IETF Security Tutorial

Radia Perlman
Intel Labs
August 2012

(radia@alum.mit.edu)

Why an IETF Security Tutorial?

- Security is important in all protocols; not just protocols in the security area
- IETF specs mandated to have a “security considerations” section
- There is no magic security pixie dust where you can ignore security and then plug in a security considerations section
- A quick intro into a potentially intimidating area

A plea for cross-fertilization

- The best way to get advice, say from a security expert, is to make it easy for them to come up to speed on your protocol
- Summarizing, explaining things at conceptual levels, is not a waste of time, even for those that (think they) understand the protocol

This tutorial

- An overview of basic security stuff, including demystifying cryptography
- Some interesting “off the beaten track” kinds of security things that protocol designers think about, in addition to the usual stuff

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

Where to start

- What problem are you trying to solve?
- What is the threat model (what can attackers do, what can you trust)

The Internet

- 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

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)

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)
  - Users may want to know who posted the data in the file store

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
  - Virus Scanning
  - Anti-spam

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

Examples, cont’d

- Routing protocol
  - Handshake with neighbor
    - Is the message from a valid router? (replay?)
    - How do we recognize a valid router? (autoconfiguration incompatible with security)
    - How do we know whether a message is new?
  - Routing messages
    - Even valid routers might lie (become subverted)
  - Forwarding

DOS

- In early days of networking, this threat was underestimated
- Then it started being a single attacker, limited by bandwidth of that attacker

Outdated DOS defenses

- Cookies (different from web cookies)
  - Proof that you could receive at the address you claim to be coming from
  - Helps to prevent one attacker from lying about its address
- IP traceback
  - If it ever worked, helps find the attacker that is lying about its source address

DDOS

- Chillingly clever attack
- Break into lots of machines
- Command them all to attack
- Cookies don’t help, IP traceback doesn’t help. Nobody is lying about its source address
- Cryptographic credentials also don’t help

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

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

Cryptography

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

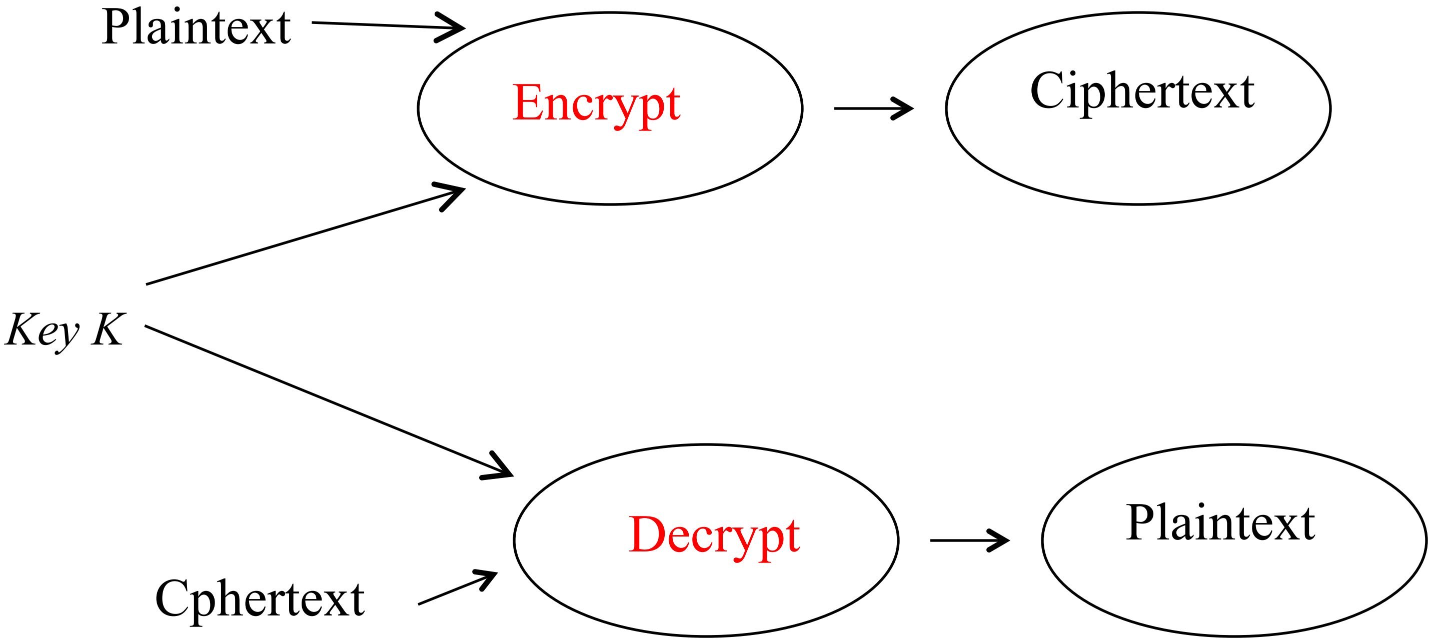
Secret Key Crypto

- 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

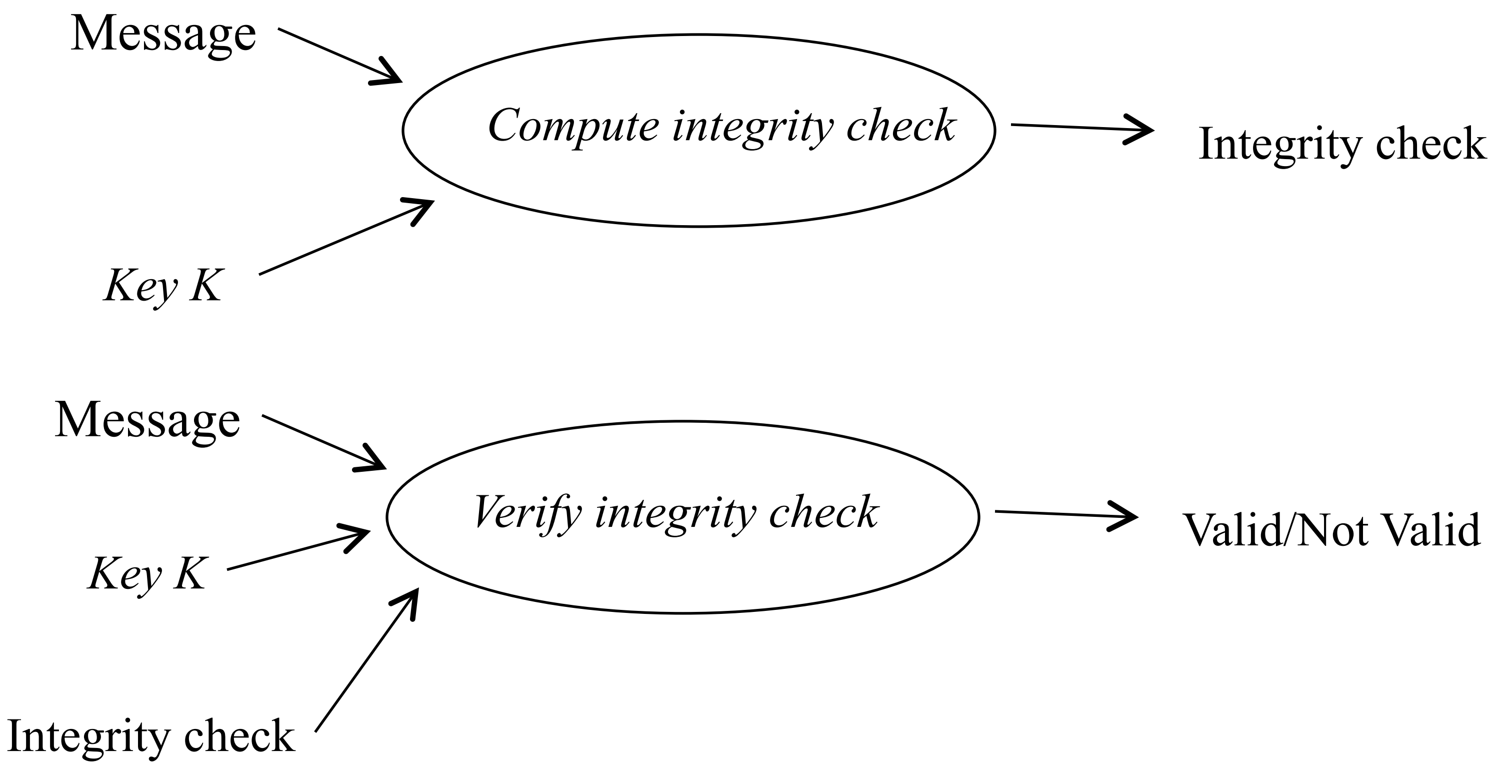
Secret key crypto, Alice and Bob share secret S

- 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)

Secret Key Encrypt



Secret Key Integrity Check



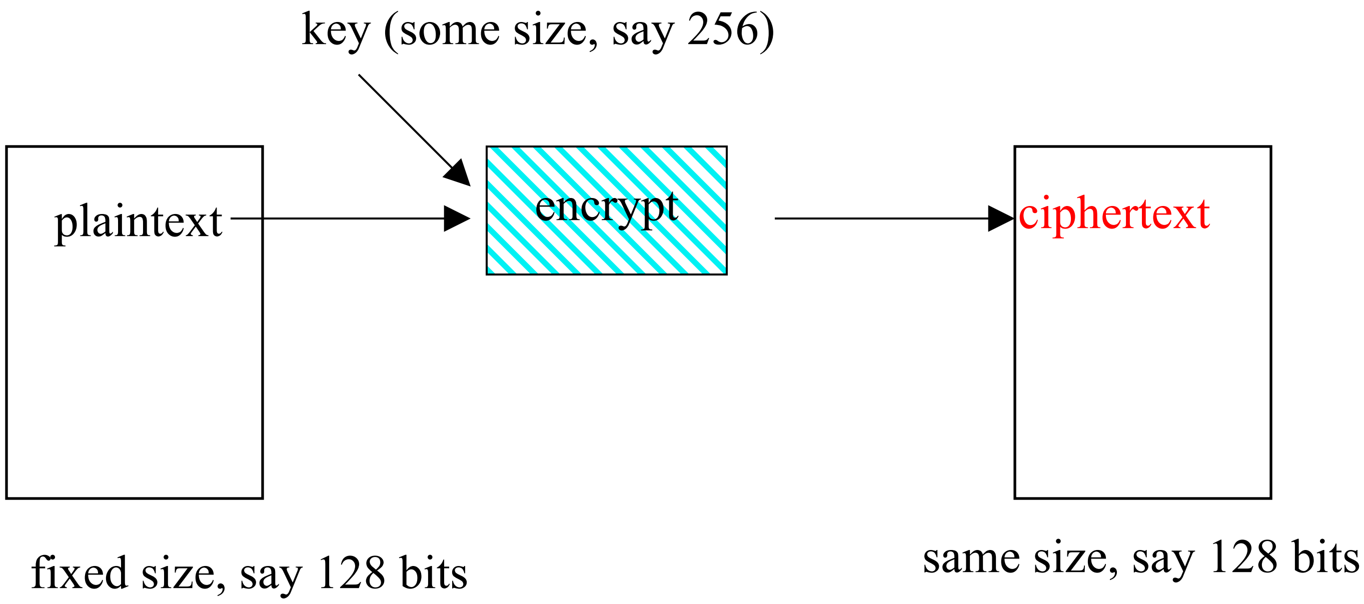
A Cute Observation

- 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!

Two types of secret key functions

- Block cipher
  - Fixed size block (e.g., 128 bits), encrypted with fixed size key (e.g., 256 bits)
  - Block size need not be the same as the key size
- Stream cipher
  - Key is used by both ends in a pseudorandom number generator
  - Result is ⊕'d with the message
  - ⊕ with same thing twice gives you the original

Block cipher

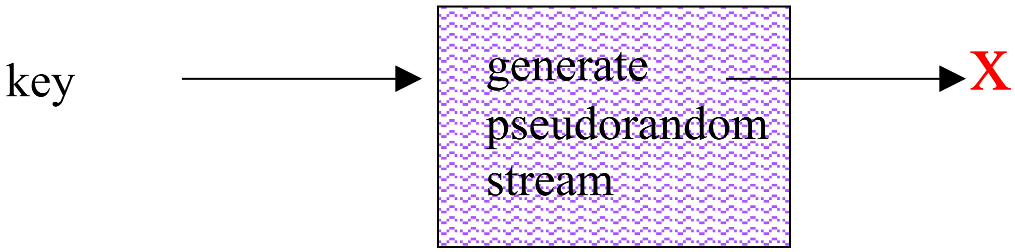


Note: block size is independent of key size

XOR (Exclusive-OR) $\oplus$

- Bitwise operation with two inputs where the output bit is 1 if exactly one of the two input bits is one
- $(B \oplus A) \oplus A) = B$
- 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



key

generate pseudorandom stream

X

msg ⊕ X = ciphertext

Encrypt

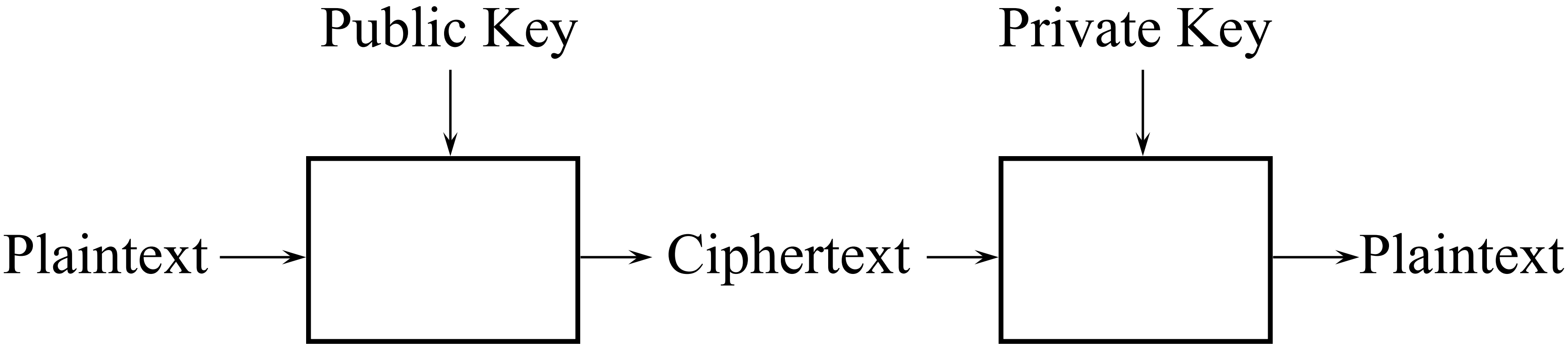
ciphertext ⊕ X = msg

Decrypt

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?

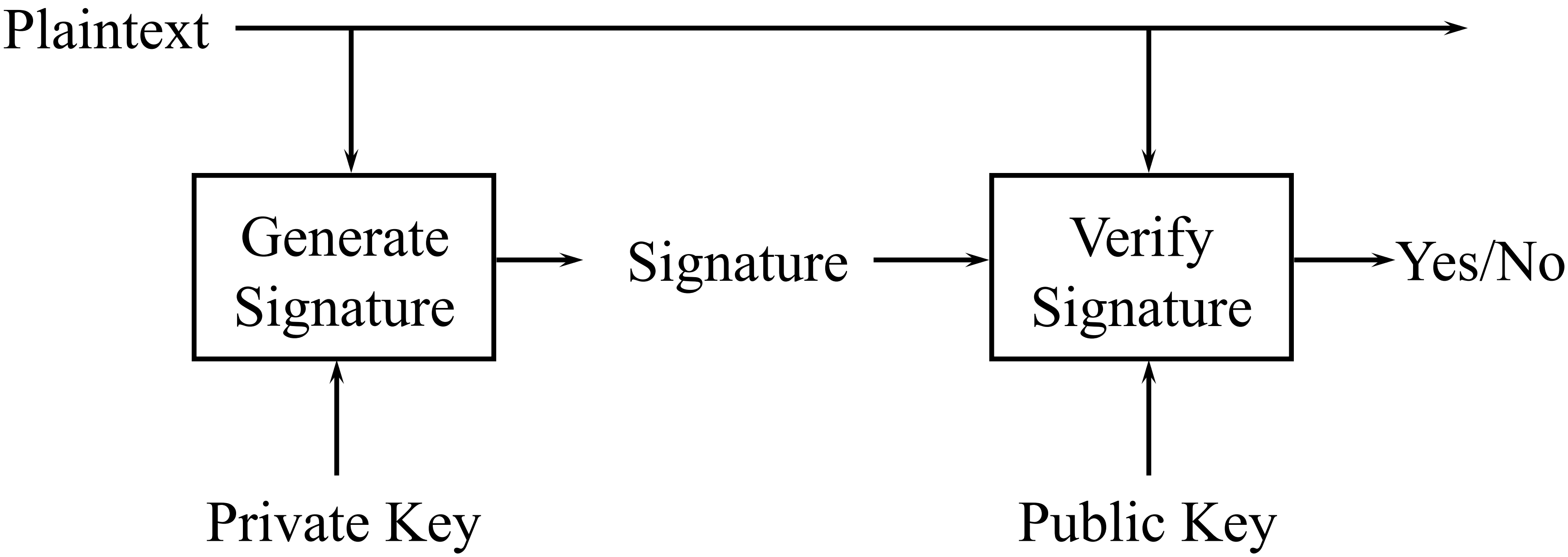
Public Key Encryption for Privacy



Digital Signatures (Public Key Integrity Check)

- 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

Public Key Integrity Protection



Repeat for emphasis

- Really important difference between secret key and public key integrity protection, especially when multiple receivers of a message
    - With secret key, verifier uses same key as sender; if you can verify a message, you can forge it
    - With public key, verifier uses public key, sender uses private key; so verifier can't forge

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!

Example Secret Key Algorithms

- DES (old standard, 56-bit key, slow, insecure)
- 3DES: fix key size but 3 times as slow
- RC4: variable length key, “stream cipher” (generate stream from key, XOR with data), really fast, stream sometimes awkward
- AES: replacement for DES

Example 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.

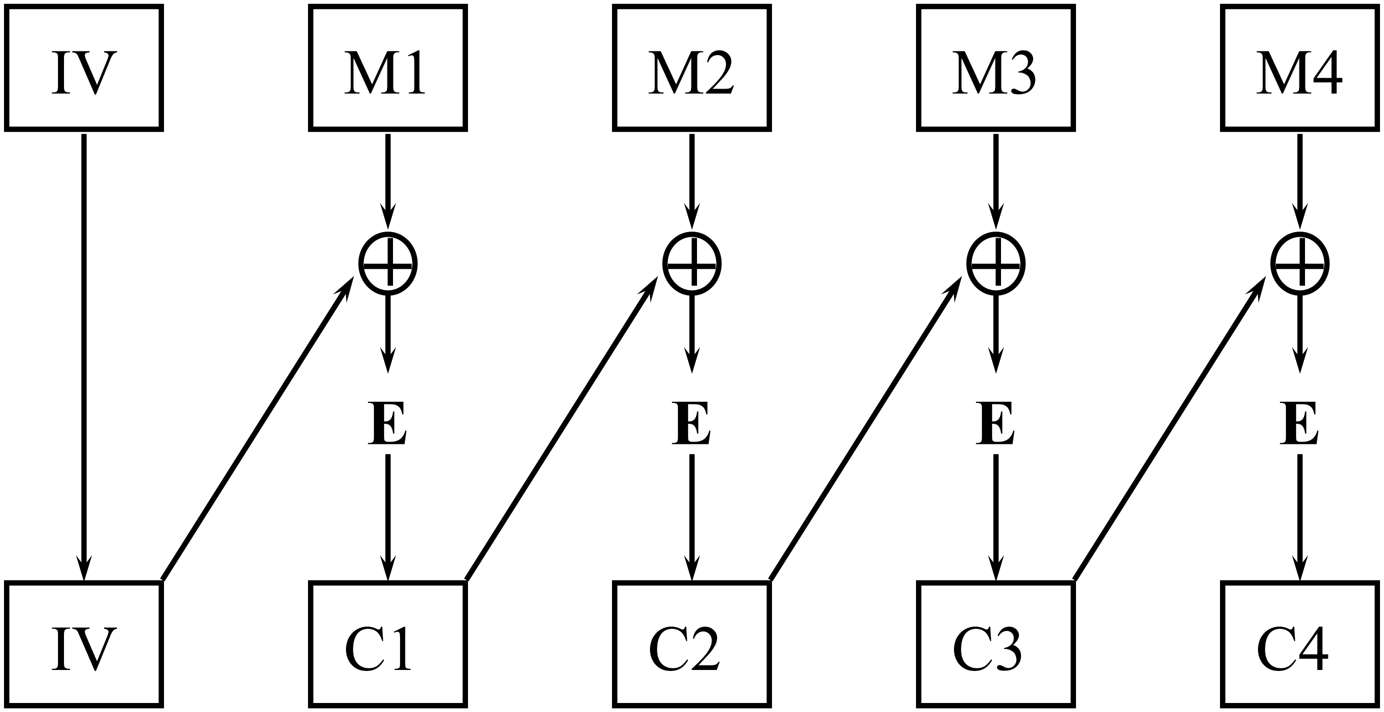
Crypto-agile

- 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

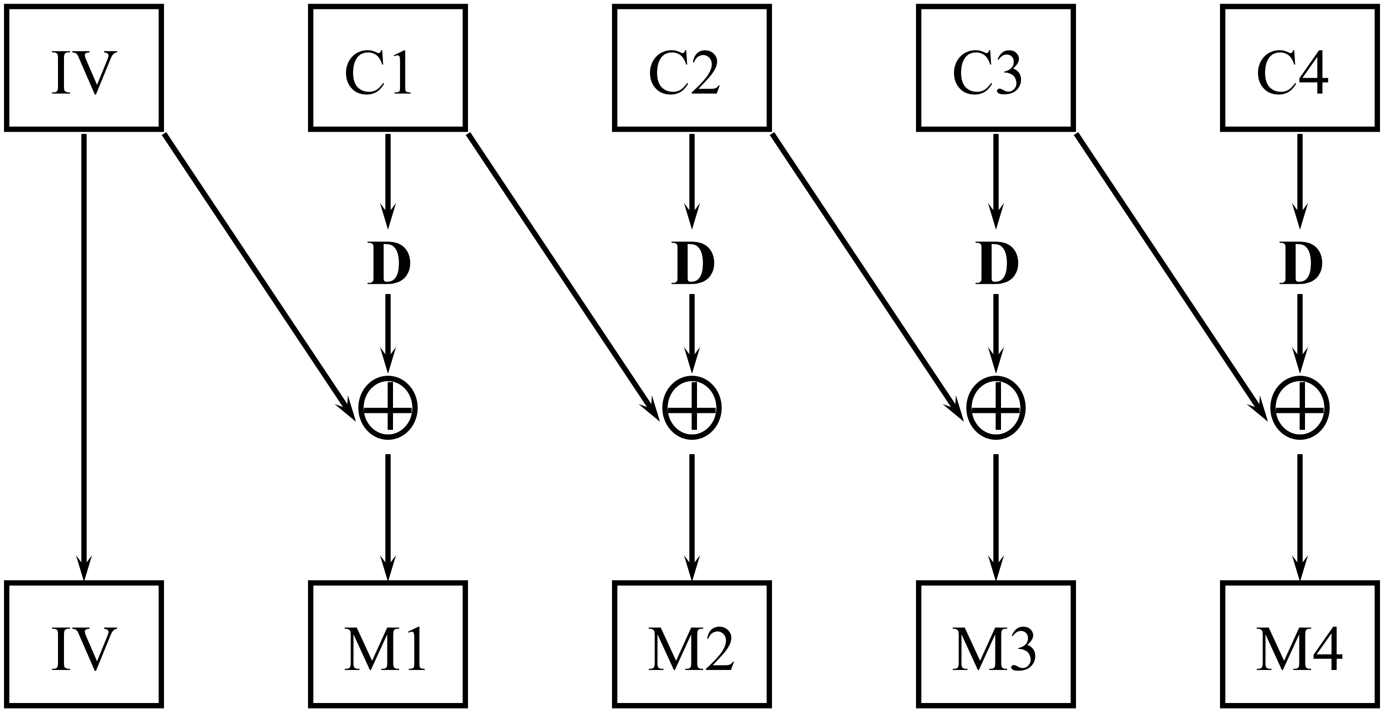
Encrypting Large Messages: “modes” and “IV”s

- 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)

CBC (Cipher Block Chaining)

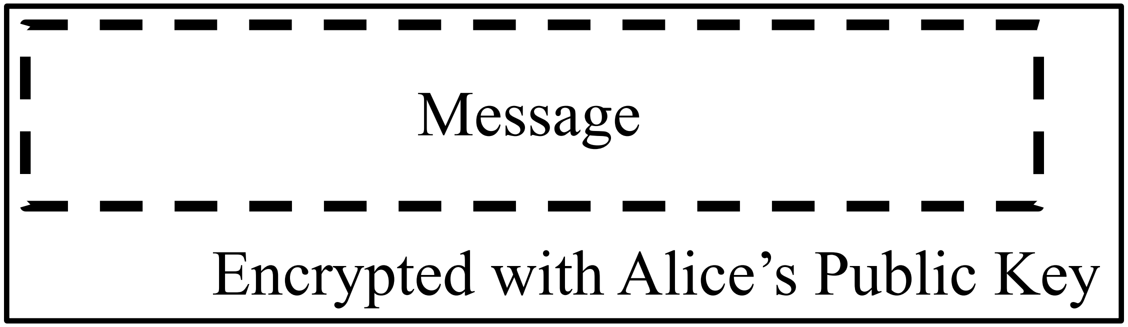


CBC Decryption



Encrypting with public key

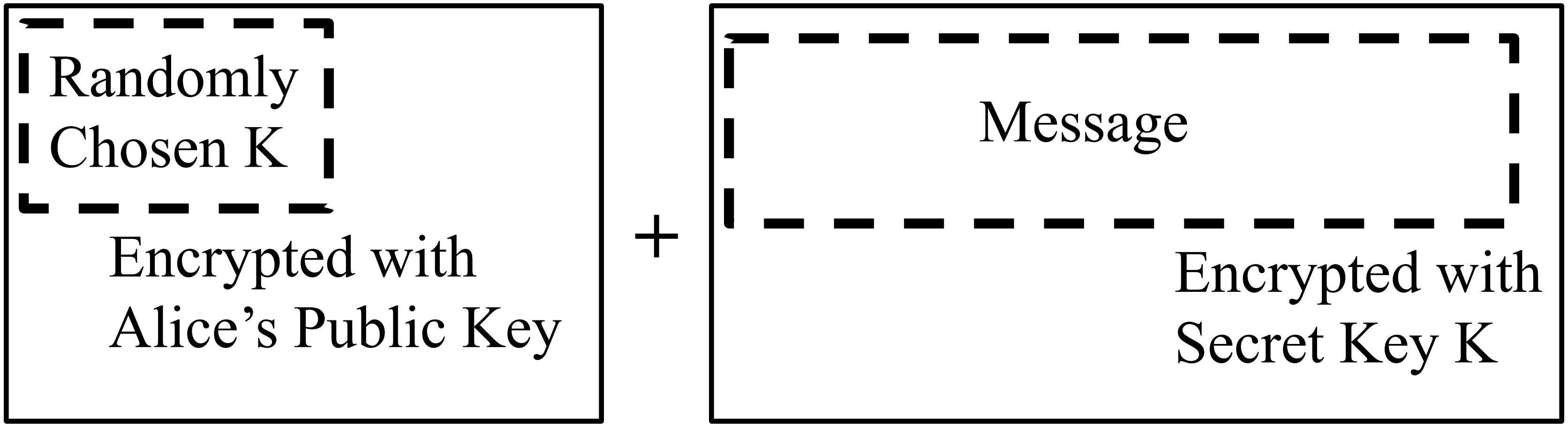
Instead of:



Message

Encrypted with Alice’s Public Key

Use:



Randomly Chosen K

Encrypted with Alice’s Public Key

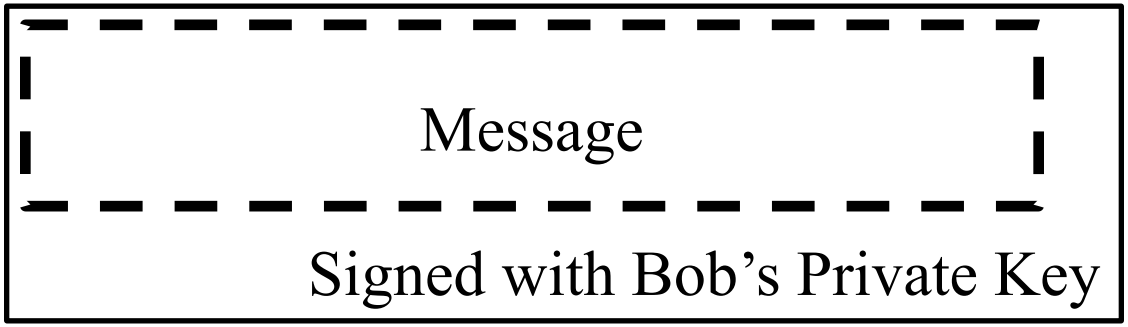
+

Message

Encrypted with Secret Key K

Digital Signatures

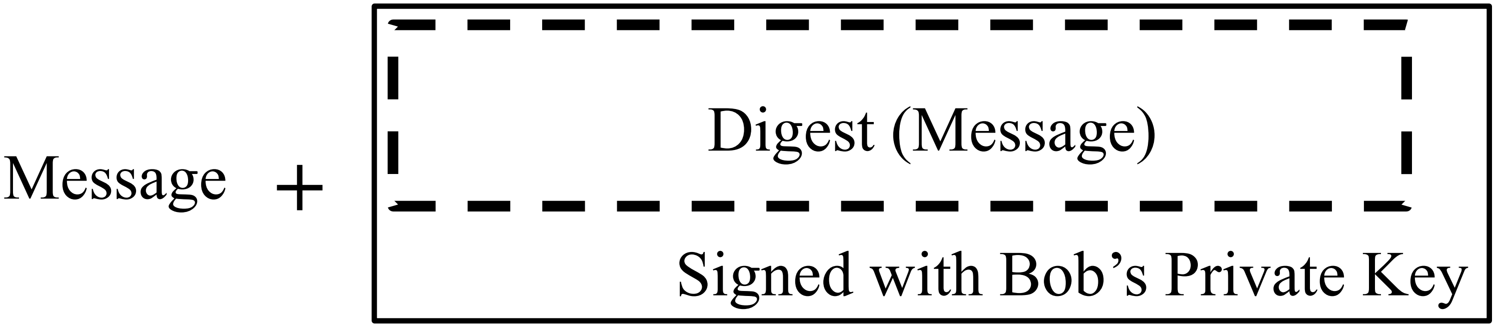
Instead of:



Message

Signed with Bob’s Private Key

Use:

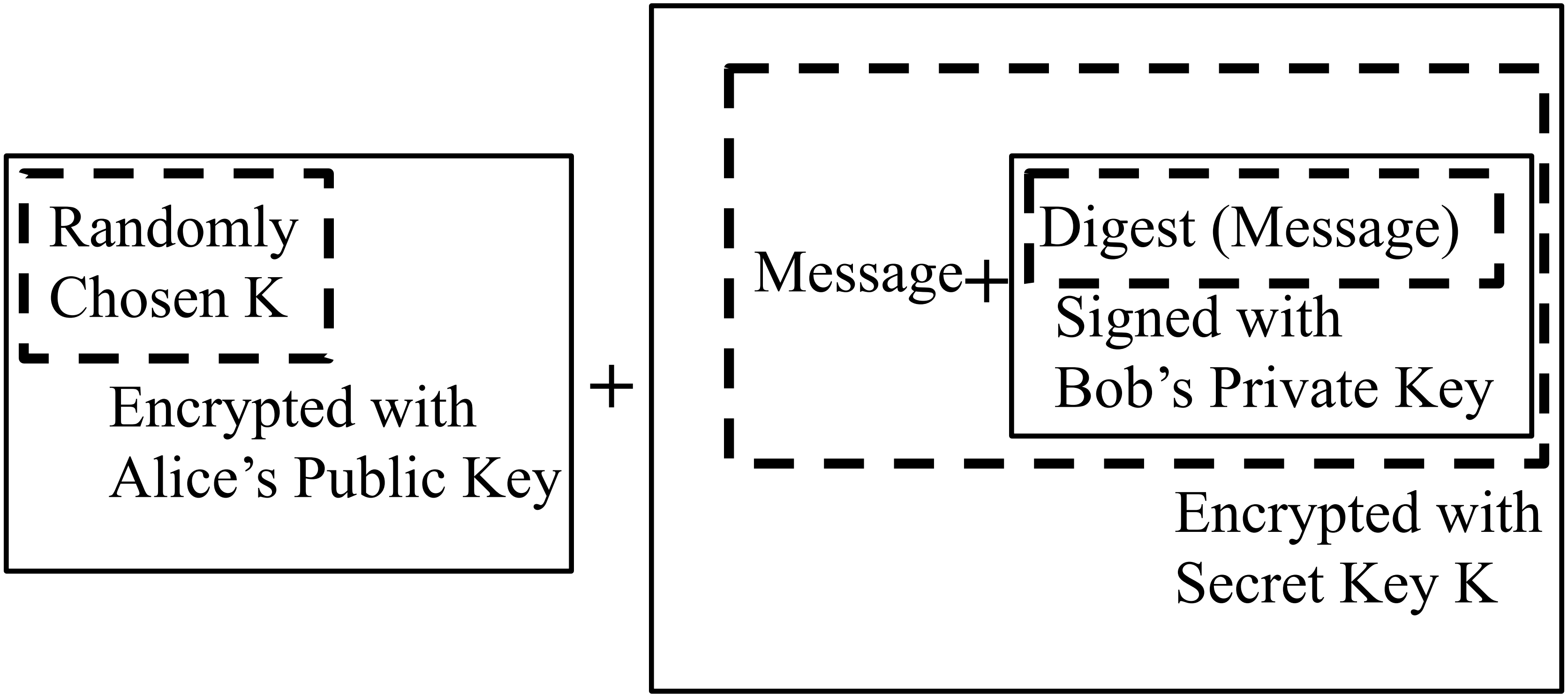


Message +

Digest (Message)

Signed with Bob’s Private Key

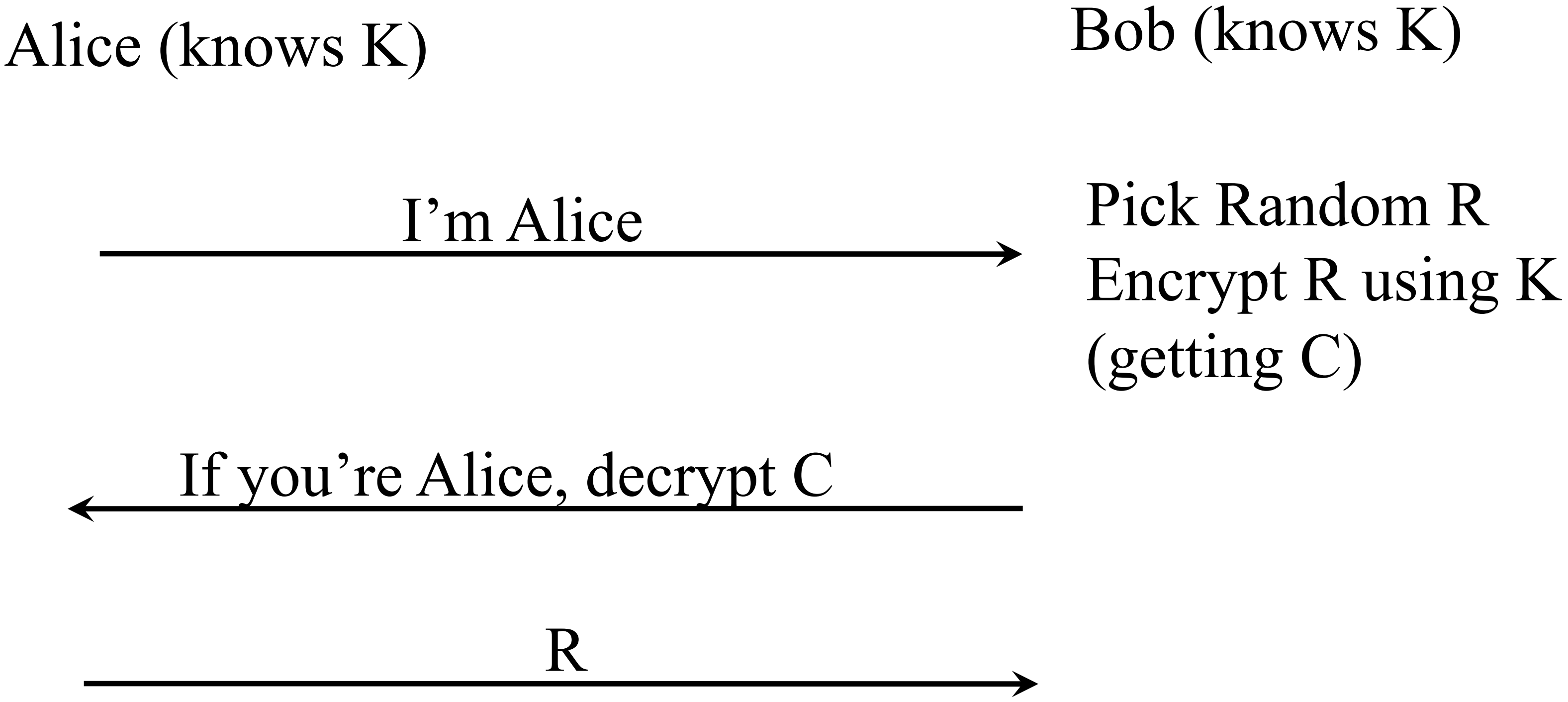
Signed and Encrypted Message



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)

Passwords are hard to get right!

- People “can’t” remember passwords with enough cryptographic strength to provide meaningful security as keys
- People reuse passwords, so it is dangerous to have servers storing passwords for their users
- Turn user authentication into real keys as close to the user as possible

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

Passwords ‘in the clear’ considered harmful

- Assume eavesdropping on the Internet is universal.
- Surest way to get your protocol bounced by IESG.

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

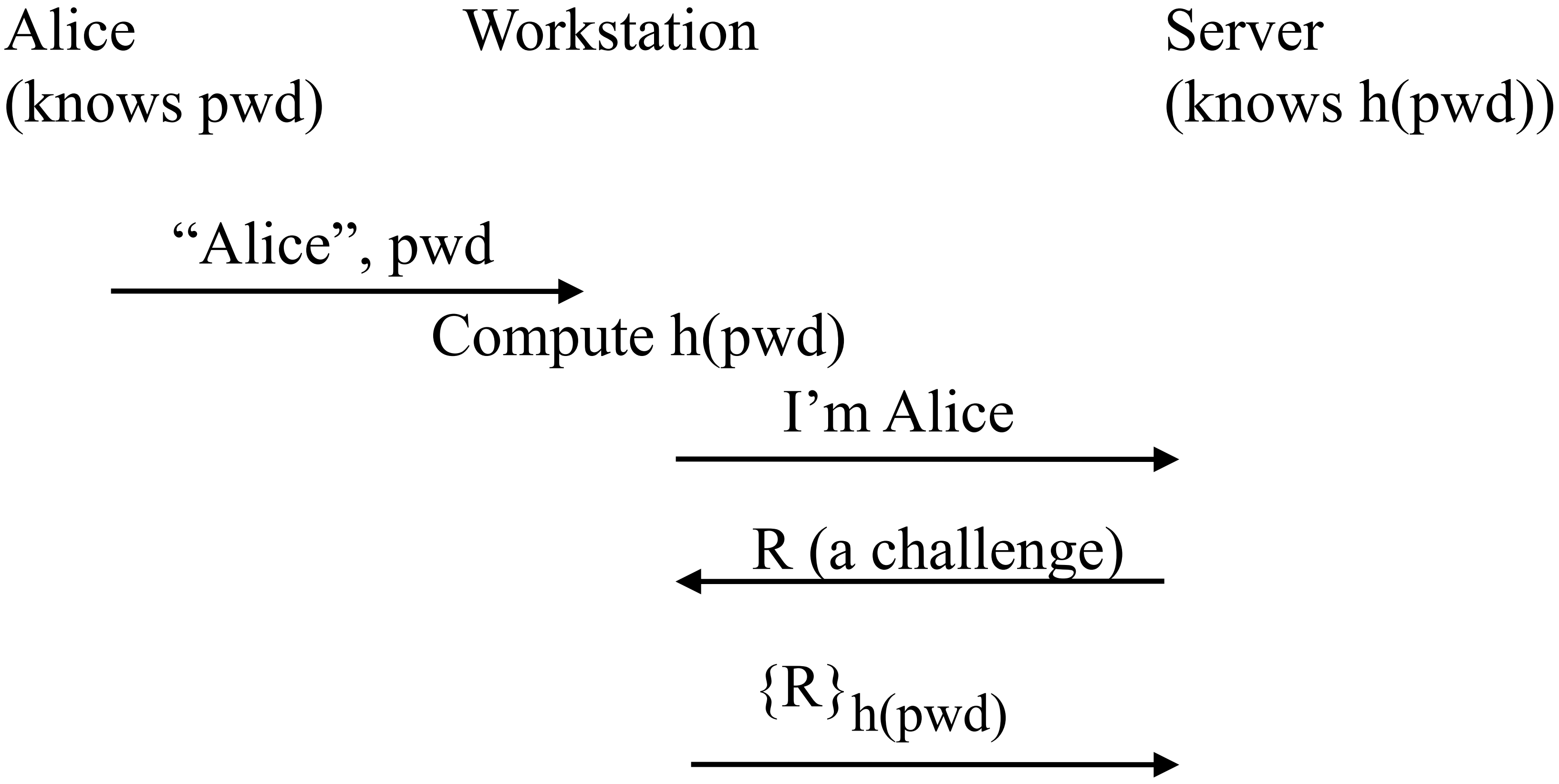
Off-Line Password Guessing

- 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(pwd))

“Alice”, pwd

Compute h(pwd)

I’m Alice

R (a challenge)

$\{R\}_{h(pwd)}$

Other ways of authenticating people

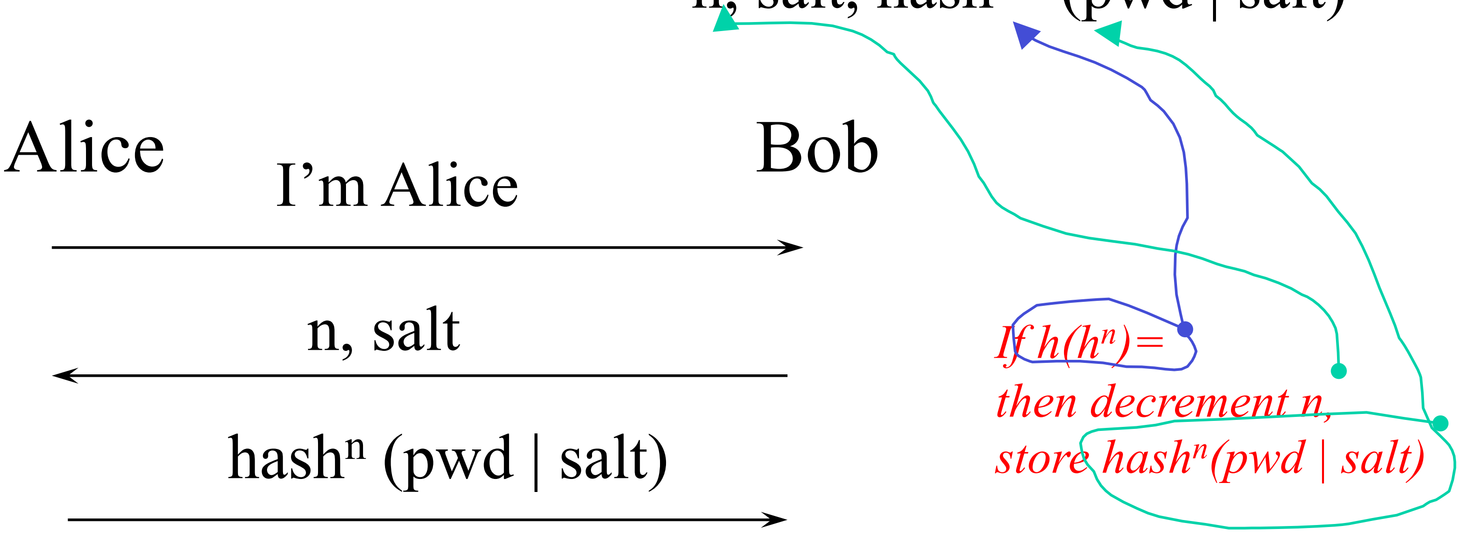
- OTP
- Tokens (e.g., challenge/response, time-based)
- SASL and EAP are frameworks for negotiating what kind of authentication to do

Token Cards

- should be 2-factor (card+PIN)
- challenge/response (need keyboard)
- time-based
  - time skew (can adjust time and rate each time)
  - if no keyboard, PIN can be sent with value
- sequence based
  - issue if mess up several times
  - same PIN issues if no keyboard

Lamport’s Hash (S/Key, OTP)

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



Alice

Bob

I’m Alice

n, salt

hash^{n} (pwd | salt)

If $h(h^{n})=$
then decrement n,
store hash^{n}(pwd | salt)

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

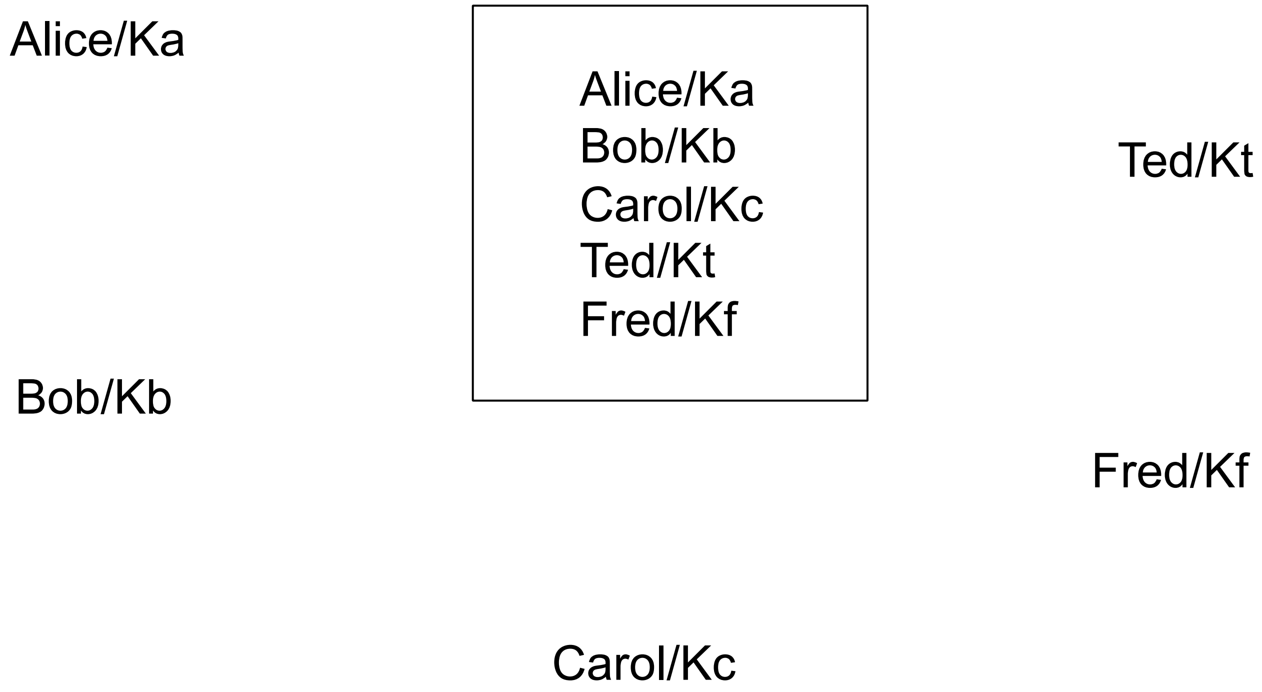
Trusted Third Parties

How do two parties get introduced?

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

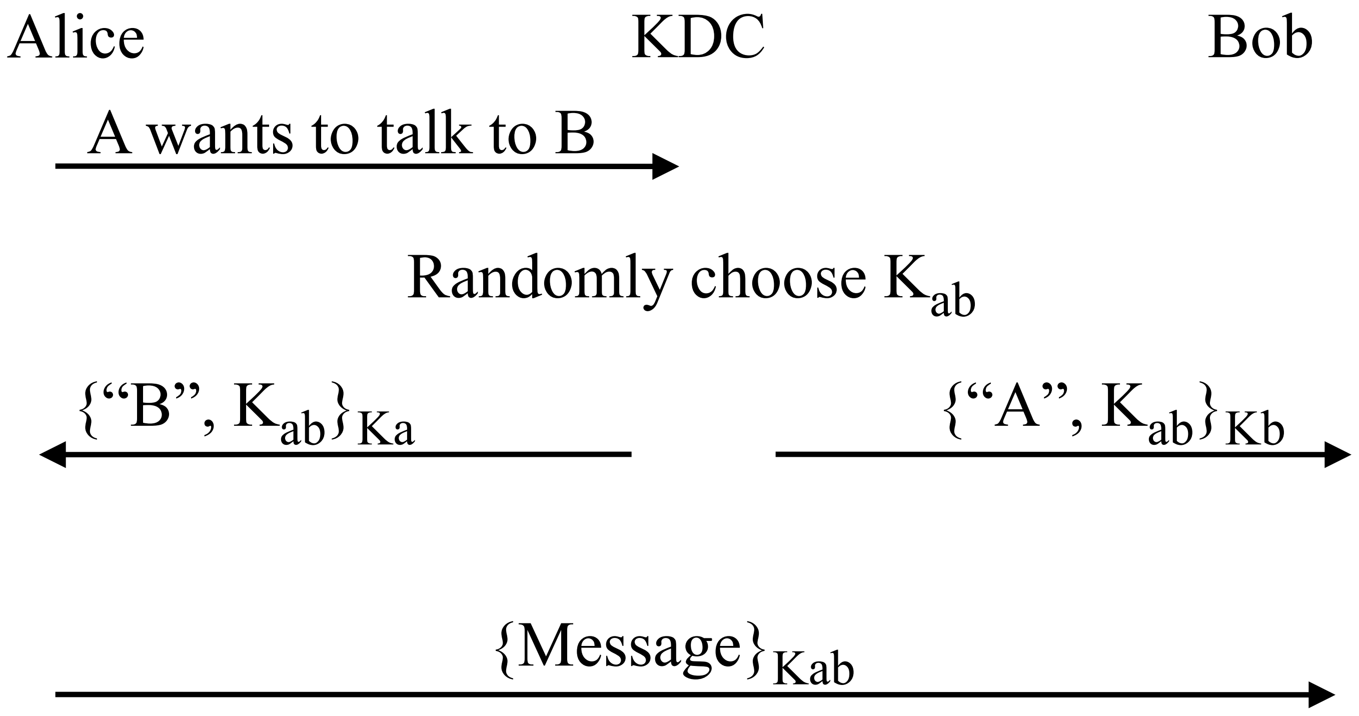
Ted/Kt

Bob/Kb

Fred/Kf

Carol/Kc

Key Distribution - Secret Keys



Alice

KDC

Bob

A wants to talk to B

Randomly choose K_{ab}

$\{\text{“B”}, K_{ab}\}_{Ka}$

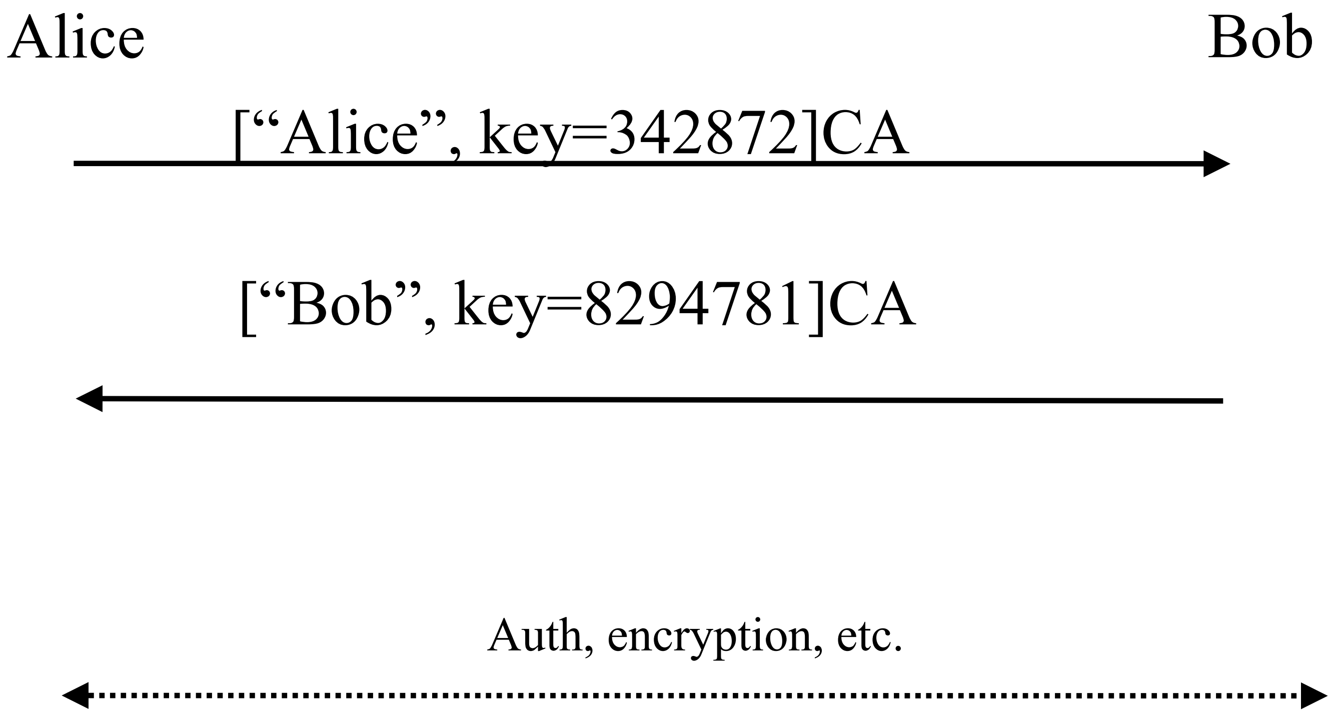
$\{\text{“A”}, K_{ab}\}_{Kb}$

$\{\text{Message}\}_{Kab}$

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



Alice

Bob

[“Alice”, key=342872]CA

[“Bob”, key=8294781]CA

Auth, encryption, etc.

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)

Another philosophy: “Identity Providers”

- For web-based protocols
- Usually based on SAML standard, which uses XML syntax
- User authenticates to an identity provider
- To authenticate to an affiliated web site, use the magic of URL rewriting, http redirection, cookies, etc., to obtain a cookie signed by the IDP saying “this is Radia”

Functional difference from Kerberos

- Although SAML could encode same information as a (PKI) certificate, the usual use is as a “bearer token”
- With Kerberos, the “ticket” doesn’t prove who you are, you have to prove knowledge of the key inside the ticket
- Likewise, with PKI, you still have to prove knowledge of the private key associated with the public key in the certificate

Other interesting issues (with all the schemes)

- You won’t have one KDC/CA/IDP for the whole world
- So how do you find a proper chain?

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

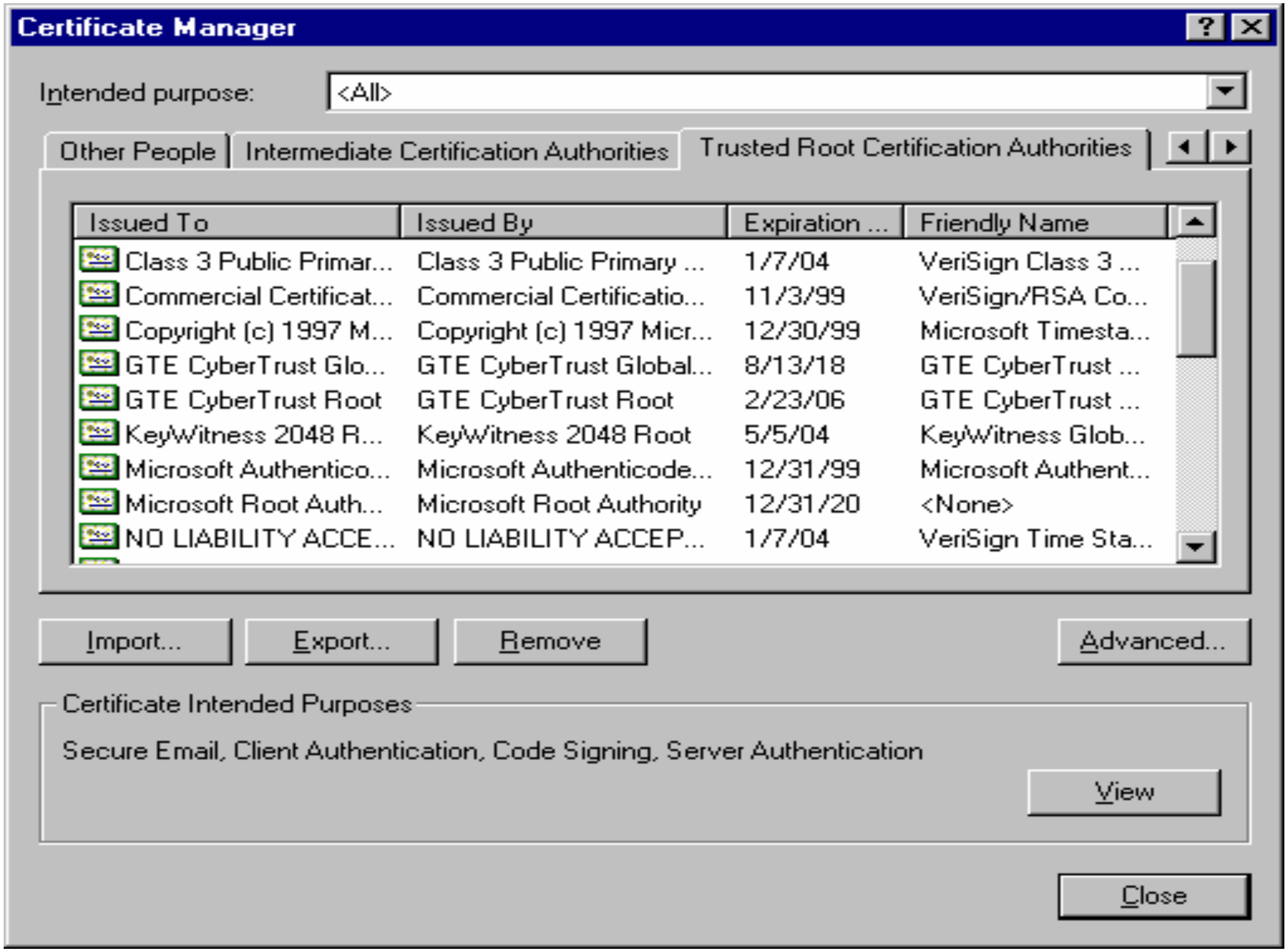
What’s wrong with one CA plus RAs?

- 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



Certificate Manager
Other People
Intermediate Certification Authorities
Trusted Root Certification Authorities
Import...
Export...
Remove
Advanced...
View
Close

Intended purpose: <All>

Issued To	Issued By	Expiration ...	Friendly Name
Class 3 Public Primar...	Class 3 Public Primary ...	1/7/04	VeriSign Class 3 ...
Commercial Certificat...	Commercial Certificatio...	11/3/99	VeriSign/RSA Co...
Copyright (c) 1997 M...	Copyright (c) 1997 Micr...	12/30/99	Microsoft Timesta...
GTE CyberTrust Glo...	GTE CyberTrust Global...	8/13/18	GTE CyberTrust ...
GTE CyberTrust Root	GTE CyberTrust Root	2/23/06	GTE CyberTrust ...
KeyWitness 2048 R...	KeyWitness 2048 Root	5/5/04	KeyWitness Glob...
Microsoft Authentico...	Microsoft Authenticode...	12/31/99	Microsoft Authent...
Microsoft Root Auth...	Microsoft Root Authority	12/31/20	<None>
NO LIABILITY ACCE...	NO LIABILITY ACCEP...	1/7/04	VeriSign Time Sta...

Certificate Intended Purposes

Secure Email, Client Authentication, Code Signing, Server Authentication

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 (computationally too difficult to find path)
  - no practical way to tell if path should be trusted
  - too many decisions for user

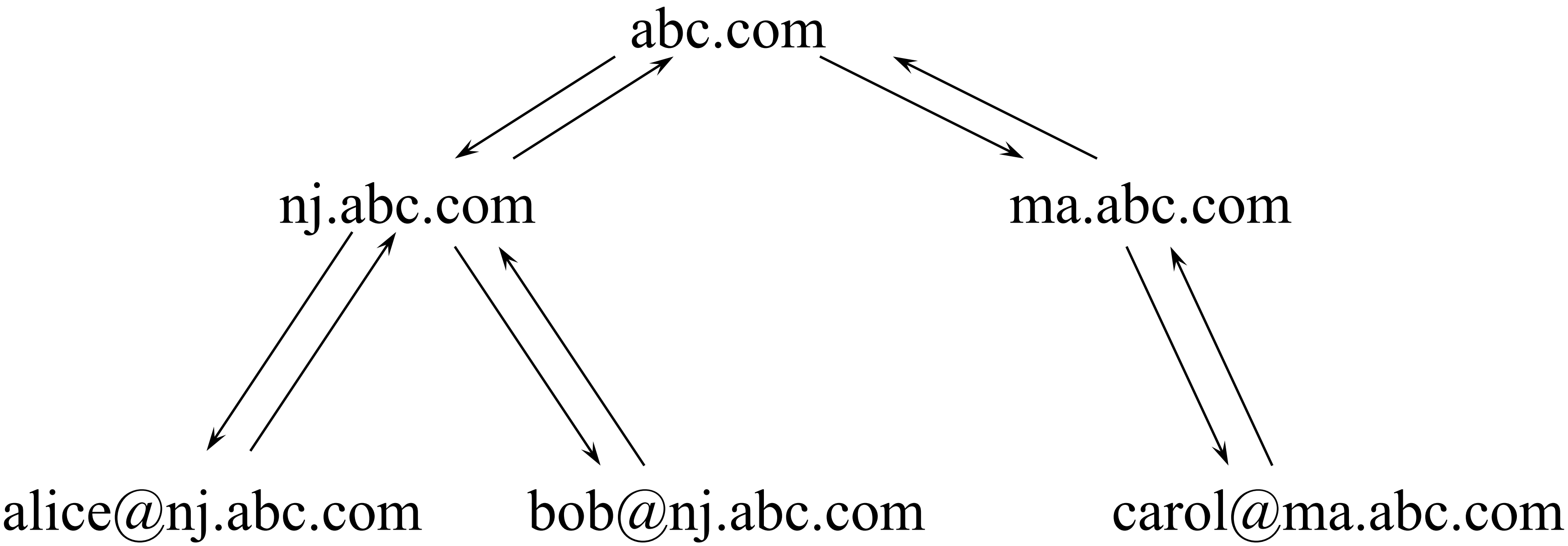
Top Down with Name Subordination

- 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


abc.com

nj.abc.com

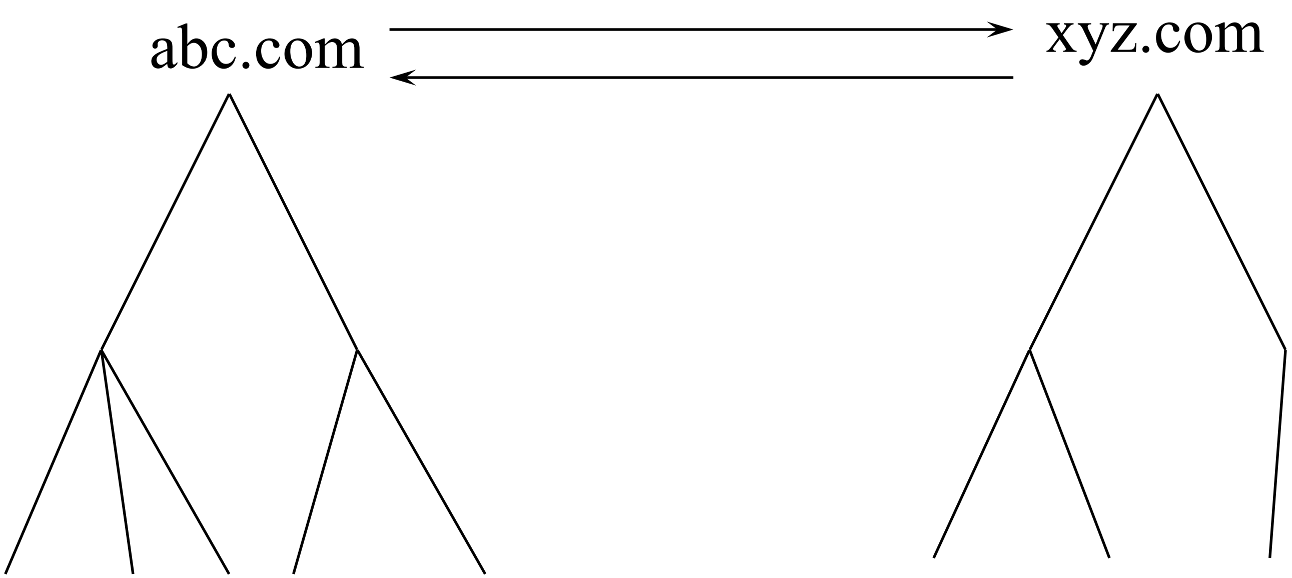
ma.abc.com

alice@nj.abc.com

bob@nj.abc.com

carol@ma.abc.com

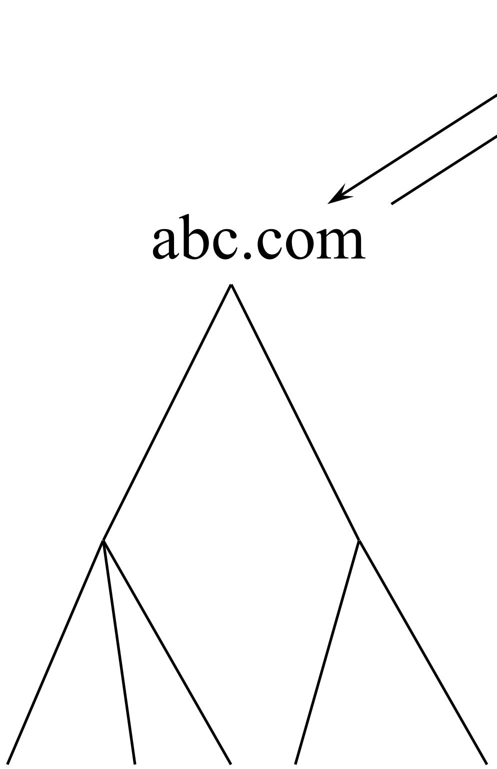
Extranets: Crosslinks



abc.com

xyz.com

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

Comparisons

- IPsec and DTLS protects packets
  - Could be end to end or between firewalls
  - Today, most uses are transparent to applications
- TLS & SSH protect TCP sessions
- OpenPGP, S/MIME and CMS, XML-DSIG and XML-encryption, protect messages (needed for store and forward)

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 advantages
  - 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)
- IPsec disadvantage
- DTLS is TLS over UDP, so it’s similar to IPsec

Creating a session key

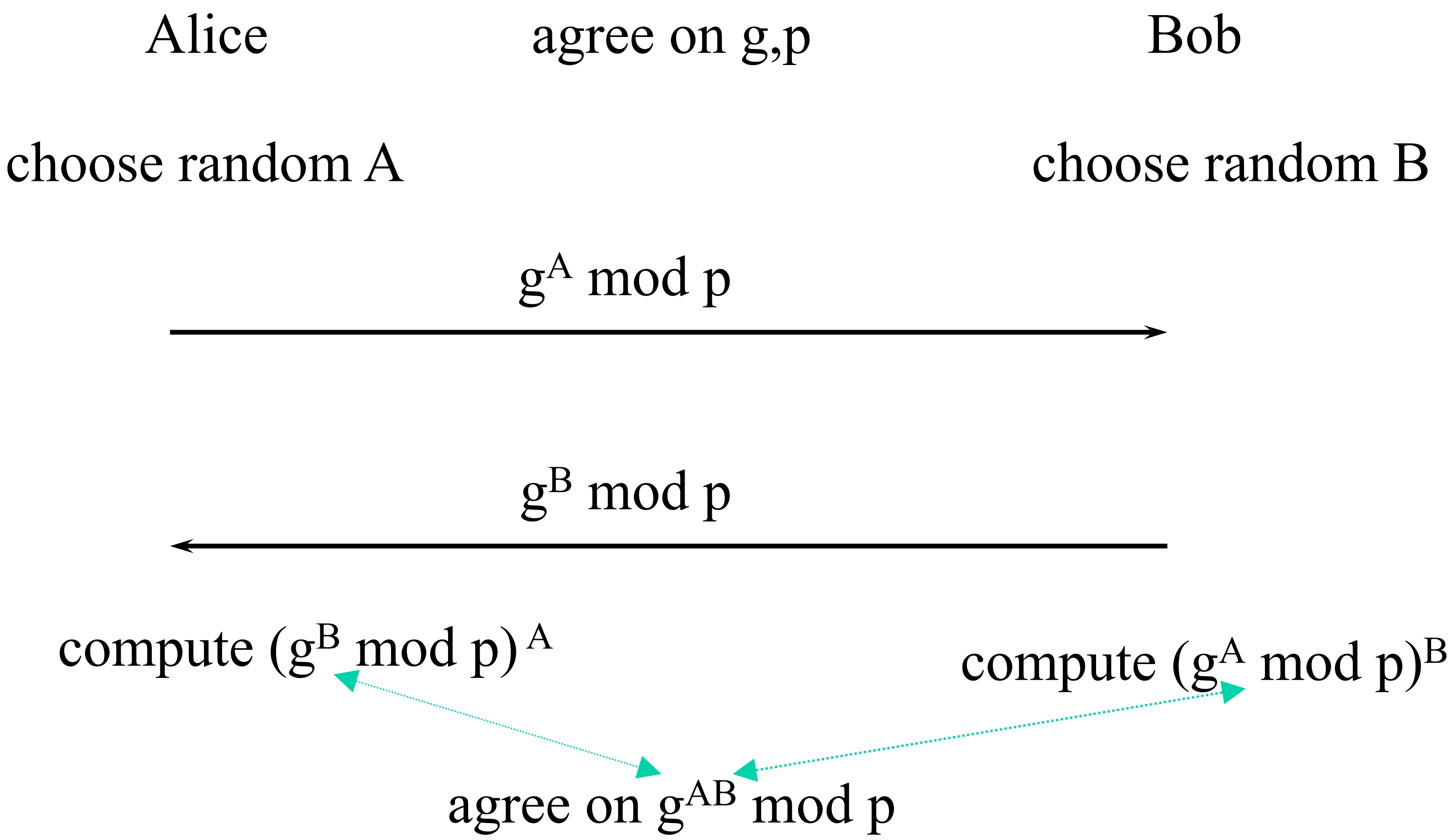
An Intuition for Diffie-Hellman

- 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?
- With the best known mathematical techniques, this is somewhat harder than factoring a composite of the same magnitude as p
- Subtlety: they haven’t proven that the algorithms are as hard to break as the underlying problem

Diffie-Hellman



Alice

agree on g,p

Bob

choose random A

choose random B

$g^A \bmod p$

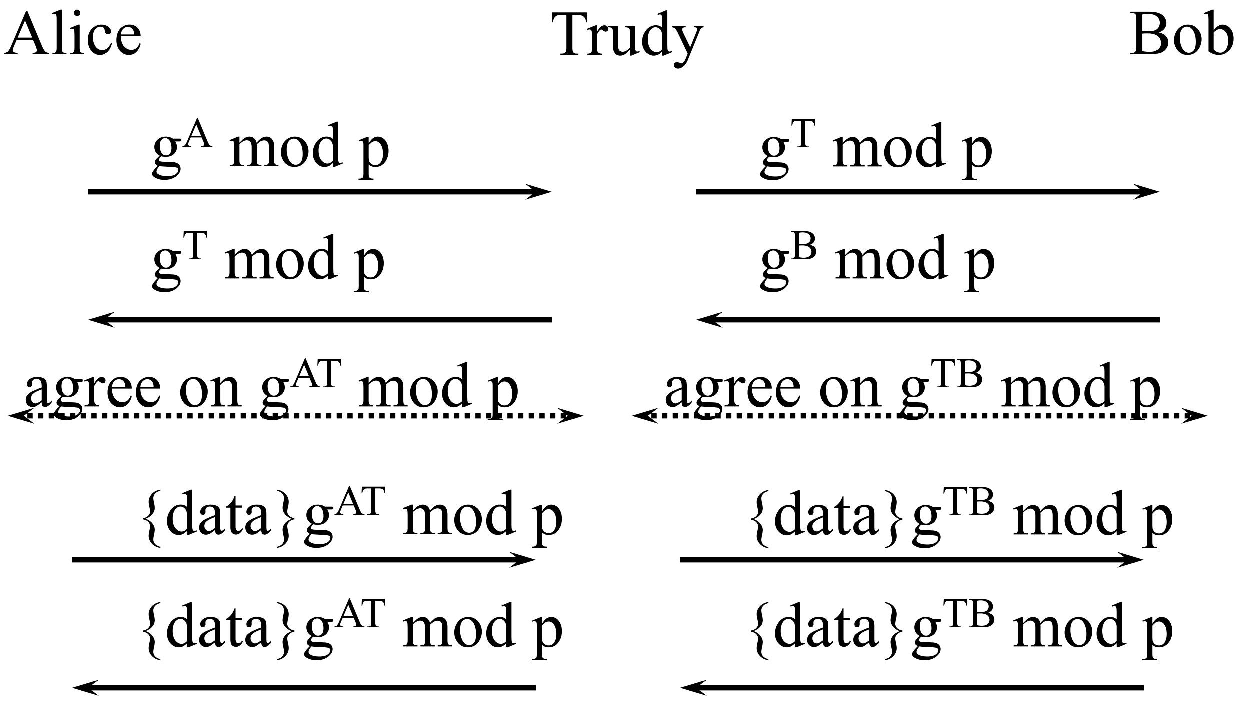
$g^B \bmod p$

compute $(g^B \bmod p)^A$

compute $(g^A \bmod p)^B$

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

Man in the Middle



Alice

Trudy

Bob

g^A mod p

g^T mod p

g^T mod p

g^B mod p

agree on g^{AT} mod p

agree on g^{TB} mod p

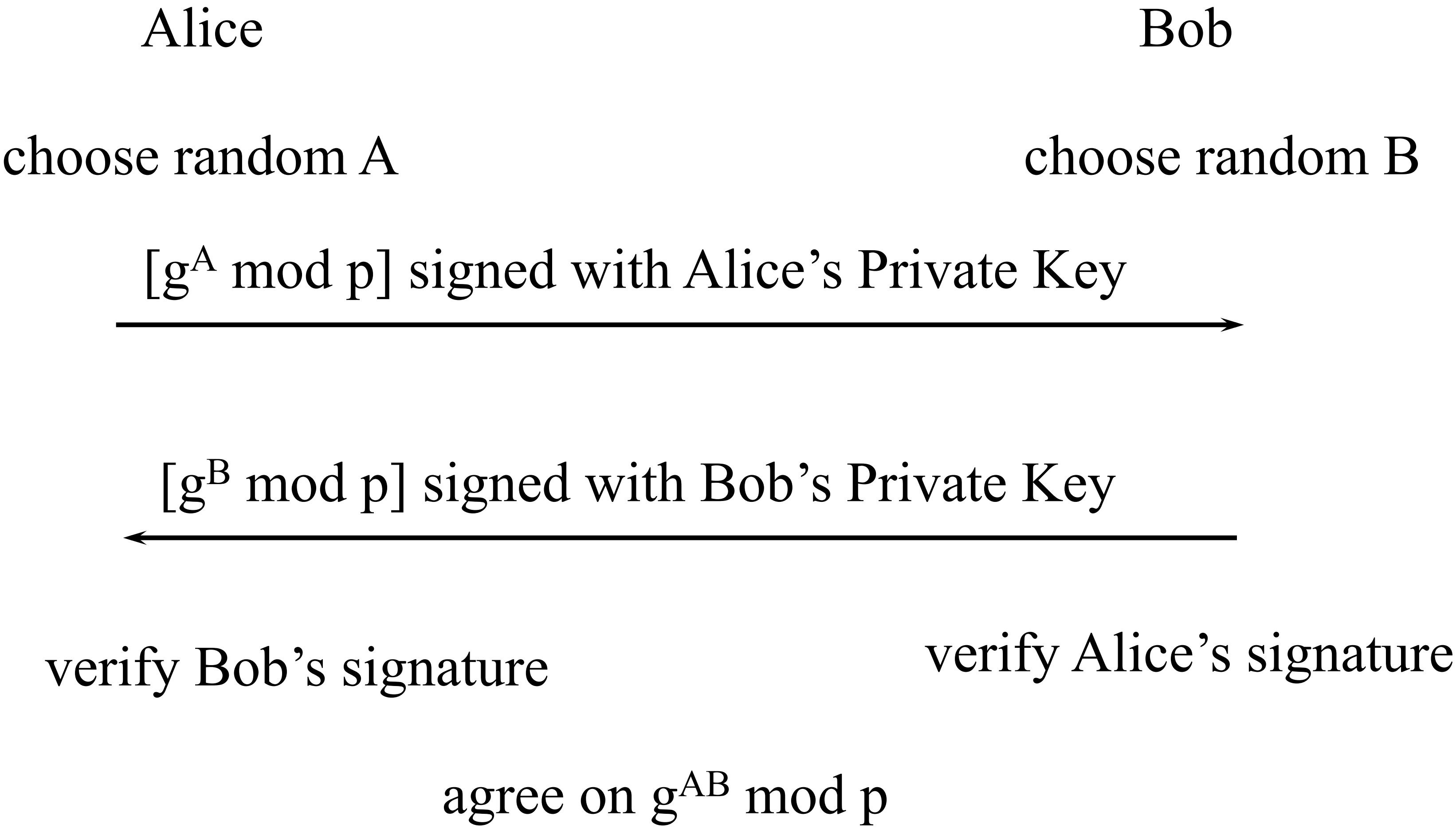
{data}g^{AT} mod p

{data}g^{TB} mod p

{data}g^{AT} mod p

{data}g^{TB} mod p

Signed Diffie-Hellman (Avoiding Man in the Middle)



Alice

Bob

choose random A

choose random B

[g^A mod p] signed with Alice’s Private Key

[g^B mod p] signed with Bob’s Private Key

verify Bob’s signature

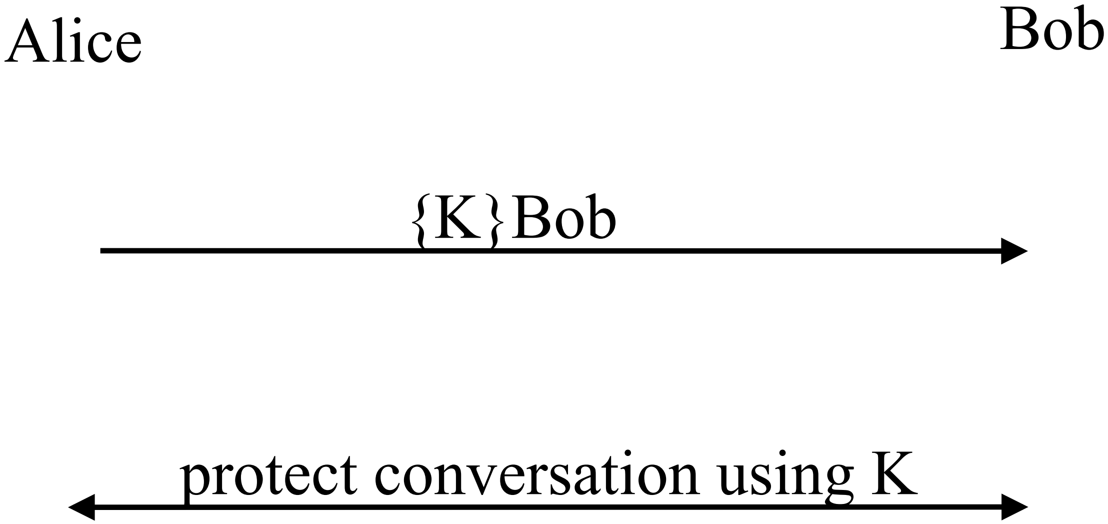
verify Alice’s signature

agree on g^{AB} mod 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)

Example non-PFS (like SSL)



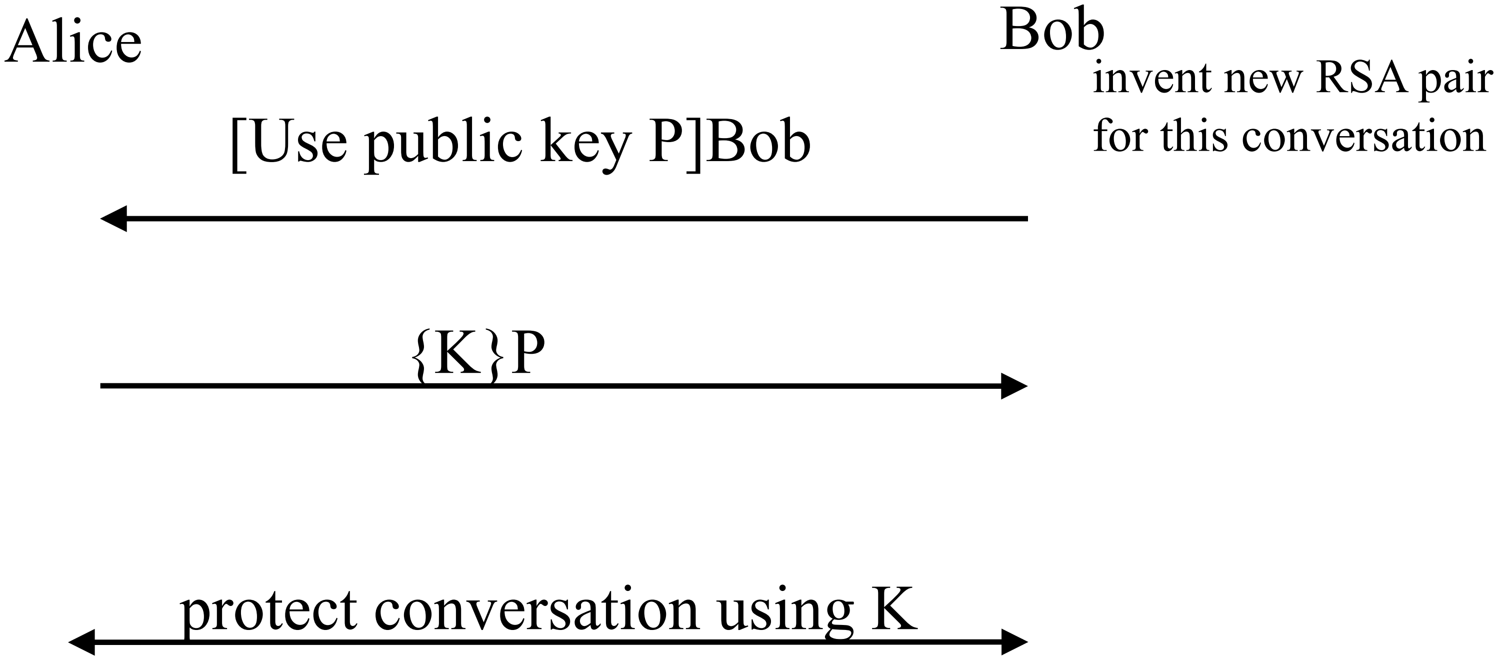
Alice

Bob

{K}Bob

protect conversation using K

PFS without Diffie-Hellman



Alice

Bob

invent new RSA pair
for this conversation

[Use public key P]Bob

{K}P

protect conversation using K

Replay Issues

- Usually use a sequence number
- Without unique session key per session, if start with same initial sequence number, can do replays
- What if sequence number too small?

Extended sequence number

- For instance, TCP’s sequence number is too small
- Instead of changing the protocol to have a larger sequence number, do the integrity check on a larger sequence number, “pretending” it’s in the packet

DNSSEC

- Provides signature for information in DNS
  - So don’t need to have super trusted DNS servers
- Also provides public key of of (child) zone
  - So it’s the top-down model
- If one of the records in a DNS entry were “public key of server”, then DNSSEC could be lightweight PKI

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**

Every RFC needs a “security considerations” section

- What do you have to think about?
- Not enough to say “just use IPsec”
- Sometimes (as with VRRP) protecting one protocol in a vacuum is wasted effort
  - putting expensive locks on one window, while the front door is wide open
- We don’t need to protect a protocol. We need to protect the user

Things to put in a security considerations section

- What are the threats? Which are in scope? Which aren’t? (and why)
- What threats are defended against? Which are the protocol susceptible
- Implementation or deployment issues that might impact security
- See RFC 3552 “Guidelines for Writing RFC Text on Security Considerations”

Examples

- Putting integrity checks on routing msgs
  - Defends against outsiders injecting routing msgs. That’s good, but
  - Doesn’t prevent outsiders from answering ARPs, or corrupting DNS info
  - Doesn’t protect against “Byzantine failures” (where a trusted thing goes bad)

Examples

- SNMP
- Should be straightforward end-to-end security
- But it has to work when the network is flaky
  - DNS not available
  - LDAP database for retrieving certificates might be down, as might revocation infrastructure

Examples

- Non-crypto things
  - Use up resources
    - DHCP, could request all possible addresses
    - Use all bandwidth on a link
  - Interface to human – internationalized names
  - Active Content
    - Too many examples of hidden places for active content!

Examples

- Email (much more detail in RFC 2552), but some cute points
  - Trivial to spoof mail
  - Message path leaks information
  - There’s a protocol for asking if an email address is valid---useful for/against spammers
  - Even with S/MIME, header fields not protected

Example

- Kerberos Network Auth Service
- Some excerpts
  - solves authentication
  - does not address authorization or DOS or PFS
  - requires on-line database of keys, so NAS must be physically secured
  - subject to dictionary attack (pick good pwds)
  - requires reasonably synchronized clocks
  - tickets might contain private information
  - NAS must remember used authenticators to avoid replay

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