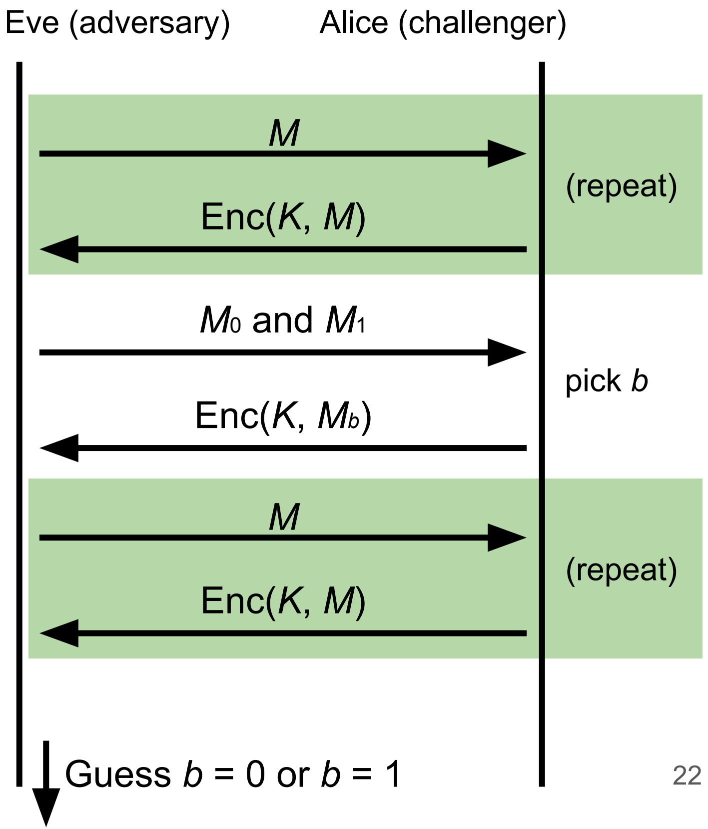
Edge Cases: Negligible Advantage

- Defining negligibility mathematically:
  - Advantage of the adversary should be exponentially small, based on the security parameters of the algorithm
  - Example: For an encryption scheme with a *k*-bit key, the advantage should be $O(1/2^k)$
- Defining negligibility practically:
  - A $1/2^{128}$ probability is completely inconceivable
  - A $1/2^{20}$ probability is fairly likely
    - "One in a million events happen every day in New York City"
  - In between these extremes, it can be messy
    - Different algorithms run faster or slower and have their own security parameters
    - Computers get more powerful over time
    - Recall: Know your threat model!
- **Takeaway**: For now, 2^{80} is a reasonable threshold, but this will change over time!

IND-CPA: Putting it together

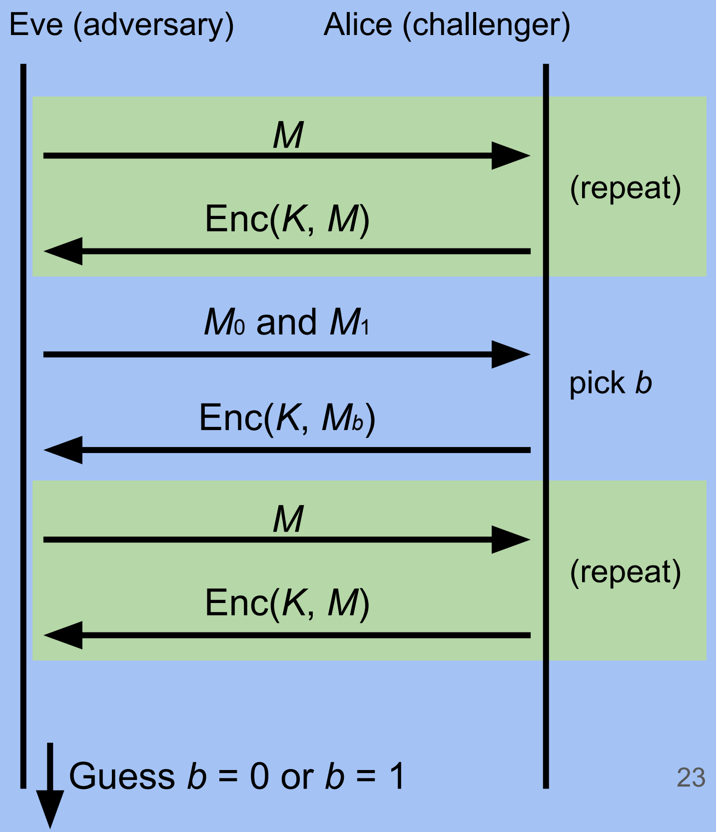
1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
   - Alice does not tell Eve which one was encrypted!
4. Eve may again choose plaintexts to send to Alice and receives their ciphertexts
5. Eventually, Eve outputs a guess as to whether Alice encrypted M_0 or M_1

- An encryption scheme is IND-CPA secure if for all polynomial time attackers Eve:
  - Eve can win with probability <= 1/2 + Ɛ, where Ɛ is *negligible.*

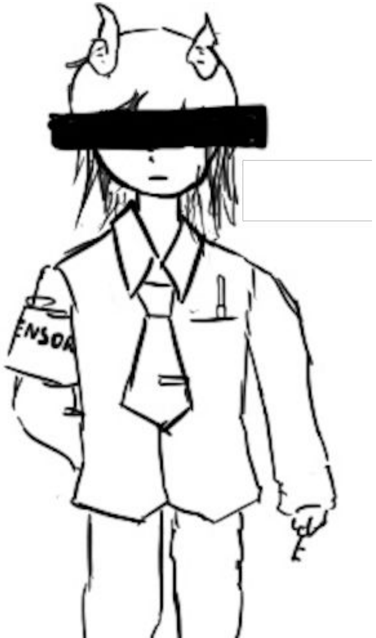


Enigma: Secure under IND-CPA?

- Enigma has a significant weakness: a letter never maps to itself!
  - No rotor maps a letter to itself
  - The reflector never maps a letter to itself
  - This property is necessary for Enigma's mechanical system to work
- What pair of messages should Eve send to Alice in the challenge phase?
  - Send $M_0 = A^k$, $M_1 = B^k$
  - M_0 is a string of k 'A' characters, M_1 is a string of k 'B' characters
- How can Eve probably know which message Alice encrypted?
  - If there are no 'A' characters, it was M_0
  - If there are no 'B' characters, it was M_1



One-Time Pads



Textbook Chapter 6.2 & 6.3

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)
- Password management

Review: XOR

The XOR operator takes two bits and outputs one bit:

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Useful properties of XOR:

$x \oplus 0 = x$
$x \oplus x = 0$
$x \oplus y = y \oplus x$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
$(x \oplus y) \oplus x = y$

Review: XOR Algebra

- Algebra works on XOR too

$y \oplus 1 = 0$	Goal: Solve for y
$y \oplus 1 \oplus 1 = 0 \oplus 1$	XOR both sides by 1
$y = 1$	Simplify with identities

One-Time Pads: Key Generation

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1

The key K is a randomly-chosen bitstring.

Recall: We are in the symmetric-key setting, so we'll assume Alice and Bob both know this key.

One-Time Pads: Encryption

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1
M	1	0	0	1	1	0	0	1	0	1	0	0

The plaintext *M* is the bitstring that Alice wants to encrypt.

Idea: Use XOR to scramble up *M* with the bits of *K*.

One-Time Pads: Encryption

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
M	1	0	0	1	1	0	0	1	0	1	0	0
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
C	1	1	1	1	1	1	0	0	0	0	1	1

Encryption algorithm: XOR each bit of *K* with the matching bit in *M*.

The ciphertext *C* is the encrypted bitstring that Alice sends to Bob over the insecure channel.

One-Time Pads: Decryption

Bob

K	0	1	1	0	0	1	0	1	0	1	1	1
C	1	1	1	1	1	1	0	0	0	0	1	1

Bob receives the ciphertext *C*. Bob knows the key *K*. How does Bob recover *M*?

One-Time Pads: Decryption

Bob

K	0	1	1	0	0	1	0	1	0	1	1	1
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
C	1	1	1	1	1	1	0	0	0	0	1	1
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
M	1	0	0	1	1	0	0	1	0	1	0	0

Decryption algorithm: XOR each bit of K with the matching bit in C.

One-Time Pad

- KeyGen()
  - Randomly generate an *n*-bit key, where *n* is the length of your message
    - Recall: For today, we assume that Alice and Bob can securely share this key
    - For one-time pads, we generate a *new* key for every message
- Enc(K, M) = $K \oplus M$
  - Bitwise XOR M and K to produce C
    - In other words: XOR the *i*th bit of the plaintext with the *i*th bit of the key.
    - $C_i = K_i \oplus M_i$
  - Alice and Bob use a different key for each encryption (this is the “one-time” in one-time pad).
- Dec(K, C) = $K \oplus C$
  - Bitwise XOR C and K to produce M
    - $M_i = K_i \oplus C_i$

One-Time Pad: Correctness

- Correctness: If we encrypt and then decrypt, we should get the original message back

$\text{Enc}(K, M)$	$= K \oplus M$	Definition of encryption
$\text{Dec}(K, \text{Enc}(K, M))$	$= \text{Dec}(K, K \oplus M)$	Decrypting the ciphertext
	$= K \oplus (K \oplus M)$	Definition of decryption
	$= M$	XOR property

One-Time Pad: Security

- Recall our definition of confidentiality: The ciphertext should not give the attacker any additional information about the plaintext
- Recall our experiment to test confidentiality from earlier:
  - Alice has encrypted and sent either M_0 or M_1
  - Eve knows either M_0 or M_1 was sent, but doesn't know which
  - Eve reads the ciphertext and tries to guess which message was sent
  - If the probability that Eve correctly guesses which message was sent is 1/2, then the encryption scheme is confidential

One-Time Pad: Security

Possibility 0: Alice sends Enc(K, M_0)

The ciphertext is $C = K \oplus M_0$

Therefore, $K = C \oplus M_0$

Possibility 1: Alice sends Enc(K, M_1)

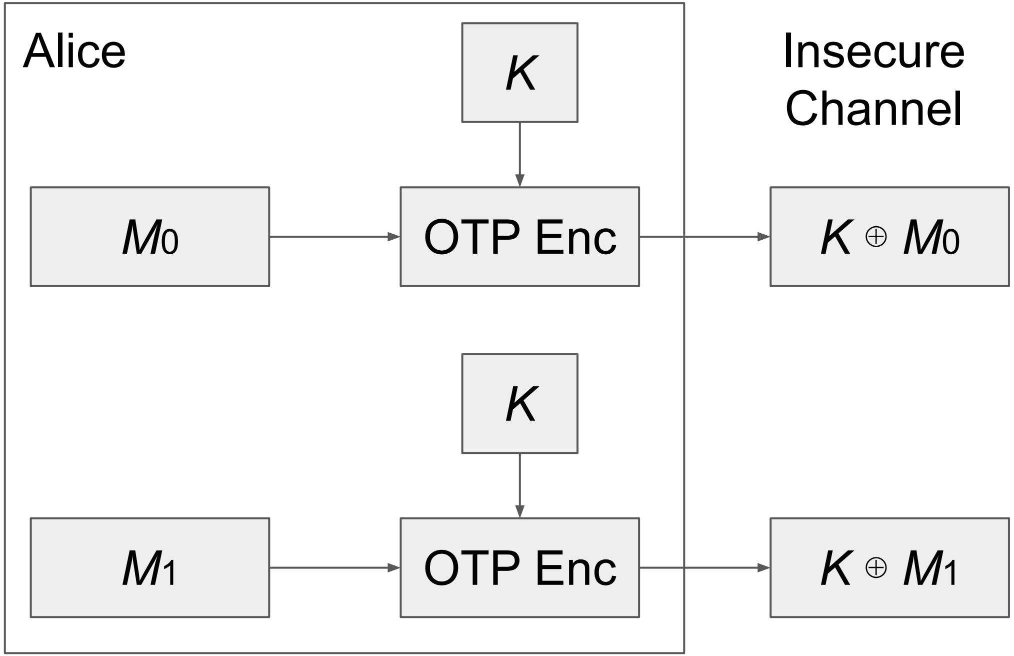
The ciphertext is $C = K \oplus M_1$

Therefore, $K = C \oplus M_1$

K was chosen randomly, so both possibilities are equally possible

Eve has learned no new information, so the scheme is **perfectly secure**

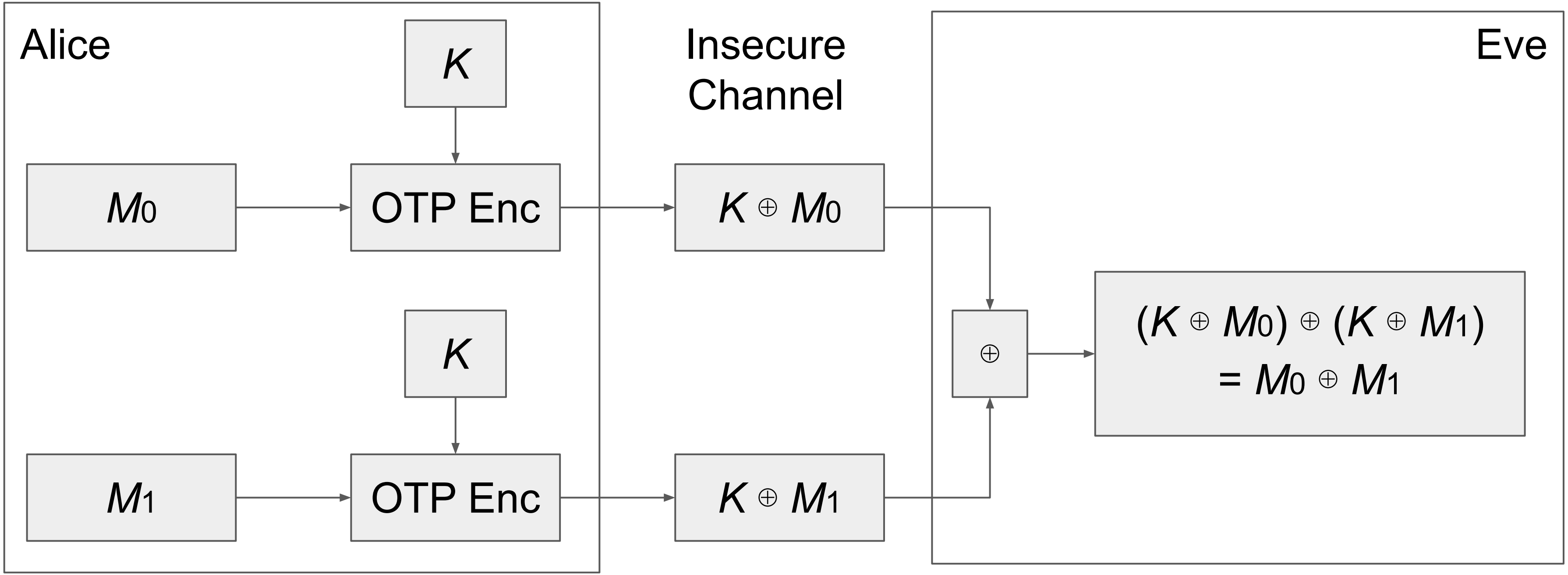
Two-Time Pads?



Eve sees two ciphertexts over the insecure channel.

What if we use the same key K to encrypt two different messages?

Two-Time Pads?



If Eve XORs the two ciphertexts, she learns $M_0 \oplus M_1$!

Two-Time Pads?

- What if we use the same key *twice*?
  - Alice encrypts M_0 and M_1 with the same key
  - Eve observes $K \oplus M_0$ and $K \oplus M_1$
  - Eve computes $(K \oplus M_0) \oplus (K \oplus M_1) = M_0 \oplus M_1$
    - Recall the XOR property: the K's cancel out
- Eve has learned $M_0 \oplus M_1$. This is partial information about the messages!
  - In words, Eve knows which bits in M_0 match bits in M_1
  - If Eve knows M_0, she can deduce M_1 (and vice-versa)
  - Eve can also guess M_0 and check that M_1 matches her guess for M_0
- Result: One-time pads are not secure if the key is reused
  - Alice and Bob must use a different key for every message!

Impracticality of One-Time Pads

- Problem #1: Key generation
  - For security to hold, keys must be randomly generated for every message, and never reused
  - Randomness is expensive, as we'll see later
- Problem #2: Key distribution
  - To communicate an *n*-bit message, we need to securely communicate an *n*-bit key first
  - But if we have a way to securely communicate an *n*-bit key, we could have communicated the message directly!
- Only practical application: Communicate keys in advance
  - You have a secure channel now, but you won't have it later
  - Use the secure channel now to communicate keys in advance
  - Use one-time pad later to communicate over the insecure channel
  - And people can compute this by hand without computers!

One-Time Pads in Practice: Spies

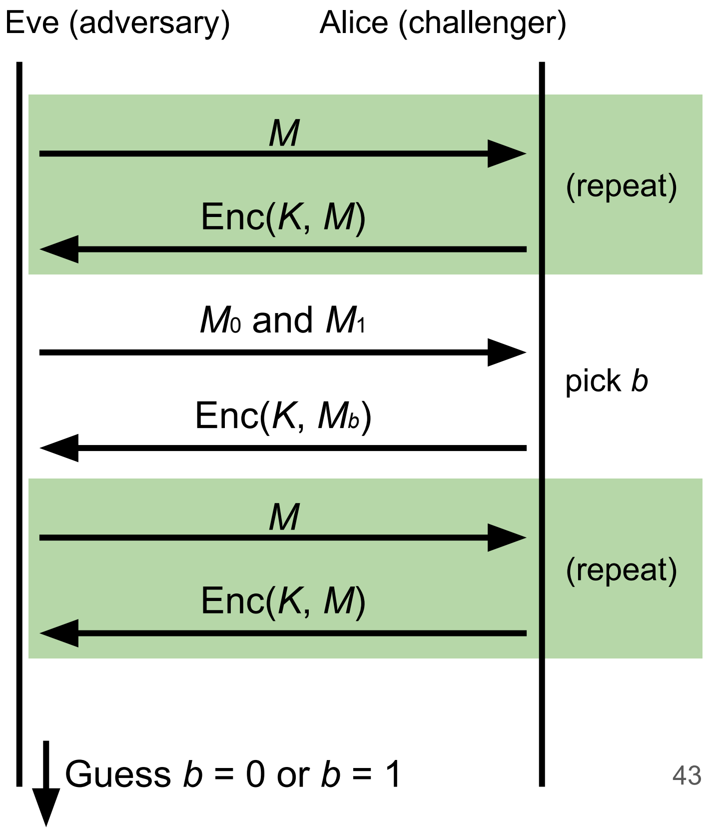
- At home base, the spy obtains a large amount of key material (e.g. a book of random bits)
- In the field, the spy listens for secret messages from their home country
  - There are shortwave and terrestrial radio “number stations”
  - At a regular time, a voice gets on the air and reads a series of numbers
  - If you don’t know the key, this looks like a meaningless sequence of random numbers
  - If you know the key, you can decrypt the spy message!
- What if you don’t want to send anything to any spies?
  - Read out a list of random numbers anyway
  - Because one-time pad leaks no information, an eavesdropper can’t distinguish between an encrypted message and random numbers!

Two-Time Pads in Practice: VENONA

- Soviet spies used one-time pads for communication from their spies in the US
- During WWII, the Soviets started reusing key material
  - Uncertain whether it was just the cost of generating pads or what…
- VENONA was a US cryptanalysis project designed to break these messages
  - Included confirming/identifying the spies targeting the US Manhattan project
  - Project continued until 1980!
- Not declassified until 1995!
  - So secret even President Truman wasn't informed about it
  - The Soviets found out about it in 1949 through their spy Ken Philby, but their one-time pad reuse was fixed after 1948 anyway
- **Takeaway**: Otherwise-secure cryptographic systems can fail very badly if used improperly!

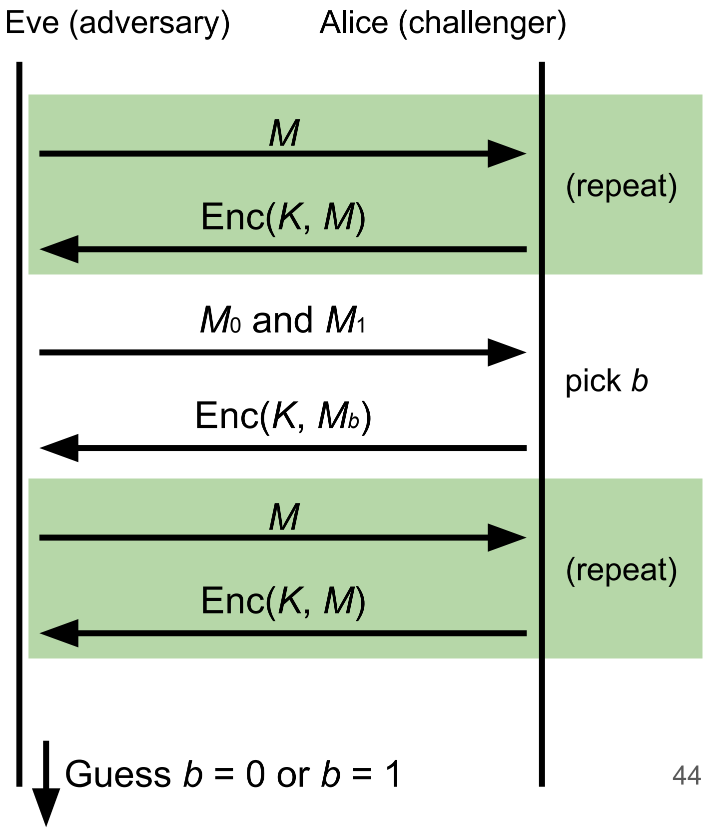
One-Time Pads: Insecurity under IND-CPA

1. Eve can skip the first query phase…
2. What pair of messages should Eve send to Alice in the challenge phase?
   - Send $M_0 = 0^k$, $M_1 \neq M_0$
   - M_0 is a string of k 0's, M_1 is anything else
3. Alice chooses M_b to encrypt and sends the message back. Eve receives $C_b = M_b \oplus K$
4. What messages should Eve for Alice to encrypt in the query phase?
   - Send $M = M_0$ and receive $C = M_0 \oplus K$
5. How can Eve know which message Alice encrypted?
   - If $C_b = C$, then Alice encrypted M_0. Otherwise, Alice encrypted M_1!
   - Eve wins the IND-CPA game with probability 1!



One-Time Pads: Insecurity under IND-CPA

- Result: One-time pad is **not** IND-CPA secure
  - Yes, it is **perfectly secure** under some models and **very insecure** under other models!
  - This is the same reason as why two-time pad is insecure



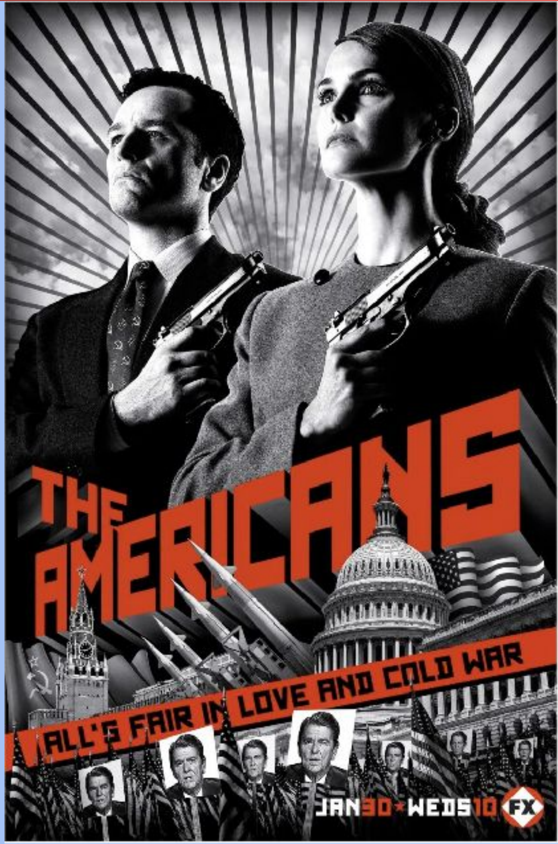
Traffic Analysis & Side Channels

Traffic Analysis & Side Channels

- **Traffic analysis**: Analyzing who is talking to whom and when
  - The encryption schemes we'll be studying do not hide the identity of who you're talking to
  - The information used for this analysis is often referred to as **metadata**: Data *about* the message and its context
- **Side channels**: Information about the plaintext revealed as a result of the *implementation* of the scheme, not the scheme itself
  - Modern crypto systems are usually broken through side channels

Traffic Analysis & Side Channels in Practice: Spies

- In the 1990s, there were some Russian spies in the US
  - The TV series “The Americans” was based on this incident
- A Cuban number station had a bug: some nights it never broadcasted “9”
  - Normally, 0–9 would be equally frequent
- It turns out this corresponded to when the Russian spies were on vacation
  - The way that random numbers were generated for cover traffic had a bug in it
  - The FBI used this as part of their investigation
- **Takeaway**: Secure algorithms can be broken in insecure implementations, leaking information



Summary: IND-CPA and One-Time Pads

- IND-CPA security
  - Even if Eve can trick Alice into encrypting some messages of Eve's choosing, given the encryption of either M_0 or M_1, Eve cannot distinguish which message was sent with probability greater than 1/2
  - We can use the IND-CPA game to test for IND-CPA security
  - Edge cases:
    - IND-CPA secure schemes can leak length
    - Eve is limited to polynomial-time algorithms, and must have a non-negligible advantage to win
- One-time pads
  - Symmetric encryption scheme: Alice and Bob share a secret key
  - Encryption and decryption: Bitwise XOR with the key
  - Can be perfectly secure, but also IND-CPA insecure
    - No information leakage if the key is never reused
    - Information leaks if the key is reused
  - Impractical for real-world usage, unless you're a spy
- Side channels: Information can be leaked due to *implementation* issues

Block Ciphers

Textbook Chapter 6.4 & 6.5

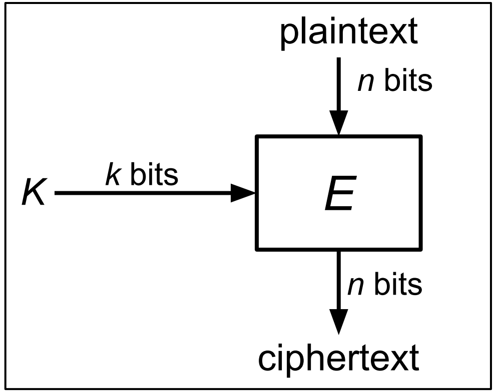
Cryptography Roadmap

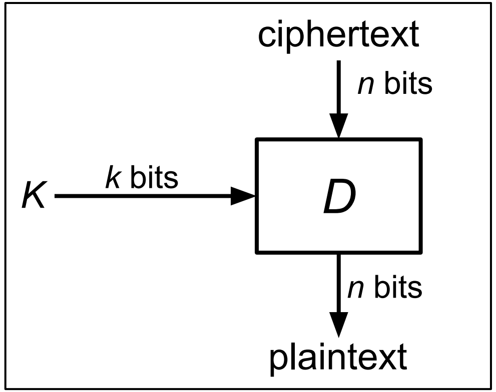
	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)
- Password management

Block Ciphers: Definition

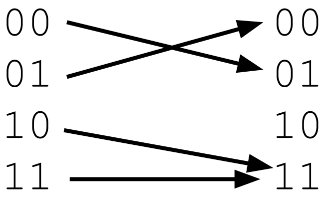
- **Block cipher**: An encryption/decryption algorithm that encrypts a fixed-sized block of bits
- $E_K(M) \rightarrow C$: Encryption
  - Inputs: k-bit key K and an n-bit plaintext M
  - Output: An n-bit ciphertext C
  - Sometimes written as: $\{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
- $D_K(C) \rightarrow M$: Decryption
  - Inputs: a k-bit key, and an n-bit ciphertext C
  - Output: An n-bit plaintext
  - Sometimes written as: $\{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
  - The inverse of the encryption function
- Properties
  - **Correctness**: E_K is a permutation, D_K is its inverse
  - **Efficiency**: Encryption/decryption should be fast
  - **Security**: E behaves like a random permutation



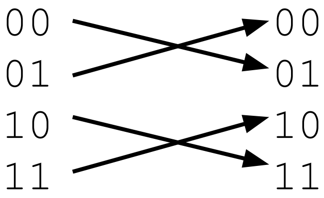


Block Ciphers: Correctness

- $E_K(M)$ must be a **permutation** (**bijective function**) on n-bit strings
  - Each input must correspond to exactly one unique output
- Intuition
  - Suppose $E_K(M)$ is not bijective
  - Then two inputs might correspond to the same output: $E(K, x_1) = E(K, x_2) = y$
  - Given ciphertext y, you can't uniquely decrypt. $D(K, y) = x_1$? $D(K, y) = x_2$?



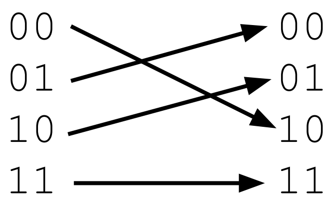
Not bijective: Two inputs encrypt to the same output

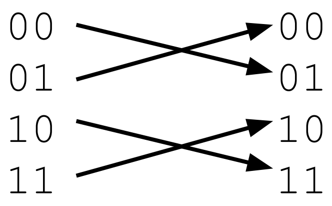


Bijective: Each input maps to exactly one unique output

Block Ciphers: Security

- A secure block cipher behaves like a randomly chosen permutation permutation from the set of all permutations on *n*-bit strings
  - A random permutation: Each *n*-bit input is mapped to one randomly-chosen *n*-bit output
- Defined by a distinguishing game
  - Eve gets two boxes: One is a randomly chosen permutation, and one is E_K with a randomly chosen key K
  - Eve should not be able to tell which is which with probability > 1/2





One of these is E_K with a randomly chosen K, and the other one is a randomly chosen permutation. Eve can't distinguish them.

Block ciphers: Brute-force attacks?

- How hard is it to run a brute-force attack on a 128-bit key?
  - We have to try 2^{128} possibilities. How big is 2^{128}?
- Handy approximation: $2^{10} \approx 10^3$
  - $2^{128} = 2^{10*12.8} \approx (10^3)^{12.8} \approx (10^3)^{13} = 10^{39}$
- Suppose we have massive hardware that can try 10^9 (1 billion) keys in 1 nanosecond (a billionth of a second). That's 10^{18} keys per second
  - We'll need $10^{39} / 10^{18} = 10^{21}$ seconds. How long is that?
  - One year $\approx 3\times10^7$ seconds
  - 10^{21} seconds / $3\times10^7 \approx 3\times10^{13}$ years $\approx$ 30 trillion years
- **Takeaway**: Brute-forcing a 128-bit key takes astronomically long. Don't even try.

Block ciphers: Brute-force attacks?

- How hard is it to run a brute-force attack on a 256-bit key in the same time?
  - We need 10^{52} of the brute-force devices from before
  - If each brute-force device from before is 1 cubic millimeter, this would take 10^{43} cubic meters of space
  - That's the volume of 7×10^{15} suns!
  - For reference, the Milky Way galaxy has just 10^{11} stars
- **Takeaway**: Brute-force attacks on modern block ciphers are not possible, assuming the key is random and secret
  - 128-bit key? Definitely not happening.
  - 256-bit key? Lol no.

Block Ciphers: Efficiency

- Encryption and decryption should be computable in microseconds
  - Formally: KeyGen(), Enc(), and Dec(), should not take exponential time
- Block cipher algorithms typically use operations like XOR, bit-shifting, and small table lookups
  - Very fast on modern processors
- Modern CPUs provide dedicated hardware support for block ciphers

DES (Data Encryption Standard)

- Designed in late 1970s
- Block size 64 bits (n = 64)
- Key size 56 bits (k = 56)
- NSA influenced two facets of its design
  - Altered some subtle internal workings in a mysterious way
  - Reduced key size from 64 bits to 56 bits
  - Made brute force attacks feasible for an attacker with massive computational resources (by 1970s standards)
- The algorithm remains essentially unbroken 40 years later
  - The NSA's tweaking hardened it against an attack publicly revealed a decade later
- However, modern computer speeds make it completely unsafe due to small key size
  - ~6.4×10^{16}, say 10^{10} tries per second on my single desktop computer's Nvidia graphics card: Takes ~6.4×10^{6} seconds or ~70 days

AES (Advanced Encryption Standard)

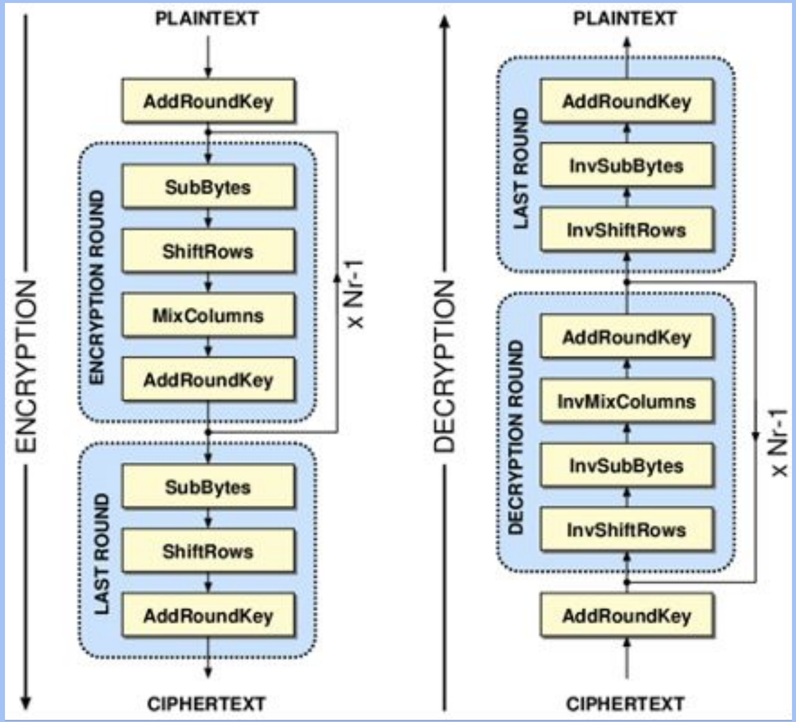
- 1997–2000: NIST (National Institute of Standards and Technology) in the US held a competition to pick a new block cipher standard
  - One of the finalists, Twofish, was designed by Berkeley professor and occasional CS 161 instructor David Wagner!
- Out of the 5 finalists:
  - Rijndael, Twofish, and Serpent had really good performance
  - RC6 had okay performance
  - Mars had ugly performance
- On any given computing platform, Rijndael was *never* the fastest
- But on every computing platform, Rijndael was *always* the second-fastest
  - Twofish and Serpent each had at least one compute platform they were bad at
- Rijndael was selected as the new block cipher standard

AES (Advanced Encryption Standard)

- Key size 128, 192, or 256 bits (k = 128, 192, or 256)
    - Actual cipher names are AES-128, AES-192, and AES-256
    - Paranoid people like the NSA use AES-256 keys, but AES-128 is just fine in practice
- Block size 128 bits (n = 128)
    - Note: The block size is still always 128 bits, regardless of key size
- You don't need to know how AES works, but you do need to know its parameters
    - here's a comic

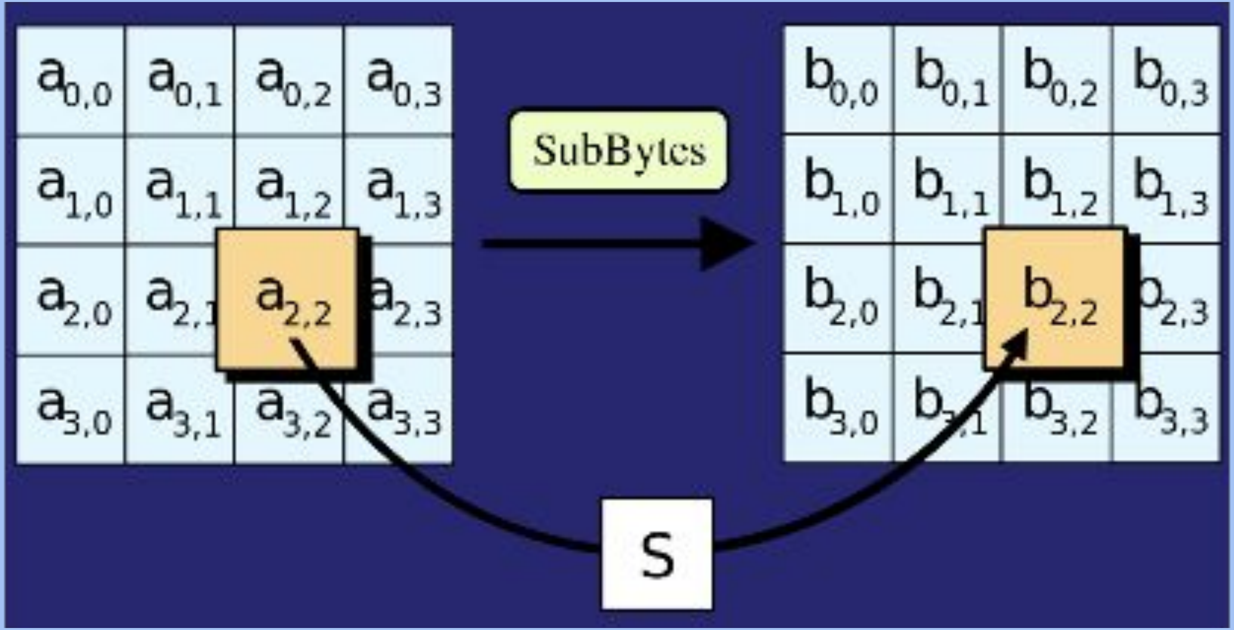
AES Algorithm

- Different key sizes use different numbers of rounds
  - 10 rounds for 128-bit keys
  - 12 rounds for 192-bit keys
  - 14 rounds for 256-bit keys
- Each round uses its own "round key" derived from the cipher key
- Each round:
  - SubBytes()
  - ShiftRows()
  - MixColumns() (if not last round)
  - AddRoundKey()



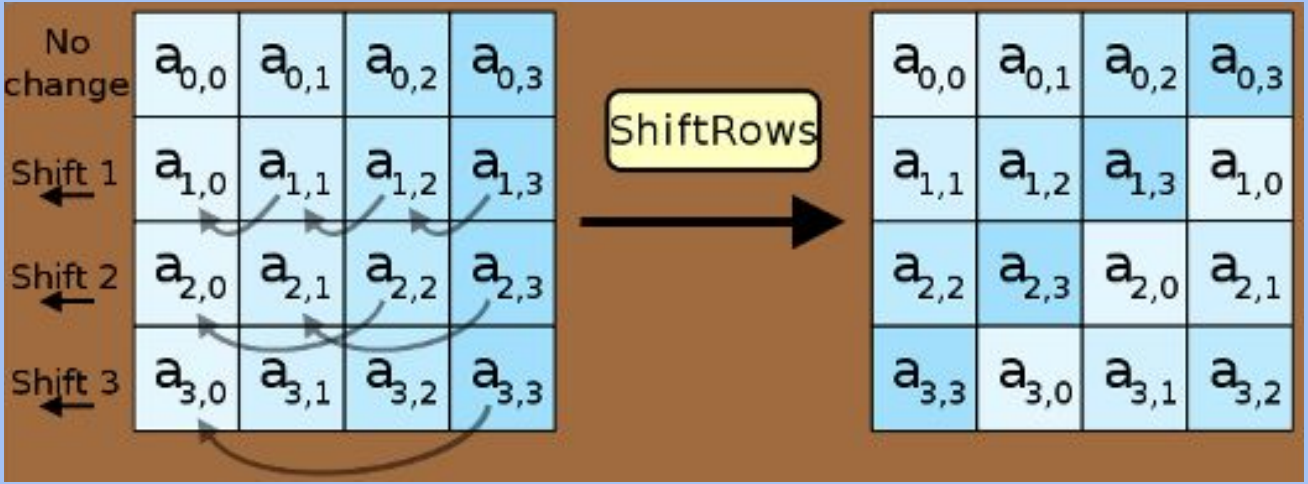
AES Algorithm: SubBytes()

- Replace each byte in the block with another byte using an 8-bit substitution box



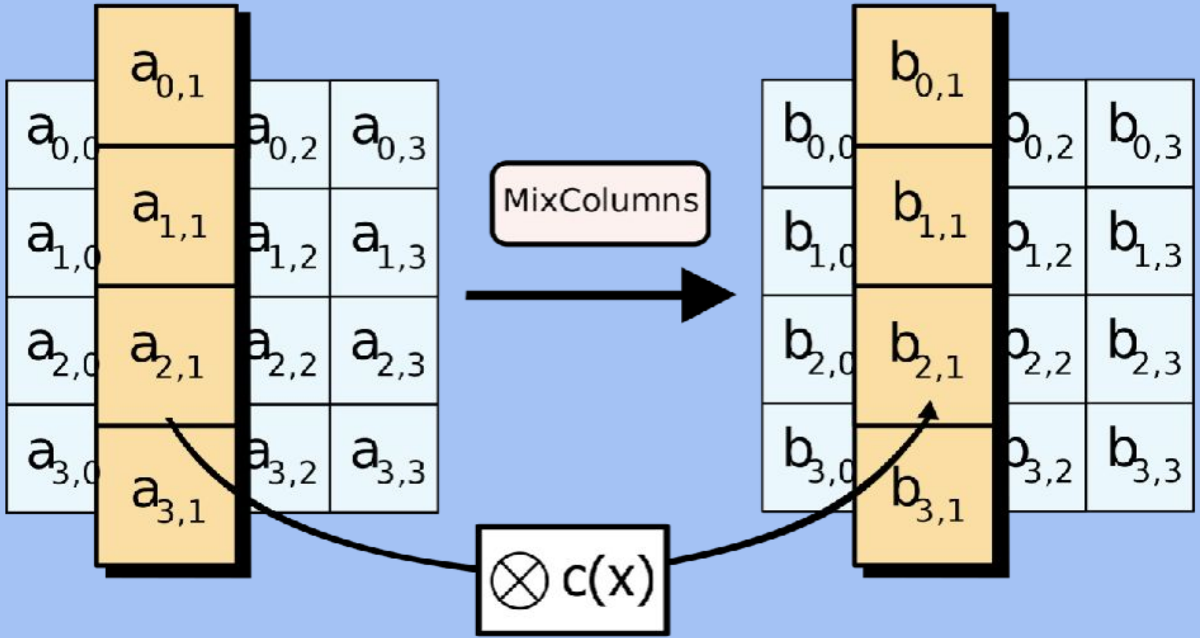
AES Algorithm: ShiftRows()

- Cyclically shifts the bytes in each row by a certain offset
- The number of places each byte is shifted differs for each row



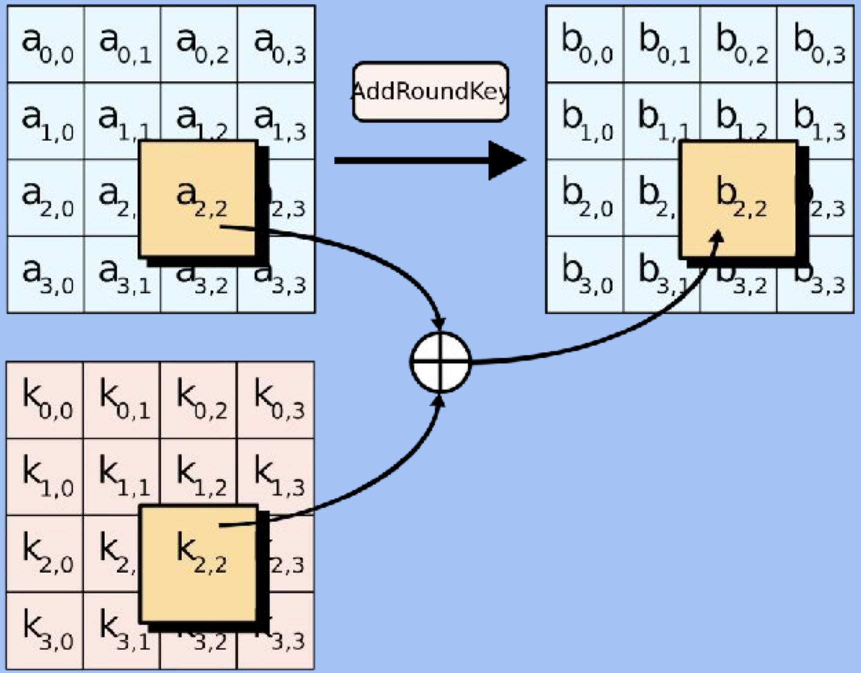
AES Algorithm: MixColumns()

- Treats the 16-byte block as a 4 × 4 matrix and multiply it by by another matrix



AES Algorithm: AddRoundKey()

- XOR the 16-byte block with the 16-byte round key

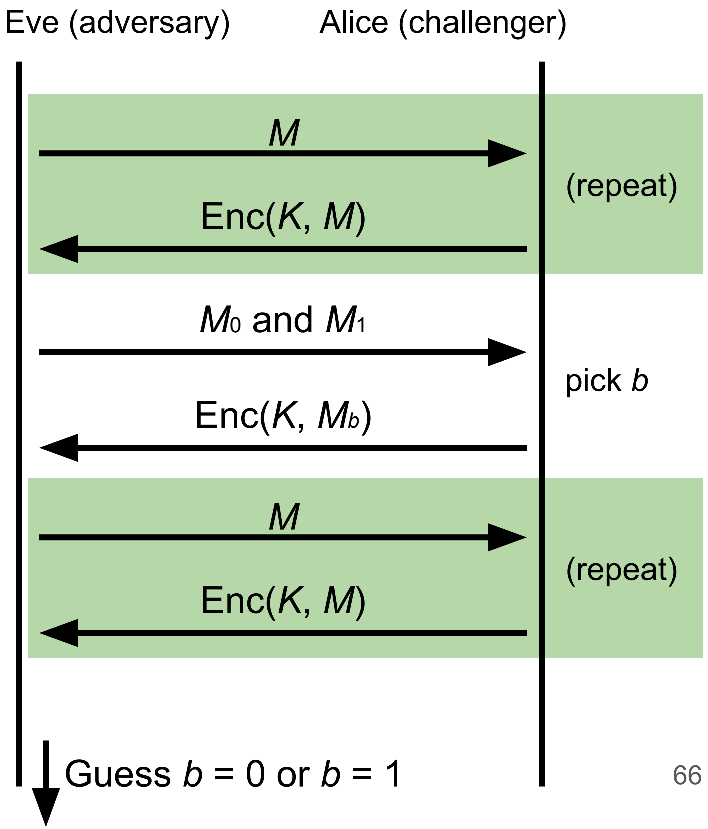


AES (Advanced Encryption Standard)

- There is no formal proof that AES is secure (indistinguishable from a random permutation)
- However, in 20 years, nobody has been able to break it, so it is *assumed* to be secure
  - The NSA uses AES-256 for secrets they want to keep secure for the 40 years (even in the face of unknown breakthroughs in research)
- **Takeaway**: AES is the modern standard block cipher algorithm
  - The standard key size (128 bits) is large enough to prevent brute-force attacks

Are Block Ciphers IND-CPA Secure?

- Consider the following adversary:
  - Eve sends two different messages M_0 and M_1
  - Eve receives either $E_K(M_0)$ or $E_K(M_1)$
  - Eve requests the encryption of M_0 again
  - Strategy: If the encryption of M_0 matches what she received, guess b = 0. Else, guess b = 1.
- Eve can win the IND-CPA game with probability 1!
  - Block ciphers are not IND-CPA secure



Issues with Block Ciphers

- Block ciphers are not IND-CPA secure, because they're deterministic
  - A scheme is **deterministic** if the same input always produces the same output
  - No deterministic scheme can be IND-CPA secure because the adversary can always tell if the same message was encrypted twice
- Block ciphers can only encrypt messages of a fixed size
  - For example, AES can only encrypt-decrypt 128-bit messages
  - What if we want to encrypt something longer than 128 bits?
- To address these problems, we'll add **modes of operation** that use block ciphers as a building block!

Summary: Block Ciphers

- Encryption: input a *k*-bit key and *n*-bit plaintext, receive *n*-bit ciphertext
- Decryption: input a *k*-bit key and *n*-bit ciphertext, receive *n*-bit plaintext
- Correctness: when the key is fixed, $E_K(M)$ should be bijective
- Security
  - Without the key, $E_K(m)$ is computationally indistinguishable from a random permutation
  - Brute-force attacks take astronomically long and are not possible
- Efficiency: algorithms use XORs and bit-shifting (very fast)
- Implementation: AES is the modern standard
- Issues
  - Not IND-CPA secure because they're deterministic
  - Can only encrypt *n*-bit messages