Introduction to Cryptography

Cryptography Introduction: Important Concepts

- Symmetric ciphers
- Public key encryption
- Digital signatures
- Cryptographic hash functions
- Message Authentication Codes (MACs)
- Certificates
- Random numbers
- Key handling

What is a Cryptosystem?

A cryptosystem is pair of algorithms that take a *key* and under control of that key converts *plaintext* to *ciphertext* and back.

Plaintext is what you want to protect; ciphertext should appear to be random gibberish.

The design and analysis of today's cryptographic algorithms is highly mathematical. Do *not* try to design your own algorithms.

Properties of a Good Cryptosystem

- There should be no way short of enumerating all possible keys to find the key from any amount of ciphertext and plaintext, nor any way to produce plaintext from ciphertext without the key.
- Enumerating all possible keys must be infeasible.
- The ciphertext must be indistinguishable from true random values.

Kerckhoffs' Law (1883)

> There must be no need to keep the system secret, and it must be able to fall into enemy hands without inconvenience.

In other words, the security of the system must rest entirely on the secrecy of the key.

Keys

- Must be strongly protected
- Ideally, should be a random set of bits of the appropriate length
- Ideally, each key should be used for a limited time only
- Ensuring that these properties hold is a major goal of cryptographic research and engineering

Cipher Strengths

- A cipher is no stronger than its key length: if there are too few keys, an attacker can enumerate all possible keys
- The old DES cipher has 56 bit keys—arguably too few in 1976; far too few today. (*Deep Crack* was built in 1996 by the EFF.)
- Strength of cipher depends on how long it needs to resist attack.
- No good reason to use less than 128-bit keys
- NSA rates 128-bit AES as good enough for SECRET traffic; 256-bit AES is good enough for TOP-SECRET traffic.
- But a cipher can be considerably weaker! (A monoalphabetic cipher (one that always maps a single character to a fixed ciphertext character) over all possible byte values has 256! keys—a length of 1684 bits—but is trivially solvable.)

Brute-Force Attacks

- Build massively parallel machine
- Can be distributed across the Internet
- Give each processor a set of keys and a plaintext/ciphertext pair
- If no known plaintext, look for probable plaintext (i.e., length fields, high-order bits of ASCII text, etc.)
- On probable hit, check another block and/or do more expensive tests

CPU Speed versus Key Size

- Adding one bit to the key doubles the work factor for brute force attacks
- The effect on encryption time is often negligible or even free
- It costs *nothing* to use a longer RC4 key
- Going from 128-bit AES to 256-bit AES takes (at most) 40% longer in CPU time, but increases the attacker's effort by a factor of 2^{128}
- Using triple DES costs $3\times$ more than DES to encrypt, but increases the attacker's effort by a factor of 2^{112}
- Moore's Law favors the defender

Block Ciphers

- Operate on a fixed-length set of bits
- Output blocksize generally the same as input blocksize
- Well-known examples: DES (56-bit keys; 64-bit blocksize); AES (128-, 192-, and 256-bit keys; 128-bit blocksize)

Stream Ciphers

- Key stream generator produces a sequence S of pseudo-random bytes; key stream bytes are combined (generally via XOR) with plaintext bytes: $P_i \oplus S_i \rightarrow C_i$
- Stream ciphers are very good for asynchronous traffic
- Best-known stream cipher is RC4; commonly used with SSL. (RC4 is now considered insecure.)
- Key stream S must *never* be reused for different plaintexts:

$$\begin{aligned} C &= A \oplus K \\ C' &= B \oplus K \\ C \oplus C' &= A \oplus K \oplus B \oplus K \\ &= A \oplus B \end{aligned}$$

- Guess at A and see if B makes sense; repeat for subsequent bytes

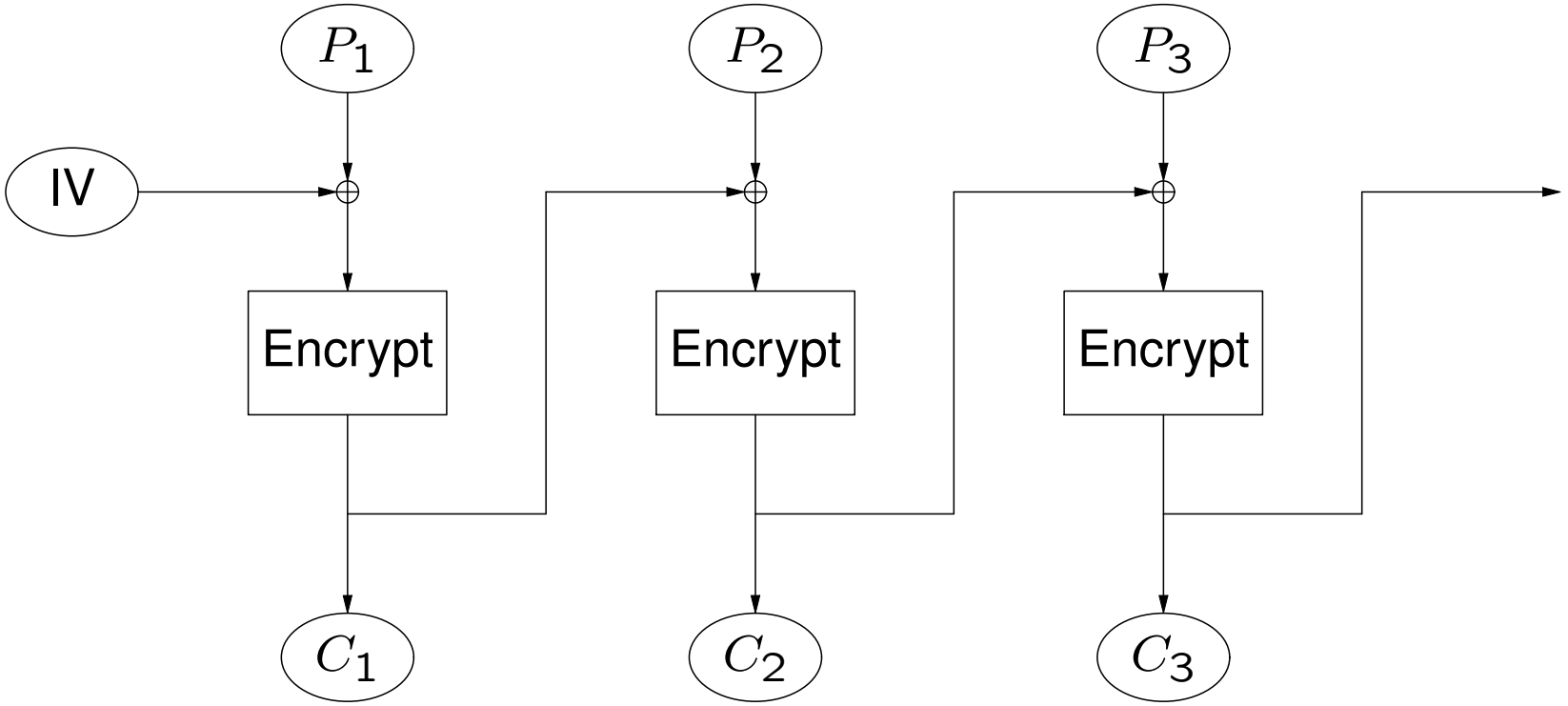
Basic Structure of (Most) Block Ciphers

- Optional key scheduling—convert supplied key to internal form
- Multiple *rounds* of combining the plaintext with the key.
- DES has 16 rounds; AES has 9-13 rounds, depending on key length

Modes of Operation

- Direct use of a block cipher is almost always wrong
- Enemy can build up “code book” of plaintext/ciphertext equivalents
- Beyond that, direct use only works on messages that are a multiple of the cipher block size in length
- Solution: several standard *Modes of Operation*, including Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR). (Some newr modes provide authentication as well as confidentiality.)
- All modes of operation except ECB require an extra block known as the *Initialization Vector* (IV). IVs *must* be unpredictable by the enemy.

Example: Cipher Block Chaining



$$\{P_i \oplus C_{i-1}\}_k \rightarrow C_i$$
$$\{C_i\}_{k^{-1}} \oplus C_{i-1} \rightarrow P_i$$

Things to Notice About CBC

- Identical plaintext blocks do not, in general, produce the same ciphertext. (Why?)
- Each ciphertext block is a function of all previous plaintext blocks. (Why?)
- The converse is not true, but we won't go into that in this class

Alice and Bob

- Alice wants to communicate security with Bob
- (Cryptographers frequently speak of Alice and Bob instead of A and B...)
- What key should she use?

Pre-Arranged Key Lists?

- A fixed key? Encrypting too much data with a single key is dangerous
- What if you run out of keys?
- What if a key is stolen?

  "Why is it necessary to destroy yesterday's [key] . . . list if it's never going to be used again?"

  "A used key, Your Honor, is the most critical key there is. If anyone can gain access to that, they can read your communications."

  (trial of Jerry Whitworth, a convicted spy.)

- What if Alice doesn't know in advance that she'll want to talk to Bob?

The Solution: Public Key Cryptography

- Allows parties to communicate without prearrangement
- Separate keys (K and K^{-1}) for encryption and decryption
- Not possible to derive decryption key from encryption key
- Permissible to publish encryption key, so that anyone can send you secret messages
- All known public key systems are very expensive to use, in CPU time and bandwidth.
- Most public systems are based on mathematical problems.

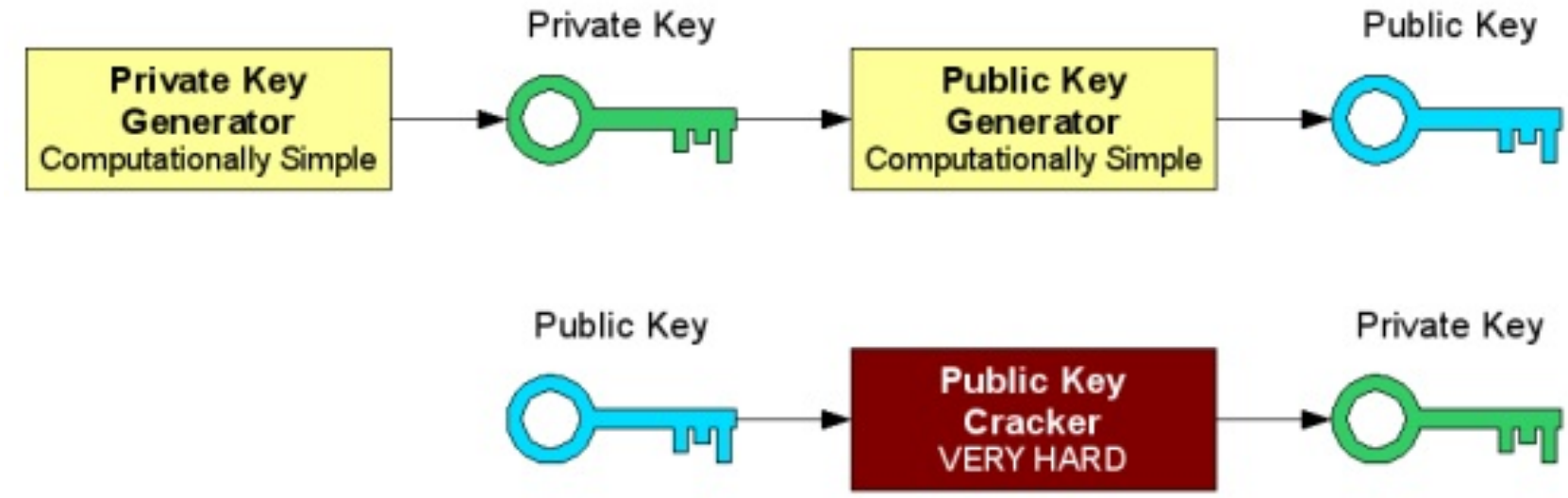
Different Keys

Private key Known only to the owner

Public key Known to the world

Relationship Given the private key, it's easy to calculate the public key.
Given just the public key, it is infeasible to calculate the private key

Key Generation



Stolen from `http://gdp.globus.org/gt4-tutorial/multiplehtml/ch09s03.html`

Mathematical Background

- A *prime number* is one divisible only by 1 and itself
- $x \bmod y$ (pronounced "x modulo y") is the remainder when x is divided by y
- ☞ $2 \equiv 23 \bmod 7$: "2 is equivalent to the remainder of 23 divided by 7"
- Most public key systems rely on modular arithmetic and very large prime numbers

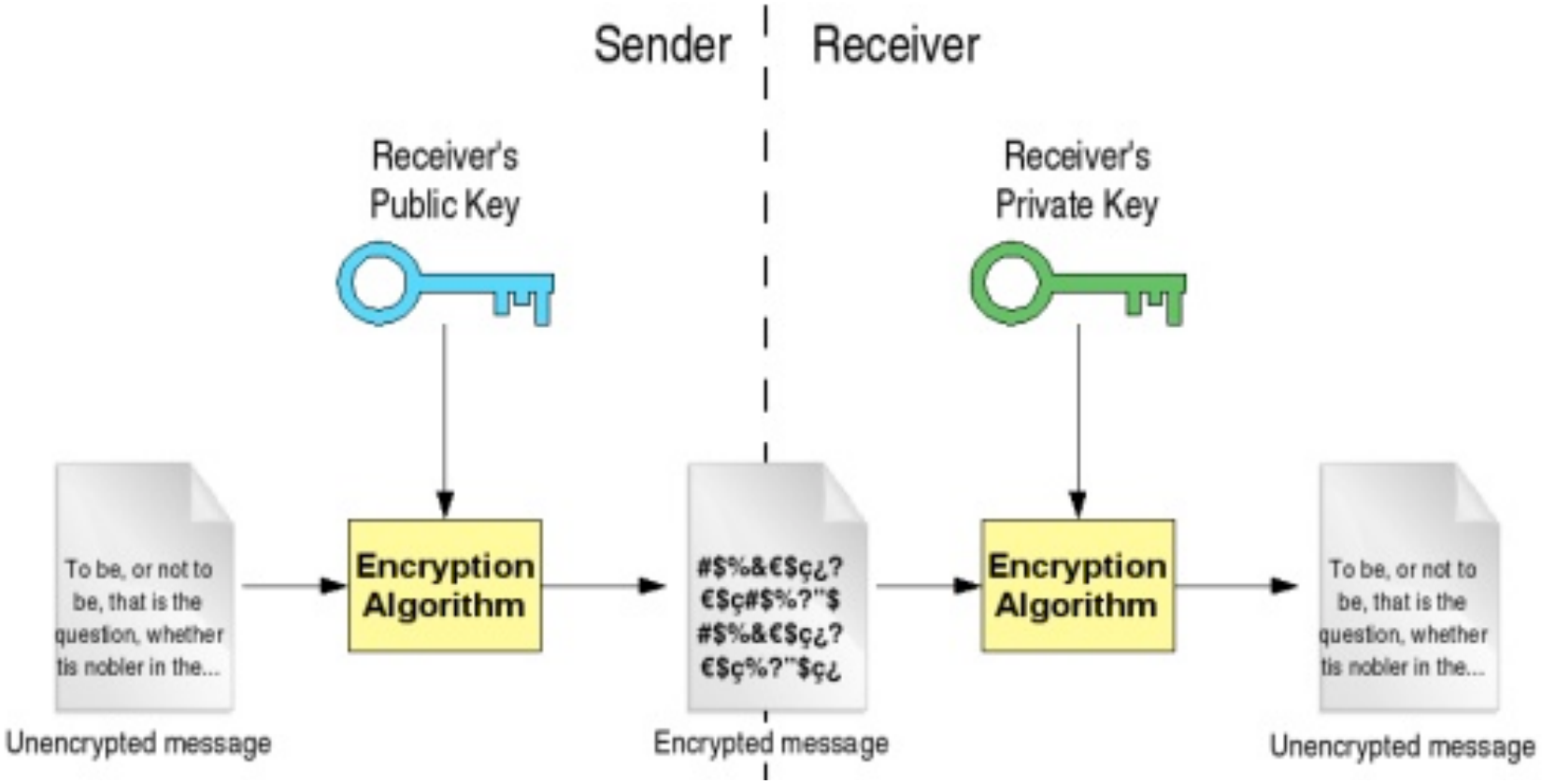
RSA

- The best-known public key system is RSA (Rivest, Shamir, Adleman)
- Generate two large (at least 1024-bit) primes p and q; let $n = pq$
- Pick two integers e and d such that $ed \equiv 1 \mod (p-1)(q-1)$. Often, $e = 65537$, since that simplifies encryption calculations. (Older systems use $e = 3$, but that's no longer recommended.)
- The public key is $\langle e, n \rangle$; the private key is the pair $\langle d, n \rangle$.
- To encrypt m, calculate $c = m^e \mod n$; to decrypt c, calculate $m = c^d \mod n$.
- The security of the system (probably) relies on the difficulty of factoring n.
- Finding such large primes is relatively easy; factoring n is believed to be extremely hard.

Classical Public Key Usage

- Alice publishes her public key in the phone book.
- Bob prepares a message and encrypts it with that key by doing a large exponentiation.
- Alice uses her private key to do a different large exponentiation.
- It's not that simple—more in a few minutes. . .

Public Key Encryption



Complexities

- RSA calculations are *very* expensive; neither Bob nor Alice can afford to do many.
- RSA is too amenable to mathematical attacks; encrypting the wrong numbers is a bad idea.
- Example: "yes"^{3} is only 69 bits, and won't be reduced by the modulus operation; finding $\sqrt[3]{5035655279015561942 83}$ is easy.
- We need a better solution

A (More) Realistic Scenario

- Bob generates a random key k for a conventional cipher.
- Bob encrypts the message: $c = \{m\}_k$.
- Bob pads k with a known amount of padding, to make it at least 512 bits long; call this k'.
- k' is encrypted with Alice's public key $\langle e, n \rangle$.
- Bob transmits $\{c, (k')^e \bmod n\}$ to Alice.
- Alice uses $\langle d, n \rangle$ to recover k', removes the padding, and uses k to decrypt ciphertext c.
- In reality, it's even more complex than that...

Who Sent a Message?

- When Bob receives a message from Alice, how does he know who sent it?
- With traditional, symmetric ciphers, he may know that Alice has the only other copy of the key; with public key, he doesn't even know that
- Even if he knows, can he prove to a third party—say, a judge—that Alice sent a particular message?

Digital Signatures

- RSA can be used backwards: you can encrypt with the private key, and decrypt with the public key.
- This is a *digital signature*: only Alice can sign her messages, but anyone can verify that the message came from Alice, by using her public key
- It's too expensive to sign the whole message. Instead, Alice calculates a *cryptographic hash* of the message and signs the hash value.
- If you sign the plaintext and encrypt the signature, the signer's identity is concealed; if you sign the ciphertext, a gateway can verify the signature without having to decrypt the message.

They're Not Like Real Signatures

- Real signatures are strongly bound to the person, and weakly bound to the data
- Digital signatures are strongly bound to the data, and weakly bound to the person—what if the key is stolen (or deliberately leaked)?
- A better term: digital signature algorithms provide *non-repudiation*

Cryptographic Hash Functions

- Produce relatively-short, fixed-length output string from arbitrarily long input.
- Computationally infeasible to find two different input strings that hash to the same value (“collision”)
- Computationally infeasible to find any input string that hashes to a given value (“pre-image”)
- Computationally infeasible to find any input string that hashes to the same value as the hash of a given input (“second preimage”)
- Strength roughly equal to half the output length
- This means that you want a hash function whose output is at least 160 bits and probably at least 256 bits

Common Hash Functions

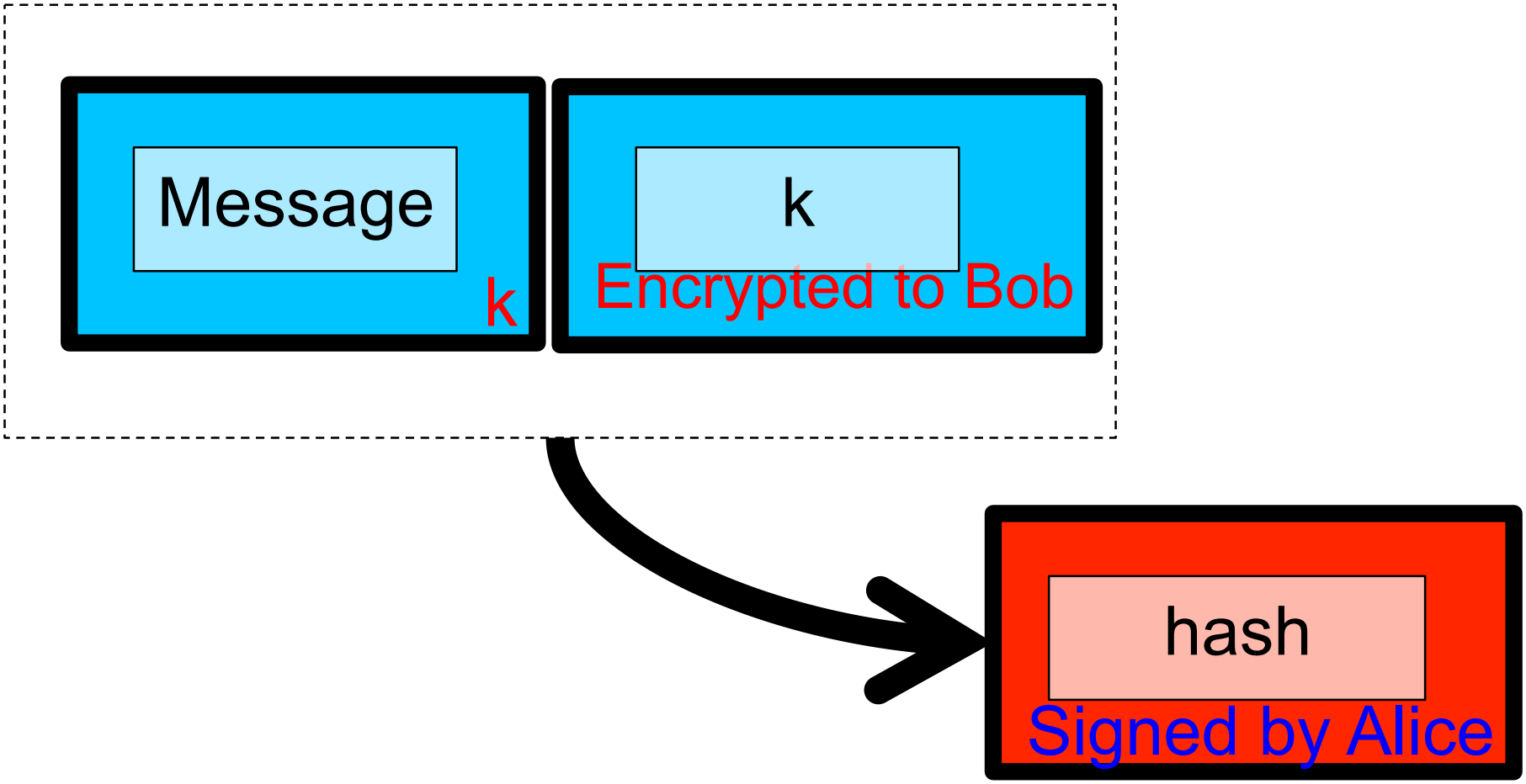
- Best-known cryptographic hash functions: MD5 (128 bits), SHA-1 (160 bits), SHA2-256/384/512 (256/384/512 bits)
- Wang et al. have found collision attacks against MD5 and SHA-1
- ☞ Never use MD5; SHA-1 is being phased out (Microsoft will stop accepting it in about 2.5 years)
- SHA2-256/384/512 have the same basic structure as MD5 and SHA-1—but NIST now believes they're secure
- NIST held a design competition for a new SHA-3 hash function; the winner (Keccak) has a completely different structure

Sending a Signed Message

- Optionally encrypt the message with a new random key k
- Encrypt k with the recipient's public key
- Hash the encrypted message
- Digitally sign the hash using the sender's private key
- The full message from Alice to Bob:

$$\boxed{\{m\}_k}, \boxed{\{k\}_{K_B}}, \boxed{\{\boxed{\mathrm{H}(\{m\}_k, \{k\}_{K_B})}\}_{K_A{}^{-1}}}$$

A Signed Message from Alice to Bob



In Detail. . .

$$\{m\}_k,\ \{k\}_{K_B},\ \{\mathsf{H}(\{m\}_k,\ \{k\}_{K_B})\}_{K_A{}^{-1}}$$

$\{m\}_k$ Message m encrypted with random key k

$\{k\}_{K_B}$ k encrypted with Bob's public key, with all the usual padding

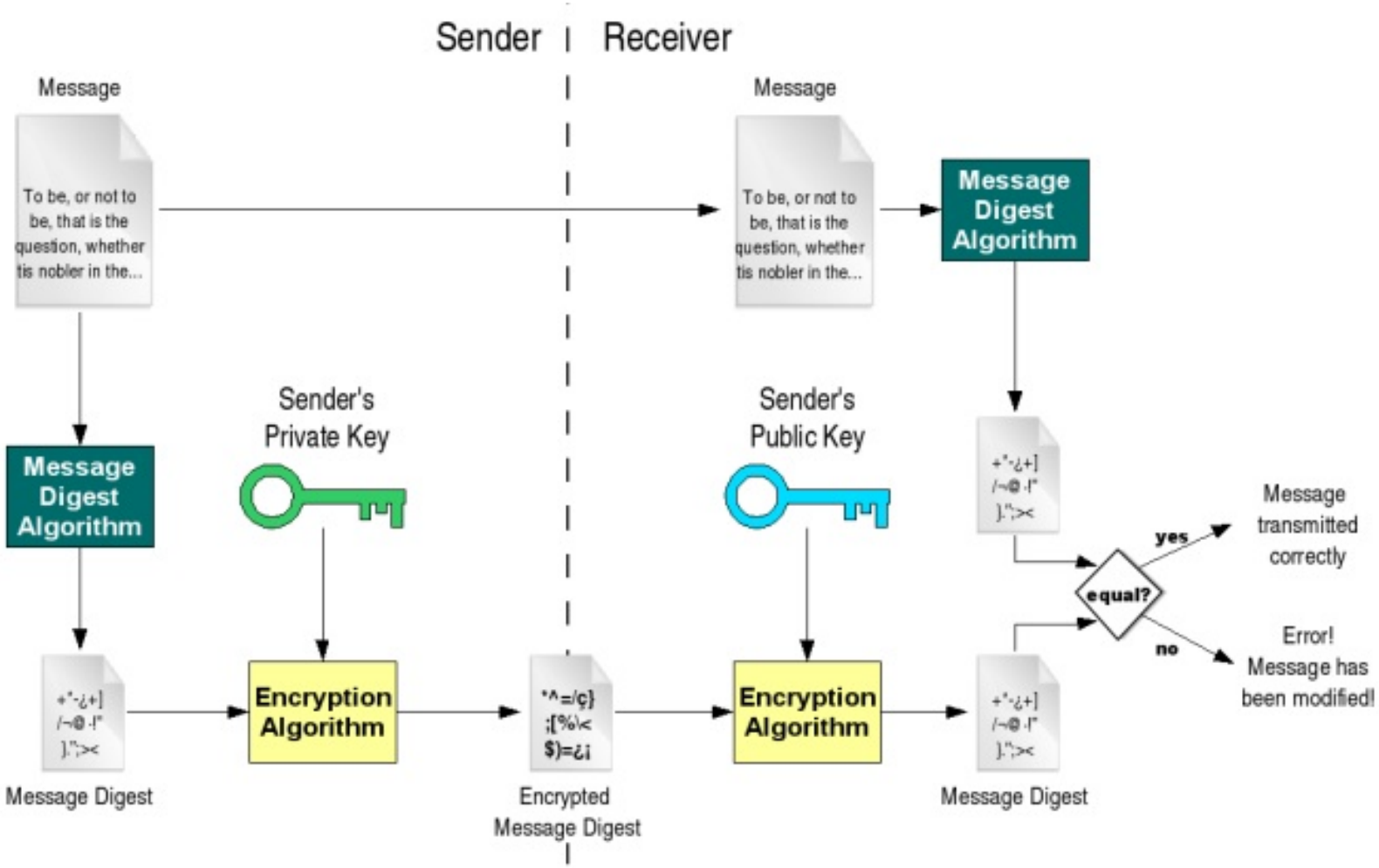
$\mathsf{H}(\{m\}_k,\ \{k\}_{K_B})$ The hash of the previous two parts

$\{\mathsf{H}(\{m\}_k,\ \{k\}_{K_B})\}_{K_A{}^{-1}}$ The hash digitally signed by Alice's private key

Receiving a Signed Message

- Hash the received message
- Use the sender's public key to verify that the signature is correct
- Use the recipient's private key to recover k
- Use k to decrypt the message

A Digitally Signed Message



The Birthday Paradox

- How many people need to be in a room for the probability that two will have the same birthday to be $> .5$?
- Naive answer: 183
- Correct answer: 23
- The question is not "who has the same birthday as Alice?"; it's "who has the same birthday as Alice or Bob or Carol or . . . " assuming that none of them have the same birthday as any of the others

The Birthday Attack

- Alice can prepare lots of variant contracts, looking for any two that have the same hash
- More precisely, she generates many trivial variants on m and m', looking for a match between the two sets
- This is much easier than finding a contract that has the same hash as a given other contract
- As a consequence, the strength of a hash function against brute force attacks is approximately half the output block size: 64 bits for MD5, 80 bits for SHA-1, etc.

Message Integrity

- We need a way to prevent tampering with messages
- We can use a key and a cryptographic hash to generate a *Message Authentication Code* (MAC).
- Simpler solutions don't work
- One bad idea: append a cryptographic hash to some plaintext, and encrypt the whole thing with, say, CBC mode

$$\{P, H(P)\}_K$$

- This can fall victim to a *chosen plaintext attack*

HMAC

- Build a MAC from a cryptographic hash function
- Best-known construct is HMAC—provably secure under minimal assumptions
- HMAC$(m, k) = H($opad $\oplus k, H($ipad $\oplus k, m))$ where H is a cryptographic hash function
- Note: authentication key *must* be distinct from the confidentiality key
- Frequently, the output of HMAC is truncated

Cryptography and Authentication

- Some way to use a cryptographic key to prove who you are
- Can go beyond simple schemes given above
- Can use symmetric or public key schemes
- Most public key schemes use *certificates*

What are Certificates

- How does Alice get Bob's public key?
- What if the enemy tampers with the phone book? Sends the phone company a false change-of-key notice? Interferes with Alice's query to the phone book server?
- Answer: use *certificates*
- A certificate is a digitally-signed message containing an identity and a public key—prevents tampering.

Why Trust a Certificate?

- Who signed it? Why do you trust them?
- Certificates are generally signed by a *Certificate Authority* (CA)
- How do you know the public key of the CA? You need that to verify the signature
- Some public key (known as the *trust anchor*) must be provided out-of-band—trust has to start somewhere.

Certificate Authorities

- Who picks CAs? No one and every one.
- Your browser has some CAs built-in—because the CA paid the browser vendor enough money. Is that grounds for trust?
- Matt Blaze: “A commercial certificate authority can be trusted to protect you from anyone from whom they won’t take money.”

Things to Notice About Certificates

- Signer
- Validity dates
- Algorithms (RSA, SHA1)
- Key sizes
- Certificate usage—encryption and authentication, but *not* for issuing other certificates
- Certificate Revocation List (CRL)
- OCSP server: Online Certificate Status Protocol
- Logos

Examining Certificates

```
$ openssl s_client -connect www.google.com:443
CONNECTED(00000003)
depth=2 /C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEdjCCA16gAwIBAgIICRt+KuhvpPowDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UE
...
```

Examining Certificates (more)

```
$ openssl x509 -text </tmp/cert.txt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            09:1b:7e:2a:e8:6f:a4:fa
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=Google Inc, CN=Google Internet Authority G2
        Validity
            Not Before: Dec 11 12:02:58 2013 GMT
            Not After : Apr 10 00:00:00 2014 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc, CN=w
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:c1:bf:4a:95:07:e1:56:72:2e:45:68:7a:8f:3d:
                    ...
                Exponent: 65537 (0x10001)
```

Examining Certificates (even more)

```
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authenticat
            X509v3 Subject Alternative Name:
                DNS:www.google.com
            Authority Information Access:
                CA Issuers - URI:http://pki.google.com/GIAG2.crt
                OCSP - URI:http://clients1.google.com/ocsp
            X509v3 Subject Key Identifier:
                83:16:B1:57:49:89:F0:B6:18:4F:8B:B0:8F:06:3F:E2:E8:43:A0:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:4A:DD:06:16:1B:BC:F6:68:B5:76:F5:81:B6:BB:62:1A:BA:
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.11129.2.5.1
            X509v3 CRL Distribution Points:
                URI:http://pki.google.com/GIAG2.crl
    Signature Algorithm: sha1WithRSAEncryption
        (actual signature)
```

Who Issues Certificates?

- Identity-based: some organization, such as Verisign, vouches for your identity
  ☞Cert issuer is not affiliated with verifier
- Authorization-based: accepting site issues its own certificates
  ☞Cert issuer acts on behalf of verifier
- Identity-based certificates are better when user has no prior relationship to verifier, such as secure Web sites
- Authorization-based certs are better when verifier wishes to control access to its own resources—no need to trust external party

Why Revoke Certificates?

- Private key compromised
- Cancel authorization associated with certificate
- Note the difference between identity and authorization certificates here
- CA key compromised, e.g., DigiNotar

How Do You Revoke a Certificate?

- Revocation is hard! Verification can be done offline; revocation requires some form of connectivity
- Publish the URL of a list of revoked certificates
  ☞One reason for certificate expiration dates; you don't need to keep revocation data forever
- Online status checking

What Certificates Do You Accept?

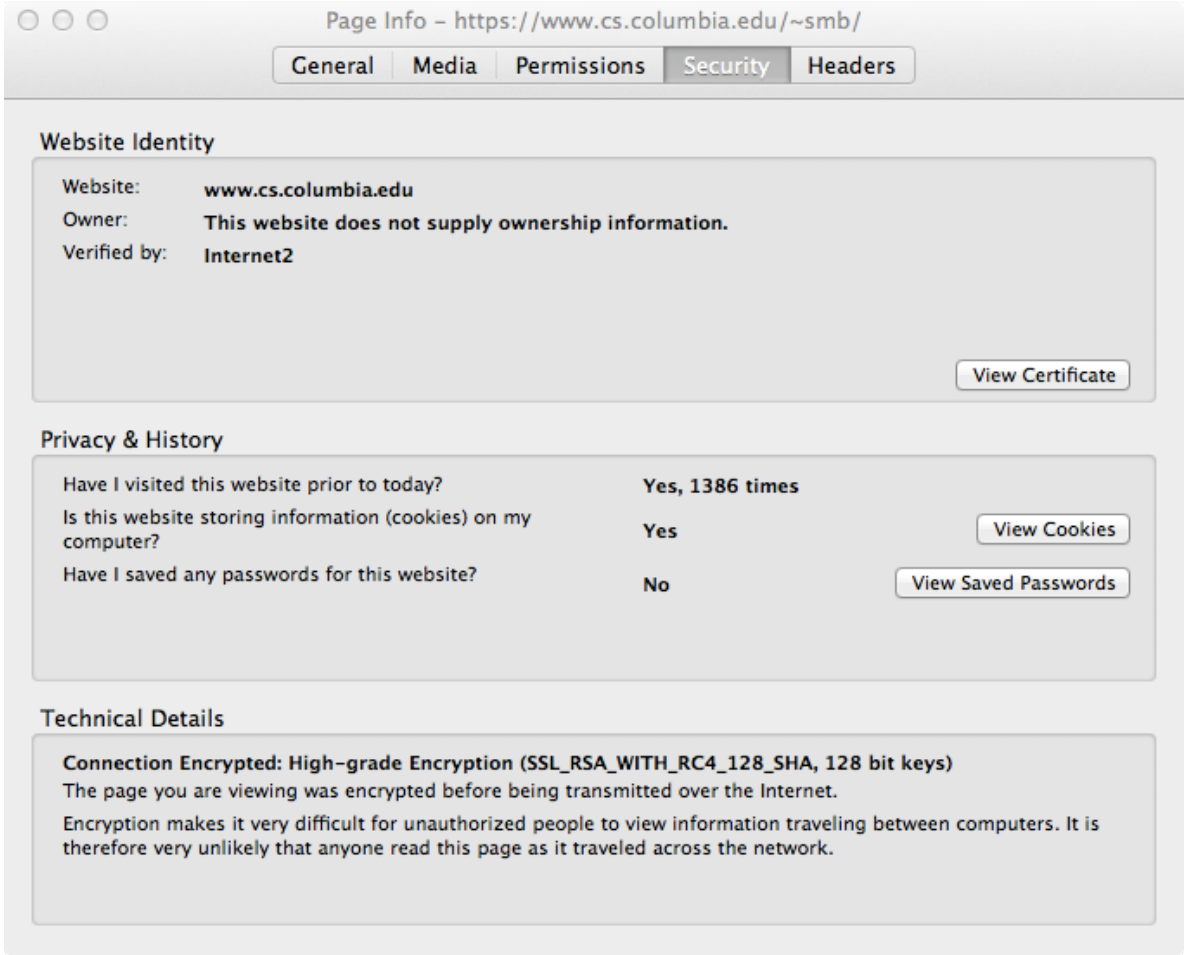
- Browers and (some) mailers have built-in list of CAs
- What were the listing criteria?
- Do you trust the CAs?
- What are their policies? Verisign's *Certification Practice Statement* (CPS) is at `http://www.verisign.com/repository/CPSv3.8.1_final.pdf`. Have you read it?
- All certificate verification has to start from *trust anchors*; these must be locally provisioned. (Firefox trusts about 200 CAs; Windows IE trusts > 300—and at least 10% are agencies of some government)

The Risks of Built-in CAs

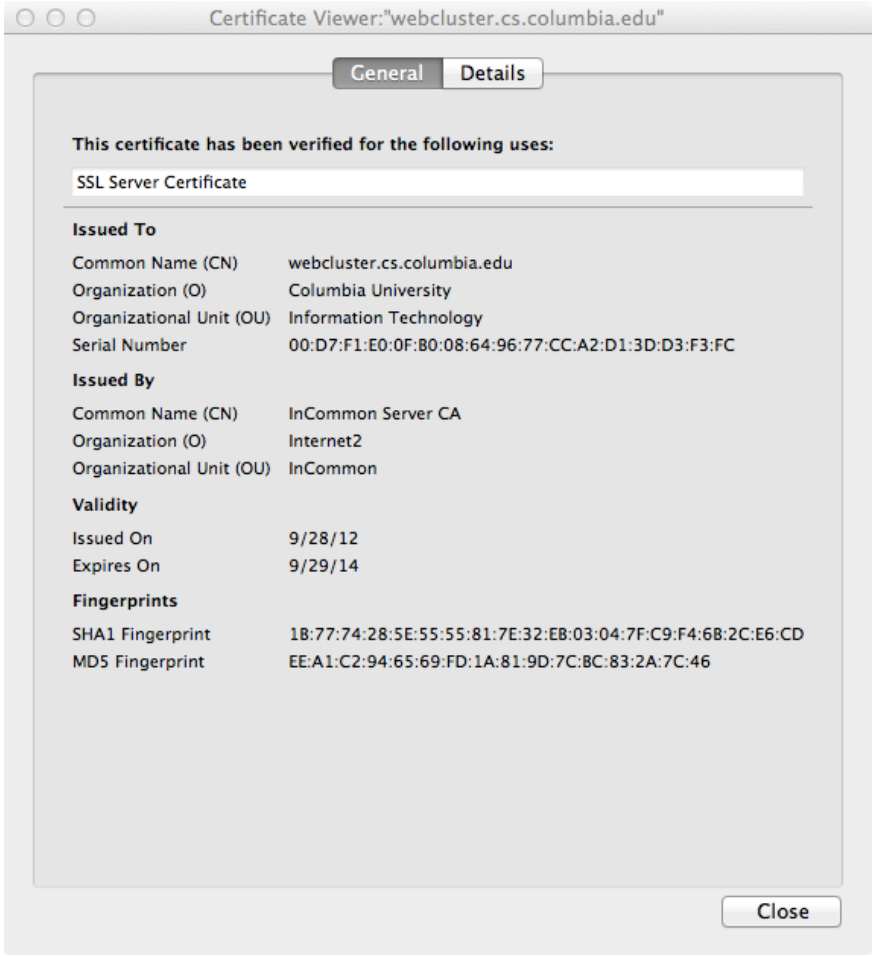
AOL Time Warner Root Certification Authorit...	Builtin Object Token
▾Autoridad de Certificacion Firmaprofesional CIF...	
Autoridad de Certificacion Firmaprofesional ...	Builtin Object Token
▾Baltimore	
Baltimore CyberTrust Root	Builtin Object Token
Baltimore CyberTrust Code Signing Root	Software Security Device
Baltimore CyberTrust Mobile Root	Software Security Device
▾BankEngine Inc.	
bankengine	Software Security Device
▾BelSign NV	
BelSign Object Publishing CA	Software Security Device
BelSign Secure Server CA	Software Security Device

It's amusing to read Baltimore's complex corporate history

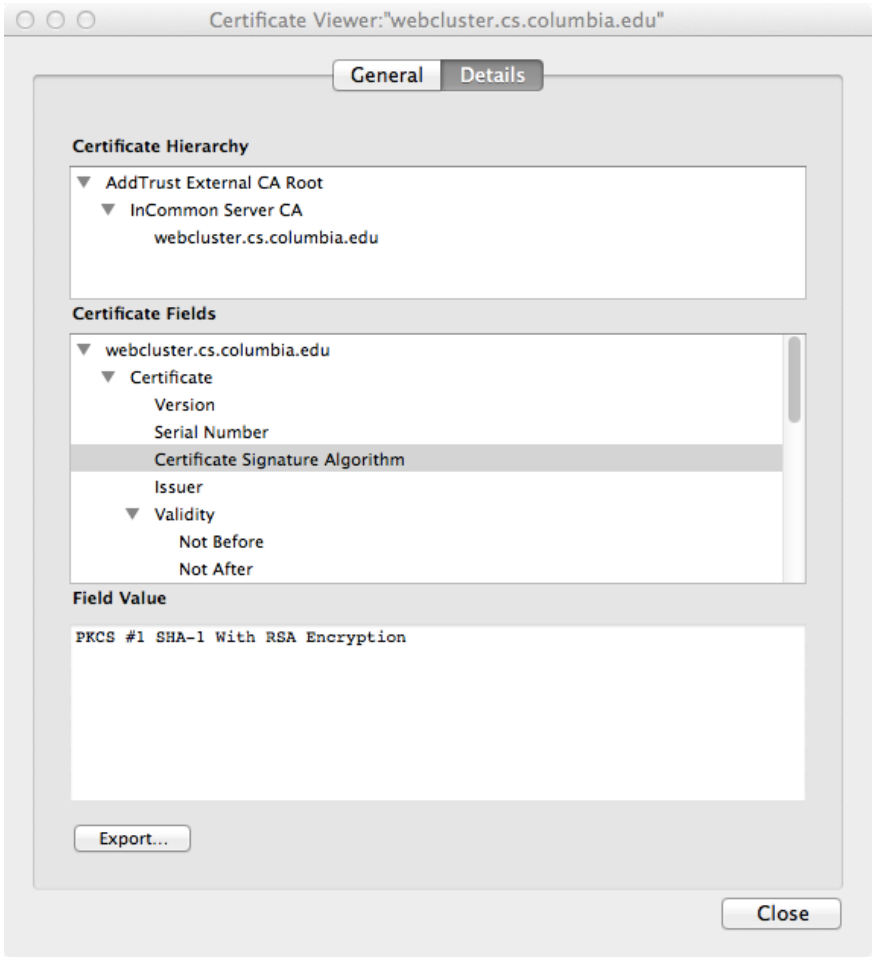
A Protected Web Page: Session Information



A Protected Web Page: Certificate Overview



A Protected Web Page: Certificate Details



Cryptographic Protocols

- Combine various cryptographic primitives in a series of messages
- Many different types, for many different goals
- Simplest example: "realistic" public key encryption message discussed earlier: $\langle \{m\}_k, (\text{pad}(k))^e \bmod n \rangle$
- Very common goal: Alice and Bob must agree on a key
- Very subtle; very hard to get right. Don't try it yourself

Recommended Primitives

- Block cipher: AES
- Stream cipher: RC4? It's now known to be cryptographically weak; AES in Output Feedback Mode or Counter Mode are better...
- Hash function: SHA2-256 (SHA-1 is being phased out)
- Public key, digital signature: RSA with 2048-bit modulus (or Elliptic Curve Cryptography—but watch out for patents)

What if You're Hacked?

- Cryptography prevents certain attacks; it doesn't guard against buggy code
- If you're hacked, the attacker can steal your private key
- If past traffic has been recorded, it can all be read, too
- We need a new cryptographic trick. . .

Diffie-Hellman (DH) Key Exchange

- Logarithms are easy to calculate: given a and a^x, it's easy to find x
- *Discrete logarithms* are difficult: given a and $a^x \bmod p$, where p is a large prime, it's very, very hard to find x
- Alice and Bob agree ahead of time on a and p
- Alice picks a random large x and sends $(a^x \bmod p)$ to Bob; Bob picks a large random y and sends $(a^y \bmod p)$ to Alice
- Alice knows x; she calculates $((a^y)^x \bmod p) \equiv (a^{xy})$. Bob knows y and can calculate the same value
- An eavesdropper knows only $(a^x \bmod p)$ and $(a^y \bmod p)$, cannot recover x or y, and hence cannot calculate a^{xy}.
- Derive a communication key from this shared secret: a^{xy}

Perfect Forward Secrecy

- DH provides an *unauthenticated* shared secret
- To use it, Alice and Bob must digitally sign the exchange
- If you set up a communications channel protected by such a shared secret, you have *perfect forward secrecy*: the attacker can read new sessions but not old ones
- Why not? Each session is protected by a fresh pair of random numbers; these are not stored, and hence are not recoverable by the attacker
- For new sessions while the attackers are still on your machine, they can see the random numbers that you use

How Does a User Store a Key?

- Store key on disk, encrypted
- Generally decrypted with passphrase
- Passphrases are weak, but they're a second layer, on top of OS file access controls

How Does a Server Store a Key?

- In a file? If the server is hacked, the key can be stolen
- Encrypted with a passphrase? What happens at reboot time?
- Secure cryptographic hardware
- ☞ Many PCs have a TPM (Trusted Platform Module) chip that can do some of this

Secure Cryptographic Hardware

- Can be used for users or servers
- More than just key storage; perform actual cryptographic operations
- Enemy has *no* access to secret or private keys
- Friends have no access, either
- Modular exponentiation can be done much faster with dedicated hardware

Hardware Issues

- Hardware must resist physical attack
- Environmental sensors: detect attack and erase keys
- Example: surround with wire mesh of known resistance; break or short circuit is detected
- Example: temperature sensor, to detect attempt to freeze battery

Limitations of Cryptographic Hardware

- Tamper-*resistant*, not tamper-*proof*
- Again: who is your enemy, and what are your enemy's powers?
- (Remember the "crypto in the hands of the enemy" problem.)
- How does Alice talk to it securely? How do you ensure that an enemy doesn't talk to it instead?
- What is Alice's *intent*? How does the crypto box know?
- What if there are bugs in the cryptographic processor software? (IBM's 4758 has a 486 inside. That can run complex programs. . .)

Summary of Key Management and Key Handling

- Sharing cryptographic keys is a delicate business
- Protecting keying material is crucial
- There are no great solutions for general-purpose systems, though proper hardware can prevent compromise (but not misuse) of long-term keys

Random Numbers

- Random numbers are vital for cryptography
- They're used for keys, nonces, primality testing, and more
- Where do they come from?

What is a Random Number?

- Must be *unpredictable*
- Must be drawn from a large-enough space
- Ordinary statistical-grade random numbers are not sufficient
- *Distribution* not an indication of randomness: loaded dice are still random!

Generating Random Numbers

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

—John von Neumann, 1951

Sources of Random Numbers

- Dedicated hardware random number sources
- Random numbers lying around the system
- Software pseudo-random generator
- Combinations

Hardware Random Number Generators

- Radioactive decay
- Thermal noise
- Oscillator pairs
- Other chaotic processes

Radioactive Decay

- Timing of radioactive decay unpredictable even in theory—it's a quantum process
- Problem: low bit rate from rational quantities of radioactive material
- Problem: not many computers have Geiger counters or radioactive isotopes attached. . .
- See `http://www.fourmilab.ch/hotbits/hardware.html` and `http://www.fourmilab.ch/hotbits/hardware3.html` for a description of how to do it. . .

Thermal Noise

- Any electronic device has a certain amount of random noise (thermal noise in the components)
- Example: Take a sound card with no microphone and turn up the gain to maximum
- Or use a digital camera with the lens cap on
- Problem: modest bit rate

Oscillator Pairs

- Have a free-running fast R-C oscillator (don't use a crystal; you don't want it accurate or stable!)
- Have a second, much slower oscillator
- At each maximum of the slow oscillator, sample the value of the fast oscillator
- Caution: watch for correlations or couplings between the two

Other Chaotic Processes

- Mouse movements
- Keystroke timing (low-order bits)
- Network packet timing (low-order bits)
- Disk seek timing: air turbulence affects disk internals (but what about solid state disks?)
- Problem: what if the enemy can observe the process?
- Cameras and Lava Lites®! (`http://www.lavarnd.org/`)

Problems

- Need deep understanding of underlying physical process
- Stuck bits
- Variable bit rate
- How do we measure their randomness?
- *Assurance*—how do we *know* it's working properly?

Software Generators

- Generally called PRNGs—pseudo-random number generators
- Again, ordinary generators, such as C's `random()` function or Java's `Random` class are insufficient
- Can use cryptographic primitives—encryption algorithms or hash functions—instead
- But—where does the seed come from?

Typical Random Number Generator

```
unsigned int
nextrand()
{
        static unsigned int state = 1;

        state = f(state);
        return state;
}
```

What's wrong with this for cryptographic purposes?

Problems

- The seed is predictable
- There are too few possible seeds
- The output is the state variable; if you learn one value, you can predict all subsequent ones

A Better Version

```
unsigned int
nextrand()
{
        static unsigned int state;
        static int first = 1;

        if (first) {first = 0; state = truerand();}
        state = f(state);
        return sha1(state);
}
```

Much Better

- State is initialized from a true-random source
- Can't invert sha1() to find state from return value
- But there is a serious problem here. What is it?

State Space

- sha1() isn't invertible, but we can do a brute force analysis
- `state` is too short, and can can be found in 2^{32} tries
- Estimated resources on a 3.4 Ghz Pentium: 3.6 hours CPU time; 150 GB
- Parallelizes nicely
- Need enough state—and hence enough true-random bits—that brute force is infeasible.

Private State

- An application can keep a file with a few hundred bytes of random numbers
- Generate some true-random bytes, mix with the file, and extract what you need
- Write the file back to disk—read-protected, of course—for next time

OS Facilities

- Many operating systems can provide cryptographic-grade random numbers
- `/dev/random`: True random numbers, from hardware sources
- `/dev/urandom`: Software pseudo-random number generator, seeded from hardware
- Windows has analogous facilities

A Well-Known Failure

- Wagner and Goldberg attacked Netscape 1.1's cryptographic random number generator
- Generator was seeded from process ID, parent process ID, and time of day
- `ps` command gives PID and PPID
- Consult the clock for time of day in seconds
- Iterate over all possible microsecond values
- Note: they did this by reverse-engineering; they did not have browser source code
- `http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html`

NIST, the NSA, and Random Numbers

- According to published reports based on the Snowden leaks, the NSA put a back door in a recent pseudo-random generator standard
- There's a number in the spec that can act like a public key
- If NSA knows the private key and 32 bytes of output from the generator, they can predict all future values

Hardware Versus Software Random Number Generators

- Hardware values can be true-random
- Output rate is rather slow
- Subject to environmental malfunctions, such as 60 Hz noise
- Software, if properly written, is fast and reliable
- Combination of software generator with hardware seed is usually best

Random Summary

- To paraphrase Knuth, random numbers should not be generated by a random process
- In many systems, hardware and software, random number generation is a very weak link
- Use standard facilities when available; if not, pay attention to RFC 4086

Cryptographic Threat Model

- Who is your enemy?
- Most ordinary attackers cannot exploit cryptographic weaknesses
- WEP is a notable exception: there are canned cracking programs
- Bad cryptography will keep out ordinary attackers
- Governments can and do exploit bad cryptography
- Note: even bad cryptography will slow down governments, especially against broad-scale monitoring

(Apparent) State-Launched Cryptographic Attacks

- Stuxnet (attributed to the United States and Israel) used genuine certificates for which the private keys had been stolen
- Flame (a relative of Stuxnet) used a previously-unknown attack on MD5 to generate fraudulent certificates
- ☞ The attack had a complexity of at least $2^{46.6}$ and was optimized for massively parallel hardware
- There have been several attacks (attributed to Iran) involving fake certificates generated by hacking a commercial certificate authority
- There are probably more of these in the wild that have not yet been detected

Putting it All Together

- Only use standard cryptographic primitives
- Only use standard cryptographic protocols
- Pick your trust anchors carefully
- Protect your long-term keys