Cryptography Basics

CS463/ECE424

University of Illinois

Goals of This Lecture

- Know the **interfaces** of basic crypto primitives
  – What guarantees they provide and not provide
  – Their inputs and outputs
  – What it means for them to be secure
  – Where and how they are used

- Primitives we will cover - hashing, symmetric & asymmetric encryption, and digital signatures

Example Scenario

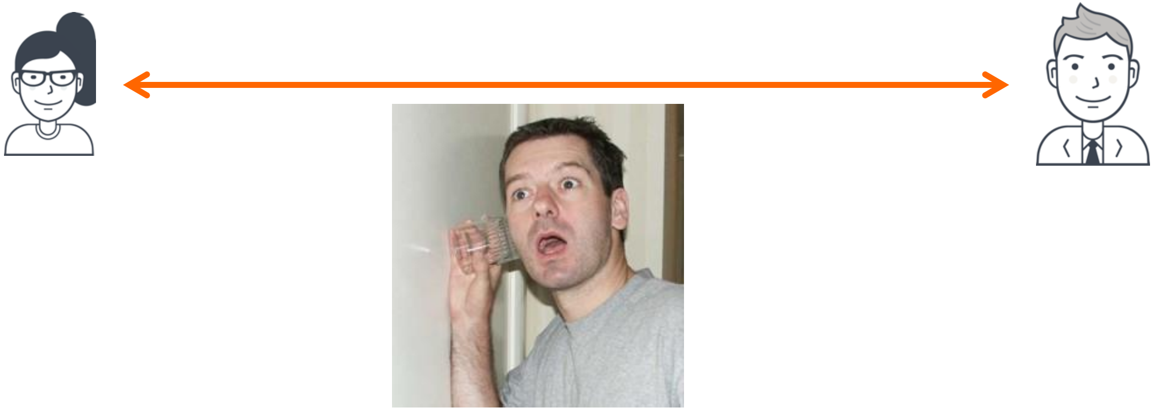
- To build a "secure" communication system, we need to ensure:
  - **C**onfidentiality
  - **I**ntegrity
  - **A**uthenticity
  - …

- How to use basic crypto primitives ensure these properties?

Symmetric Encryption
Hash Function
Asymmetric Encryption (public-key)
Digital Signatures

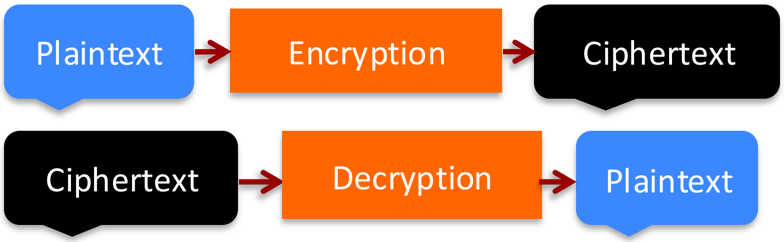
We start with Confidentiality

- Alice wants to talk to Bob in a **confidential** way
  - Bob can read/understand Alice’s message
  - Anyone other than Bob cannot eavesdrop with the conversation content



Encryption for Confidentiality

- **Encryption:** encode data such that only authorized parties can read
  - **Plaintext:** the intended communication information (original message)
  - **Ciphertext:** encrypted message (usually not understandable)
  - **Cipher:** encryption/decryption algorithm
  - **Key:** a parameter of the (en-)decryption algorithm that determines output
- Confidential communication
  - Alice encrypts her message M using K
    - C = Enc (M, K)
  - Alice shares K' w/ Bob (sometimes K=K')



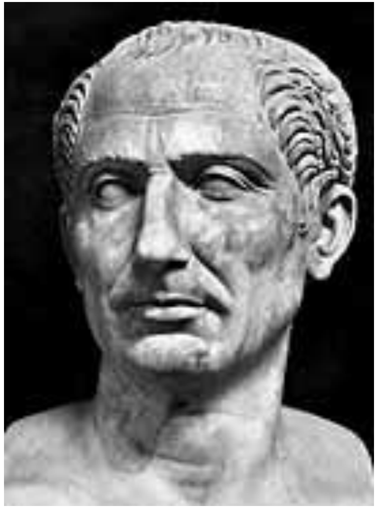
  - Only Bob (owner of K') can decrypt the message C
    - M = Dec (C, K')

Symmetric Encryption vs. Asymmetric Encryption

- Symmetric-key scheme (e.g., AES)
  - The keys for encryption and decryption **are the same**
  - Communicating parties **must have the same key before communication**

- Asymmetric/Public key scheme (e.g., RSA)
  - Public key is published **for anyone to encrypt a message**
  - **Only authorized parties have the private key** to decrypt the message

Substitution Ciphers



- **<u>Caesar cipher</u>** shifts letters with a constant of K
  - Encryption: $\mathbf{c_i} := (\mathbf{m_i} + \mathbf{k}) \bmod 26$
  - Decryption: $\mathbf{m_i} := (\mathbf{c_i} - \mathbf{k}) \bmod 26$
- K=3: "TREATY IMPOSSIBLE" → "wuhdwb lpsrvvleoh"
- Pros: Easy to remember and use
- Cons:
  - Obvious patterns in ciphertext
  - Ciphertext is deterministic: same plaintext always gives the same ciphertext

vjku oguucig ku pqv vqq jctf vq dtgcm

too to

this message is not too hard to break

Julius Caesar used caesar cipher to communicate with his generals 2100 years ago

Make substitution cipher more secure?

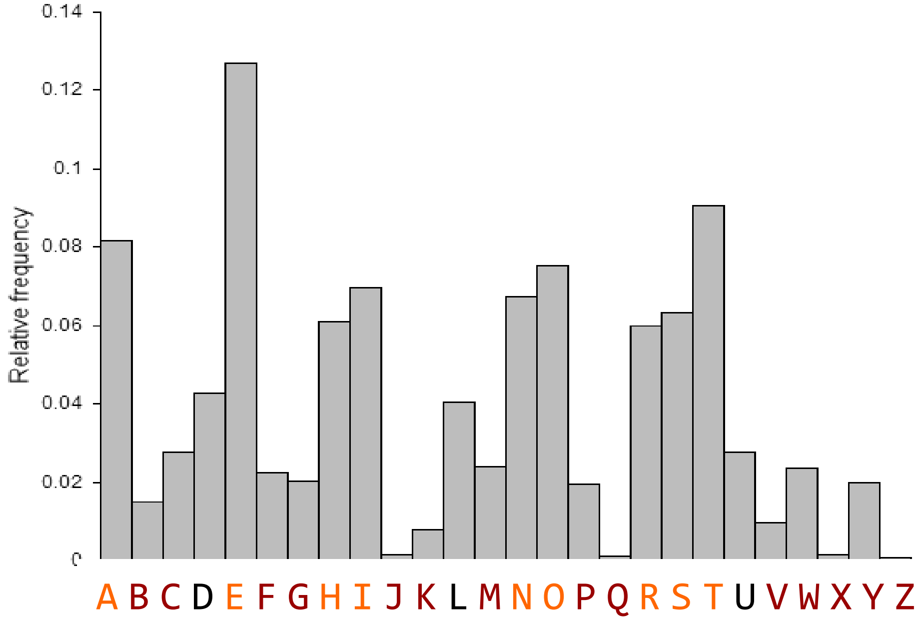
- Caesar cipher shifts letters with a constant of K
  - Brute-force attack: try all possible K → 26 tests
- How about substituting letters more randomly?
  - The key is a mapping function (A → C, B → Z, W→ B …)
  - Two different letters cannot map to the the same letter (why)?

Cannot decrypt!

- Now, how many combinations needed to the break cipher?
  - Brute-force to guess the key: 26*25*24…*1 = 26!
  - A more efficient way: frequency distribution analysis
    - Hints: E,T,O,A are more frequent in English words than J,Q,Z,X

Caesar Cipher Cryptanalysis

- Simple substitution ciphers don't alter symbol frequency



XOR Cipher

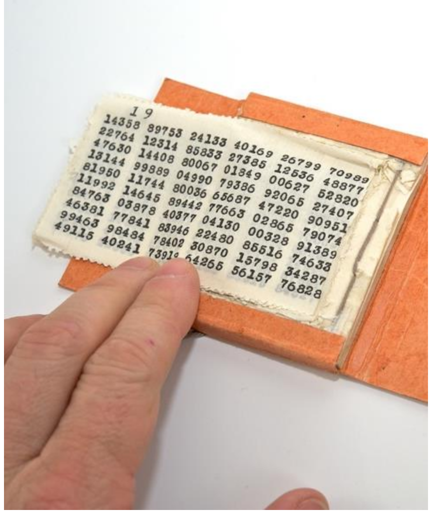
- XOR cipher $\oplus$
  - XOR the ith bit of your message with the ith bit of the key
  - Key is a random bit string (e.g., 011101000101)
- Sender: $M \oplus K$ → C
- Receiver: $C \oplus K$ → M
- Why this works: XOR operation has some nice properties
  - $M = K \oplus M \oplus K.$

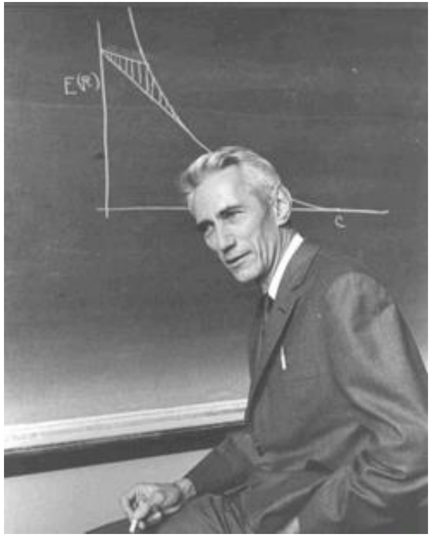
a	**b**	**a** xor **b**
0	0	0
0	1	1
1	0	1
1	1	0

a xor **b** xor **b** = **a**
a xor **b** xor **a** = **b**

One-time Pad (OTP)

- Alice and Bob jointly generate a secret: long bit stream of length K
  - To **encrypt**: $c_i = m_i \text{ xor } k_i$
  - To **decrypt**: $m_i = c_i \text{ xor } k_i$
- One-time: never reuse any part of the pad
- One-time pad has "perfect secrecy", but is not practical
  - Require "truly random" keys
  - Very long keys: the same length of the message
  - Need a new key each time



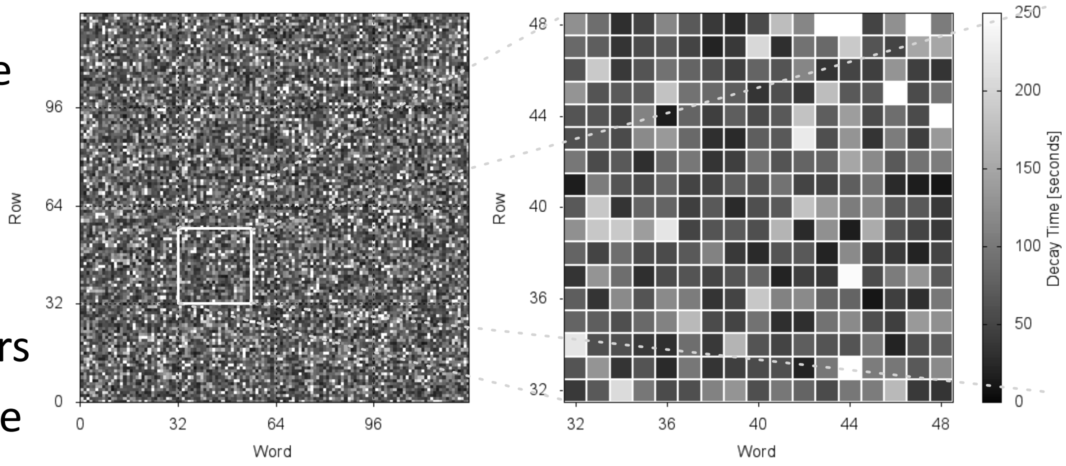


Random vs. Pseudorandom

- Pseudorandom sample is generated by an algorithm that generates a series of numbers that has “no internal pattern”
  - However, the series **requires a starting seed**; *if the algorithm is started repeatedly with the same seed, it will go through precisely the same sequence of numbers*
- A random sample, which is a common concept in statistics, is a sample drawn from a population **such that there is no bias in the process that selects the sample, and all members of the population have an equal chance of being selected**

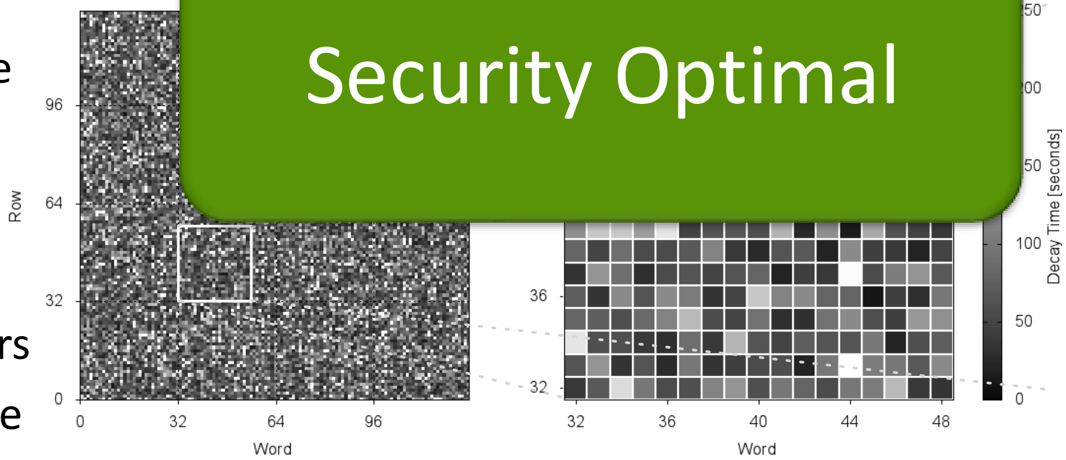
Sources of Randomness

- Coin flips
- Atomic decay
- Thermal noise
- Electromagnetic noise
- Physical variation
  - Clock drift
  - DRAM decay
  - Image sensor errors
  - SRAM startup-state
- Lava Lamps



Sources of Randomness

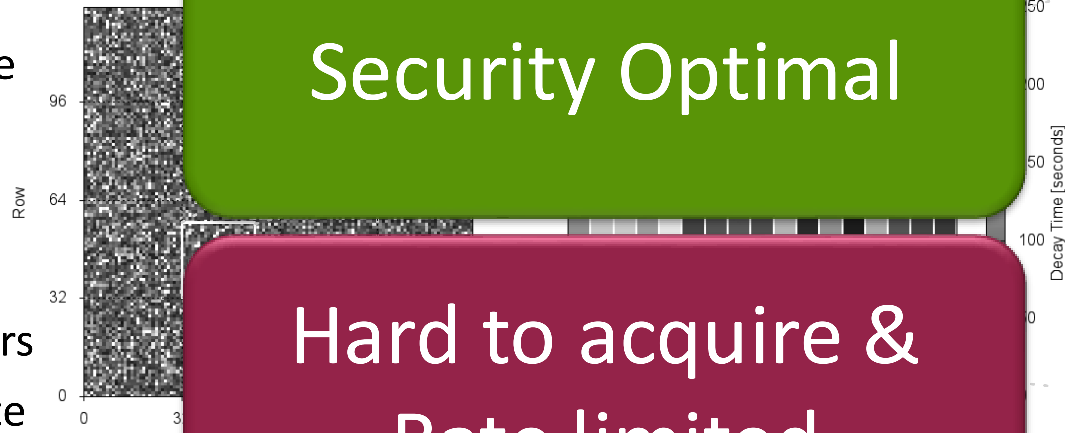
- Coin flips
- Atomic decay
- Thermal noise
- Electromagnetic noise
- Physical variation
  - Clock drift
  - DRAM decay
  - Image sensor errors
  - SRAM startup-state
- Lava Lamps



Security Optimal

Sources of Randomness

- Coin flips
- Atomic decay
- Thermal noise
- Electromagnetic noise
- Physical variation
  - Clock drift
  - DRAM decay
  - Image sensor errors
  - SRAM startup-state
- Lava Lamps



Security Optimal

Hard to acquire & Rate limited

Symmetric Key Encryption Methods

- **Stream cipher:** operates on individual bits (or bytes); one at a time
- **Block cipher:** operates on fixed-length groups of bits called *blocks*
- Only a few symmetric methods are used today

Methods	Year approved	Comments
Data Encryption Standard - DES	1977	1998: Electronic Frontier Foundation's Deep Crack breaks a DES key in 56 hrs
DES-Cipher Block Chaining		
Triple DES – TDES or 3DES	1999	
Advanced Encryption Standard – AES	2001	among the most used today
Other symmetric encryption methods		
IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish		

~~Symmetric Encryption~~

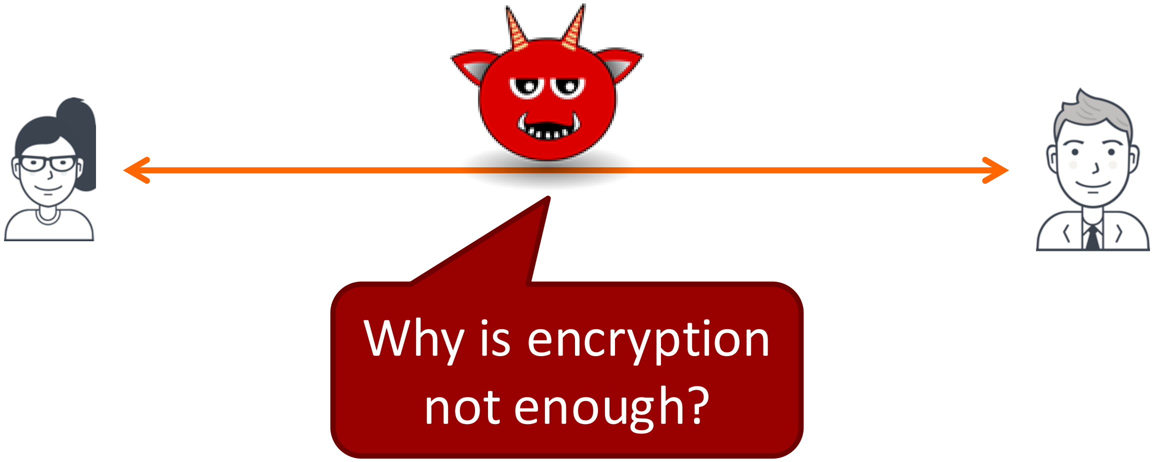
Hash Function

Asymmetric Encryption (public-key)

Digital Signatures

Let's Examine Integrity

- Alice wants to talk to Bob without disruption
  - Bob can read/understand Alice's message
  - Attackers cannot tamper with the message without being noticed



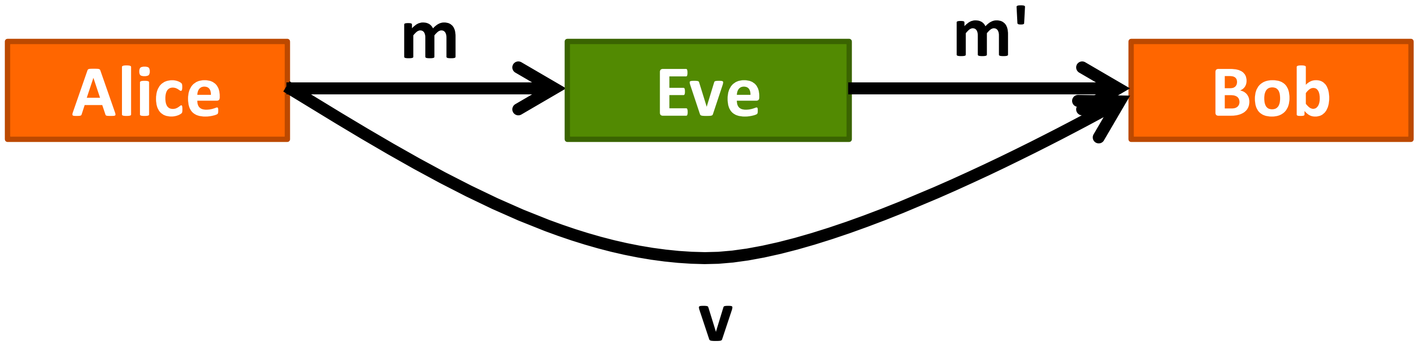
Message Integrity

- Eve (attacker) changes Alice's message M to M', Example:
  - Alice sends a passcode "1234"
  - Bob receives a passcode "5678" (integrity violation)
    - Such an encryption scheme is deemed "malleable"

Message Integrity

Ignore confidentiality for now

- **Approach:** Compute message-dependent data along with the original message
  - let **v = h(m)**
  - Bob computes **v’ = h(m’);** checks whether **v’== v**



Cryptographic Hash Functions

- Input – data of an **arbitrary** length
- Output – **fixed length**, e.g., 256 bits

- **Deterministic**: same input always produces the same output
- **Length compressing** (output length <= input length)
- **Hard to invert** (one-way, OW)
- **Hard to find collisions** (collision-resistant, CR)
- Examples: MD5, SHA1, SHA2, SHA3
  - SHA3-256("welcome") =
    - 64db51f8f79ca7ec522a6b4a e5fc7e896daac5318b2e82730d7c7926b66d36eb
  - SHA3-256(”Welcome") =
    - 18ec669de973b4483db9b64 b2746ceda564cd2cdec2277169382944675a2ff9e

Definition

- A cryptographic hash function H with n-bit output is a function:

$$y = H(x): \{0,1\}^* \rightarrow \{0,1\}^n$$

Definition

- A cryptographic hash function H with n-bit output is a function:

$$y = H(x): \{0,1\}^* \rightarrow \{0,1\}^n$$

- One-way (OW, also called preimage resistance): given any y, **infeasible** to find x s.t. H(x) = y

- Collision-resistance (CR): **infeasible** to find x and x' s.t. $x \neq x'$ but H(x) = H(x')

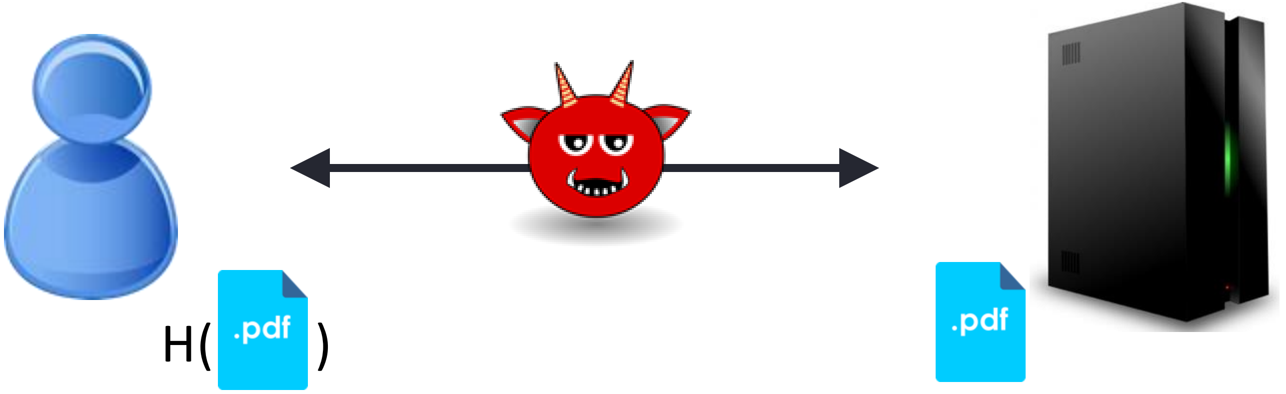
Hash Function Applications

- Downloading software online
- Email signing
- De-duplication
- Verifying the integrity of remote storage

Hash Function Applications

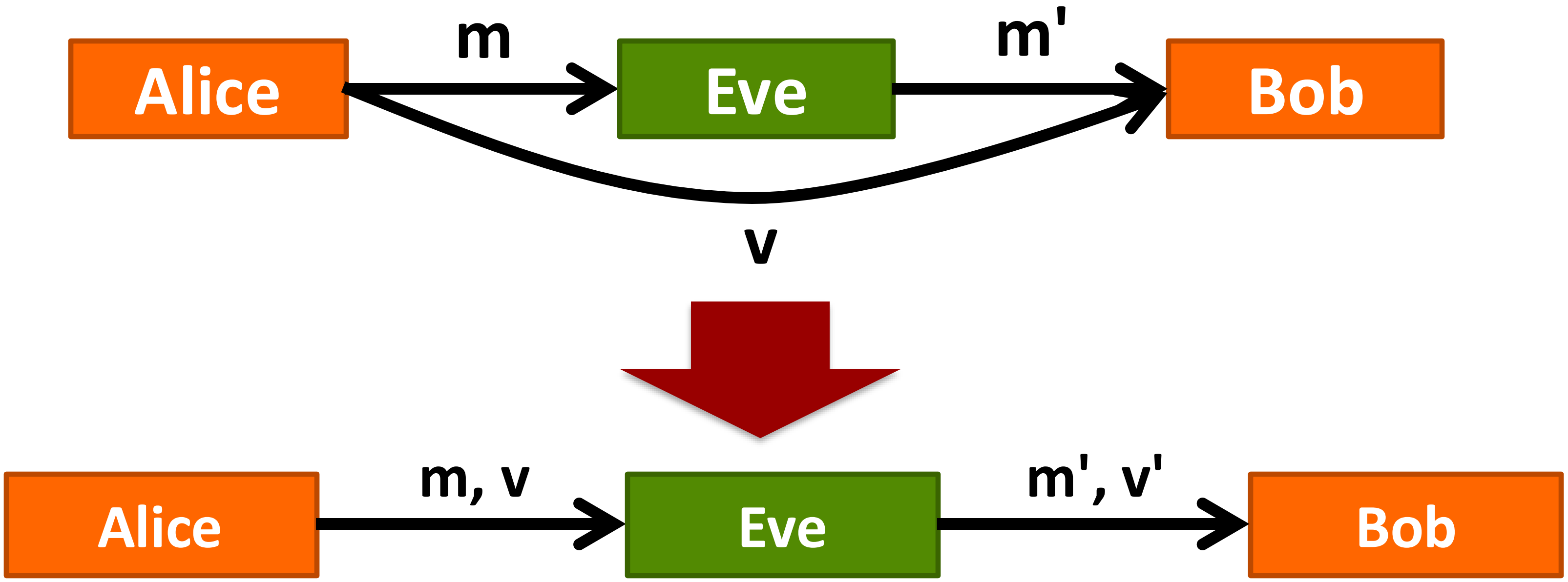
- Integrity of remote/external storage
  - User computes and stores H(file) locally
  - Compare hash upon download

Dropbox /
Google Drive



What if “v” also needs to be sent over network?

- **Approach:** Send message-dependent data along with the original message



What if “v” also needs to be sent over network?

- **Approach:** Send message-dependent data along with the original message
- **Function h(m)=v is:**
  - Deterministic: same input always produce the same output
  - One-way and collision-resistant
- If Eve knowns h(), Eve can compute a new v’ where v’ = h(m’)

Better Solution: “Keyed Hash”

- **Approach:**
  - Let $\mathbf{h_k}$ be a **keyed hash function**
  - In advance, choose a random **k** known only to Alice and Bob
  - let $\mathbf{v = h_k(m)}$
  - Bob checks that $\mathbf{h_k}$(**m'**) **== v'**, otherwise **m'** untrusted

~~Symmetric Encryption~~

~~Hash Function~~

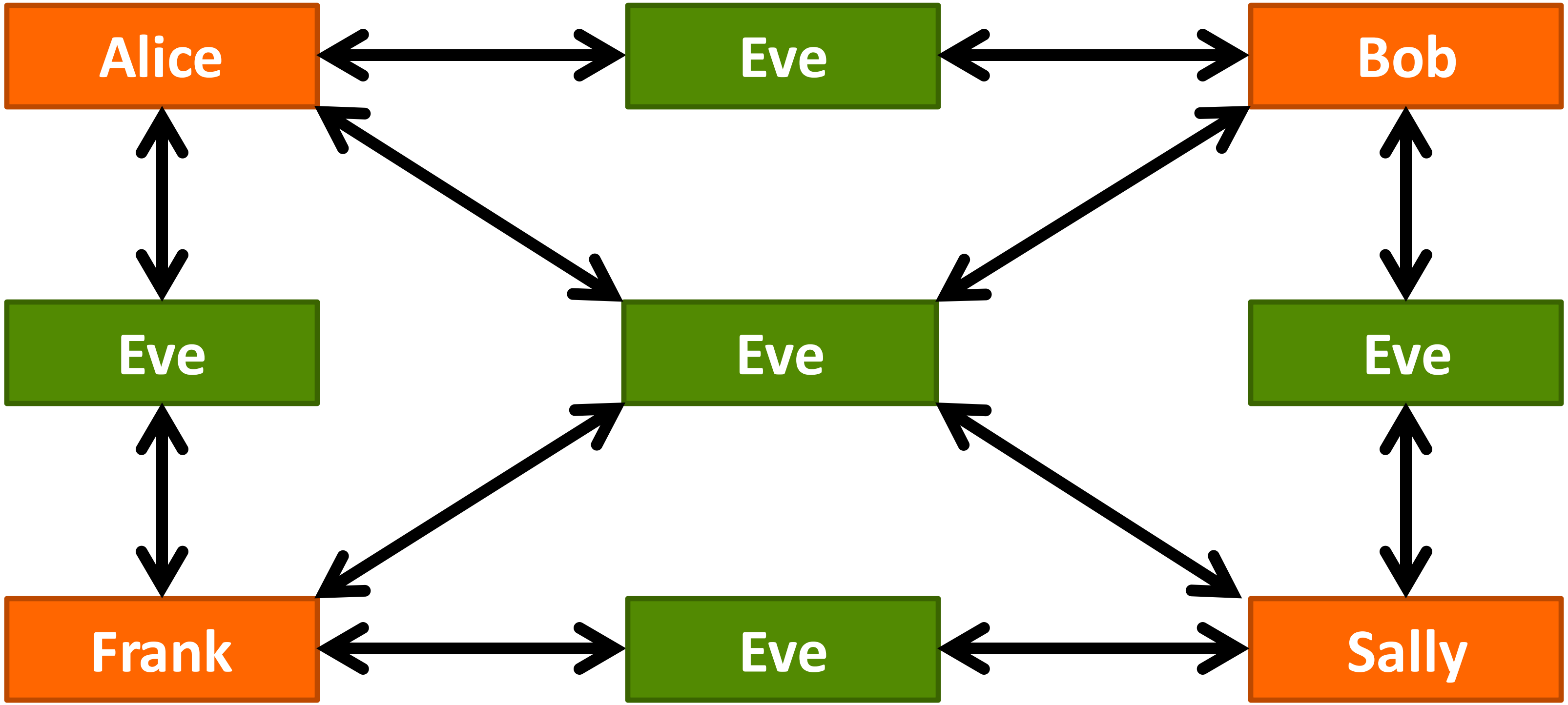
Asymmetric Encryption (public-key)

Digital Signatures

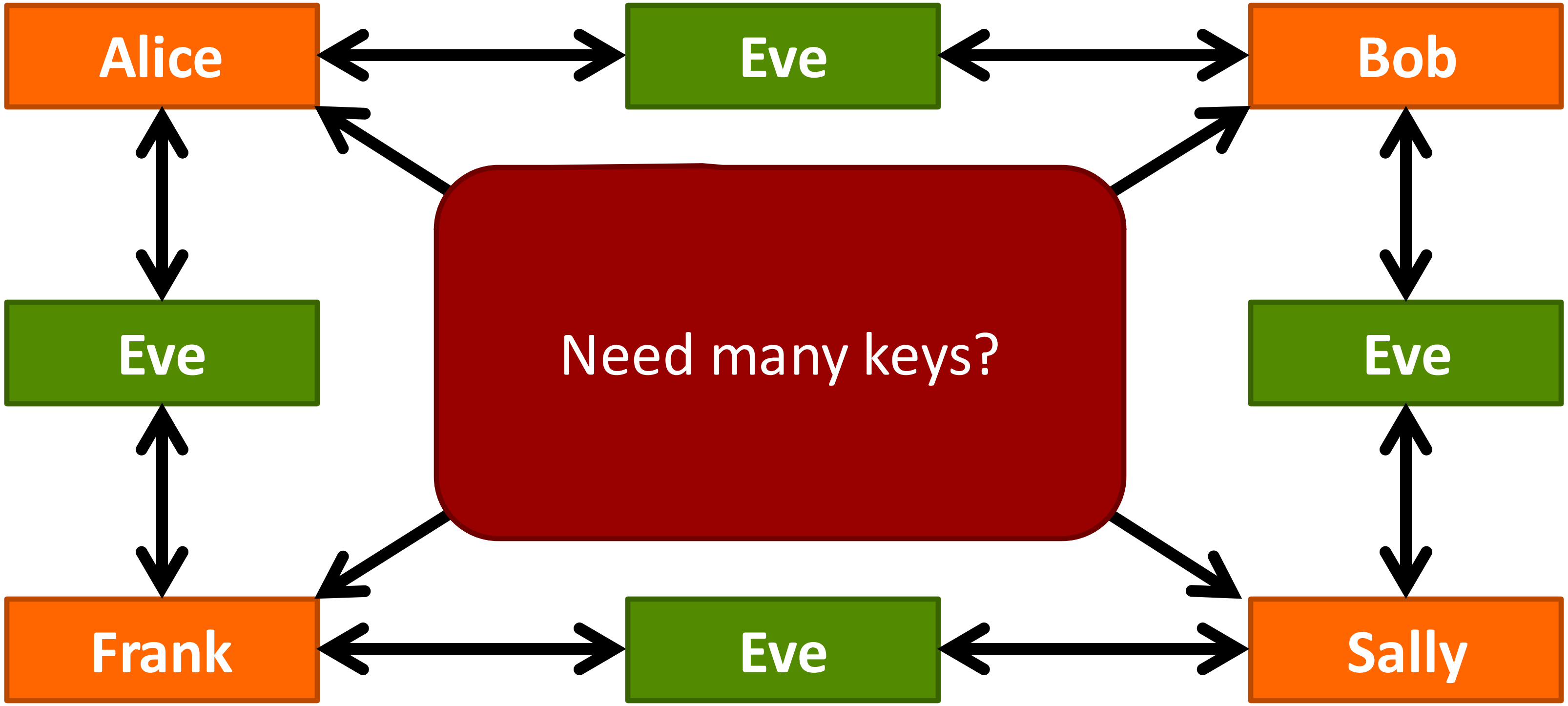
Symmetric Encryption Needs to Pre-Share a Key

- Alice wants to talk to Bob

Multi-party Confidential Communication

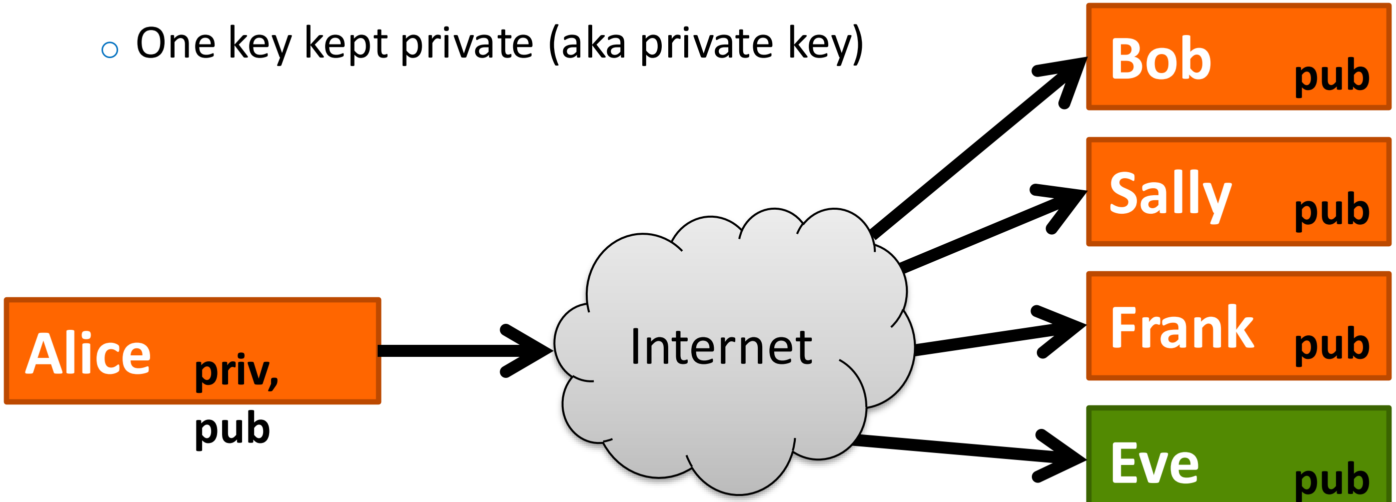


Multi-party Confidential Communication



Public-key Crypto (asymmetric key)

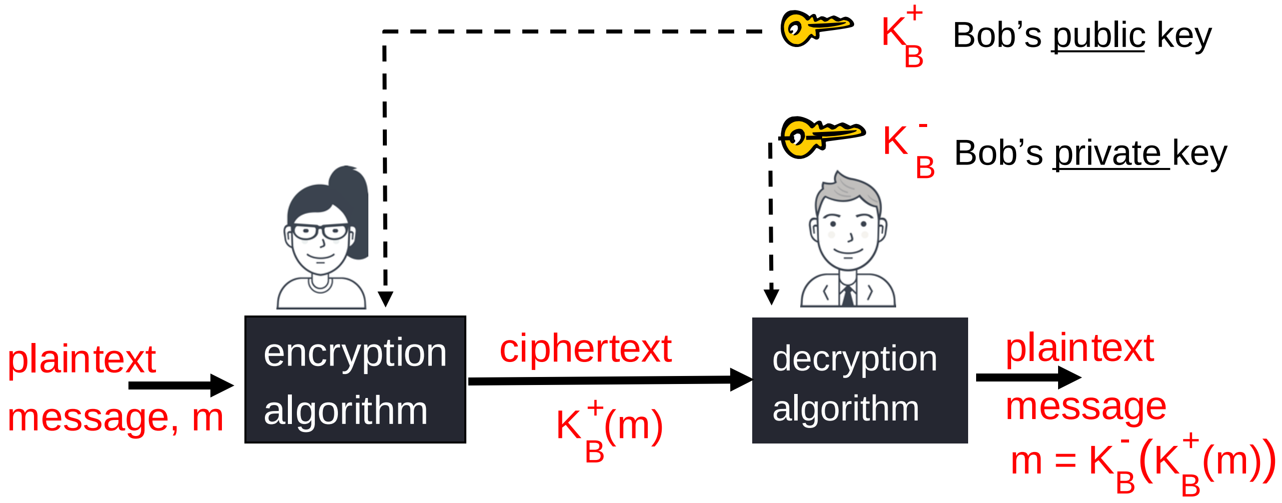
- Users can have two keys
  - Keys generated in pairs using well-understood mathematical relationship
    - One key shared publicly (aka public key)
    - One key kept private (aka private key)



Public-key Requirements

- Computationally easy for user to generate public-private key pair
- Computationally easy for Bob to generate ciphertext given arbitrary plaintext: $\mathbf{C = E_{A\_pub}(M)}$
- Computationally easy for Alice to generate plaintext given arbitrary ciphertext: $\mathbf{M = D_{A\_priv}(C)}$
- Computationally infeasible to determine **A_priv** from **A_pub**
- Computationally infeasible to determine **M** given **A_pub** and **C**

Public key Cryptography



Alice encrypts her message with Bob's public key;

Bob decrypts with his private key

RSA

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

RSA scheme: Choosing keys

1. Choose two large prime numbers p, q.
   (e.g., 2048 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$ ← totient function
3. Choose e (with $e<n$) that has no common factors with z.
   (e, z are "relatively prime").
4. Choose d such that $ed-1$ is exactly divisible by z.
   (in other words: $ed \bmod z = 1$).
5. *Public* key is (n,e). *Private* key is (n,d).

   $\underbrace{(n,e)}_{K_B^+}$ $\underbrace{(n,d)}_{K_B^-}$

RSA: Encryption, decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt bit pattern, m, compute

   $c = m^e \bmod n$ (i.e., remainder when m^e is divided by n)

2. To decrypt received bit pattern, c, compute

   $m = c^d \bmod n$ (i.e., remainder when c^d is divided by n)

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

RSA: Why It works?

$m = (m^e \bmod n)^d \bmod n$

Useful number theory: If p, q prime and $n = pq$, then:

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose *ed* to be divisible by *(p-1)(q-1)* with remainder 1)

$$= m$$

RSA vs. AES

- AES is **1000x faster** than RSA
- AES is **less complex** than RSA
- AES has **10x shorter keys** than RSA (e.g., 192 bits vs. 2048 bits)
- **RSA requires no shared secrets**

Attacks on RSA encryption scheme

- Basic RSA is **deterministic** encryption scheme
  - Same message always gives same ciphertext

- Basic RSA is **multiplicative**:
  - $(m_1^e \bmod n)\,(m_2^e \bmod n) = (m_1 m_2)^e \bmod n$
  - This property may enable attacker to decrypt ciphertext of his choice

Solution: a special *structured and randomized* padding

Padding introduces randomized information to ciphertext

~~Symmetric Encryption~~

~~Hash Function~~

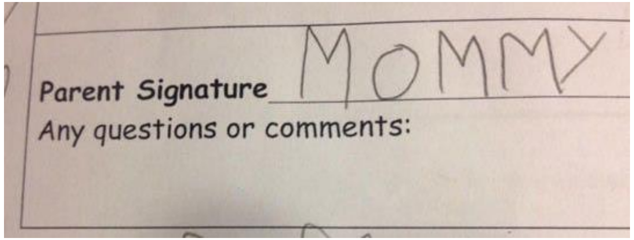
~~Asymmetric Encryption (public-key)~~

Digital Signatures

Authenticity: How Can I Know it is Indeed Alice?

- Alice wants to talk to Bob

Digital Signature schemes



- For authenticating origin of a message
- For message integrity
- Sender cannot deny having sent message (i.e., nonrepudiation)
  - Limited to *technical* proofs
    - Inability to deny one's cryptographic key was used to sign
  - However, one could claim the cryptographic key was stolen
    - Legal proofs, *etc.,* probably required; not dealt with here

RSA digital Signature Scheme

Use encryption in reverse!

- Setup:
  - Alice generates (d, e, n) as in RSA encryption scheme
  - Alice (signer) keeps private key (d, n) and publishes public key (e, n)
  - (d, n) private key for signing
  - (e, n) public key for verifying
- Sign(m, d, n): Alice computes $s = m^d \bmod n$
  - s is the signature of message m by Alice
- Verify(s, m, e, n): Bob (verifier) computes $m' = s^e \bmod n$
  - Bob makes sure m and m' are the same

One catch: this naïve use is not secure

Attack on the naïve RSA Signature Scheme

- Example: *Bob's keys:* $n_B = 77$, $e_B = 53$, $d_B = 17$
- 26 contracts, numbered 00 to 25
- Alice (attacker) has made Bob sign 05 and 17
  - $s1 = m_1^{d_B} \bmod n_B = 05^{17} \bmod 77 = 3$
  - $s2 = m_2^{d_B} \bmod n_B = 17^{17} \bmod 77 = 19$
- Alice computes a new contract number:
  - $m1 \times m2 \bmod 77 = 05 \times 17 \bmod 77 = 08$;
  - corresponding new signature: $s1 \times s2 \bmod 77 = 03 \times 19 \bmod 77 = 57$;
- Alice claims Bob signed contract 08!
  - Judge computes $(s1 \times s2)^{eB} \bmod n_B = 57^{53} \bmod 77 = 08$
  - Signature validated; Alice successfully forges Bob's signature

Solution: need to sign on the hash of the message H(m)

Summary

- Crypto primitives: hashing, symmetric & asymmetric encryption, and digital signatures
  - What guarantees they provide and not provide
  - Their inputs and outputs
  - What it means for them to be secure
  - Where and how they are used

- Security properties: confidentiality, integrity, authenticity
- Costs: computation overhead, key management, etc.