Security: Cryptography II

Lecture 39

Symmetric Key

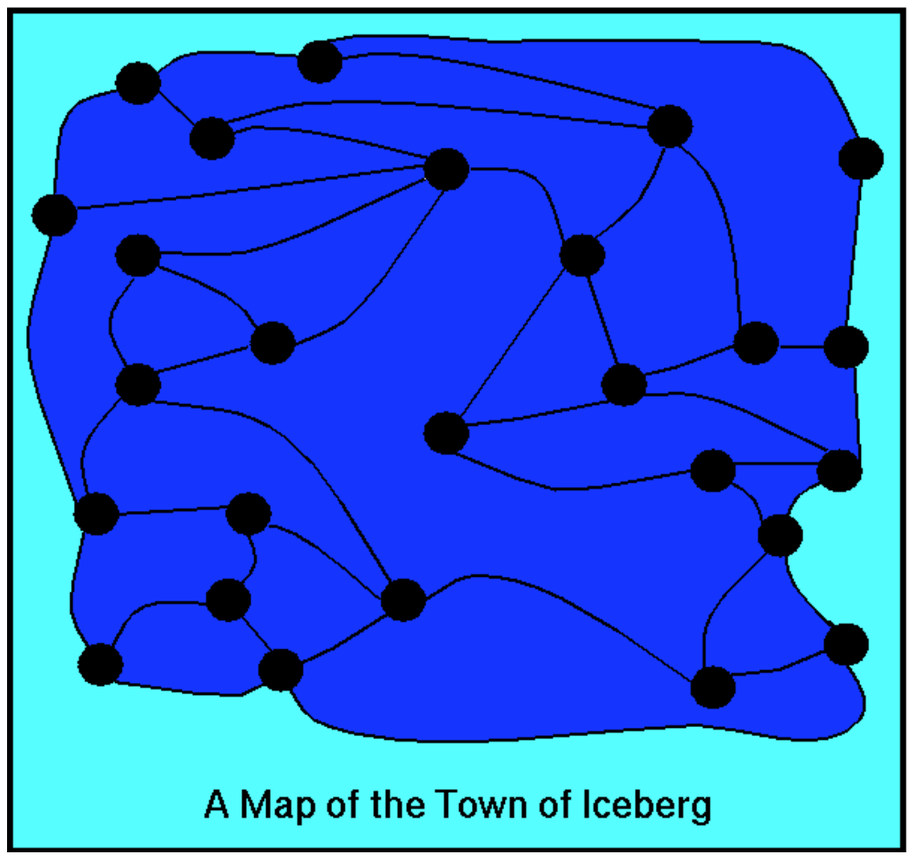
- For ciphers (so far): Knowing E is enough to figure out D (its inverse)
  - If you know how to encrypt, you can decrypt too
  - Known as a *symmetric key* cipher
- Example: Caesar cipher
  - If $E(m) = m + 3$, $D(m) = m - 3$
- Example: One-time pad
  - Use same pad and same operation (xor)
- Example: AES
  - Use same key, reverse rounds and steps

One-Way Functions

- For some functions, the inverse is hard to calculate
  - One direction (P→Q) is easy, but opposite direction (Q→P) is hard/expensive/slow
- Intuition:
  - Given a puzzle *solution*, easy to design a puzzle with that solution (the "forward" direction)
  - Given the puzzle, hard to come up with the solution (the "inverse" direction)

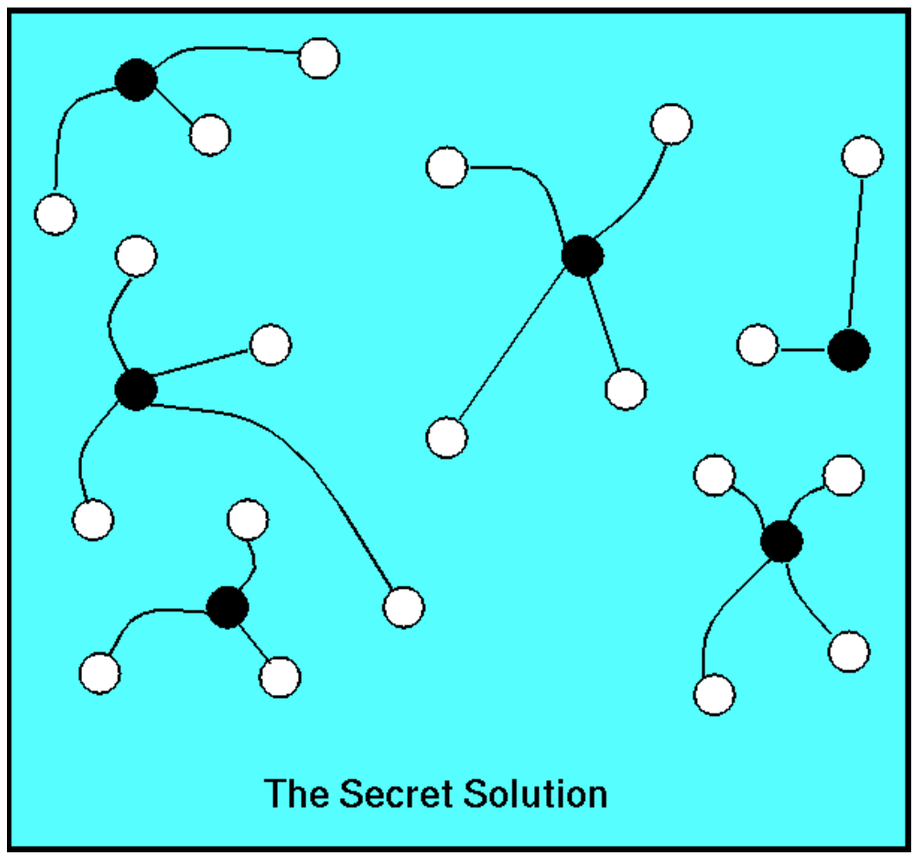
Example: Dominating Set

- Hard direction: Find a dominating set of size at most 6 in the following graph…



Example: Dominating Set (Solution)

- Easy direction: Create a graph with a dominating set of size 6 from this forest…



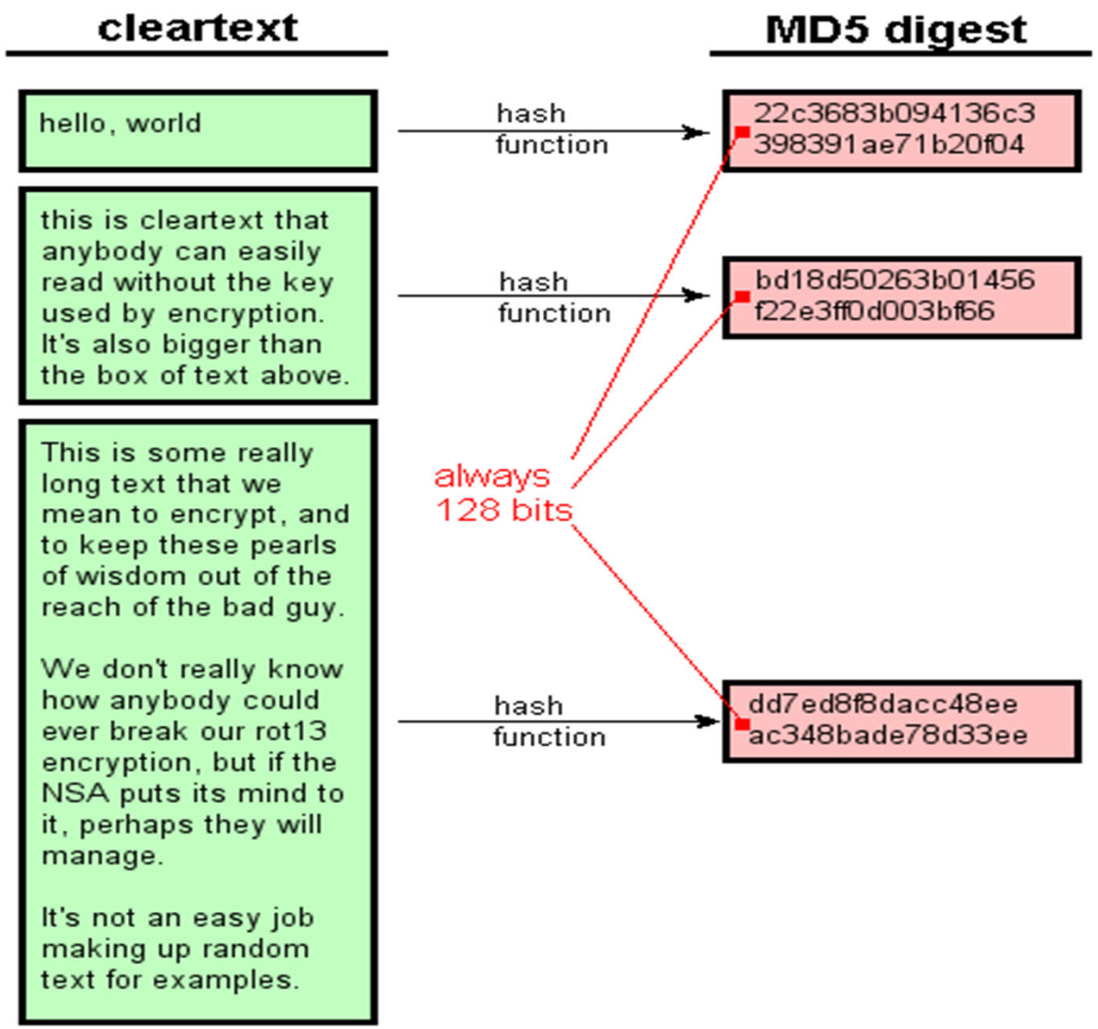
Example: Factoring

- Multiplying numbers is easy (*i.e.* fast)
  - Multiplying 2 n-bit numbers takes n^2 steps
- Factoring a number is hard (*i.e.* slow)
  - To factor an n-bit number, need 2^n steps (approximately the number's value)
- Aside:
  - Primality *testing* is fast (recall lab activity in Software I and Fermat's Little Theorem)
  - But this fast test doesn't reveal the *factors* of a composite number

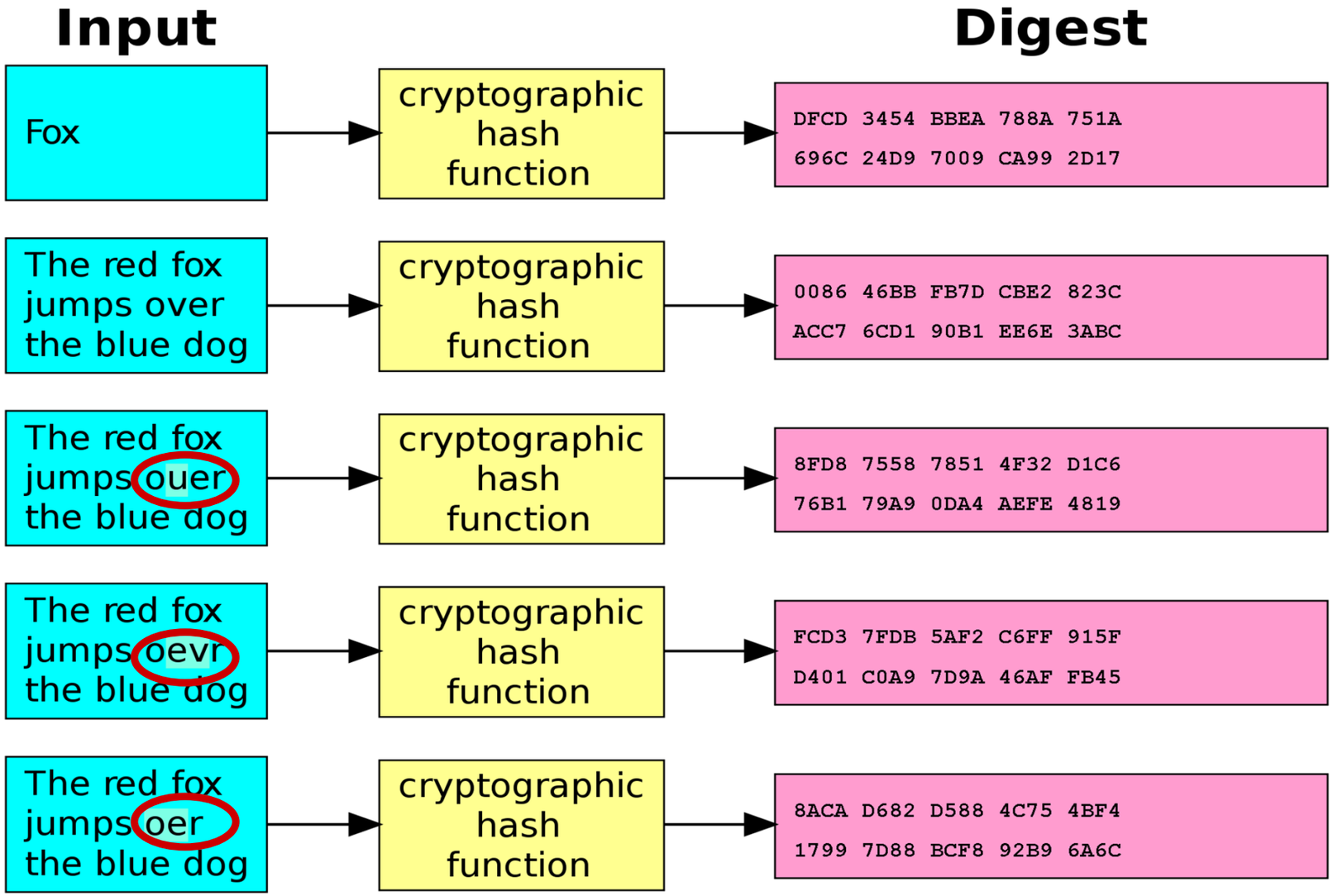
Cryptographic Hash Functions

- A hash function: $\mathbb{Z} \rightarrow \mathbb{Z}_B$
  - *Every* message, regardless of its length, maps to a number in the range $0..B-1$
  - Result called a *digest* (constant-length, $\lg B$)
  - Good hashes give uniform distribution:
    small diff in message → big diff in digest
- *Cryptographic* hash func's are one-way
  - Given a digest, computationally infeasible to find *any* m that hashes to it
  - Collisions must still exist ($B \ll |\text{messages}|$), but are infeasible to find for large enough B
  - Digest = a *fingerprint* of m (small, fixed-size)

Fixed-Length Digests



Crypto. Hash as Fingerprint



Common Cryptographic Hashes

- MD5
  - Flaws discovered: “cryptographically broken”
  - Do not use!
- SHA-1: deprecated
  - Windows, Chrome, Firefox reject (2017)
  - 160-bit digests (*i.e.* 40 hex digits)
- Replaced by SHA-2 (still common)
  - A family of 6 different hash functions
  - Digest sizes: 224, 256, 384, or 512 bits
  - Names: SHA-224, SHA-256, SHA-512, etc
- Current state-of-the-art is SHA-3
  - Entirely different algorithm
  - Names: SHA3-224, SHA3-256, SHA3-512, etc

Utility of Crypto. Hashes

- Integrity verification (super-checksum)
  - File download, check digest matches
- Password protection
  - Server stores the *hash* of user's password
  - Check entered password by computing its hash and comparing hash to the stored value
  - Benefit: Passwords are not stored (directly) in the database! If server is compromised, intruder finds hashes but not passwords
- Problem:
  - See md5decrypt.net/en/Sha256/

c023d5796452ad1d80263a05d11dc2a42b8c19c5d7c88c0e84ae3731b73a3d34

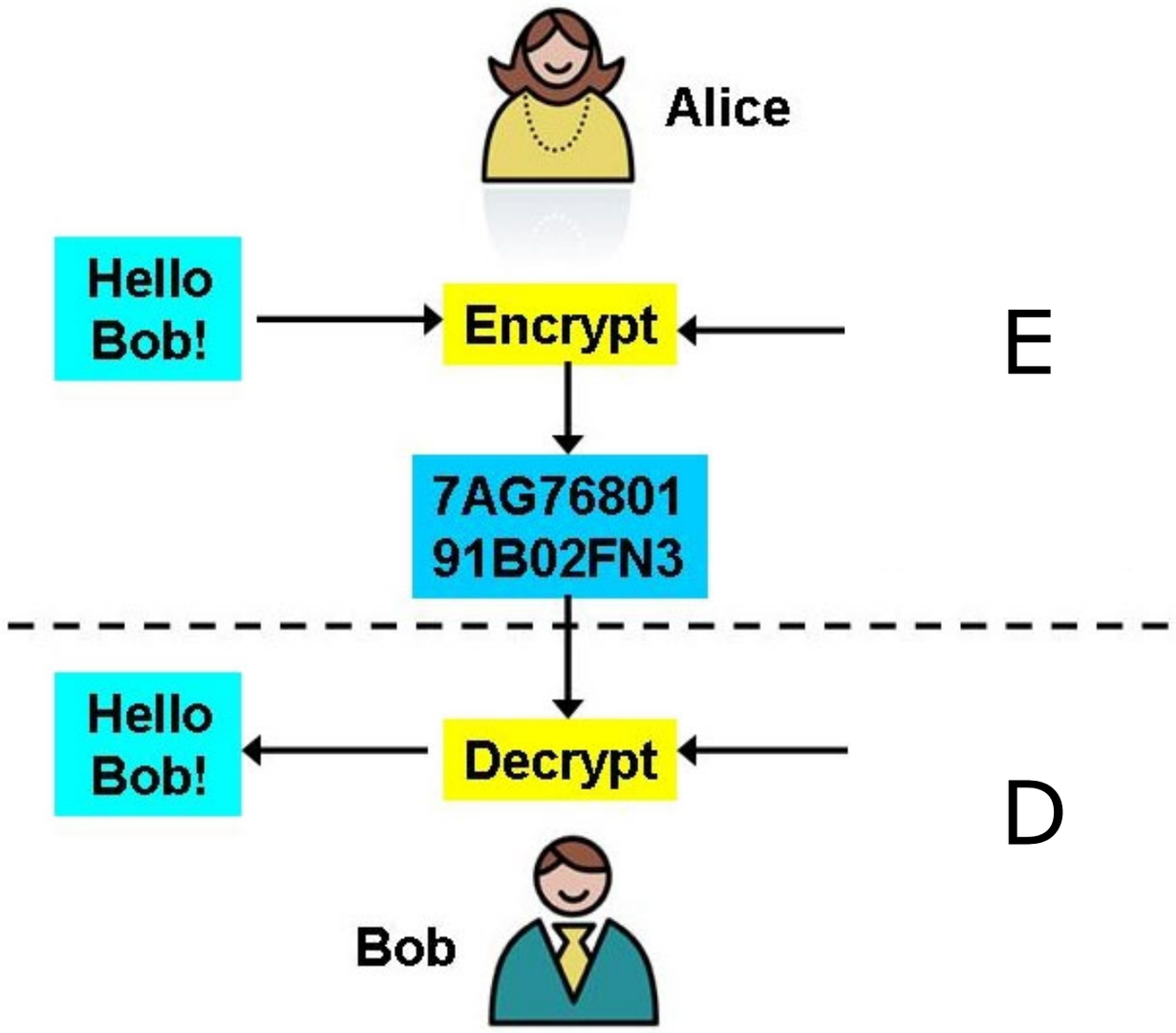
Role of Salt

- Danger:
  - Intruder pre-computes hashes for many (common) passwords: aka a *rainbow table*
  - Scan stolen hashes for matches
- Solution: *salt*
  - Server prepends text to password before hashing
  - Text must be *unique* to user
  - Text does not *need* to be secret
    - Ok: Deterministic value based on user name
    - Better: Random value, stored in the table
- Protects the fingerprint, by making it not mass pre-computable

One-Way Function with Trapdoor

- Function *appears* to be one-way
  - In reality, however, the inverse is easy if one knows a secret (the "trapdoor")
- There are two very different functions:
  - The one-way-seeming function, E
  - The trapdoor for its inverse, D
- Knowing E is *not enough* to infer D
- Creates an asymmetry:
  - Alice knows E
  - Bob (and only Bob) knows D

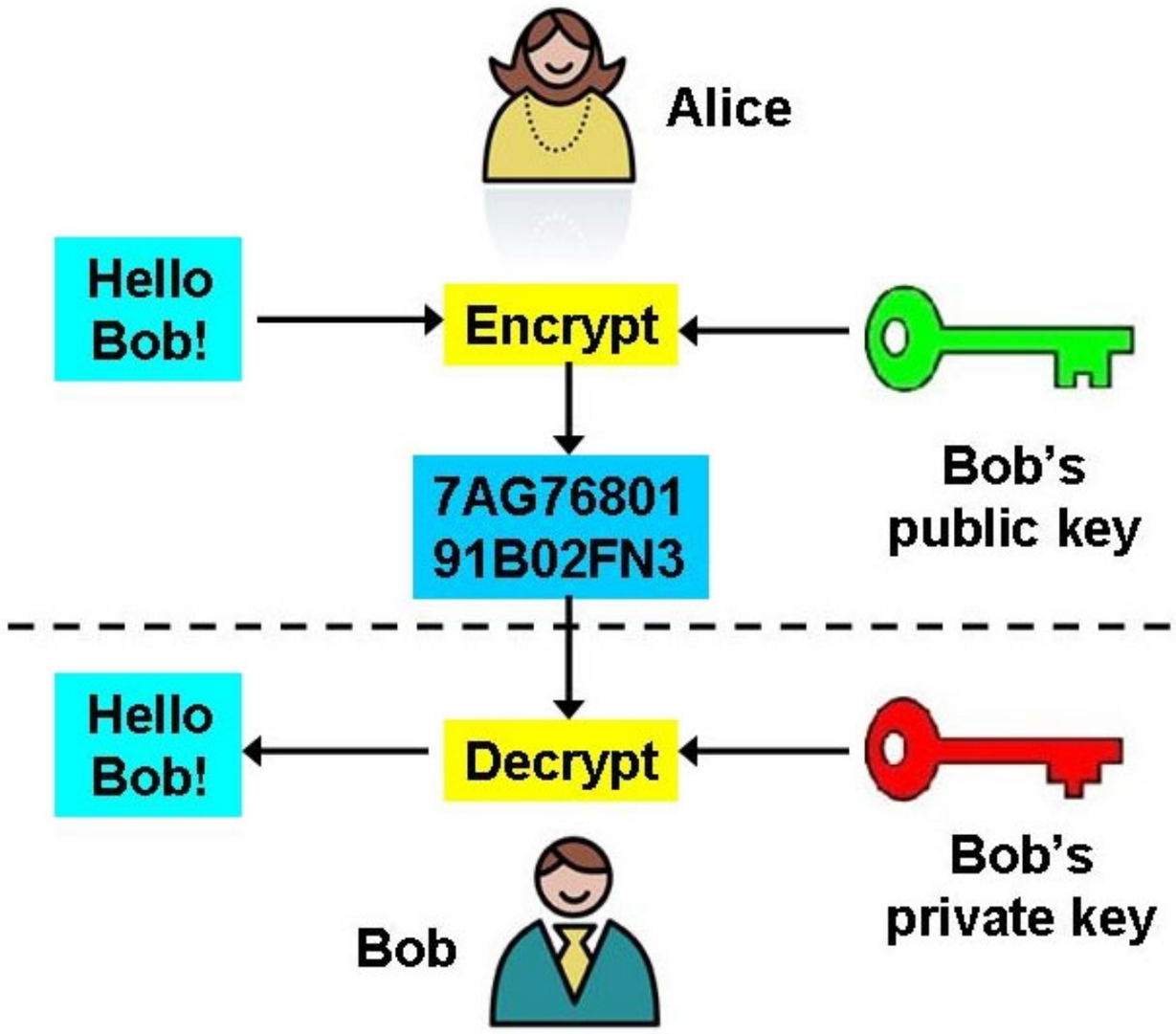
Asymmetry: Alice vs Bob



Public-Key Encryption

- Algorithms for E and D known by all
  - But parameterized by matched keys
- Asymmetry
  - Everyone knows key for Bob's E (*public)*
  - Only Bob knows key for Bob's D (*private)*
- Anyone can encrypt messages for Bob
- Only Bob can decrypt these messages
- Important consequences
  - Each agent needs only 1 public key
  - No pre-existing shared secret needed

Public and Private Keys



RSA

- E and D are actually the same function

  $m^k \bmod n$

  - Parameterized by pair (k, n), *i.e.* the key
- Private key: (d, n)
  - $D(m) = m^d \bmod n$
- Public key: (e, n)
  - $E(m) = m^e \bmod n$
- Choice of *e* & *d* is based on factoring
  - Choose 2 large **prime** numbers, *p* and *q*
  - Calculate their product, *n = pq*
  - Pick any *d* relatively prime with *(p-1)(q-1)*
  - Find an *e* s.t. *ed = 1 mod (p-1)(q-1)*

Digital Signature

- Usual direction for encryption:

  $D(E(m)) = (m^e)^d = m^{ed} = m, \mod n$

- One-to-one, so backwards works too!

  $E(D(m)) = (m^d)^e = m^{de} = m, \mod n$

- Consider:
  - Bob "encrypts" *m* using his **private** key, *d*
  - Bob sends **both** *m* and *D(m)*
  - Anyone can undo the "encrypted" part using Bob's **public** key, *e*
  - Result will be *m*
- D(m) serves as a digital **signature** of *m*
  - Only Bob could have created this signature
  - Use: non-repudiation

Performance Considerations

- Symmetric key algorithms are faster than public key algorithms
- Optimization for encryption (RSA)
  - Create a fresh symmetric key, *k*
  - Use symmetric algorithm to encrypt *m*
  - Use recipient's public key to encrypt *k*
- Optimization for digital signatures
  - Calculate the digest for *m* (always short)
  - Use sender's private key to encrypt digest

Take Home Message

- Don’t try to roll your own crypto/security implementation
- Use (trusted) libraries
- Recognize role and importance of (eg):
  - Initialization vector
  - Cryptographic hash/digest
  - Salt
  - Private key vs public key

Summary

- One-way function
  - Cryptographic hash creates a fingerprint
- Public key encryption
  - Matching keys: $k_{private}$, k_{public}
  - Anyone can use public key to encrypt
  - Only holder of private key can decrypt
  - Use private key to create a digital signature