CY 2550 Foundations of Cybersecurity

Signatures and PKI

February 6

Alina Oprea

Associate Professor, Khoury College

Northeastern University

Outline

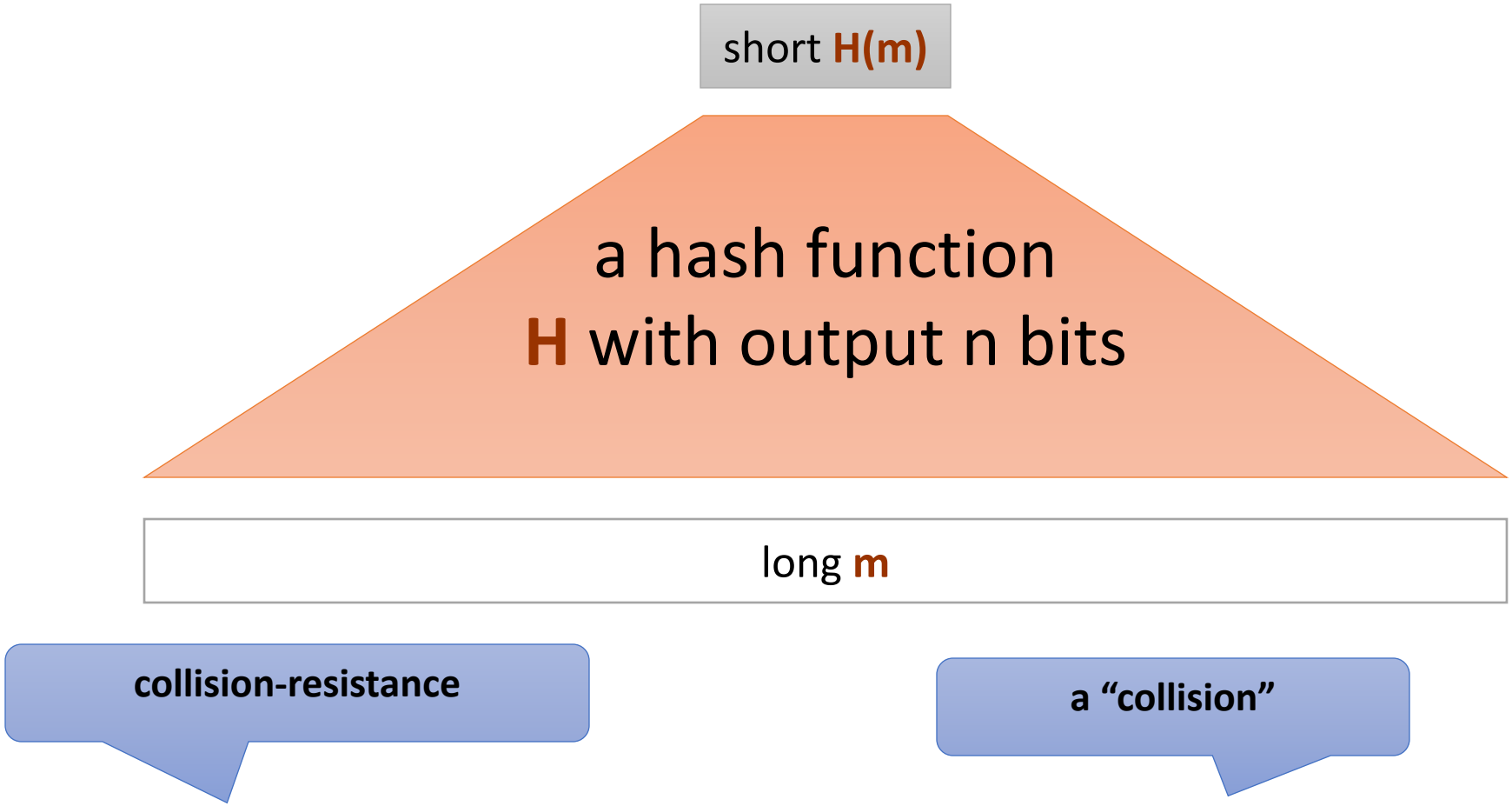
- MACs for integrity
- Digital signatures
- Certificate authorities
- Secure web communication: TLS

- Announcements
  - Distinguished Lecture by Laurel Riek, UCSD, **on Feb 7 in ISEC Auditorium. 11:45am-1:00pm**. “Human Robot Teaming in Healthcare ”

Recap

- Modes of operation for symmetric-key encryption
  - CBC and CTR mode
  - Both are IND-CPA secure
- RSA public-key encryption
  - Textbook RSA is insecure
  - Needs preprocessing for randomization (e.g. OAEP)
- How to exchange a key on untrusted channel
  - Diffie-Hellman key exchange
- In the real world
  - AES key encrypted with public key encryption
  - AES key used for encrypting longer messages for performance reasons

Collision-resistant hash functions



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

H(m) =H(m’)

Hash functions

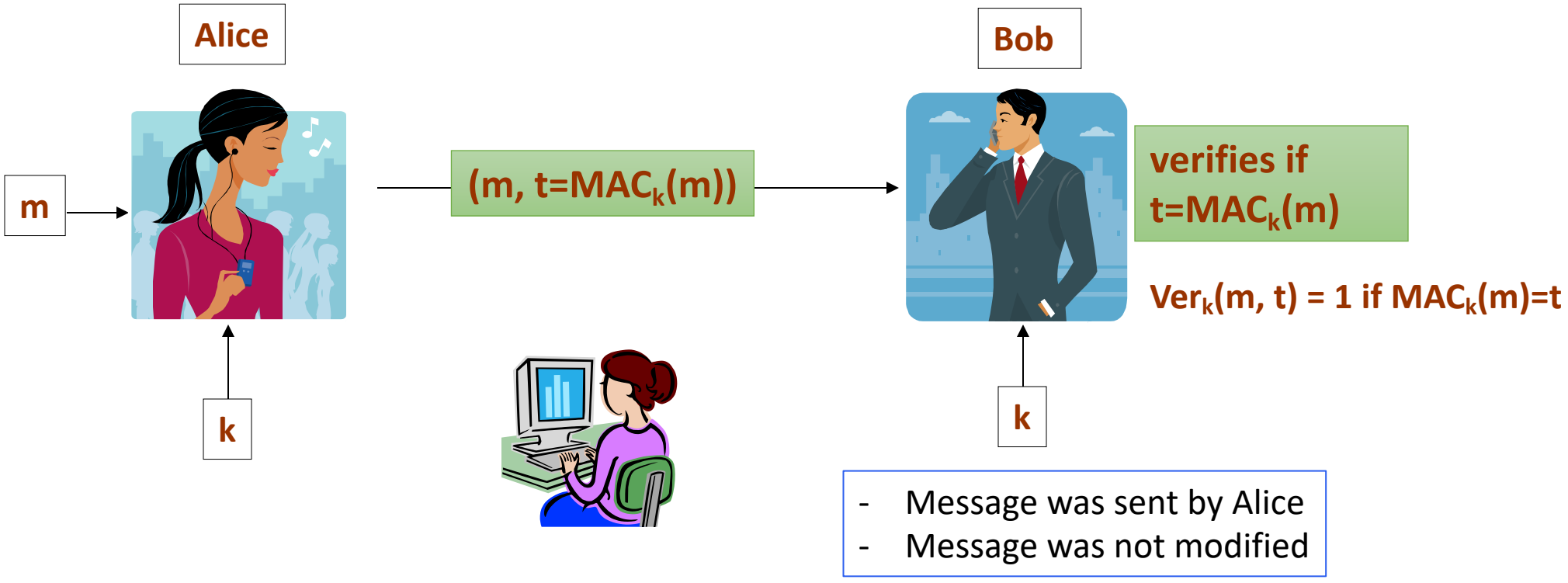
Family name	Year	Output size		Alternate names and notes
		bitlength	bytes	
SHA-3	2015	224, 256	28, 32	SHA3-224, SHA3-256
		384, 512	48, 64	SHA3-384, SHA3-512 (NOTE 1)
SHA-2	2001	256, 512	32, 64	SHA-256, SHA-512
SHA-1	1995	160	20	Deprecated (2017) for browser certificates
MD5	1992	128	16	Widely deprecated, for many applications

Table 2.1: Common hash functions and example parameters. Additional SHA-2 variants include SHA-224 (SHA-256 truncated to 224 bits), SHA-384 (SHA-512 truncated), and further SHA-512 variations, which use specially computed initial values and truncate to 224 or 384 bits resp., giving SHA-512/224 and SHA-512/256. NOTE 1: SHA-3's most flexible variation allows arbitrary bitlength output; SHA-3 is based on the *Keccak* family. For context: Bitcoin computations execute about 2^{90} SHA-2 hashes per year (circa 2019).

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

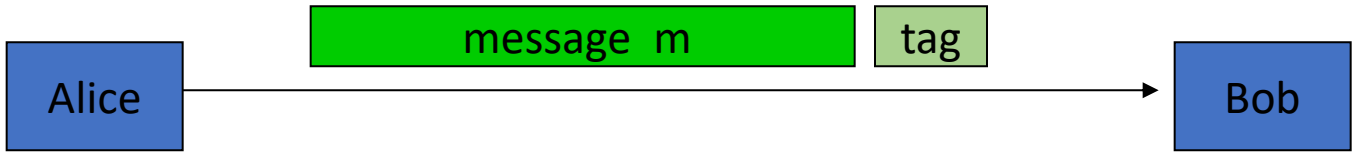
Message Authentication



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



Generate MAC:

$t \leftarrow H(m)$

Verify MAC:

$Ver(m, t) \stackrel{?}{=}$ `yes'

- Attacker can easily modify message m and re-compute the hash.

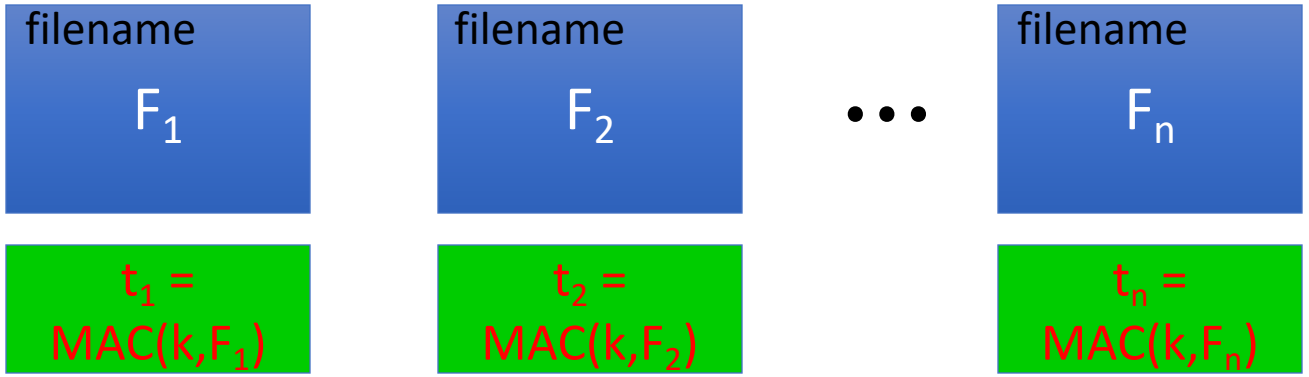
- Hash designed to detect **random**, not malicious errors.

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 $(m_1, t_1), \ldots (m_q, t_q)$ 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:



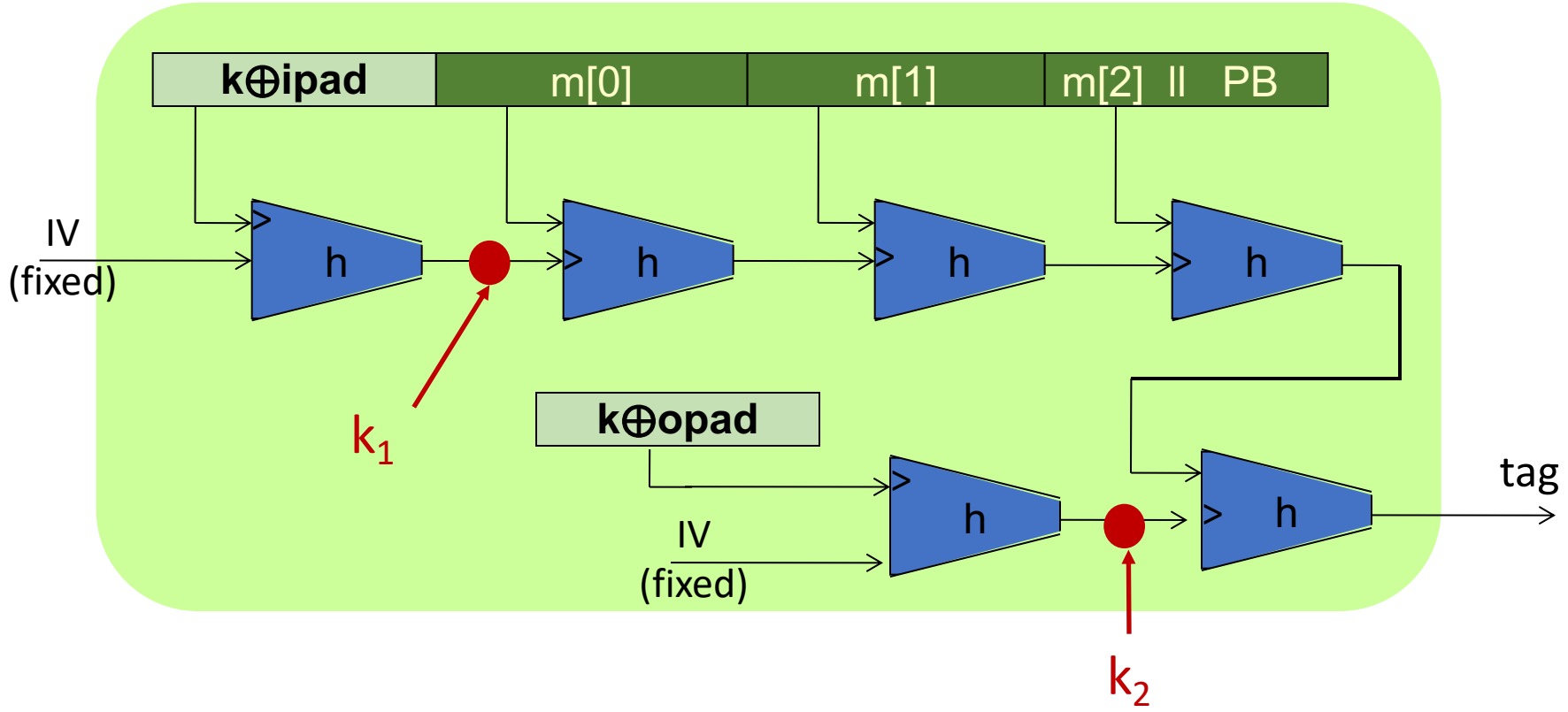
k derived from user's password (e.g., using a hash)

Later malware infects system and modifies system files

User reboots into clean OS and supplies his password

- Then: secure MAC ⇒ all modified files will be detected

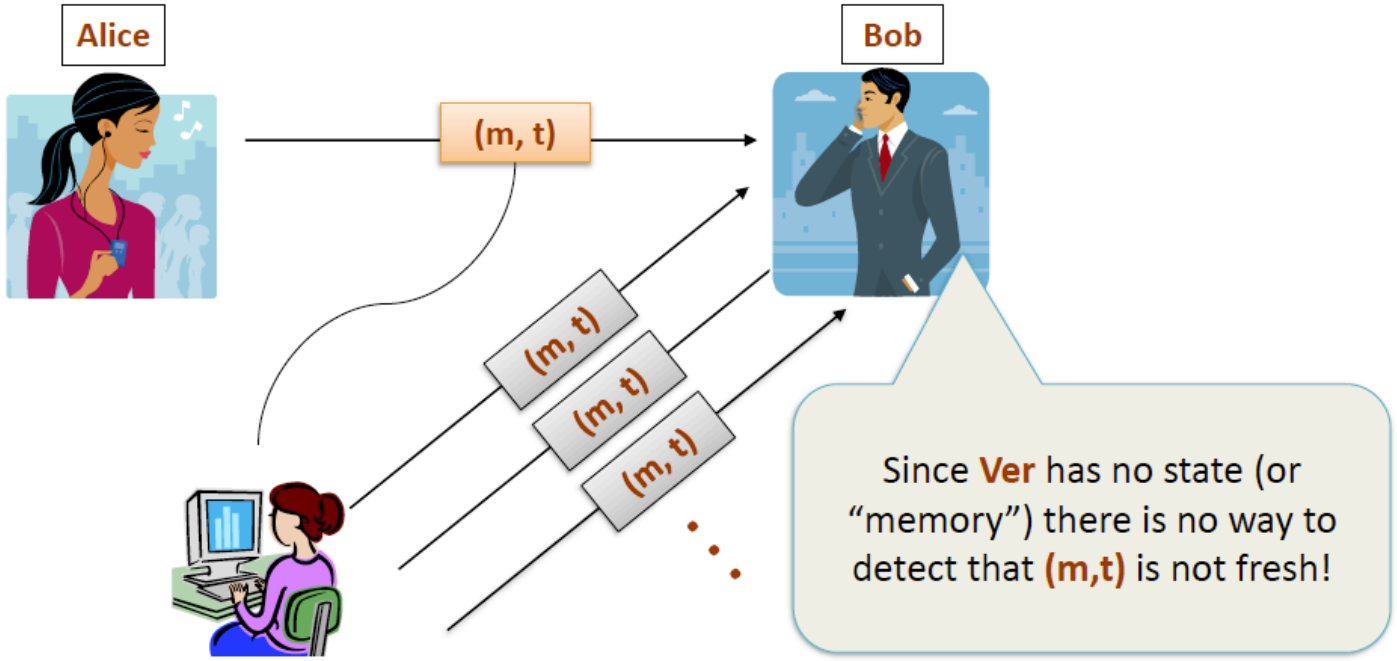
HMAC: Design a MAC from a hash function



- 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".



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)

Some history

Authenticated Encryption (AE): introduced in 2000 [KY'00, BN'00]

Crypto APIs before then: (e.g. MS-CAPI)

- Provide API for CPA-secure encryption (e.g. CBC with rand. IV)
- Provide API for MAC (e.g. HMAC)

Every project had to combine the two itself without a well defined goal

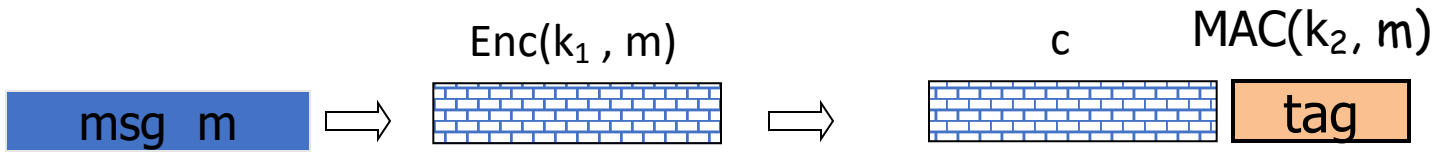
- Not all combinations provide AE …

Authenticated Encryption: Combining MAC and ENC

Encryption key k_1. MAC key = k_2

Option 1: (SSH)

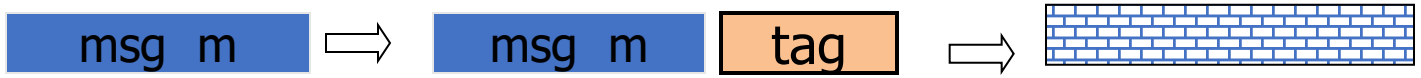
Enc-and-MAC



Homework

Option 2: (SSL)

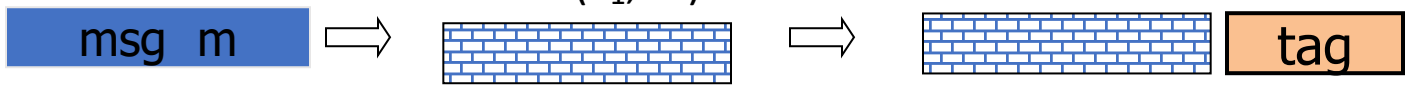
MAC-then-Enc



Padding oracle attacks

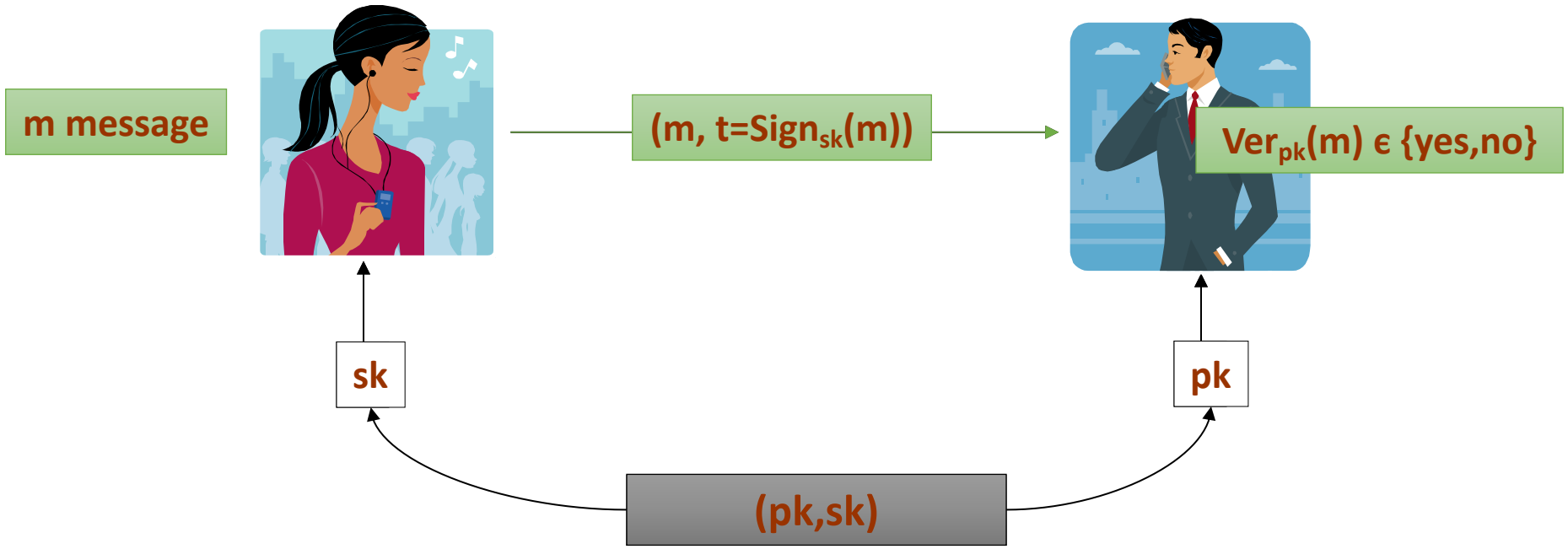
Option 3: (IPsec)

Enc-then-MAC



Always Secure!

Signature Schemes



Public key equivalent of MAC

Digital Signatures

- 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, ϕ(N)) = 1**
d is secret key such that **ed = 1 (mod ϕ(N))**

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

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

Ver is defined as:

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

Hash-and-sign paradigm

Encryption vs. Signatures

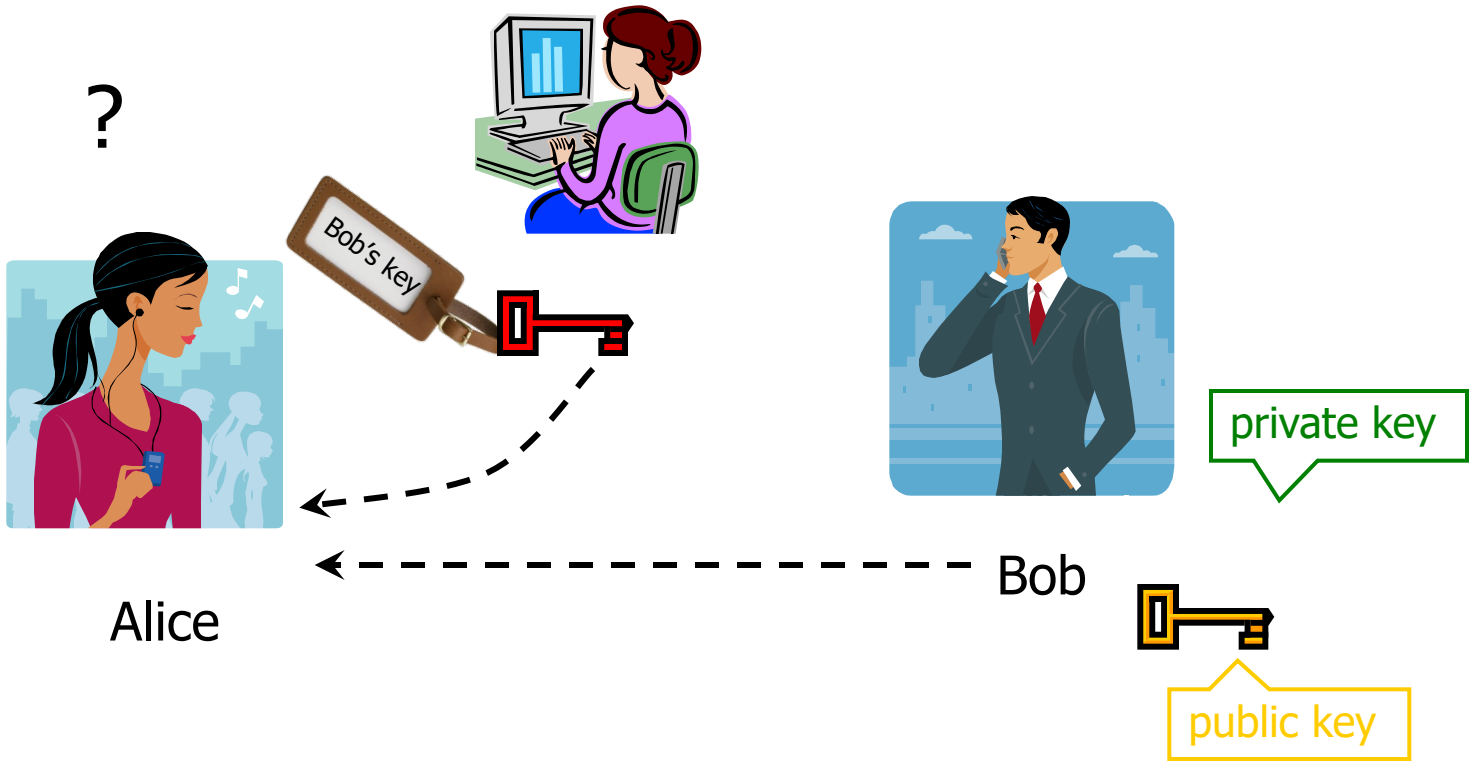
Encryption

- 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

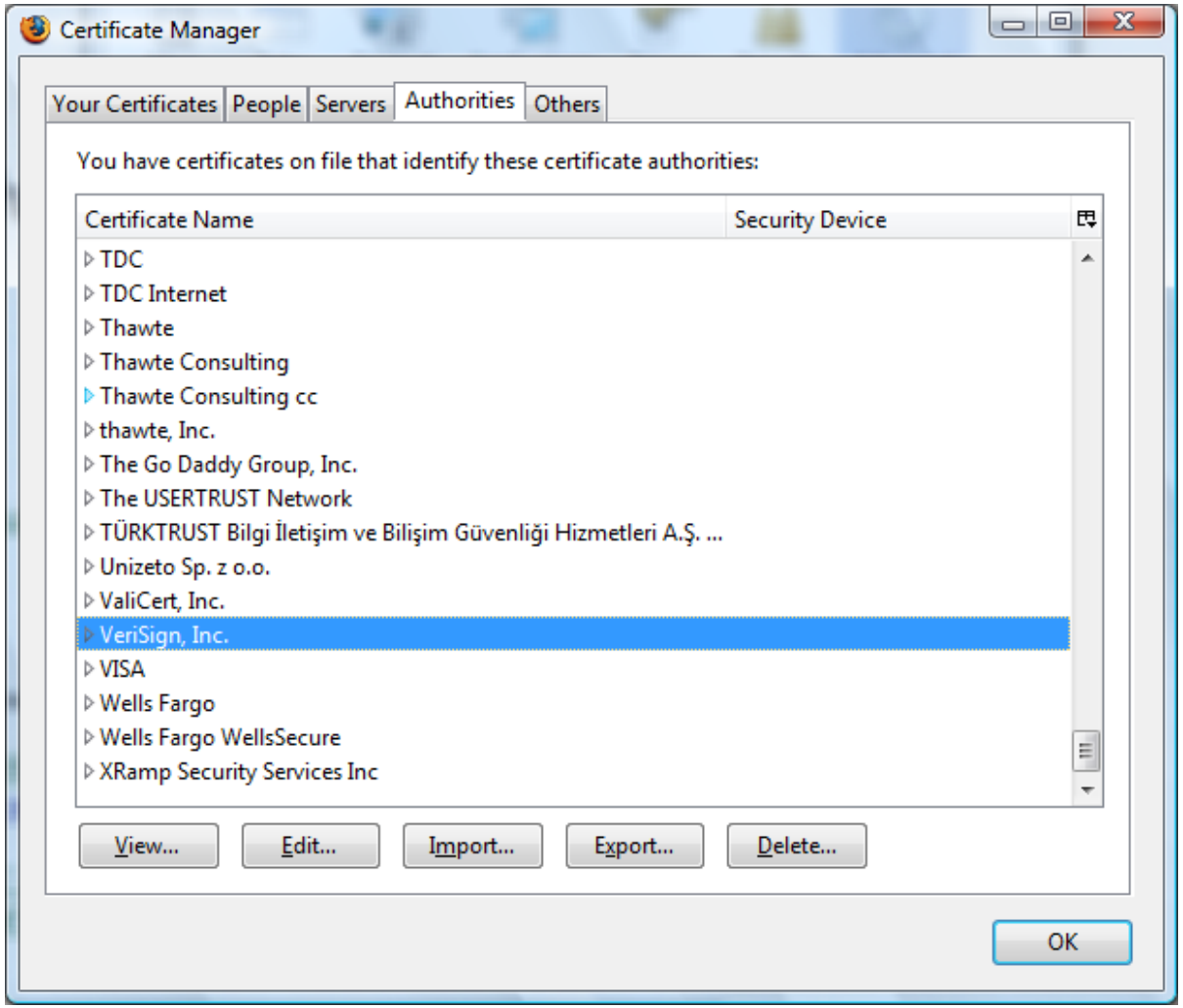


Problem: How does Alice know that the public key she received is really Bob's public key?

PKI: Public Key Infrastructure

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - $Sig_{Charlie}$(“Bob”, PK_{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



Warning

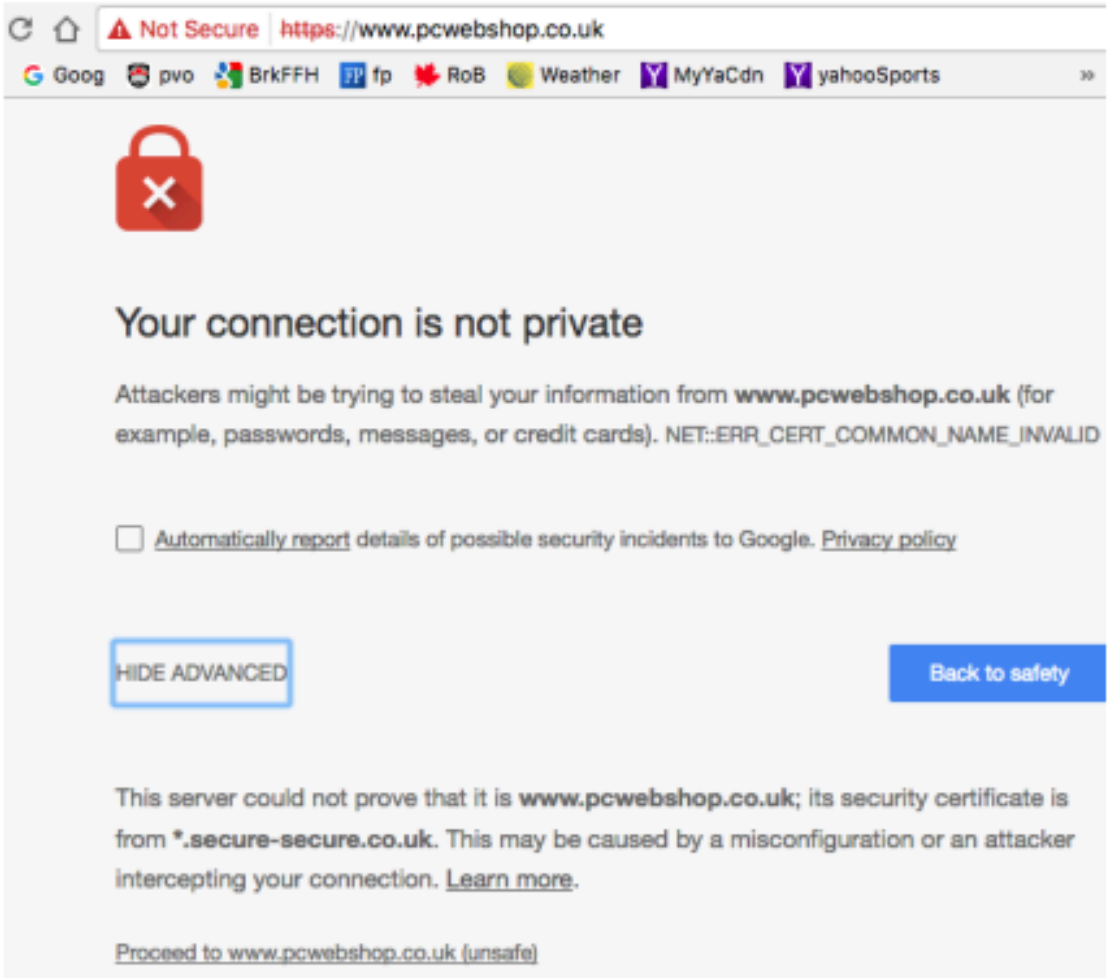


Figure 8.3: Self-signed site certificate—browser warning (Chrome 56.0.2924.87). The "Back to safety" tab discourages clicking-through to the site despite the browser being unable to verify the certificate chain (this happens when one or more certificates are signed by CAs unrecognized by the browser, 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

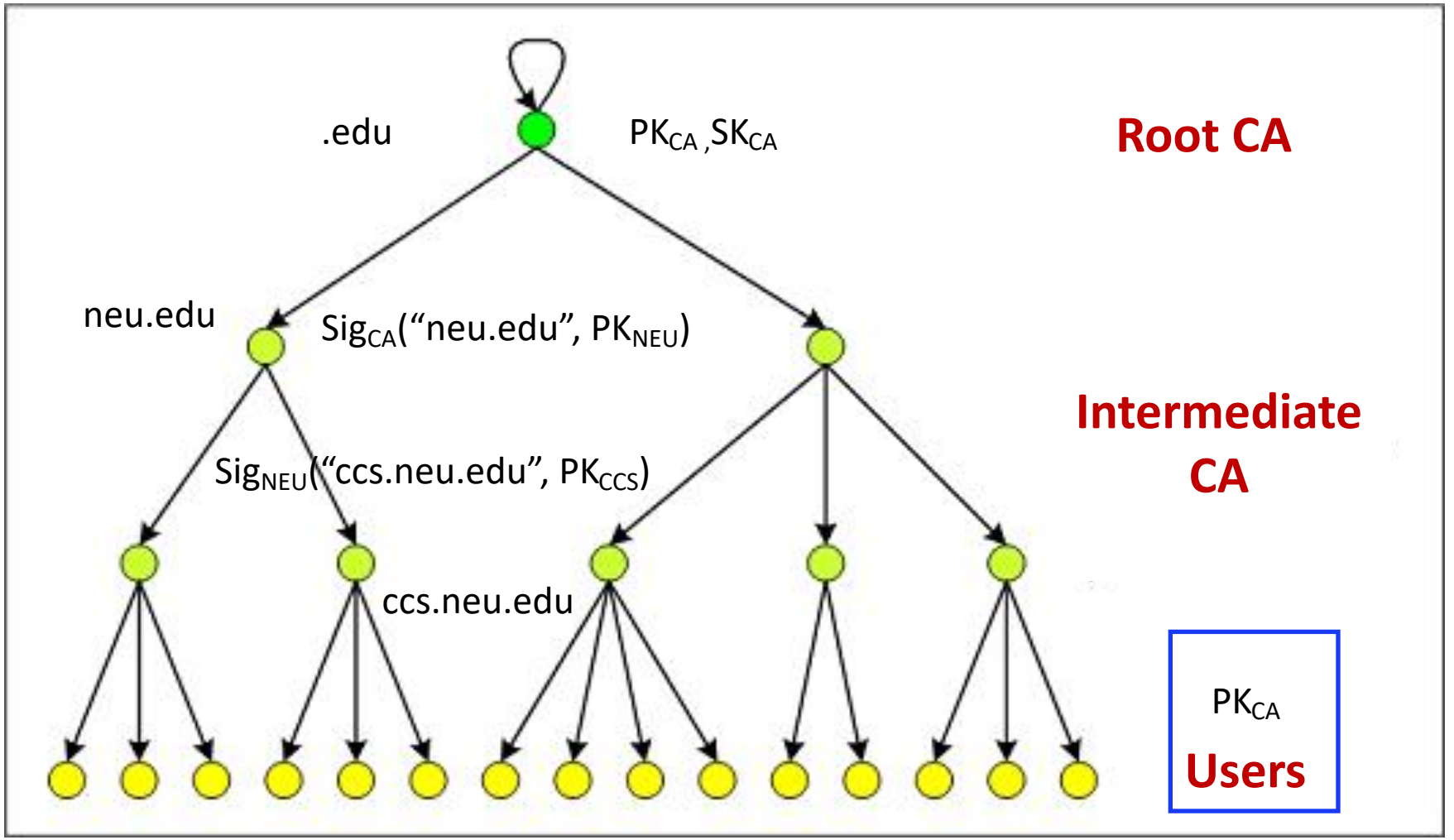
```
1. Hello
2.
3. I'm writing this to the world, so you'll know more about me..
4.
5. At first I want to give some points, so you'll be sure I'm the hacker:
6.
7. I hacked Comodo from InstantSSL.it, their CEO's e-mail address mfpenco@mfpenco.com
8. Their Comodo username/password was: user: gtadmin password: [trimmed]
9. Their DB name was: globaltrust and instantsslcms
```

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

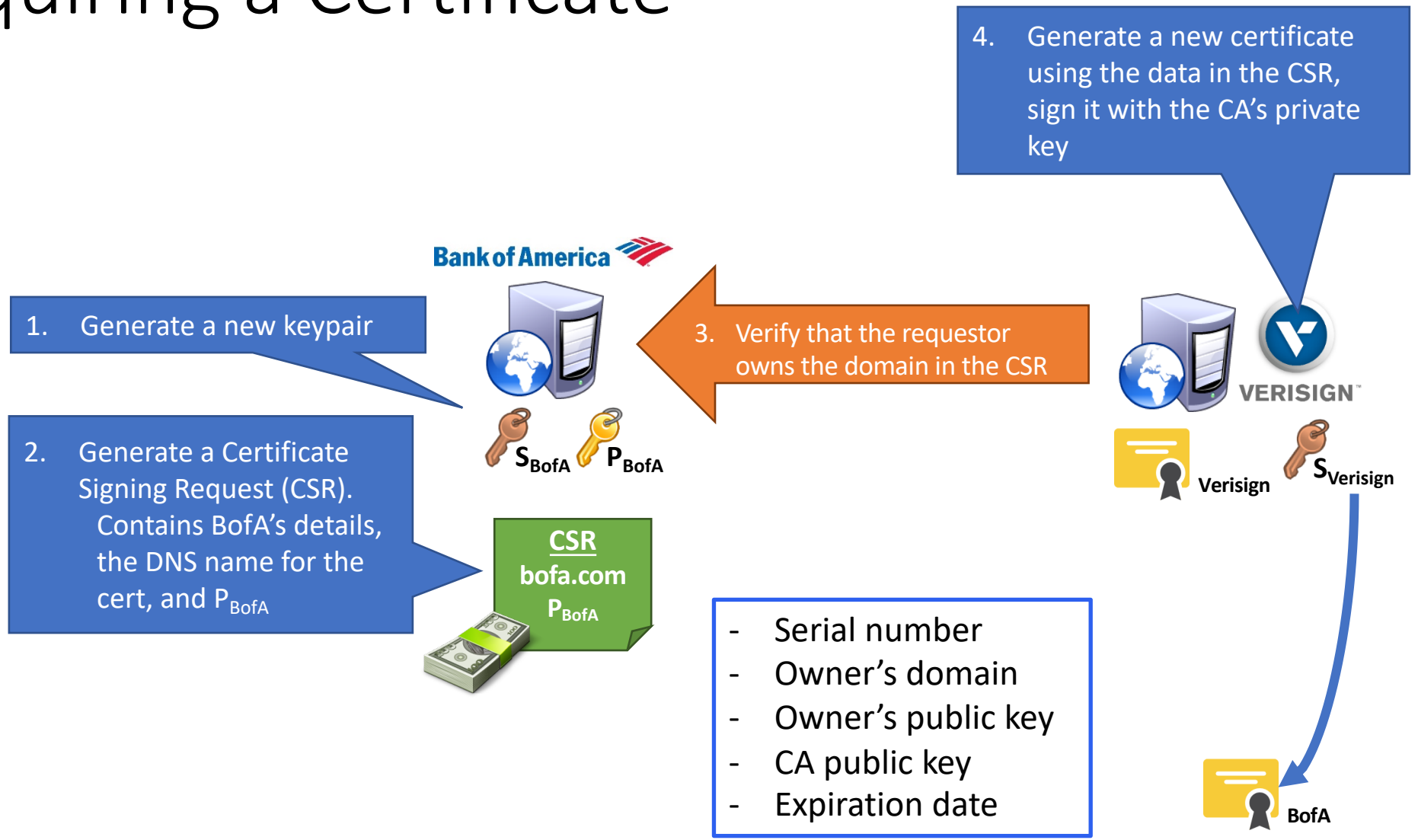
The hack that resulted in Comodo creating certificates for popular e-mail providers including Google Gmail, Yahoo Mail, and Microsoft Hotmail has been claimed as the work of an independent Iranian patriot. A post made to data sharing site pastebin.com by a person going by the handle "comodohacker" claimed 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



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 very 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

Transport Layer Security (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 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