# One-Time Pads and Block Ciphers

CSE 405 Lecture 3

# Summary



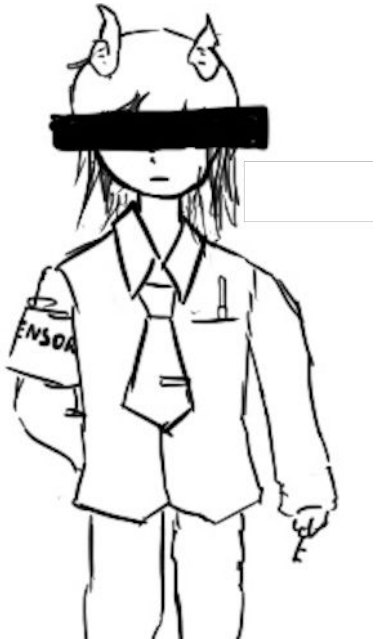
- What's cryptography?
  - Communicating securely over insecure channels
  - You should never write your own crypto! Use existing libraries instead.
- Definitions
  - Alice and Bob want to send messages over an insecure channel. Eve can read anything sent over the insecure channel. Mallory can read or modify anything sent over the insecure channel.
  - We want to ensure confidentiality (adversary can't read message), integrity (adversary can't modify message), and authenticity (prove message came from sender)
  - Crypto uses secret keys. Kerckhoff's principle says to assume the attacker knows the entire system, except the secret keys.
  - There are several different threat models. We'll focus on the chosen plaintext attack, where Eve tricks Alice into encrypting some messages.

# Summary



- 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



Textbook Chapter 6.2 & 6.3

# Cryptography Roadmap



| | Symmetric-key | Asymmetric-key |
|---|---|---|
| Confidentiality | • One-time pads<br>• Block ciphers with chaining modes (e.g. AES-CBC)<br>• Stream ciphers | • RSA encryption<br>• 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

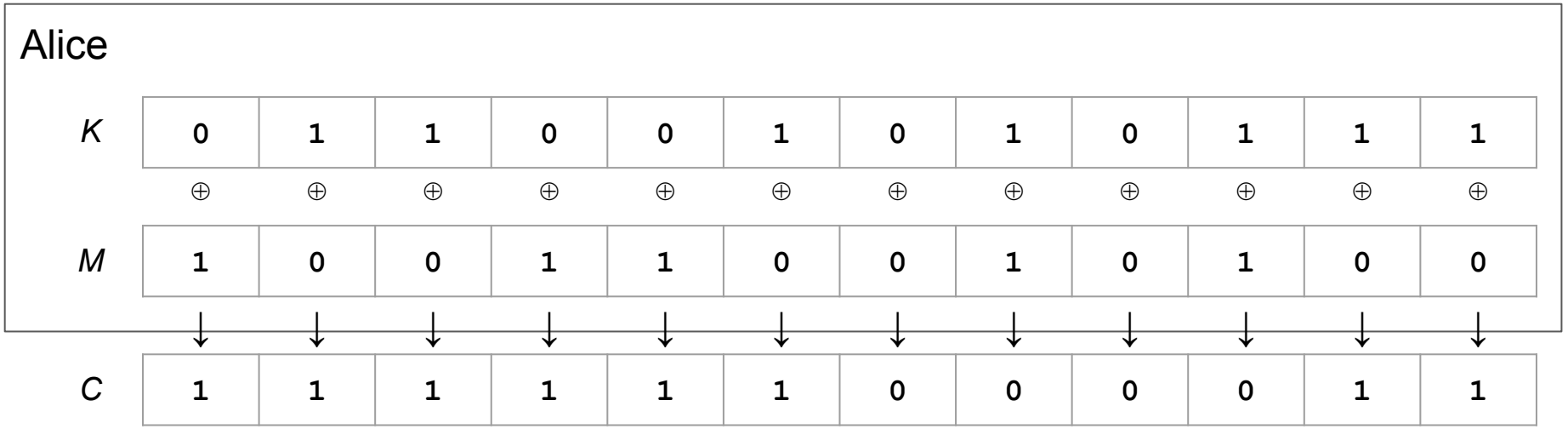
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *K* | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| *M* | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

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






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

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

# 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 $C = K \oplus M_1$

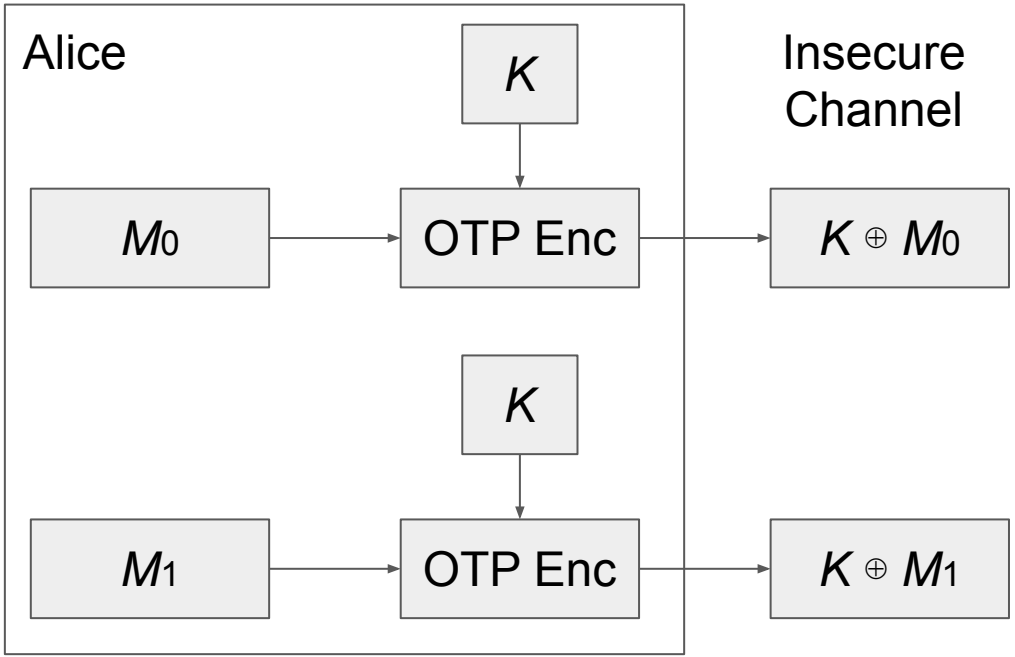
Therefore, $K = C \oplus M_1$

*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?



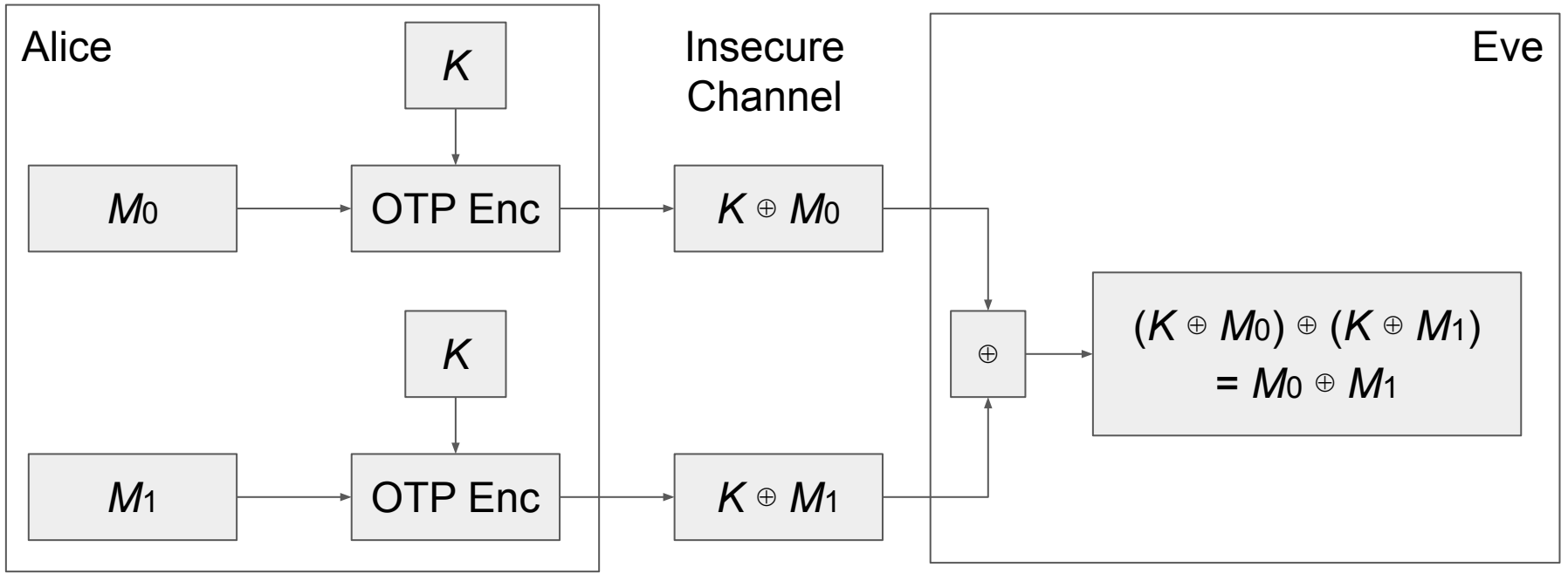



Eve sees two ciphertexts over the insecure channel.

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!

# Block Ciphers

Textbook Chapter 6.4 & 6.5

# Cryptography Roadmap



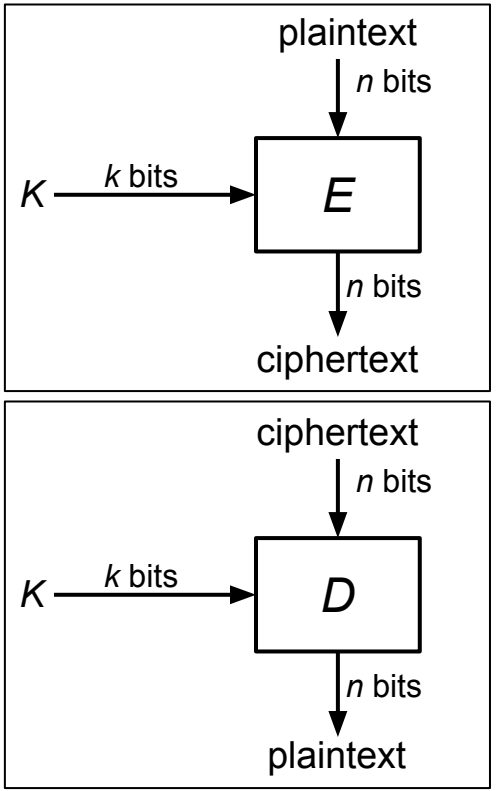
| | Symmetric-key | Asymmetric-key |
|---|---|---|
| Confidentiality | • One-time pads<br>• Block ciphers with chaining modes (e.g. AES-CBC)<br>• Stream ciphers | • RSA encryption<br>• ElGamal encryption |
| Integrity, Authentication | • MACs (e.g. HMAC) | • Digital signatures (e.g. RSA signatures) |

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

# Block Ciphers: Definition



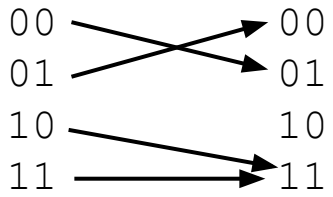
- **Block cipher**: An encryption/decryption algorithm that encrypts a fixed-sized block of bits
- $E_K(M) \rightarrow C$: Encryption
  - Inputs: $k$-bit key $K$ and an $n$-bit plaintext $M$
  - Output: An $n$-bit ciphertext $C$
  - Sometimes written as: $\{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
- $D_K(C) \rightarrow M$: 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



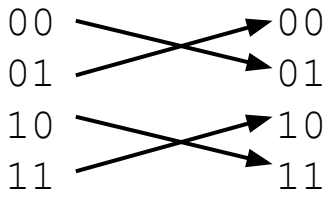
# Block Ciphers: Correctness



- $E_K(M)$ 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 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 $\approx 3 \times 10^7$ seconds
  - $10^{21}$ seconds / $3 \times 10^7 \approx 3 \times 10^{13}$ years ≈ 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 $7 \times 10^{15}$ 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
  - ~$6.4 \times 10^{16}$, say $10^{10}$ tries per second on my single desktop computer's Nvidia graphics card: Takes ~$6.4 \times 10^{6}$ 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)

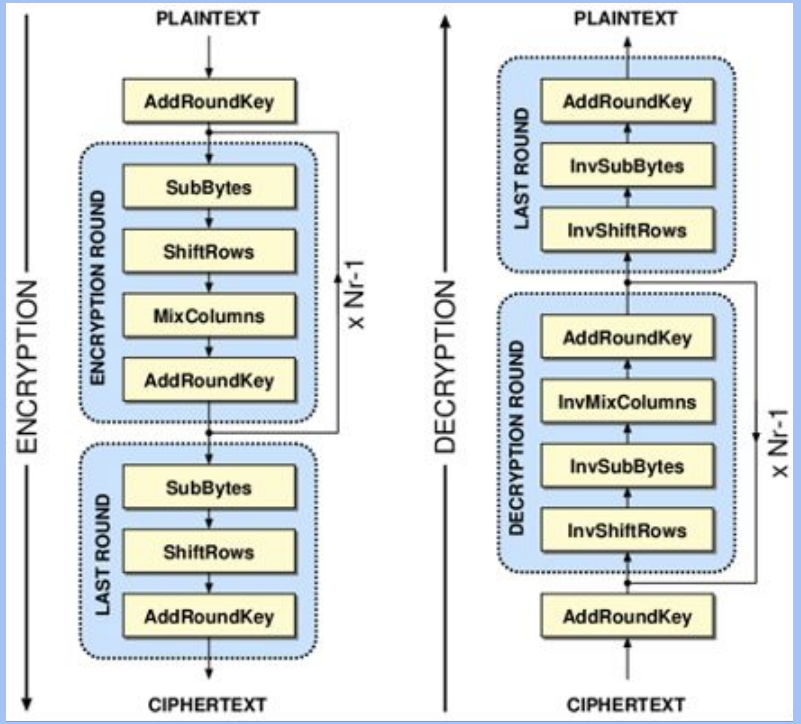


- Key size 128, 192, or 256 bits ($k$ = 128, 192, or 256)
  - Actual cipher names are AES-128, AES-192, and AES-256
  - Paranoid people like the NSA use AES-256 keys, but AES-128 is just fine in practice
- Block size 128 bits ($n$ = 128)
  - Note: The block size is still always 128 bits, regardless of 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



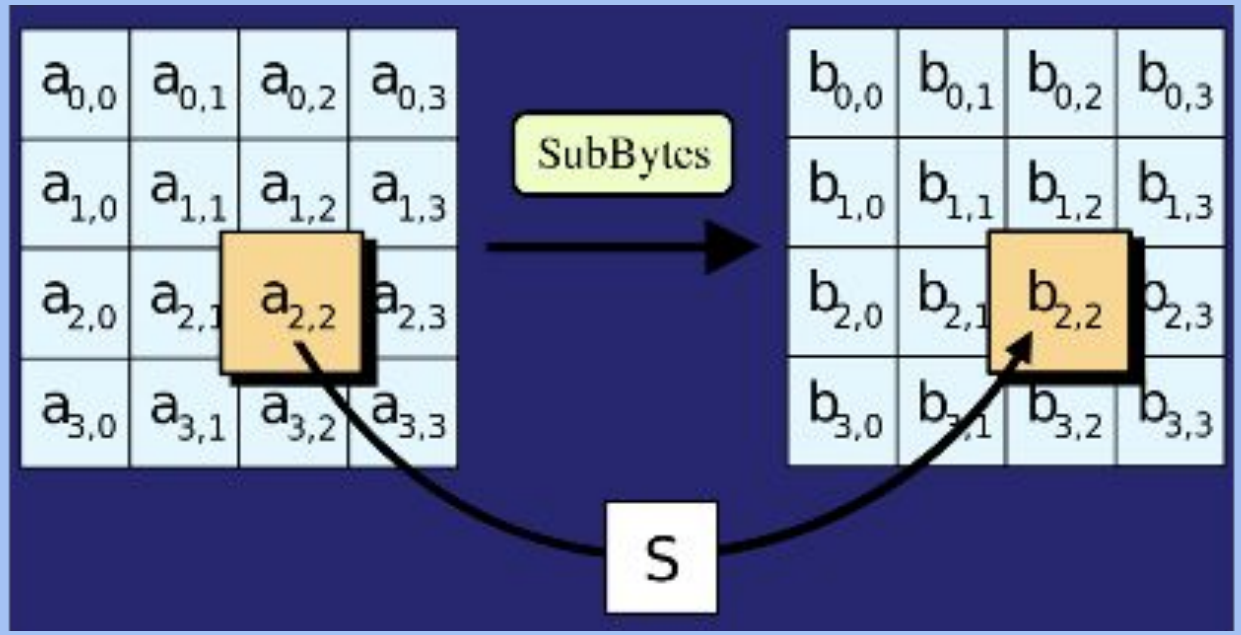
- Different key sizes use different numbers of rounds
  - 10 rounds for 128-bit keys
  - 12 rounds for 192-bit keys
  - 14 rounds for 256-bit keys
- Each round uses its own “round key” derived from the cipher key
- Each round:
  - SubBytes()
  - ShiftRows()
  - MixColumns() (if not last round)
  - AddRoundKey()



# AES Algorithm: SubBytes()



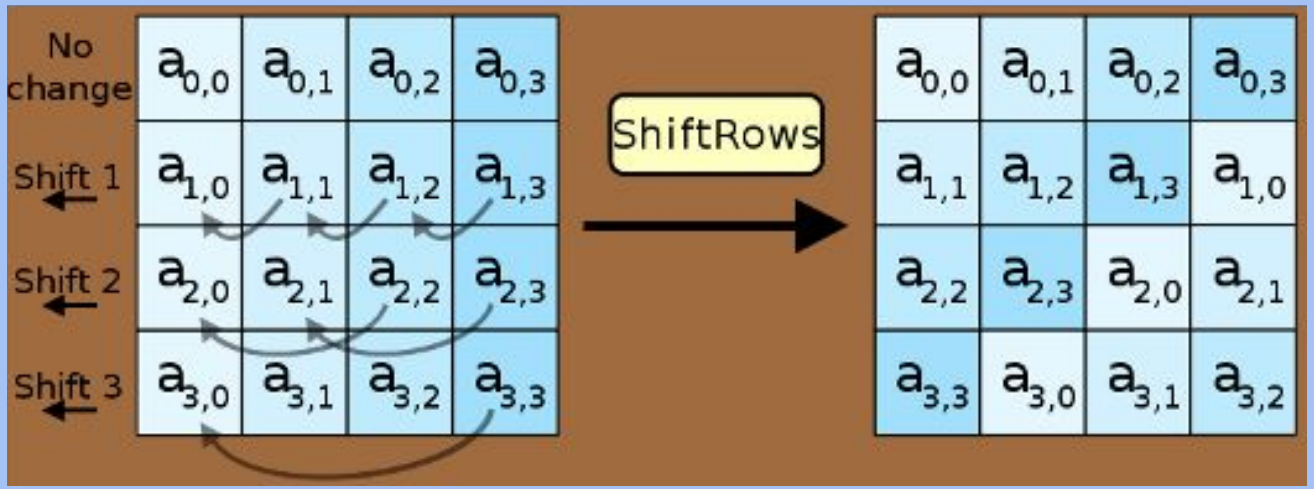
- Replace each byte in the block with another byte using an 8-bit substitution box



# AES Algorithm: ShiftRows()



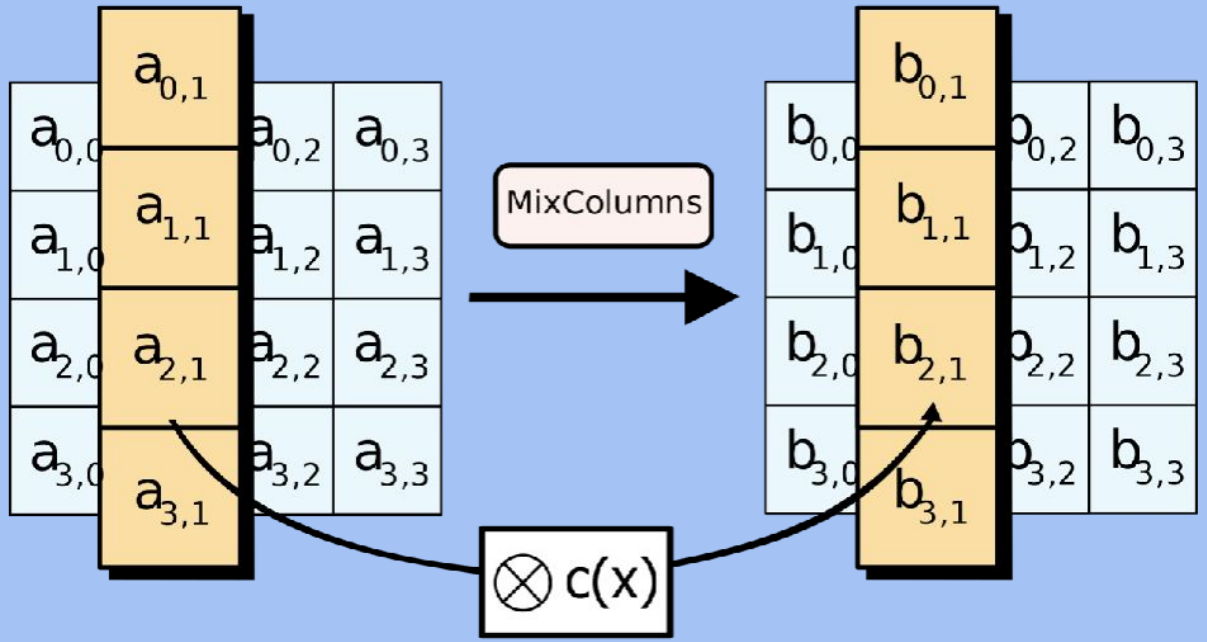
- Cyclically shifts the bytes in each row by a certain offset
- The number of places each byte is shifted differs for each row



# AES Algorithm: MixColumns()



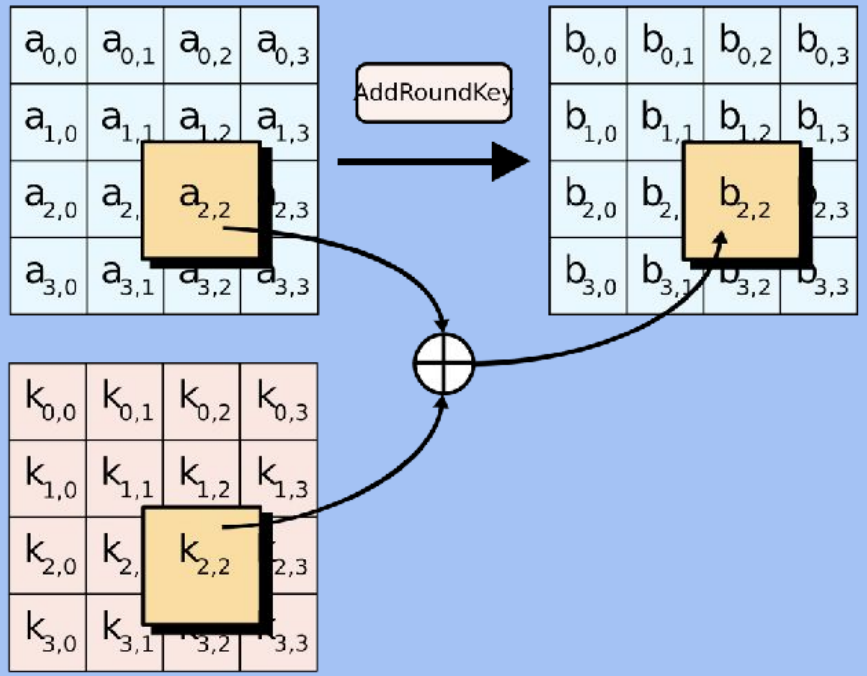
- Treats the 16-byte block as a 4 × 4 matrix and multiply it by by another matrix



# AES Algorithm: AddRoundKey()



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



# AES (Advanced Encryption Standard)

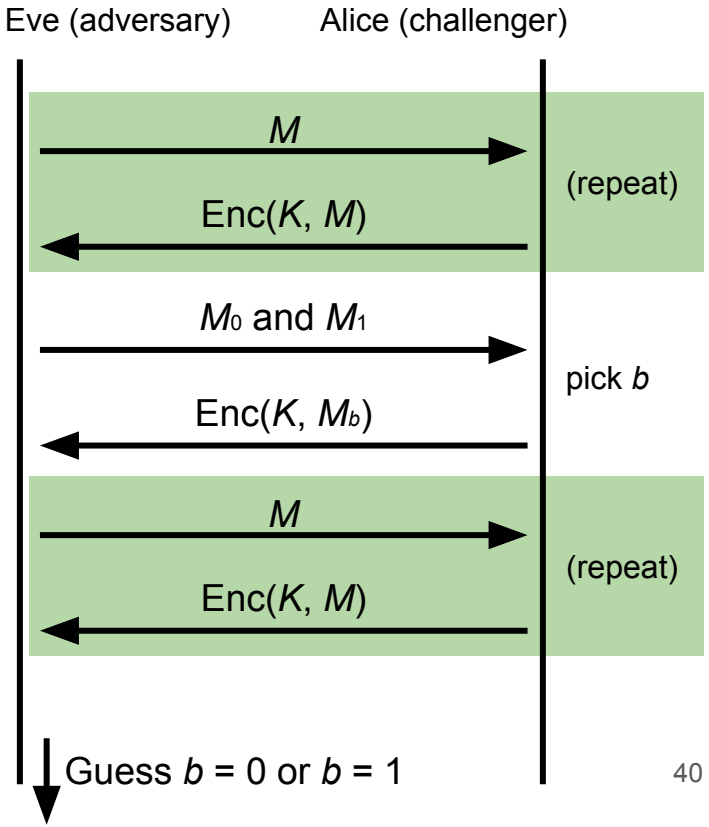


- There is no formal proof that AES is secure (indistinguishable from a random permutation)
- However, in 20 years, nobody has been able to break it, so it is *assumed* to be secure
  - The NSA uses AES-256 for secrets they want to keep secure for the 40 years (even in the face of unknown breakthroughs 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: One-Time Pads



- One-time pads
  - Symmetric encryption scheme: Alice and Bob share a secret key.
  - Encryption and decryption: Bitwise XOR with the key.
  - 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.

# 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