Section 5
Public Key Crypto Topics: RSA, Cryptanalysis with CBC-MAC

Administrivia

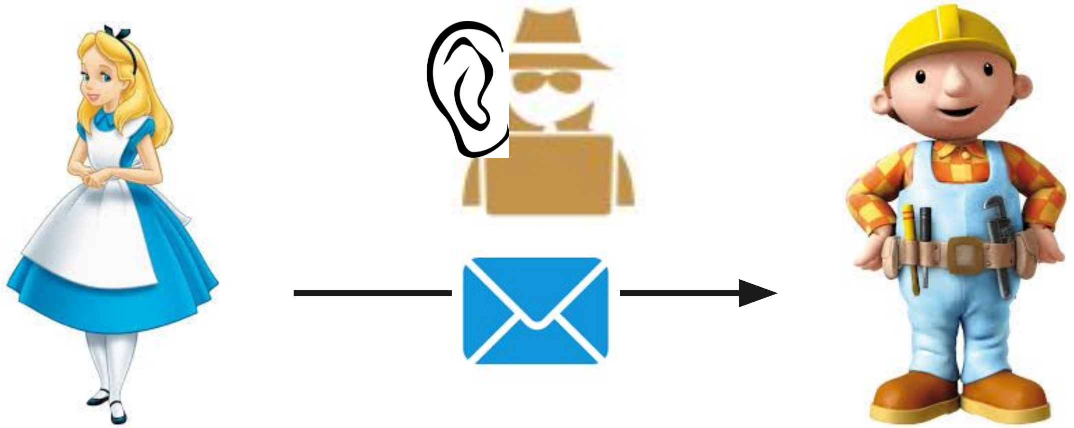
- Homework 2 due next Friday (11/5)
  - Individual assignment
  - Hands-on cryptography
- Final Project checkpoint #1 due next next Friday (11/12)
  - Group members' names and UWNetIDs
  - Presentation topic

RSA:
Key generation, encryption, and decryption

Public Key Cryptography Review

Alice wants to send Bob an encrypted message

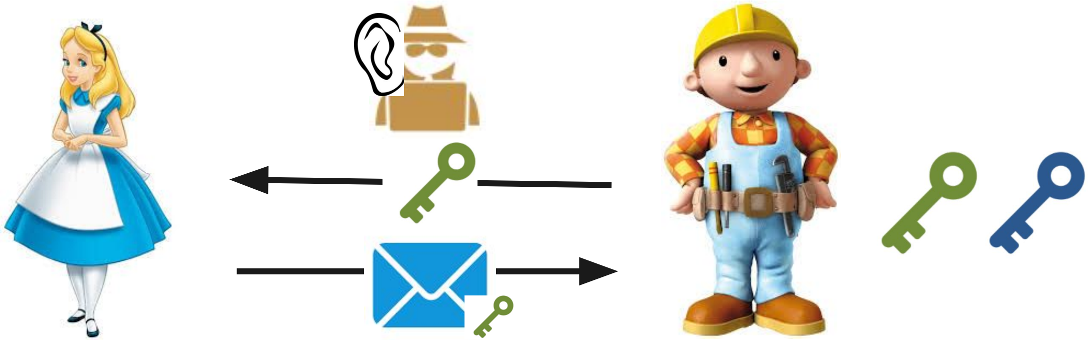
- Goal: Confidentiality
- Problem: Eve can intercept key



Public Key Cryptography Review

Solution: public key cryptography (aka **asymmetric** cryptography)

- Public-private keypair
- Alice encrypts using Bob's public key
- Bob decrypts using Bob's private key



RSA Cryptosystem Review

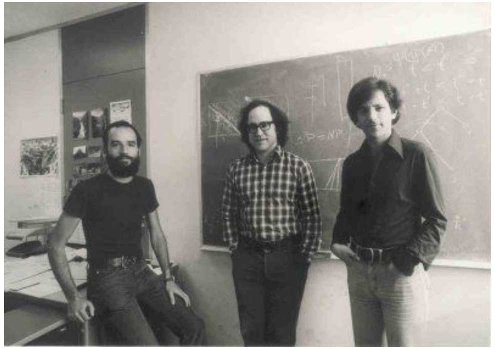
Key generation:

- Generate large primes p, q
- Compute N=pq and φ(N)=(p-1)(q-1)
- Choose **e** coprime to φ(N)
    - Typically **e=3** or **$e=2^{16}+1=65537$**
- Find (unique) **d** such that ed ≡ 1 (mod φ(N))
    - (equivalent to solving 1 = (e·d) mod φ(n))

Public key = (e, N); Private key = (d, N)

Encryption of m: $c = m^e \bmod N$

Decryption of c: $c^d \bmod N = (m^e \bmod N)^d \bmod N = m^1 \bmod N = m$



Adi **S**hamir, Ron **R**ivest, Len **A**dleman
[Photo from Dan Wright]

RSA Practice

Public key: N = 33, e = 7

Step 1: Find φ(N)
Step 2: Find the decryption key, d

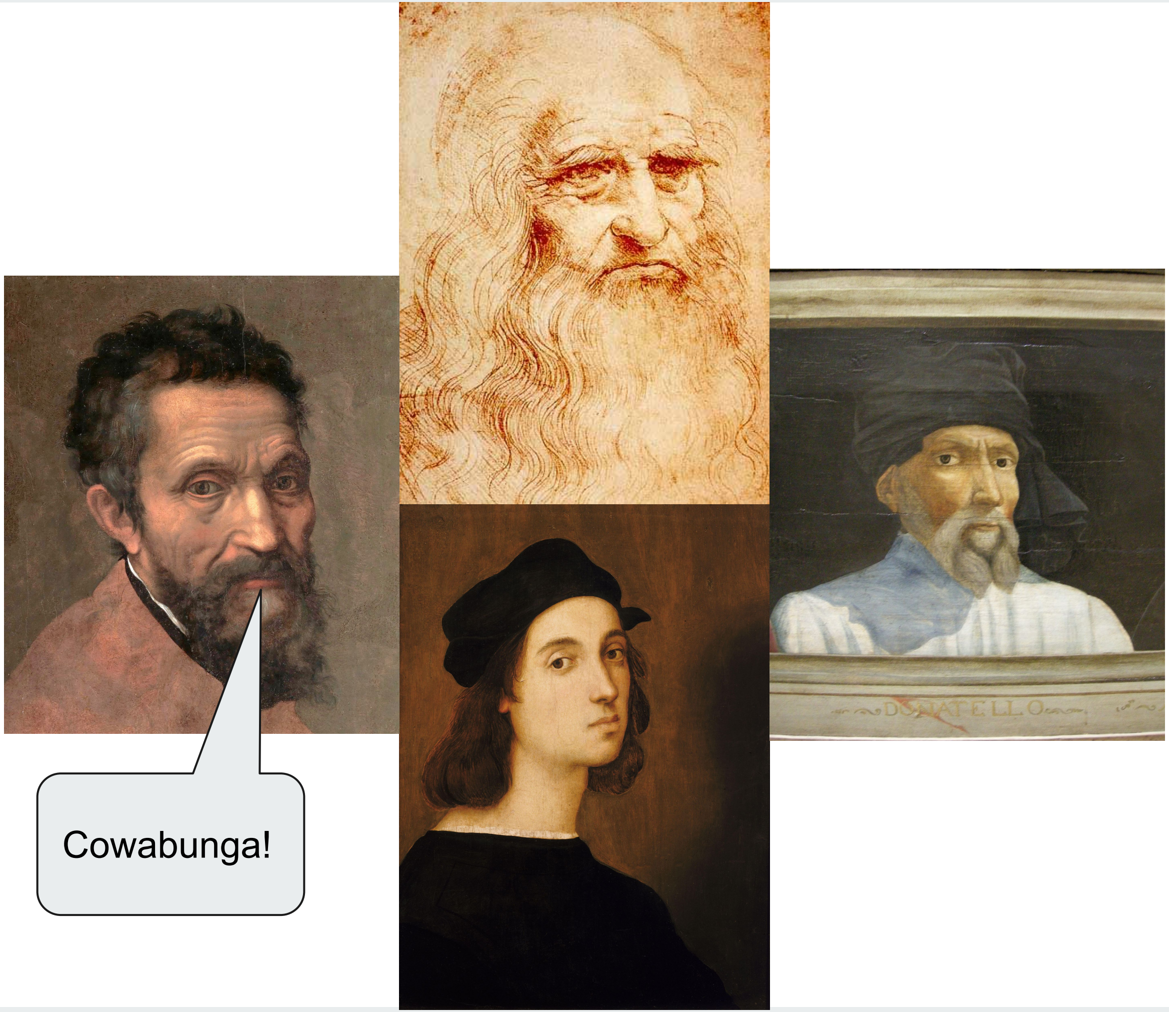
- $ed \equiv 1 \pmod{\varphi(N)}$

Step 3: Decrypt the cryptogram

- $c^d \bmod N = m$
- 'A' = 1, 'B' = 2, ...

Cryptograms:

```
12 14 27 20 1 6 16 27
6 1 25 2 1 14 12
7 15 9 2 14 12 1 20 28 14 12 27
16 27 20 1 26 14 12 12 27
```


Cowabunga!


TEENAGE MUTANT NINJA
TURTLES

RSA Strength

"RSA problem": decrypt only using the public key

- Factoring N is hard
- No known efficient algorithm
- Trapdoor function: easy to go forward, hard to go back

RSA Factoring Challenge (1991-2007)

- Cash prizes for factoring large N values (up to $200,000 (!))
- Only the smallest 23 of 54 factored so far...

Shor's Algorithm

- Quantum computer algorithm to factor integers
- Largest number factored so far: 21 😎

RSA-2048:

```
251959084756578934940271832400
483985714292821262040320277771
378360436620207075955562640185
258807844069182906412495150821
892985591491761845028084891200
728449926873928072877767359714
183472702618963750149718246911
650776133798590957000973304597
488084284017974291006424586918
171951187461215151726546322822
168699875491824224336372590851
418654620435767984233871847744
479207399342365848238242811981
638150106748104516603773060562
016196762561338441436038339044
149526344321901146575444541784
240209246165157233507787077498
171257724679629263863563732899
121548314381678998850404453640
235273819513786365643912120103
97122822120720357
```

RSA Today

- Still used today but mostly in legacy applications
  - SSH keys, TLS, etc.
  - But not preferred...
- Need big keys for RSA
  - At least 2048 bits
- Bigger keys ⇒ slower computation
- Modern encryption schemes exist, such as Elliptic-Curve Cryptography (ECC)
  Explanation and Diagram for ECC

Demonstration: Finding vulnerabilities in CBC-MAC with cryptanalysis

Is encryption (confidentiality) enough?

Scenario: David wants to send out an email about exam times - and a hacker has learned the encryption key

ok

dkohlbre@cs

“Final!!!
KNE 110
Monday
2:30PM”

AES 128-bit key,
CBC mode

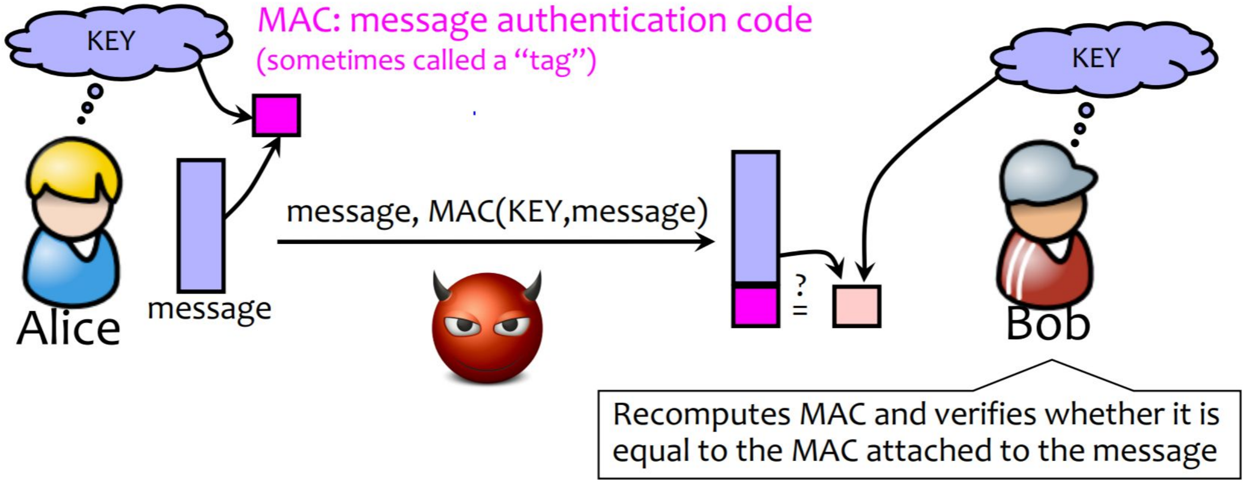
In this case, an adversary doesn’t gain anything important by learning the content of this message.

Is encryption (confidentiality) enough?

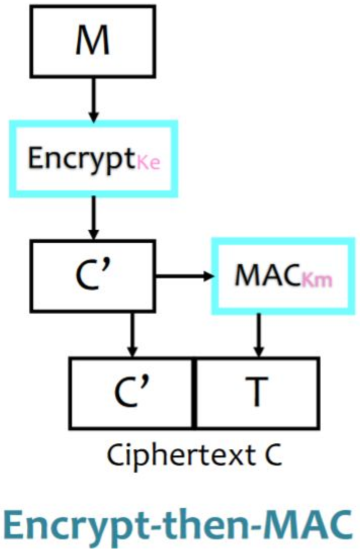
But, the attacker could tamper with the message during transmission, and the recipient would not know - so we need to ensure **integrity**

MAC (Message Authentication Code)

Provides integrity and authentication: only someone who knows the KEY can compute correct MAC for a given message.



When do we MAC?



The good:

Encrypt-then-MAC

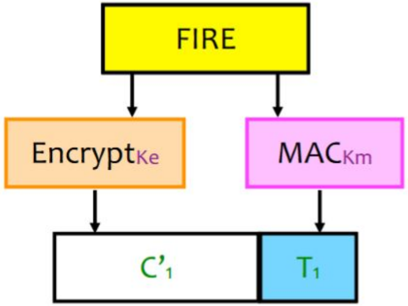
MAC-then-encrypt

Not as good as Encrypt-then-MAC

The bad (& ugly):

Encrypt-and-MAC

MAC is deterministic! Same plaintext → same MAC

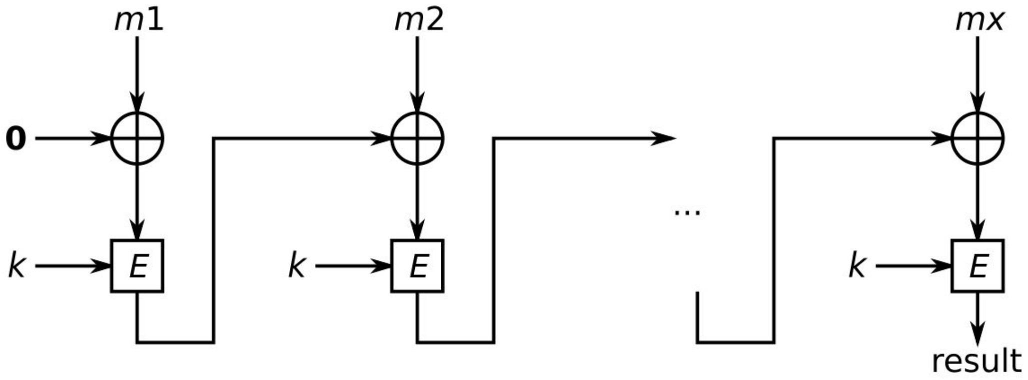


How do we create a MAC?

CBC-MAC: Encrypt the message in CBC mode, use the last block as the MAC

Initialization vector is 0

k = secret key



Last block of ciphertext used as MAC

*CBC-MAC is not the only MAC algorithm - today most use HMAC; we'll show why next

Is CBC-MAC vulnerable?

- How could we find out?
  - Cryptanalysis: using mathematical analysis to rigorously reason about a cryptographic system

- Let’s use cryptanalysis to find a collision
  - two different inputs leading to the same MAC tag
  - (violating collision resistance)

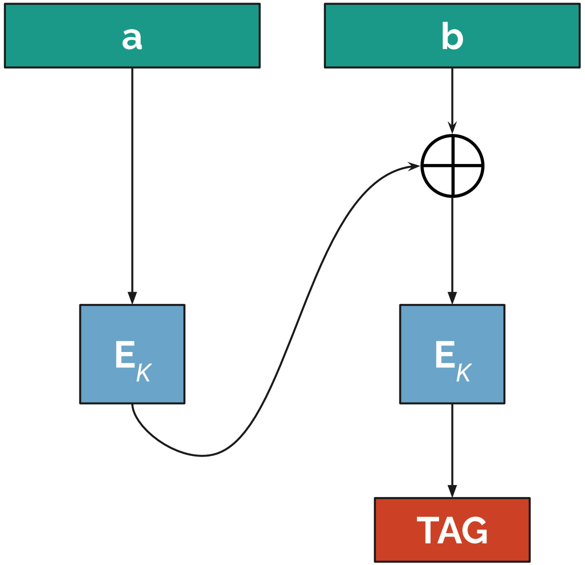
Exercise: CBC-MAC collision vulnerability

Suppose *a* and *b* are both one block long, and suppose the sender MACs *a*, *b*, and *a* || *b* with CBC-MAC.

An attacker who intercepts the MAC tags for these messages can now forge the MAC for the message

$$b \,||\, (M_K(b) \oplus M_K(a) \oplus b)$$

which the sender never sent. The forged tag for this message is equal to $M_K(a \,||\, b)$, the tag for $a \,||\, b$. Justify mathematically why this is true.



a || b: a and b concatenated
M_K(a): MAC for message a
E_K(a): ciphertext for message a

(Ferguson, Schneier, & Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing 2010. Exercise 6.3 p. 97)

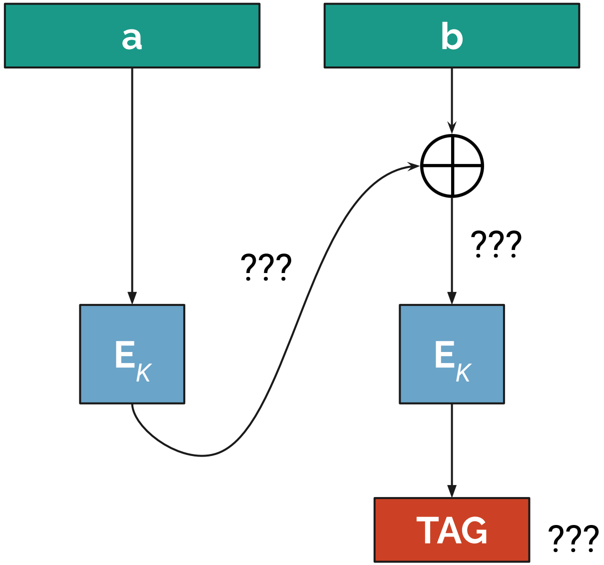
Exercise: CBC-MAC collision vulnerability

Prove:

$$\boldsymbol{M_K}(b \,||\, (M_K(b) \oplus M_K(a) \oplus b)) = \boldsymbol{M_K}(a \,||\, b)$$

Step 1: Figure out what M_K(a), M_K(b), and M_K(a || b) in terms of the encryption key.

Annotate sketch with the sender's messages and MACs.



(Ferguson, Schneier, & Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing 2010. Exercise 6.3 p. 97)

Exercise: CBC-MAC collision vulnerability

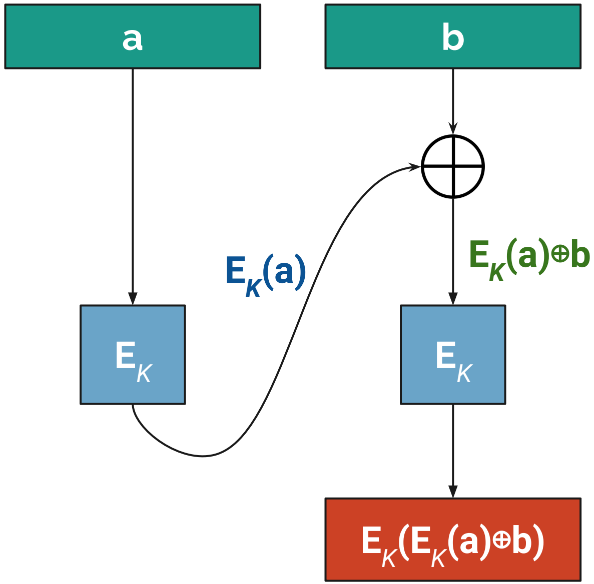
Prove:

$$\mathbf{M_K}(b \,||\, (M_K(b) \oplus M_K(a) \oplus b)) = \mathbf{M_K}(a \,||\, b)$$

$M_K(a) = E_K(a)$

$M_K(b) = E_K(b)$ (not shown)

$M_K(a \,||\, b) = E_K(E_K(a) \oplus b)$



(Ferguson, Schneier, & Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing 2010. Exercise 6.3 p. 97)

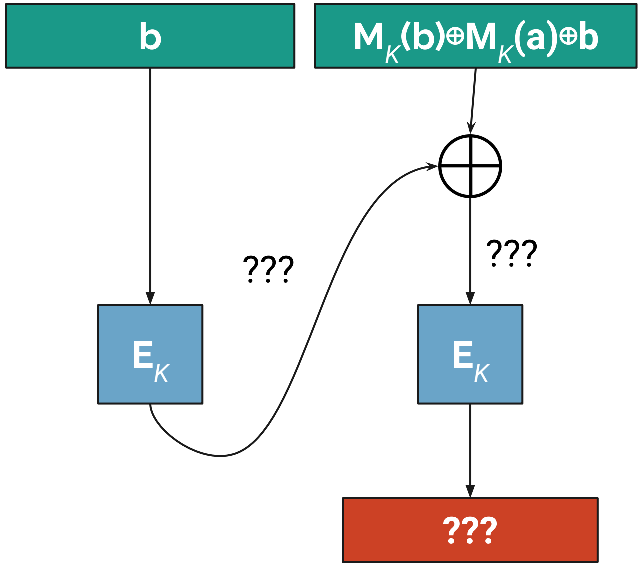
Exercise: CBC-MAC collision vulnerability

Prove:

$$\mathbf{M_K}(b \,||\, (M_K(b) \oplus M_K(a) \oplus b)) = \mathbf{M_K}(a \,||\, b)$$

Step 2: Figure out $M_K(b \,||\, (M_K(b) \oplus M_K(a) \oplus b))$.

For the MAC of the attacker's message $b \,||\, (M_K(b) \oplus M_K(a) \oplus b)$, what are the values of the ???'s?



(Ferguson, Schneier, & Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing 2010. Exercise 6.3 p. 97)

Exercise: CBC-MAC collision vulnerability

Prove:

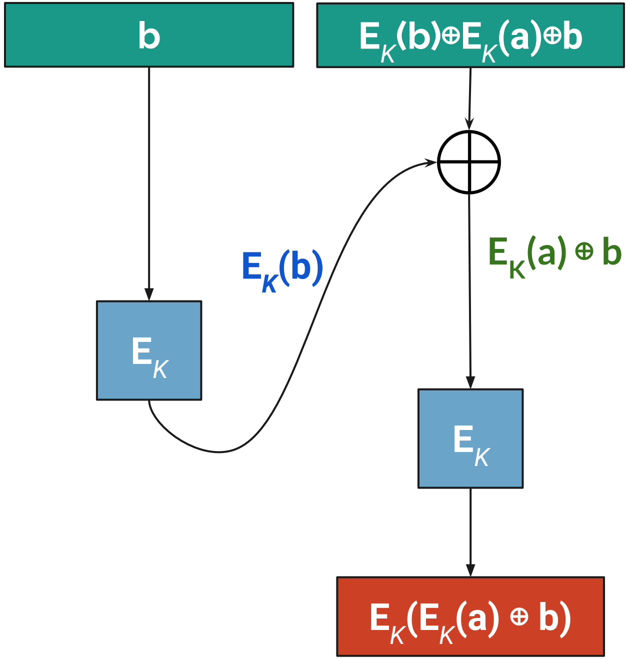
$$M_K(b \,||\, (M_K(b) \oplus M_K(a) \oplus b)) = M_K(a \,||\, b)$$

$M_K(b \,||\, (M_K(b) \oplus M_K(a) \oplus b))$

$= M_K(b \,||\, (E_K(b) \oplus E_K(a) \oplus b))$

$= E_K(E_K(b) \oplus E_K(b) \oplus E_K(a) \oplus b)$ ← These terms cancel out

$= E_K(E_K(a) \oplus b)$ ← *This is the same as $M_K(a \,||\, b)$!*

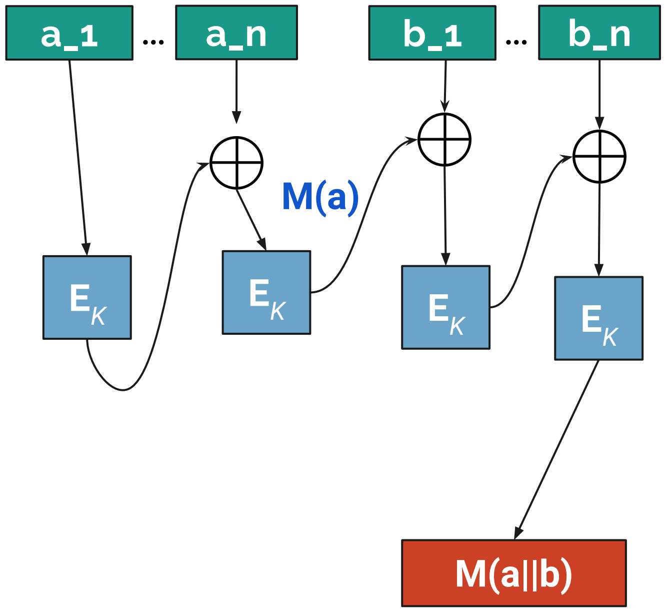


So what?

- We can prove, just using the specification of CBC-MAC, that the messages $b \,||\, (M(b) \oplus M(a) \oplus b)$ and $a \,||\, b$ share the same tag. This approach is a common method used in cryptanalysis.
- We broke the *theoretical* guarantee that no two different messages will never share a tag.
- If you were to use CBC-MAC in a protocol, it provides information about specific weaknesses and how not to use it.

Generalized

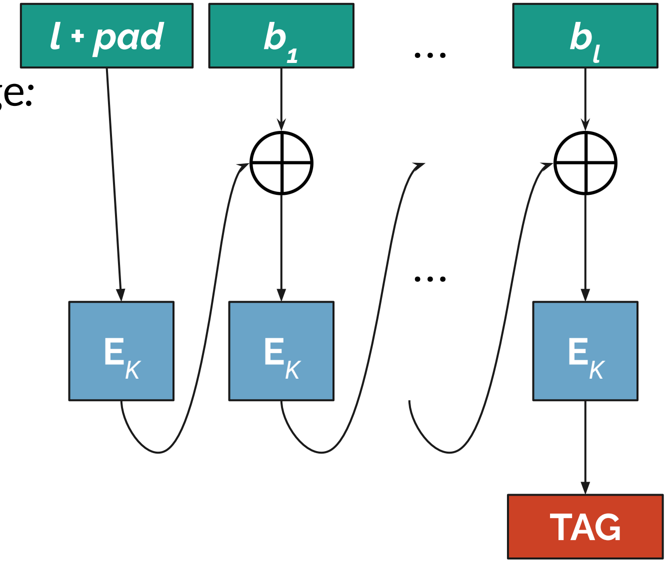
- For any length a, b: M(a) ⊕ b, a || b have same tag
- M(a || b) = M(M(a) ⊕ b)



Safer CBC-MAC for variable length messages

For a message *m* of length *l*:

1. Construct *s* by prepending the length of *m* to the message: *s = concat(l, m)*
2. Pad *s* until the length is a multiple of the block size
3. Apply CBC-MAC to the padded string *s*.
4. Output the last ciphertext block, or a part of it. Don't output intermediates.



- Now sM(a || b) != sM(sM(a) ⊕ b)
- Because sM(a||b)=M(concat(l, a || b))

Or....

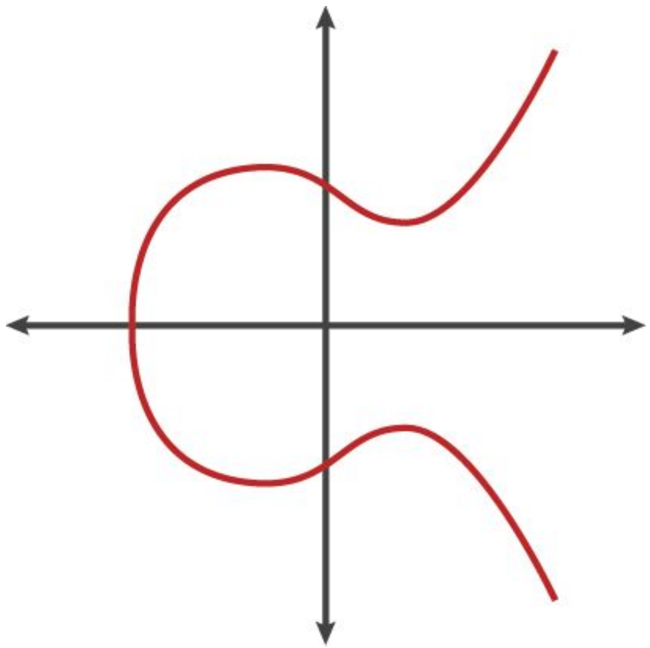
- Or encrypt output with another block cipher under a different key (CMAC). Or use HMAC, UMAC, GMAC.

Thanks for coming to section!

Elliptic-Curve Cryptography (ECC)

$y^2 = x^3 + ax + b$

- First suggested independently by Neal Koblitz (UW Math faculty!) and Victor S. Miller in 1985
- Widespread adoption started in the last 2 decades



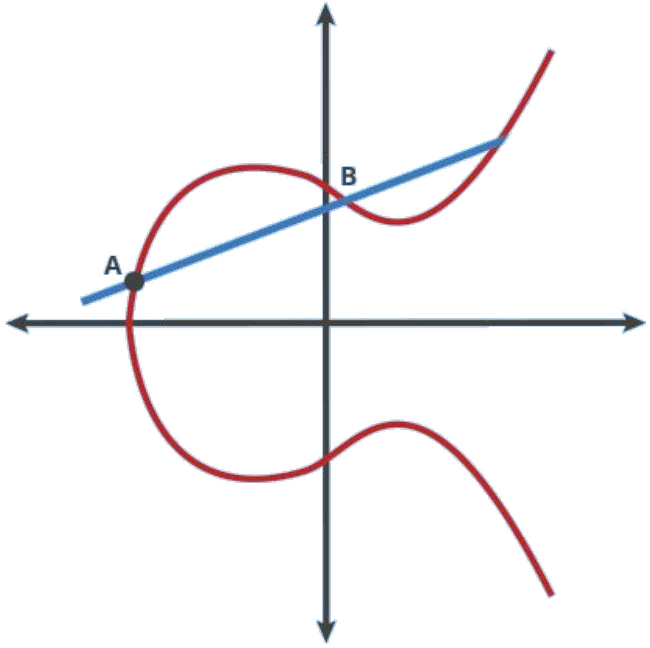
[visuals from Cloudflare]

Elliptic-Curve Cryptography (ECC)

Special operation: ◦ (“dot”)

- A ◦ B = C, A ◦ C = D, …
- nA = A ◦ … ◦ A (n times)
- x(yA) = y(xA) = xyA
- Given point P, hard to find n s.t. nA = P
- Pattern behaves “randomly”

Private key: n (integer)
Public key: P (point on curve, P = nG)
Public knowledge: G (generator point) and curve parameters



[visuals from Cloudflare]

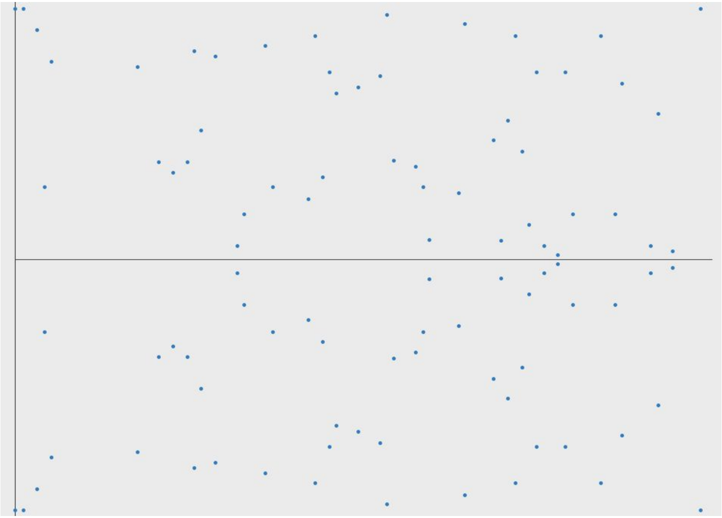
ECC In Practice

Wrap the graph about x and y axes

- Achieves the same effect as modulo, in RSA
- Want prime numbers as the bounds
- Elliptic Curve Discrete Logarithm Problem™

“Safe” Curves?

- NIST recommendations are “fast”, but suspicious
- djb et al. show their work for recommendations
- More: https://safecurves.cr.yp.to/



[visuals from Cloudflare]

ECC vs RSA

Pros:

- Same strength using smaller keys
- Smaller keys ⇒ faster computation
- ECDLP harder(?) than DLP

Cons:

- Hard to understand
- Hard to implement correctly
- Suspicious implementations (NSA 🤔)

Security Strength	Symmetric Key Algorithms	FFC (DSA, DH, MQV)	IFC* (RSA)	ECC* (ECDSA, EdDSA, DH, MQV)
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

[table from NIST (SP 800-57 PART 1 REV. 5)]

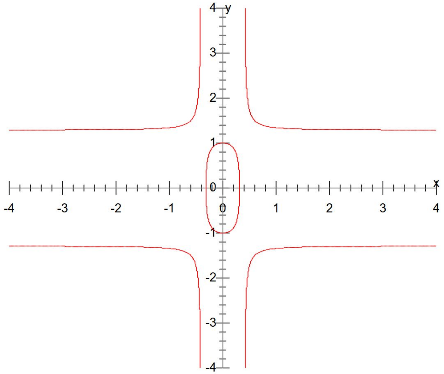
Ultimately: ECC can achieve the same security with smaller keys and faster operations.

ECC In The Wild

ECC can be substituted for $(\mathbb{Z}_p)^\times$ in DL-based protocols:

- Elliptic Curve **Diffie-Hellman**
- Elliptic Curve Integrated **Encryption** Scheme
- Elliptic Curve Digital **Signature** Algorithm
- Edwards-curve* Digital **Signature** Algorithm

Most digital certificates use ECDSA (e.g. P-256) or EdDSA (e.g. ed25519)



*Twisted Edwards curve [Wikipedia]

Certificates in Practice
&
Certificate Authority
(CA)

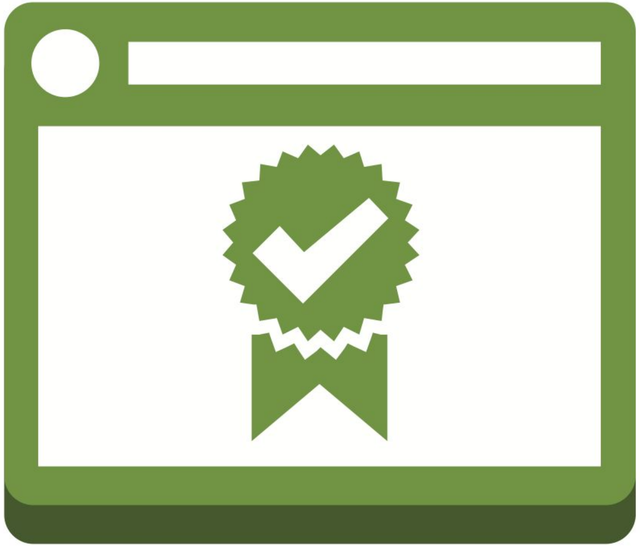
What are certificates

- A security certificate is a small data file used to establish the identity, authenticity and reliability of a website.

  Think of it as a passport!

- TLS/SSL: Encryption and authentication for connections

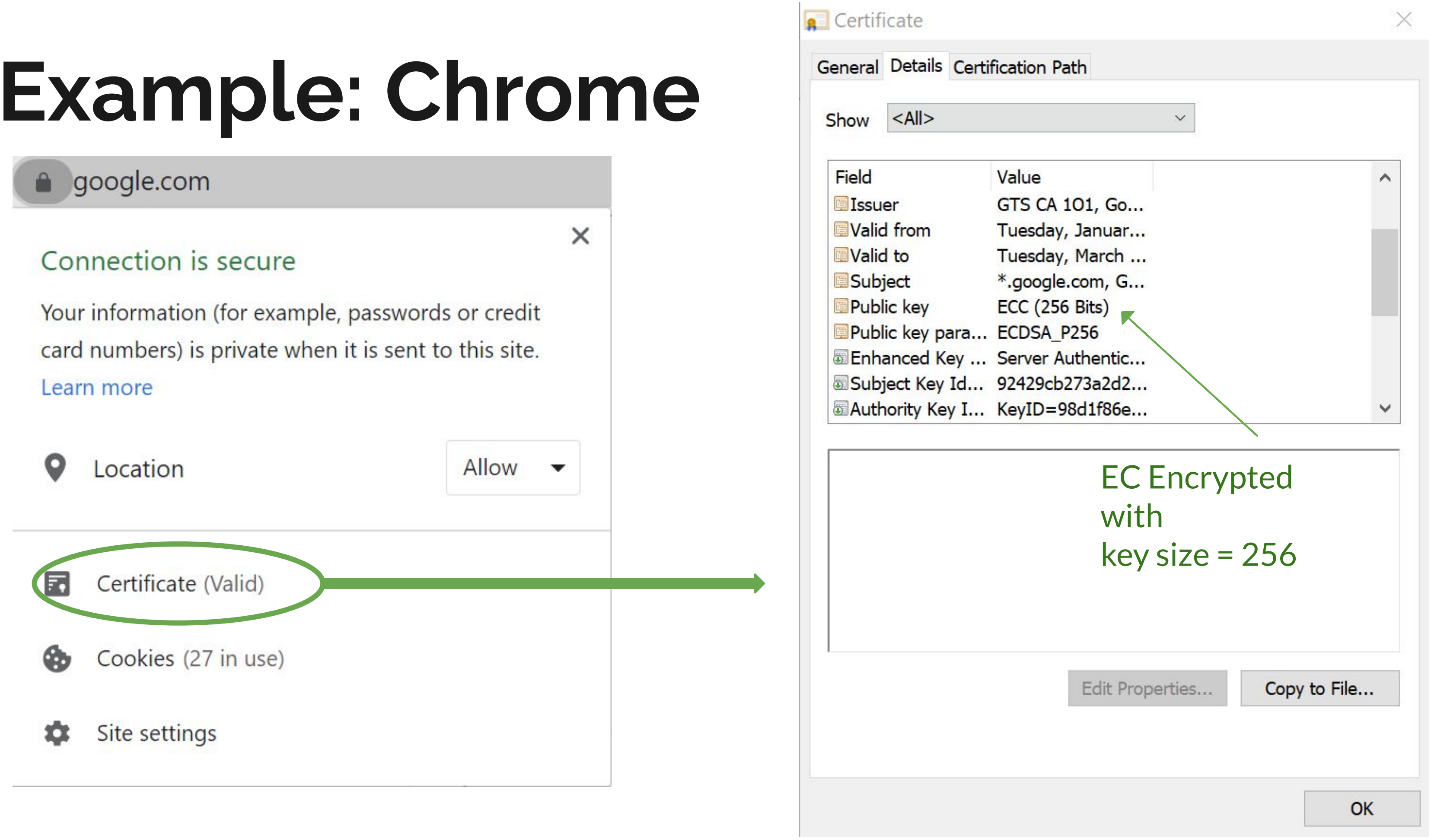
  Note that certificates are not dependent on protocols.



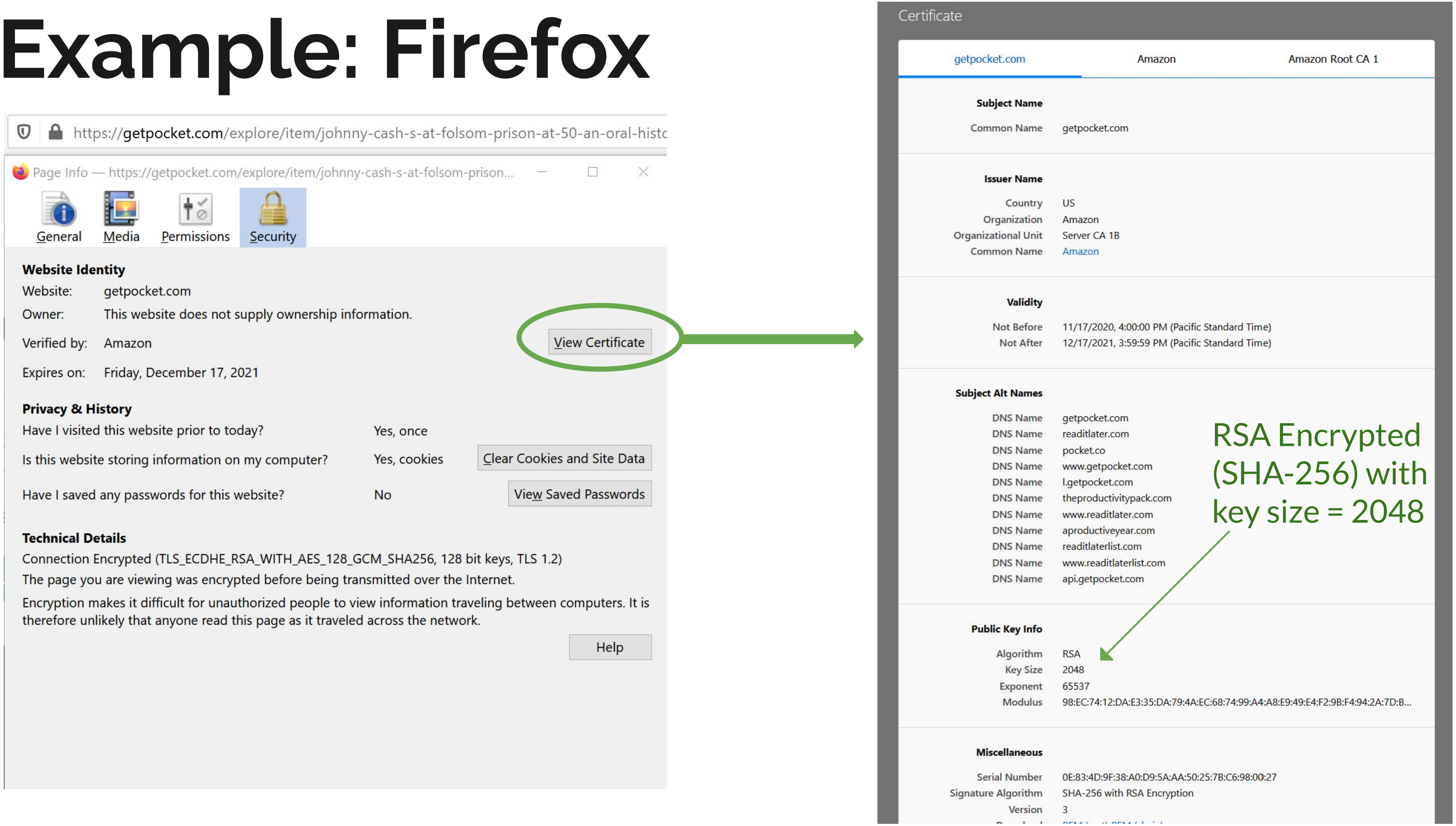
Information on a certificate

- An X.509 certificate a standard format for public key certificates.
  - Different versions, most common: X.509 v3
  - Not all certificates require public trust
- Includes:
  - public key
  - digital signature
  - Issuing CA
  - Additional information about the certificate

Example: Chrome

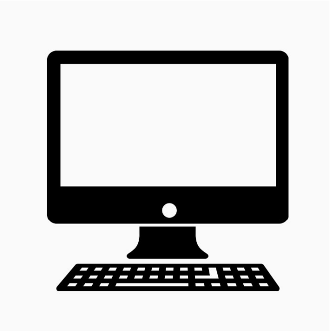


Example: Firefox



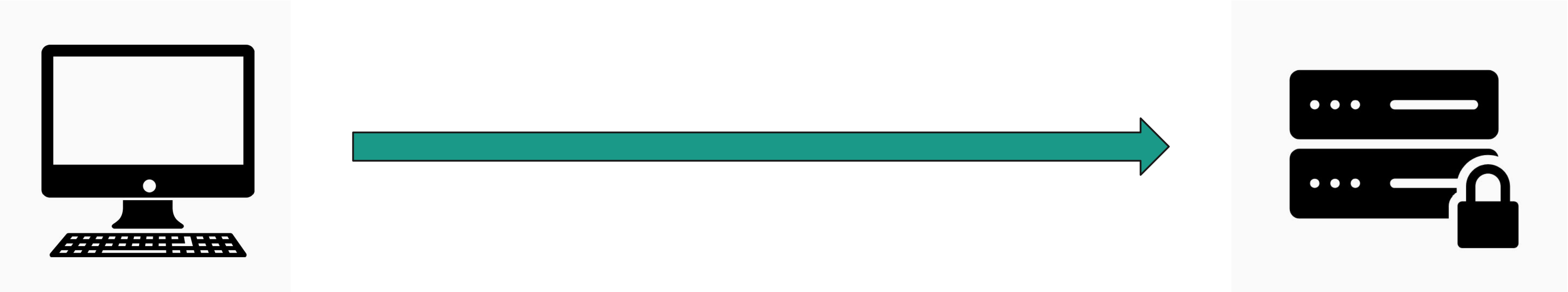
The Handshake

The Handshake



- Server hello
- Client certificate request

The Handshake

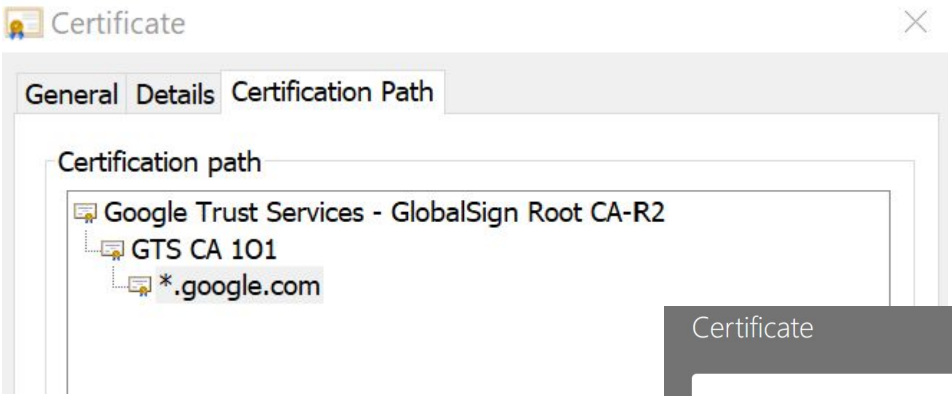


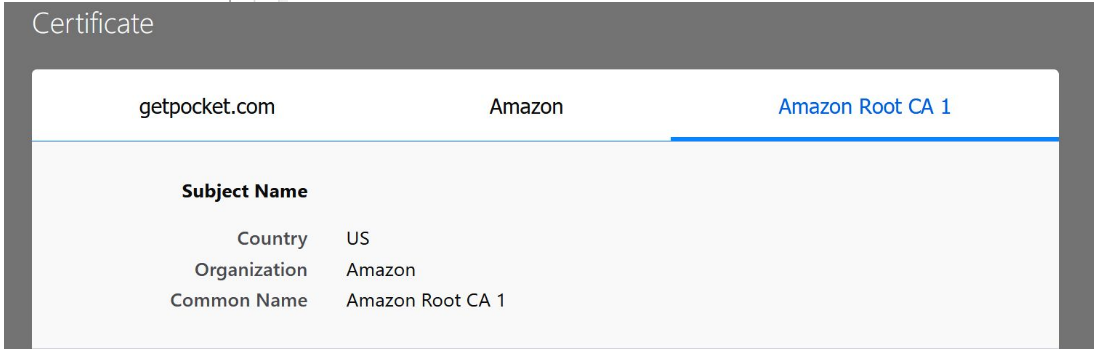
- Client certificate
- Client sends key info (encrypted with server’s public key)
- Certificate verify (with digital signature)
- Finished message (encrypted with symmetric key)

The Handshake

Finished message (encrypted with symmetric key)

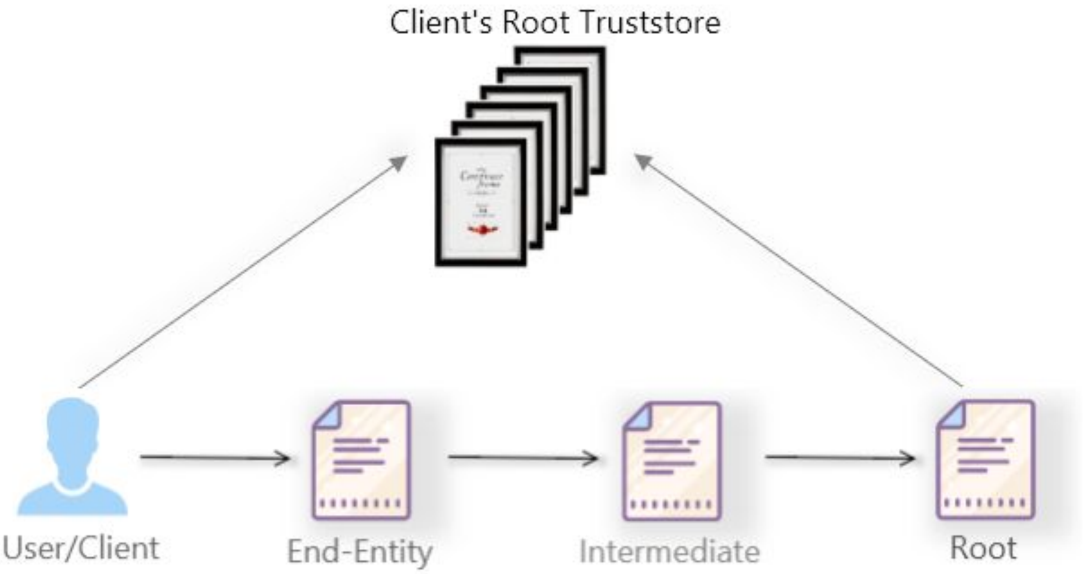
Chain of Trust



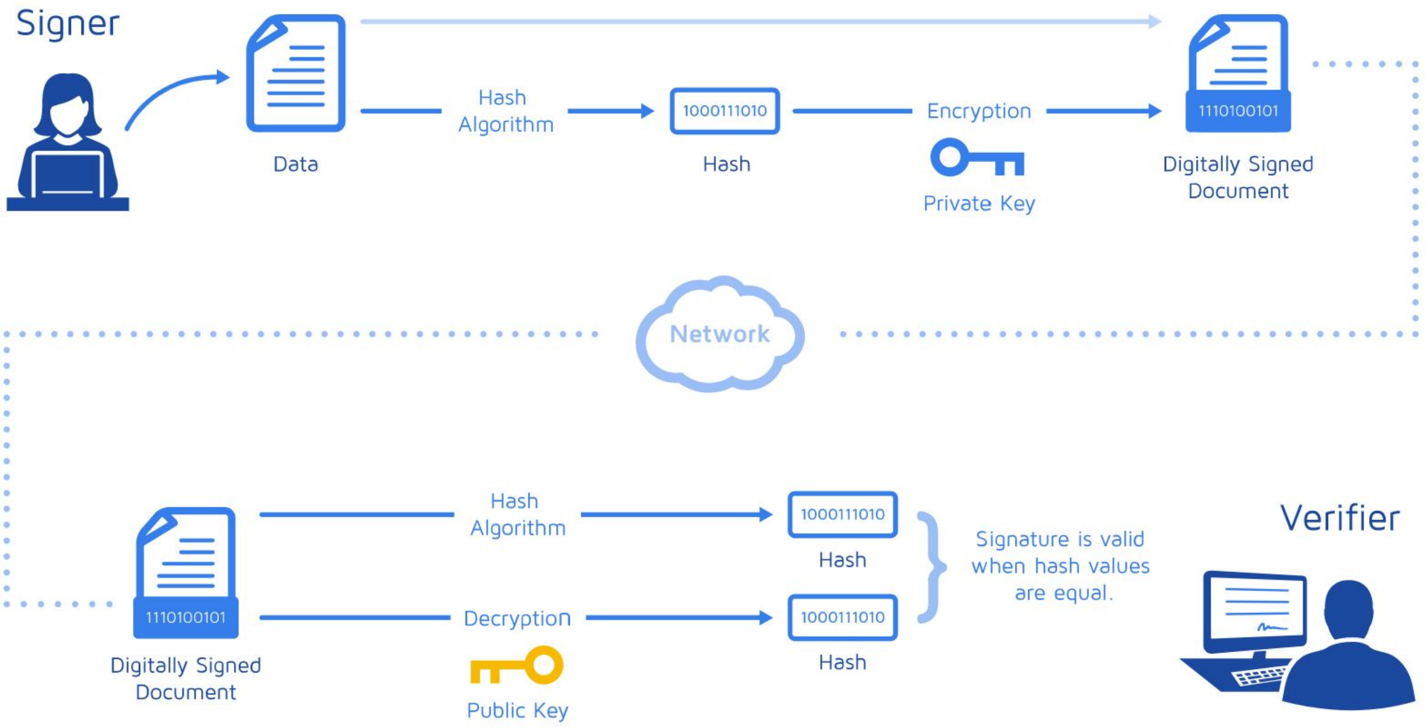


Certificate Authority (CA)

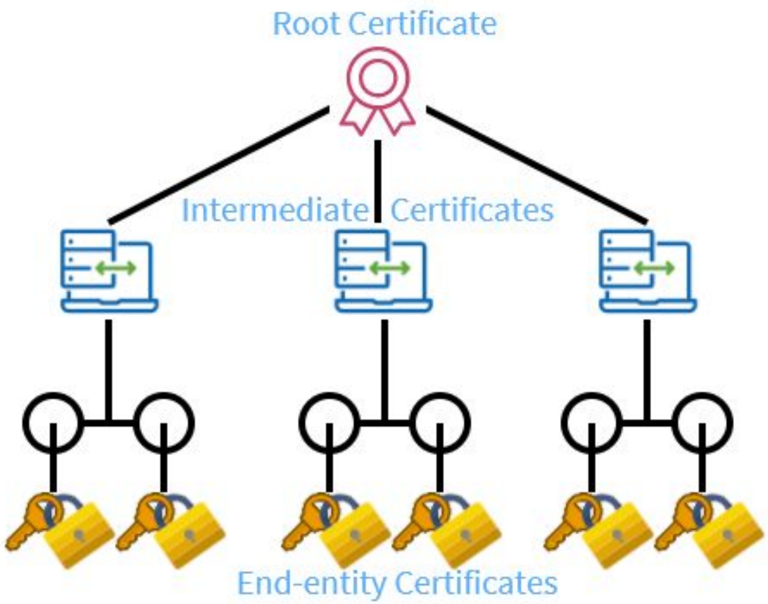
A company or organization that acts to validate the identities of entities and bind them to cryptographic keys through the issuance of digital certificates.



Digital Signatures & Root Certificates



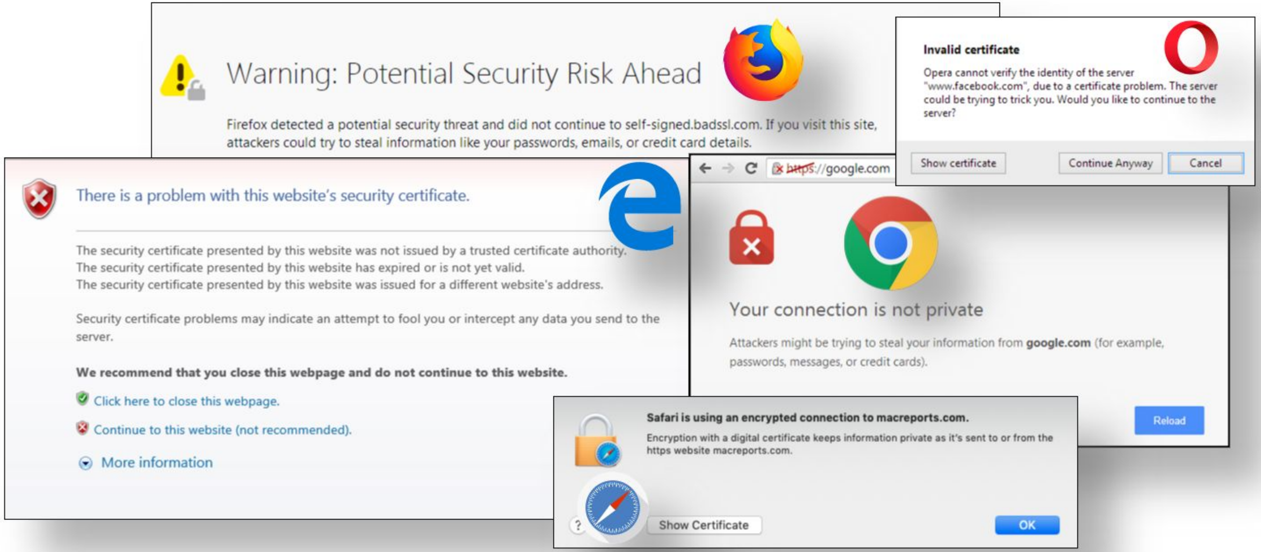
Certification Path



- The hierarchy:
  Website certificate - Intermediate CA certificate - Root CA certificate

- Multiple certification paths could exist - could lead to errors

Certificate Errors



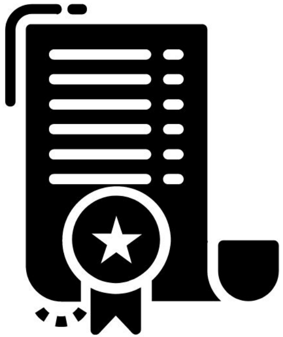
The Heartbleed Bug

- In March 2014, Google discovered a programming mistake in the popular OpenSSL library's implementation of the TLS Heartbeat Extension.
- Allows attackers to read sensitive memory from vulnerable servers, potentially including cryptographic keys, login credentials, and other private data.
- Recovery:
  Patching, revocation of the keys, reissuing keys and replacing certificates.
- Lesson:
  Support for critical projects;
  Develop a method for scalable revocation that can gracefully accommodate mass revocation events;
  Vulnerability disclosure;
  Notification and patching;

Certificate Rotation

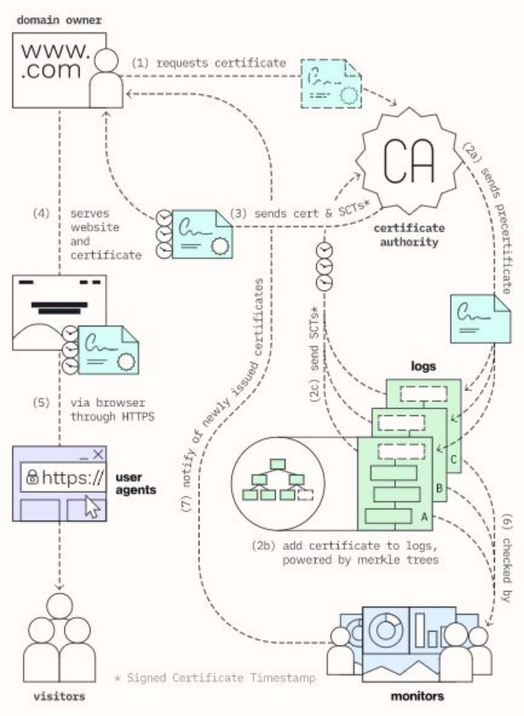
- The replacement of existing certificates with new ones
  Happens when:
  1. Any certificate expires.
  2. A new CA authority is substituted for the old; thus requiring a replacement root certificate for the cluster.
  3. New or modified constraints need to be imposed on one or more certificates.
  4. A security breach has occurred, such that existing certificate-chains can no longer be trusted.

- Example:
  Internal certificate rotation within a company: use of thumbprints vs subject name



Certificate Transparency

- Used for monitoring and auditing digital certificates
- Steps:
    - Website owner requests a certificate from the CA
    - CA issues a precertificate
    - CA sends precertificates to logs
    - Precertificates are added to the logs
    - Logs returns signed certificate timestamps (SCTs) to the CA
    - CAs send the certificate to the domain owner
    - Browsers and user agents help keep the web secure
    - Logs are cryptographically monitored



Thanks for coming to section!