CSCI E-170
October 17, 2005
L04: Public Key Cryptography

Today's Outline

1. LJ Nonsense
2. HW2 - Graded and returned.
3. HW3 - How are things going?
4. Review of L03 (hash functions & ciphers)
5. Public Key Cryptography
6. Applications of Public Key

Nonsense

Web Server Migration (EECS to Simson.NET)

EECS Mail Outage.

LJ Post-dated entries. Why?

HW2 - Forensics

Most of the homeworks were excellent.

What did people learn?

Grades have been sent out.

Outstanding questions?

HW3 - Crypto

Posted a few days late. Sorry!

(We made it easier.)

Due October 24th; web submission

Questions to csci_e_170a

http://www.simson.net/e170/hw3.php

HW3 - Hashing Issues 1

Does MD5 have a key?

Why do we use MD5 and not SHA-1?

It is “safe” to use MD5?

Impact of hash databases

Google me a hash…

```
% echo -n "foo" | md5
acbd18db4cc2f85cedef654fccc4a4d8
% echo -n "life" | md5
e155e1bb4a9c38e3baf90637ab7865df
% echo -n "smith" | md5
a66e44736e753d4533746ced572ca821
% echo -n "computer" | md5
df53ca268240ca76670c8566ee54568a
% echo -n "something" | md5
437b930db84b8079c2dd804a71936b5f
% echo -n "else" | md5
2954e92a9b4d0e998fe4893f8141649a
% echo -n "garfinkel" | md5
0c404a59bf8704d0059c0c0f8a2753a4
%
```

How do you defeat a hash database?

Ways of defeating a hash database…

Salt.

Change the hash algorithm.

Hash bigger things.

HW3 - Hashing Issues 2

Think about the difference between these two commands:

```
% echo "foo" | md5sum
d3b07384d113edec49eaa6238ad5ff00

% echo -n "foo" | md5sum
acbd18db4cc2f85cedef654fccc4a4d8
```

What’s going on here?

HW3 - Second Half is Public Key

Get a certificate.

OpenSSL

Both of these will be explained now….

Public Key Algorithms

DH

RSA

Digital Signatures

Certs and Certification

Public Key: One key seals (encrypted), the other key unseals (decrypts)

$$M' = f(M,K_1)$$

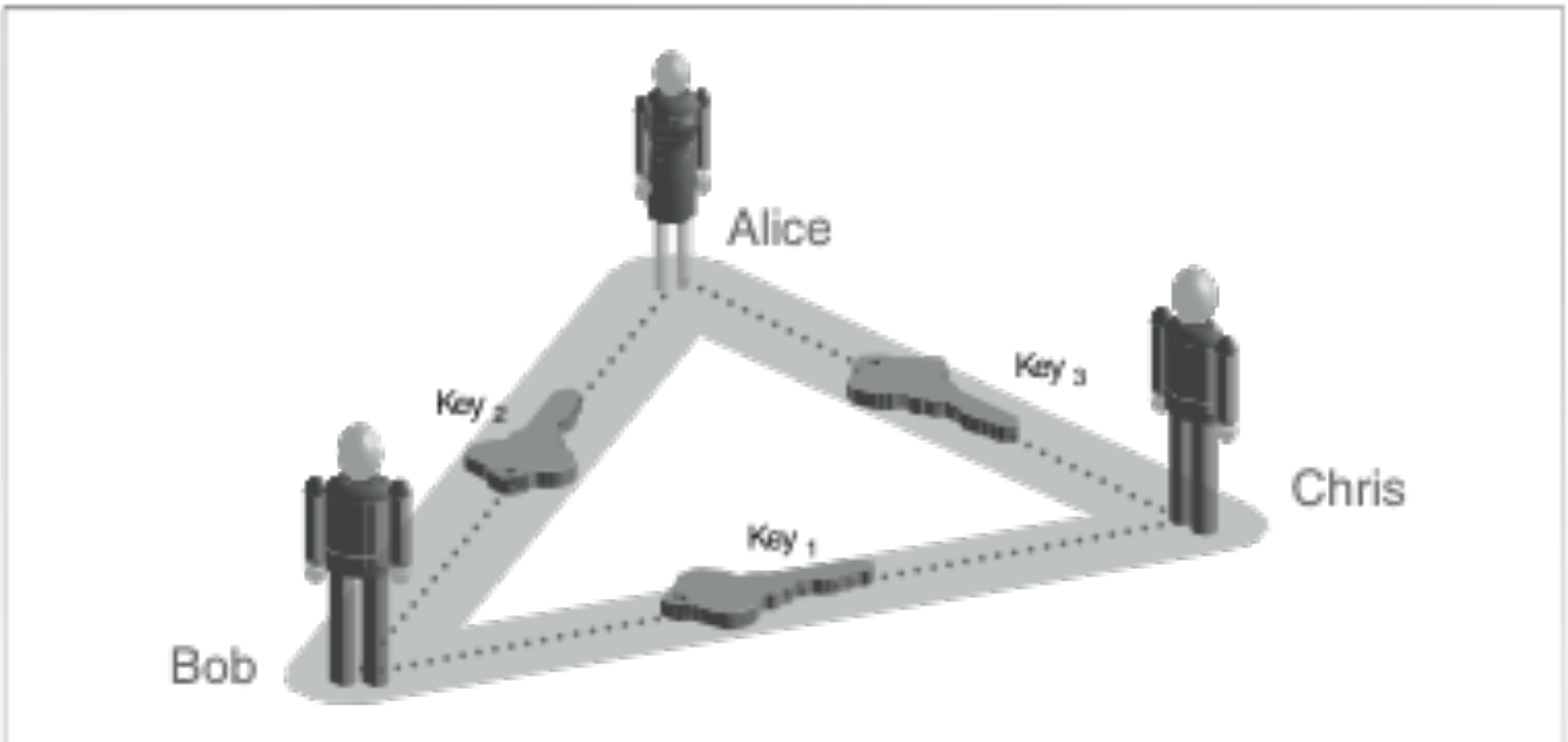
$$M = f'(M',K_2)$$

Obvious today; was revolutionary in 1974!

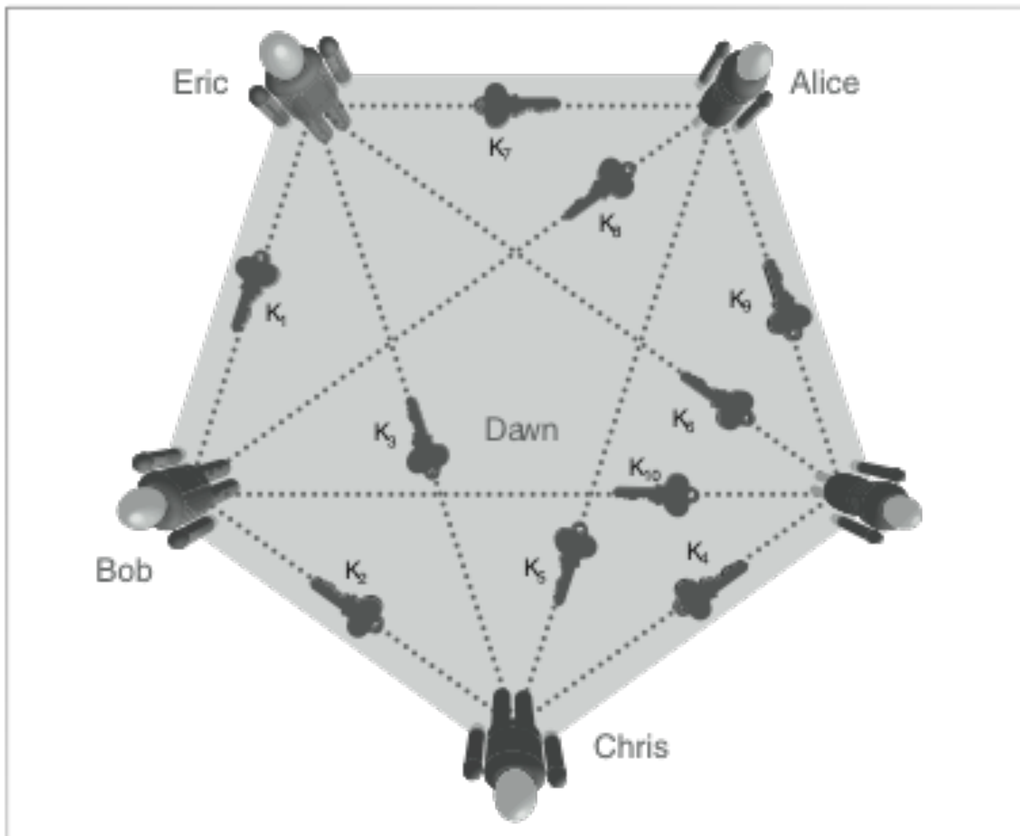
Secret Key vs. Public Key

	secret key	**public key**
algorithm type	symmetric	asymmetric
basis	substitution and transposition	math
speed	fast	slow
encrypts	blocks of data	numbers
uses	encrypting files	encrypting email

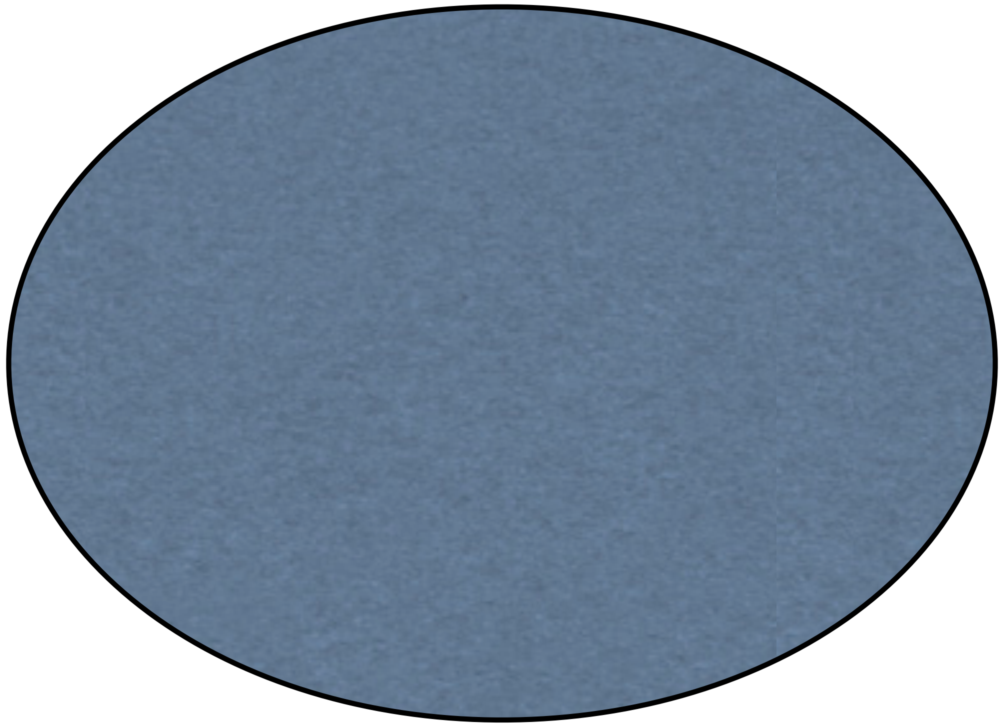
With symmetric cryptography, 3 people need 3 keys to communicate.



Five people need 9 keys to communicate.



And 1000 people need 499,550 keys to communicate.

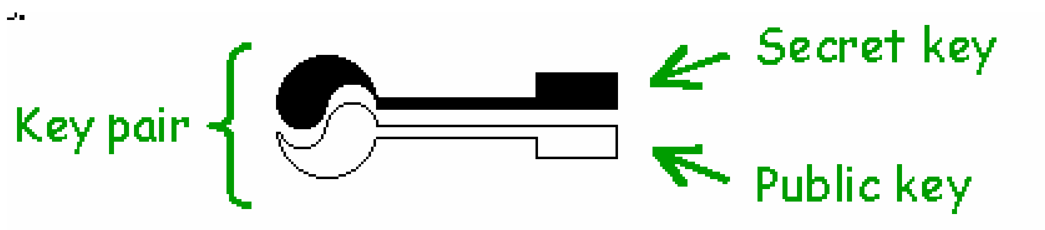


$$\# \text{ keys} = \frac{(n)(n-1)}{2}$$

Public key cryptography uses two keys.

Public key = seals/encrypts data

Private key = unseals/decrypts data



Whitten's "Metaphor Tailoring."

Public key cryptography offers several advantages over symmetric cryptography:

1. Participants can communicate securely without prior arrangement.
   - Secure e-mail. (Alice sends a message to Bob.)
   - Interactively. (Alice and Bob have a phone call.)

2. If public keys can be *published*, then we can have digital signatures.

Ralph Merkle's Puzzles allowed secure interactive communication in 1974…

Puzzle P(M)— takes 1000 minutes to compute P^{-1} and find M.

Alice creates keys K_1 through K_{1000} and sends Puzzles P(1,K_1) through P(1000,K_{1000}) to Bob in *random order*.

Bob picks P(n, K_n) at r*andom*, cracks it, sends P(n) to Alice.

Time for Alice and Bob to crack: 1000*2

Time for an observer to crack: 1000*1000

Ralph Merkle figured this out in 1974, but nobody understood it!

Reviewers at ACM didn’t understand the project!

- “Too far out of the mainstream of cryptography.”
- “Bad science: everybody knows that it is important to keep cryptography keys secret.”

Communications finally published the paper in 1978, with an editorial note.

Whitfield Diffie & Matin Hellman: A more secure interactive protocol

"Multi-User Cryptographic Techniques," written in fall 1975 for the 1976 National Computer Conference

Proposed the idea of Public Key Cryptography.

May 1976 - Diffie Hellman algorithm invented.

Interactive protocol for 2 participants.

Diffie Hellman Algorithm

Relies on the fact that $g^{ab} \pmod{p} = g^{ba} \pmod{p}$

System Parameters: Prime p=23, base g=5

Alice and Bob choose secret integers

(Alice a=6; Bob b=15)

Alice computes 5^a (mod p)=8 and sends to Bob

Bob computes 5^b(mod p)=19 and sends to Alice

Alice computes 19^a(mod p) = 2

Bob computes 8^b(mod p) = 2

2 is the encryption key!

Problems with Diffie-Hellman (circa 1976)

Exponential math was slow.

(computers got faster)

DH is an *interactive* protocol.

(Taher ElGamal solved this in 1984)

The RSA algorithm

Invented by Rivest, Shamir and Adelman

(Previously invented by Clifford Cocks at GCHQ in ‘73, but ignored.)

First, Alice and Bob make keys.

- Each choose different prime numbers p & q; compute n=pq
- Choose e=65
- Compute d such that de=1 (mod (p-1)(q-1))
- Public key: n & e
- Private key: n & d

Using the RSA Algorithm

Encrypt a message:

$$c = m^e \pmod{n}$$

Decrypt a message:

$$m = c^d \pmod{n}$$

Notice that encryption and decryption are symmetric. This has caused much confusion!

Padding and RSA

It is vital to “pad” m with random prefix and suffix.

$c = m^e \pmod{n}$

$m = c^d \pmod{n}$

Typical pad:

m’ = {rand1,m,rand2}

Beware of “raw RSA.”

Most public key systems are actually hybrid systems.

- Use Diffie-Hellman or RSA to exchange a 128-bit session key
- Use RC2/RC4/AES to encrypt bulk information
- Use certificates to vouch for public keys.

Random Numbers are *Very Important* for public key cryptography:

Random Numbers

- –Use them to pick your initial public/private key pair.
- –Use them for picking session keys

Come to think of it, they are important for symmetric key cryptography too!

Sources of Random Numbers

good	bad
keystroke timing	time of day
packet timing (*)	process ID
radiation, lava lamp	rand(), random()
FM radio	ethernet address
microphone	blocks of CDROMs

There are many famous cases in which a poor random number compromised security.

Early Netscape Navigator

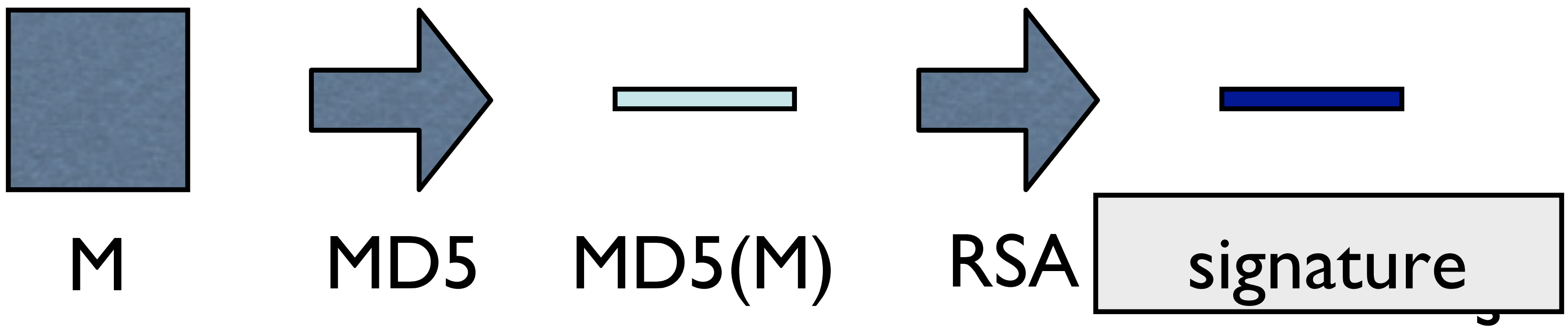
Kerberos R4 & R5

Is this sequence random: 1, 1, 1, 1, 1 … ?

http://www.random.org/ ?

RFC 1750 discusses "best practice" for random numbers. (http://www.faqs.org/rfcs/rfc1750.html)

Digital Signatures

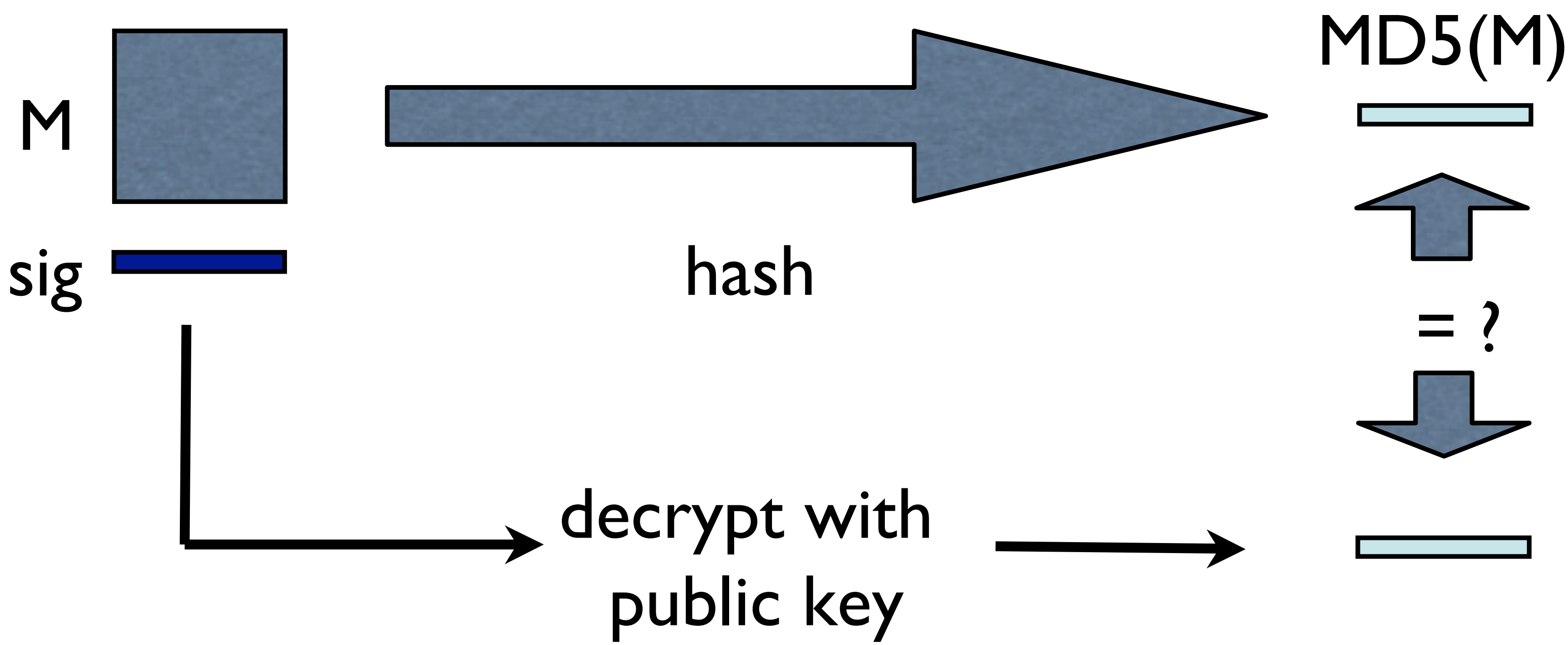


Encrypt with the *secret* key, decrypt with the *public* key.

Used for verifying that the signer had the private key.

Instead of encrypting the entire Message, we usually encrypt a hash

Verifying a Digital Signature



If the hash matches the decrypted signature, the signature verifies!

Using Digital Signatures

To sign a digital signature, you need...

–your private key.

To verify a digital signature, you need...

–the other person’s public key...

–the name of the algorithm the person has used for the digital signature.

Certificates bind public keys to identities. [Kohnfelder '78]

"Simson Garfinkel"
KeyID 9c309

Signed by KeyCertCo

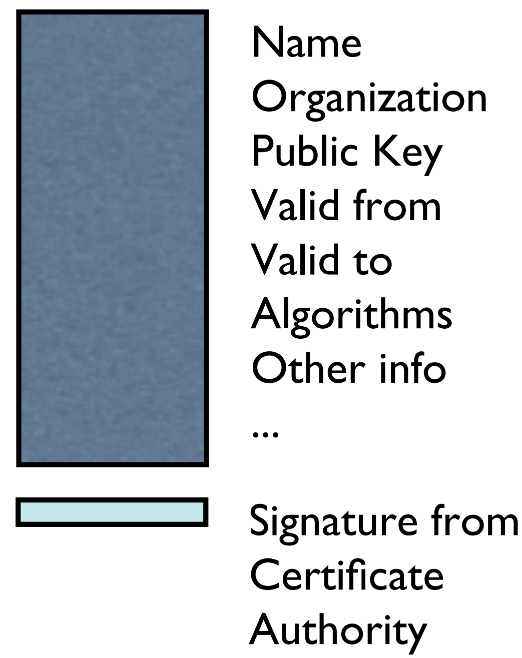
Digital Certificates

Certificates “register” public keys

Certificates are signed with digital signatures!

Certificates signed by a “Certificate Authority”

X.509:



There are many kinds of X.509(v)3 certificates.

Certificate Authorities

User Certificates

Server Certificates

All of these certificates have the same format, but different purpose.

Demo: Look at the MacOS certificates with Keychain

Certificate Authorities issue Certificates, not Keys

Process:

- User creates public/private keypair
- User sends Certificate Signing Request (CSR) to the CA.
- CA verifies the sender's identity.
- CA sends the certificate back to the user

The CA's public key must be widely distributed.
("Download here" doesn't work; why not?)

DEMO:
Certificate Authorities in Internet Explorer

How many can you find?

Who are these companies?

What does their presence mean?

What good is a Certificate from a CA?

In Theory:

- –Allows you to “prove your identity” on the Internet. (Age, Sex, Name)
- –Allows you to digitally sign documents.
- –Allows users to prove “membership” without having to distribute a membership list.

In practice:

- –Allows you to run an SSL server without a warning

Certificate Revocation Lists (CRLs)

List of “mistakes?”

- –User lost their Private key.
- –CA signed the wrong key.
- –http://crl.verisign.com/

Technically, should be checked whenever a CA cert is trusted.

Most application do not check CRLs. Why not?

Certs and Keys with OpenSSL

OpenSSL command-line interface:

–Useful for making keys, certs and CSRs.

–Useful for simple testing

–Useful for converting one format to another (handles PKCS, PEM, and others)

–Useful for testing SSL servers

Public key systems today

PGP

S/MIME

SSL, Authenticode.

Questions to consider:

- How do you make trust decisions understandable and relevant?
- Absolute identity or continuity of identity?
- Why are some of these systems successful but not others?

OpenSSL Commands

ca - Certificate Authority Management

ciphers - lists ciphers in your implementation

crl - Manage Certificate Revocation Lists

dgst - calculation of md digests

dsa - Manages DSA algorithm

dsparam - Generate and manage DH keys