Cristina Nita-Rotaru

CY2550: Foundations of Cybersecurity Section 03

Crypto Module: Public-key cryptography, Diffie Hellman, RSA, hash functions, integrity and authenticated encryption, digital signatures, authenticity of public keys, TLS

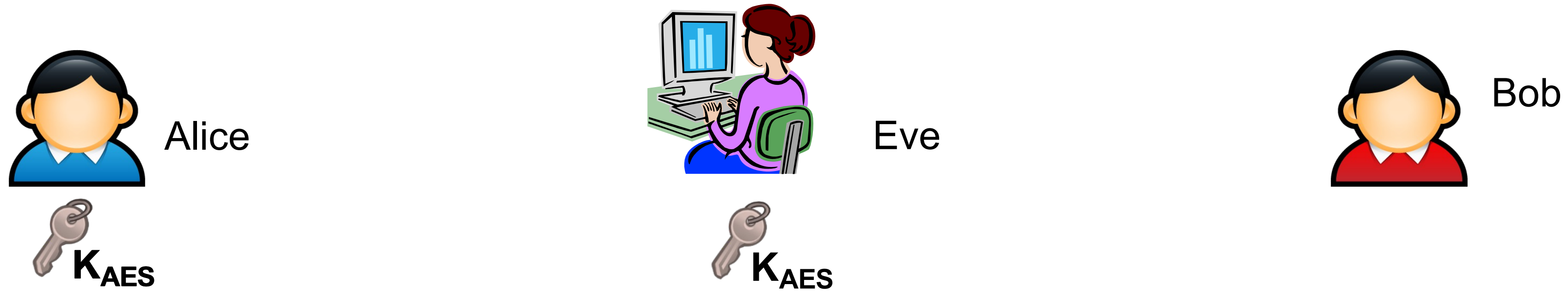
Outline

- Public-key cryptography
- Diffie Hellman Key Exchange
- RSA Encryption
- Cryptographic hash functions
- Integrity and authenticated encryption
- Digital signatures
- Authenticity of public keys
- TLS

Public Key Cryptography

Limitation of symmetric key crypto

- How do you securely exchange keys with someone?
- Easy(ish) to do if you can meet them in person
- However, the Internet is untrusted
  - You can't exchange shared secrets over an untrusted medium



Public Key Cryptography

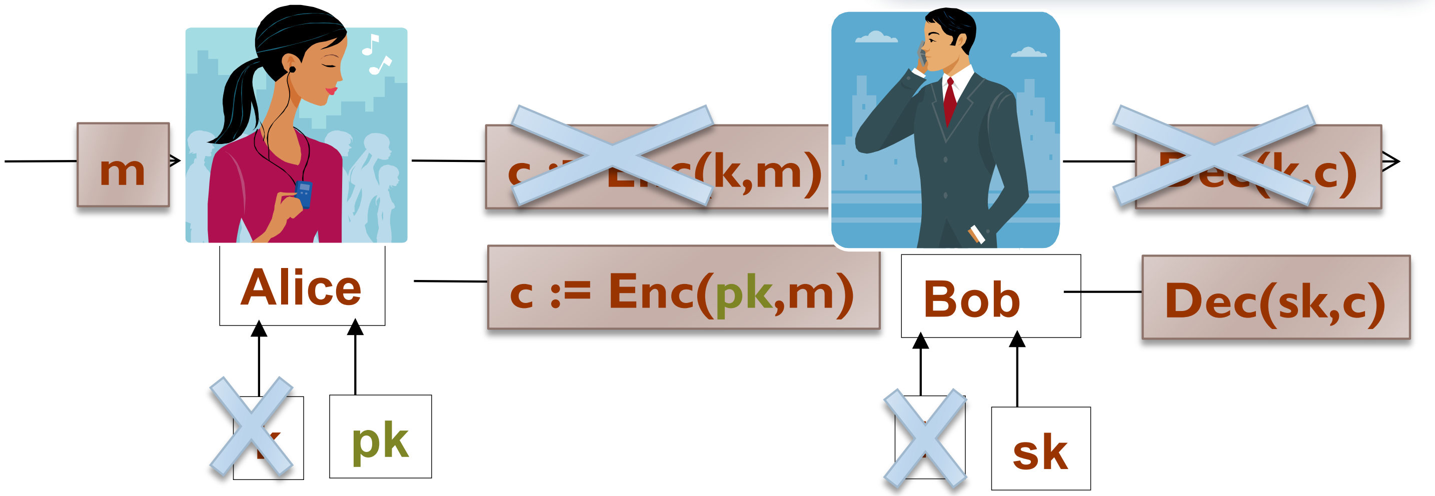
- Public key cryptography, a.k.a. asymmetric cryptography
  - Each principal has two keys: private (secret) and public
  - A message encrypted with one key must be decrypted by the other
  - Thus, the public key can be sent in-the-clear over the Internet
- Security is based on **Very Hard Math Problems**
  - Fast to verify a given solution for a given instance
  - Hard to finds solutions for a given instance in polynomial time
- Many different algorithms that offer different security properties
  - Diffie-Hellman, RSA, Goldwasser-Micali, ElGamal
- Forms the basis for most modern secure communication protocols
  - IPsec, SSL, TLS, S/MIME, PGP/GPG, etc.

Public Key Encryption

Instead of using one key **k**,
use **2** keys **(pk,sk)**, where
pk is used for **encryption**,
sk is used for **decryption**.

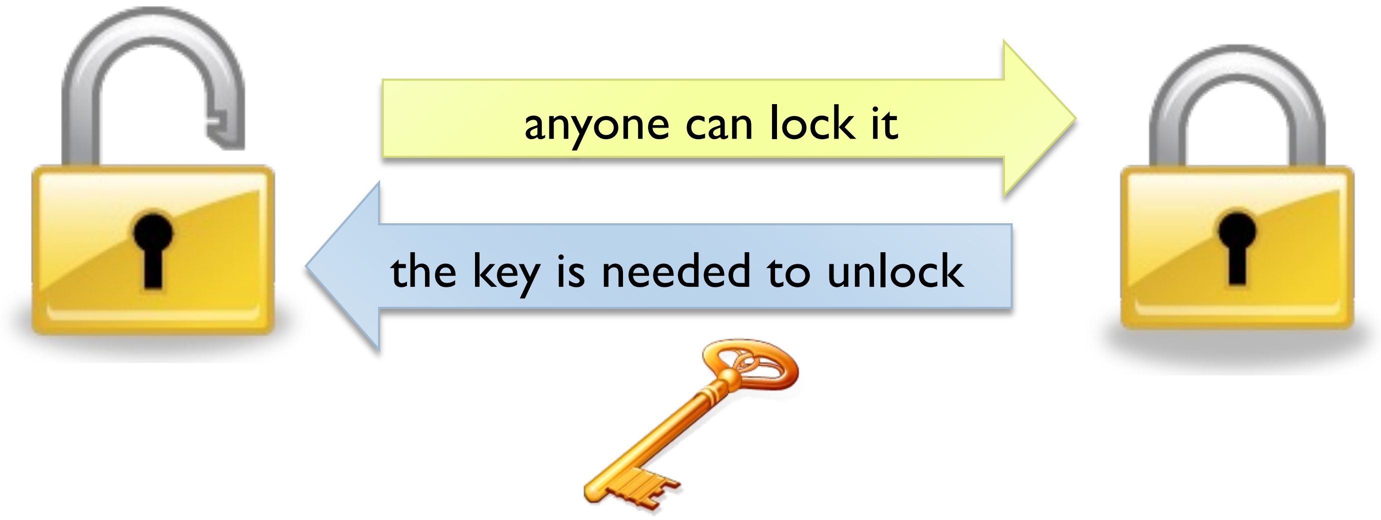
pk can be public, and only sk has to be kept secret!

That's why it's called:
public-key cryptography



Analogy

Examples padlocks:



Use of Public-Key Cryptography

- Public-Key Encryption
  - Examples: RSA, ElGamal
- Digital Signatures:
  - Authenticate messages
  - Examples: RSA, DSA
- Key Exchange
  - Protocols to establish a secret key between two parties
  - Examples: Diffie-Hellman
- Intuition for all these
  - Computation in one direction is "easy", but "hard" in the reverse
  - Hardness assumptions imply that adversary cannot reverse computation

A little bit of history

- **Diffie and Hellman** were the first to publish a paper containing the idea of the public-key cryptography:

  W.Diffie and M.E.Hellman, **New directions in cryptography** IEEE Trans. Inform. Theory, IT-22, 6, **1976**, pp.644-654.

- A similar idea was described by **Ralph Merkle**:
  - in **1974** he described it in a project proposal for a Computer Security course at UC Berkeley (it was rejected)
  - in **1975** he submitted it to the CACM journal (it was rejected)

  (see http://www.merkle.com/1974/)

- 1977: R. Rivest, A. Shamir and L. Adelman published the first construction of public-key encryption (RSA)

- It 1997 the GCHQ (the British equivalent of the NSA) revealed that they knew it already in **1973**.

Diffie Hellman Key Exchange

Diffie-Hellman Key Exchange (Turing Award 2015)

- Goal
  - Share a secret key over a public channel in presence of eavesdropping adversary
- Really should be called Diffie-Hellman-Merkle
  - Ralph Merkle developed the mathematical theories
  - Whitfield Diffie and Martin Hellman developed the protocol
- Security is based on the discrete logarithm problem
  - Compute k such that $b^k = g$ mod p, where b, g, and k are all integers and p is a large prime
  - Possible that no solution exists given arbitrary b and g
  - Best known algorithms are exponential time

The Diffie-Hellman protocol

Fix a large prime p (e.g. 600 digits)

Fix an integer g in $\{1, \ldots, p\}$

Alice | **Bob**

choose random a in $\{1,\ldots,p-1\}$ | choose random b in $\{1,\ldots,p-1\}$

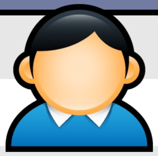
$$\xrightarrow{\quad p, g, A \leftarrow g^{a} \bmod p \quad}$$

$$\xleftarrow{\quad B \leftarrow g^{b} \bmod p \quad}$$

$$B^{a} \pmod{p} = (g^{b})^{a} = k_{AB} = g^{ab} \pmod{p} = (g^{a})^{b} = A^{b} \pmod{p}$$

Diffie-Hellman Example

Knows	Doesn't Know
$p = 23, g = 5$	
$a = 6$ Alice	$b = ?$
$A = g^a \bmod p$ $A = 5^6 \bmod 23 = 8$	
$B = 19$	
$s = B^a \bmod p$ $= 19^6 \bmod 23 =$	



Eavesdropper

Bob

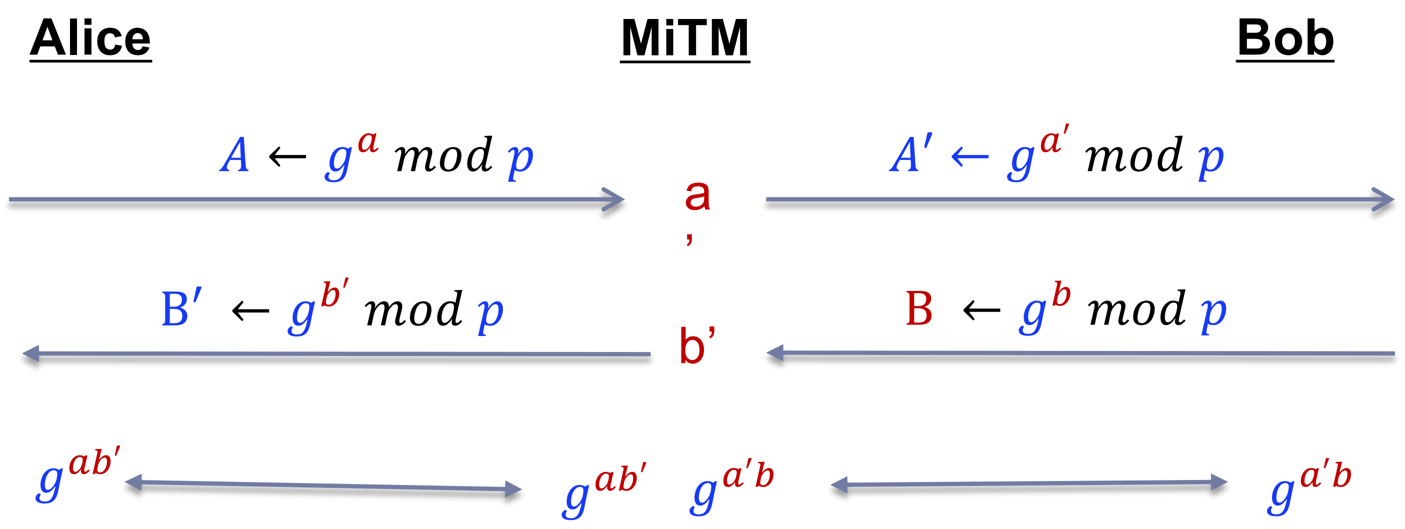
Knows	Doesn't Know
$p = 23, g = 5$	
	$a = ?, b = ?$
$A = 8, B = 19$	

Knows	Doesn't Know
$p = 23, g = 5$	
$b = 15$	$a = ?$
$B = g^b \bmod p$ $= 5^{15} \bmod 23$ $=19$	
$A = 8$	
$s = A^b \bmod p$ $= 8^{15} \bmod 23$ 2	

Calculating s requires solving for *a* or *b*, which is the discrete logarithm problem

Man-in-the-middle attack

As described, the protocol is insecure against **active** attacks


Alice
MiTM
Bob
$A \leftarrow g^{a} \bmod p$
a'
$A' \leftarrow g^{a'} \bmod p$
$B' \leftarrow g^{b'} \bmod p$
b'
$B \leftarrow g^{b} \bmod p$
$g^{ab'}$
$g^{ab'}$
$g^{a'b}$
$g^{a'b}$

Attacker relays traffic from Alice to Bob and reads it in clear

We will see later in the class how to fix this

RSA

Public-key Encryption

- Encryption algorithm: Enc(pk, m); decryption Dec(sk, c)
- RSA algorithm invented by Rivest, Shamir, and Adleman in 1978
  - Equivalent system invented by Clifford Cox in 1973, but GCHQ classified it
- RSA is the dominant public key cryptosystem today
  - Algorithm was commercialized by RSA Security
  - RSA Security created a certificate authority that eventually became Verisign

RSA Algorithm

- Security is based on the difficulty of factoring the product of primes
  - Alice chooses two secret primes p and q, $n = pq$, $\phi(n) = (p-1)(q-1)$
  - Choose e such that $1 < e < \phi(n)$, and $gcd(e, \phi(n)) = 1$
  - $<n, e>$ is Alice's public key
  - Private key $d = e^{-1} \bmod \phi(n)$; $d \cdot e = 1 \bmod \phi(n)$
- Encryption and decryption
  - Given a message M, $0 < M < n$
  - Compute ciphertext $C = M^e \bmod n$
  - To decipher, compute $C^d \bmod n = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n = M$
  - Use Euler's theorem: $x^{\phi(n)} = 1 \bmod n$

RSA Example

$p = 11, q = 7, n = pq = 77, \phi(n) = 60$

$e = 37, d = 13$ ($ed = 481$, $ed \bmod 60 = 1$)

If $M = 15$ then $C = M^e \bmod n = 15^{37} \bmod 77 = 71$

$C^d \bmod n = 71^{13} \bmod 77 = 15 = M$

IND-CPA security for Public-Key Encryption

- In public-key encryption, everyone knows the public key
- That means everyone (including the adversary) can encrypt any message
- IND-CPA and IND-EAV are equivalent notions of security!
- Another reason why we demand IND-CPA at a minimum for symmetric-key encryption

Plain RSA Encryption

Plain (textbook) RSA encryption:

- public key: <n, e> Encrypt: $c \longleftarrow M^e \bmod n$
- secret key: < p, q, d > Decrypt: $c^d \longrightarrow M \bmod n$

Insecure cryptosystem !! !

- Is not IND-CPA secure and many attacks exist
- Deterministic (public key) encryption is never IND-CPA secure

Attacks Against RSA

- The length of $n=pq$ reflects the strength
  - 700-bit n factored in 2007
  - 768 bit factored in 2009
- 1024 bit for minimal level of security today
  - Likely to be breakable in near future
  - Recommended use of 2048 or 4096 bits
- RSA encryption/decryption speed is quadratic in key length

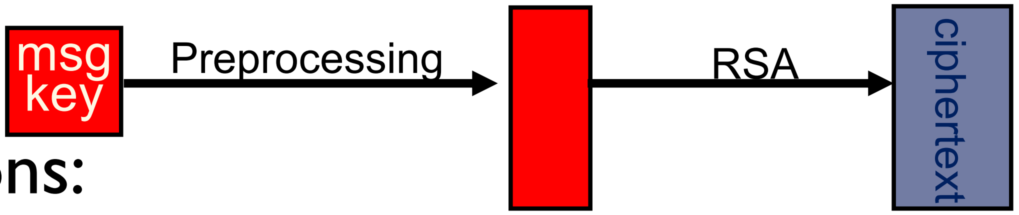
Computationally Hard Problems

- RSA problem:
  - Given public RSA key, decrypt $m^e \bmod n$ for a random message m
- RSA assumption:
  - Solving the RSA problem is difficult
- Factoring assumption
  - If $n=pq$ with p and q primes, cannot factor for large n
  - If factoring can be done in polynomial time, then RSA problem can be solved in polynomial time

RSA encryption in practice

Never use plain RSA.

RSA in practice

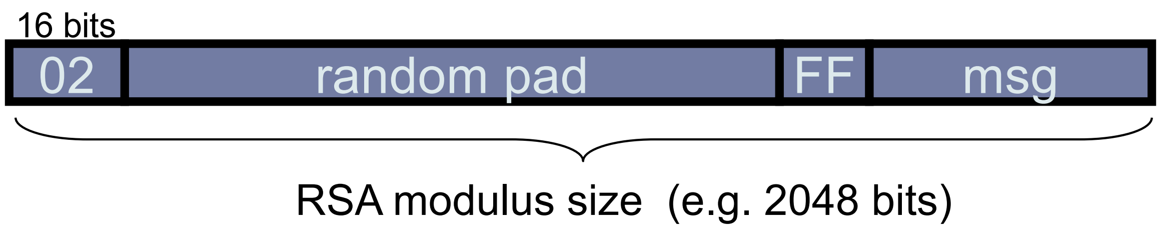


Main questions:

- How should the preprocessing be done?
- Can we argue about security of resulting system?
- How can we randomize it to be IND-CPA secure?

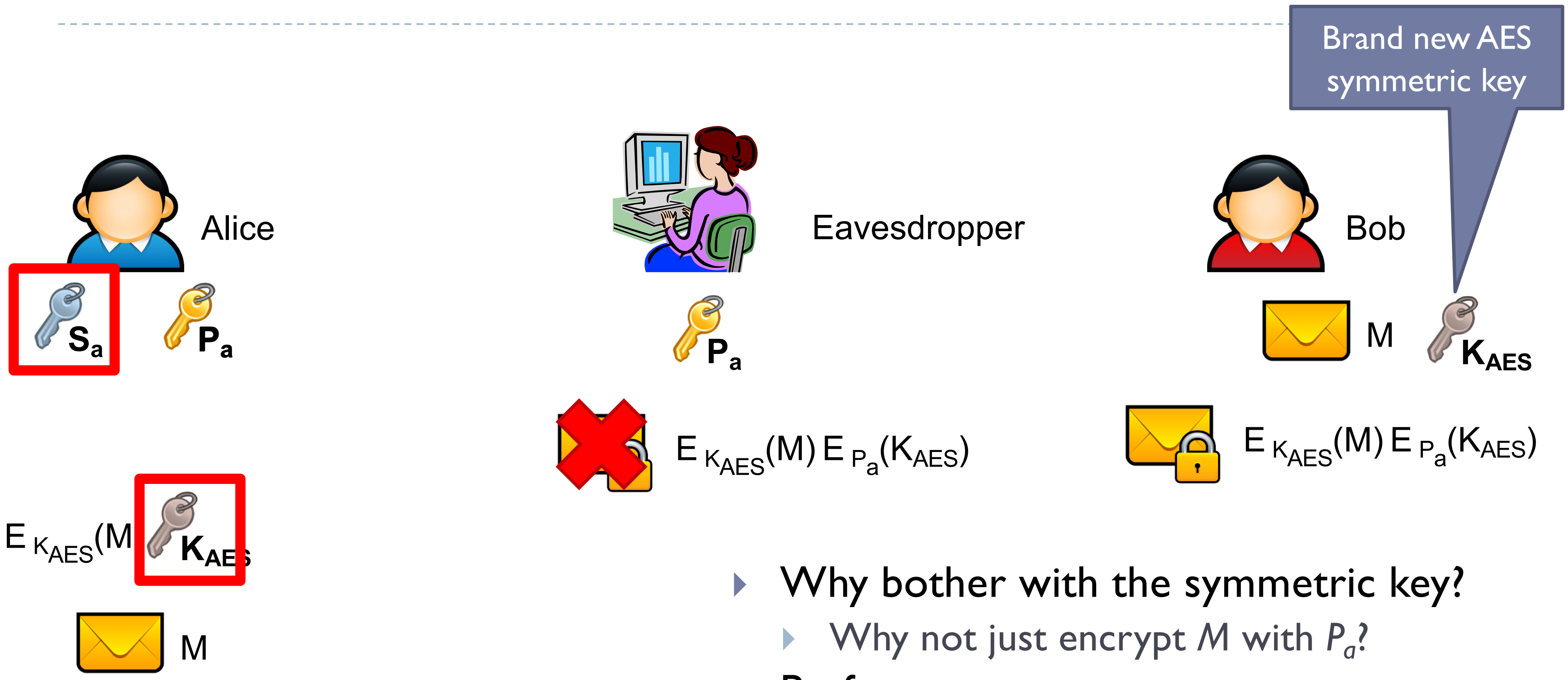
PKCS1 v1.5

PKCS1 mode 2: (encryption)



- Add random pad before the message
- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS, but it is not IND-CPA secure!
- There are newer versions that are secure (e.g., OAEP)

Public Key Crypto Example


Brand new AES symmetric key
Alice
Eavesdropper
Bob
S_a
P_a
P_a
M
K_{AES}
$E_{K_{AES}}(M)\ E_{P_a}(K_{AES})$
$E_{K_{AES}}(M)\ E_{P_a}(K_{AES})$
$E_{K_{AES}}(M$
K_{AE}
M

- Why bother with the symmetric key?
  - Why not just encrypt *M* with *P_a*?
- Performance
  - Asymmetric crypto is slow, symmetric is fast
  - Use asymmetric for K (which is small)
  - Use symmetric for M (which is large)

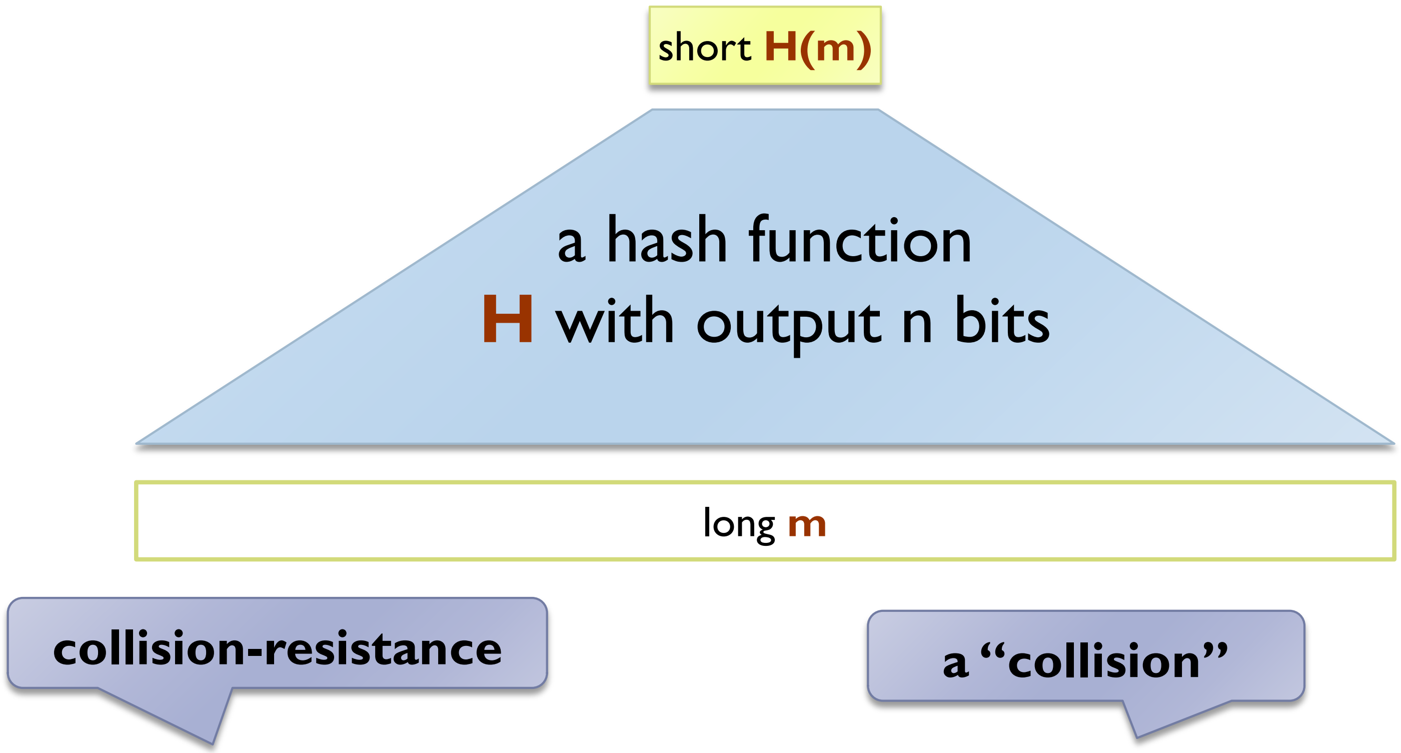
Key sharing can be done with a Key Exchange protocol (e.g., Diffie-Hellman)

Cryptographic hash functions

Cryptographic Hash Functions

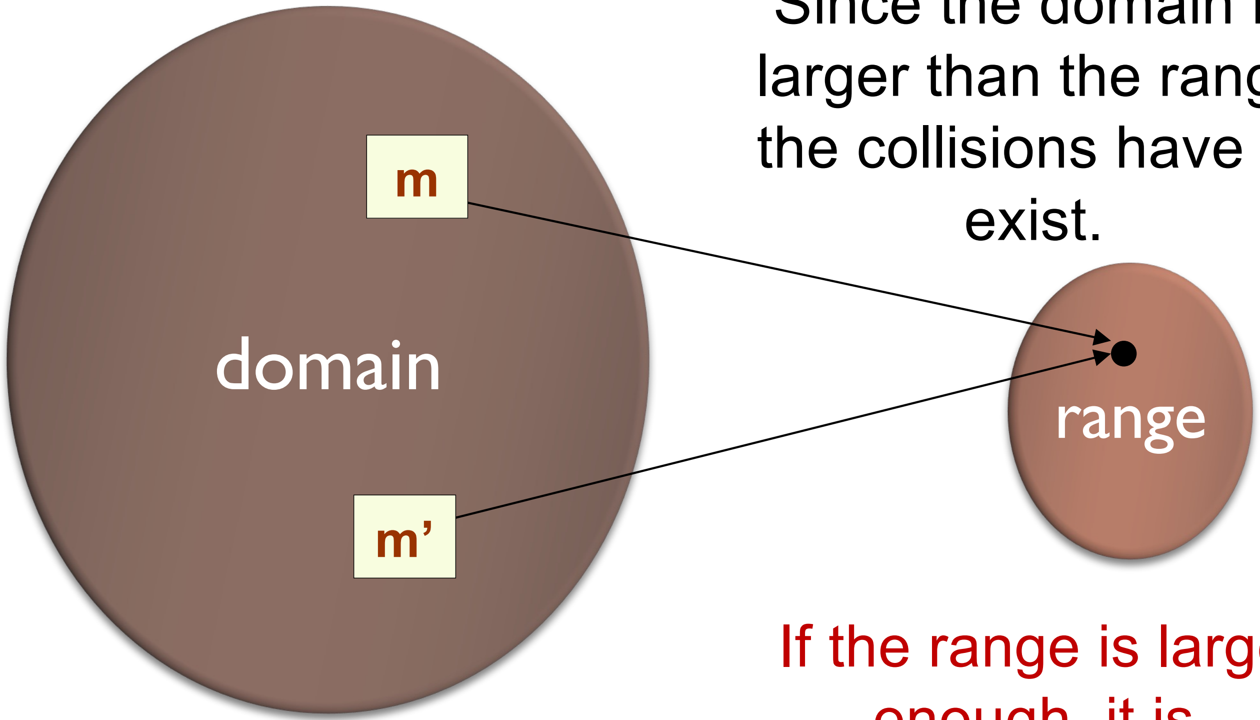
- Cryptographic hash function transform input data into scrambled output data
  - Arbitrary length input → fixed length output
  - Deterministic: H(A) is always the same
  - High entropy:
    - md5('security') = e91e6348157868de9dd8b25c81aebfb9
    - md5('security1') = 8632c375e9eba096df51844a5a43ae93
    - md5('Security') = 2fae32629d4ef4fc6341f1751b405e45
  - Collision resistant
    - Locating A' such that H(A) = H(A') takes a long time
    - Example: 2^{21} tries for md5

Collision-resistant hash functions



Requirement: it should be hard to find a pair **(m,m')** such that **H(m) =H(m')**

Collisions always exist



Since the domain is larger than the range the collisions have to exist.

If the range is large enough, it is computationally hard to find collisions.

Examples

Are these hash functions collision resistant?

- H:$\{0,1\}^{2n}$ -> $\{0,1\}^n$
  - H(x||y) = x XOR y
- H:$\{0,1\}^{2n}$ -> $\{0,1\}^n$
  - Let p be an n-bit prime
  - H(x||y) = x + y mod p
- H: N -> $\{0,1\}^n$
  - Let p be an n-bit prime
  - H(x) = ax + b mod p, p prime

History of hash functions

H is a **collision-resistant hash function** if it is "*practically impossible to find collisions in **H***".

- **1991**: MD5
- **1995**: SHA1
- **2001**: SHA2 -- SHA-256 and SHA-512
- **2004**: Team of Chinese researchers found collisions in MD5
- **2007**: NIST competition for new SHA3 standard
- **2012**: Winner of SHA3 is Keccak

Well known hash functions

- MD5
  - Outputs 128 bits
  - Collision resistance totally broken in 2004
- SHA1
  - Outputs 160 bits
  - Partially broken: method exists to find collisions in 2^{80} tries
  - Deprecated
- SHA2 family (SHA-224, SHA-256, SHA-384, SHA-512)
  - SHA-224 matches the 112 bit key length of 3DES
  - SHA-256, SHA-384, SHA-512 match the key lengths of AES (128, 192, 256 bits)
  - Considered safe

SHA3

- 2007: NIST opens competition for new hash functions
- 2008: Submission deadline, 64 entries, 51 make the cut
- 2009: 14 candidates move to round 2
- 2010: 5 candidates move to round 3
- 2011: final round of public comments
- 2012: NIST selects *keccak* (pronounced "catch-ack") as SHA3
  - Created by Guido Bertoni, Joan Daemen, Gilles Van Assche, Michaël Peeters

Birthday paradox

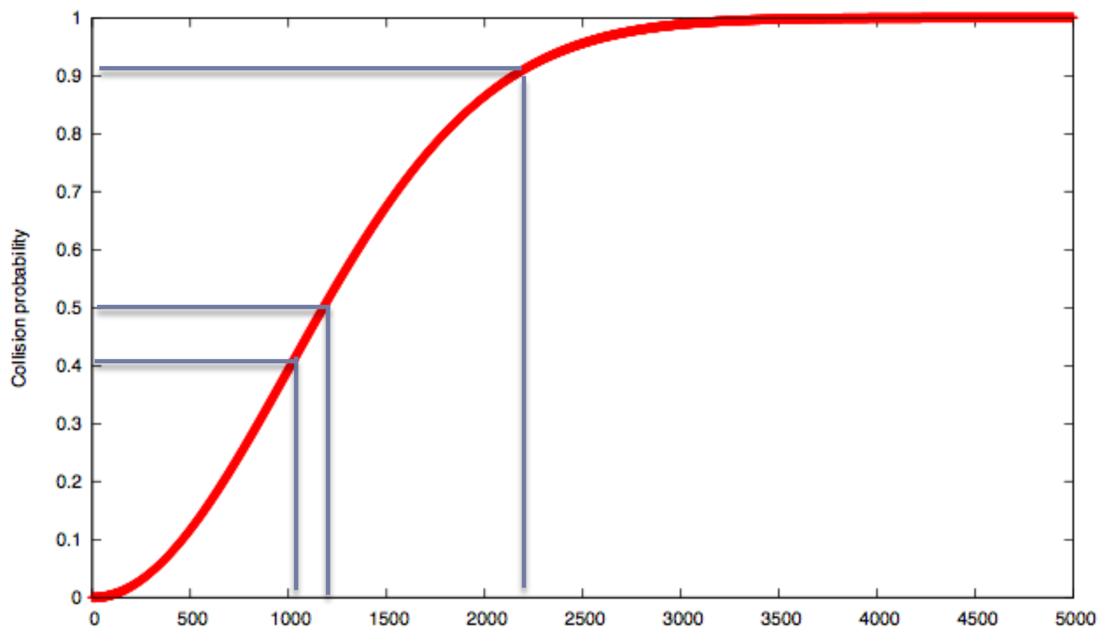
- If we choose q elements $y_1, \ldots y_q$ at random from {1,…,N}, what is the probability that there exists i and j such that $y_i = y_j$?

N=365 possible days

- What is the probability that two people have the same birthday?
- When is this probability higher than 0.5?

Collision probability

N=10^6



# samples n

- If $q = \Theta(\sqrt{N})$ items, then probability of collision is approx. ½
- Birthday paradox
  - N = 365, q = 23
- Hash functions
  - $N = 2^{256}, q = 2^{128}$
- Implies n/2 level of security for n-bit hash function in best case

Security of encryption vs hash functions

- Encryption:
  - Size of key determines resilience to brute-force attacks
  - A key size of n bits, means 2^n work for the attacker to find the key
- Hash functions:
  - Size of the output of the hash function determines security to finding collisions
  - Birthday paradox tells us that for a hash function with on output of n bits, the work that an attacker needs to do to find a collision is $2^{n/2}$

For comparable security a block cipher of 128 bits must be paired with a hash function of 256 bits!

Applications of hash functions

- Password authentication, store a hash of the password
  - We will see more about this in about 2 weeks when we will talk about authentication and passwords
- Integrity
  - Communication
  - Storage

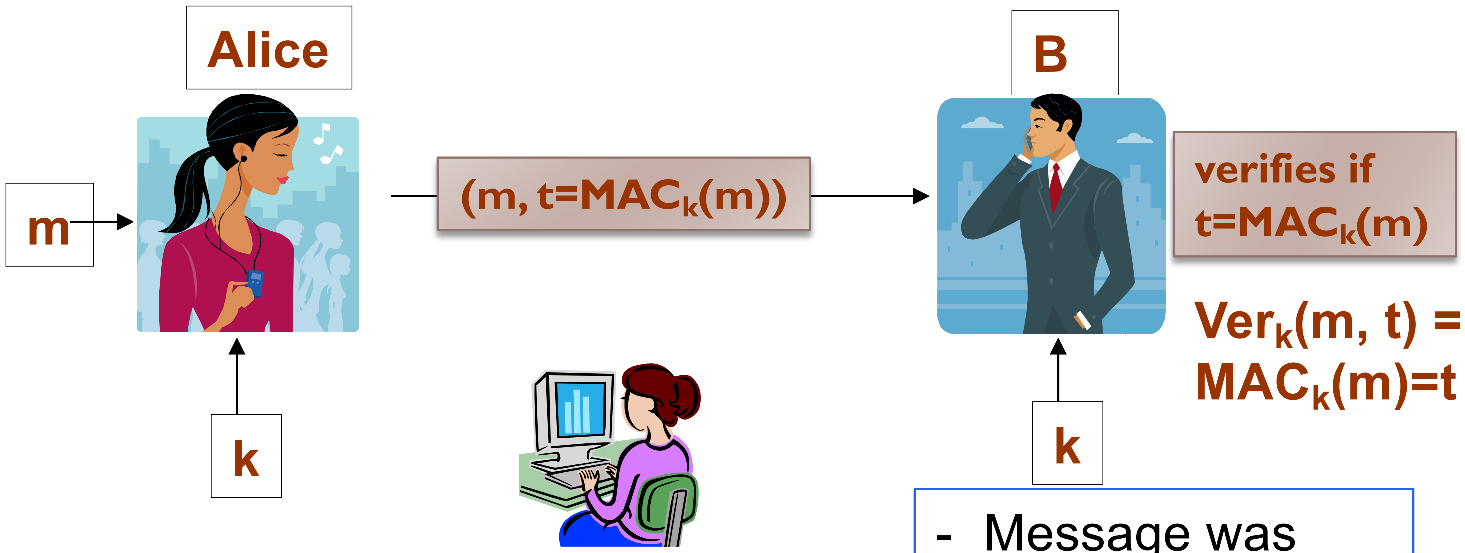
Integrity and authenticated encryption

Integrity

- Active adversaries
  - Can modify messages/ciphertexts in transit
  - Encryption alone (even IND-CPA secure) does not guarantee integrity!
- Protect message integrity
  - Message received by Bob is the original one sent by Alice
  - Message was not modified by adversary
- Scenarios
  - Secure communication on network
  - Protect files stored on disk
- Can be achieved in symmetric-key or public-key settings

Message authentication

- MAC = message authentication code



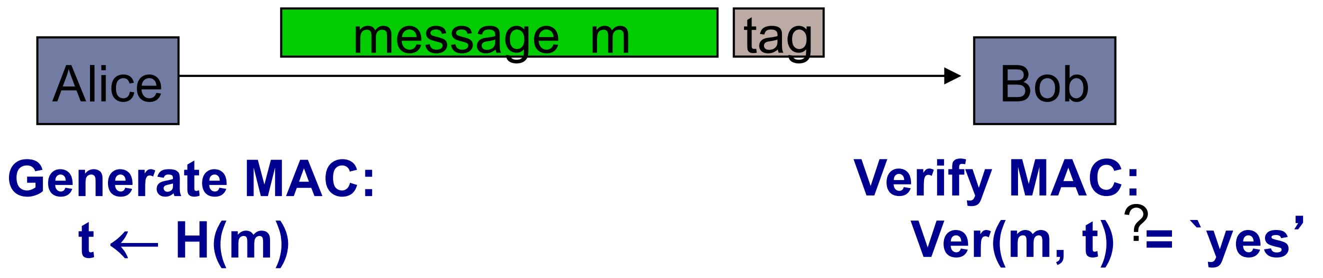
Ver$_k$(m, t) = 1 if MAC$_k$(m)=t

- Message was sent by Alice
- Message was not modified

Adversary can see **(m, t=MAC$_k$(m))**

She should not be able to compute a valid MAC **t'** on any other message **m'**.

Integrity requires a secret key



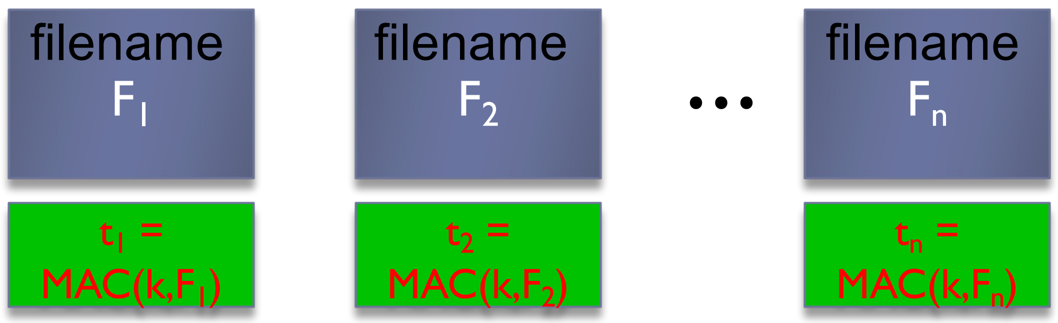
- Attacker can easily modify message m and re-compute the hash.
- Hash designed to detect **random**, not malicious errors.

Message authentication security

- Properties
  - Correctness: If $t = MAC_k(m)$, then $Ver_k(m, t) = 1$
  - MAC is a deterministic function
  - The output of MAC is fixed size (n bits), independent of the length of the input message
- Security (unforgeability)
  - If an attacker has many pairs of messages and integrity tags, he cannot compute a new tag on a message
  - If A is given $(m_1, t_1), \ldots (m_\mathrm{q}, t_\mathrm{q})$ then A cannot output (m', t') such that: $Ver_k(m', t') = 1 \quad and\ m' \notin \{m_1, \ldots m_q\}$

Example: protecting system files

Suppose at install time the system computes:



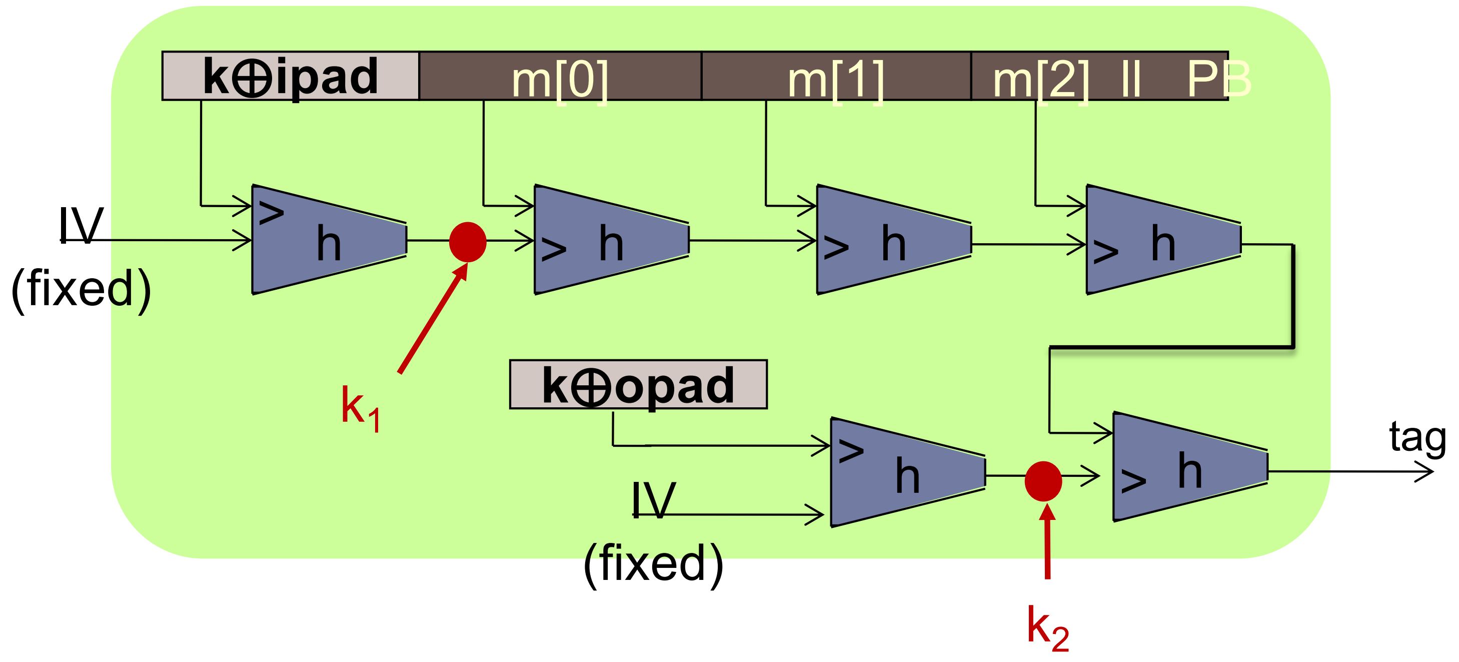
k derived from user's password (e.g., using a hash)

Later malware infects system and modifies system files

User reboots into clean OS and supplies his password

- Then: secure MAC $\Rightarrow$ all modified files will be detected

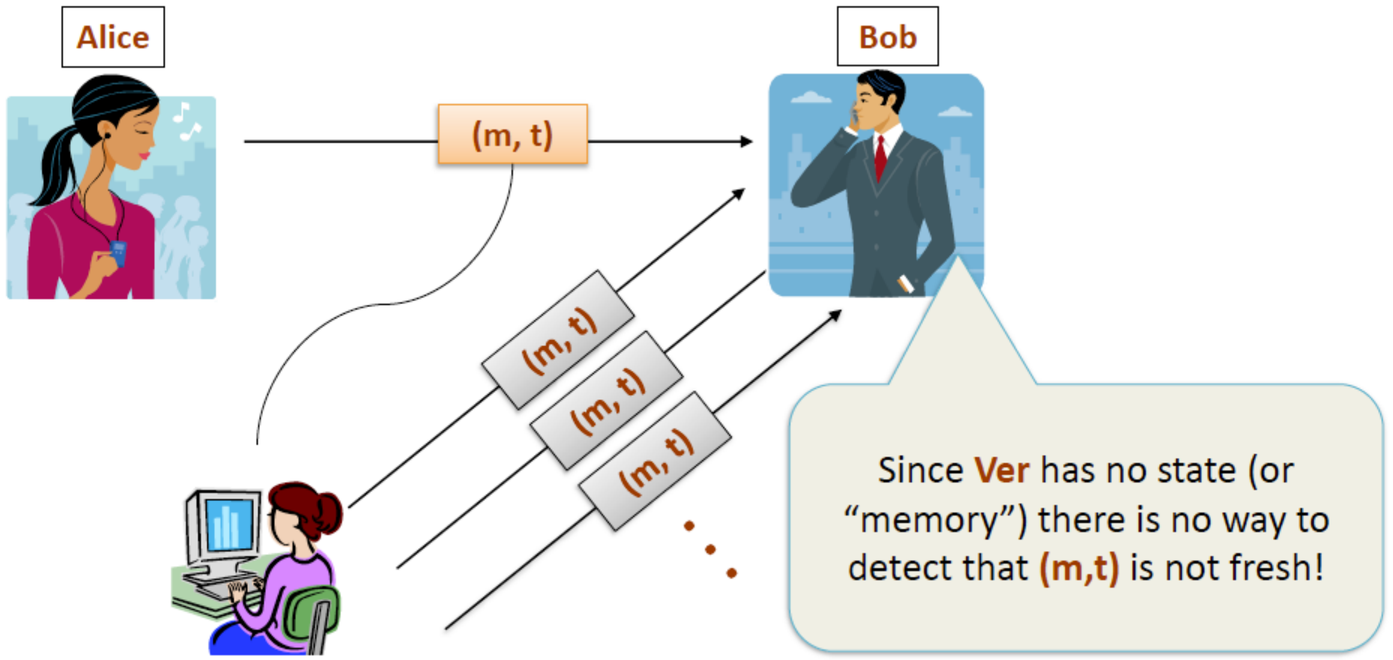
HMAC: Design a MAC from a hash function



- Uses Merkle-Damgaard construction (chaining a collision-resistant hash function)
- Output: last hashed block (no need to recover the message)
- Most widely used MAC on the Internet

Replay attacks

Warning: MACs do not offer protection against the "replay attacks".



Since **Ver** has no state (or "memory") there is no way to detect that **(m,t)** is not fresh!

This problem has to be solved by the higher-level application (methods: **time-stamping**, **sequence numbers**...).

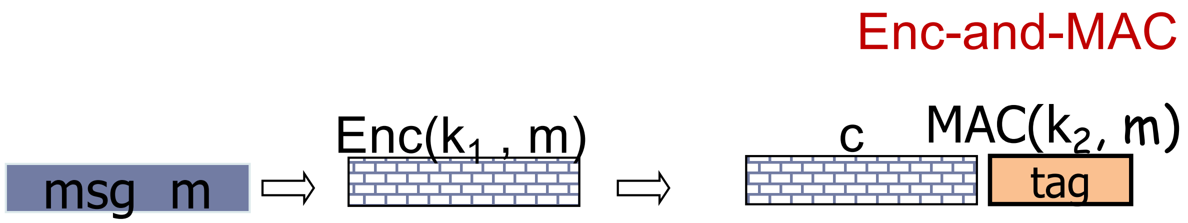
Authenticated encryption

- Combines confidentiality and integrity
- Security properties
  - *Confidentiality*: ciphertext does not leak any information about the plaintext
  - *Integrity*: attacker cannot create new ciphertexts that decrypt properly
- Decryption returns either
  - Valid messages
  - Or invalid symbol (when ciphertext is not valid)

Authenticated Encryption: combining MAC and ENC

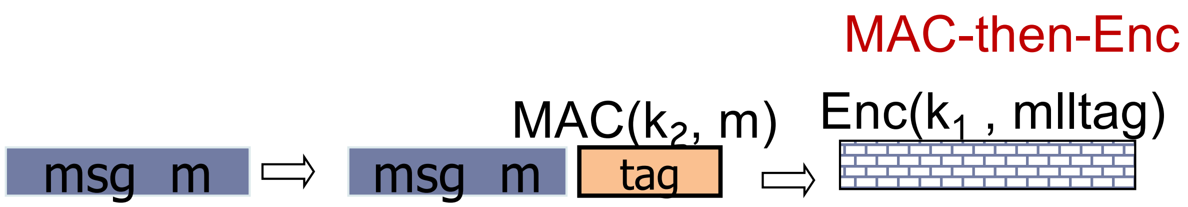
Encryption key k_1. MAC key = k_2

Option 1: (SSH)



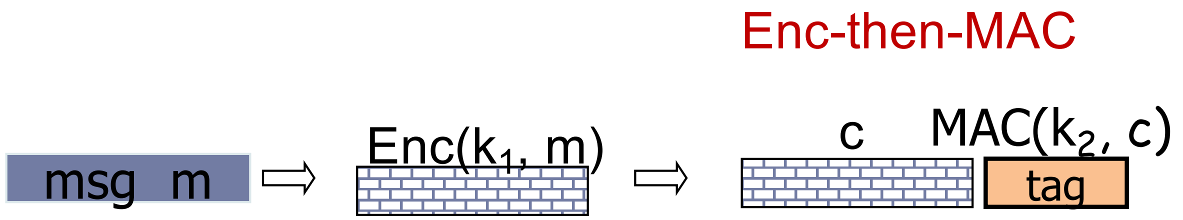
Homework

Option 2: (SSL)



Padding oracle attacks

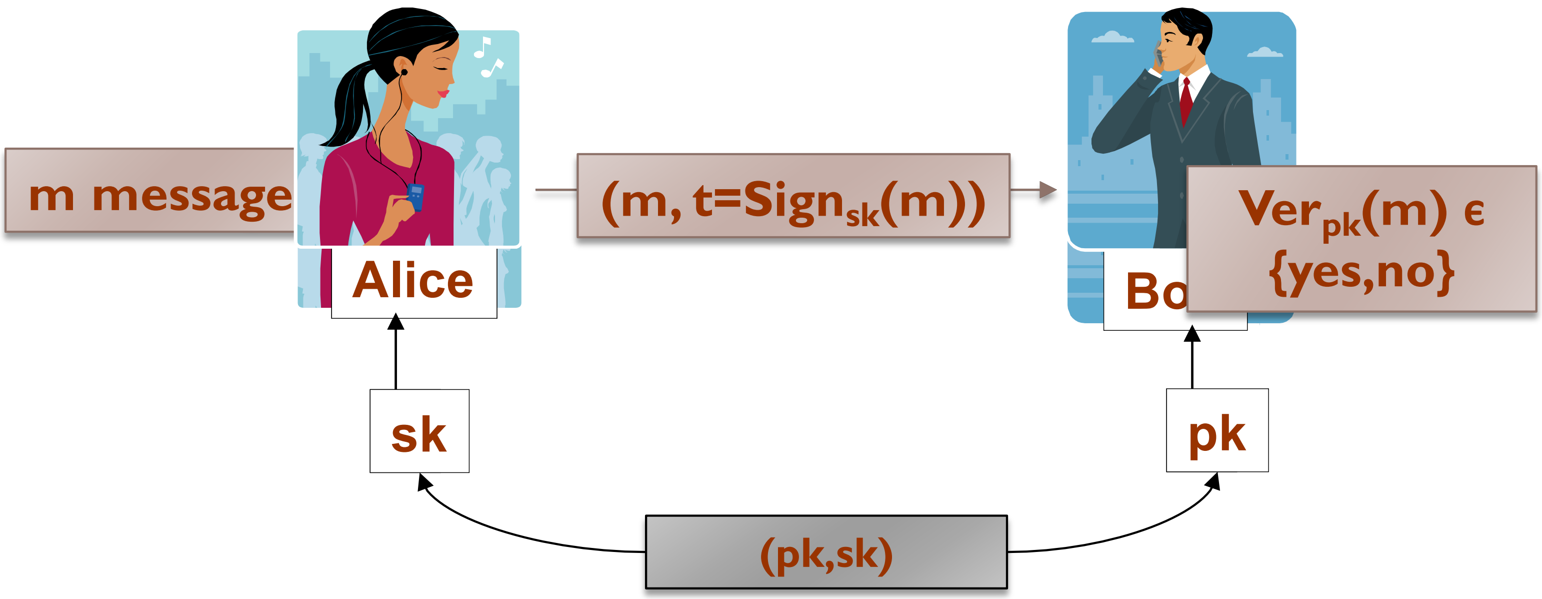
Option 3: (IPsec)



Always Secure!

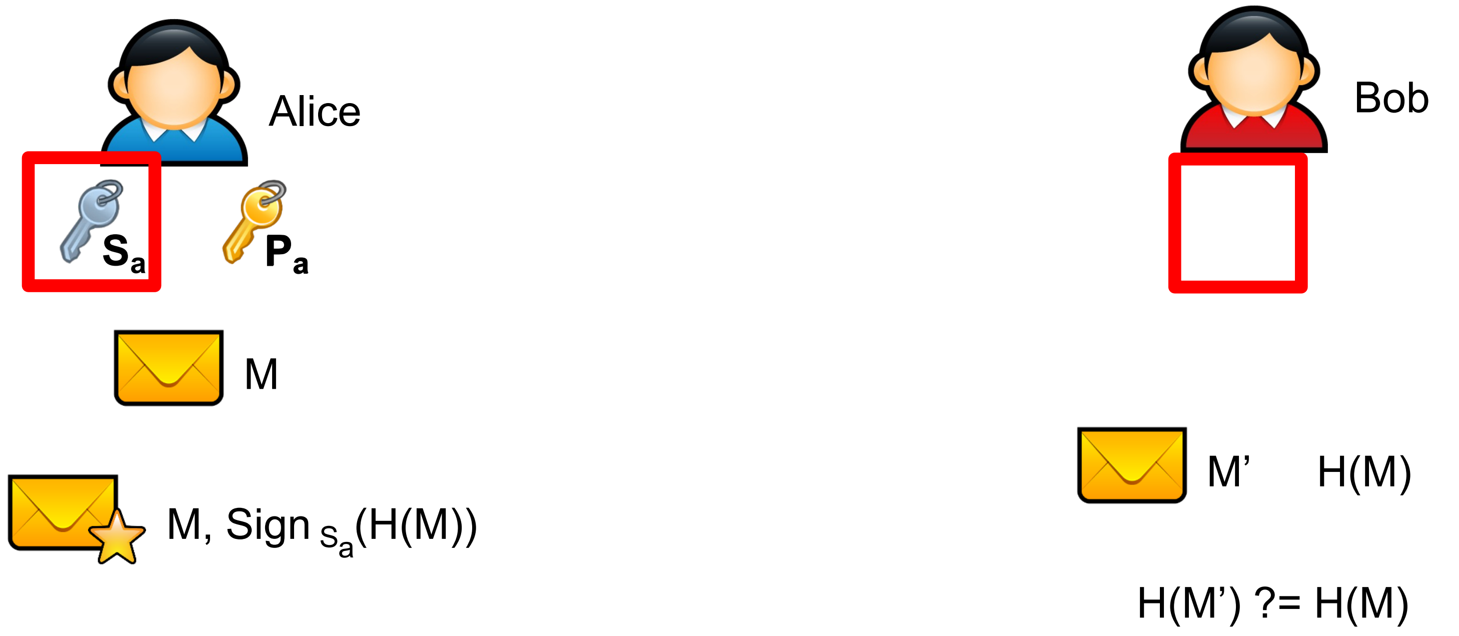
Digital signatures

Digital signature schemes


m message
Alice
$(m, t=\text{Sign}_{sk}(m))$
Bo
$\text{Ver}_{pk}(m) \in \{\text{yes},\text{no}\}$
sk
pk
(pk,sk)

Public key equivalent of MAC

Digital signatures and hash functions



- What can you infer about a signed message?
  - The holder of S_a must have produced the signature
  - The message was not modified, otherwise the hash would not match
  - Assuming hash is collision resistant

Advantages of signature schemes

Digital signatures are equivalent of MACs in public-key world

Provide message integrity

Additional properties (compared to MACs):

1. **Publicly verifiable:** anyone with PK can verify (not for MAC)
2. **Transferable:** can be transferred to another user and verified
3. Provide **non-repudiation:** cannot deny that you signed a message

RSA signature

Before computing the RSA function – apply hash function **H**.

$N = pq$, such that **p** and **q** are large random primes
e is public key such that **$\gcd(e, \varphi(N)) = 1$**
d is secret key such that **$ed = 1 \pmod{\varphi(N)}$**

Sign: $Z_N^* \to Z_N^*$ is defined as:

$$\mathrm{Sign}(m) = \sigma = H(m)^d \bmod N.$$

Ver is defined as:

$$\mathrm{Ver}(m,\sigma) = \text{yes} \quad \text{iff } \sigma^e = H(m) \pmod N$$

Hash-and-sign paradigm

Encryption vs. signatures

Encryption

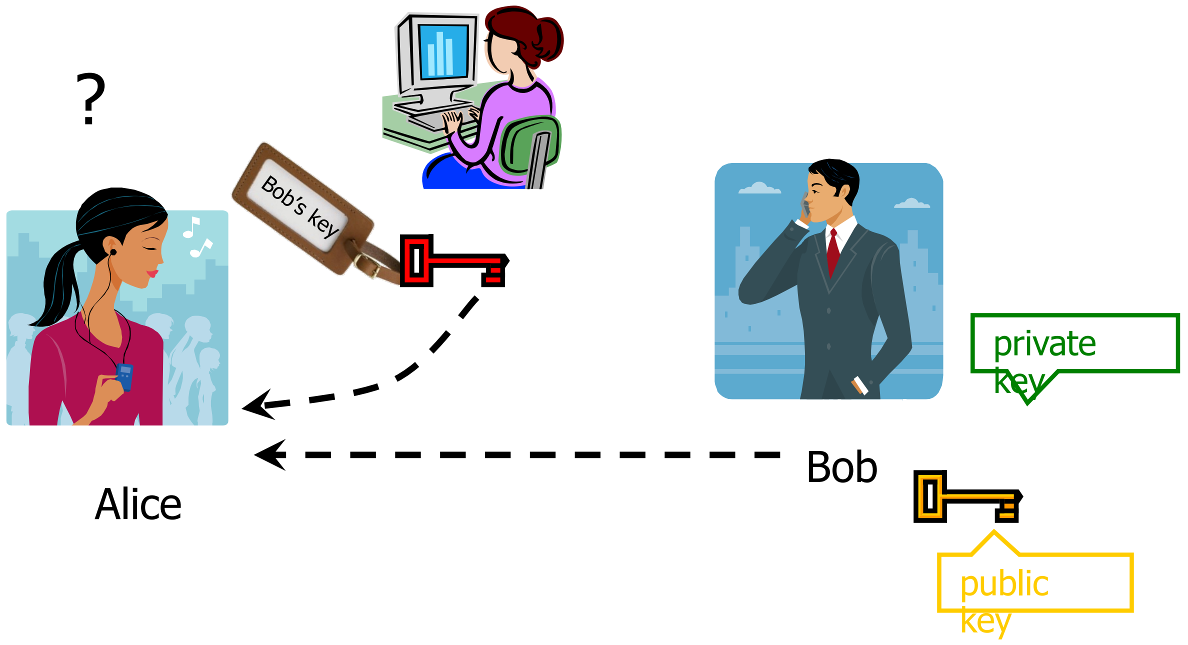
- What does encryption give you?
  - Confidentiality – only the holder of the private key can read the message
- What does authenticated encryption give you in addition?
  - Integrity – if the ciphertext is modified, it will no longer decrypt properly
- What does encryption not give you?
  - Authentication – you have no idea who used your public key to encrypt the message

Digital Signatures

- What do signatures give you?
  - (Weak) Authentication – only the holder of the private key could have signed the message
  - Integrity – if the message is modified, the signature will be invalid
- What do signatures not give you?
  - Confidentiality – the message is not encrypted, it's public

Authenticity of public keys.

Authenticity of Public Keys

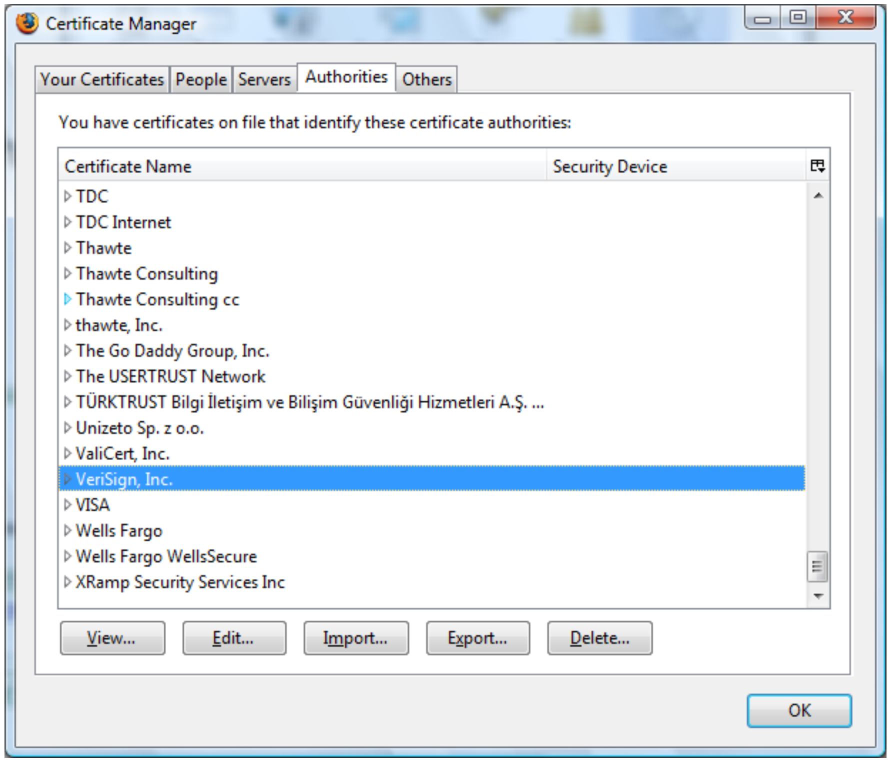


Problem: How does Alice know that the public key she received is really Bob's public key?

PKI: Public Key Infrastructure

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - $\text{Sig}_{\text{Charlie}}(\text{“Bob”}, \text{PK}_{\text{Bob}})$
  - Could Bob sign his own certificate?
  - Web of trust (PGP): users signing each other’s keys
- Common approach: certificate authority (CA)
  - An agency responsible for certifying public keys
  - It generates certificates for domain names (example.com) on the web

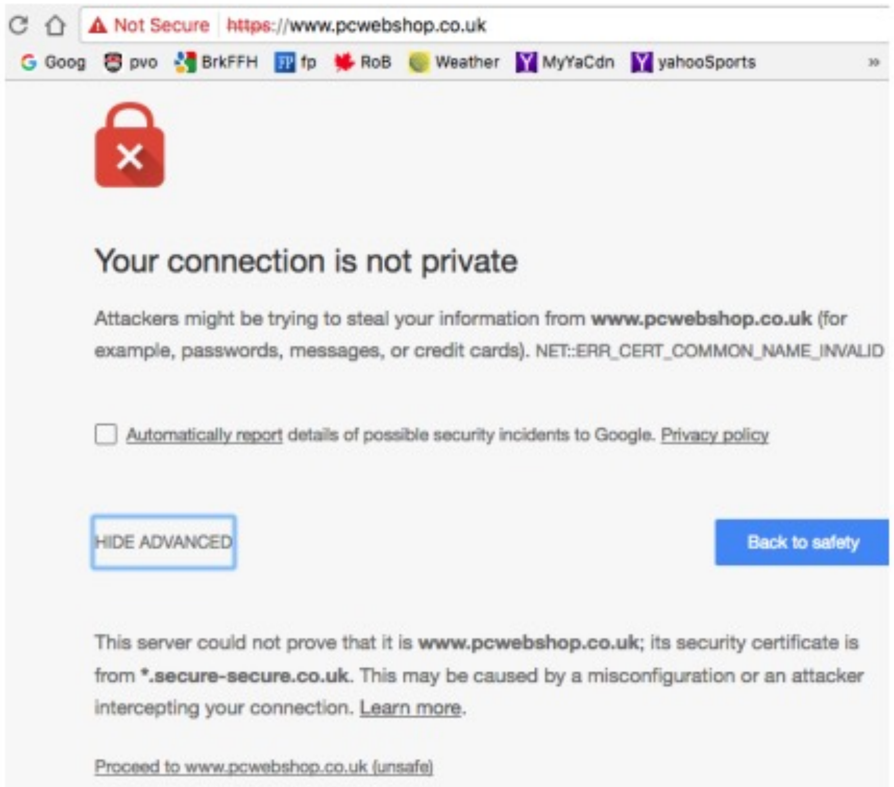
Trusted Certificate Authorities



You have certificates on file that identify these certificate authorities:

Certificate Name	Security Device
TDC	
TDC Internet	
Thawte	
Thawte Consulting	
Thawte Consulting cc	
thawte, Inc.	
The Go Daddy Group, Inc.	
The USERTRUST Network	
TÜRKTRUST Bilgi İletişim ve Bilişim Güvenliği Hizmetleri A.Ş. ...	
Unizeto Sp. z o.o.	
ValiCert, Inc.	
VeriSign, Inc.	
VISA	
Wells Fargo	
Wells Fargo WellsSecure	
XRamp Security Services Inc	

Warning


Not Secure https://www.pcwebshop.co.uk

Your connection is not private

Attackers might be trying to steal your information from **www.pcwebshop.co.uk** (for example, passwords, messages, or credit cards). NET::ERR_CERT_COMMON_NAME_INVALID

Automatically report details of possible security incidents to Google. Privacy policy

This server could not prove that it is **www.pcwebshop.co.uk**; its security certificate is from ***.secure-secure.co.uk**. This may be caused by a misconfiguration or an attacker intercepting your connection. Learn more.

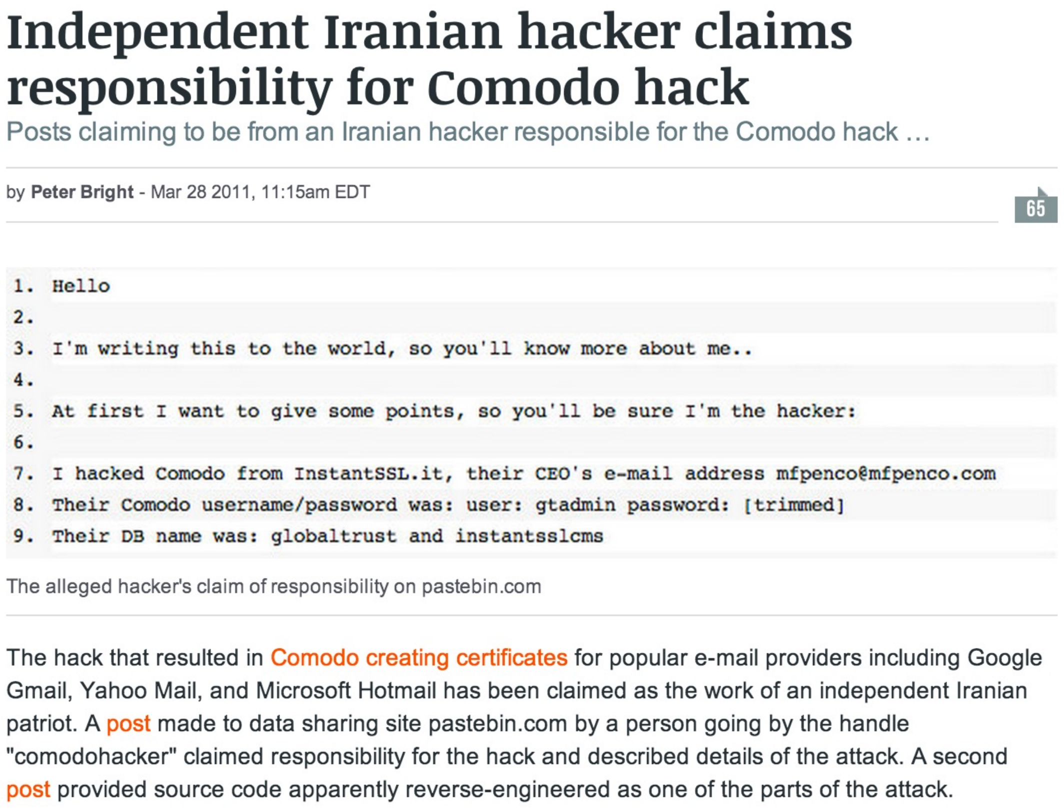
Proceed to www.pcwebshop.co.uk (unsafe)

Figure 8.3: Self-signed site certificate—browser warning (Chrome 56.0.2924.87). The "Back to safety" tab discourages clicking-through to the site despite the browser being unable to verify the certificate chain (this happens when one or more certificates are signed by CAs unrecognized by the browser, i.e., not verifiable using the browser's trust anchors).

CA Hierarchy or PKI

- Browsers, operating systems, etc. have trusted root certificate authorities
  - Firefox 3 includes certificates of 135 trusted root CAs
- A Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - Certificate "chain of trust"
    - $Sig_{Verisign}$("neu.edu", PK_{NEU}), Sig_{NEU}("ccs.neu.edu", PK_{CCS})
- CA responsibilities
  - Verify that someone buying a cert for a domain (e.g., *example.com)* actually controls the domain
  - Verify that buyer knows the secret key associated with the public key
  - Protect its own secret key

Comodo



Independent Iranian hacker claims responsibility for Comodo hack

Posts claiming to be from an Iranian hacker responsible for the Comodo hack …

by **Peter Bright** - Mar 28 2011, 11:15am EDT

65

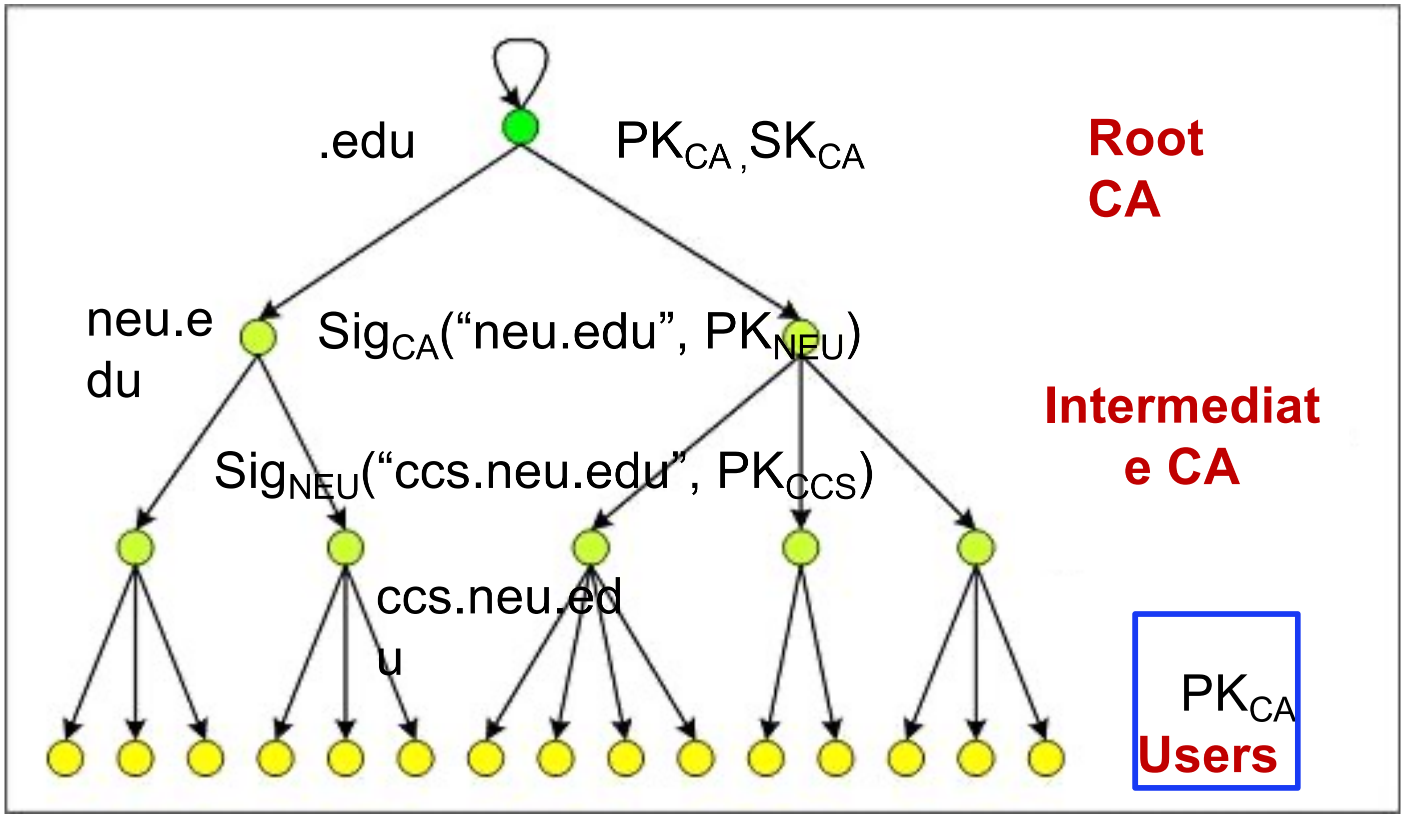
```
Hello

I'm writing this to the world, so you'll know more about me..

At first I want to give some points, so you'll be sure I'm the hacker:

I hacked Comodo from InstantSSL.it, their CEO's e-mail address mfpenco@mfpenco.com
Their Comodo username/password was: user: gtadmin password: [trimmed]
Their DB name was: globaltrust and instantsslcms
```

The alleged hacker's claim of responsibility on pastebin.com

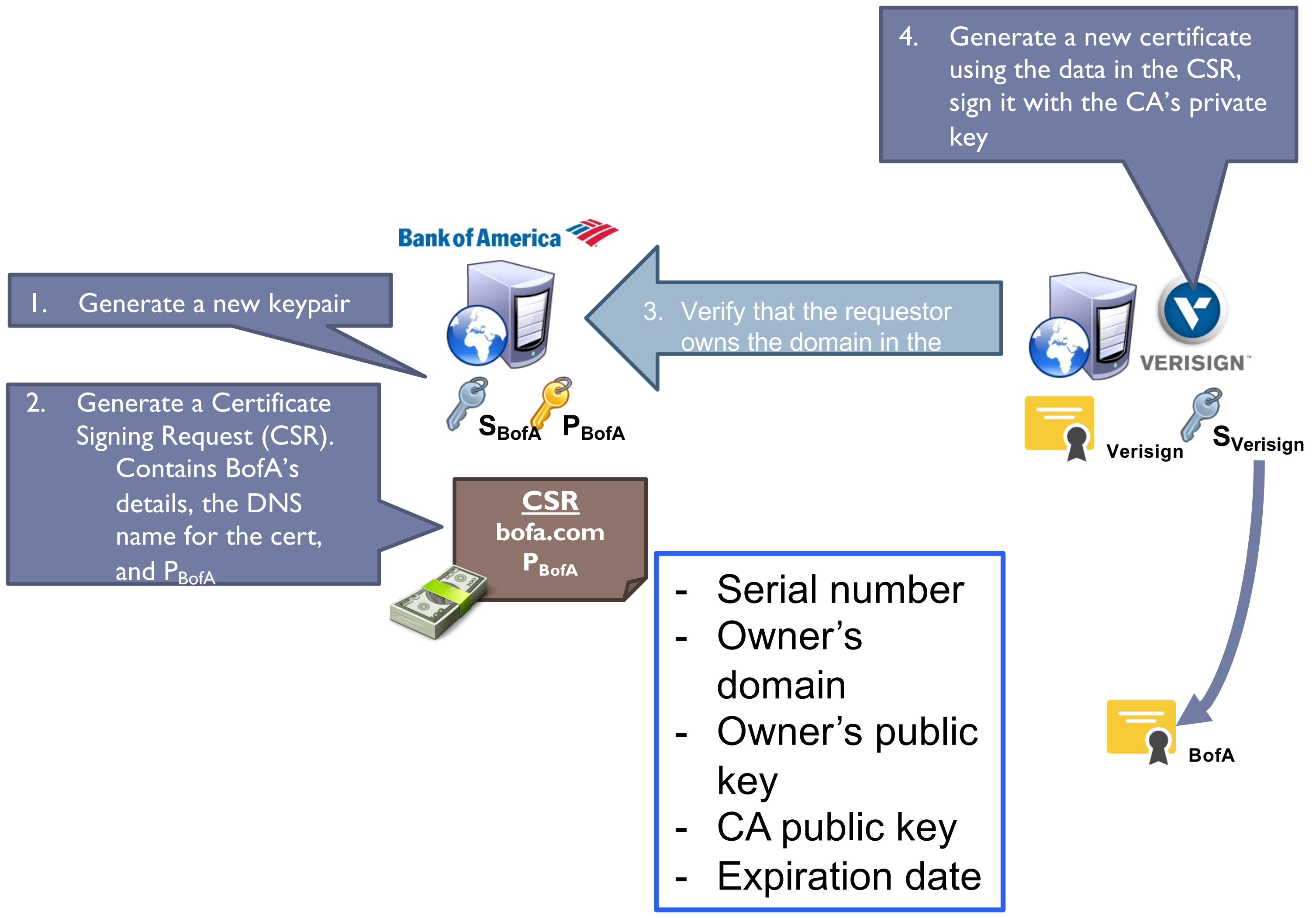
The hack that resulted in Comodo creating certificates for popular e-mail providers including Google Gmail, Yahoo Mail, and Microsoft Hotmail has been claimed as the work of an independent Iranian patriot. A post made to data sharing site pastebin.com by a person going by the handle "comodohacker" claimed responsibility for the hack and described details of the attack. A second post provided source code apparently reverse-engineered as one of the parts of the attack.

What if CA secret key is compromised?

Certificate Hierarchy - PKI



Acquiring a Certificate



1. Generate a new keypair
2. Generate a Certificate Signing Request (CSR). Contains BofA's details, the DNS name for the cert, and P_{BofA}
3. Verify that the requestor owns the domain in the
4. Generate a new certificate using the data in the CSR, sign it with the CA's private key

- Serial number
- Owner's domain
- Owner's public key
- CA public key
- Expiration date

X.509 Certificate

```
Certificate:
    Data:
        Version: 3 (0x2)
         Serial Number:
            0c:00:93:10:d2:06:db:e3:37:55:35:80:11:8d:dc:87
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 Extended Validation Server
CA
        Validity
            Not Before: Apr  8 00:00:00 2014 GMT
            Not After : Apr 12 12:00:00 2016 GMT
         Subject: businessCategory=Private
Organization/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=Delaware/serialNumber=5157550/street=548
4th Street/postalCode=94107, C=US, ST=California, L=San Francisco, O=GitHub, Inc., CN=github.com
         Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b1:d4:dc:3c:af:fd:f3:4e:ed:c1:67:ad:e6:cb:
```

Issuer: who generated this cert? (usually a CA)

Certificates expire

Used for revocation

Github's public key

- Subject: who owns this cert?
- This is Github's certificate
- **Must be served from github.com**

Recover from secret key compromise

- Revocation is <u>very</u> important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to the CA and the CA no longer wishes to certify him
  - CA's certificate has been compromised!
- Methods
  - Certificate expiration
  - Certificate revocation
    - Certificate Revocation Lists (CRL)
    - Online Certificate Status Protocol (OCSP)

Expiration

- Certificate expiration is the simplest, most fundamental defense against secret key compromise
  - All certificates have an expiration date
  - A stolen key is only useful before it expires
- Ideally, all certs should have a short lifetime
  - Months, weeks, or even days
- Problem: most certs have multi-year lifetimes
  - This gives an attacker plenty of time to abuse a stolen key

X.509 Certificate

```
Validity
            Not Before: Apr  8
00:00:00 2014 GMT
            Not After : Apr 12
12:00:00 2016 GMT
```

Revocation

- Certificate revocations are another fundamental mechanism for mitigating secret key compromises
  - After a secret key has been compromised, the owner is supposed to revoke the certificate
- CA's are responsible for hosting databases of revoked certificates that they issued
- Clients are supposed to query the revocation status of all certificates they encounter during validation
  - If a certificate is revoked, the client should never accept it
- Two revocation protocols for TLS certificates
  1. Certificate Revocation Lists (CRLs): download list of revoked certificated
  2. Online Certificate Status Protocol (OCSP): API to query status of certificate

TLS

What Is SSL / TLS?

- Secure Sockets Layer and Transport Layer Security protocols
  - Same protocol design, different crypto algorithms
- De facto standard for Internet security
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
- Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc

SSL / TLS Guarantees

- End-to-end secure communications at transport layer in the presence of a network attacker
  - Attacker completely owns the network: controls Wi-Fi, DNS, routers, his own websites, can listen to any packet, modify packets in transit, inject his own packets into the network
- Properties
  - Authentication of server (optionally, client authentication)
  - Confidentiality of communication
  - Integrity against active attacks

History of the Protocol

- SSL 1.0 – internal Netscape design, early 1994
  - Lost in the mists of time
- SSL 2.0 – Netscape, Nov 1994
  - Several weaknesses
- SSL 3.0 – Netscape and Paul Kocher, Nov 1996
- TLS 1.0 – Internet standard, Jan 1999
  - Supersedes SSL: **SSL is known to be insecure**
  - Based on SSL 3.0, but not interoperable (uses different cryptographic algorithms)
- TLS 1.1 – Apr 2006
- TLS 1.2 – Aug 2008
- TLS 1.3 – Aug. 2018

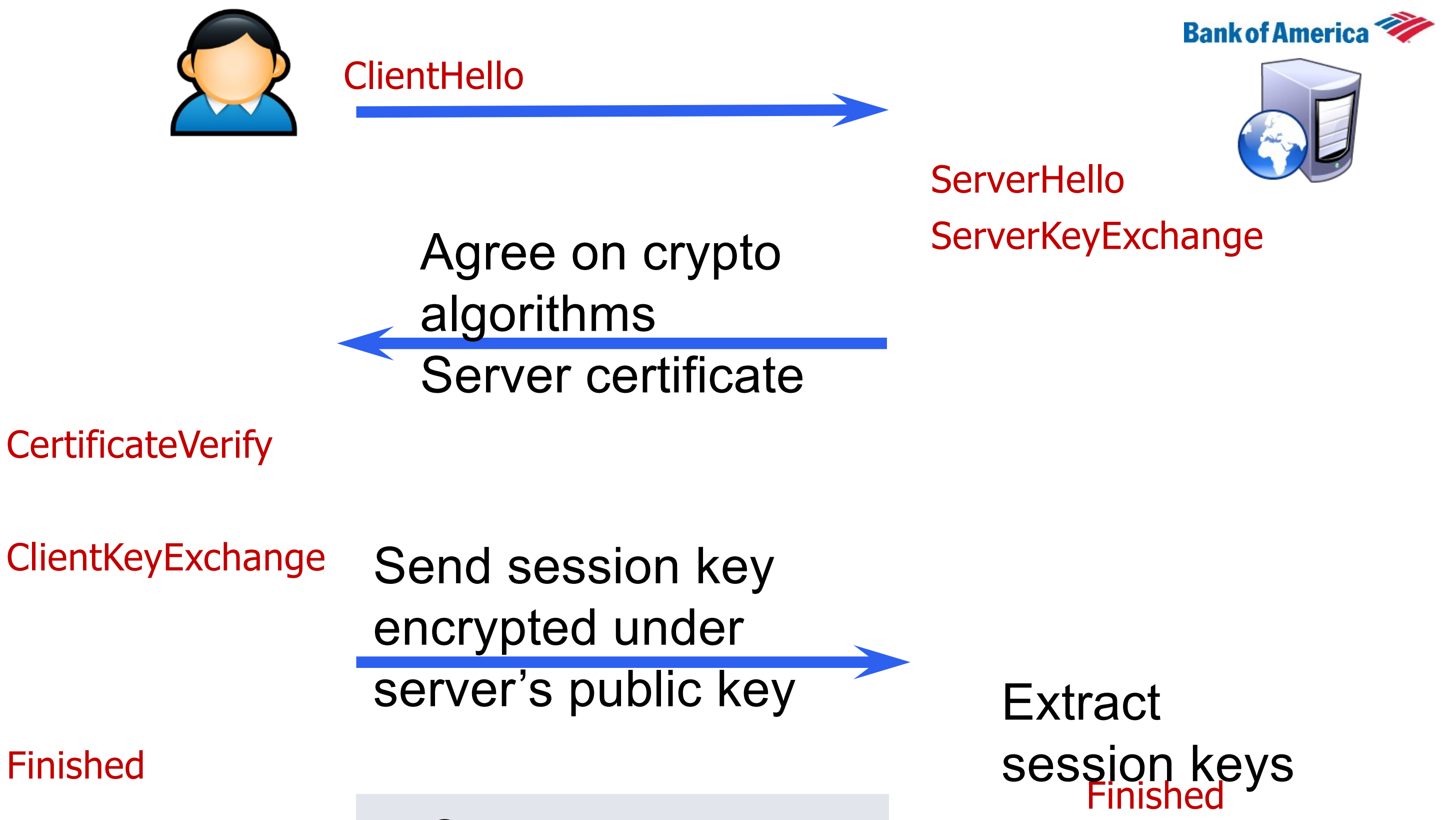
TLS Basics

- TLS consists of two protocols
- Handshake protocol
  - Session initiation by client
  - Uses public-key cryptography to establish several shared secret keys between the client and the server
  - Server must have an asymmetric keypair
    - X.509 certificates contain signed public keys rooted in PKI
- Record protocol
  - Uses the secret keys established in the handshake protocol to protect confidentiality and integrity of data exchange between the client and the server

TLS Handshake Protocol

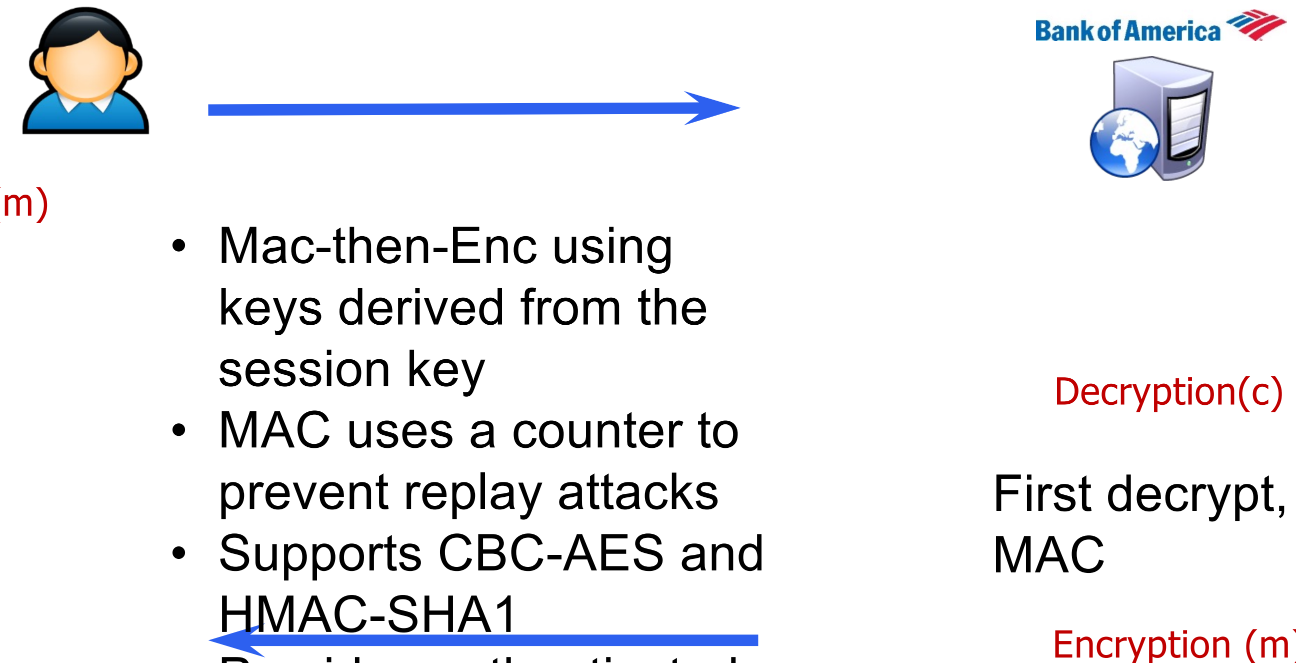
- Runs between a client and a server
  - Client = Web browser
  - Server = website
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations
- Authenticate server
  - Use digital certificates to learn server's public keys and verify the certificate
  - Client authentication is optional
- Use public keys to establish a shared secret

Handshake Protocol Structure



- Common algorithms
- Session key

Record Protocol Structure



Encryption (m)

- Mac-then-Enc using keys derived from the session key
- MAC uses a counter to prevent replay attacks
- Supports CBC-AES and HMAC-SHA1
- Provides authenticated encryption

Decryption(c)

First decrypt, then check MAC

Decryption(c)

First decrypt, then check MAC

Encryption (m)

Similar algorithm, but needs different keys (set of keys for each communication direction)