Network Security Protocols: A Tutorial

Radia Perlman
May 2005

(radia.perlman@sun.com)

Purpose of this tutorial

- A quick intro into a somewhat scary field
- A description of what you need to know vs what you can trust others to do
- A description of the real problems
- “How to build an insecure system out of perfectly good cryptography”

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
- 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

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)

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
- losing data vs disclosure (copies of keys)
- denial of service vs preventing intrusion

Cryptography

- Crypto
  - 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

- encrypt=f(S, plaintext)=ciphertext
- decrypt=f(S, ciphertext)=plaintext
- authentication: send f(S, challenge)
- integrity check: f(S, msg)=X
- verify integrity check: f(S, X, 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”
- Even 64 bits is daunting to search through
- Faster computers work to the benefit of the good guys!

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(priv key, data)
  - verified as f(public key, data, 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

Enough crypto to impress a date

- Secret key and hash algorithms just look like a messy way to mangle bits
- The public key algorithms, though, are quite understandable
- Based on some particular math problem we assume is hard
- I'll explain Diffie-Hellman

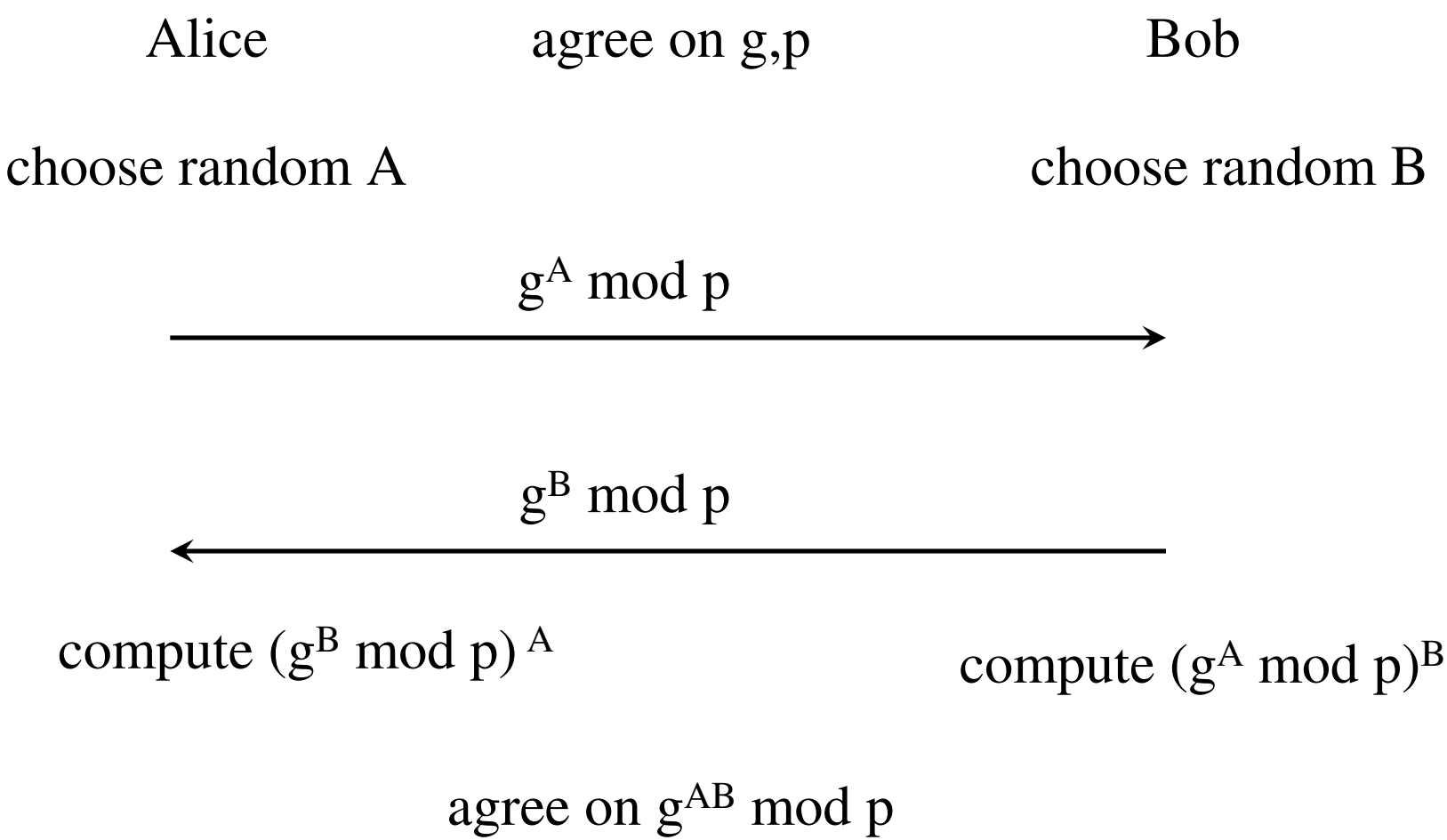
An Intuition for Diffie-Hellman

- Allows two individuals to agree on a secret key, even though they can only communicate in public
- Alice chooses a private number and from that calculates a public number
- Bob does the same
- Each can use the other's public number and their own private number to compute the same secret
- An eavesdropper can't reproduce it

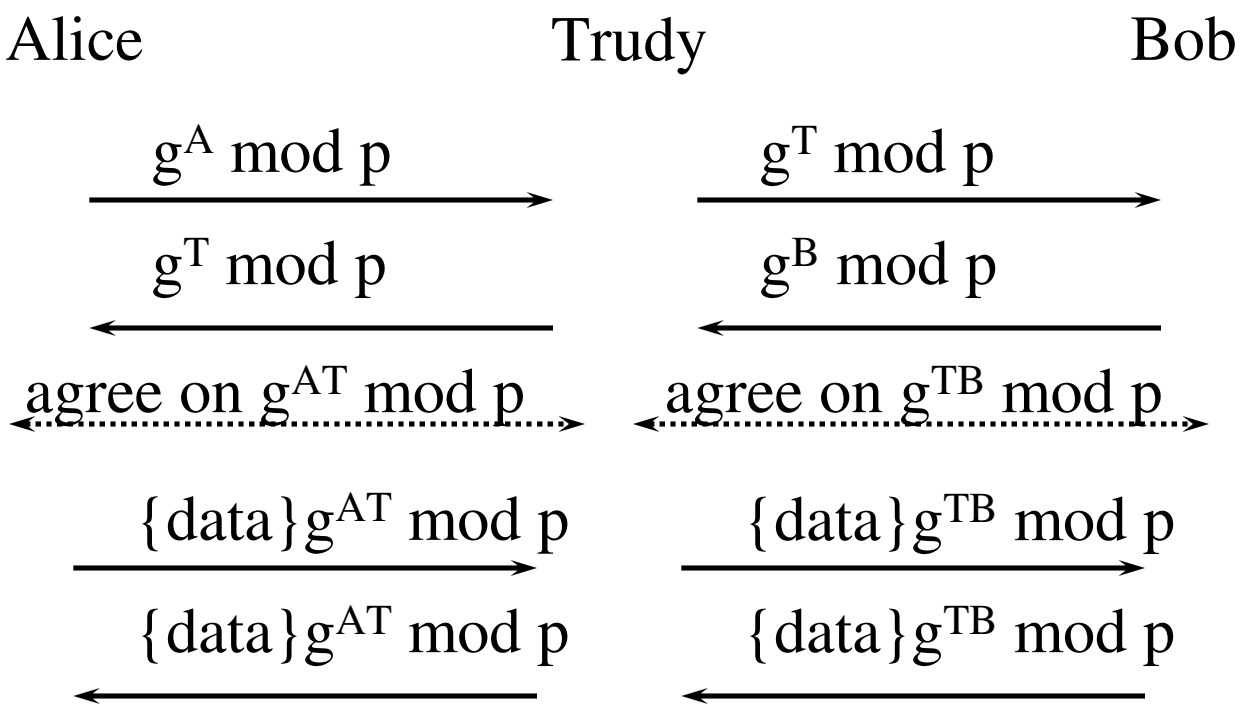
Why is D-H Secure?

- We assume the following is hard:
- Given g, p, and g^X mod p, what is X?

Diffie-Hellman



Man in the Middle



Signed Diffie-Hellman (Avoiding Man in the Middle)

Alice Bob

choose random A choose random B

$[g^A \bmod p]$ signed with Alice's Private Key

$[g^B \bmod p]$ signed with Bob's Private Key

verify Bob's signature verify Alice's signature

agree on $g^{AB} \bmod p$

If you have keys, why do D-H?

- “Perfect Forward Secrecy” (PFS)
- Prevents me from decrypting a conversation even if I break into both parties after it ends (or if private key is escrowed)
- Ex. non-PFS: A chooses key S, encrypts it with B’s public key and sends it to B (SSL)
- IESG strongly encourages PFS in protocols

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 Secret Key Algorithms

- DES (old standard, 56-bit key, slow)
- 3DES: fix key size but 3 times as slow
- RC4: variable length key, “stream cipher” (generate stream from key, XOR with data)
- AES: replacement for DES, will probably take over

Popular Public Key Algorithms

- RSA: nice feature: public key operations can be made very fast, but private key operations will be slow. Patent expired.
- ECC (elliptic curve crypto): smaller keys, so faster than RSA (but not for public key ops). Some worried about patents

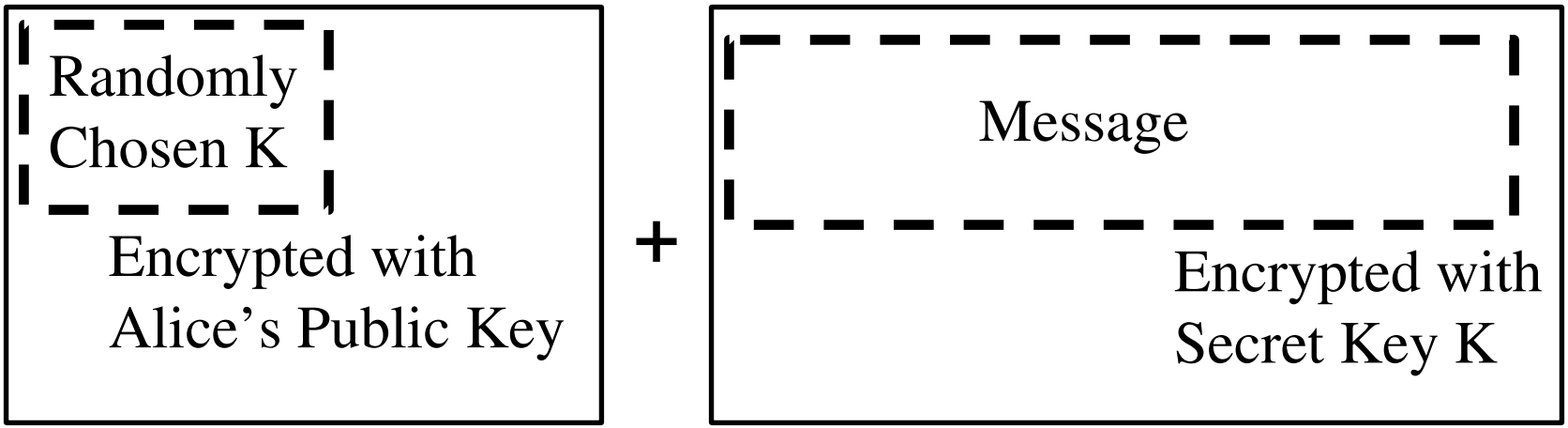
Hash stuff

- Most popular hash today SHA-1 (secure hash algorithm)
- Older ones (MD2, MD4, MD5) still around, but “broken”
- Popular secret-key integrity check: hash together key and data
- One popular standard for that within IETF: HMAC

Hybrid Encryption

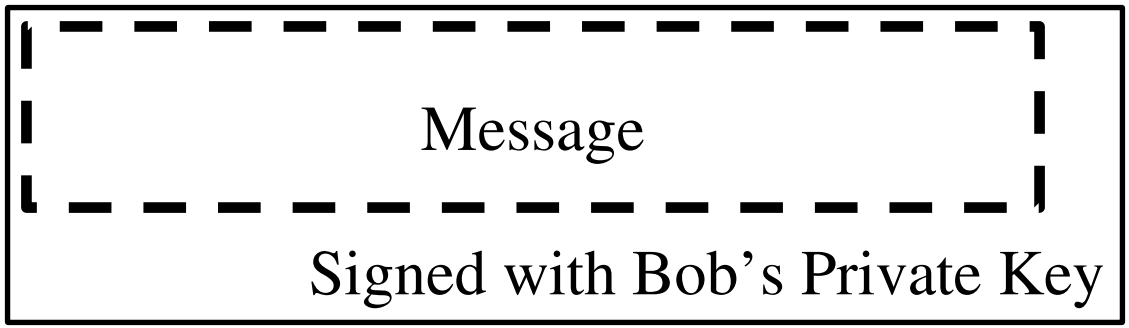
Instead of:

Use:



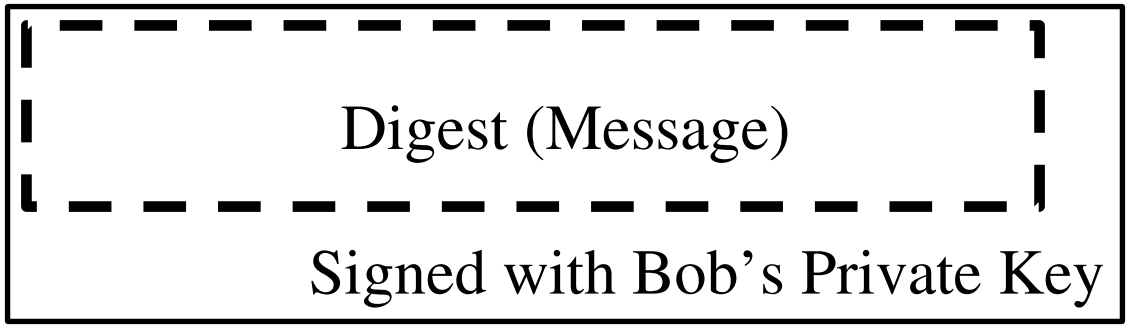
Hybrid Signatures

Instead of:

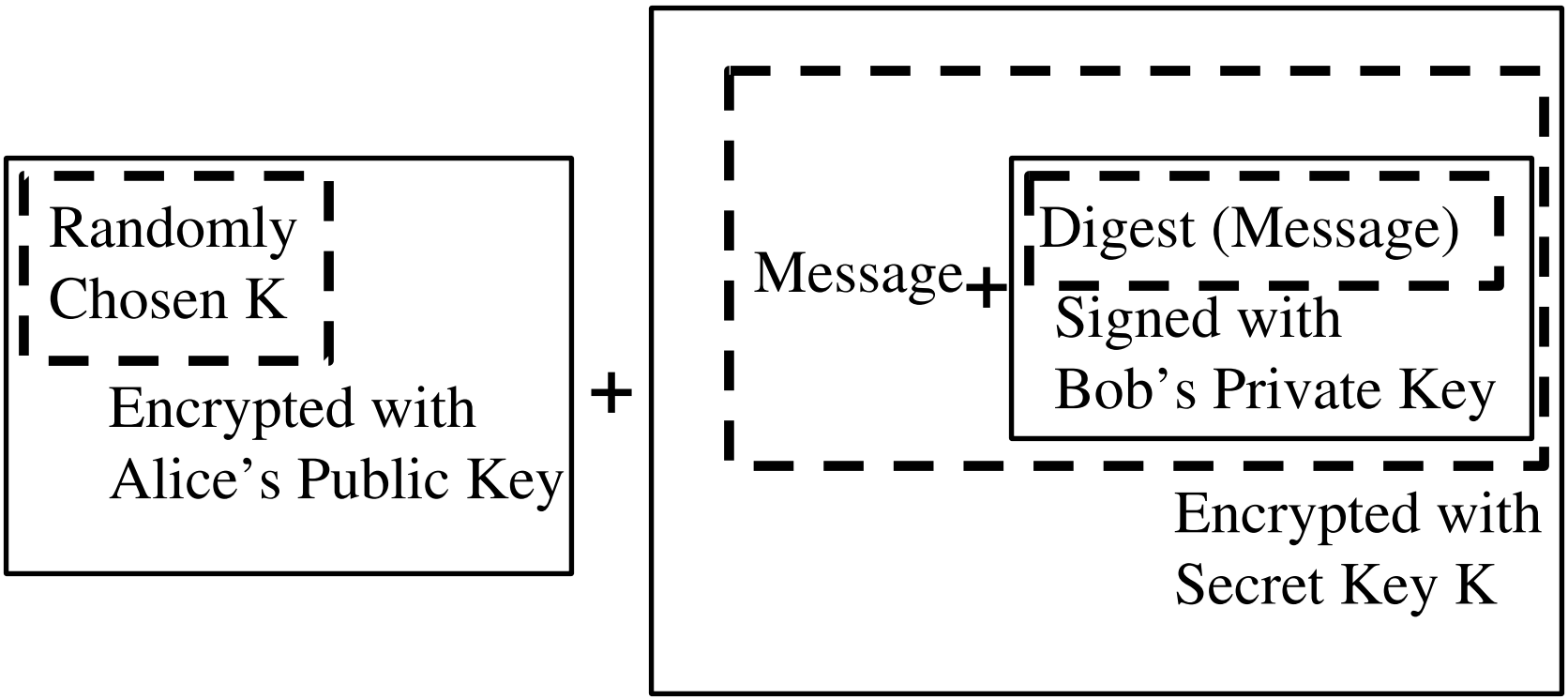


Use:

Message +



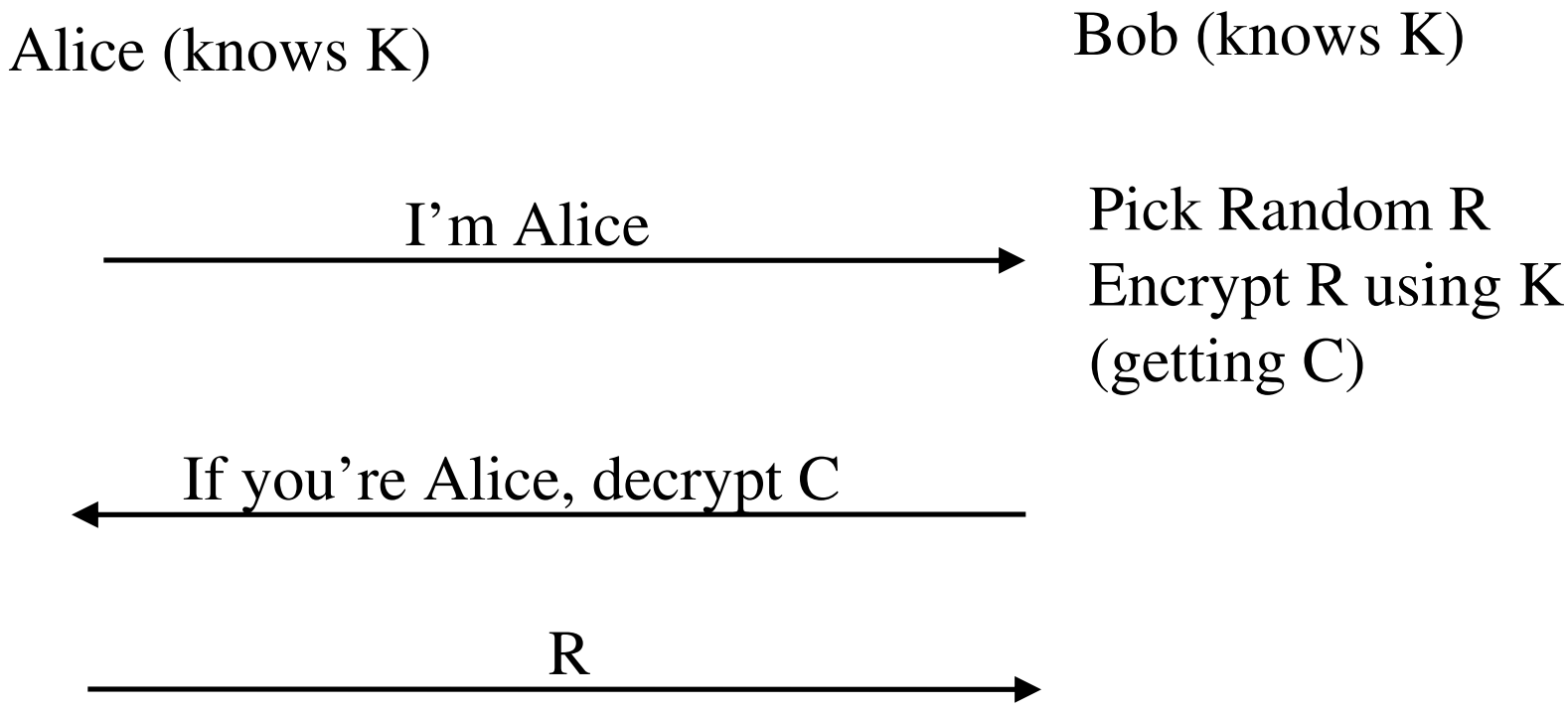
Signed and Encrypted Message



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



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

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

Authenticating people

- What you know
- What you have
- What you are

What You Know...

- Mostly this means passwords
  - Subject to eavesdropping
  - Subject to on-line guessing
  - Subject to off-line guessing

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

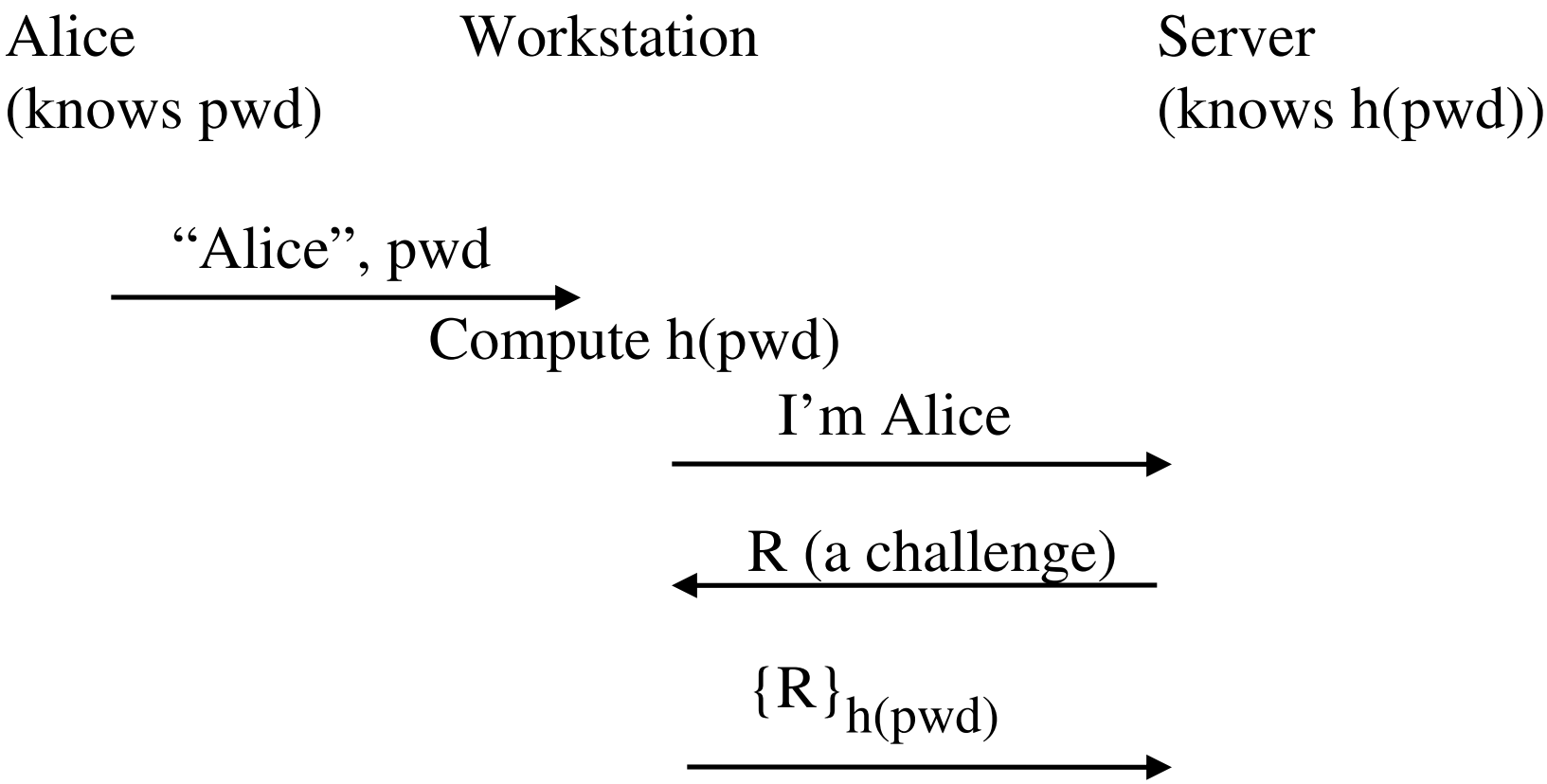
Off-Line Password Guessing

- 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

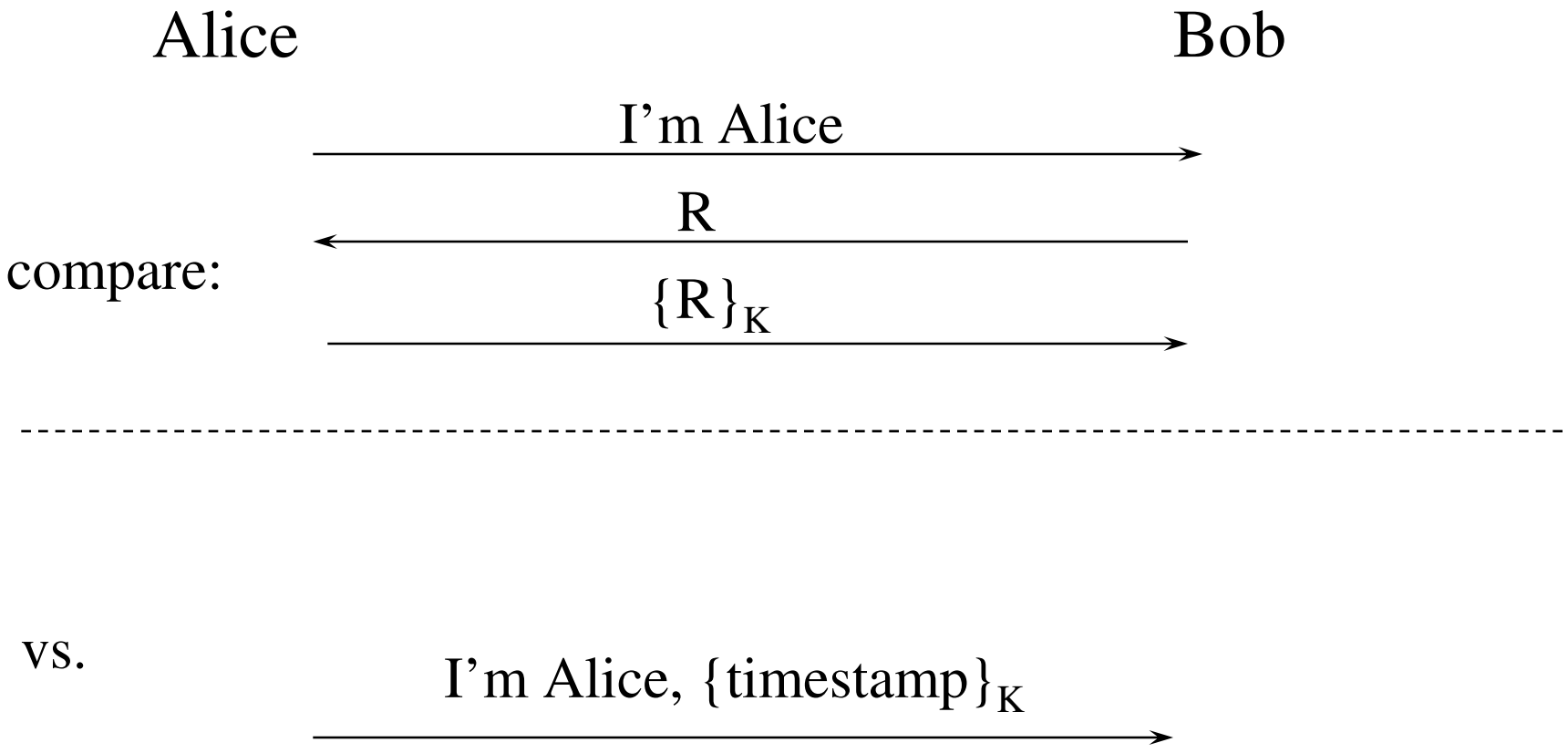
Off-line attack possible



Cryptographic Handshakes

- Once keys are known to two parties, need a handshake to authenticate
- Goals:
  - Mutual authentication
  - Immune from replay and other attacks
  - Minimize number of messages
  - Establish a session key as a side effect

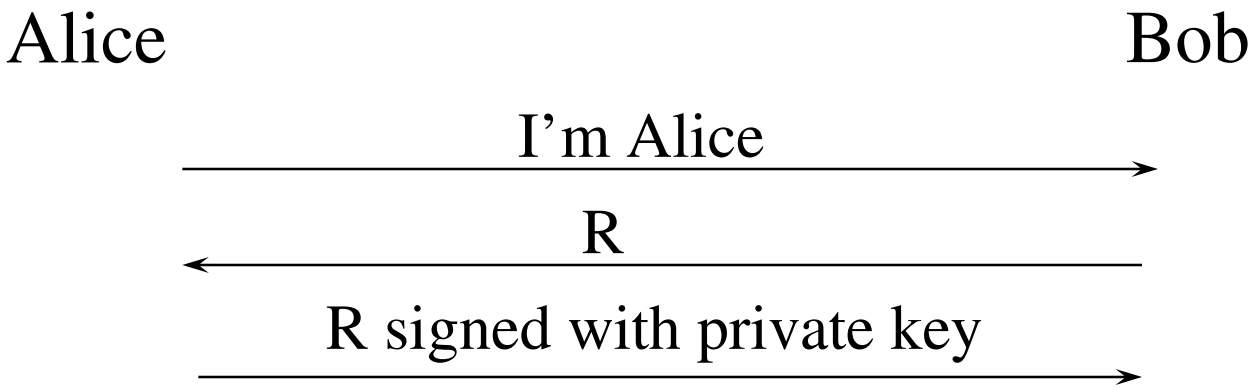
Challenge/Response vs. Timestamp



Challenge/Response vs. Timestamp

- Second protocol saves messages, fits more easily into existing protocols that expect passwords
- First protocol does not require synchonized clocks
- Second protocol must keep a list of unexpired timestamps to avoid replay

Pitfalls with Public Key



This might trick Alice into signing something, or possibly decrypting something

Eavesdropping/Server Database Stealing

- pwd-in-clear, if server stores h(pwd), protects against database stealing, but vulnerable to eavesdropping
- Standard challenge/response, using K=h(pwd), foils eavesdropping but K is pwd-equivalent so server database vulnerable
- Lamport's hash solves both

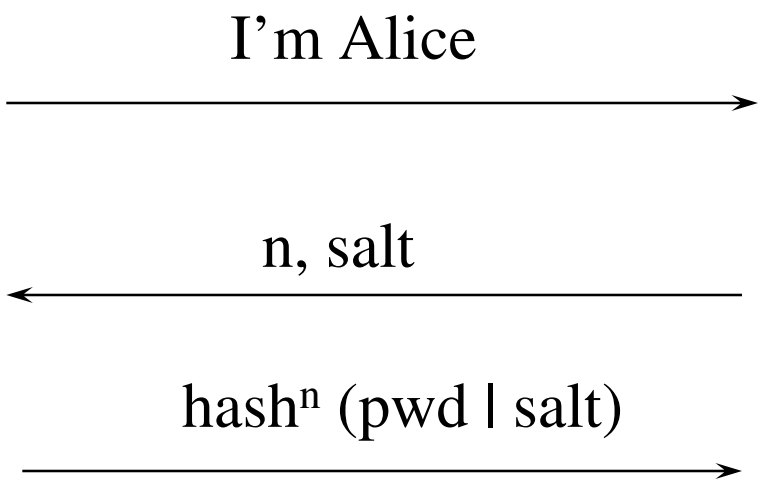
Salt

- Protects a database of hashed passwords
- Salt is non-secret, different for each user
- Store hash(pwd, salt)
- Users with same pwd have different hashes
- Prevents intruder from computing hash of a dictionary, and comparing against all users

Lamport's Hash (S/Key)

Bob's database holds:
n, salt, $hash^{n+1}$(pwd | salt)

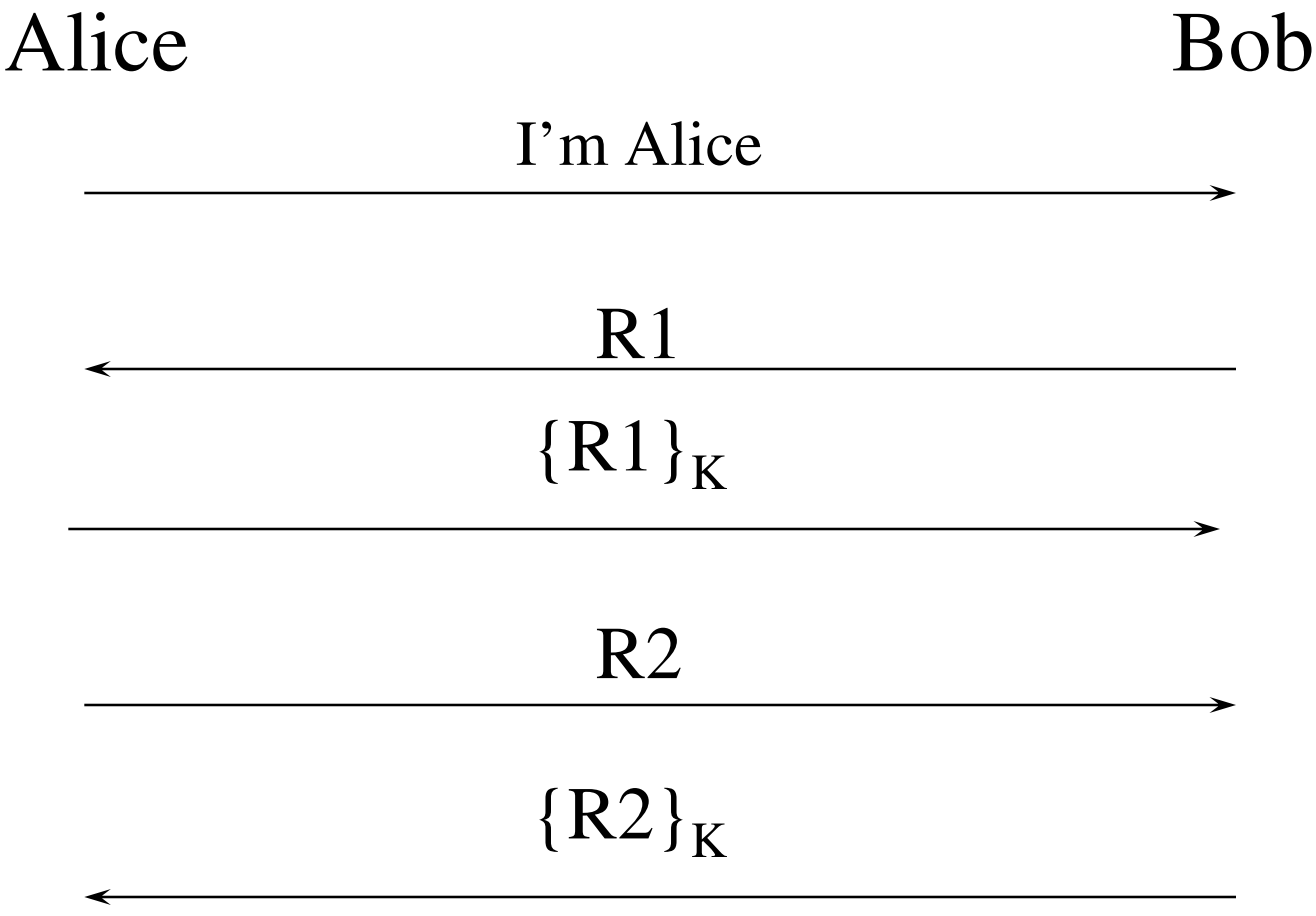
Alice Bob



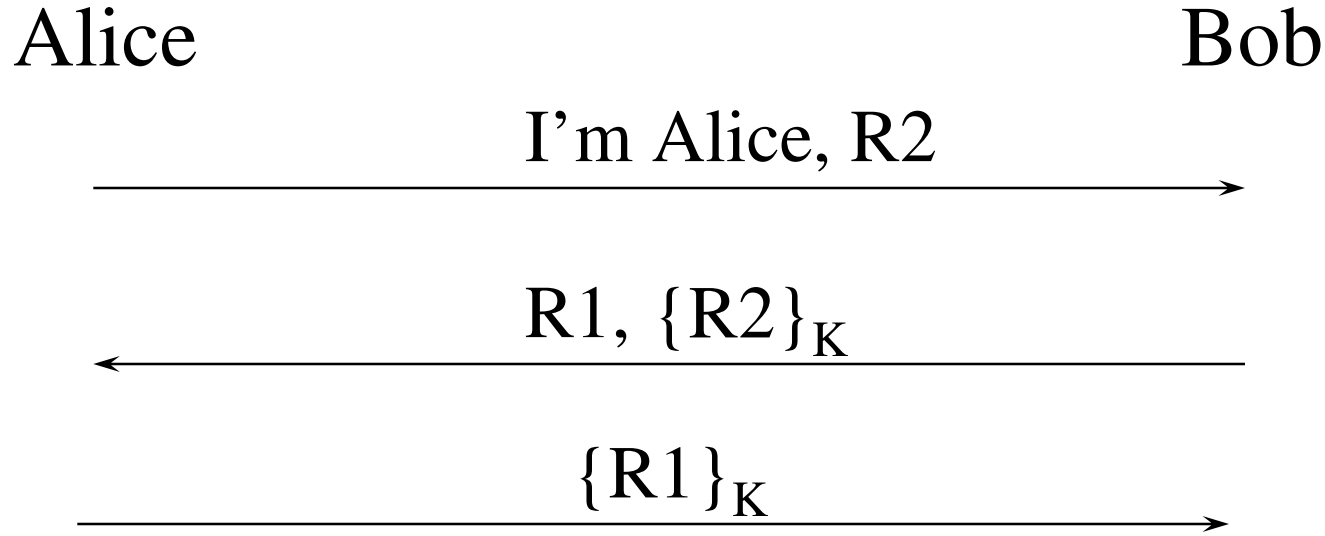
Lamport's Hash (S/Key)

- Offers protection from eavesdropping and server database reading without public key cryptography
- No mutual authentication
- Only finitely many logins
- Small n attack: someone impersonates Bob

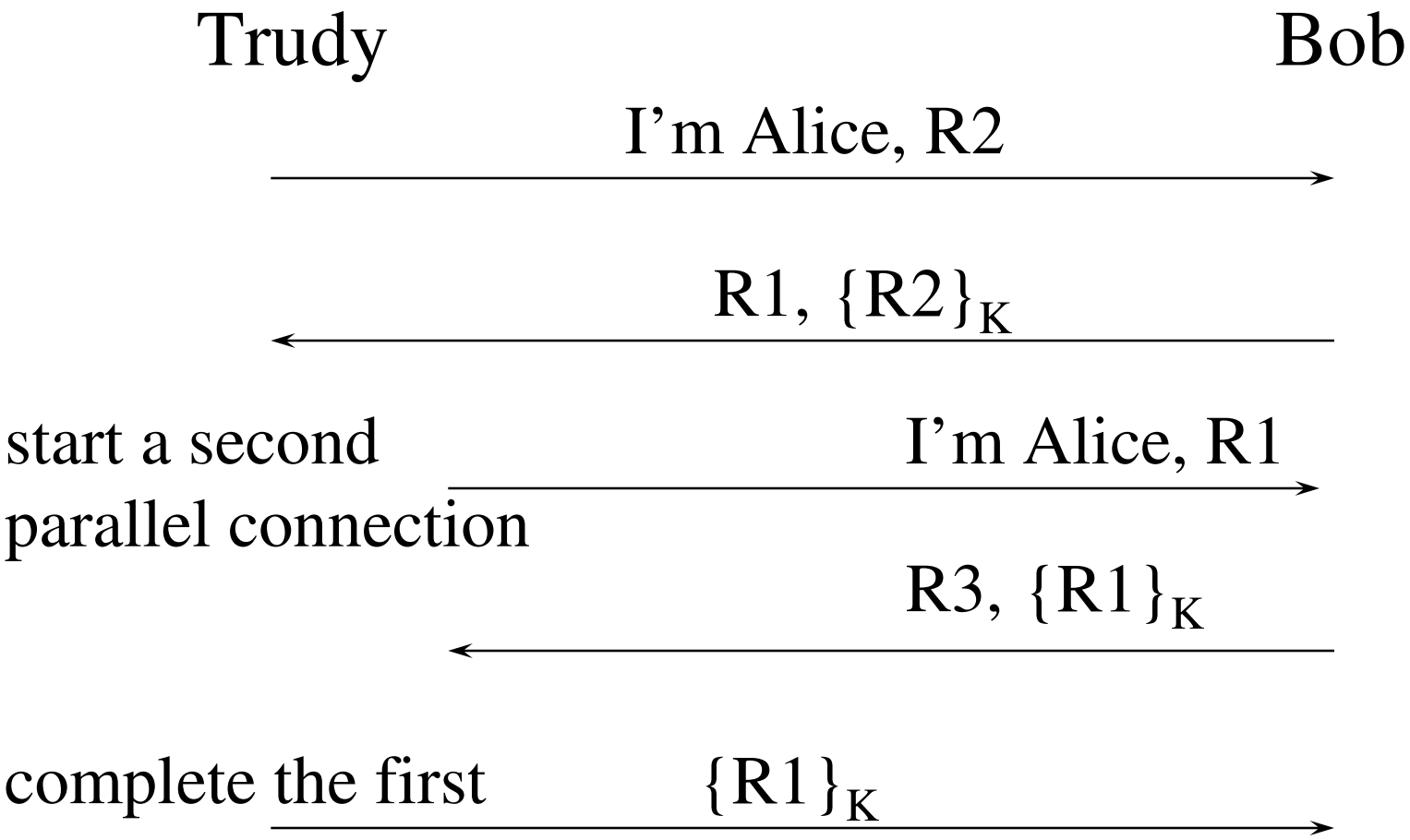
Mutual Authentication



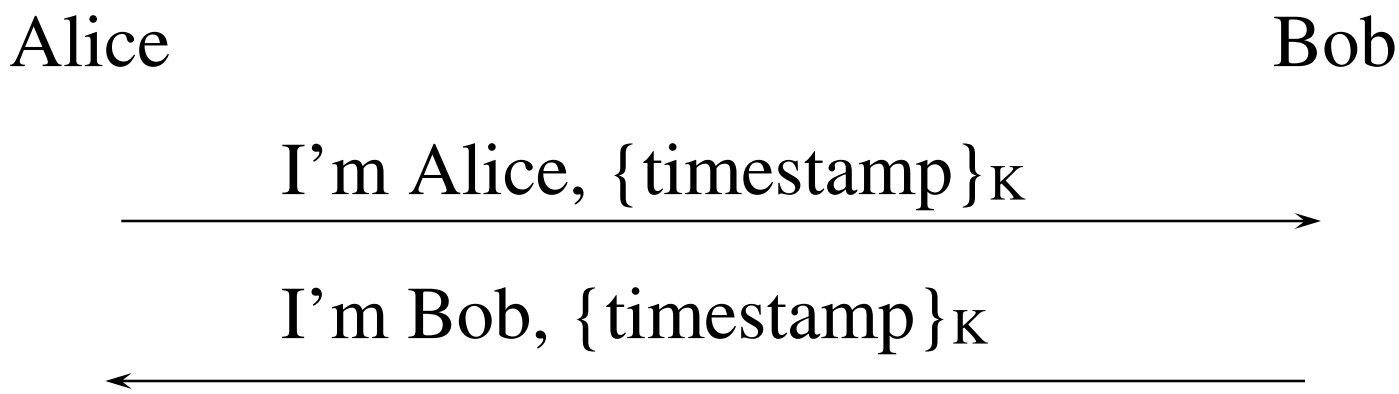
More Efficient Mutual Authentication



Reflection Attack



Timestamp Based Mutual Authentication

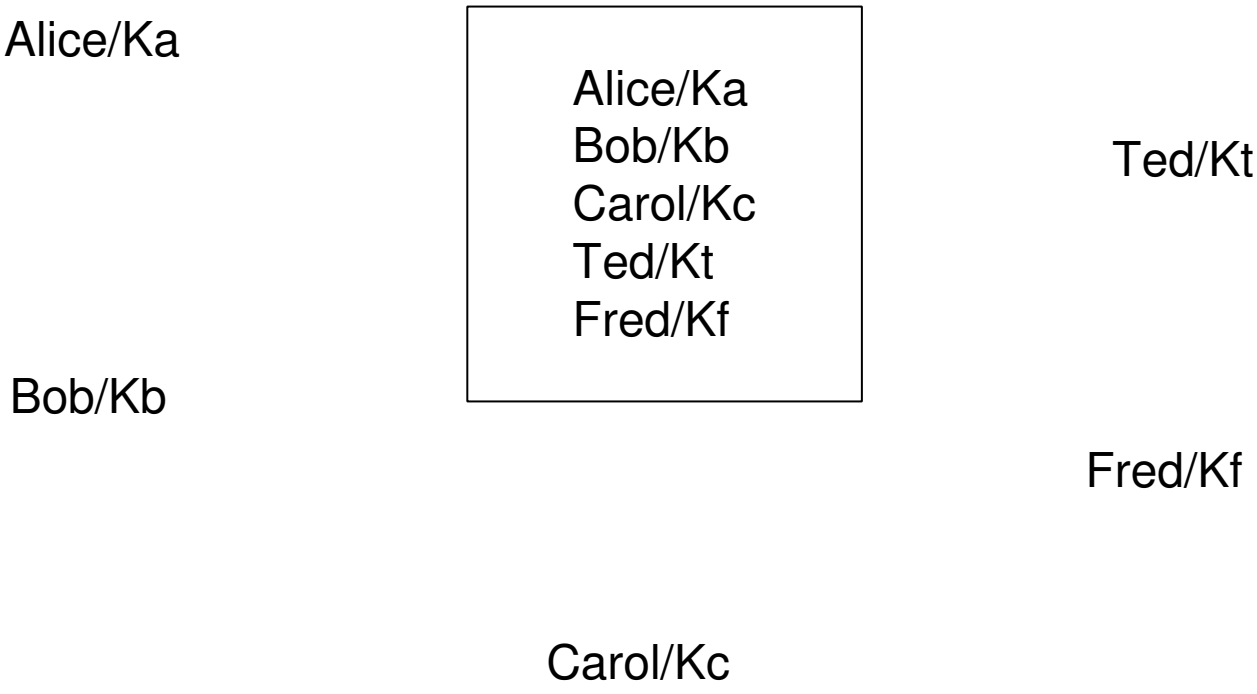


Two messages instead of three
Must assure Bob's timestamp is different

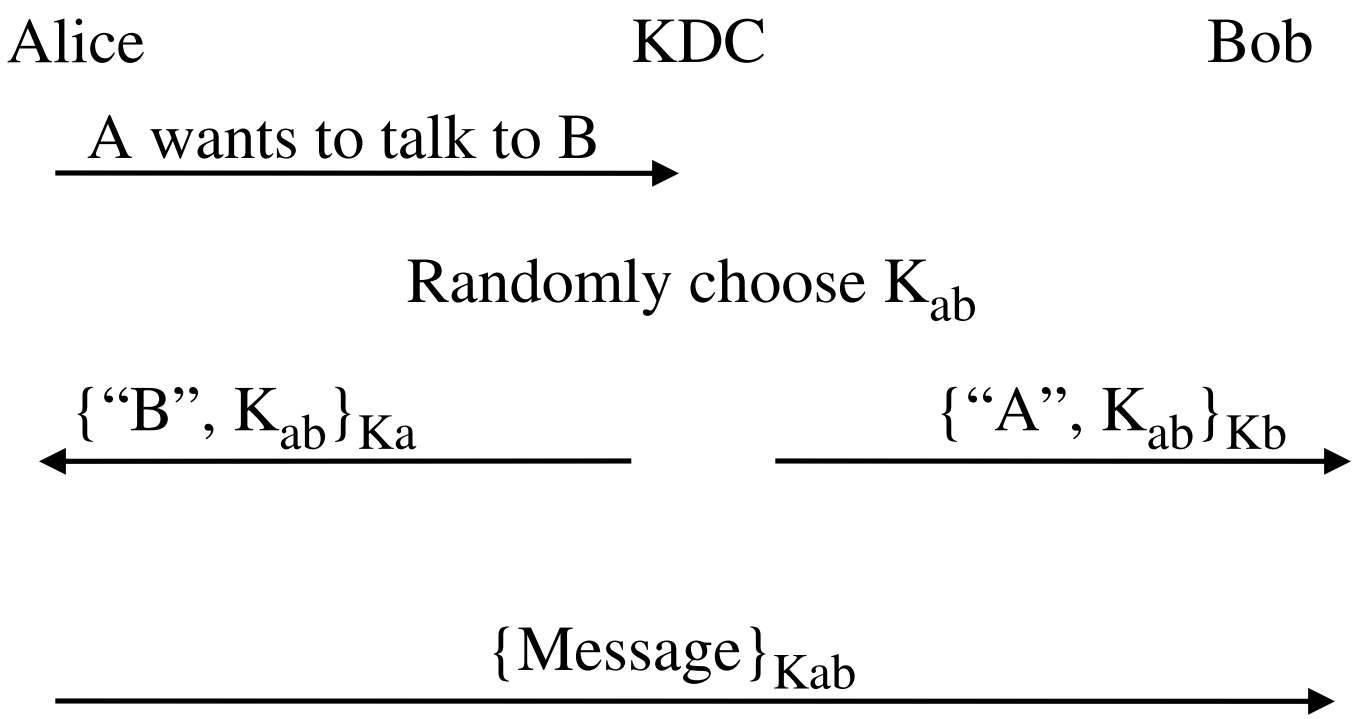
Key Distribution - Secret Keys

- Could configure n^2 keys
- Instead use Key Distribution Center (KDC)
  - Everyone has one key
  - The KDC knows them all
  - The KDC assigns a key to any pair who need to talk
- This is basically Kerberos

KDC


Alice/Ka
Alice/Ka
Bob/Kb
Carol/Kc
Ted/Kt
Fred/Kf
Ted/Kt
Bob/Kb
Fred/Kf
Carol/Kc

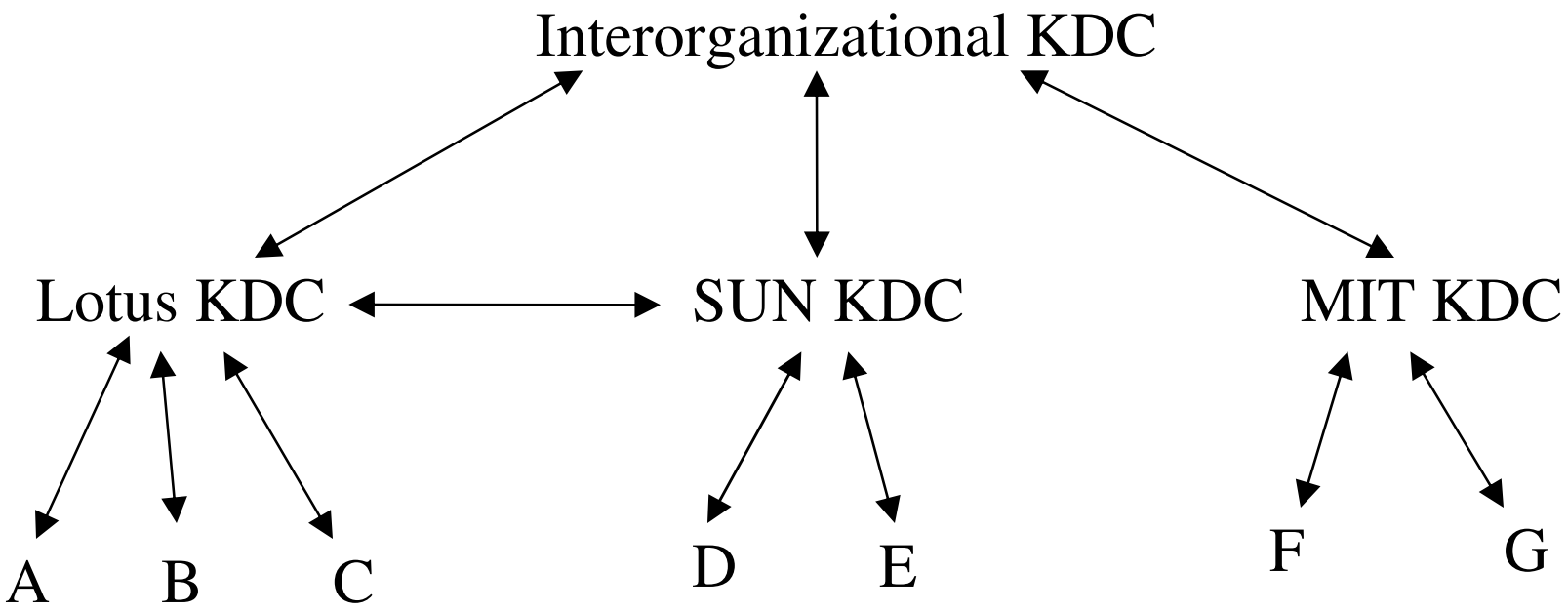
Key Distribution - Secret Keys



KDC Realms

- KDCs scale up to hundreds of clients, but not millions
- There's no one who everyone in the world is willing to trust with their secrets
- Can do cross-realm authentication, if KDCs trust each other
- But Kerberos protocol doesn't say how to find the path

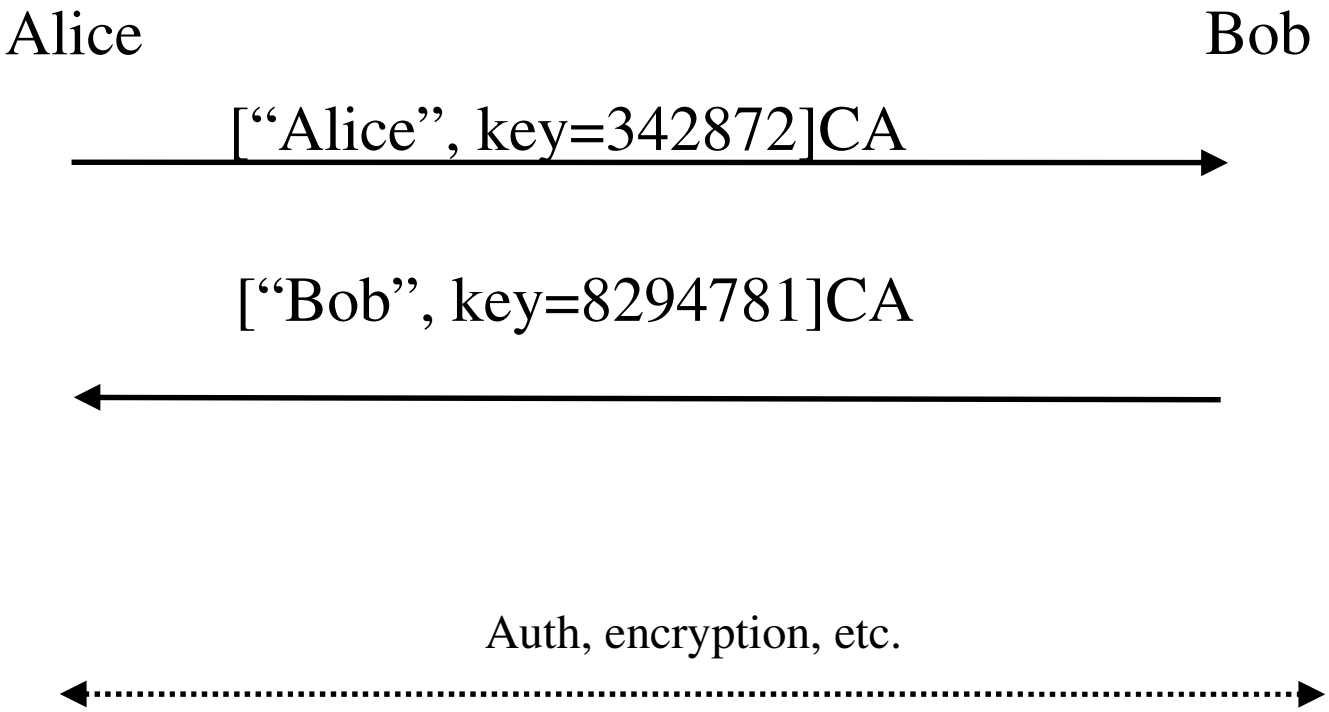
KDC Realms



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 - Public Keys



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

Revocation

- What if someone steals your credit card?
  - depend on expiration date?
  - publish book of bad credit cards (like CRL mechanism …cert revocation list)
  - have on-line trusted server (like OCSP … online certificate status protocol)

CRL mechanism

- CRL must be published periodically, *even if no new revocations have taken place*
- Enchancement: delta CRL
  - these are changes since base CRL, Jan 3, 2 PM
  - Only need to issue new base CRL if delta CRL gets large

Strategies for PKI 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

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

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?

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

Important idea

- CA trust shouldn't be binary: "is this CA trusted?"
- Instead, a CA should only be trusted for certain certificates
- Name-based seems to make sense (and I haven't seen anything else that does)

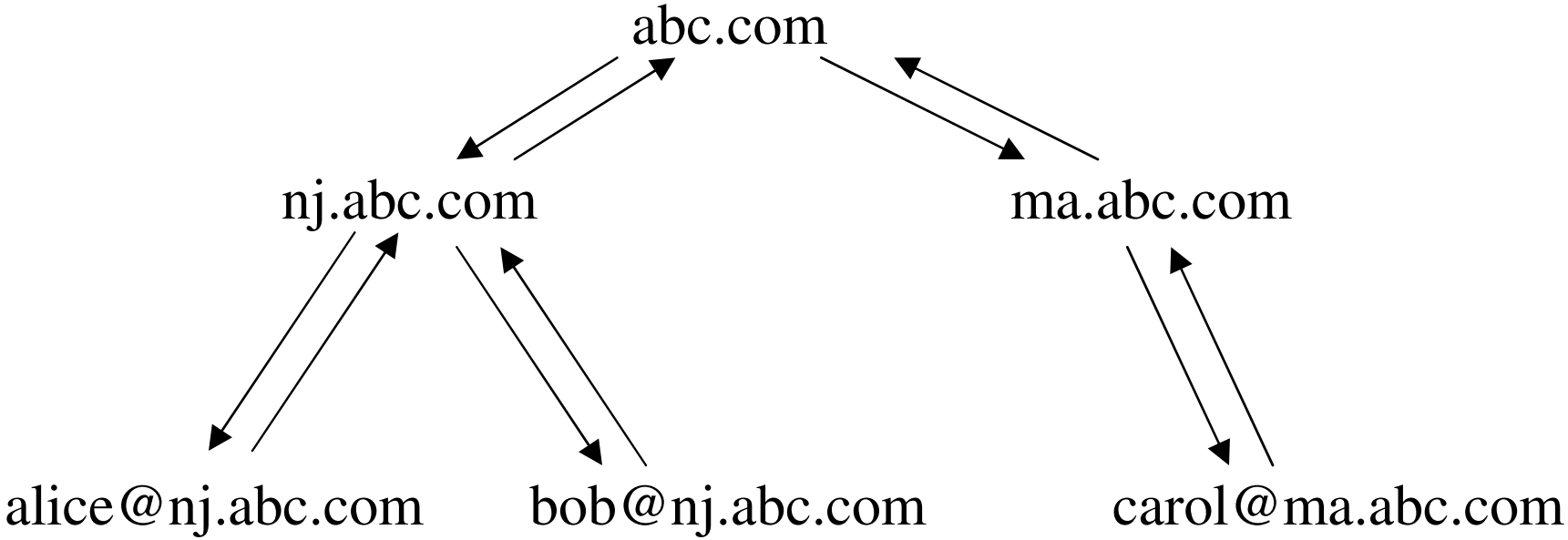
Top Down with Name-based policies

- Assumes hierarchical names
- Each CA only trusted for the part of the namespace rooted at its name
- Easy to find appropriate chain
- This is a sensible policy that users don't have to think about
- But: Still monopoly at top, since everyone needs to be configured with that key

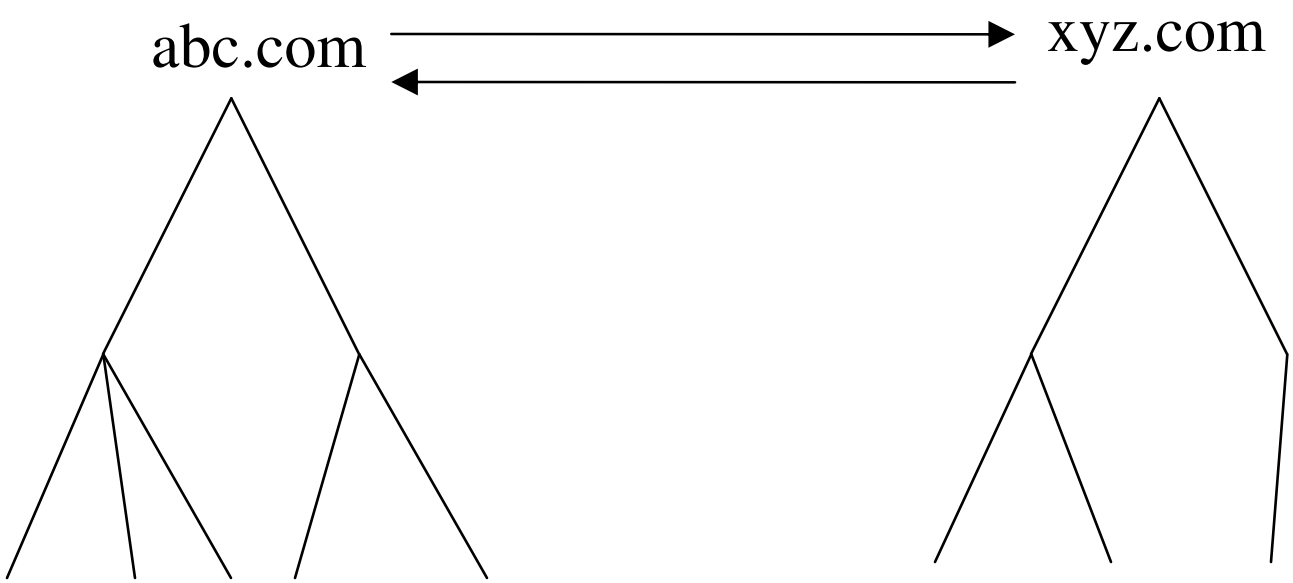
Bottom-Up Model

- Each arc in name tree has parent certificate (up) and child certificate (down)
- Name space has CA for each node
- Cross Links to connect Intranets, or to increase security
- Start with your public key, navigate up, cross, and down

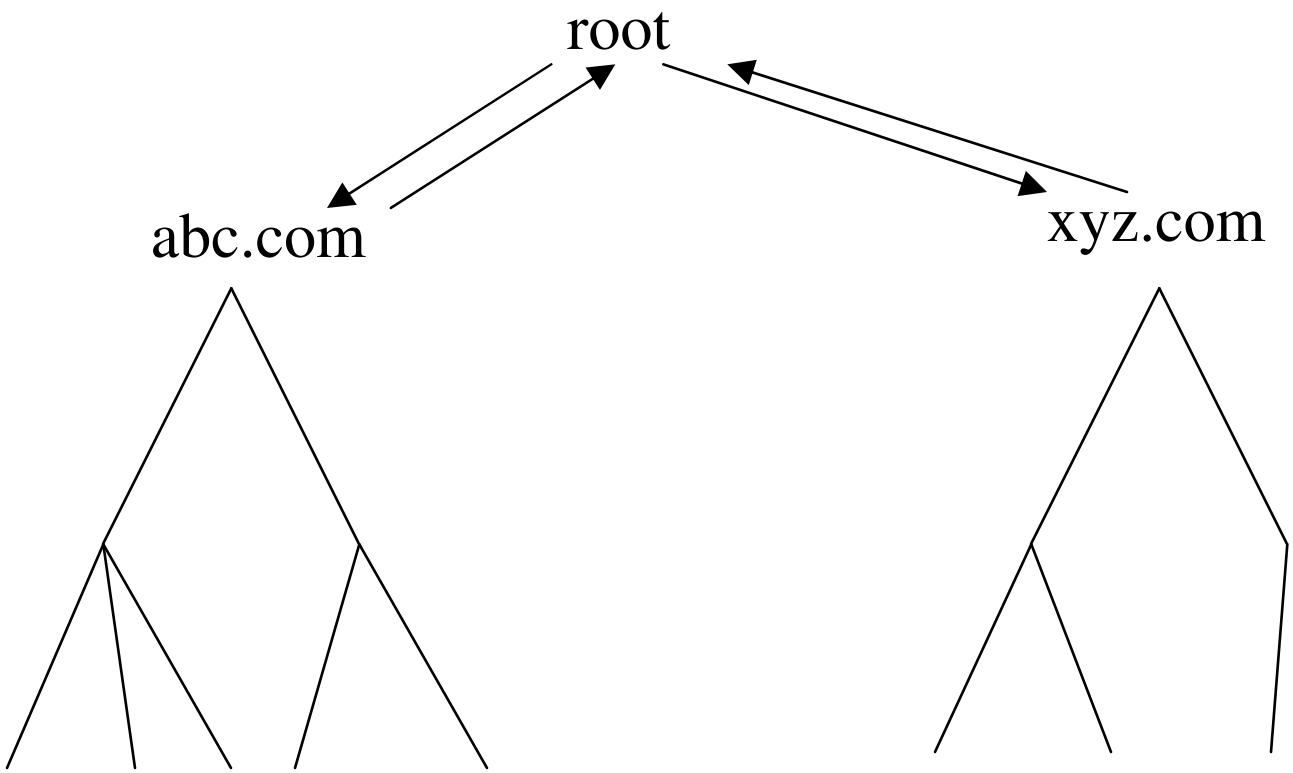
Intranet


abc.com
nj.abc.com
ma.abc.com
alice@nj.abc.com
bob@nj.abc.com
carol@ma.abc.com

Extranets: Crosslinks



Extranets: Adding Roots



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

What layer?

- Layer 2
  - protects link hop-by-hop
  - IP headers can be hidden from eavesdropper (protects against “traffic analysis”)
- Layer 3/4 (more on next slide)
  - protects end-to-end real-time conversation
- Upper layer (e.g., PGP, S/MIME, XML-DSIG, XML-encryption)
  - protects msgs. Store/forward, not real-time

“Key Exchange”

- Mutual authentication/session key creation (create “security association”)
- Good to cryptographically protect entire session (not just initial authentication)
- Good to have new key for each session
- Examples
  - SSL/TLS or Secure Shell (“layer 4”)
  - IPsec (“layer 3”)

Layer 3 vs layer 4

- Layer 3 idea: don’t change applications or API to applications, just OS
- layer 4 idea: don’t change OS, only change application. They run on top of layer 4 (TCP/UDP)

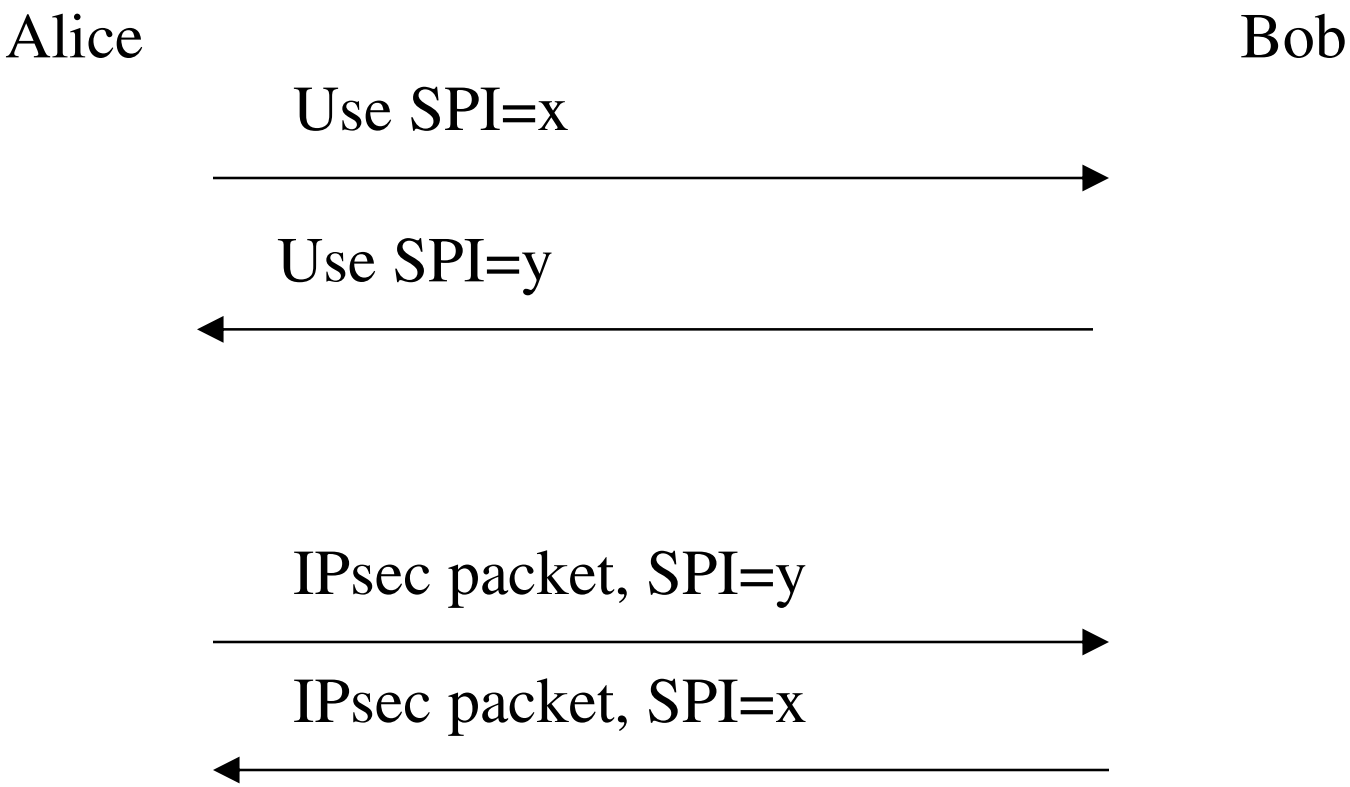
AH / ESP

- extra header between layers 3 and 4 (IP and TCP) to give dest enough info to identify “security association”
- AH does integrity only - includes source and destination IP addresses
- ESP does encryption and integrity protection

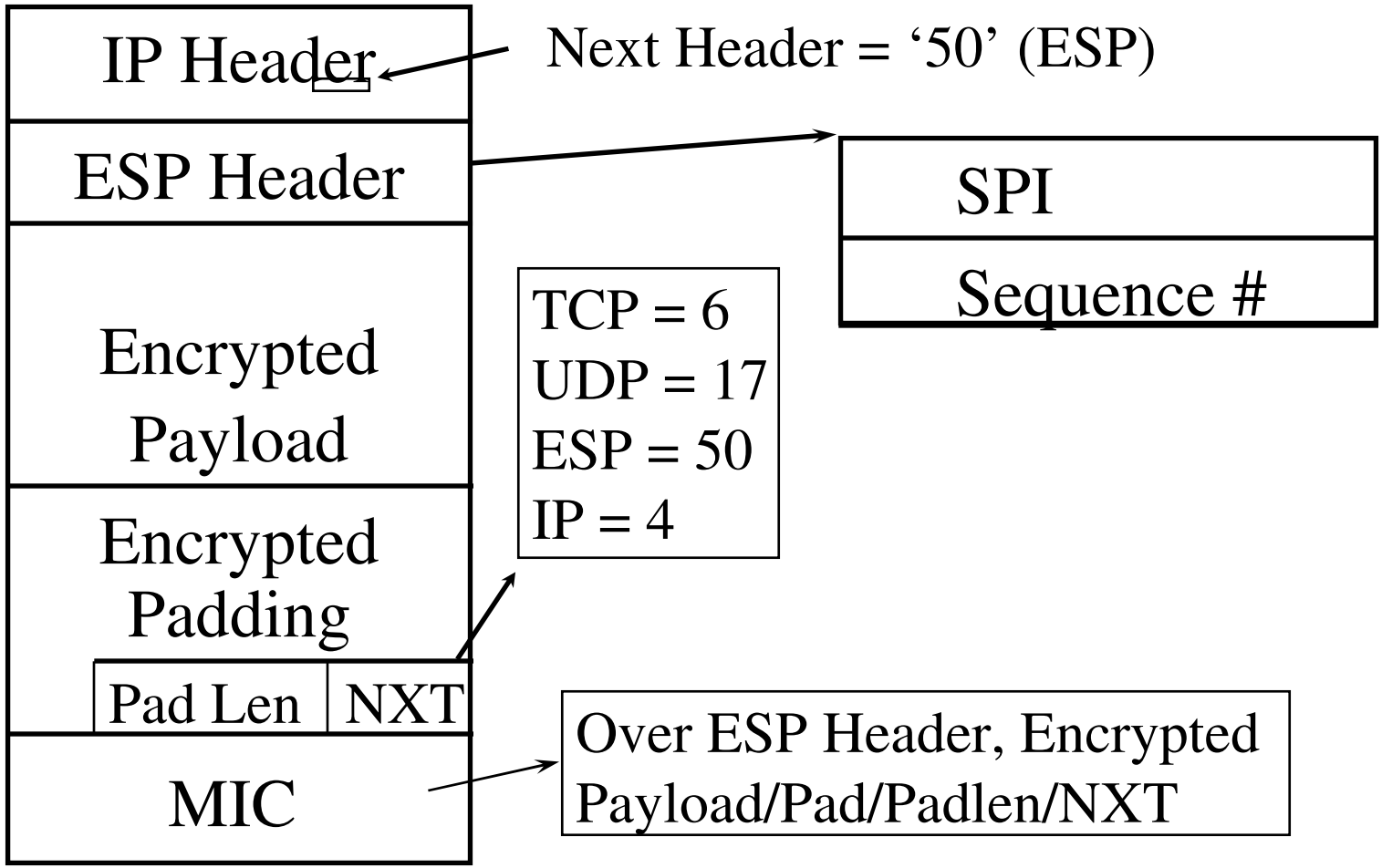
Security Association

- First Alice and Bob establish a “security association” (an SA)
  - Session key
  - Crypto algorithms
  - Identity of other end
  - Sequence number
  - SPI chosen by other end
  - Etc.

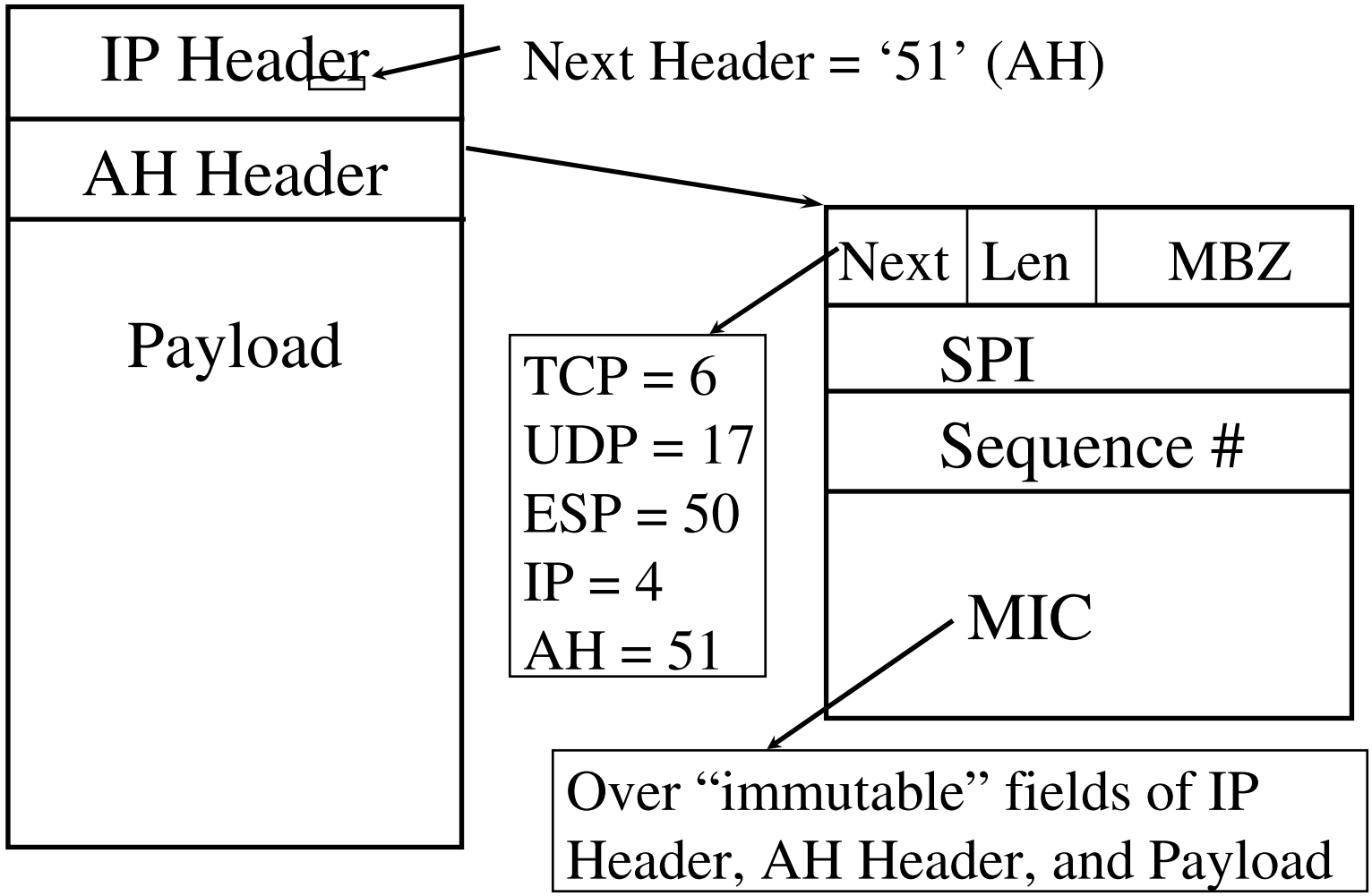
SPI (“security parameters index”)



ESP
Encapsulating Security Payload



AH (Authentication Header)



Layer 3 vs layer 4

- layer 3 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)
- layer 4 easier to deploy
- And unless API changes, layer 3 can't pass up authenticated identity

What's going on in IETF Security Area

- Kerberos
- PKIX (certificate format) (see next slide)
- S/MIME, PGP
- IPsec, SSL/TLS, Secure Shell
- SASL (syntax for negotiating auth protocol)
- DNSSEC (public keys, signed data in DNS)
- sacred (downloading credentials)

PKIX

- Based on X.509 (!)
- Two issues:
  - ASN.1 encoding: big footprint for code, certs bigger
  - names not what Internet applications use! So …
    - ignore name, or
    - DNS name in alternate name, or
    - CN=DNS name, or
    - DC=

PKI, cont'd

- PKIX is used (more or less successfully) in SSL/TLS, IPsec, and S/MIME
- Names problematic no matter what
  - What if there are several John Smith's at the organization?
  - Just an example of the deeper issues beyond crypto, provably secure handshakes, etc.

Is PKI dead?

- We use it every day with SSL
- Maybe people mean we don’t have user certificates
- This stuff shouldn’t be hard

How it ought to work

- "The network is the computer"
- Create an account (username, pwd)
  - System calculates a key pair
  - Certifies the public key, puts it in the directory
  - Encrypts the private key with the pwd, gives it to the user or makes it downloadable
- User types name/pwd, obtains private key and certificate, does single signon

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