

CIS 4930: Secure IoT

Lecture 3

Prof. Kaushal Kafle

Class Notes

- **Homework 1 due today!**
- **Project proposals + team due Thursday!**
 - Plan to visit me personally with your group asap (within the next week).
- Class Website updated to include bug-bounty readings!
 - They are marked as **[BB]**
 - 11 papers in total
- Class schedule *slightly adjusted* for project deadlines
 - Both sections moved slightly later
 - First section deadline on 10/10
 - Second section deadline on 12/12
- See the updated syllabus and class schedule for details!

Cryptography



Crypto in IoT Apps

- Networks designed for data transport, *not for data confidentiality or privacy*
 - Internet eavesdropping is (relatively) easy
- Sensitive data is often stored locally on the device.
 - Other apps/root can get to it.
- Where have you seen crypto in practice?
- Crypto enables:
 - e-commerce and e-banking
 - confidential messaging
 - data transfer between IoT devices and cloud
 - protection of personal data
 - ...


Why is crypto useful?

MacBook-Pro-4 [redacted] \$ echo "Security is Fun" | netcat -v localhost 8080
localhost [127.0.0.1] 8080 (http-alt) open

No.	Time	Source	Destination	Protocol	Length	Info
160	2...	127.0.0.1	127.0.0.1	TCP	56	59584 → 8080 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSv...
161	2...	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8080 → 59584 [ACK] Seq=1 Ack=1 ...
162	2...	127.0.0.1	127.0.0.1	TCP	72	59584 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=408288 Len=...
163	2...	127.0.0.1	127.0.0.1	TCP	56	8080 → 59584 [ACK] Seq=1 Ack=17 Win=408256 Len=0 TS...
164	2...	127.0.0.1	127.0.0.1	TCP	68	59585 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
165	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59585 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
166	2...	127.0.0.1	127.0.0.1	TCP	68	59586 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
167	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59586 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
168	2...	127.0.0.1	127.0.0.1	TCP	68	59587 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
169	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59587 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
170	2...	127.0.0.1	127.0.0.1	TCP	68	59588 → 19536 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0...
171	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59588 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
172	2...	127.0.0.1	127.0.0.1	TCP	68	59589 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
173	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59589 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

- ▶ Frame 162: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0
- ▶ Null/Loopback
- ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- ▶ Transmission Control Protocol, Src Port: 59584 (59584), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 16

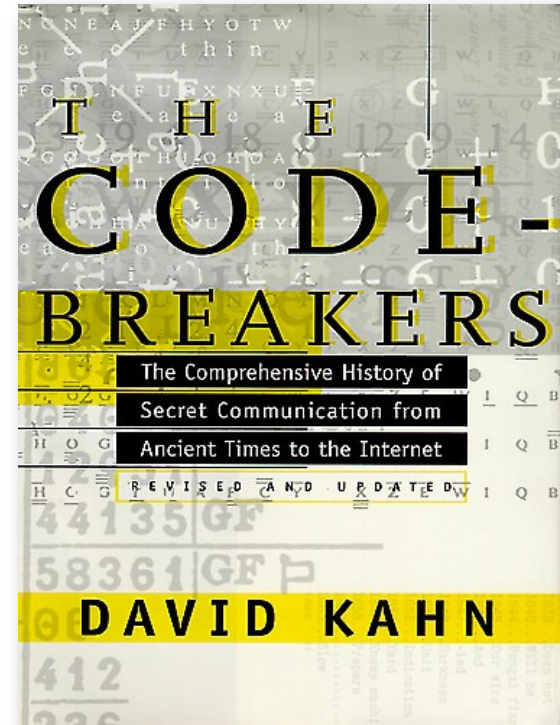
```
0000  02 00 00 00 45 00 00 44 2a cb 40 00 40 06 00 00  ....E..D *.@.@@...
0010  7f 00 00 01 7f 00 00 01 e8 c0 1f 90 44 fd 6b e1  ....D.k.
0020  d9 cd 38 c7 80 18 31 d7 fe 38 00 00 01 01 08 0a  ..8...1. .8.....
0030  6a 50 15 48 6a 50 15 47 53 65 63 75 72 69 74 79  jP.HjP.G Security
0040  20 69 73 20 46 75 6e 0a                               is Fun.
```



Packets: 199 · Displayed: 199 (100.0%) · Load time: 0:0.3 Profile: Default

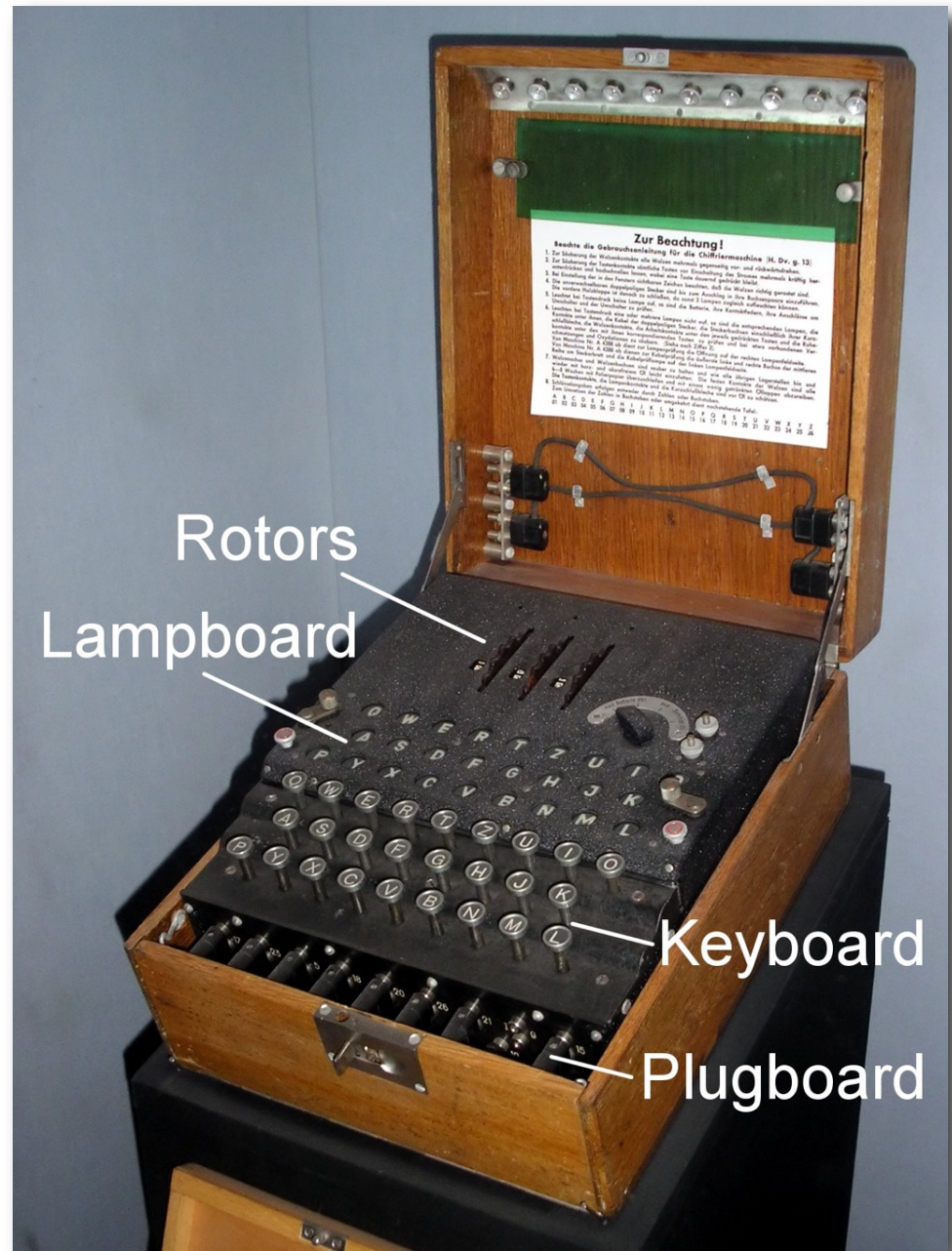
Cryptographic History

- hide secrets from your enemy
- ~4000 year old discipline
 - Egyptians' use of non-standard hieroglyphics
 - Spartans used *scytale* to perform transposition cipher
 - Italian Leon Battista Alberti (“founder of western cryptography”) invents polyalphabetic ciphers in 1466



Enigma

- German WWII encryption device
- Used polyalphabetic substitution cipher
- Broken by Allied forces
- Intelligence called Ultra
- Codebreaking at Bletchley Park
- See original at the International Spy Museum at DC



Some terminology

- **cryptosystem**: method of disguising (encrypting) plaintext messages so that only select parties can decipher (decrypt) the ciphertext
- **cryptography**: the art/science of developing and using cryptosystems
- **cryptanalysis**: the art/science of breaking cryptosystems
- **cryptology**: the combined study of cryptography and cryptanalysis

What can crypto do?

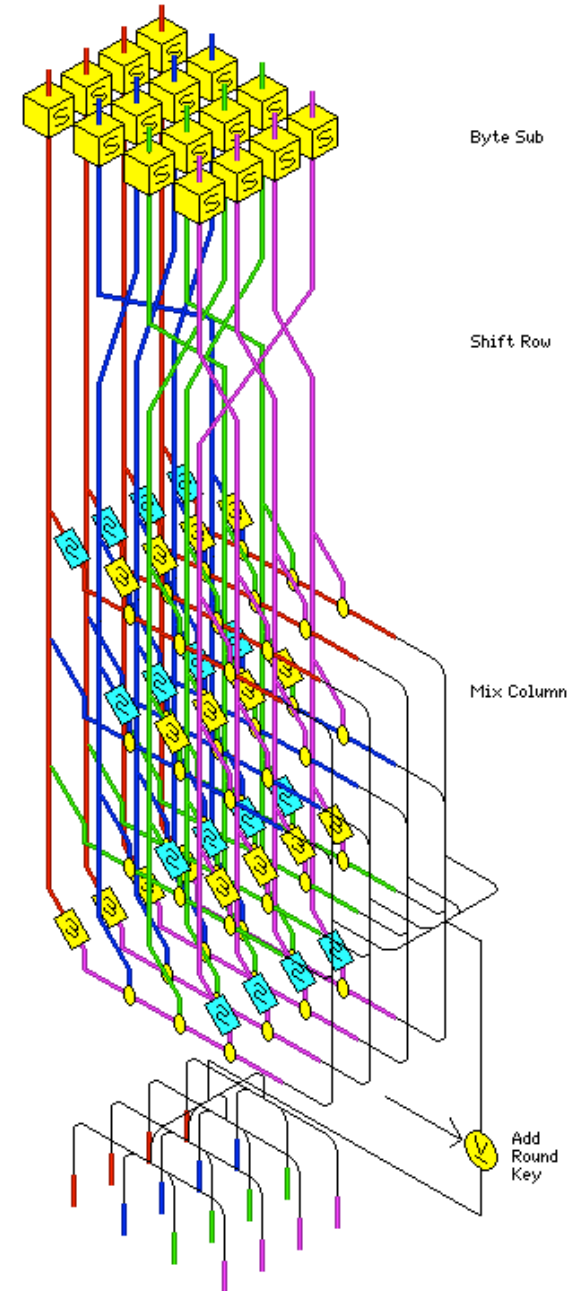
- **Confidentiality**
 - Keep data and communication secret
 - Encryption / decryption
- **Integrity**
 - Protect reliability of data against tampering
 - “Was this the original message that was sent?”
- **Authenticity**
 - Provide evidence that data/messages are from their purported originators
 - “Did Alice really send this message?”

cryptography < security

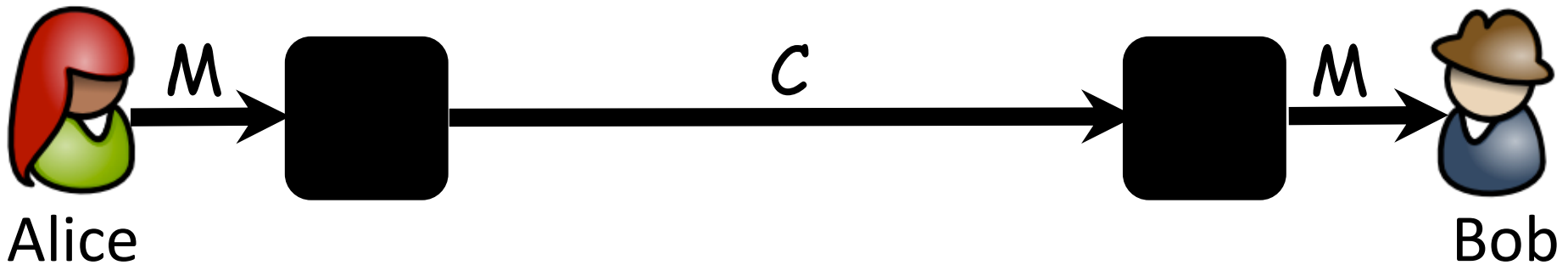
- Cryptography isn't the solution to security
 - Buffer overflows, worms, viruses, trojan horses, SQL injection attacks, cross-site scripting, bad programming practices, etc.
- It's a tool, not a solution
- **It is difficult to get right: choices.. choices....**
 - Choice of encryption algorithms (many tradeoffs)
 - Choice of parameters (key size, IV, ...)
 - Implementation (std. libraries work in most cases)
 - Hard to detect errors
 - Even when crypto fails, the program may still work
 - May not learn about crypto problems until after they've been exploited

Crypto is really, really, really, really, hard

- Task: develop a cryptosystem that is secure against all conceivable (and inconceivable) attacks, and will be for the foreseeable future
- If you are inventing your own crypto, you're doing it wrong
- Common security idiom: “no one ever got fired for using AES”



Encryption and Decryption



$$C = E(M)$$

$$M = D(C)$$

i.e.,

$$M = D(E(M))$$

where

M = plaintext

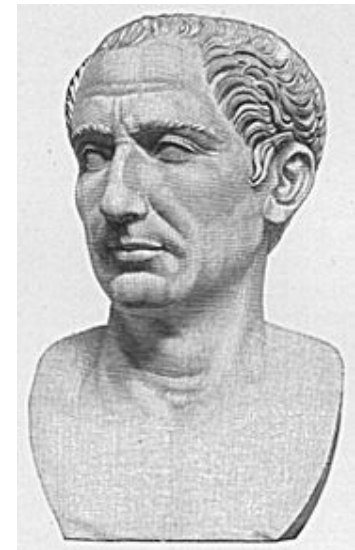
C = ciphertext

$E(x)$ = encryption function

$D(y)$ = decryption function

Let's look at some old
crypto algorithms
(don't use these)

Caesar Cipher



- A.K.A. Shift Cipher or ROT-x cipher (e.g., ROT-13)
- Used by Julius to communicate with his generals
- x is the key:
- Encryption: Right-shift every character by x: $c = E(x, p) = (p + x) \bmod 26$
- Decryption: Left-shift every character by x: $p = D(x, c) = (c - x) \bmod 26$

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

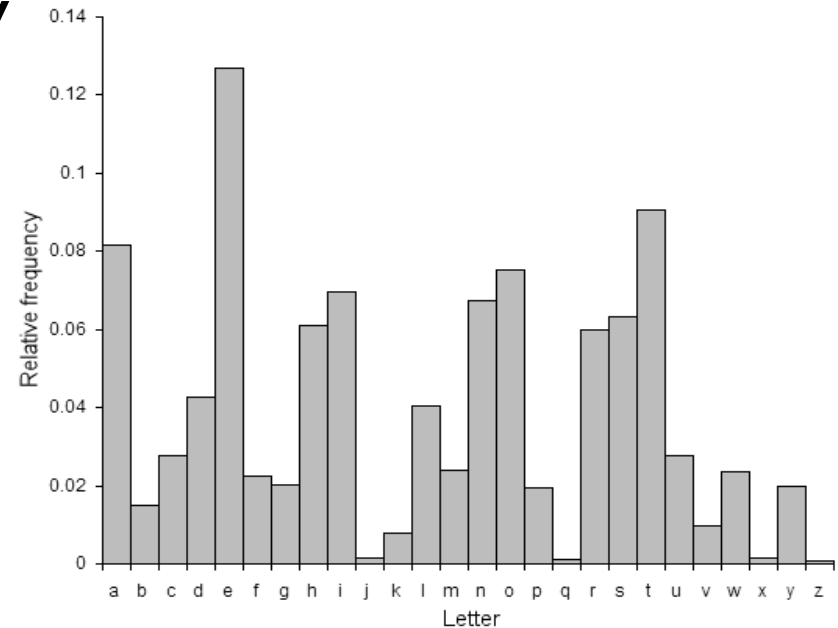
S E C U R I T Y A N D P R I V A C Y
V H F X U L W B D Q G S U L Y D F B

Cryptanalyze this ...

“GUVF VF N TERNG PYNFF”

Cryptanalyzing the Caesar Cipher

- Cryptanalysis:
 - **Brute-force attack:** try all 26 possible shifts (i.e., values of x)
 - **Frequency analysis:** look for frequencies of characters
 - Also, same plaintext (repetitions) *always* leads to same ciphertext, since monoalphabetic



Polyalphabetic Cipher

- Improves on the simple monoalphabetic ciphers by using multiple monoalphabetic substitutions
- Example: Vigenère Cipher
 - A set of Caesar Ciphers where each cipher is denoted by a key letter that designates the shift
 - The key repeats for the length of the message

key: deceptivedeceptivedeceptive
plaintext: wearediscoveredsaveyourself
ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

One-time Pads

- To produce ciphertext, XOR the plaintext with the **one-time pad** (secret key)
 - $E(M) = M \oplus \text{Pad}$
 - $D(E(M)) = E(M) \oplus \text{Pad}$
- Requires $\text{sizeof}(\text{pad}) == \text{sizeof}(\text{plaintext})$
- Offers **perfect secrecy**:
 - *a posteriori* probability of guessing plaintext given ciphertext equals the *a priori* probability
 - given a ciphertext without the pad, any plaintext of same length is possible input (there exists a corresponding pad)
 - $\Pr[M=m|C=c] = \Pr[M=m]$ (you learn nothing from the ciphertext)
- **Never reuse the pad (hence “one-time”)! Why not?**

XOR properties

- $A \oplus A = ?$

➤ 0

- $A \oplus 0 = ?$

➤ A

- $C1 = M1 \oplus \text{Pad}, C2 = M2 \oplus \text{Pad}$

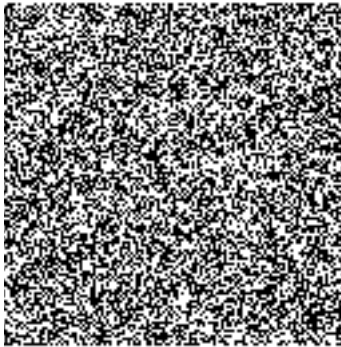
- $C1 \oplus C2 = ?$

$$M1 \oplus M2!$$

M1

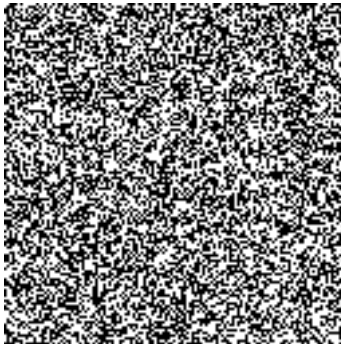
SEND
CASH

⊕



=

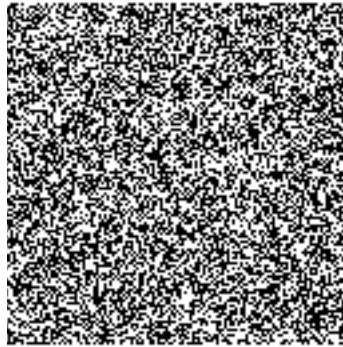
C1



M2

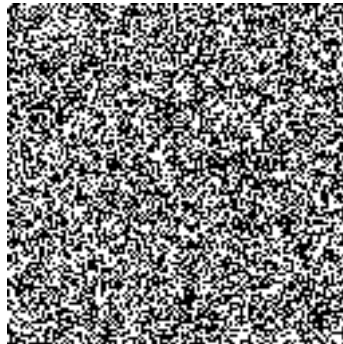


⊕



=

C2



SAME
PAD

⊕

=



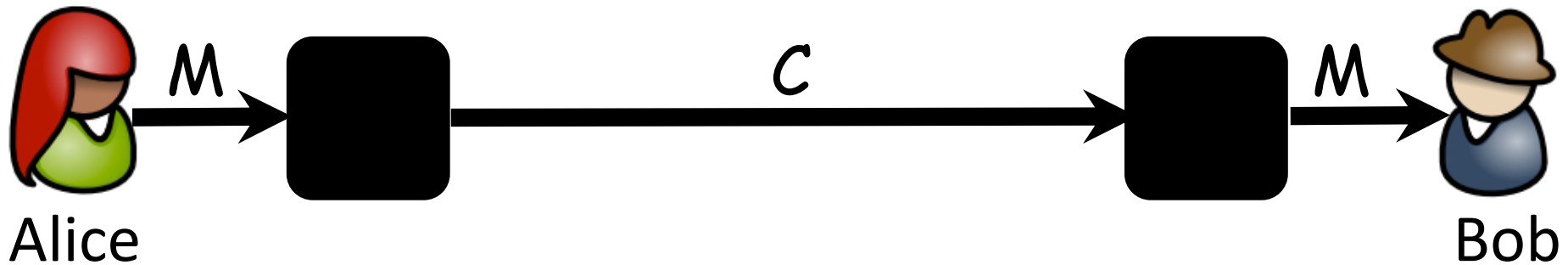
Modern Cryptography



Two flavors of confidentiality

- **Unconditional** or **probabilistic security**: cryptosystem offers provable guarantees, irrespective of computational abilities of an attacker
 - given ciphertext, the probabilities that bit i of the plaintext is 0 is p and the probability that it is 1 is $(1-p)$
 - e.g., one-time pad
 - often requires key sizes that are equal to size of plaintext
- **Conditional** or **computational security**: cryptosystem is secure assuming a computationally bounded adversary, or under certain hardness assumptions (e.g., $P \neq NP$)
 - e.g., DES, 3DES, AES, RSA, DSA, ECC, DH, MD5, SHA
 - Key sizes are much smaller (~ 128 bits)
- Almost all deployed modern cryptosystems are conditionally secure

Recall: Encryption and Decryption



$$C = E(M)$$

$$M = D(C)$$

i.e.,

$$M = D(E(M))$$

where

M = plaintext

C = ciphertext

$E(x)$ = encryption function

$D(y)$ = decryption function

Kerckhoffs' Principles

- Modern cryptosystems use a key to control encryption and decryption
- Ciphertext should be undecipherable without the correct key
- Encryption key may be different from decryption key.
- **Kerckhoffs' principles** [1883]:
 - Assume Eve knows cipher algorithm
 - Security should rely on choice of key
 - If Eve discovers the key, a new key can be chosen



Kerckhoffs' Principles

- Kerckhoffs' Principles are contrary to the principle of “**security by obscurity**”, which relies only upon the secrecy of the algorithm/cryptosystem
 - If security of a keyless algorithm compromised, cryptosystem becomes permanently useless (and unfixable)
 - Algorithms relatively easy to reverse engineer

Key Sizes

- Original DES used 56-bit keys, 3DES uses 168-bit keys
- AES uses 128-, 192- or 256-bit keys
- Are these numbers big enough?
 - DES has $2^{56} = 72,057,594,037,927,936$ possible keys
 - In Feb 1998, distributed.net cracked DES in 41 days
 - In July 1998, the Electronic Frontier Foundation (EFF) and distributed.net cracked DES in 56 hours using a \$250K machine
 - In Jan 1999, the team did in less than 24 hours
 - **Each additional bit adds 2X brute-force work factor (exponential security for linear keysize increase)**
 - There are approximately 2^{250} atoms in the universe, so don't expect 256-bit keys to be brute forced anytime in the foreseeable future (*with conventional computing*).
- Takeaway: 128-keys are reasonably secure

$$2^{256} =$$

115,792,089,237,316,195,
423,570,985,008,687,907,
853,269,984,665,640,564,
039,457,584,007,913,129,
639,936

Cryptanalysis

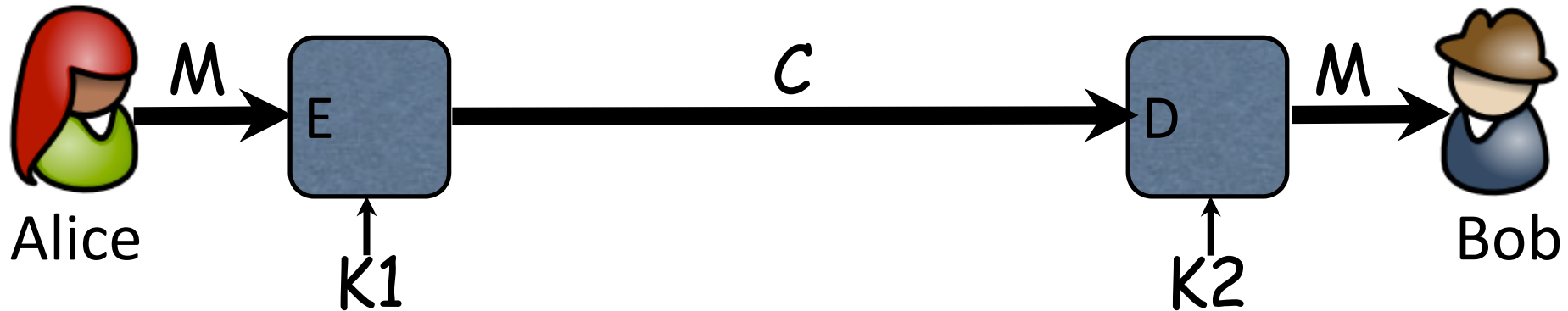
- Goal: learn the key
- Classifications:
 - **ciphertext-only** attack: Eve has access only to ciphertext
 - **known-plaintext** attack: Eve has access to plaintext and corresponding ciphertext
 - **chosen-plaintext** attack: Eve can choose plaintext and learn ciphertext
 - **chosen-ciphertext** attack: Eve can choose ciphertext and learn plaintext

Which of these are passive/active attacks?

Other cryptanalysis ...

- Brute force cryptanalysis
 - Just keep trying different keys and check result
- Not covered in this class:
 - Linear cryptanalysis
 - Construct linear equations relating plaintext, ciphertext and key bits that have a high bias
 - Use these linear equations in conjunction with known **plaintext-ciphertext pairs** to derive key bits
 - Differential cryptanalysis
 - Study how differences in an input can affect the resultant difference at the output
 - Use **chosen plaintext** to uncover key bits

Symmetric and Asymmetric Crypto



- **Symmetric crypto:** (also called **private key crypto**)

- Alice and Bob share the same key ($K=K1=K2$)
- K used for both encrypting and decrypting
- Doesn't imply that encrypting and decrypting are the same algorithm
- Also called **private key** or **secret key** cryptography, since knowledge of the key reveals the plaintext

- **Asymmetric crypto:** (also called **public key crypto**)

- Alice and Bob have different keys
- Alice encrypts with $K1$ and Bob decrypts with $K2$
- Also called **public key** cryptography, since Alice and Bob can publicly post their *public* keys

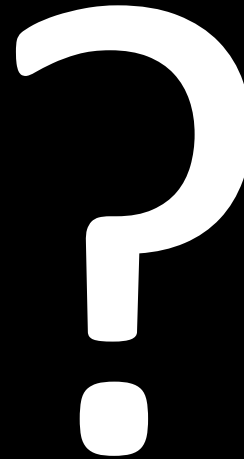
Confidentiality: Encryption and Decryption Functions

Private Key

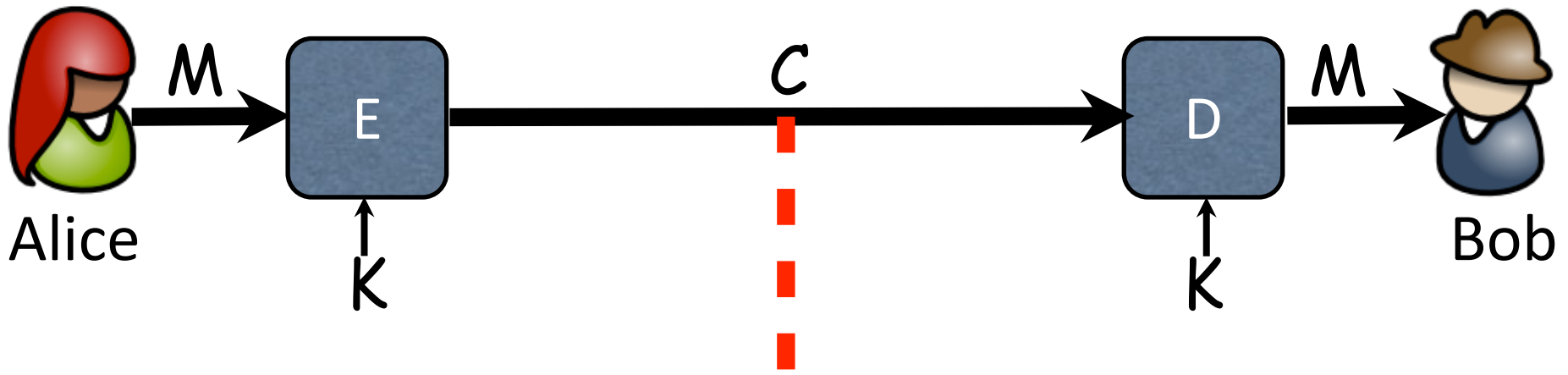
Stream
Ciphers

Block
Ciphers

Public Key



Secret Key Crypto



Without K ,
Eve cannot
decrypt C



Block ciphers vs. Stream ciphers

- **Stream Ciphers**

- Combine (e.g., XOR) plaintext with pseudorandom stream of bits
- Pseudorandom stream generated based on key
- XOR with same bit stream to recover plaintext
- E.g., RC4, FISH

- **Block Ciphers**

- Fixed block size
- Encrypt block-sized portions of plaintext
- Combine encrypted blocks (more on this later)
- E.g., DES, 3DES, AES

Stream Ciphers

- $E(M) = M \oplus C(K)$
 - $[C(K) = \text{pseudorandom stream produced using key } K]$
- Useful when plaintext arrives as a stream (e.g., 802.11's WEP)
- Vulnerable if used incorrectly

Stream Ciphers

- **Key reuse:** [C(K) = pseudorandom stream produced using key K]
 - $E(M1) = M1 \oplus C(K)$
 - $E(M2) = M2 \oplus C(K)$
 - Suppose Eve knows ciphertexts $E(M1)$ and $E(M2)$
 - $E(M1) \oplus E(M2) = M1 \oplus C(K) \oplus M2 \oplus C(K) = M1 \oplus M2$
 - $M1$ and $M2$ can be derived from $M1 \oplus M2$ using frequency analysis
- Countermeasure is to use IV (**initialization vector**)
 - IV sent in clear and is combined with K to produce pseudorandom sequence
 - E.g., replace $C(K)$ with $C(K \oplus IV)$
 - IVs should never be reused and should be sufficiently large
 - WEP broken partly because IVs were insufficiently large
 - modern stream ciphers take IVs, but it's up to the programmer to generate them

Stream Ciphers

- **Substitution Attack:**

- $M = \text{“Pay me \$100.00”}$

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

- Suppose Eve knows M and $E(M)$ but doesn't know K

- She can substitute M for M' by replacing $E(M)$ with:

- $E'(M) = E(M) \oplus M \oplus M' = M \oplus C(K) \oplus M \oplus M' = C(K) \oplus M'$

- Eve can then replace $E(M)$ with $E'(M)$, which Bob will decrypt message as M' (“Pay me \$900.00”)

- *Encryption alone does not provide integrity:* Countermeasure is to include message authentication code (more on this later) that helps detect manipulation (i.e., provides integrity and authenticity)