

Challenges in Network Security Protocols

Radia Perlman
November 2006

(radia.perlman@sun.com)

SI SPY NET WORK, BIG FEDJAW IOG LINK KYXOGY

Section: Introduction

- What is secure communication?
- What can the attackers do?

Some Examples to Motivate the Problems

- Sharing files between users
 - File store must authenticate users
 - File store must know who is authorized to read and/or update the files
 - Information must be protected from disclosure and modification on the wire
 - Users must know it's the genuine file store (so as not to give away secrets or read bad data)

Examples cont'd

- Electronic Mail
 - Send private messages
 - Know who sent a message (and that it hasn't been modified)
 - Non-repudiation - ability to forward in a way that the new recipient can know the original sender
 - Anonymity

Examples cont'd

- Electronic Commerce
 - Pay for things without giving away my credit card number to an eavesdropper or phony merchant
 - Buy anonymously
 - Merchant wants to be able to prove I placed the order

Sometimes goals conflict

- Privacy vs. company (or govt) wants to be able to see what you're doing
- Anonymity vs knowing who sent a message
- Losing data vs. disclosure (copies of keys)

The Problem

- Internet evolved in a world w/out predators. DOS was viewed as illogical and undamaging.
- The world today is hostile. Only takes a tiny percentage to do a lot of damage.
- Must connect mutually distrustful organizations and people with no central management.
- And society is getting to depend on it for reliability, not just “traditional” security concerns.

Security means different things to different people

- Limit data disclosure to intended set
- Monitor communications to catch terrorists
- Keep data from being corrupted
- Make sure nobody can access my stuff without paying for it
- Destroy computers with pirated content
- Track down bad guys
- Communicate anonymously

Insecurity

*The Internet isn't insecure. It may be unsecure.
Insecurity is mental state. The users of
the Internet may be insecure, and perhaps
rightfully so.....* Simson Garfinkel

Intruders: What Can They Do?

- Eavesdrop--(compromise routers, links, routing algorithms, or DNS)
- Send arbitrary messages (including IP hdr)
- Replay recorded messages
- Modify messages in transit
- Write malicious code and trick people into running it
- Exploit bugs in software to ‘take over’ machines and use them as a base for future attacks

Some basic terms

- Authentication: “Who are you?”
- Authorization: “Should you be doing that?”
- DOS: denial of service
- Integrity protection: a checksum on the data that requires knowledge of a secret to generate (and maybe to verify)

Sometimes goals conflict

- privacy vs. company (or govt) wants to be able to see what you're doing
- losing data vs. disclosure (copies of keys)
- denial of service vs. preventing intrusion
- privacy vs. virus scanning

So...you need to know what
problem you are solving!

Section: Cryptography

Cryptography

- It's not as scary as people make it out to be
- You don't need to know much about it to understand what it can and can't do for you

Features

- Main features
 - Encryption
 - Integrity protection
 - Authentication
- More things
 - Denial of service defense
 - Nonrepudiation
 - Privacy

Cryptography

- Three kinds of cryptographic algorithms you need to understand
 - secret key
 - public key
 - cryptographic hashes
- Used for
 - authentication, integrity protection, encryption

Secret Key Crypto

- Two operations (“encrypt”, “decrypt”) which are inverses of each other. Like multiplication/division
- One parameter (“the key”)
- Even the person who designed the algorithm can’t break it without the key (unless they diabolically designed it with a trap door)
- Ideally, a different key for each pair of users

Secret key crypto, Alice and Bob share secret S

- $\text{encrypt} = f(S, \text{plaintext}) = \text{ciphertext}$
- $\text{decrypt} = f(S, \text{ciphertext}) = \text{plaintext}$
- authentication: send $f(S, \text{challenge})$
- integrity check: $f(S, \text{msg}) = X$
- verify integrity check: $f(S, X, \text{msg})$

A Cute Observation

- Security depends on limited computation resources of the bad guys
- (Can brute-force search the keys)
 - assuming the computer can recognize plausible plaintext
- A good crypto algo is linear for “good guys” and exponential for “bad guys”
- Faster computers work to the benefit of the good guys!

Block cipher



Block size might be
64, 128, or 256 bits

Secret Key Block Ciphers

- Examples, AES, DES, 3DES
- Differ in block size, key size

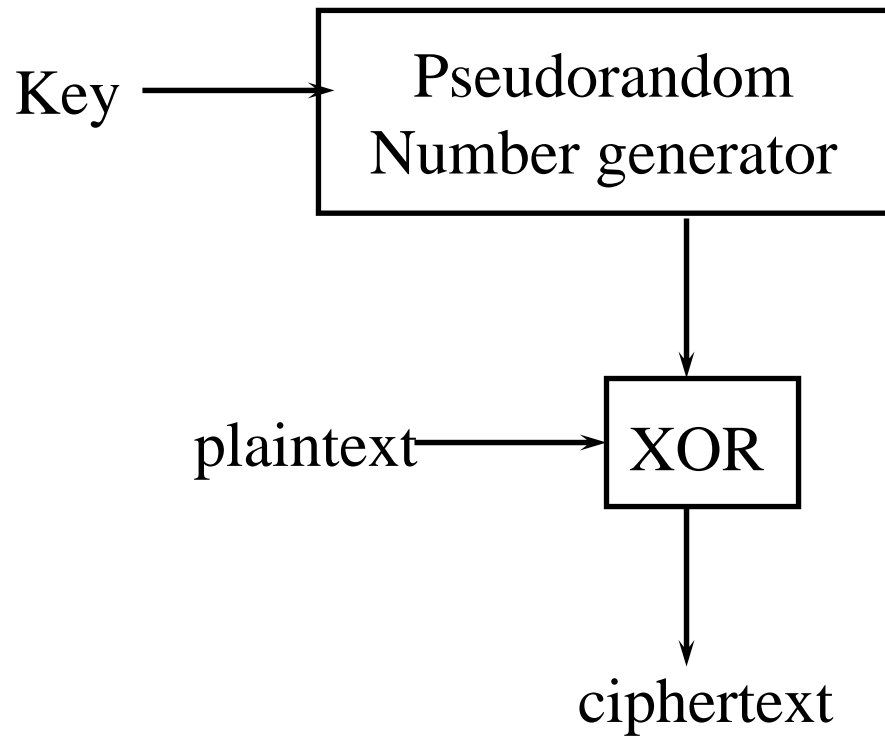
Secret Key Stream Ciphers

- Examples, RC4 (or block ciphers can be used as stream ciphers)

XOR (Exclusive-OR)

- Bitwise operation with two inputs where the output bit is 1 if exactly one of the two input bits is one
- $(B \text{ XOR } A) \text{ XOR } A = B$
- If A is a “one time pad”, very efficient and secure
- Common encryption schemes (e.g. RC4) calculate a pseudo-random stream from a key

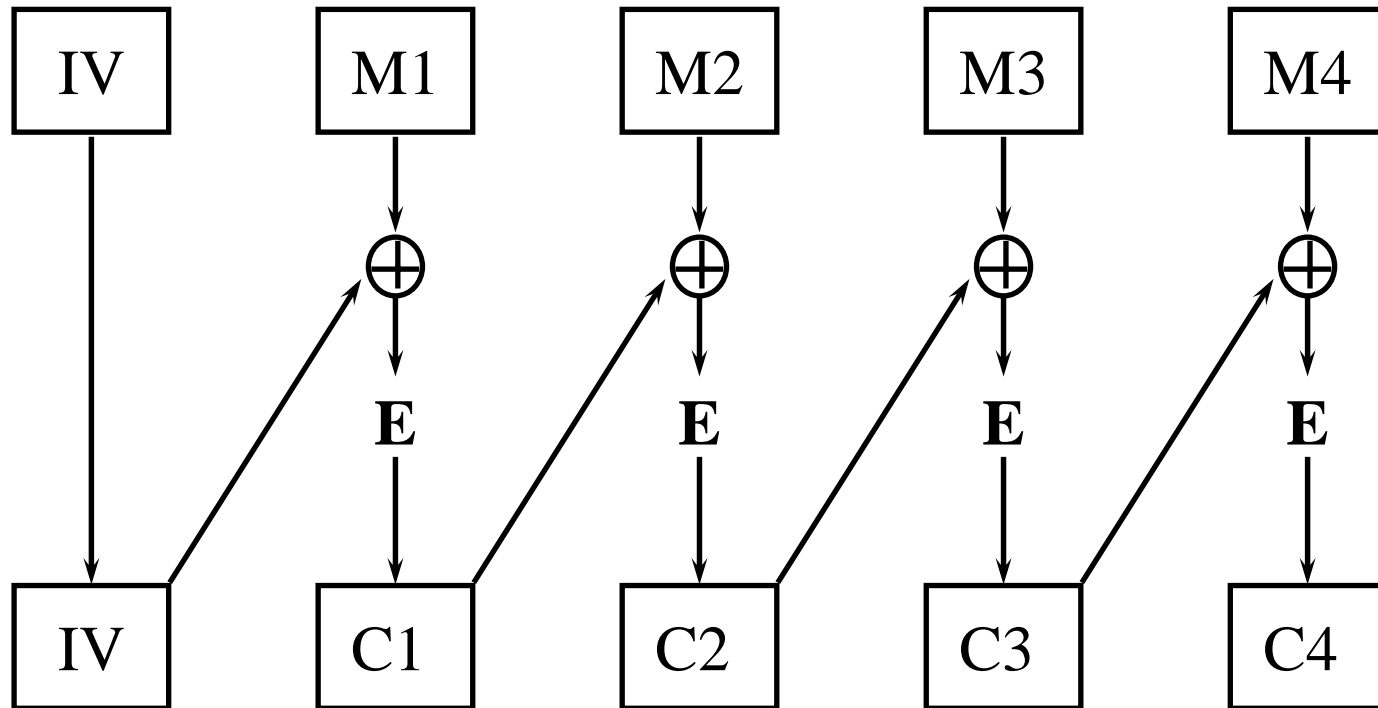
Stream cipher



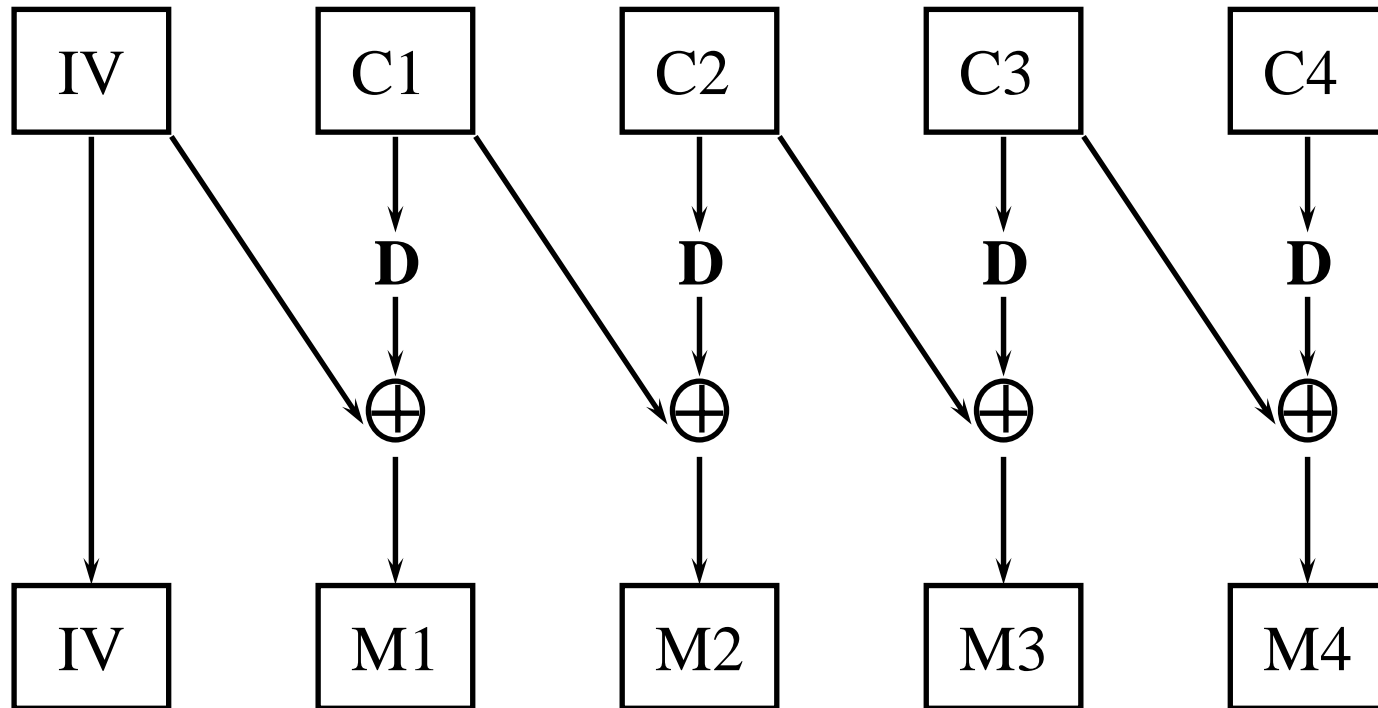
Encrypting Large Messages

- The basic algorithms encrypt a fixed size block
- Obvious solution is to encrypt a block at a time. This is called Electronic Code Book (ECB)
- Repeated plaintext blocks yield repeated ciphertext blocks
- Other modes “chain” to avoid this (CBC, CFB, OFB)
- Encryption does not guarantee integrity!

CBC (Cipher Block Chaining)



CBC Decryption



Public Key Crypto

- Two keys per user, keys are inverses of each other (as if nobody ever invented division)
 - public key “e” you tell to the world
 - private key “d” you keep private
- Yes it’s magic. Why can’t you derive “d” from “e”?
- and if it’s hard, where did (e,d) come from?

Digital Signatures

- One of the best features of public key
- An integrity check
 - calculated as $f(\text{priv key}, \text{data})$
 - verified as $f(\text{public key}, \text{data}, \text{signature})$
- Verifiers don't need to know secret
- vs. secret key, where integrity check is generated and verified with same key, so verifiers can forge data

Cryptographic Hashes

- Invented because public key is slow
- Slow to sign a huge msg using a private key
- Cryptographic hash
 - fixed size (e.g., 160 bits)
 - But no collisions! (at least we'll never find one)
- So sign the hash, not the actual msg
- If you sign a msg, you're signing all msgs with that hash!

Popular Public Key Algorithms

- RSA: nice feature: public key operations can be made very fast, but private key operations will be slow. Patent expired.
- DSS: Digital Signature Standard – pushed by U.S. government
- ECC (elliptic curve crypto): smaller keys, so faster than RSA (but not for public key ops). Some worried about patents

Popular Hashes

- Most popular hash today SHA-1 (secure hash algorithm)
- Starting to roll out: SHA-256
- Older ones (MD2, MD4, MD5) still around
- Popular secret-key integrity check: hash together key and data
- One popular standard for that within IETF: HMAC

Hash function security controversy

- Security of a hash function defined in terms of collision resistance
- In most uses, a much lower standard of security is required
- For use in HMAC, lowest of all
- MD2, MD4, MD5 “broken”. SHA-1 has “weaknesses”.
- Beware the New York Times attack!
- Make your protocols “crypto-agile”.

Crypto-agile

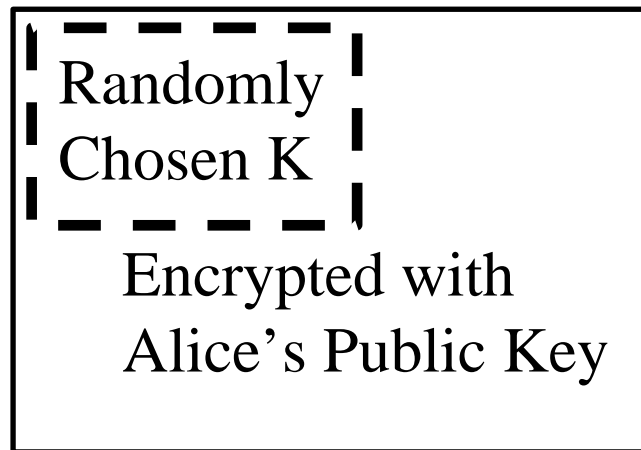
- Notice all the crypto algorithms
- The cryptographers can tell you at any time that the one you picked isn't good
- So you have to design your protocols to be able to switch crypto algorithms
- Which means for interoperability your protocol has to do negotiation

Encrypting with public key

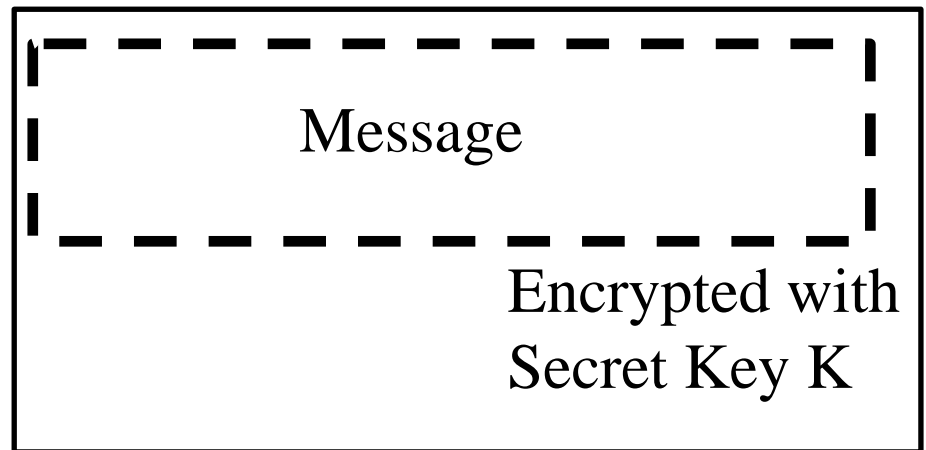
Instead of:



Use:

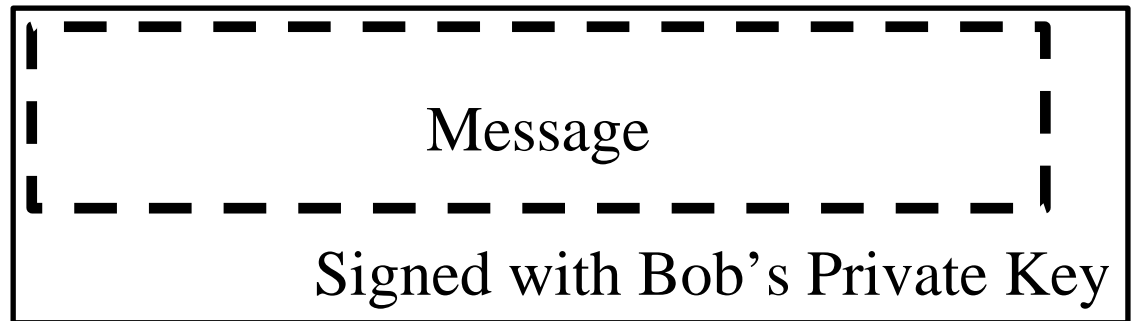


+



Digital Signatures

Instead of:

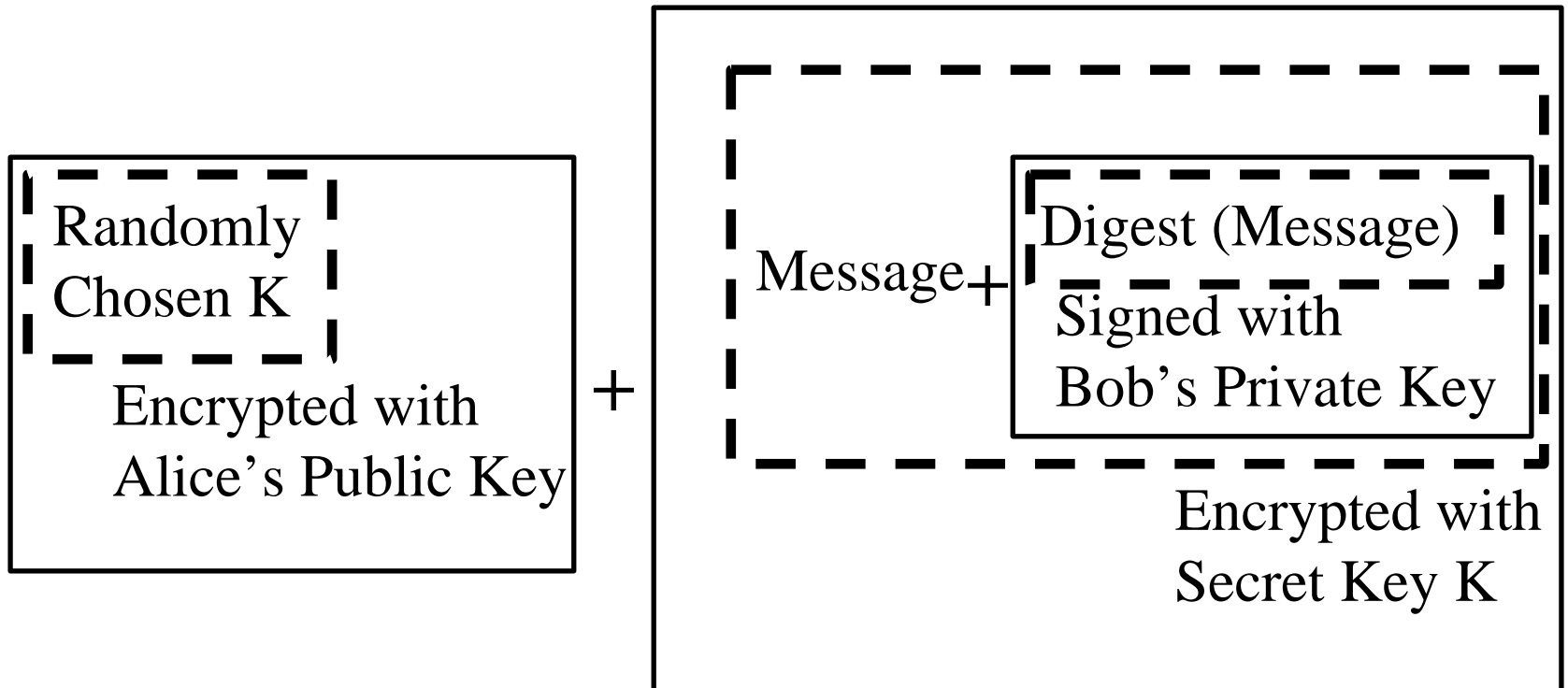


Use:

Message +



Signed and Encrypted Message



Don't try this at home

- No reason (except for the **Cryptography Guild**) to invent new cryptographic algorithms
- Even if you could invent a better (faster, more secure) one, nobody would believe it
- Use a well-known, well-reviewed standard

Challenge / Response Authentication

Alice (knows K)


Bob (knows K)

I'm Alice



Pick Random R
Encrypt R using K
(getting C)

If you're Alice, decrypt C



R



Non-Cryptographic Network Authentication (olden times)

- Password based
 - Transmit a shared secret to prove you know it
- Address based
 - If your address on a network is fixed and the network makes address impersonation difficult, recipient can authenticate you based on source address
 - UNIX `.rhosts` and `/etc/hosts.equiv` files

Agenda

- Introduction to Security
- Introduction to Cryptography
- **Authenticating People**
- Security mechanisms to reference rather than invent
 - Public Key / Secret Key infrastructures
 - Formats
- Security Considerations Considerations
- Security Working Groups

Authenticating people

- What you know (passwords)
- What you have (smart cards, SecurID cards, challenge/response calculators)
- What you are (biometrics)
- N-factor (e.g., 2): n of the above

Passwords

- Why are we still using passwords?
 - They are annoying
 - Susceptible to keyboard loggers, eavesdropping, on-line and off-line guessing

People

- “Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed, but they are sufficiently pervasive that we must design our protocols around their limitations.”
 - Network Security: Private Communication in a Public World

On-Line Password Guessing

- If guessing must be on-line, password need only be mildly unguessable
- Can audit attempts and take countermeasures
 - ATM: eat your card
 - military: shoot you
 - networking: lock account (subject to DOS) or be slow per attempt

Off-Line Password Guessing

- If a guess can be verified with a local calculation, passwords must survive a very large number of (unauditable) guesses

Passwords as Secret Keys

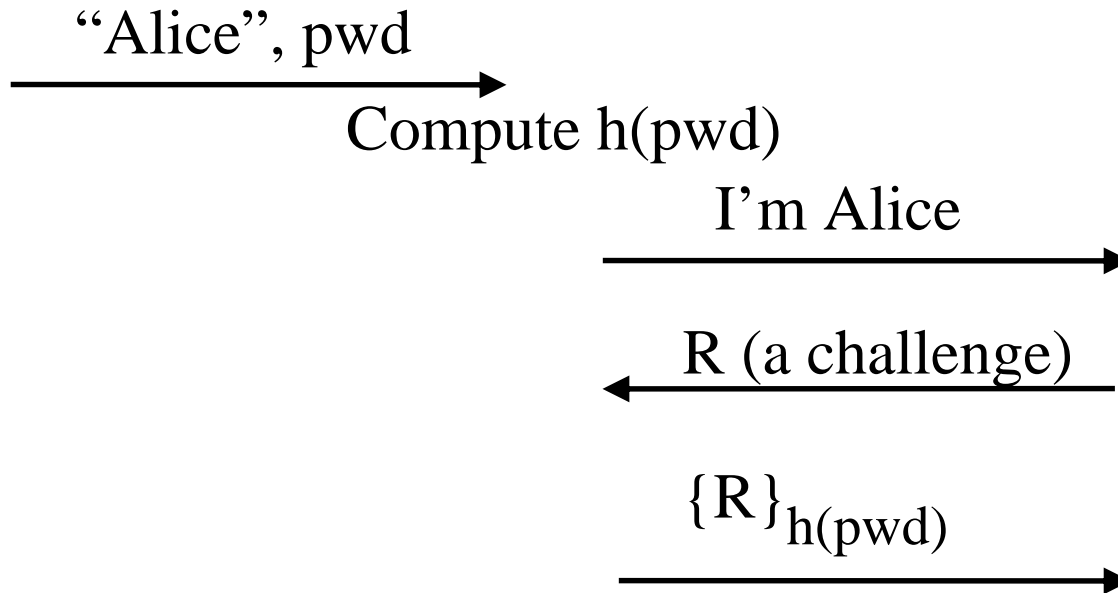
- A password can be converted to a secret key and used in a cryptographic exchange
- An eavesdropper can often learn sufficient information to do an off-line attack
- Most people will not pick passwords good enough to withstand such an attack

Off-line attack possible

Alice
(knows pwd)

Workstation

Server
(knows $h(\text{pwd})$)



Other ways of authenticating people

- OTP
- Tokens (e.g., challenge/response, time-based)

Section: Key distribution

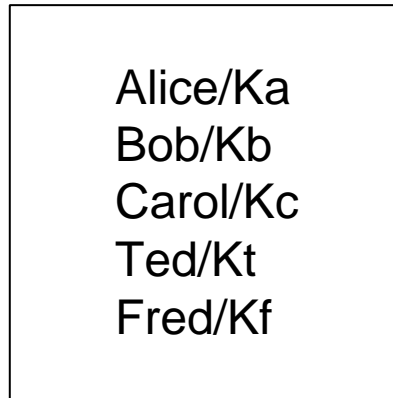
Key Distribution - Secret Keys

- What if there are millions of users and thousands of servers?
- Could configure n^2 keys
- Better is to use a Key Distribution Center
 - Everyone has one key
 - The KDC knows them all
 - The KDC assigns a key to any pair who need to talk

KDC

Alice/Ka

Bob/Kb



Ted/Kt

Fred/Kf

Carol/Kc

Key Distribution - Public Keys

- Certification Authority (CA) signs “Certificates”
- Certificate = a signed message saying “I, the CA, vouch that 489024729 is Radia’s public key”
- If everyone has a certificate, a private key, and the CA’s public key, they can authenticate

Key distribution with CAs

Alice

Bob

[Alice's key is x]CA



[Bob's key is y]CA



data signed and/or encrypted

KDC vs. CA Tradeoffs

- KDC solution less secure
 - Highly sensitive database (all user secrets)
 - Must be on-line and accessible via the net
 - complex system, probably exploitable bugs, attractive target
 - Must be replicated for performance, availability
 - each replica must be physically secured

KDC vs. CA

- KDC more expensive
 - big, complex, performance-sensitive, replicated
 - CA glorified calculator
 - can be off-line (easy to physically secure)
 - OK if down for a few hours
 - not performance-sensitive
- Performance
 - public key slower, but avoid talking to 3rd party during connection setup

KDC vs. CA Tradeoffs

- CA's work better interrealm, because you don't need connectivity to remote CA's
- Revocation levels the playing field somewhat

Strategies for CA Hierarchies

- Monopoly
- Oligarchy
- Anarchy
- Bottom-up

Monopoly

- Choose one universally trusted organization
- Embed their public key in everything
- Give them universal monopoly to issue certificates
- Make everyone get certificates from them
- Simple to understand and implement

What's wrong with this model?

- Monopoly pricing
- Getting certificate from remote organization will be insecure or expensive (or both)
- That key can never be changed
- Security of the world depends on honesty and competence of that one organization, forever

One CA Plus RAs

- RA (registration authority), is someone trusted by the CA, but unknown to the rest of the world (verifiers).
- You can request a certificate from the RA
- It asks the CA to issue you a certificate
- The CA will issue a certificate if an RA it trusts requests it
- Advantage: RA can be conveniently located

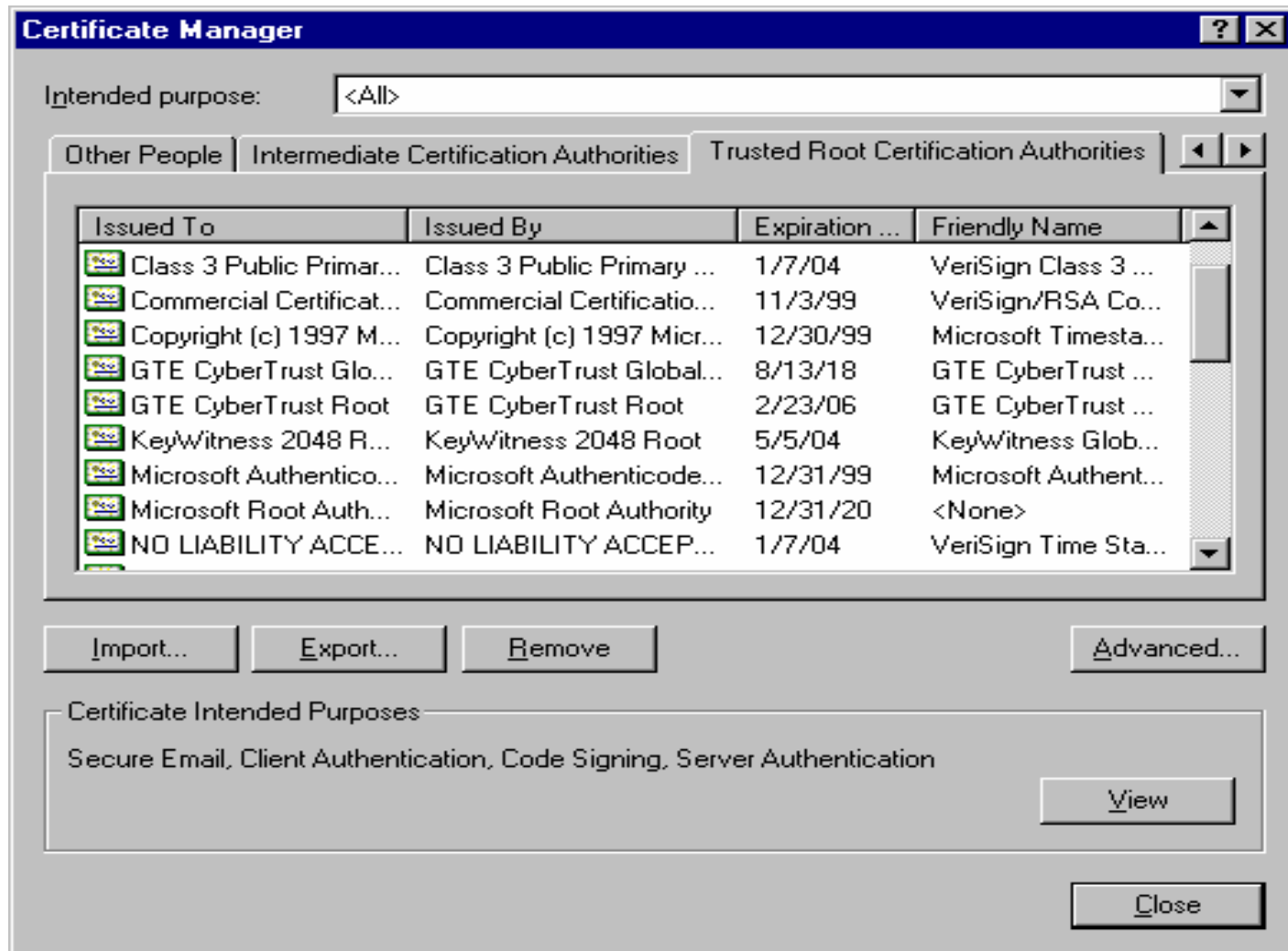
What's wrong with one CA plus RAs?

- Still monopoly pricing
- Still can't ever change CA key
- Still world's security depends on that one CA key never being compromised (or dishonest employee at that organization granting bogus certificates)

Oligarchy of CAs

- Come configured with 80 or so trusted CA public keys (in form of “self-signed” certificates!)
- Usually, can add or delete from that set
- Eliminates monopoly pricing

Default Trusted Roots in IE



What's wrong with oligarchy?

- Less secure!
 - security depends on ALL configured keys
 - naïve users can be tricked into using platform with bogus keys, or adding bogus ones (easier to do this than install malicious software)
 - impractical for anyone to check trust anchors
- Although not monopoly, still favor certain organizations. Why should these be trusted?

CA Chains

- Allow configured CAs to issue certs for other public keys to be trusted CAs
- Similar to CAs plus RAs, but
 - Less efficient than RAs for verifier (multiple certs to verify)
 - Less delay than RA for getting usable cert

Anarchy

- Anyone signs certificate for anyone else
- Like configured+delegated, but user consciously configures starting keys
- Problems
 - won't scale (too many certs, computationally too difficult to find path)
 - no practical way to tell if path should be trusted
 - too much work and too many decisions for user

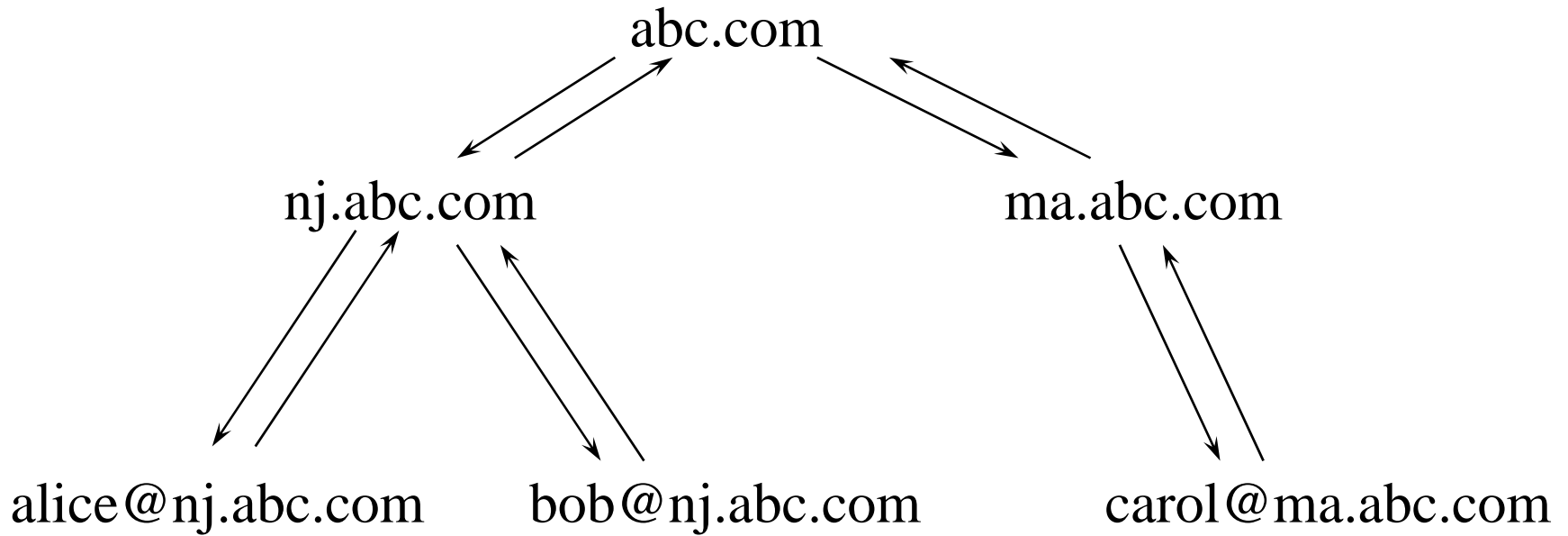
Top Down with Name Subordination

- Assumes hierarchical names
- Each CA only trusted for the part of the namespace rooted at its name
- Can apply to delegated CAs or RAs
- Easier to find appropriate chain
- More secure in practice (this is a sensible policy that users don't have to think about)

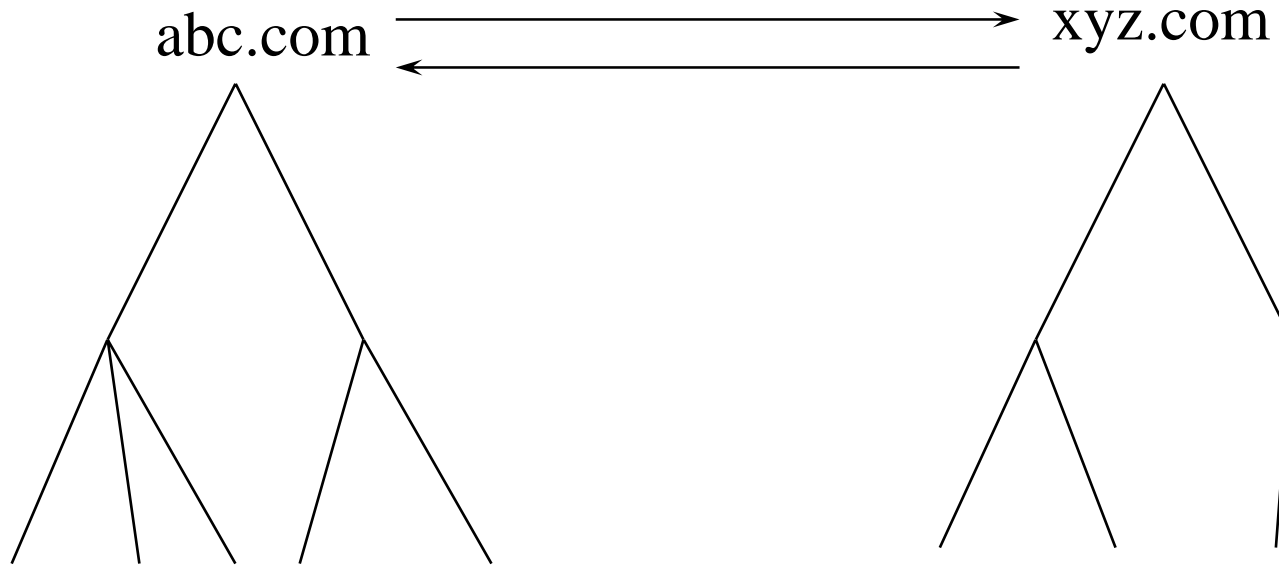
Bottom-Up Model

- Each arc in name tree has parent certificate (up) and child certificate (down)
- Name space has CA for each node
- “Name Subordination” means CA trusted only for a portion of the namespace
- Cross Links to connect Intranets, or to increase security
- Start with your public key, navigate up, cross, and down

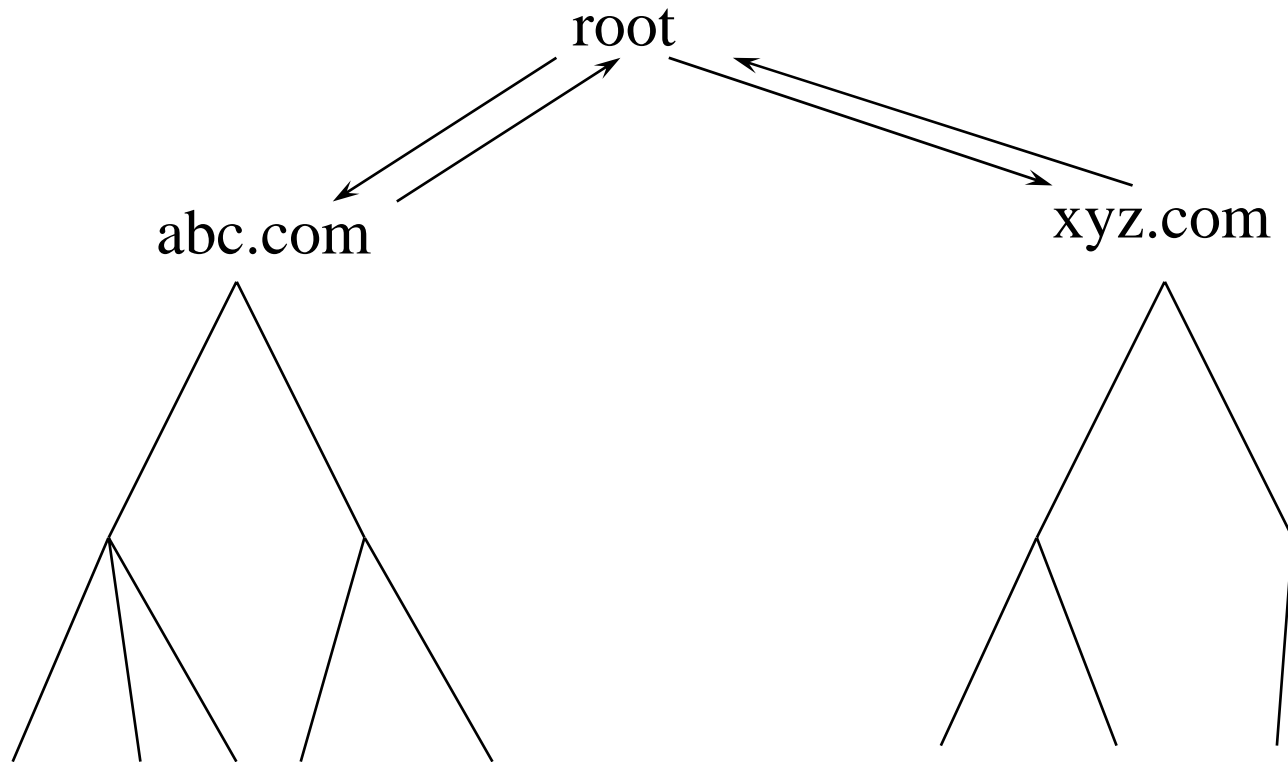
Intranet



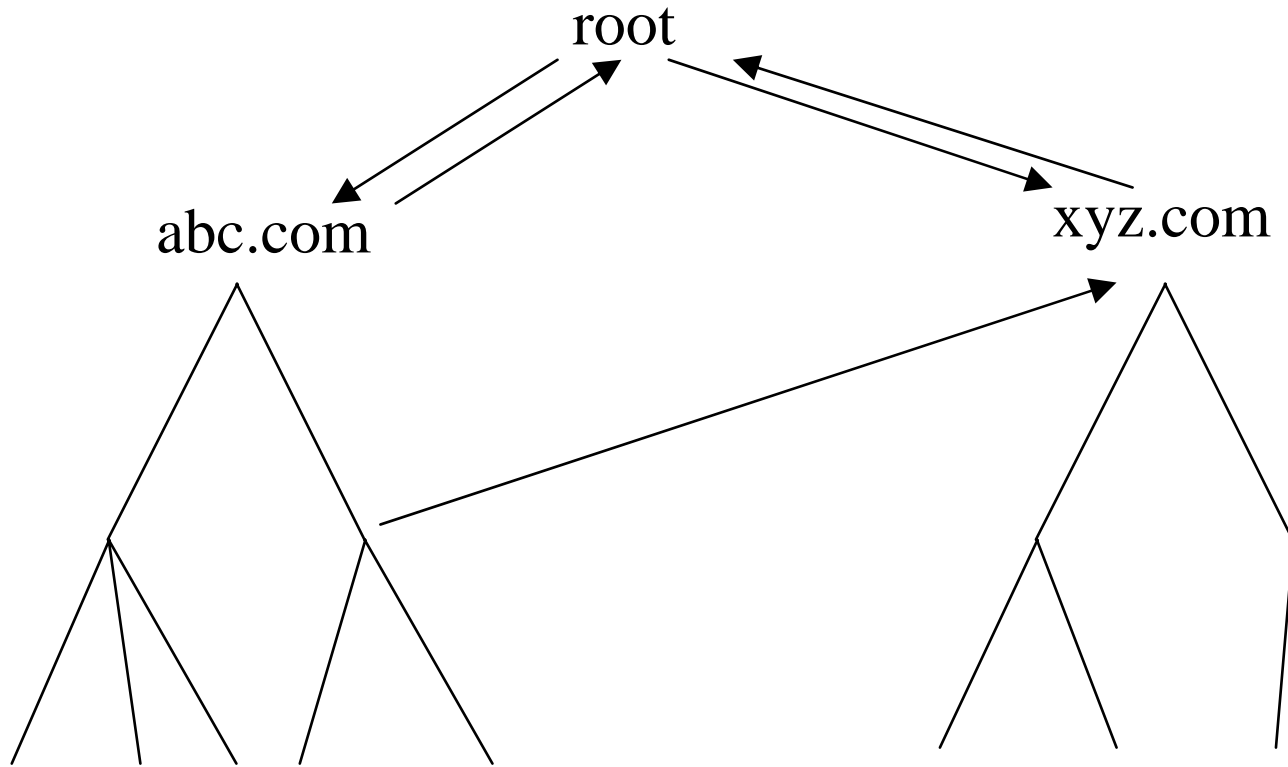
Extranets: Crosslinks



Extranets: Adding Roots



Cross-link for added security



Advantages of Bottom-Up

- For intranet, no need for outside organization
- Security within your organization is controlled by your organization
- No single compromised key requires massive reconfiguration
- Easy configuration: public key you start with is your own

Section: Revocation

Revocation Problem

- Suppose a bad guy learns your password or steals your smart card...
- Notify your KDC and it will stop issuing “tickets”
- Notify your CA and it will give you a new certificate
- How do you revoke your old certificate?

Revocation Problem

- Tickets can have short lifetimes; they can even be “one-use” with nonces
- Certificates have expiration dates, but it is inconvenient to renew them frequently
 - If sufficiently frequent and automated, CA can no longer be off-line
- Supplement certificate expirations with Certificate Revocation Lists (CRLs) or a blacklist server

CRL Enhancement

- Delta CRL
 - Base CRL is complete
 - Delta is changes since base CRL
 - Only need new base CRL if delta CRL gets too big

On-line revocation server

- OCSP (on-line certificate status protocol) is an IETF standard for checking validity status of a cert with on-line server
- I like “client does the work”, getting an “I was not revoked as of 8 AM” cert
 - one interaction amortized over lots of resources
 - can proactively (in background) refresh

Section: What layer?

What layer?

- Link by link: traffic analysis, must trust routers
- End-to-end, real time (e.g., TLS/SSL, IPsec, SSH): protects conversation
- Data at rest: (e.g., S/MIME, XML, PGP): good for backup media, not trusting storage server

IPsec vs. TLS

- IPsec idea: don't change applications or API to applications, just OS
- TLS idea: don't change OS, only change application (if they run over TCP)
- but... unless OS can set security context of application, server applications need to know identity of their clients

IPsec vs. TLS

- IPsec technically superior
 - Rogue packet problem
 - TCP doesn't participate in crypto, so attacker can inject bogus packet, no way for TCP to recover
 - easier to do outboard hardware processing (since each packet independently encrypted)
- TLS easier to deploy
- And unless API changes, IPsec can't pass up authenticated identity

Section: Authorization

All sorts of issues

- Identities, groups, roles
- ACL (access control list associated with resource)
 - Lists Boolean combination of roles, IDs, groups, and what rights they have
- RBAC

Heresy

- These terms are not well defined
- RBAC is often a synonym for “good”

How I'd define IDs, groups, roles

- ID
 - Something you explicitly authenticate as
 - Multiple IDs might go with the same carbon-based life form
- Group
 - Dynamically created
 - Preferably globally unique name
 - Can go on ACLs to make ACLs more scalable
 - Your ID always has privileges of all its groups

IDs, groups, roles, cont'd

- Role
 - Similar to a group, makes ACLs scalable
 - Except you have to explicitly authenticate
 - And roles might be mutually exclusive
 - If you have to authenticate, why is it different from an ID?
 - For auditing, you retain your ID regardless of the role you invoke

How others see roles

- A name like “administrator” that is local to a resource

Proposed Constitutional Amendment

- Congress shall pass no law restricting the size of numbers which may be multiplied together; or the number of times by which a number be multiplied by itself; or the modulus by which a number be reduced.

Conclusions

- *Until a few years ago, you could connect to the Internet and be in contact with hundreds of millions of other nodes, without giving even a thought to security. The Internet in the '90's was like sex in the '60's. It was great while it lasted, but it was inherently unhealthy and was destined to end badly. I'm just really glad I didn't miss out again this time.* —Charlie Kaufman