Security and secure systems

COS 518: *Advanced Computer Systems*
Lecture 15

Michael Freedman

Intro to crypto in 10 minutes

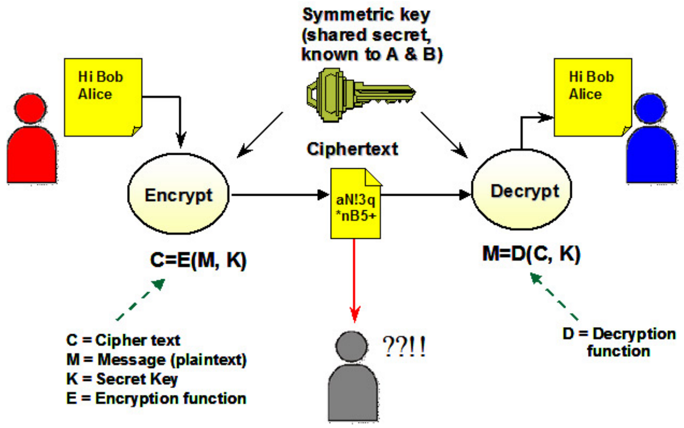
What is Cryptography?

- From Greek, meaning "secret writing"
- Confidentiality: encrypt data to hide content
- Include "signature" or "message authentication code"
  - Integrity: Message has not been modified
  - Authentication: Identify source of message

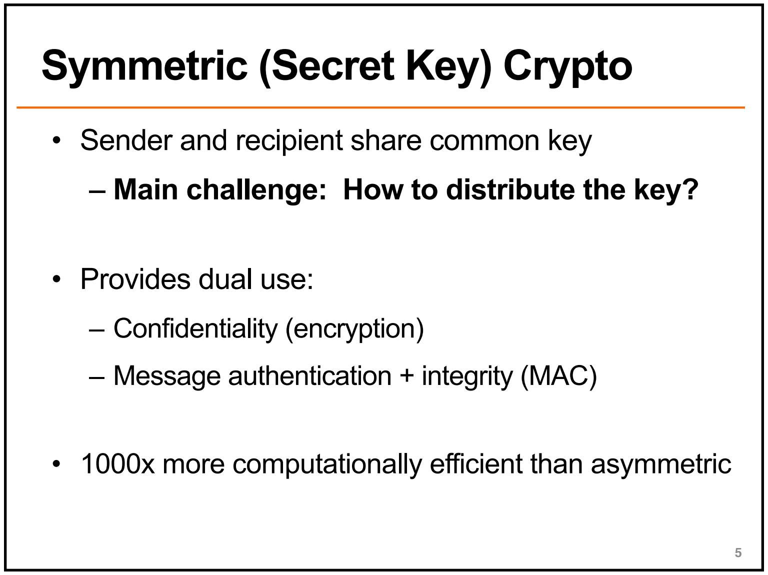
plaintext $\xrightarrow{\text{encryption}}$ ciphertext $\xrightarrow{\text{decryption}}$ plaintext

- Modern encryption:
  - *Algorithm* public, *key* secret and provides security
  - Symmetric (shared secret) or asymmetric (public-private key)

Symmetric Cipher Model

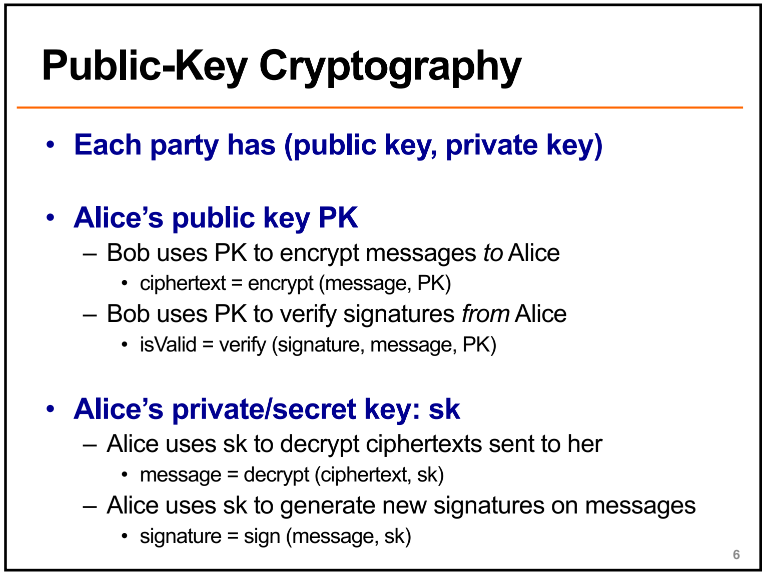


C = Cipher text
M = Message (plaintext)
K = Secret Key
E = Encryption function



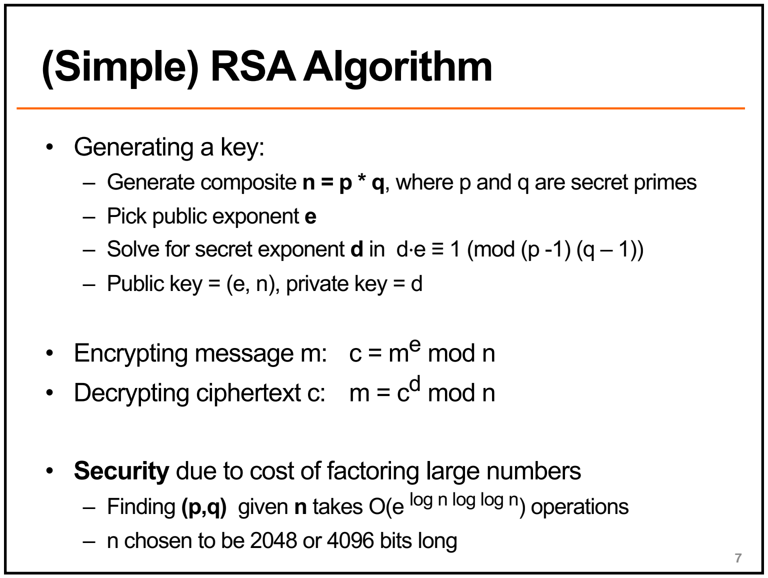
Symmetric (Secret Key) Crypto

- Sender and recipient share common key
  - **Main challenge: How to distribute the key?**
- Provides dual use:
  - Confidentiality (encryption)
  - Message authentication + integrity (MAC)
- 1000x more computationally efficient than asymmetric



Public-Key Cryptography

- **Each party has (public key, private key)**
- **Alice's public key PK**
  - Bob uses PK to encrypt messages *to* Alice
    - ciphertext = encrypt (message, PK)
  - Bob uses PK to verify signatures *from* Alice
    - isValid = verify (signature, message, PK)
- **Alice's private/secret key: sk**
  - Alice uses sk to decrypt ciphertexts sent to her
    - message = decrypt (ciphertext, sk)
  - Alice uses sk to generate new signatures on messages
    - signature = sign (message, sk)



(Simple) RSA Algorithm

- Generating a key:
  - Generate composite **n = p * q**, where p and q are secret primes
  - Pick public exponent **e**
  - Solve for secret exponent **d** in $d \cdot e \equiv 1 \pmod{(p-1)(q-1)}$
  - Public key = (e, n), private key = d
- Encrypting message m: $c = m^e \bmod n$
- Decrypting ciphertext c: $m = c^d \bmod n$
- **Security** due to cost of factoring large numbers
  - Finding **(p,q)** given **n** takes $O(e^{\log n \log \log n})$ operations
  - n chosen to be 2048 or 4096 bits long

Cryptographic hash function

(and using them in systems)

Cryptography Hash Functions I

- Take message m of arbitrary length and produces fixed-size (short) number $H(m)$
- One-way function
  - Efficient: Easy to compute $H(m)$
  - **Hiding property:** Hard to find an m, given $H(m)$
    - Assumes “m” has sufficient entropy, not just {“heads”, “tails”}
  - **Random:** Often assumes for output to “look” random

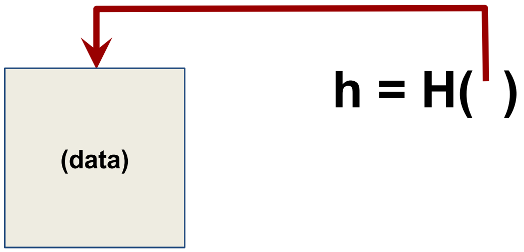
Cryptography Hash Functions II

- Collisions exist: | possible inputs | >> | possible outputs |

  … but hard to find

- Collision resistance:
  - Strong resistance: Find any m != m’ such that H(m) == H(m’)
  - Weak resistance: Given m, find m’ such that H(m) == H(m’)
  - For 160-bit hash (SHA-1)
    - Finding any collision is birthday paradox: 2^{160/2} = 2^80
    - Finding specific collision requires 2^160

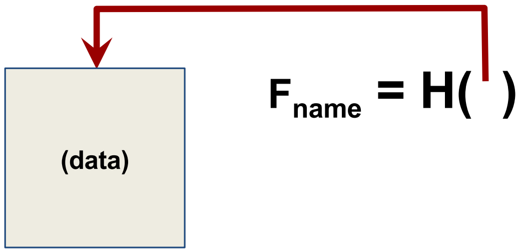
Hash Pointers



(data)

h = H( )

Self-certifying names

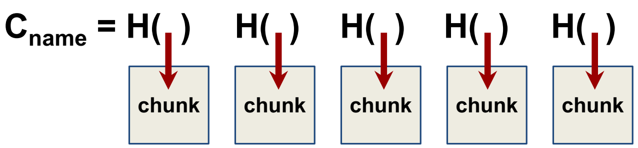


(data)

F_{name} = H( )

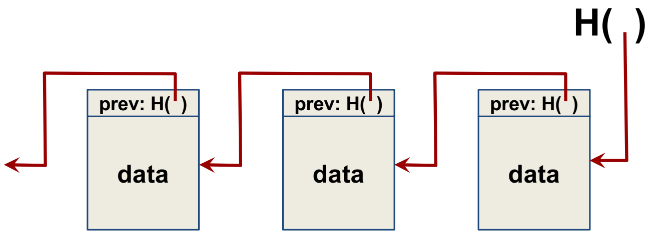
- P2P file sharing software (e.g., Limewire)
  - File named by F_{name} = H (data)
  - Participants verify that H (downloaded) == F_{name}

Self-certifying names



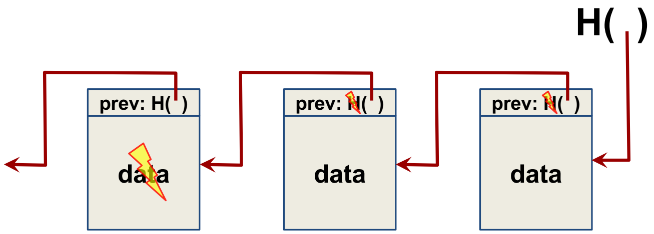
- BitTorrent
  - Large file split into smaller chunks (~256KB each)
  - Torrent file specifies the name/hash of each chunk
  - Participants verify that H (downloaded) == C_{name}
  - Security relies on getting torrent file from trustworthy source

Hash chains



Creates a “tamper-evident” log of data

Hash chains



If data changes, all subsequent hash pointers change

Otherwise, found a hash collision!

Examples:
HTTP and DNS security

“Securing” HTTP

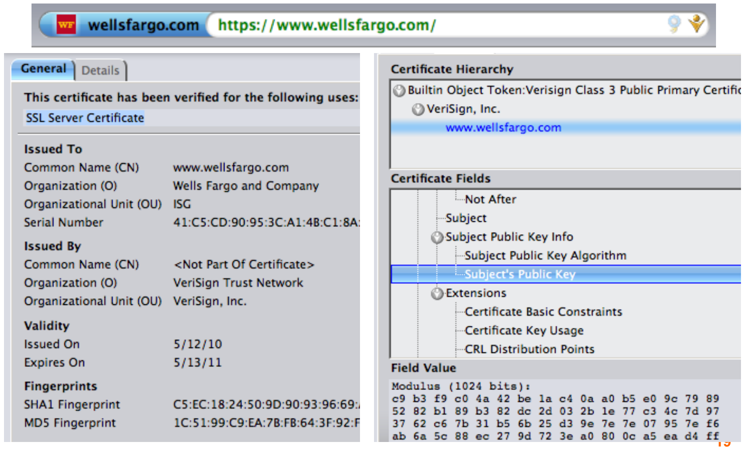
- Threat model
  - Eavesdropper listening on conversation (confidentiality)
  - Man-in-the-middle modifying content (integrity)
  - Adversary impersonating desired website (authentication, and confidentiality)
- Enter HTTP-S
  - HTTP sits on top of secure channel (SSL/TLS)
  - All (HTTP) bytes written to secure channel are encrypted and authenticated
  - Problem: What is actually authenticated to prevent impersonation? Which keys used for crypto protocols?

Learning a valid public key

- What is that lock?
  - Securely binds domain name to public key (PK)
    - Believable only if you trust the attesting body
    - Bootstrapping problem: Who to trust, and how to tell if this message is actually from them?
  - If PK is authenticated, then any message signed by PK cannot be forged by non-authorized party

---

How to authenticate PK



This certificate has been verified for the following uses:

SSL Server Certificate

Issued To	
Common Name (CN)	www.wellsfargo.com
Organization (O)	Wells Fargo and Company
Organizational Unit (OU)	ISG
Serial Number	41:C5:CD:90:95:3C:A1:4B:C1:8A:
Issued By	
Common Name (CN)	<Not Part Of Certificate>
Organization (O)	VeriSign Trust Network
Organizational Unit (OU)	VeriSign, Inc.
Validity	
Issued On	5/12/10
Expires On	5/13/11
Fingerprints	
SHA1 Fingerprint	C5:EC:18:24:50:9D:90:93:96:69:
MD5 Fingerprint	1C:51:99:C9:EA:7B:FB:64:3F:92:F

Certificate Hierarchy

- Builtin Object Token:Verisign Class 3 Public Primary Certific
  - VeriSign, Inc.
    - www.wellsfargo.com

Certificate Fields

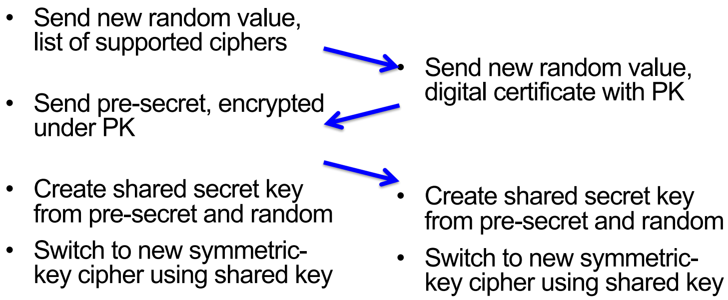
- Not After
- Subject
- Subject Public Key Info
  - Subject Public Key Algorithm
  - Subject's Public Key
- Extensions
  - Certificate Basic Constraints
  - Certificate Key Usage
  - CRL Distribution Points

Field Value

```
Modulus (1024 bits):
c9 b3 f9 c0 4a 42 be 1a c4 0a a0 b5 e0 9c 79 89
52 82 b1 89 b3 82 dc 2d 03 2b 1e 77 c3 4c 7d 97
37 62 c6 7b 31 b5 6b 25 d3 9e 7e 7e 07 95 7e f6
ab 6a 5c 88 ec 27 9d 72 3e a0 80 0c a5 ea d4 ff
```

Transport Layer Security (TLS)

(Enhances/Replaces SSL)



- Send new random value, list of supported ciphers
- Send pre-secret, encrypted under PK
- Create shared secret key from pre-secret and random
- Switch to new symmetric-key cipher using shared key

- Send new random value, digital certificate with PK
- Create shared secret key from pre-secret and random
- Switch to new symmetric-key cipher using shared key

Comments on HTTPS

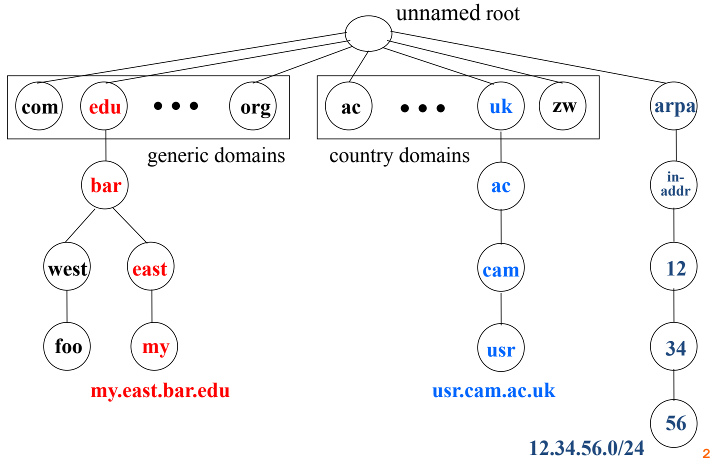
- Note that HTTPS authenticates server, not content
- Switch to symmetric-key crypto after public-key ops
  - Symmetric-key crypto much faster (100-1000x)
  - PK crypto can encrypt message only approx. as large as key (2048 bits – this is a simplification) – afterwards uses hybrid
- HTTPS on top of TCP, so reliable byte stream
  - Can leverage fact that transmission is reliable to ensure: each data segment received exactly once
  - Adversary can't successfully drop or replay packets

The trouble with CAs

- Browse/OS vendors pick which CAs to trust
  - Sometimes they revoke this trust – e.g. DigiNotar.
- No notion of CAs having authority over only given TLD
- Trust the {Iranian, Chinese, US} national authorities?
- What standards does Apple use to pick root certs? Google? MSFT?
  - There's a restraint-of-trade issue here. Can't enter the CA business without vendor support…

DNS Security

Hierarchical naming in DNS



DNS Integrity: Trust the TLD operators?

- If domain name doesn't exist, DNS should return NXDOMAIN (non-existant domain) msg
- Verisign instead creates wildcard DNS record for all .com and .net domain names not yet registered
  - September 15 – October 4, 2003
- Redirection for these domain names to Verisign web portal: "to help you search"
  - and serve you ads…and get "sponsored" search
  - Verisign and online advertising companies make money…

DNS Integrity:
Answer from authoritative server?

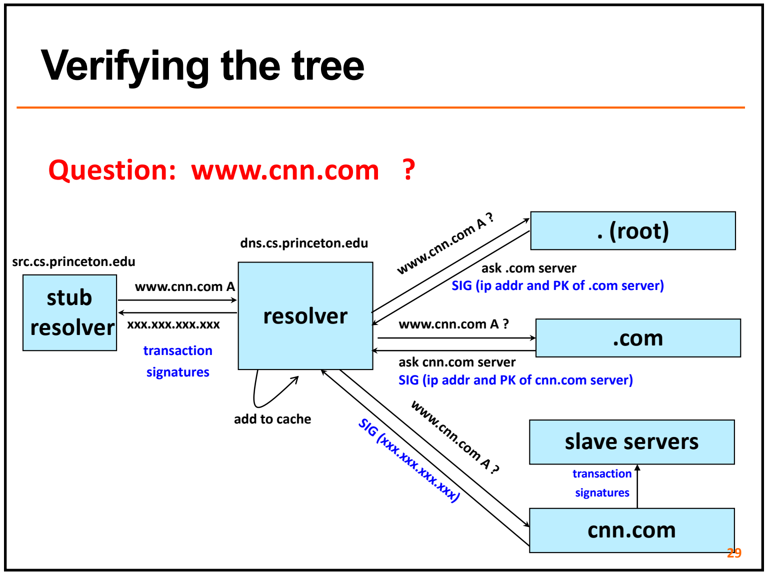
- DNS cache poisoning
  - Client asks for www.evil.com
  - Nameserver authoritative for www.evil.com returns additional section for (www.cnn.com, 1.2.3.4, A)
  - Thanks! I won't bother check what I asked for

DNS Integrity:
Answer from authoritative server?

- To prevent cache poisoning, client remembers domain and 16-bit request ID (used to demux UDP response)
- But…DNS hijacking attack:
  - 16 bits: 65K possible IDs
    - What rate to enumerate all in 1 sec? 64B/packet
    - 64*65536*8 / 1024 / 1024 = 32 Mbps
  - Prevention: Also randomize the DNS source port
    - Windows DNS alloc's 2500 DNS ports: ~164M possible IDs
    - Would require 80 Gbps
    - Kaminsky attack: this source port…wasn't random after all

Let's strongly believe the answer!
Enter DNSSEC

- The DNS servers sign the hash of resource record set with its private (signature) keys
- Public keys can be used to verify the SIGs
- Leverages hierarchy:
  - Authenticity of nameserver's public keys is established by a signature over the keys by the parent's private key
  - In ideal case, only roots' public keys need to be distributed out-of-band



Verifying the tree

Question: www.cnn.com ?

src.cs.princeton.edu

stub resolver

www.cnn.com A

xxx.xxx.xxx.xxx

transaction signatures

dns.cs.princeton.edu

resolver

add to cache

www.cnn.com A ?

. (root)

ask .com server

SIG (ip addr and PK of .com server)

www.cnn.com A ?

.com

ask cnn.com server

SIG (ip addr and PK of cnn.com server)

www.cnn.com A ?

SIG (xxx.xxx.xxx.xxx)

slave servers

transaction signatures

cnn.com