Security: Cryptography

Lecture 37

Some High-Level Goals

- Confidentiality
  - Non-authorized users have limited access
- Integrity
  - Accuracy/correctness/validity of data
- Availability
  - No down-time or disruptions
- Authenticity
  - Agents are who they claim to be
- Non-repudiation
  - A party to a transaction can not later deny their participation

Methods of Attack

- Target people (“social engineering”)
  - Phishing: email, phone, surveys, …
  - Baiting: click & install, physical media, …
- Target software (“exploits”)
  - Unpatched OS, browser, programs
  - Buffer overflow
  - Code injection and cross-site scripting
- Target channel (“man-in-the-middle”)
  - Eavesdropping
  - Masquerading, tampering, replay

Cryptography

- Etymology (Greek)
  - *kryptos*: hidden or secret
  - *grapho*: write
- Basic problem:
  - 2 agents (traditionally "Alice" and "Bob")
  - A & B want to exchange private messages
  - Channel between A & B is not secure ("Eve" is eavesdropping)
- Solution has other applications too
  - Protect *stored* data (*e.g.* on disk, or in cloud)
  - Digital *signatures* for non-repudiation
  - Secure *passwords* for authentication

Core Idea: A Shared Secret

- Alice & Bob share some *secret*
  - Secret can not be the message itself
  - Secret used to protect arbitrary messages
- Crude analogy: a padlock
  - Copies of the physical key are the secret
  - Alice puts message in box and locks it
  - Bob unlocks box and reads message
- But real channels are bit streams
  - Eve can see the bits!
  - Message must be garbled in some way
  - Secret is strategy for garbling/degarbling

Protecting Messages

- Alice garbles (encrypts) the message
- Sends the encrypted cipher-text
- Bob knows how to degarble (decrypt) cipher-text back into plain-text

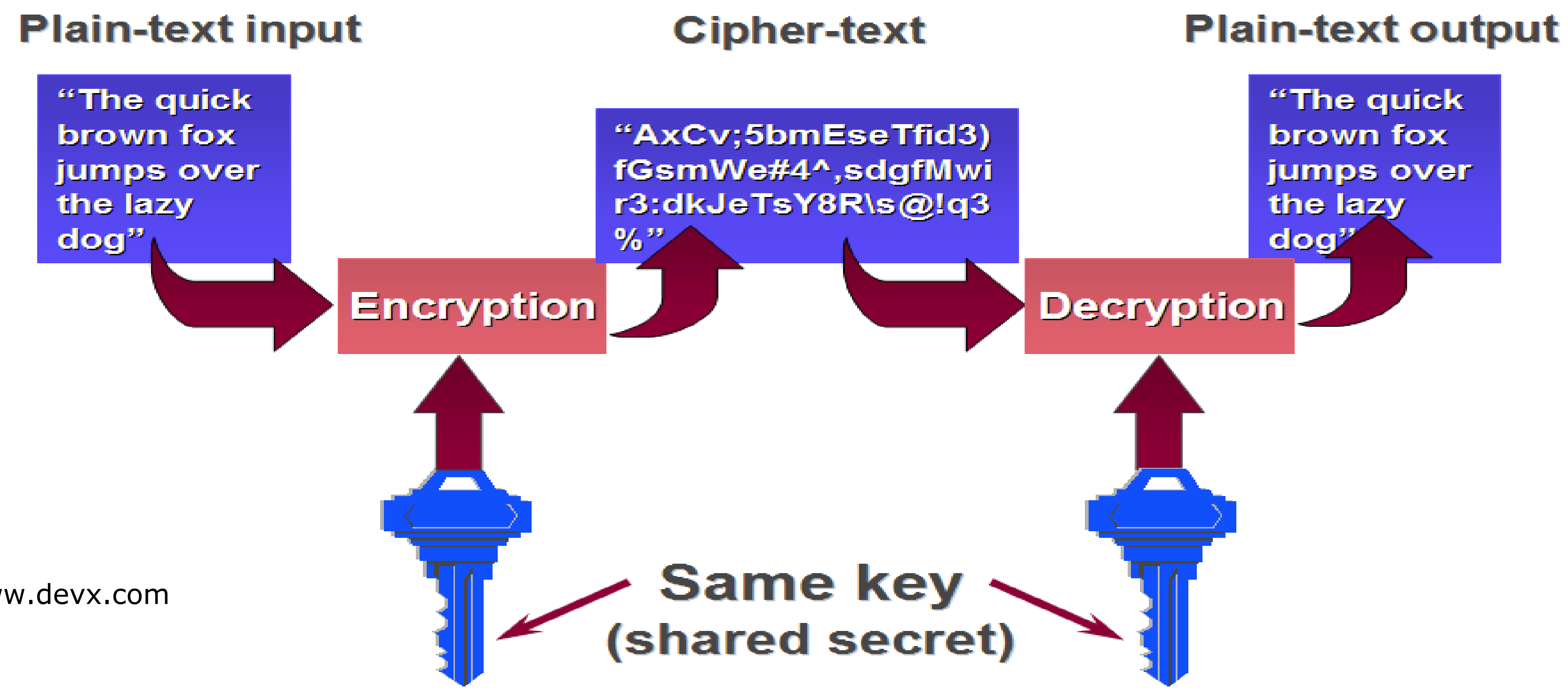
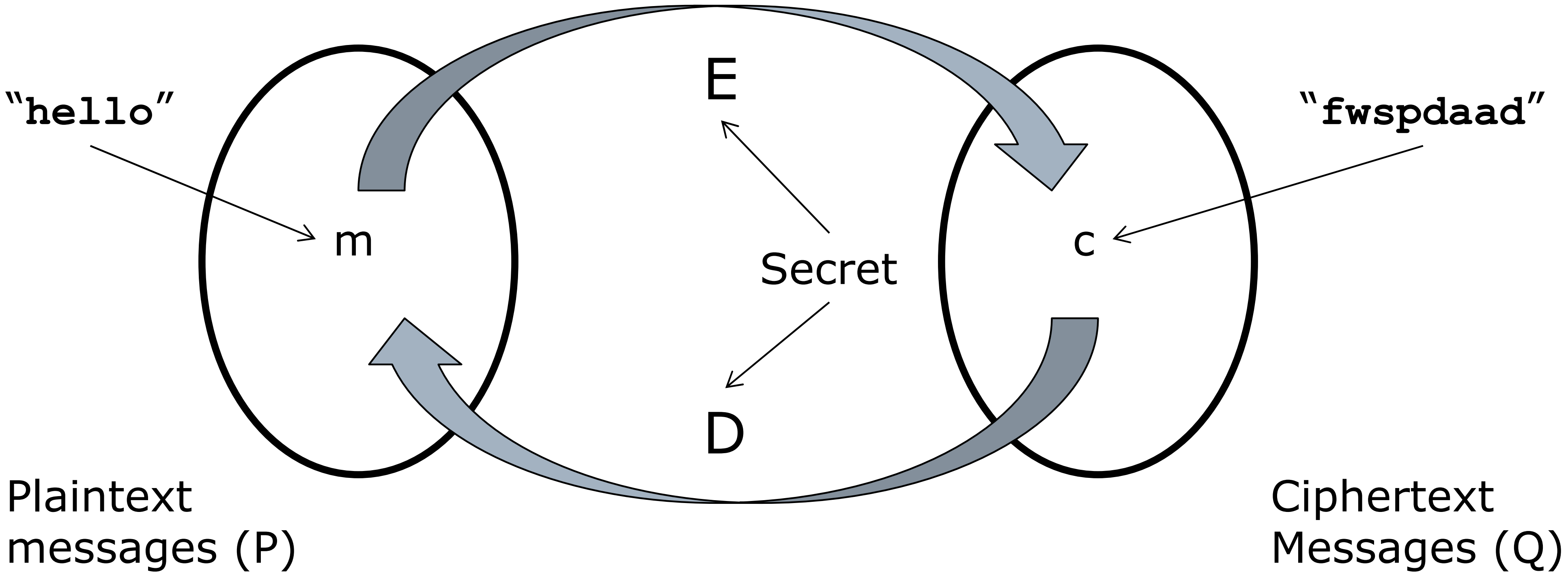

Plain-text input
Cipher-text
Plain-text output
"The quick brown fox jumps over the lazy dog"
"AxCv;5bmEseTfid3)fGsmWe#4^,sdgfMwir3:dkJeTsY8R\s@!q3%"
"The quick brown fox jumps over the lazy dog"
Encryption
Decryption
Same key
(shared secret)

Image: www.devx.com

Encryption/Decryption Function



E: P → Q
D: Q → P

E(m) = c
D(c) = m
i.e. D = E^{-1}

Note: often P = Q
So E is a *permutation*

Families of Encryption Functions

- Each pair of agents needs their own E
  - Many E's (& corresponding D's) needed
- But good E's are hard to invent
- Solution: design one (good) E, which is *parameterized* by a number
  - That is, have a huge *family* of E's: E_0, E_1, E_2, … E_K
  - Everyone knows the family of E's
  - Secret: *which* E_i is used (*i* is the key)

Classic Example: Caesar Cipher

- Shift each letter by *x* positions in alphabet
  - Example: *x* = 3
    `a` → **`d`**, **`b`** → **`e`**, **`c`** → **`f`**, **`d`** → **`g`**, **`e`** → **`h`**, …
  - The key is *x*
- Encode a string character-by-character
  - For m = "**`hello world`**", E_3(m) = "**`khoor zruog`**"
- Questions:
  - What is P (set of plaintext messages)?
  - What is Q (set of ciphertext messages)?
  - How many different ciphers?
  - Is this a strong or weak cipher?

Classic Example: Caesar Cipher

- Shift each letter by *x* positions in alphabet
  - *E.g. x* = 3
    `a` → **`d`**, **`b`** → **`e`**, **`c`** → **`f`**, **`d`** → **`g`**, **`e`** → **`h`**, …
  - The key is *x*
- Encode a string character-by-character
  - For m = "**`hello world`**", E_3(m) = "**`khoor zruog`**"
- Questions:
  - What is P (set of plaintext messages)?
    - The alphabet, ie {"**`a`**", "**`b`**", "**`c`**", "**`d`**", "**`e`**", …}
  - What is Q (set of ciphertext messages)?
    - The alphabet, ie {"**`a`**", "**`b`**", "**`c`**", "**`d`**", "**`e`**", …}
  - How many different ciphers?
    - 26
  - Is this a strong or weak cipher?
    - Weak: Just try all 26 possibilities

Generalized Caesar Cipher

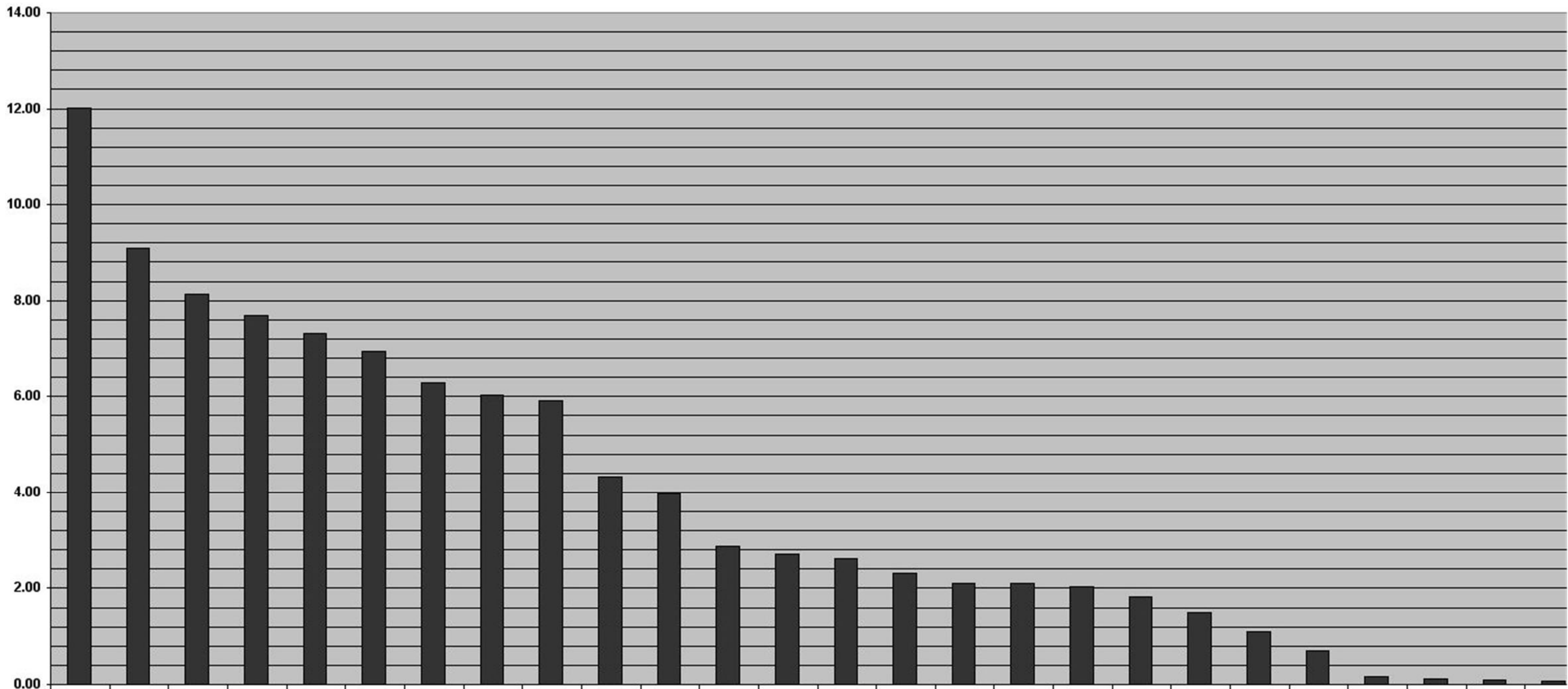
- Generalization: arbitrary mapping
  - Example: The qwerty shift

    `a` → `s`, `b` → `n`, `c` → `v`, `d` → `f`, `e` → `r`, …
  - For m = "`hello world`",
    E(m) = "`jraap eptaf`"
  - 26! possible ciphers… that's a lot!
    - Approximately 4 x 10^{26}
    - There are ~10^{18} nanoseconds/century
- Weakness?

Generalized Caesar Cipher

- Generalization: arbitrary mapping
  - Example: The qwerty shift
    `a → s, b → n, c → v, d → f, e → r, …`
  - For m = "**`hello world`**",
    E(m) = "**`jraap eptaf`**"
  - 26! possible ciphers… that's a lot!
    - Approximately 4 x 10^{26}
    - There are ~10^{18} nanoseconds/century
- Weakness?
  - In English text, letters appear in predictable ratios
  - From enough ciphertext, can infer E

Frequency Analysis



	E	T	A	O	I	N	S	R	H	D	L	U	C	M	F	Y	W	G	P	B	V	K	X	Q	J	Z
Frequency	12.02	9.10	8.12	7.68	7.31	6.95	6.28	6.02	5.92	4.32	3.98	2.88	2.71	2.61	2.30	2.11	2.09	2.03	1.82	1.49	1.11	0.69	0.17	0.11	0.10	0.07

Leon Battista Alberti

Santa Maria Novella Facade «» Leon Battista Alberti (1470) «» Florence

WW II: Enigma Machine

Polyalphabetic Cipher

- Alberti's idea: Use different E_i's within the same message
  - E("**`hello world`**") = E_a("**`h`**")E_b("**`e`**")E_c("**`l`**")E_d("**`l`**")E_e("**`o`**")…
- Alice & Bob need to agree on the *sequence* of E's to use
- Claude Shannon proved that this method is perfectly secure (1949)
  - Precise information-theoretic meaning
  - Known as a *one-time pad*

One-Time Pad

- Message is a sequence of bits

  $m_0\ m_1\ m_2\ m_3\ m_4\ m_5\ m_{6\ldots}$

- One-time pad is *random* bit sequence

  $x_0\ x_1\ x_2\ x_3\ x_4\ x_5\ x_{6\ldots}$

- E is bit-wise XOR operation, $\oplus$
- Cipher text is

  $m_0 \oplus x_0\ m_1 \oplus x_1\ m_2 \oplus x_2\ m_3 \oplus x_3\ m_4 \oplus x_4\ m_5 \oplus x_5\ m_6 \oplus x_{6\ldots}$

- Problem: Pad is long and cannot be re-used (hence cumbersome to share)
- In practice: pseudo-random sequence, generated from a seed (the key)
  - *Not* perfectly secure, in Shannon sense

Comparison: Stream *vs* Block

Stream Cipher

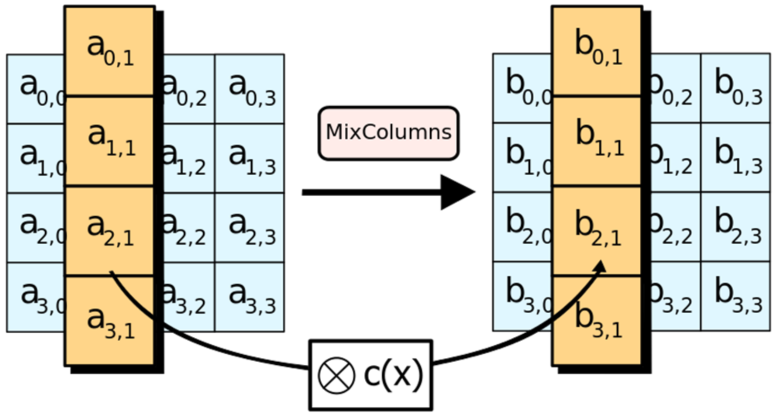
- Encrypts bit-by-bit
- $|P| = |Q| = 2$
- Few choices for E (roughly 2)
- Message can have any length

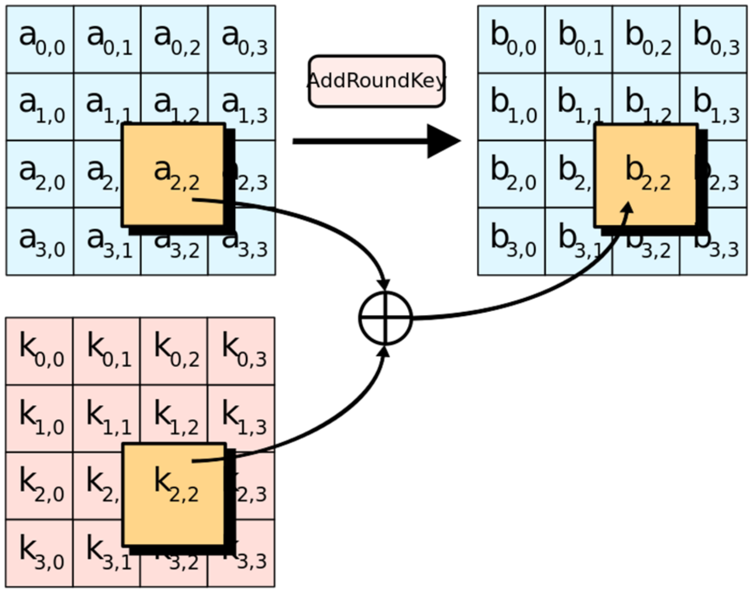
Block Cipher

- Encrypts a fixed-length (k-bit) sequence
- $|P| = |Q| = 2^k$
- Many choices for E (roughly $2^k!$)
- Padding added s.t. $|m|$ mod $k = 0$

Example of Block Cipher: AES

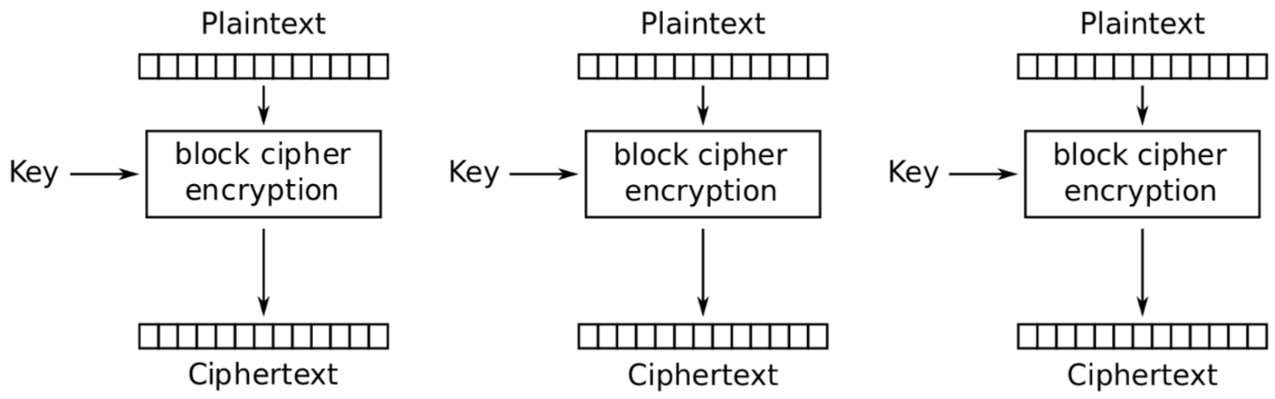
- Advanced Encryption Standard (2001)
  - Replaced DES (1977)
- Block size always 128 bits (4x4 bytes)
- Key size is 128, 192, or 256 bits
- Multi-step algorithm, many rounds


MixColumns
⊗ c(x)
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$ $b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

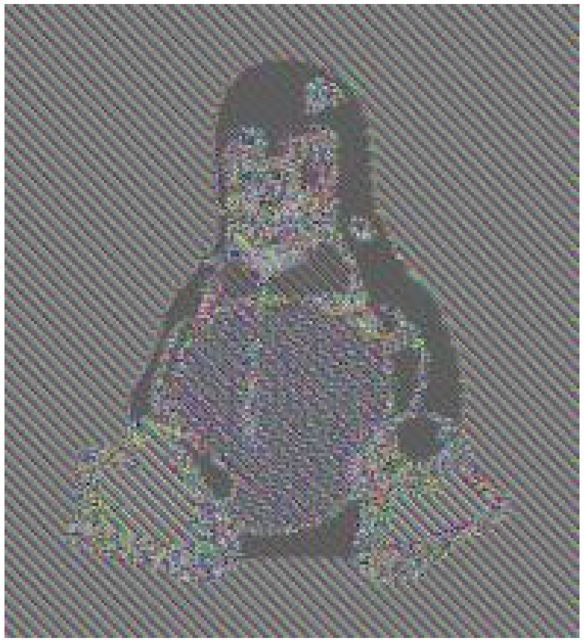

AddRoundKey
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{2,0}$ $a_{2,2}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$ $b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{2,0}$ $b_{2,2}$ $b_{2,3}$ $b_{3,0}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$ $k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$ $k_{2,0}$ $k_{2,2}$ $k_{2,3}$ $k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$

Limitation of Fixed Block Size

- Message can be longer than block size
- Reuse same E for each block?
  - Danger: Frequency analysis vulnerability
  - Don’t do this (for multiblock messages)!



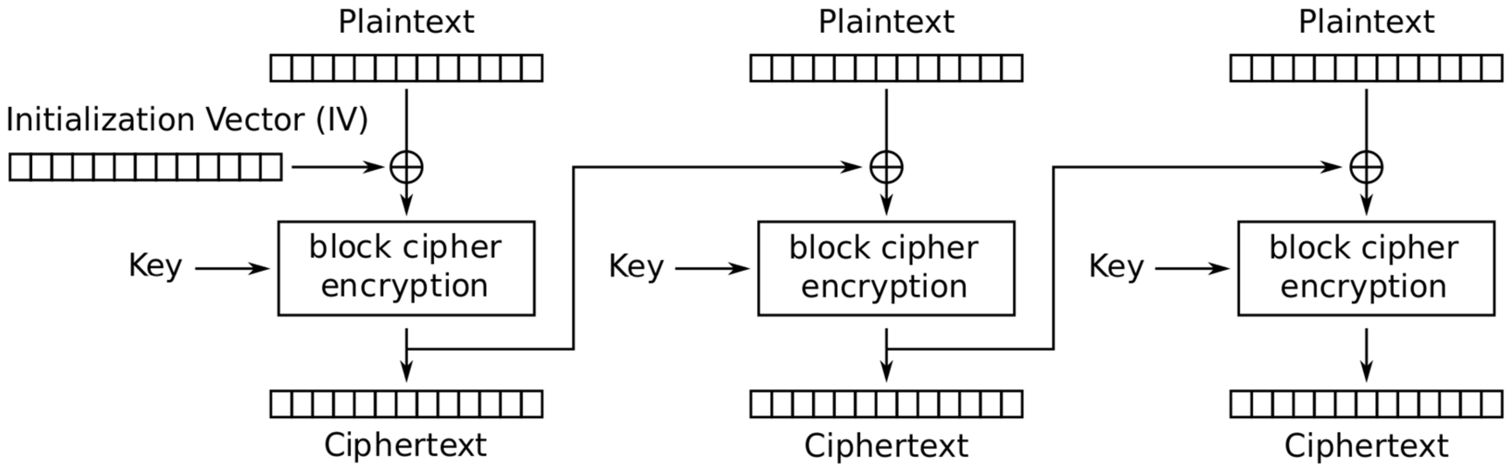
Electronic Codebook (ECB) mode encryption



https://en.wikipedia.org/wiki/Image:Tux_ecb.jpg
https://commons.wikimedia.org/wiki/File:Tux.jpg

Solution: Initialization Vector

- Add a random block to start
- Combine adjacent blocks to make ciphertext block
  - Many combination strategies (aka modes)



Cipher Block Chaining (CBC) mode encryption

Summary

- Cryptography
  - Encryption: Maps plaintext → ciphertext
  - Decryption is the inverse
- Symmetric-key encryption
  - Sender and receiver share (same) secret key
  - Stream ciphers work one bit at a time (*e.g.*, one-time pad)
  - Block ciphers work on larger blocks of bits (*e.g.*, AES)

Security: Cryptography II

Lecture 38

Symmetric Key

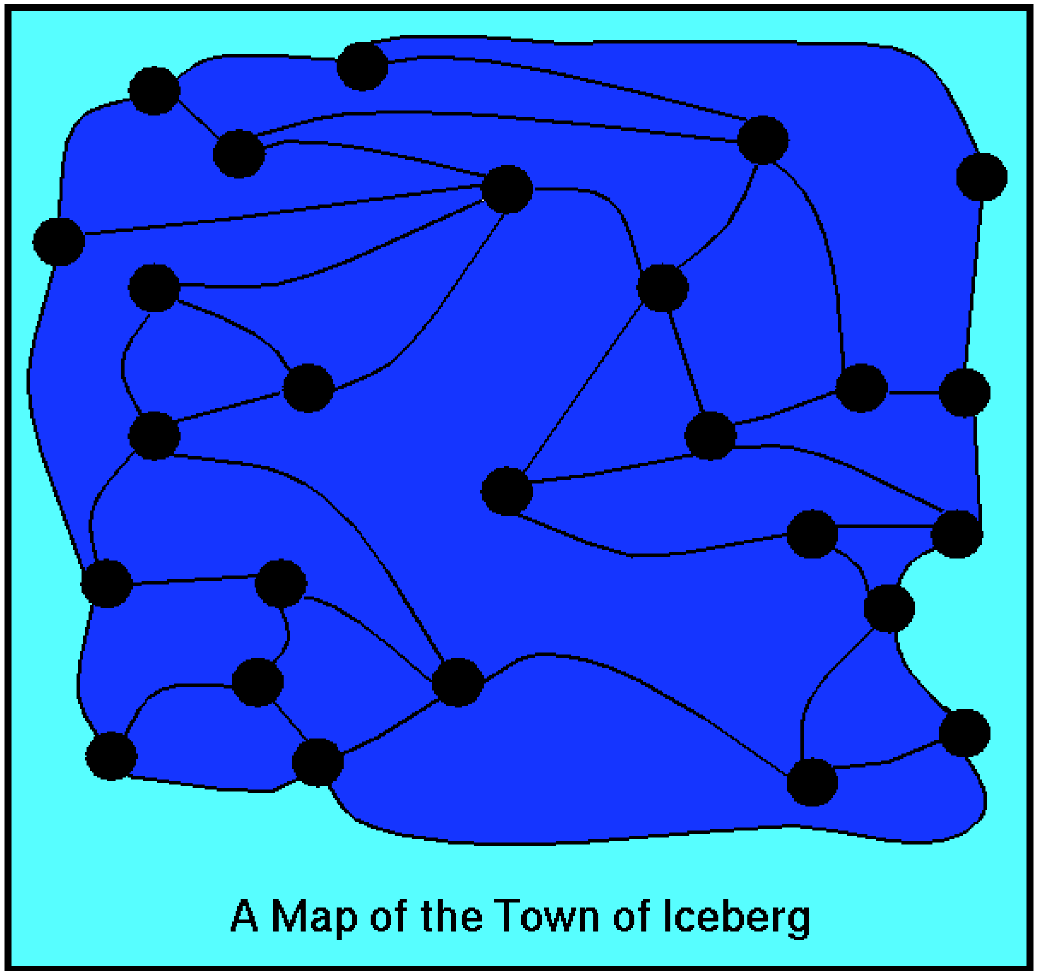
- For ciphers (so far): Knowing E is enough to figure out D (its inverse)
  - If you know how to encrypt, you can decrypt too
  - Known as a *symmetric key* cipher
- Example: Caesar cipher
  - If $E(m) = m + 3$, $D(m) = m - 3$
- Example: One-time pad
  - Use same pad and same operation (xor)
- Example: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

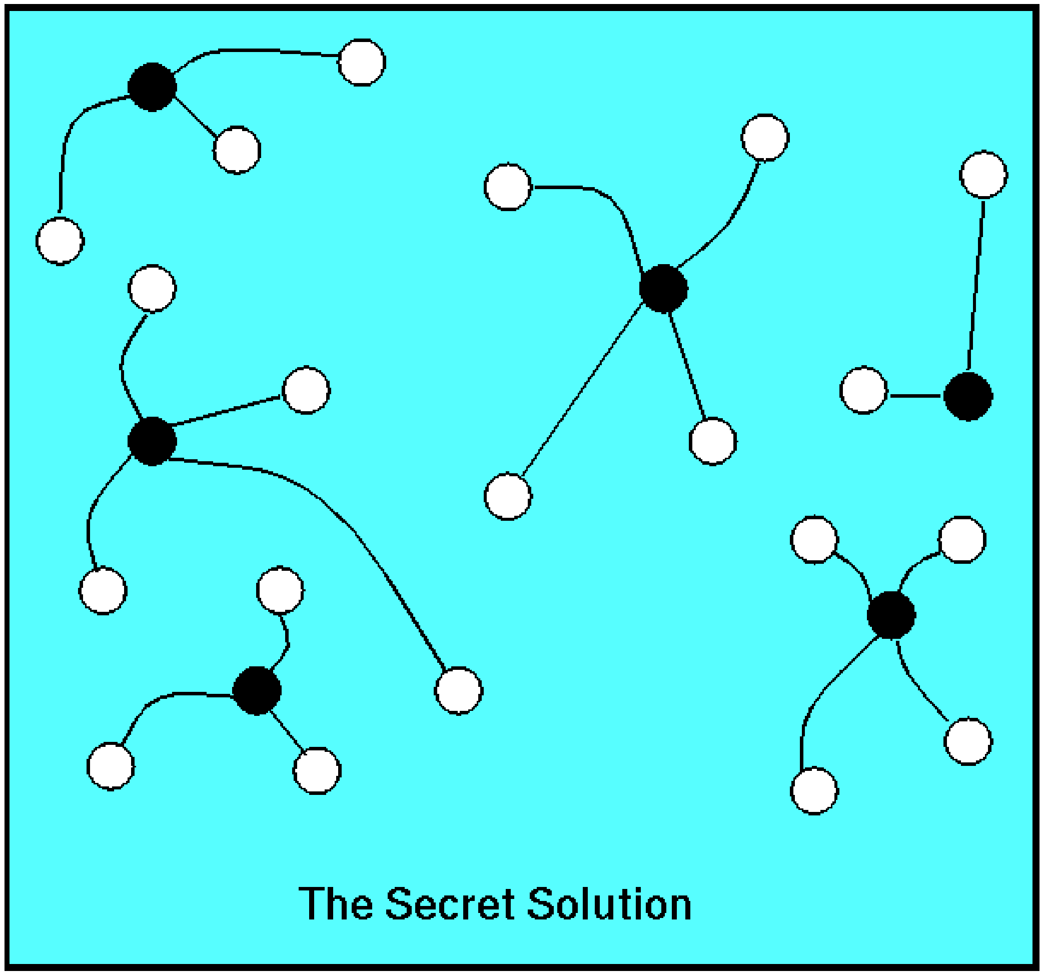
Example: Dominating Set

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



Example: Dominating Set

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



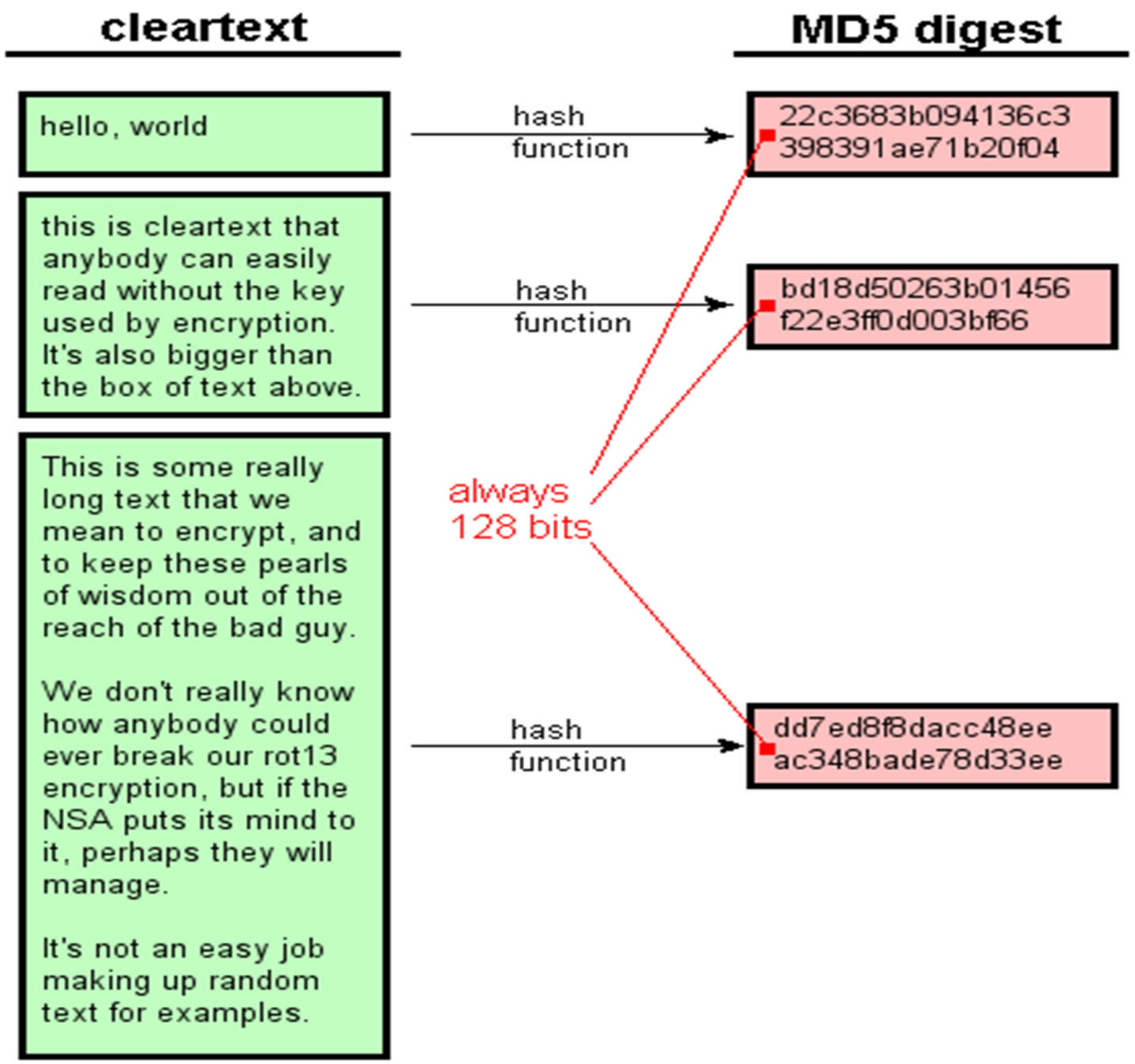
Example: Factoring

- Multiplying numbers is easy (*i.e.* fast)
  - Can multiply 2 n-bit numbers in 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

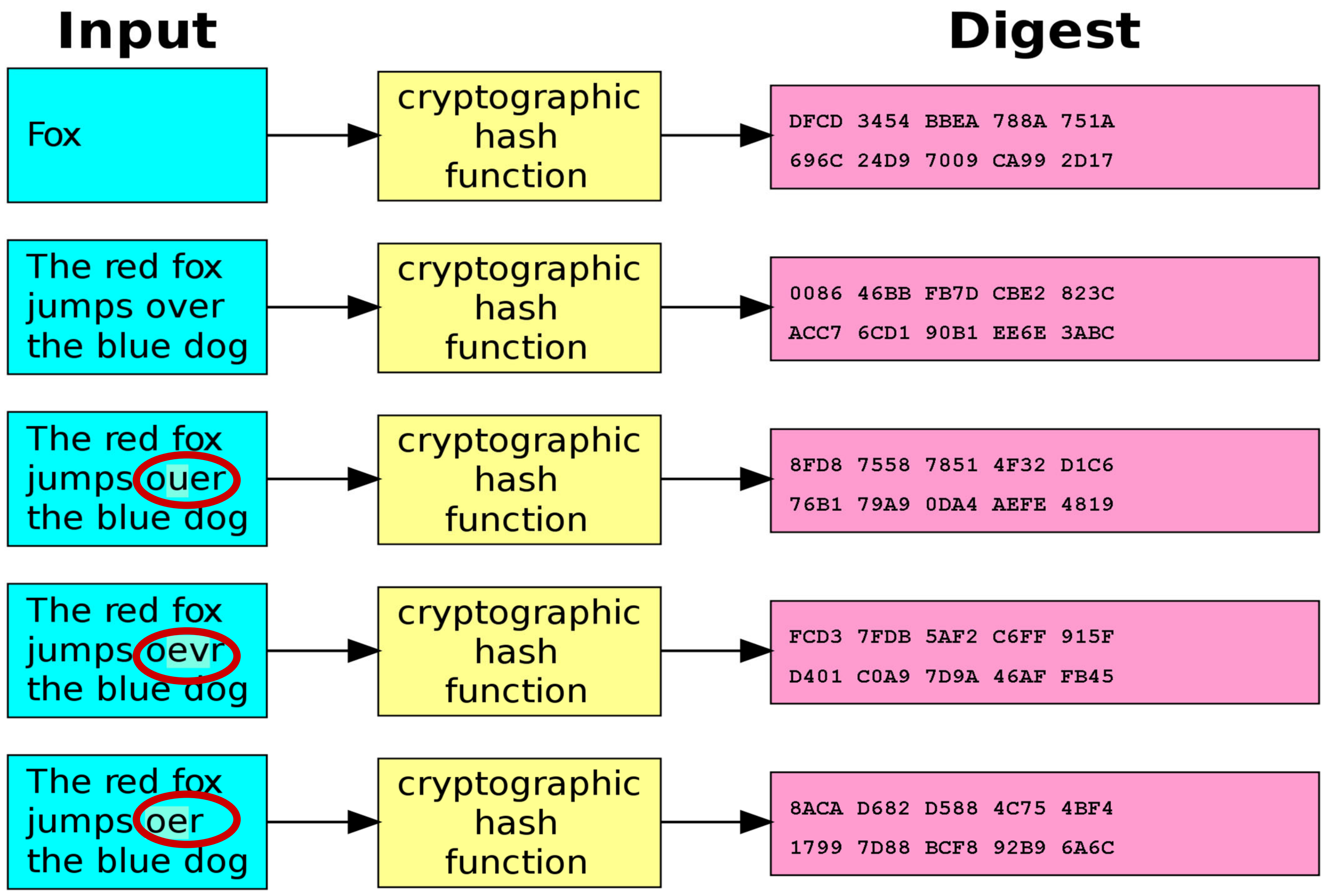
Cryptographic Hash Functions

- 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


cleartext
MD5 digest
hello, world
this is cleartext that anybody can easily read without the key used by encryption. It's also bigger than the box of text above.
This is some really long text that we mean to encrypt, and to keep these pearls of wisdom out of the reach of the bad guy.
We don't really know how anybody could ever break our rot13 encryption, but if the NSA puts its mind to it, perhaps they will manage.
It's not an easy job making up random text for examples.
hash function
hash function
hash function
22c3683b094136c3 398391ae71b20f04
bd18d50263b01456 f22e3ff0d003bf66
dd7ed8f8dacc48ee ac348bade78d33ee
always 128 bits

Crypto. Hash as Fingerprint


Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A
696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C
ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6
76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F
D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4
1799 7D88 BCF8 92B9 6A6C

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

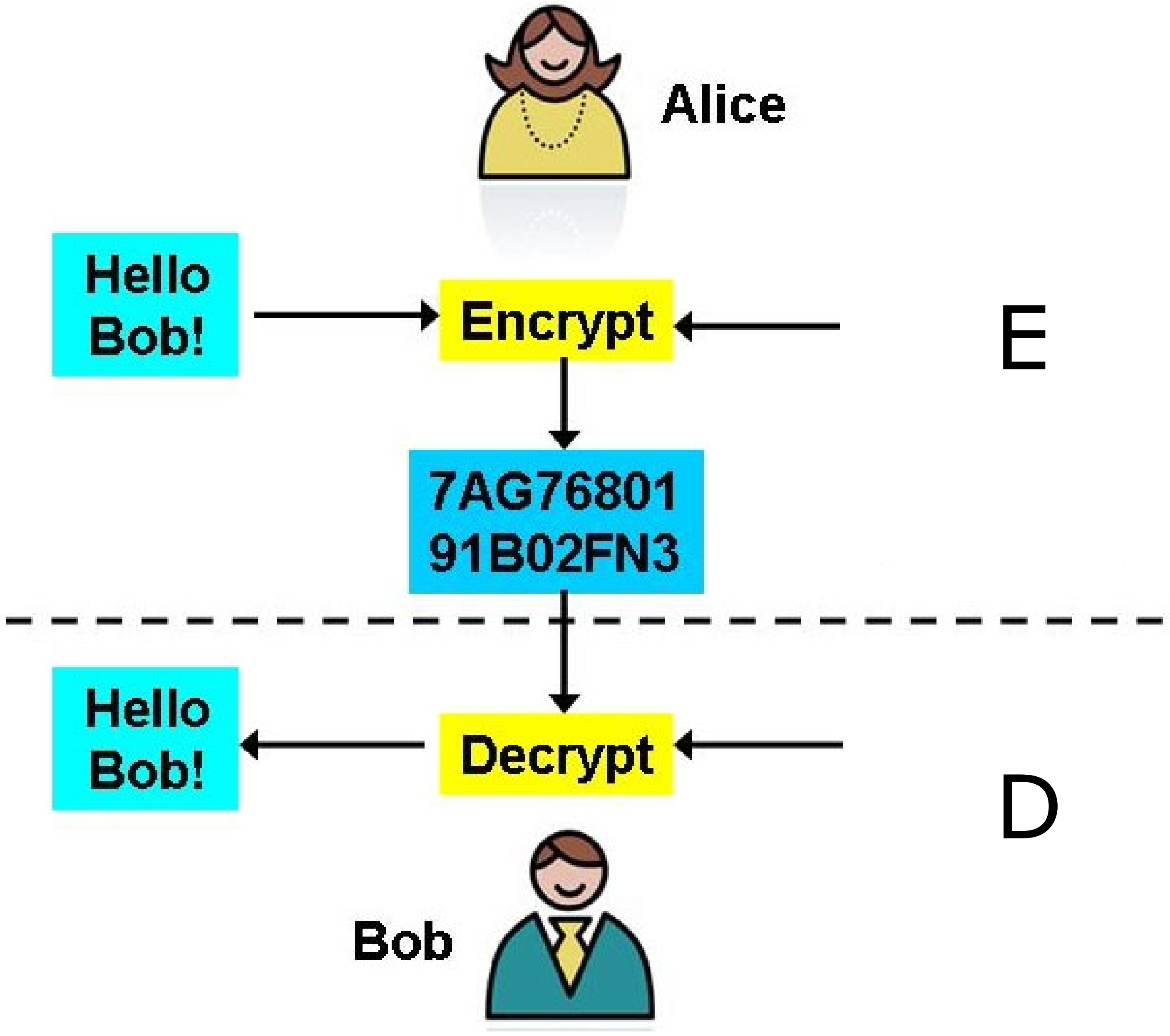
Role of Salt

- 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
  - But, in reality, 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



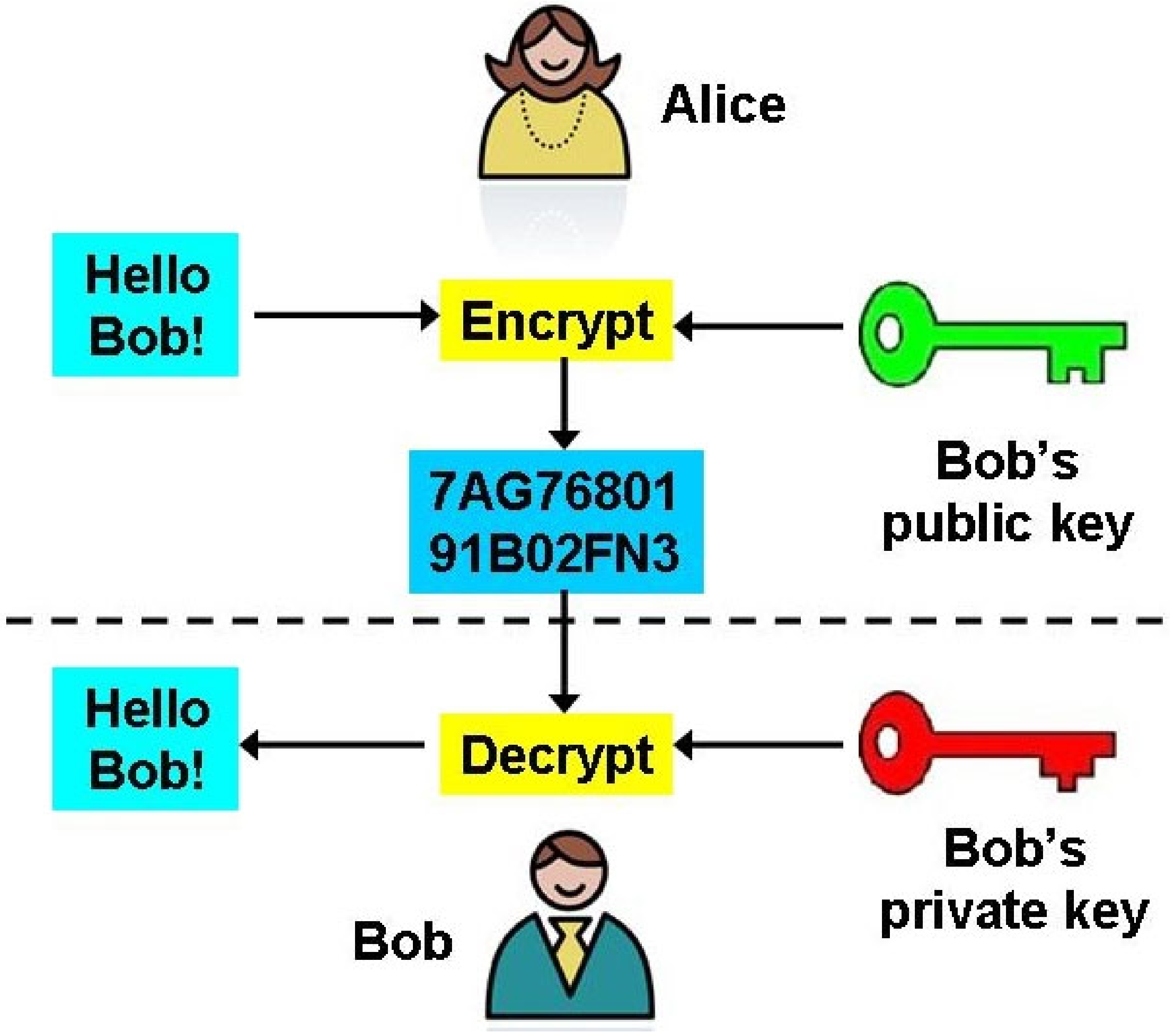
E

D

Public-Key Encryption

- Algorithms for E and D known by all
  - But parameterized by matched keys
- Asymmetry
  - Key for Bob’s E is *public*
  - Key for Bob’s D is *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:

  $D(E(m)) = (m^e)^d = m^{ed} = m, \bmod n$
- One-to-one, so backwards works too!

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

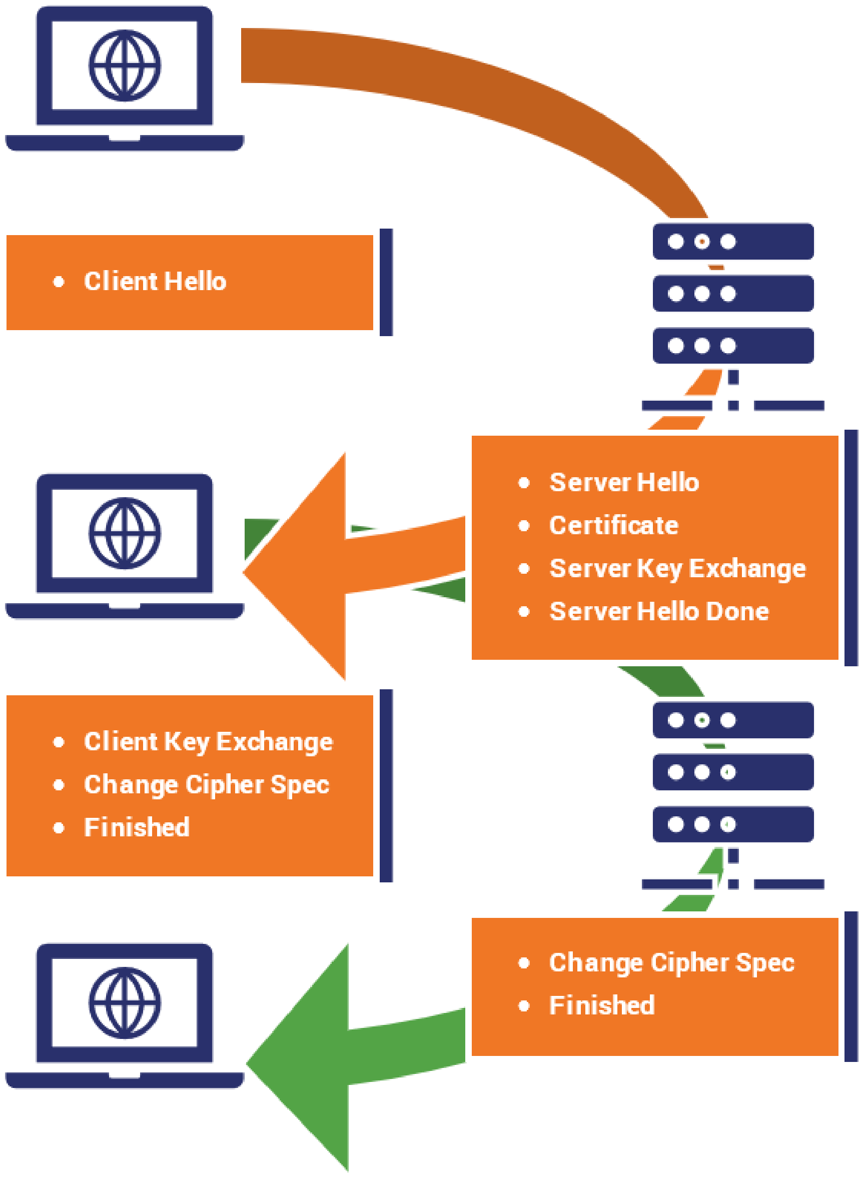
- Don’t try to roll your own crypto/security implementation
- Use (trusted) libraries
- Recognize role and importance of (eg):
  - Initialization vector
  - Cryptographic hash/digest
  - Salt
  - Private key vs public key

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

TLS 1.3: Handshake

- Certificate authority
  - Connects public key to identity
- Client:
  - Get server's public key
  - Make new (symmetric) session key
  - Sends this key to server (encrypted with public key)


Client Hello
Server Hello
Certificate
Server Key Exchange
Server Hello Done
Client Key Exchange
Change Cipher Spec
Finished
Change Cipher Spec
Finished