Cryptography: Integrity

Applied Information Security
Lecture 8

Last Lecture

you don't control the wire. (Dolev-Yao adversary).

- tamper, delete, delay: MitM! **active**

need to

- detect tampering of messages
  message == expected message
- detect spoofing
  sender == expected sender

with that, we can exchange keys… to create **secure channels**

“Securely”

- Confidentiality:
  only the intended recipient of a message should be able to read it.

- Integrity:
  An adversary cannot (undetectedly) tamper with a message.

- Authenticity [new!]:
  An adversary cannot (undetectedly) forge a message from either party

Today’s Topics

cryptography for authentication!

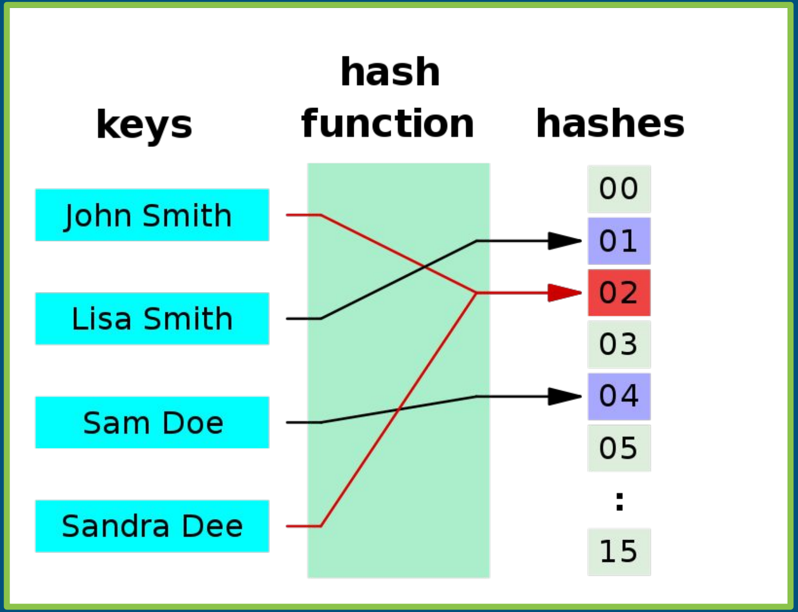
● hashing	SHA secure hash algorithms
● authentication	
○ Message	MAC message authentication code
○ User	RSA one-time pad
● cryptosystems	
○ in motion	TLS transport layer security
○ at rest	PGP pretty good privacy
○ off-the-record	OTR off the record
● password storage	

Hashing

MD5, SHA-1, SHA-2, SHA-3

Hash Functions

- Input: Arbitrary size string
- Output: fixed-size string
  - Obs: collisions may occur (see John Smith and Sandra Dee on the right)
  - Collisions expected; we are mapping from infinite to finite domains.
- Properties:
  - Easy to compute $h(m)$ given that we know m
  - For two identical inputs always produce the same output; $s = s' \Rightarrow h(s) = h(s')$



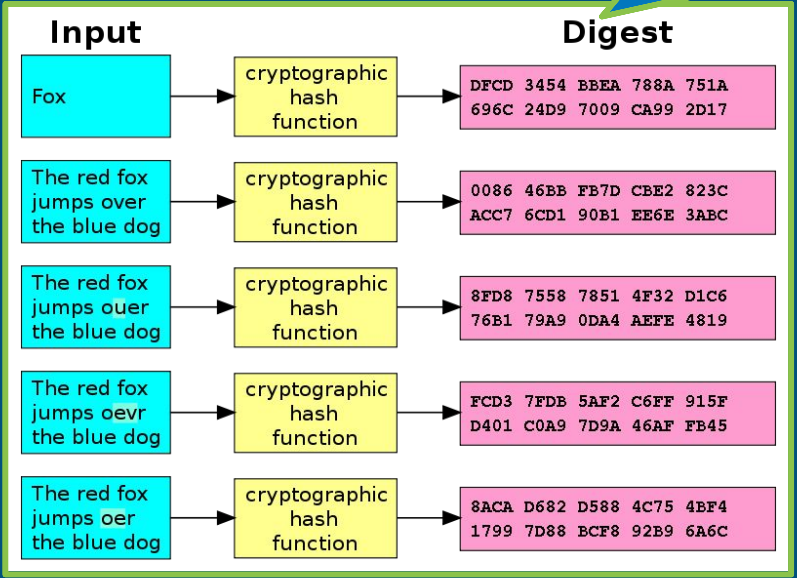
https://en.wikipedia.org/wiki/Hash_function

(Ideal) Cryptographic Hash Functions

The output of cryptographic hash functions is typically called *digest*

Additional stronger properties required:

- Infeasible to find a message given a hash value
  - One way function (remember colours video in Lec 7)
  - Infeasible to find m given that we know h(m)
- Infeasible to find two different messages with the same hash (collision resistance)
  - Infeasible to find h(m) = h(m’) where m ≠ m’
- Small modification on messages trigger significant changes
  - Avalanche effect
  - Similar m and m’ implies very different h(m) and h(m’)



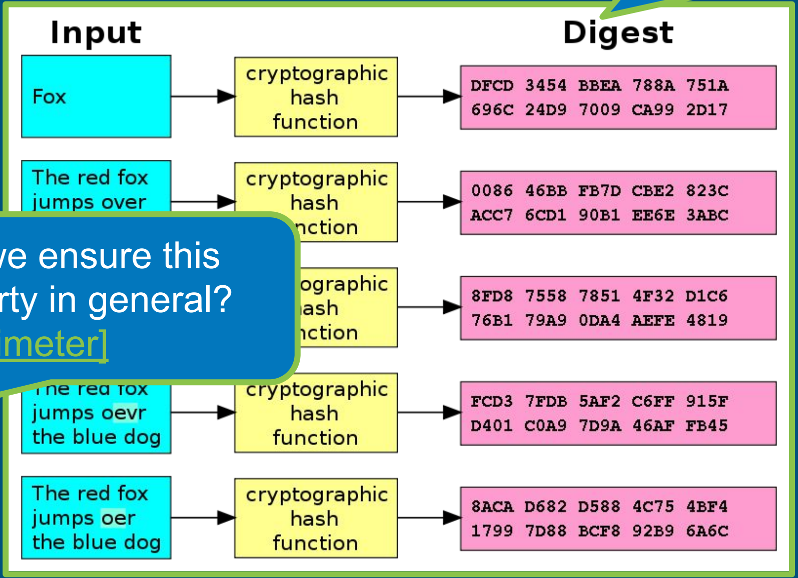
https://en.wikipedia.org/wiki/Cryptographic_hash_function

(Ideal) Cryptographic Hash Functions

Additional stronger properties required:

- Infeasible to find a message given a hash value
  - One way function (remember colours video in Lec 7)
  - Infeasible to find m given that we know h(m)
- Infeasible to find two different messages w[...] the same hash (collision resistance)
  - Infeasible to find h(m) = h(m’) where m ≠ m’
- Small modification on messages trigger significant changes
  - Avalanche effect
  - Similar m and m’ implies very different h(m) and h(m’)

The output of cryptographic hash functions is typically called *digest*



Can we ensure this property in general? [Mentimeter]

https://en.wikipedia.org/wiki/Cryptographic_hash_function

Is this property needed?

Real Cryptographic Hash Functions

- MD5 Ron Rivest (1991)
  - 128 bits output
  - Collision resistance broken
  - Can find collisions in seconds
- SHA-1 NSA (1995)
  - 160 bits output
  - Deprecated; broken for pdf files (http://shattered.io/)
- SHA-2 NSA (2001)
  - Family of functions with output sizes: 225, 256, 385, 513 bits
  - Not broken yet, believed to be vulnerable to same attacks than SHA-1
- SHA-3 NIST competition (2015)
  - Same output sizes as SHA-2
  - Strongest security properties

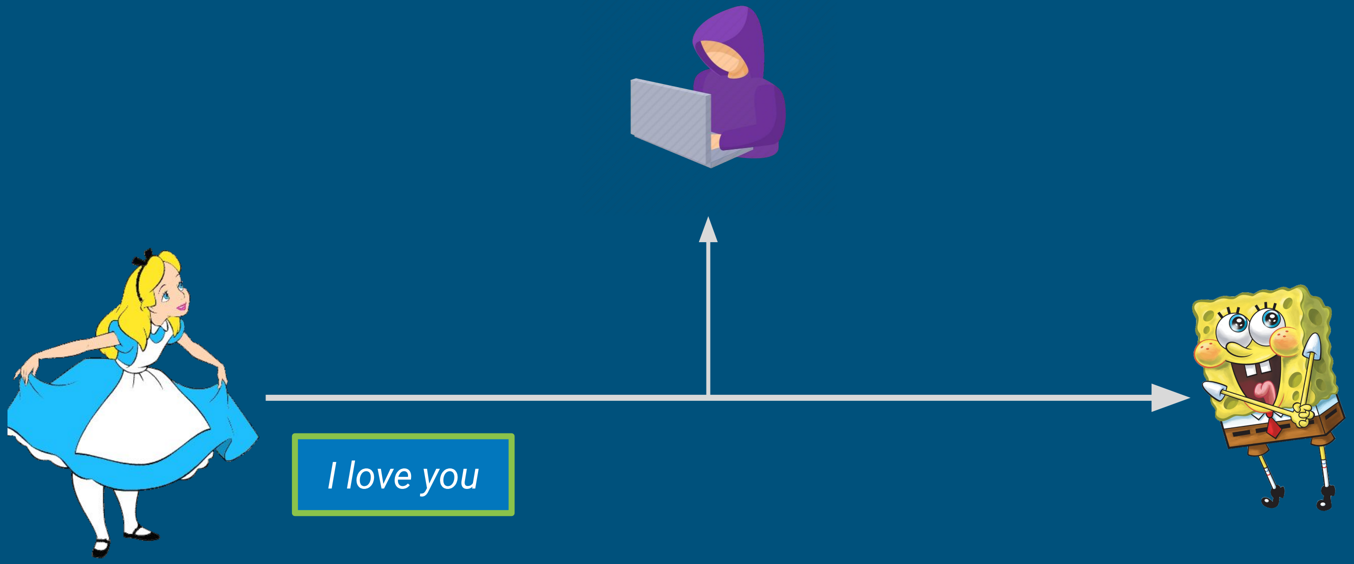
SHA-0 released and shortly after replaced by SHA-1 due to an undisclosed "significant flaw"

authentication

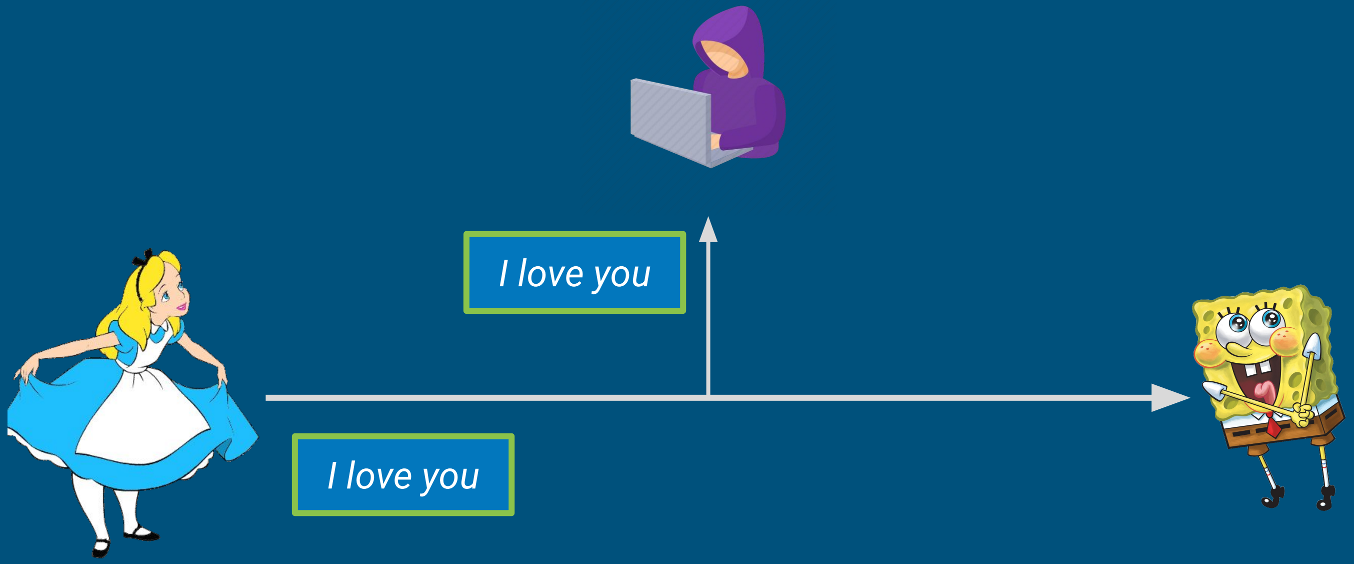
MAC, RSA

messages

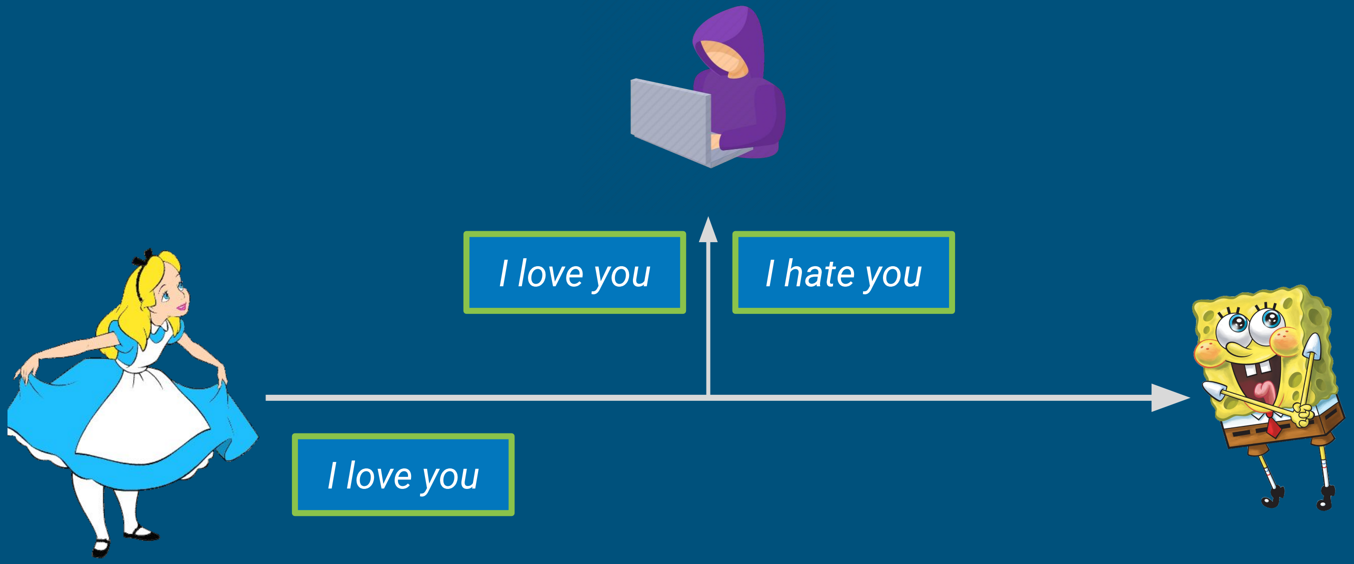
Authenticating Messages (Problem)



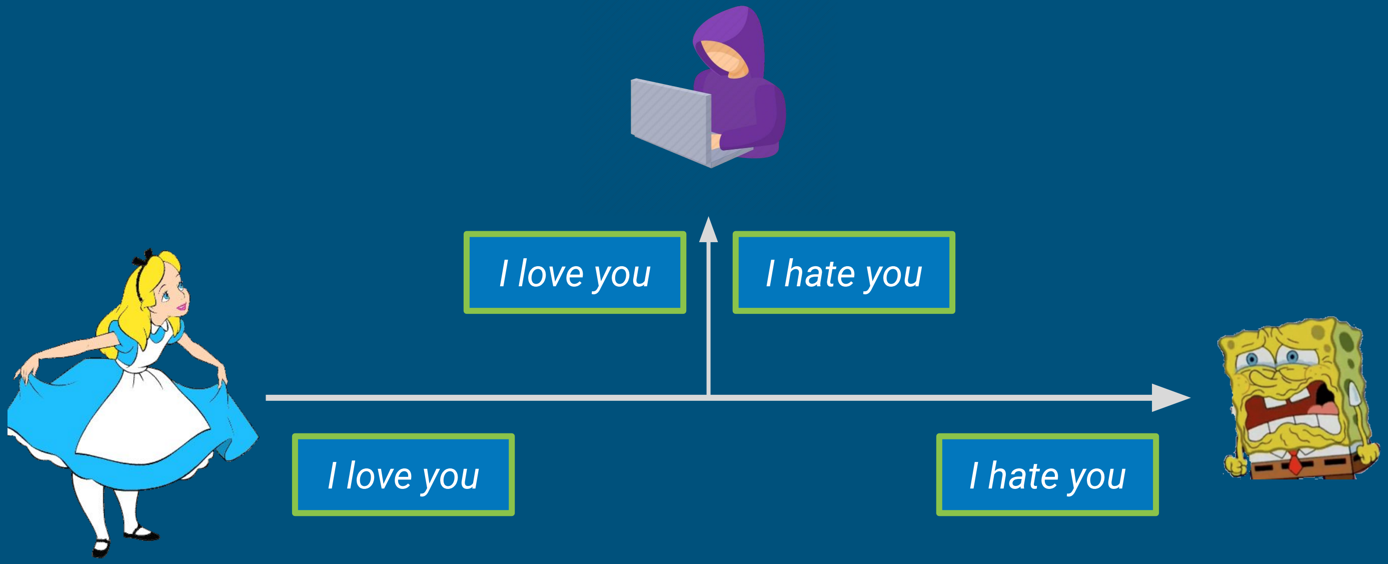
Authenticating Messages (Problem)



Authenticating Messages (Problem)



Authenticating Messages (Problem)



Authenticating Messages (Problem)

- The attacker can:
  - Tamper with the message
  - Delete the message
  - Delay sending

How can Bob know that the message was sent by Alice?

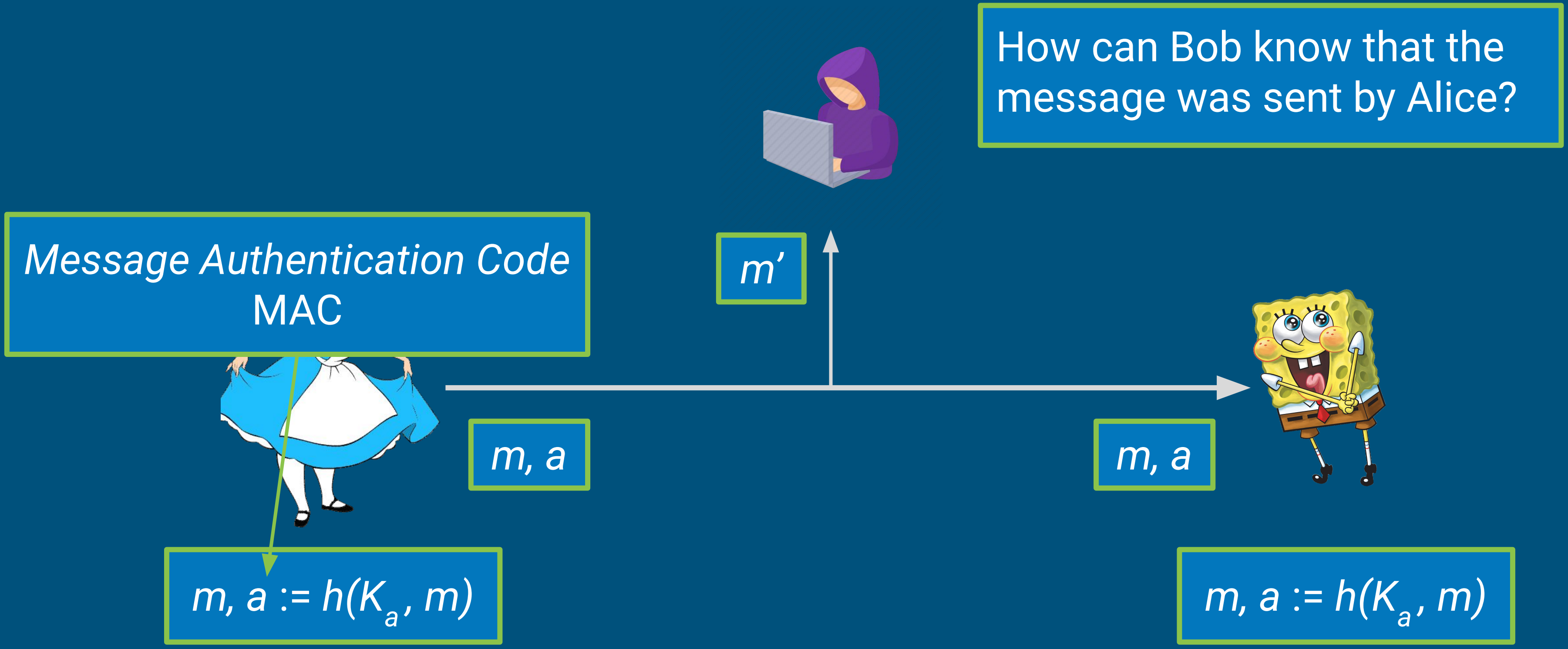
I love you

I hate you

I love you

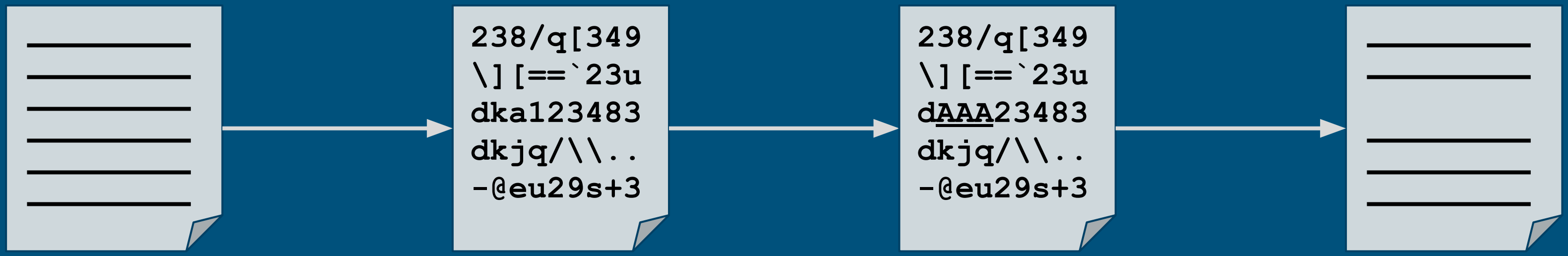
I hate you

Authenticating Messages (Solution)



What's the Problem?

- If I encrypt the message, wouldn't changes turn it rubbish? **NO!**
- The message could be a random number
    - Receiver cannot detect modifications on an unknown random number
- Some cipher modes only part of the message may be corrupted
    - Flipping bits
    - Change text
    - See last week's demo



Ideal MAC: Unforgeability Problem

Let an attacker select n different messages, for which he is given the MAC value. The attacker then has come up with a message n+1, with a valid MAC value.

$$\begin{array}{c} (m_1, a_1) \\ (m_2, a_2) \\ \ldots \\ (m_n, a_n) \end{array} \longrightarrow (m_{n+1}, a_{n+1})$$

CBC-MAC and CMAC

- Turns a CBC block cipher mode into a MAC function

- Encrypt the whole message as CBC and keep only the last block

$$H_0 = \mathrm{IV}$$
$$H_i = E(K, P_i \oplus H_{i-1}) \text{ for } i = 1, 2, \ldots, n$$
$$\boldsymbol{MAC = H_k}$$

- CMAC works similarly, except it xors H_k with a special value derived from the key prior encryption
    - Recommended and standardized

BIRTHDAY PARADOX

- Consider that there are 23 people in a room. The *birthday paradox* states that there is 50% chance that two people have their birthdays on the same day.
- Birthday attack: It is an attack where duplicate values, aka *collisions*, appear.
- Collisions are more frequent than intuition might suggest:
  - Consider a 64-bit block size for authentication. There are 2^{64} possible values.
  - Due to the birthday paradox after 2^{32} transactions a collision occurs, i.e., same value used twice
- This limits authentication security to *n/2* bits where *n* is the block size.

Attacks Example: CBC-MAC

- CBC-MAC suffers from some vulnerabilities that exploit the birthday paradox
  - When used carelessly; renew keys when approaching the limit $2^{n/2}$ messages
- Let M be a CBC-MAC function.
  - If $M(a) = M(b)$, then $M(a \| c) = M(b \| c)$ for any a, b, c. By the structure of CBC-MAC.
  - Consider a c of block length 1. Then we have:
  - $M(a \| c) = E_K(c \oplus M(a))$
  - $M(b \| c) = E_K(c \oplus M(b))$
  - Thus, $M(a) = M(b)$
- Attack in two stages
  1. Attacker collects MACs until he finds a collision. This takes 2^{64} steps for 128 block size.
  2. Next time the attacker receives a || c, he can replace it with b || c without changing the MAC.

MAC via Cryptographic Hash Functions

- Compute MAC using a cryptographic hash function
  - ~~MD5~~, ~~SHA-1,~~ SHA-2, and SHA-3
- Simple prefix-hashing $\mathrm{MAC} = h(K \,||\, m)$
  - Not only collisions, but
  - **<u>Insecure even if $h(\cdot)$ is a cryptographically secure hash function!</u>**
    - Vulnerable to length extension attacks
      - Internal state of the hash functions equals last digest
      - Given $h(m)$, the attacker can compute $h(m||m')$
- Instead use **HMAC**,
  - $\mathrm{MAC} = h(K \oplus a \,||\, h(K \oplus b \,||\, m))$ where *a* and *b* are derived keys (see Chapter 13 in Crypto101)
  - Prevents length extension attacks

PUTTING THINGS TOGETHER

- Using MACs we can ensure the **integrity** of a message
  - We can detect whether and attacker has tampered with the message
- Using block cipher modes we can ensure the **secrecy** of the message
  - We can prevent an attacker from reading the content of a message

BLOCK CIPHERS AND AUTHENTICATION

- Modern block ciphers modes include authentication
    - OCB: Offset Codebook Mode
    - CCM: Counter with CBC-MAC
    - GCM: Galois Counter Mode

- If not, combine block cipher modes with MAC functions
    - Encrypt and authenticate
    - Encrypt then authenticate
    - Authenticate then encrypt

ENCRYPT AND MAC

New operator, assign

1.
```
encryptionKey := gen(len)
macKey        := gen(len)
```

Initialization, keys shared by Alice and Bob

2.
```
Alice: ciphertext := E(encryptionKey, message)
       mac        := h(macKey, m)
```

MAC of the plaintext

3.
```
Alice -> Bob: mac || ciphertext
```

4.
```
Bob: m'   := D(encryptionKey, ciphertext)
     mac' := h(macKey, m')
     if (mac = mac')
     then output m'
     else abort
```

ENCRYPT AND MAC

- Pros: MAC and ciphertext can be computed in parallel

- Cons: MAC must offer confidentiality
    - A requirement never stipulated

- Example: ssh (secure shell) protocol
    - Recommends AES-128-CBC for encryption
    - Recommends HMAC with SHA-2 for MAC

MAC THEN ENCRYPT

```
1.  Alice: mac        := h(macKey, m)
           ciphertext := E(encryptionKey, mac || m)

2.  Alice -> Bob: ciphertext

3.  Bob: mac' || m' := D(encryptionKey, ciphertext)
         if (mac' = h(macKey, m'))
         then output m'
         else abort
```

MAC included in ciphertex

MAC THEN ENCRYPT

- Pros: Second most secure
- Cons: Computationally expensive
  - Always requires to sequentially compute decrypt and then mac

- Example SSL (Secure Socket Layers)
  - Recommends AES-128-CBC among others
  - For MAC it recommends HMAC, e.g., HMAC-SHA256

ENCRYPT THEN MAC

```
1.  Alice: ciphertext := E(encryptionKey, m)
           mac        := h(macKey, ciphertext)

2.  Alice -> Bob: mac || ciphertext

3.  Bob: mac' := h(macKey, ciphertext)
         if (mac = mac')
             then output D(encryptionKey, ciphertext)
             else abort
```

MAC of the ciphertex

ENCRYPT THEN MAC

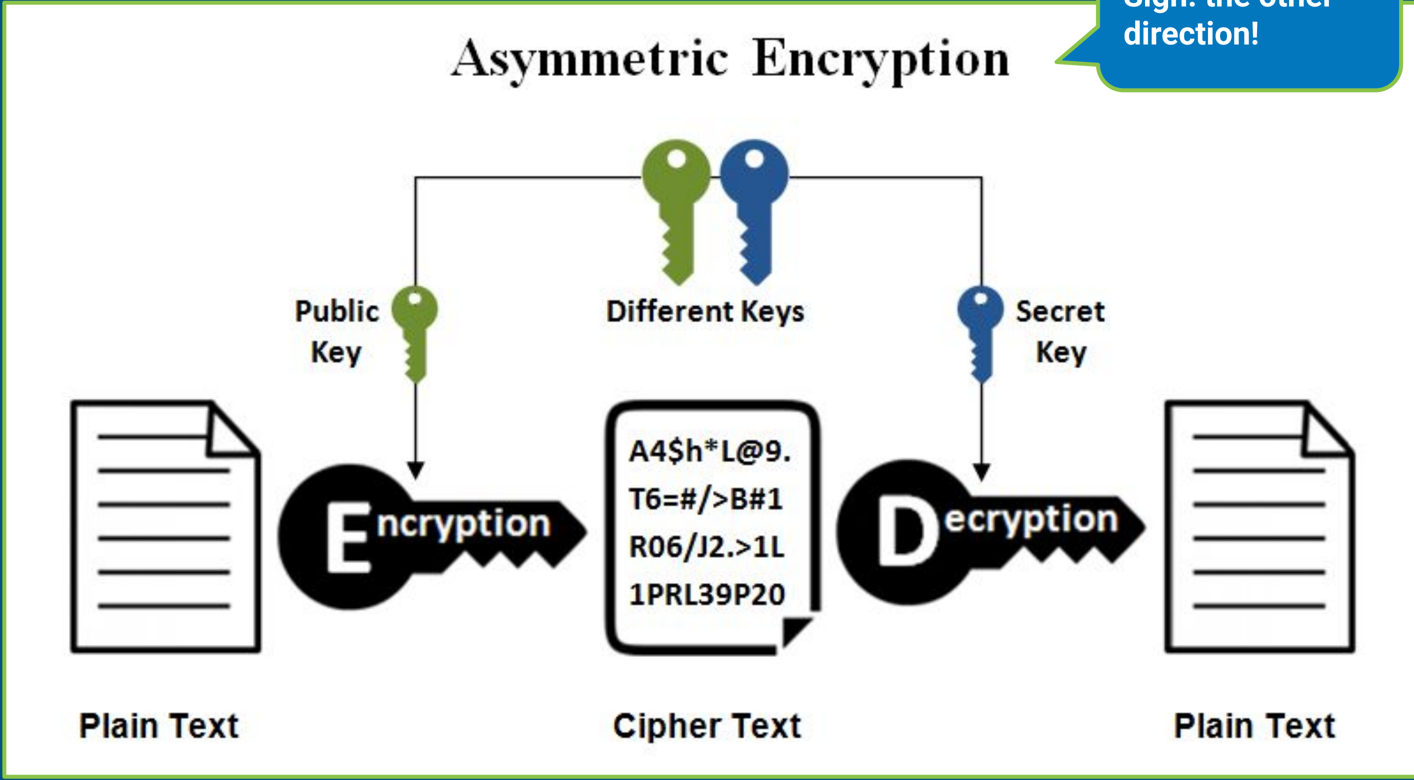
- Pros: Considered most secure version (see textbook)

- Pros2: Only computes decrypt if mac succeeds
  - Potential increase in complexity
  - Less likely DoS attacks

- Example IPSec
  - Recommends AES-CBC for encryption and HMAC for MAC
  - or AES-GCM

users

Sign: the other direction!



Asymmetric Encryption

Each Principal creates a key pair (S_i, P_i) where:

- $\mathbf{P_i}$ is called the **public key**
- $\mathbf{S_i}$ is called the **secret key**

Public keys are known to everyone

Secret keys only to the principal who created them

c

c

c

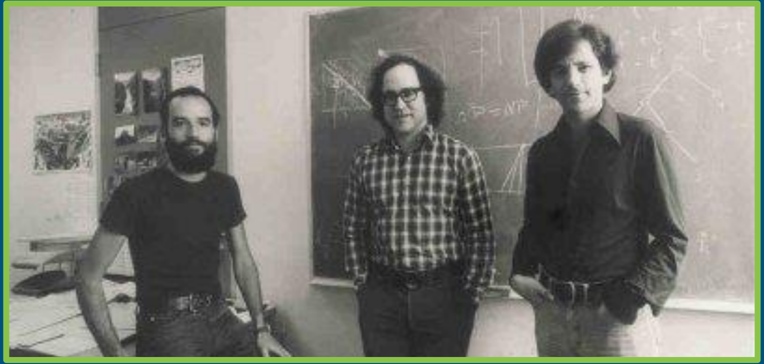
$m, c := E(\mathbf{P}_{\mathbf{Bob}}, m)$

$c, m := D(\mathbf{S}_{\mathbf{Bob}}, c)$

users

RSA

Rivest-Shamir-Adleman



By: Rivest, Shamir & Adleman, 1977

Based on the difficulty of factoring two large prime numbers

The factoring problem

Turing award recipients in 2002 contribution to making public-key cryptography useful in practice

Can be used for **encryption** and **signing**.

Slow. Huge key size.

RSA: KEY GENERATION

1. Choose two large primes p and q
2. Let $n := p*q$, n is the **modulus** for the public and private keys
3. Compute $\lambda(n) = \mathrm{lcm}(p - 1, q - 1)$
    - Carmichael's totient function
4. Choose an integer e such that
    - $1 < e < \lambda(n)$
    - $\gcd(e, \lambda(n)) = 1$
    - In other words, e and $\lambda(n)$ are coprimes
5. Compute d, as $d*e = 1 \bmod \lambda(n)$
    - Modular multiplicative inverse
6. **Public key** is (n,e)
7. **Private key** is (n,d)

See Chapter 12 in the textbook for details (and references therein)

RSA: ENCRYPTION AND DECRYPTION

- Encryption, given a message m

$$c = m^e \bmod n$$

- Decryption

$$m = c^d \bmod n$$

RSA: ENCRYPTION AND DECRYPTION

- Encryption, given a message m

$$c = m^e \bmod n$$

- Decryption

$$m = c^d \bmod n$$

Note that this operations are computationally more expensive than those of block ciphers

DIGITAL SIGNATURES

Public-key equivalent of **authentication**

m,s

$m, s := \sigma(\mathbf{S}_{Alice}, m)$

$\nu(\mathbf{P}_{Alice}, m, s)$?

RSA: SIGNING

- Signing (σ) a message m

$$h := \mathrm{hash}(m)$$
$$s := h^d \bmod n$$

- Verify (v)

$$\mathrm{hash}(m) = s^e \bmod n$$

Remember

$$c = m^e \bmod n$$
$$m = c^d \bmod n$$

Plain RSA not fully secure

- It is *deterministic*, same plaintext and key, always produces the same ciphertext

- Solution: Add a nonce to messages
  - Traditionally called padding
  - As described in the textbook, better use well-studied padding algorithms

Digital Signatures (Signing)

Public-key equivalent of **authentication**

m,s

$m, s := \sigma(\mathbf{S}_{\mathbf{Alice}}, m)$

$\nu(\mathbf{P}_{\mathbf{Alice}}, m, s)?$

RSA Signatures

- Signing (σ) a message m

$$h := hash(m)$$
$$s := h^d \bmod n$$

The sender signs with her Secret key *(d,n)*

- Verify (ν)

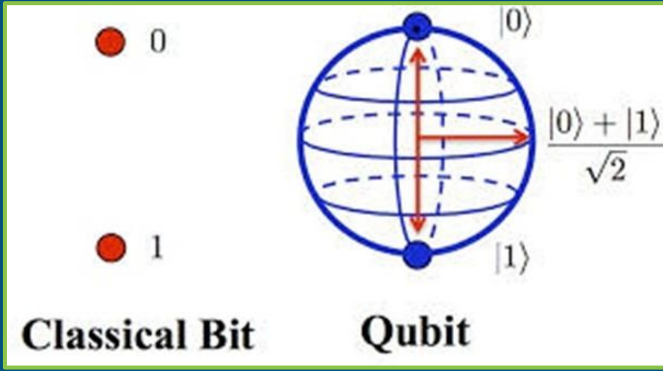
$$hash(m) = s^e \bmod n$$

Cryptographic hash function

The recipient verfies with the Public key *(e,n)* of the sender

RSA, Post-Quantum

the factoring problem can be efficiently solved by a quantum computer w/ sufficiently many qubits. →



- ⇒ RSA 2048 can be cracked by such a computer! (w/ 4099 qubits)
- Q-Day: when such computers become available.
  - several companies already possess quantum computers w/ ~100 qubits.
  - IBM announced it would have a ~1000-qubit quantum computer in the cloud in 2023.
  - estimation: QDay in 5-30 years.

similar to how we got AES

NIST Post-Quantum Cryptography Standardization, 2016: call for proposals for post-quantum safe public-key ciper. candidates passed scrutiny in 2022.
keep an eye on this, soon we'll have new ciphers.

ciphers are a moving target; new threats (attackers, tech) ⇒ new ciphers needed

Digital Signature Algorithm (DSA)

- Standardized by NIST in 1991
- Public key signing algorithm
  - Cannot encrypt/decrypt

Implementations details in Chapter 12 of Crypto101 but similar to Diffie-Hellman

- Security based on the complexity of the *discrete logarithm problem*
  - Recall Diffie-Hellman (Lec 7)
  - RSA relies on complexity of the prime factorization problem
- Security heavily relies on entropy, secrecy, and uniqueness of a random signature chosen for signing
  - Break any of those ⇒ attackers can recover the secrets

cryptosystems

TLS, PGP, OTR

cryptosystems

in motion

transport layer security
TLS

TLS

transport layer security

secures traffic on the Web: **https**

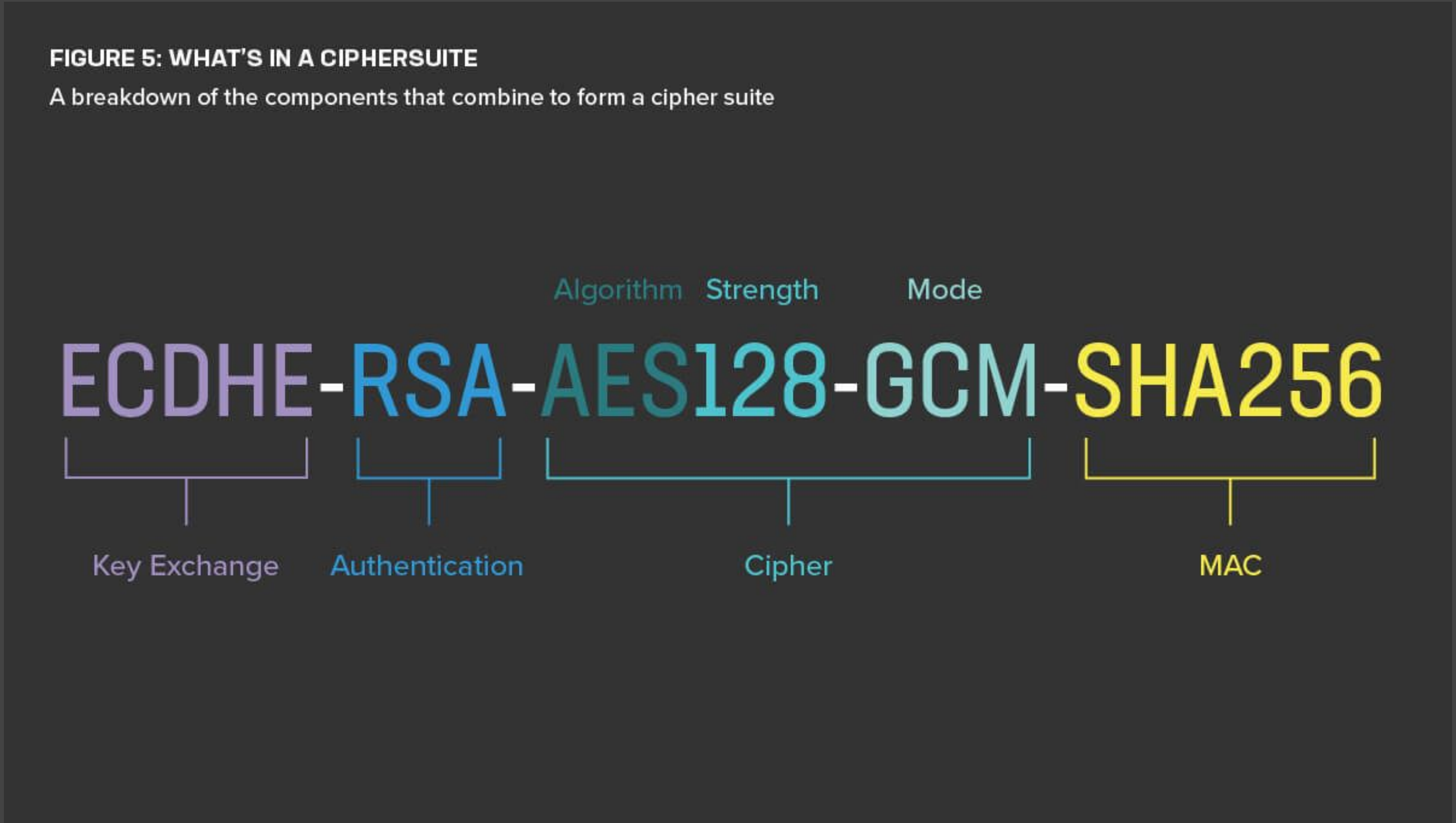
standard published by the IETF.

what's in the cryptosystem:

- integrity: MAC
- confidentiality: DH
- key sharing: RSA/DSA

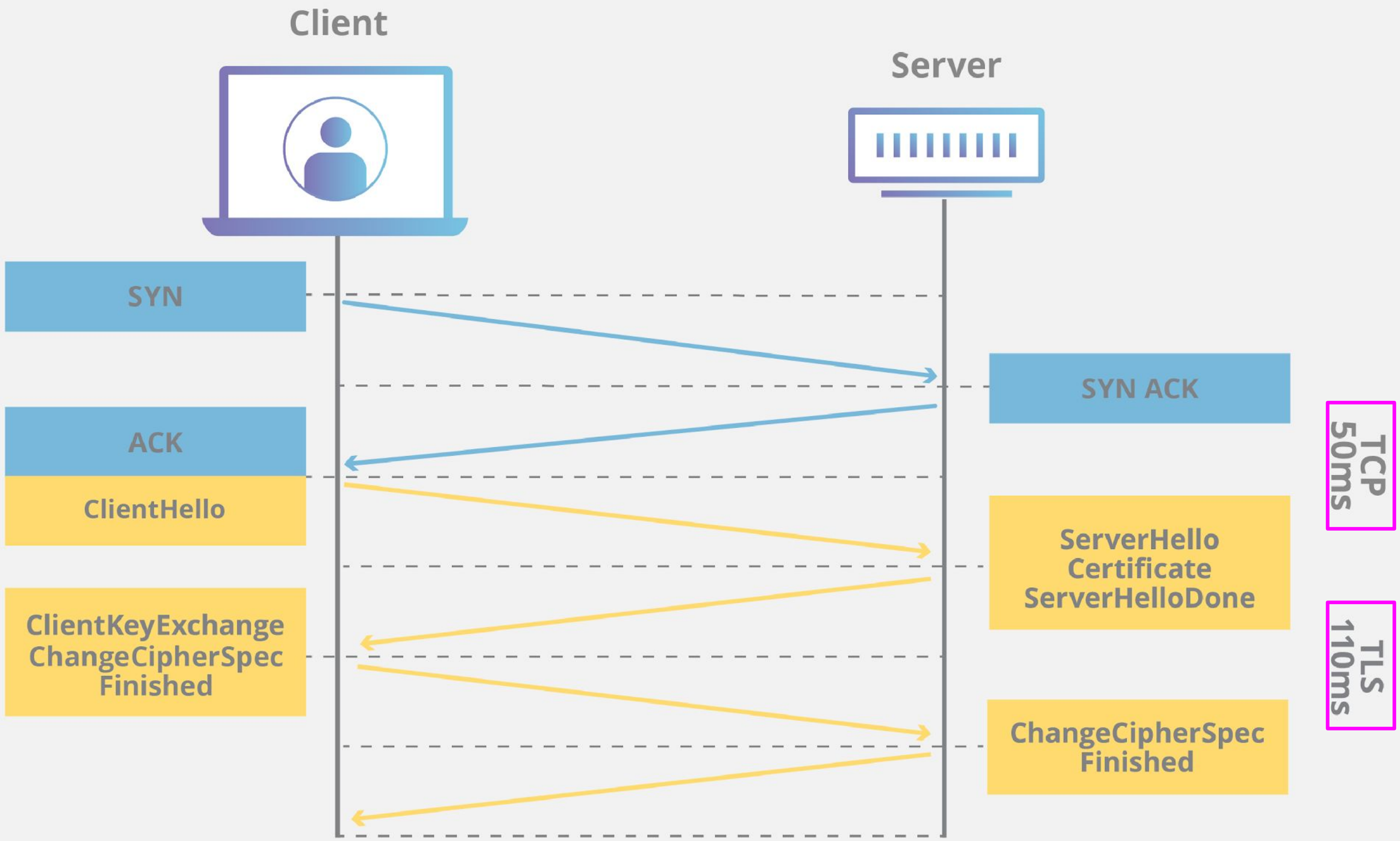
server & client must agree on which algorithms to use: TLS Handshake

cryptosystems - in motion - TLS



cryptosystems - in motion - TLS

(TCP +) TLS Handshake

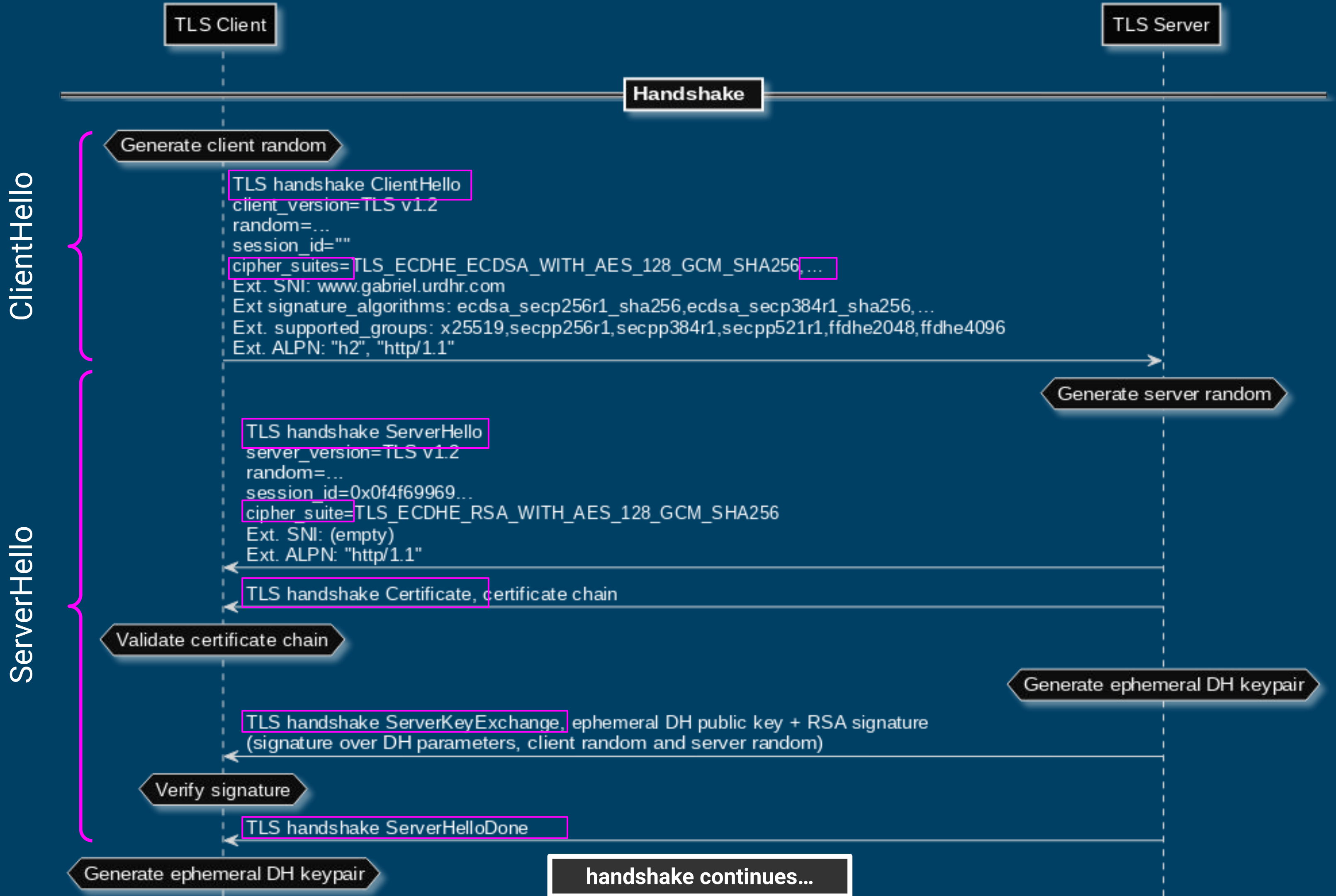


TLS Handshake

1. Client: ClientHelloMessage
   a. Maximum TLS version it supports.
2. Server: ServerHelloMessage
   a. Protocol version, random version, cipher suite and compression method
3. Server: Certificate
   a. Sends server certificate and
4. Server: ServerKeyExchange (optional)
5. Server: CertificateRequest
   a. Request the certificate of the Client
6. ServerHelloDone
   a. Server done with handshake
6. Client: Certificate
   a. Sends client certificate
7. ClientKeyExchange
   a. PreMasterKey encrypted with public key of server certificate
8. Client: CertificateVerify
   a. Signature over previous messages using private key. Allows server to confirm client’s access to private key
9. Client: ChangeCipherSpec
   a. From now on auth and enc
10. Server: ChangeCipherSpec
    a. From now on auth and enc

TLS Handshake, details

cryptosystems - in motion - TLS

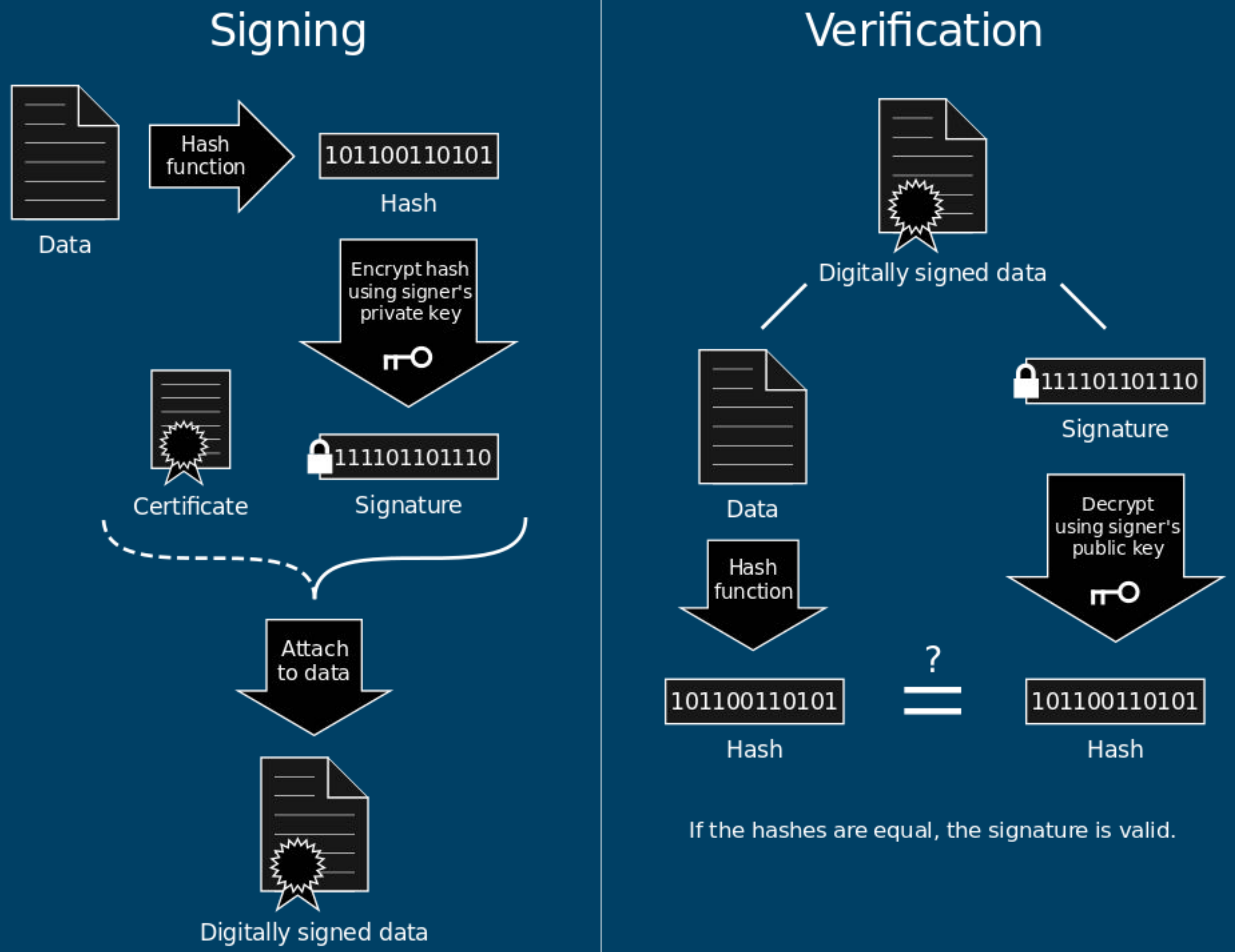


TLS Handshake, details

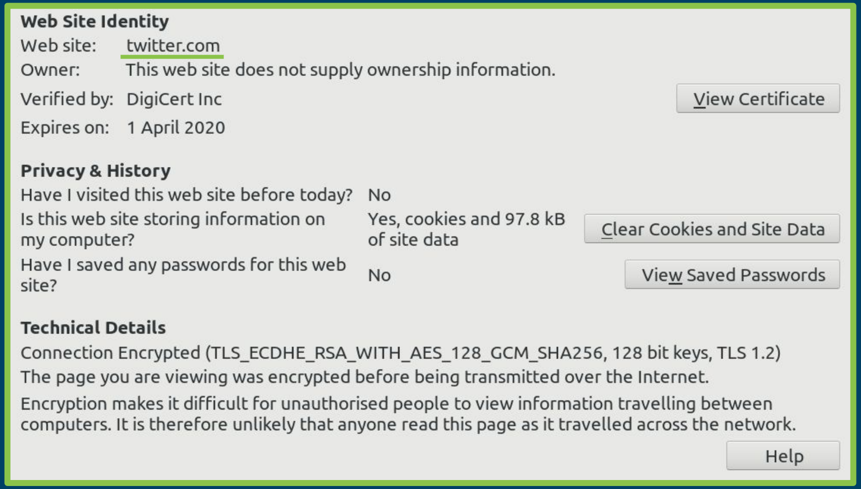
cryptosystems - in motion - TLS

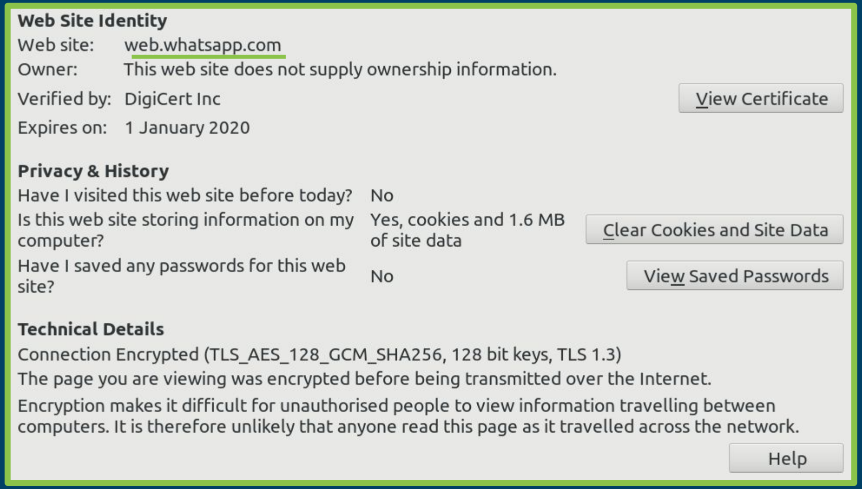
Hashes

cryptosystems - in motion - TLS



TLS 1.2 VS TLS 1.3





Downgrade attack

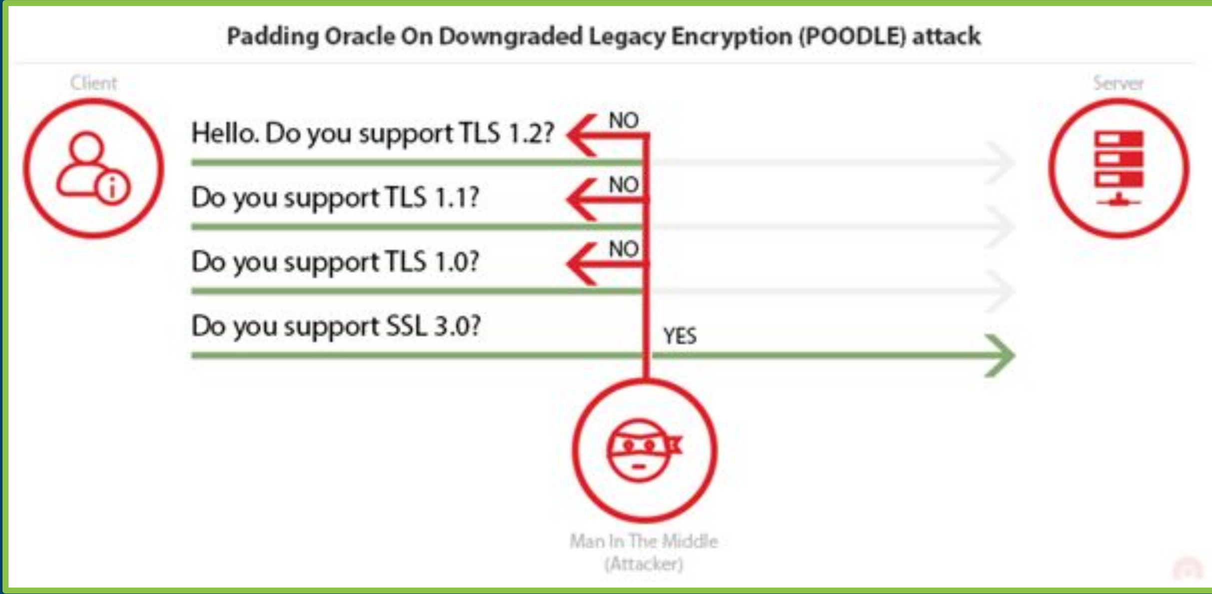
POODLE attack.

<u>fix:</u> disable SSL 3.0 support.
on the server.

in fact, disable

- TLS 1.1 (predictable IV)
- TLS 1.2

while you're at it!!!



cryptosystems - in motion - TLS

Session Hijacking

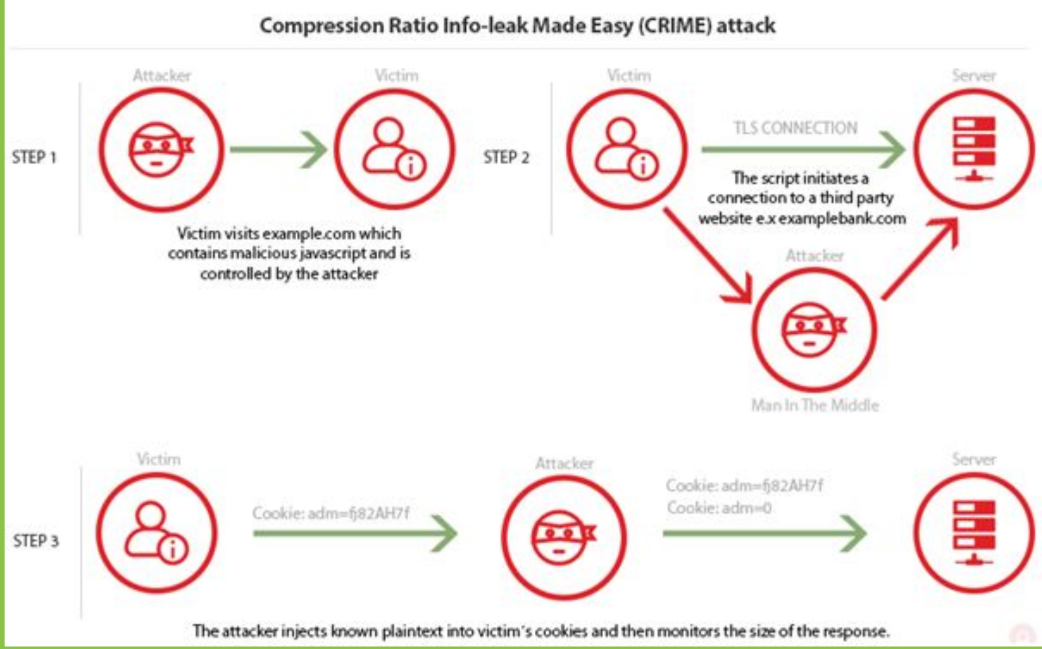
CRIME attack. TLS compression.

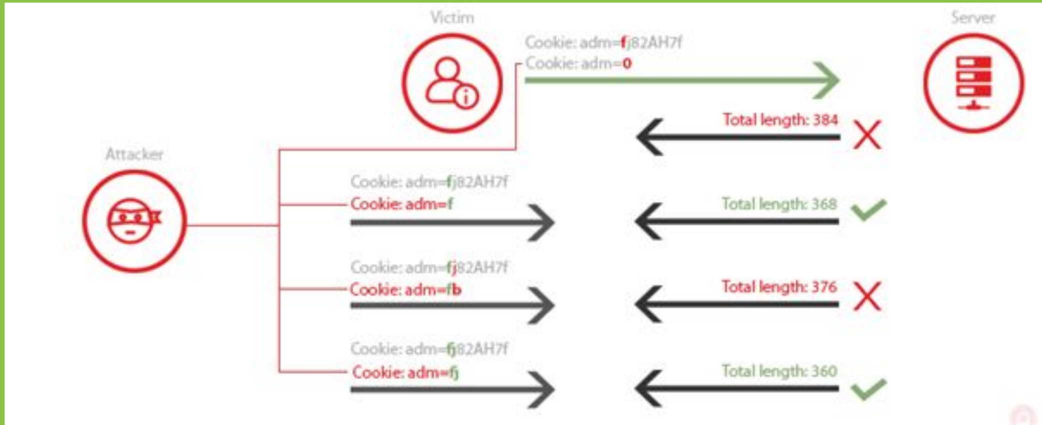
fix: disable TLS compression

compression was actually recommended in Standards!

BREACH: HTTP compression

fix: disable HTTP compression





victory!

secure, authenticated connections to anyone!

… but, they are who they say they are?
how do I know (unless they hand me the key in person)?

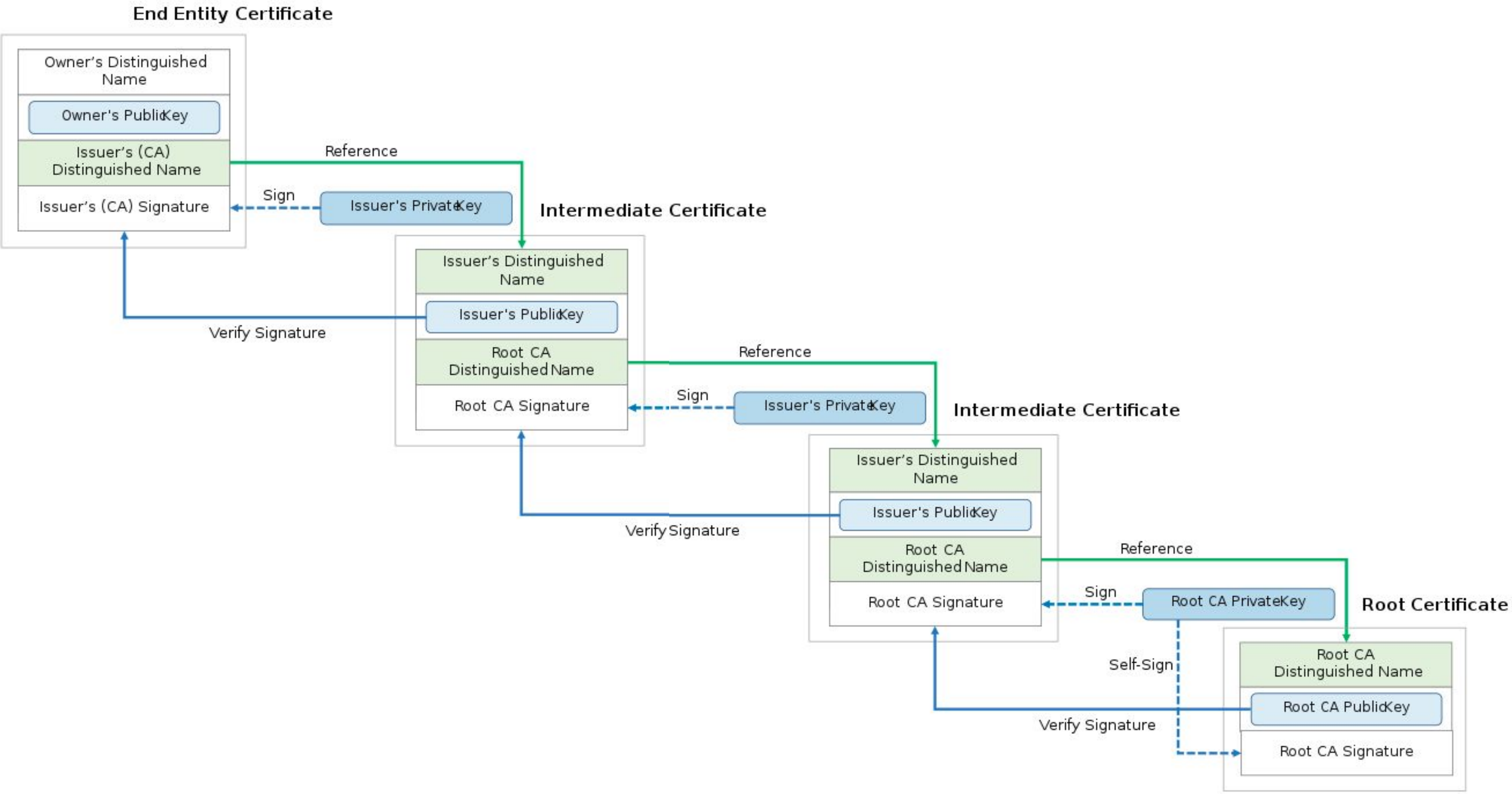
Public Key Infrastructure (PKI)

certificate authority
(trusted)

chain of trust

cryptosystems - in motion - TLS

Chain of Trust



victory!

finally.

wait, what’s all that other stuff?

cryptosystems

at rest

pretty good privacy
PGP

PGP

pretty good privacy

standard for encrypting & sign data.

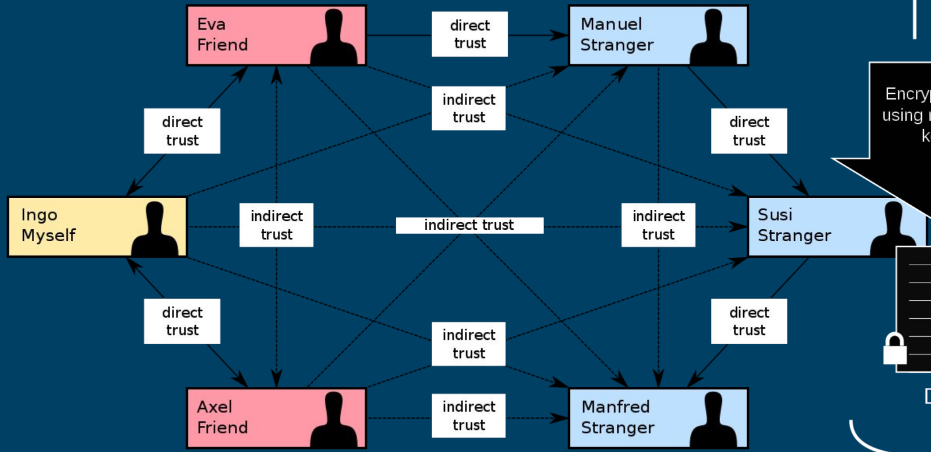
what's in the cryptosystem:

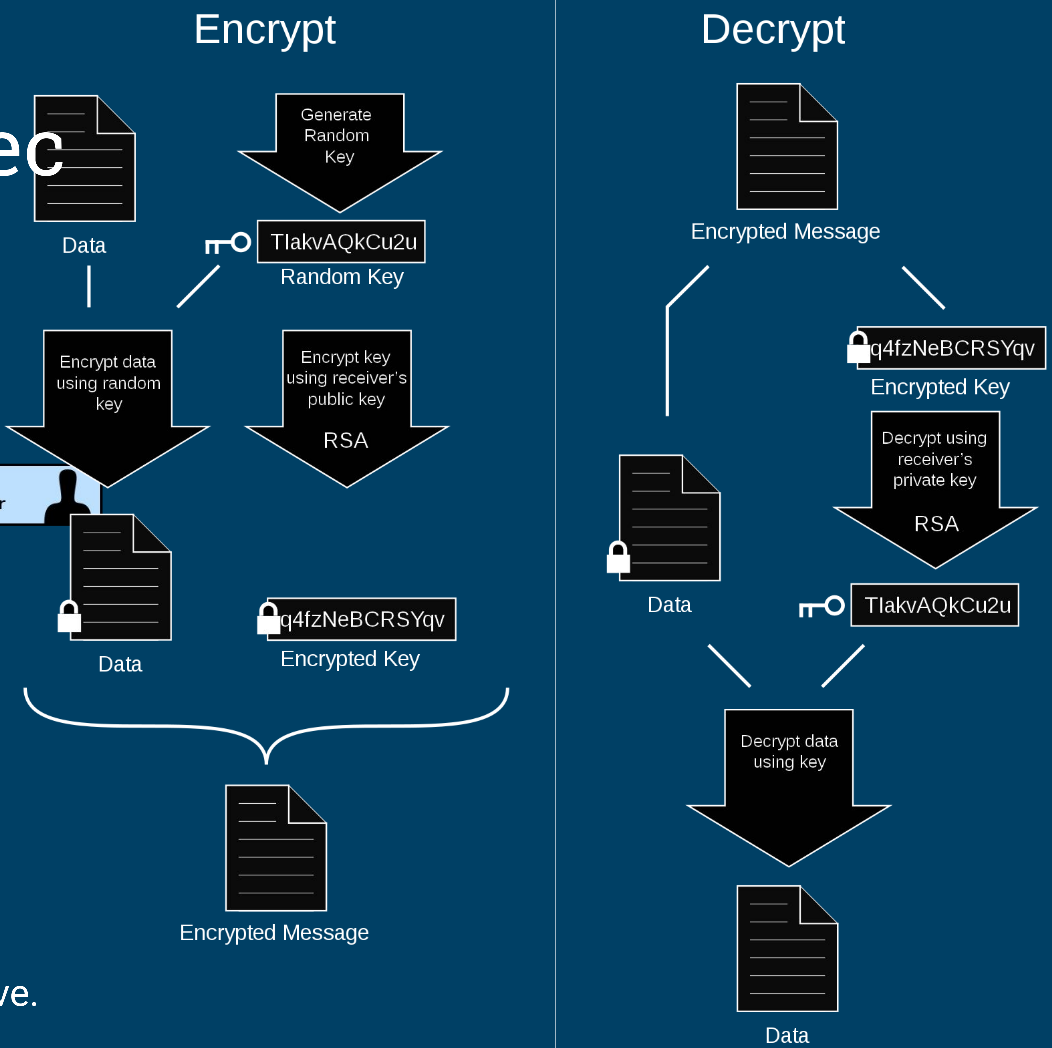
- integrity: hash
- confidentiality: gen key
- authenticity: RSA
- non-repudiation

users sign each other's keys, forming a web of trust.

Cryptosystems - at rest - PGP

Web of Trust, Enc, Dec





Anecdotes:

- Snowden: "What's your public key?"
- to classify as munition?
- alleged child-porn hoarder released;
  FBI couldn't decrypt his PGP-encrypted drive.

Password Storage `pass`

- encrypt/decrypt passwords using PGP keys (`gpg`)
    - encrypt using public key. store.
    - decrypt using privacy key
    - note: a high level explanation; details probably more complex
- can be combined with physical tokens (Lec 5)
- demo!

Cryptosystems

off the record

OTR

OTR

off the record

secure instant messaging between people (E2EE: end-to-end encrypt)

what's in the cryptosystem:

- integrity: SHA-1 HMAC
- confidentiality: AES
- key sharing: DH

properties

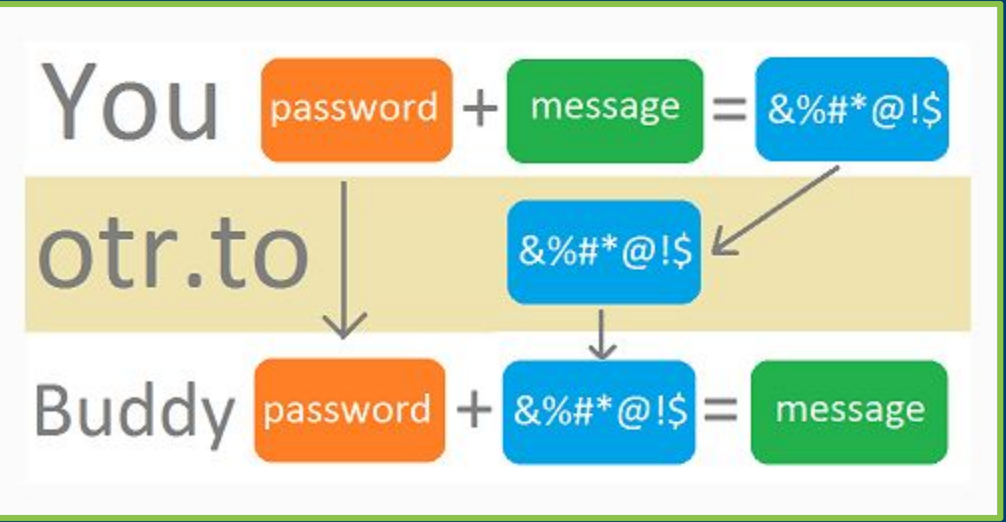
- forward secrecy
- malleable encryption
- deniable authentication.

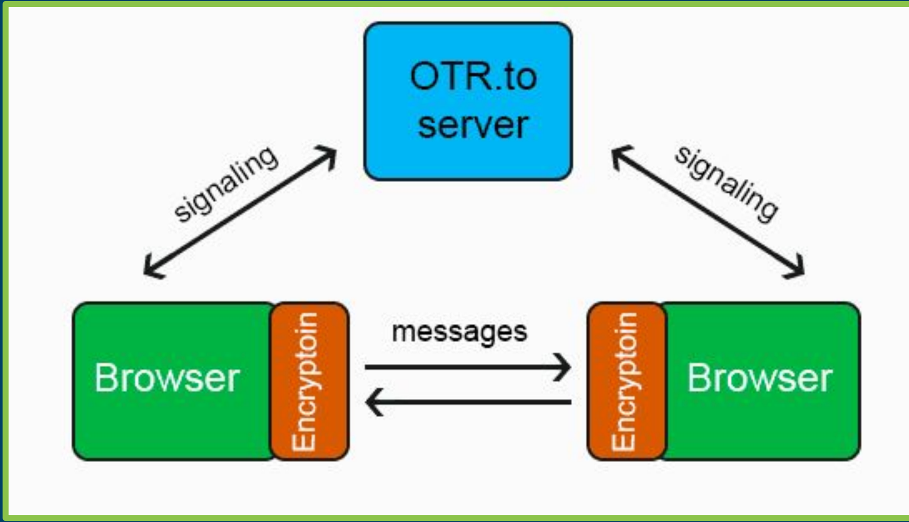
I didn't say that

What wasn't me

Cryptosystems - off the record - OTR

Connections & Sharing Messages





Password Storage

By machines

Storage by Machines

- Passwords are typically stored in a file or database in the computer

- Store passwords in plaintext
  - Not a good idea
  - Requires perfect unbreakable access control (next lecture)
  - Requires trusted sysadmins

- Not unlikely that a password file is stolen
  - https://haveibeenpwned.com/

Storage by Machines

- Use a function f that:
  1. Makes easy to compute $f(p)$ for a password p
     - Even though relatively slow authentication is not necessarily bad
  2. It is hard to compute p from $f(p)$
  3. Hard to find $f(q) = f(p)$ where $p \neq q$

Storage by Machines

- Use a function $f(\cdot)$ that:
  1. Makes easy to compute $f(p)$ for a password p
     - Even though relatively slow authentication is not necessarily bad
  2. It is hard to compute p from $f(p)$
  3. Hard to find $f(q) = f(p)$ where $p \neq q$

- Cryptographic hash functions are enough!
  1. One-way property fulfills 1. and 2.
  2. Collision resistance fulfills property 3.

Storage via Cryptographic Hash Function

- Let the password file (or database) be composed of pairs
  - $\langle \mathrm{uid}_i, \mathrm{h}(\mathrm{pass}_i) \rangle$ where
    - uid_i is an identifier
    - pass_i is the corresponding password
    - $\mathrm{h}(\cdot)$ is a cryptographic hash function

- Authentication protocol in a `System` with a password file `Pwd` = {<uid_1, h(p_1)>, <uid_2, h(p_2)>, ...}:

```
1.  Alice -> System: uid, pass
2.  System: if <uid, h(pass)> ∈ Pwd
                then Deem Alice authenticated
```

Storage via Cryptographic Hash Function

Assumption: the communication channel between Alice and the system is be secure:

- Keyboard
  - Trust driver and hardware
- Network
  - TLS (coming in a few slides)

latabase) be composed of pairs

nding password

ic hash function

- Authenticatio col in a `System` with a password fil wd = $\{\langle uid_1, h(p_1)\rangle, \langle uid_2, h(p_2)\rangle, \ldots\}$:

  1. `Alice -> System: uid, pass`
  2. `System:` **`if`** `<uid, h(pass)> ∈ Pwd`

     `then` Deem `Alice` authenticated

Offline Attacks

- Attackers may build a *dictionary* containing hashes of common passwords
  - Top password rankings
  - Password recipes
  - $\mathrm{Dict} = \{<p_1, h(p_1)>, <p_2, h(p_2)>, <p_3, h(p_3)>, \ldots\}$
- Approaches
  - **Build Dict only once**, attack many systems (*rainbow tables*)
  - **Build Dict on demand** for specific systems
- A dictionary attack tries to find the hashes in a dictionary (Dict) that also appear in a stolen password file (`PwD`)

$$\mathrm{FoundPwd} = \{ <uid, p> \mid <uid, h(p)> \in \mathtt{PwD} \wedge <p, h(p)> \in \mathrm{Dict} \}$$

Adding Salt

- A protection against offline attacks is making computing Dict unfeasible

- Add a nonce n_i , called **salt**, to each pair in `Pwd`
  - SaltyPwd = $\{<uid_i, n_i, h(p_i \| n_i)>\}$ for $i = 1,2,3, \ldots$
  - Salt is not secret

- Computing SaltyDict is harder than computing Dict
  - Given nounces of size b bits SaltyDict is $j=2^b$ times larger than Dict

Can the attacker reduce the size of SaltyDict without losing accuracy? [Mentimeter]

$$\text{SaltyDict} = \{<p_1, h(p_1\|n_1)>, <p_1, h(p_1\|n_2)>, \ldots, <p_1, h(p_1\|n_j)>, <p_2, h(p_2\|n_1)>, <p_2, h(p_2\|n_2)> , \ldots\}$$

Limited Offline Attacks

- Since salt is stored in plain in SaltyPwd, attacker can reduce the size of SaltyDict by focusing only on the nounces appearing in SaltyPwD

- If SaltyPwd contains N entries, SaltyDict will have N|Dict|
  - As opposed to the $2^b|Dict|$ that we mentioned earlier

- Possible Solution: Keep the salt secret
  - SecretSaltyPwd = {<uid, $h(p_i \,||\, n_i)$>} for i = 1, 2, 3, ...

Authentication

```
1.   Alice -> System: uid, password
2.   System: if (∃ n : <uid, h(p || n)> ∈ SecretSaltyPwd)
                then Deem Alice authenticated
```

Limited Offline Attacks

- Since salt is stored in plain in SaltyPwd, attacker can reduce the size of SaltyDict by focusing only on the nounces appearing in SaltyPwD

- If SaltyPwd contains N entries, SaltyDict will have N|Dict|

  e mentioned earlier

- Computationally expensive. We need to search for all possible nounces
- Solution? Salt and pepper. See Chapter 5 of Fred Schneider's book.

lt secret

)>} for i = 1, 2, 3, ...

Authentication

```
1.   Alice -> Syst    uid, password
2.   System: if (∃ n : <uid, h(p || n)>  ∈  SecretSaltyPwd)
             then Deem Alice authenticated
```

Linux Password Storage

- `/etc/passwd`
  - Contains
    - Username, and user related information
- `/etc/shadow`
  - Hashing algorithm
  - Salt (not secret --- not well seasoned 😏)
  - The hash of the password concatenated with the salt

Linux Password Storage

- `/etc/passwd`
  - Contains
    - Username, and user related information

- `/etc/shadow`
  - Hashing algorithm
  - Salt (not secret --- not well seasoned 😏)
  - The hash of the password concatenated with the salt

1. U -> S: uid, $pass_U$
2. S: **if** <uid, salt||hpass,_> ∈ /etc/shadow
   then if h(salt||$pass_U$) = hpass
   then Deem U authenticated

Summary

Summary

"Securely"

CIA: Confidentiality, Integrity, Availability? ...

- Confidentiality:
  only the intended recipient of a message should be able to read it.

- Integrity:
  An adversary cannot (undetectedly) tamper with a message.

- Authenticity [new!]:
  An adversary cannot (undetectedly) forge a message from either party

Summary

The Devil is in the Details

what **cryptographic engineers** do:

- domain knowledge
- design, implement, test, validate cryptographic systems
- cryptanalysis

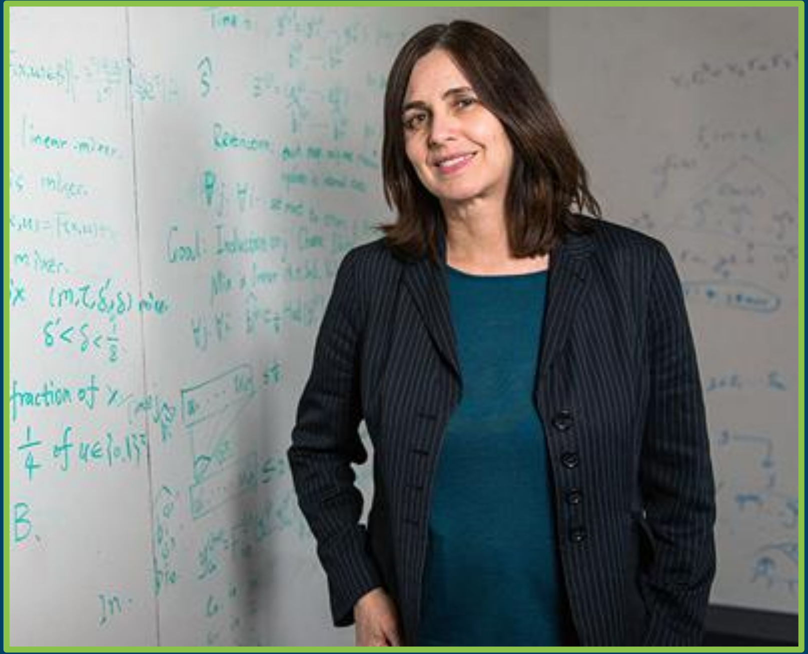
security vs. performance: crypto **breaks.**

don't roll your own crypto! if you type

AES: doing it wrong

DES: doing it extra wrong

MD5, SHA: maybe wrong?



Shafi Goldwasser
Professor, Cryptographer, Turing Award winner