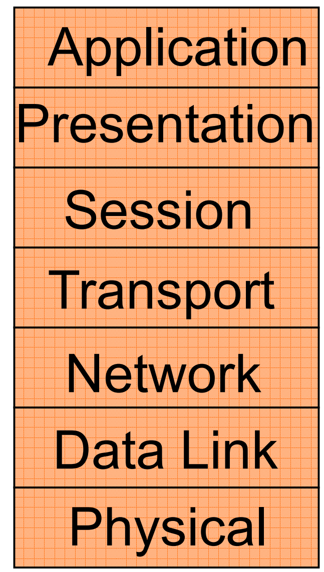
Network Security I

- Focus
  - How do we secure network systems?

- Topics
  - Privacy, integrity, authenticity, timeliness
  - Cryptography



Preliminaries: End-Host Security

- Traditional security concepts:
  - Integrity
    - My files shouldn't be modifiable by an unauthorized user
  - Privacy
    - My files shouldn't be readable by an unauthorized user

- Traditional security mechanisms:
  - Authentication
    - Who are you?
  - Authorization
    - What are you allowed to do?

Preliminaries (cont.)

- "Trusted computing base"
  - Components of the system that you believe are respecting the security policy but that are not verified as doing so
    - The user trusts the operating system
      - E.g., won't leak your files to unauthorized users, won't spuriously delete/modify them
- User trusts applications
  - Emacs isn't mailing your file to its authors
- User trusts the hardware
  - Is your keyboard trustworthy?
  - Is an ATM trustworthy?
- Does the OS trust users?
  - Mandatory access control

Preliminaries: Network Security

- Most of the technologies in lower protocol layers were developed pre-Internet
- Pre-Internet:
  - There weren't many network services (telnet, mail, ftp, a few others)
  - There weren't many machines on networks
    - Many local networks, but not very interconnected
  - "End-to-end security" made sense
    - Trusted OSes running trusted applications run by trusted users
      - At the very least, you could probably track down a malicious user
- Result: no security mechanisms were built into protocols themselves
  - E.g., mail spoofing was trivial

Preliminaries: Post-Internet

- Really an entirely new situation
  - Servers want “anonymous” users
  - Users want to talk with unverified servers
  - Users want to run unverified code
- Possible approaches:
  - Verification of identity + trust
    - X.509 certificates
  - Enforcement
    - Java security model

Network Security

- What properties would we like the network to offer?
  - Privacy: messages can’t be eavesdropped
  - Integrity: messages can’t be tampered with
  - Authenticity: we can verify who created the message
  - Timeliness: we can verify that the packet was sent not too long ago
  - Availability: I can send and receive the packets I want
  - Non-repudiation: you can’t claim you didn’t say something you did

  - Anonymity: not only can’t you tell what the content of my conversation is, you can’t even tell who I’m talking with

- There are other properties we would like from the distributed services that run on top, as well
  - E.g., if I send you my medical records, you can’t send them to anyone else

Achieving Security

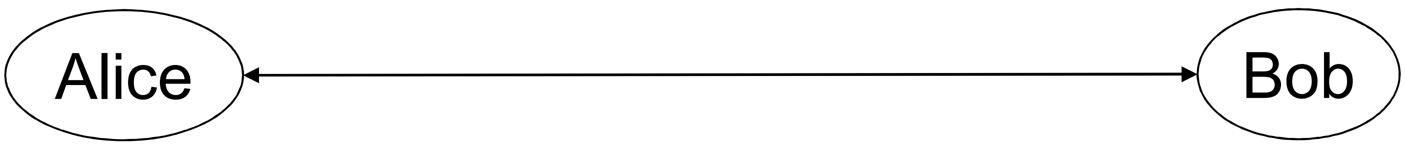
- It’s not about making security violations impossible, it’s about making them too expensive to be worth it to the attacker
  - Example: There’s a simple method to break passwords: try them all

- Security is a negative goal
  - Proof that something can’t be done within some cost model is often followed by demonstration that it can be done by stepping outside the model
    - Example: dictionary attacks
      (Goal isn’t “break into account gwb,” it’s “break into any account”)

- There is a long-standing debate about the roles of prevention and retaliation
  - Steel plates over your doors and windows or deadbolts and the legal system?

- To publish or not to publish?
  - “Security through obscurity”

Attack / Threat Models



- eavesdropper
- man-in-the-middle
- replay attack
- spoof
- phishing
- …

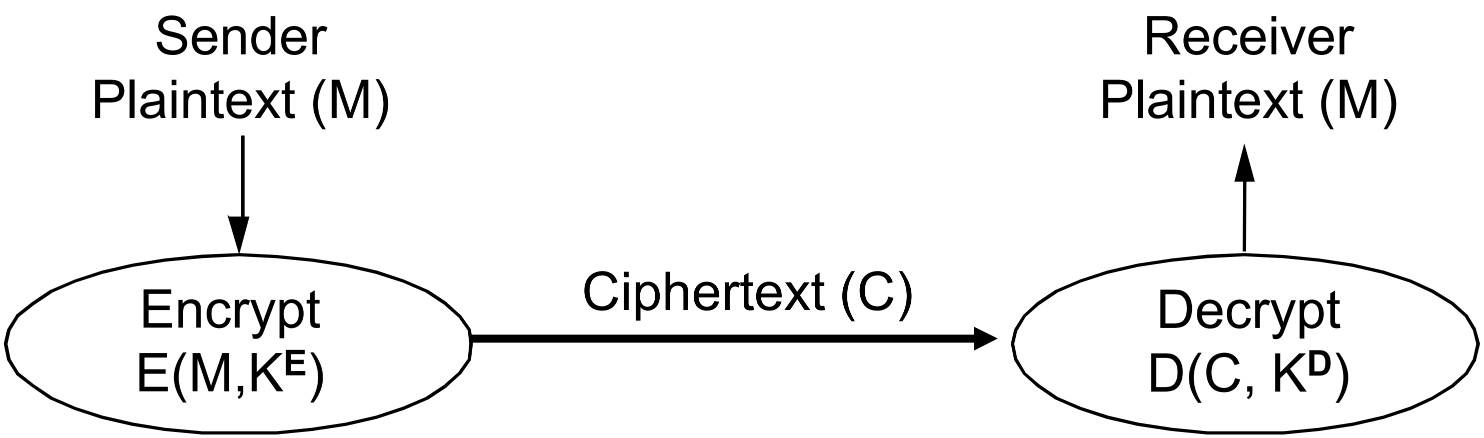
Part I: Privacy/Secrecy

- Main goal: prevent an eavesdropper from understanding what is being sent

Basic Tool: Cryptography

- Cryptography (encryption) directly addresses the eavesdropper problem
- It turns out it can also be used to address some of the other problems
  - E.g., authenticity
- Encryption is a building block
  - A *security protocol* is needed to achieve some more complex goal

Basic Encryption for Privacy



- Cryptographer chooses functions E, D and keys K^E, K^D
  - Mathematical basis
- Cryptanalyst try to “break” the system
  - Depends on what is known: E and D, M and C?

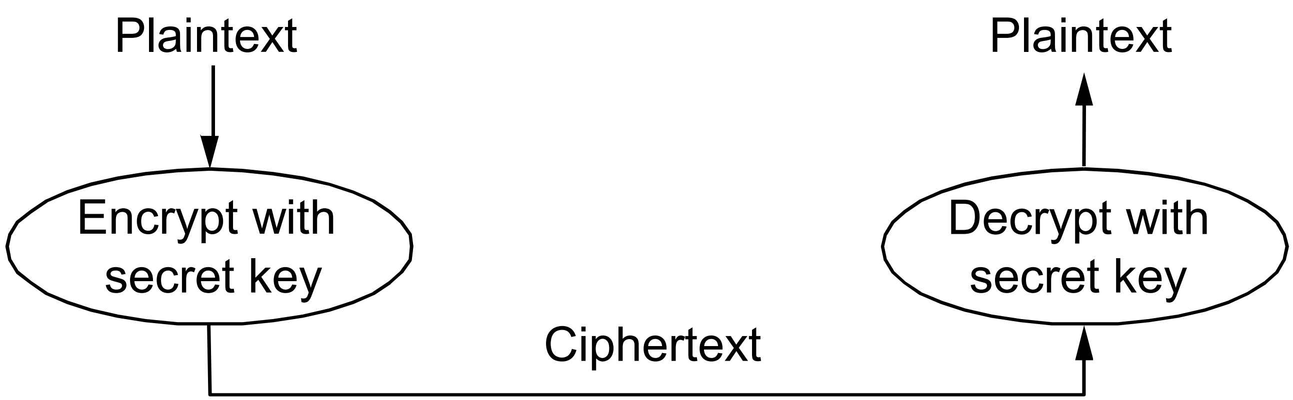
Perfect Secrecy: One Time Pad

- Messages
  - n-bit strings $[b_1,\ldots,b_n]$
- Keys
  - Random n-bit strings $[k_1,\ldots,k_n]$
- Encryption/Decryption
  - $c = E(b, k) = b \oplus k = [b_1 \oplus k_1, \ldots, b_n \oplus k_n]$
    - $\oplus$ denotes exclusive or
  - $b = D(b, k) = c \oplus k = b \oplus k \oplus k = b \oplus [0, \ldots, 0] = b$
- Properties
  - Provably unbreakable if used properly
  - Keys must be truly random
  - must not be used too often
  - Key same size as message

Simple Permutation Cipher

- Messages
  - n-bit strings $[b_1,\ldots,b_n]$
- Keys
  - Permutation $\square$ of n
  - Let $\blacklozenge = \square^{-1}$
- Encryption/Decryption
  - $E([b_1,\ldots,b_n], \square) = [b_{\square(1)},\ldots,b_{\square(n)}]$
  - $D([b_1,\ldots,b_n], \square) = [b_{\blacklozenge(1)},\ldots,b_{\blacklozenge(n)}]$
- Properties
  - Cryptanalysis possible

Secret Key Functions (DES, IDEA)



- Also called "shared secret"
- Single key (symmetric) is shared between parties
  - Used both for encryption and decryption
- Pro's:
  - Fast; hard to break given just ciphertext
- Con's:
  - key distribution problem
    - Suppose you want to create an account at youTube.com?
- The key distribution problem is crippling
  - Every client must share a (distinct!) secret with every server

Data Encryption Standard (DES)

- History
  - Developed by IBM, 1975
  - Modified slightly by NSA
  - U.S. Government (NIST) standard, 1977
- Algorithm
  - Uses 64-bit key, really 56 bits plus 8 parity bits
  - 16 "rounds"
    - 56-bit key used to generate 16 48-bit keys
    - Each round does substitution and permutation using 8 S-boxes
- Strength
  - Difficult to analyze
  - Cryptanalysis believed to be exponentially difficult in number of rounds
  - No currently known attacks easier than brute force
  - But brute force is now (relatively) easy

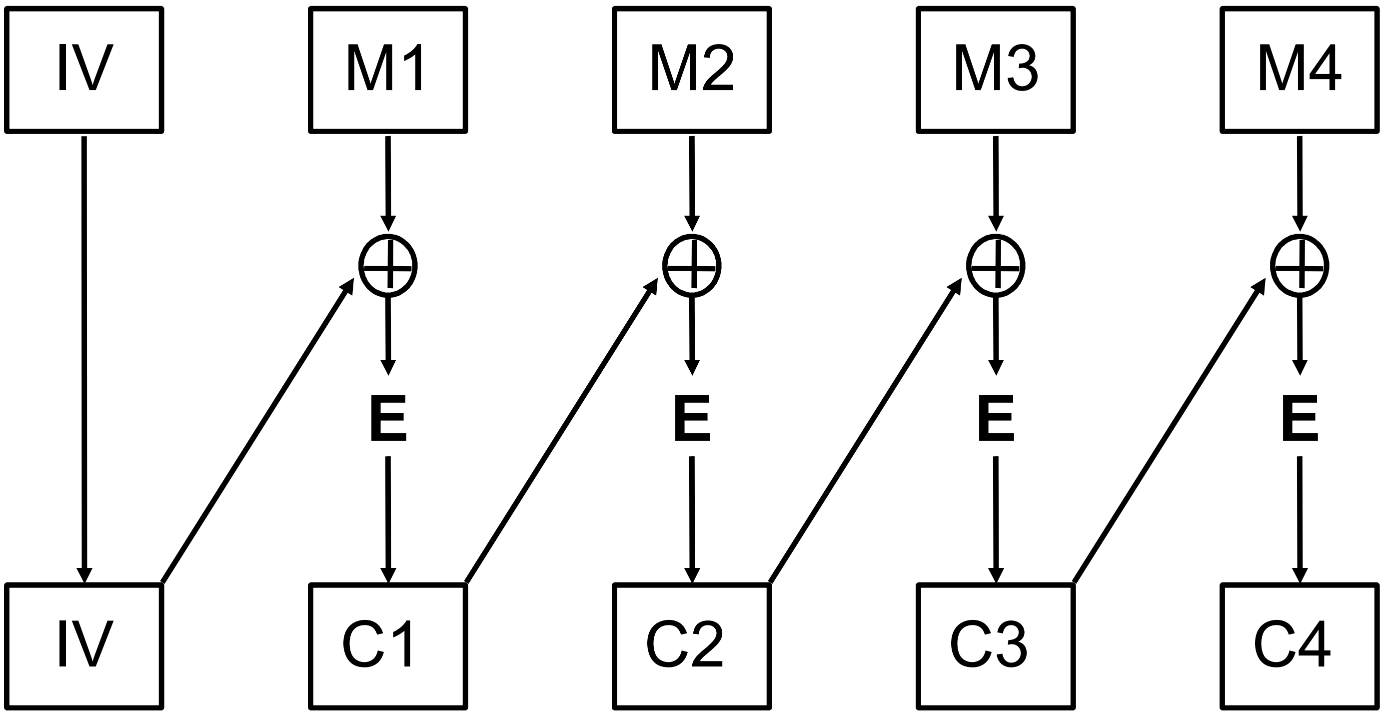
Other Ciphers

- Triple-DES
  - DES three times
    - $m_c = E(D(E(m_p, k_1), k_2, k_3)$
  - Effectively 112 bits
  - Three times as slow as DES
- Blowfish
  - Developed by Bruce Schneier circa 1993
  - Variable key size from 32 to 448 bits
  - Very fast on large general purpose CPUs (modern PCs)
  - Not very easy to implement in small hardware
- Advanced Encryption Standard (AES)
  - Selected by NIST as replacement for DES in 2001
  - Uses the Rijndael algorithm
  - Keys of 128, 192 or 256 bits

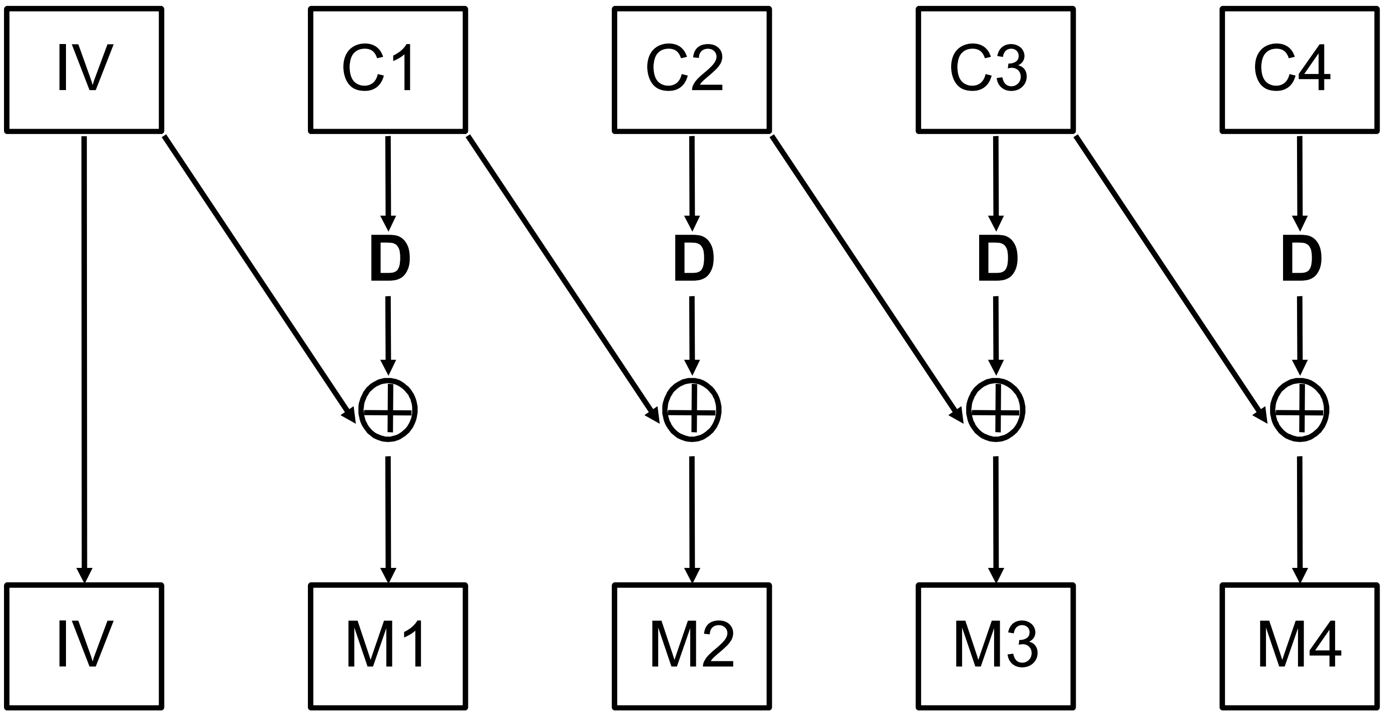
Encrypting Large Messages

- 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)
  - Leaks data: repeated plaintext blocks yield repeated ciphertext blocks
  - Does not guarantee integrity!
- Other modes “chain” to avoid this (CBC, CFB, OFB)

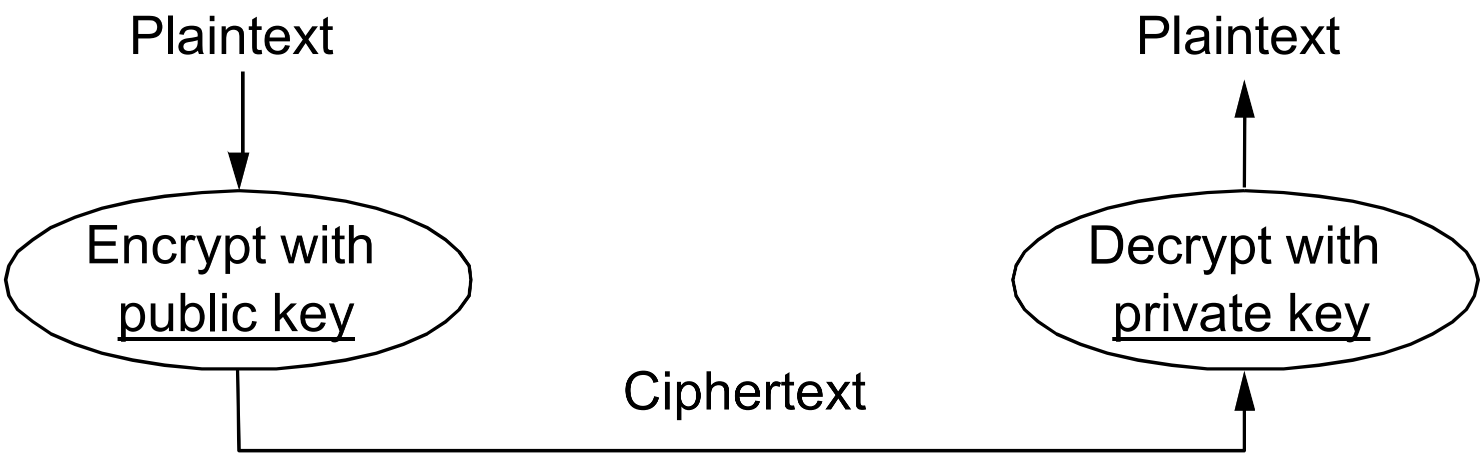
CBC (Cipher Block Chaining)



CBC Decryption



Public Key Functions (RSA)



- Public key can be published; private is a secret
  - Still have a key distribution problem, though…

RSA scheme

- Choose primes p and q, and let n = pq
- Find e and d such that ed mod (p-1)(q-1) = 1
  - Nits: e < (p-1)(q-1) and coprime with it.
- Public key is (n, e), private key is (n, d)

- To encrypt: c = m^e mod n
- To decrypt: m = c^d mod n
- This works because:
  - c^d mod n = m^ed mod n = m mod n by Euler's theorem
- Best approach to compute m w/o d is to factor n

- Had enough?

Properties of Public Key Encryption

- Let K^1 be the private key, and K^* be the public key
- $D(E(M,K^*), K^1) = M = D(E(M,K^1), K^*)$
- Implications
  - Anonymous client can send private message to server knowing only K^*
  - Server can prove authenticity by encrypting with K^1

Improving performance

- Public key crypto is sloooow compared to secret key:
  - MD5: 600 Mbps, DES: 100 Mbps, RSA: 0.1 Mbps (from P&D)
- But public key is more convenient & secure in setting up keys
- We can combine them to get the best of both
- Hybrid encryption: encrypt message with random secret key and encrypt secret key with public key.

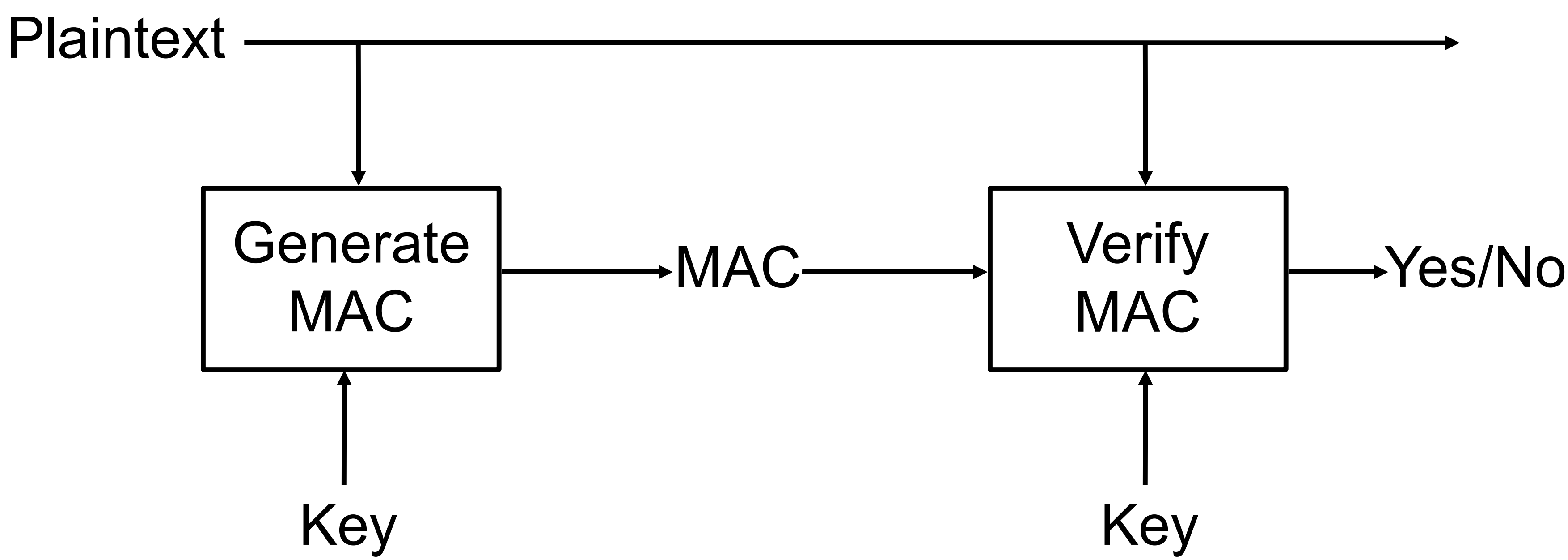
Part II: Integrity & Authenticity

- Main goal: verify that a message has not been altered and that it comes from who it claims

- Message Authentication Code (MAC) allows verifiers (who hold the secret key) to detect changes to content.
  - Sometimes called a MIC, I = Integrity
- Digital signatures allow recipients to verify message integrity and authenticity

- Q: why isn't encryption enough?

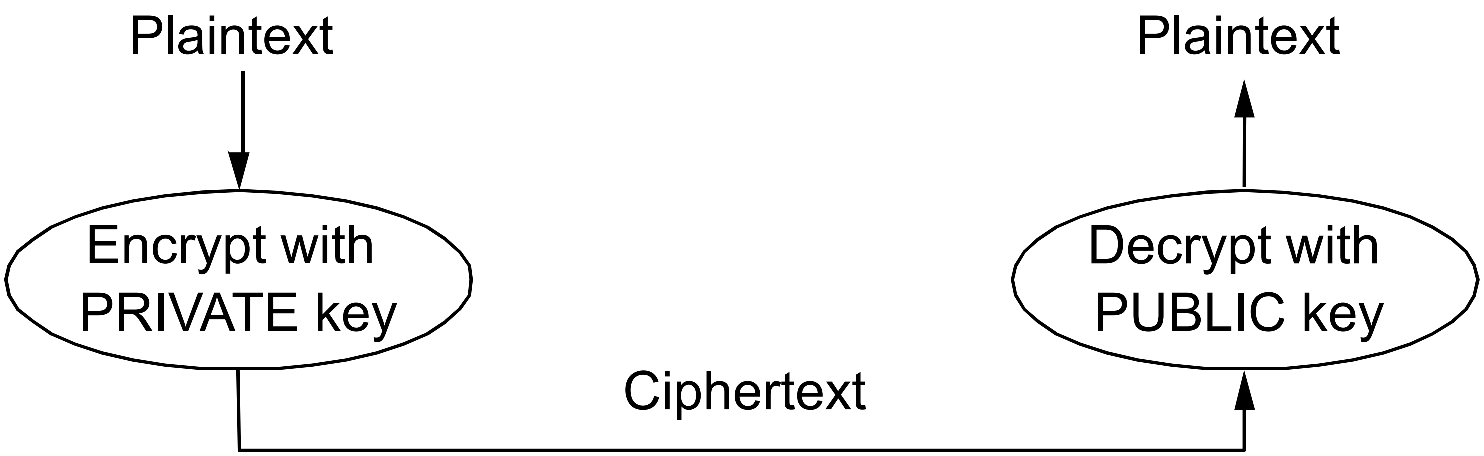
Secret Key Integrity



E.g.: Use DES in CBC-MAC mode (with IV of 0) and the residue (last encryption) is the MAC

Need to use a different key than for secrecy!

RSA Digital Signature



- Notice that we reversed the role of the keys (and the math just works out) so only one party can send the message but anyone can check it's authenticity

A Faster "RSA Signature"

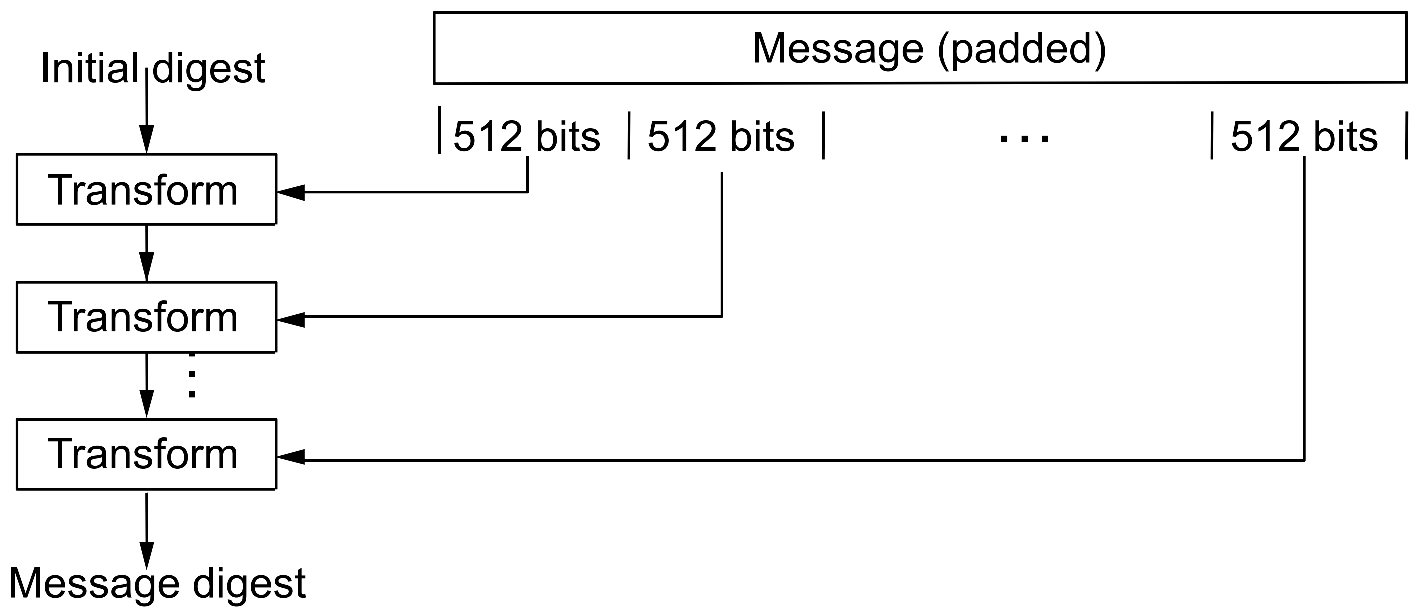
- Encryption can be expensive, e.g., RSA 1Kbps
- To speed up, let's sign just the checksum instead!
  - Check that the encrypted bit is a signature of the checksum
- Problem: Easy to alter data without altering checksum
- Answer: Cryptographically strong "checksums"

Cryptographic Hash

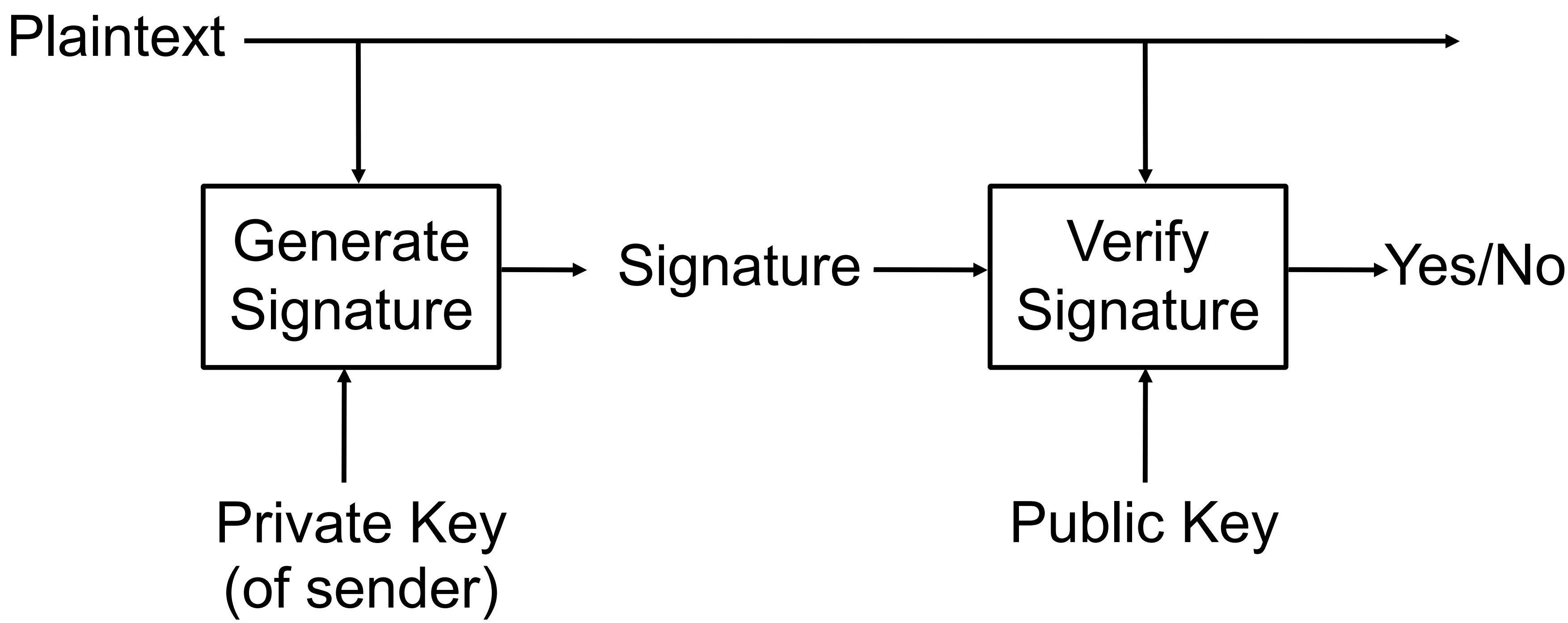
- Basically:
  - A hash function (maps arbitrary sized data to a fixed number of bits)
  - Given message M, is cheap to compute
  - Give a hash value, it's hard to find data that produces that value
    - Ideally, a change to any one bit of the message flips each bit of the hash value with probability 0.5
- Result:
  - Even if the attacker knows the authenticator value, can't produce bogus data that matches it

Message Digests (MD5, SHA)

- Act as a cryptographic checksum or hash
  - Typically small compared to message (MD5 128 bits)
  - “One-way”: infeasible to find two messages with same digest



Public Key Integrity Protection



Keyed Hash MAC (HMAC)

- **Start with HMAC = H**(K,m), but it's vulnerable.

- **From RFC 2104:**
- **HMAC**(K,m) = **H**(($K \oplus$ opad) ‖ **H**(($K \oplus$ ipad) ‖ m))
  - $\oplus$ is XOR, opad = 0x5c5c5c…, ipad = 0x363636 ...

Part III: Authentication

- Main goal: Verify that you are talking to who you think you are talking to.

Private Key Authentication

- Alice wants to talk to Bob
  - Needs to convince him of her identity
  - Both have private key k
- Naive scheme

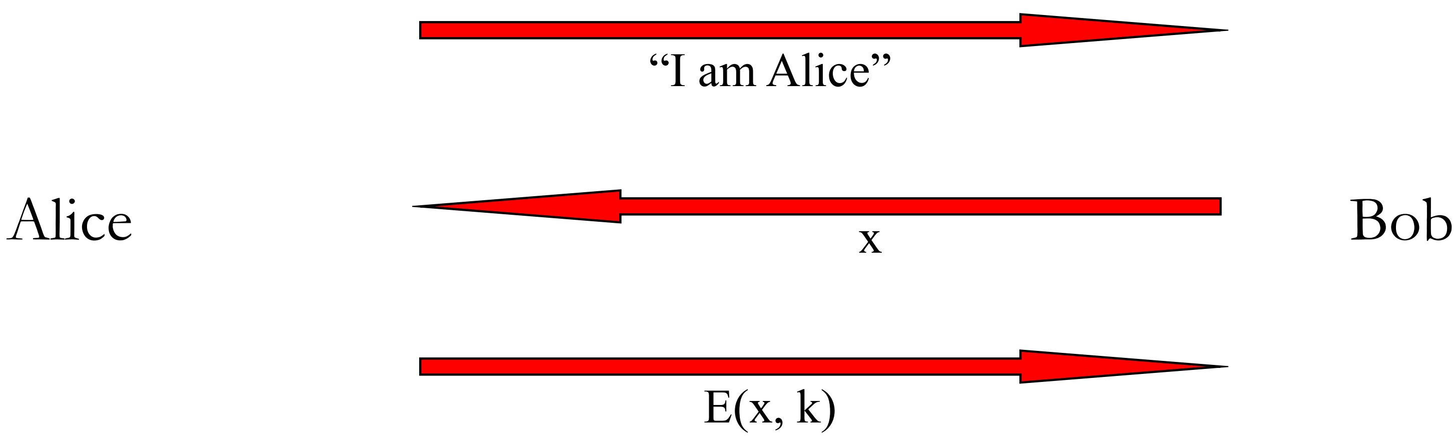
Alice → Bob

"I am Alice", x, E(x, k)

- Vulnerability?

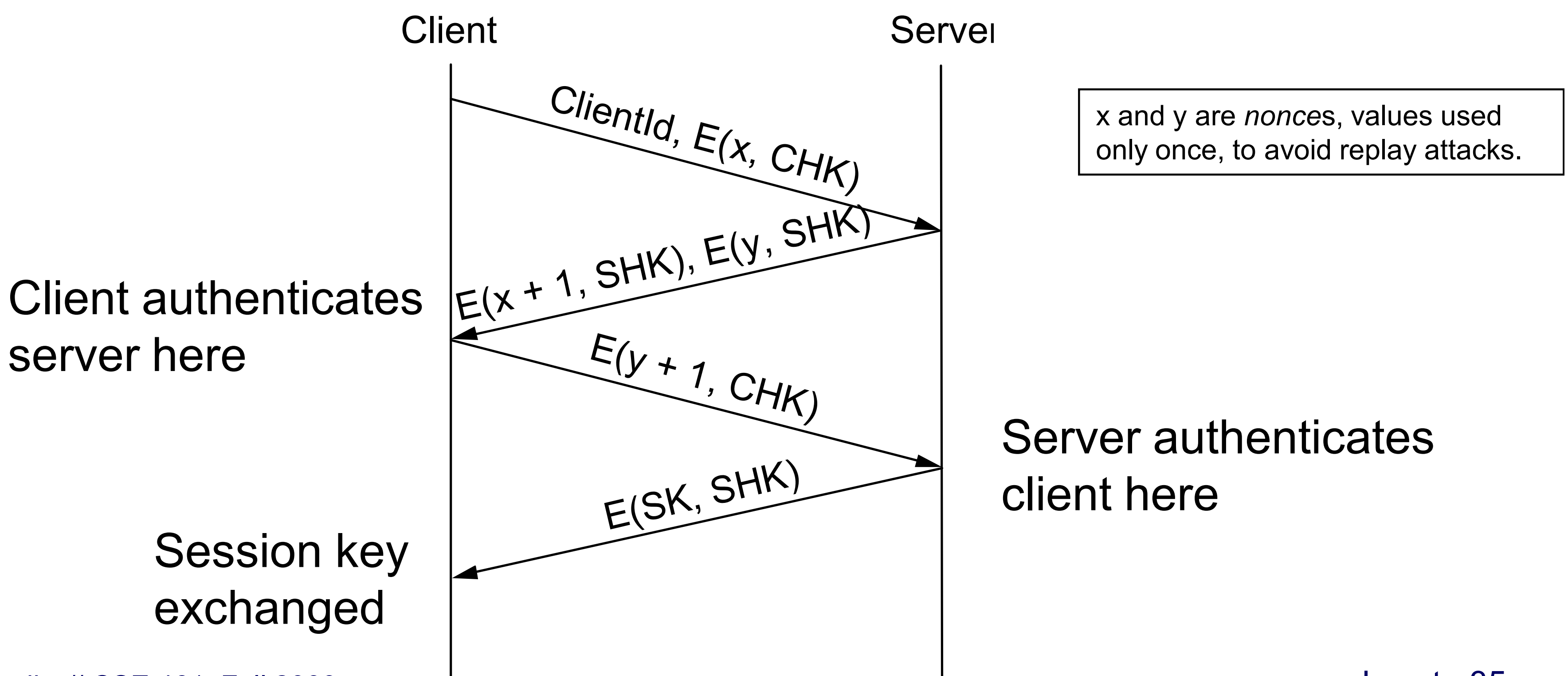
Preventing Replay Attacks

- Bob can issue a challenge phrase to Alice

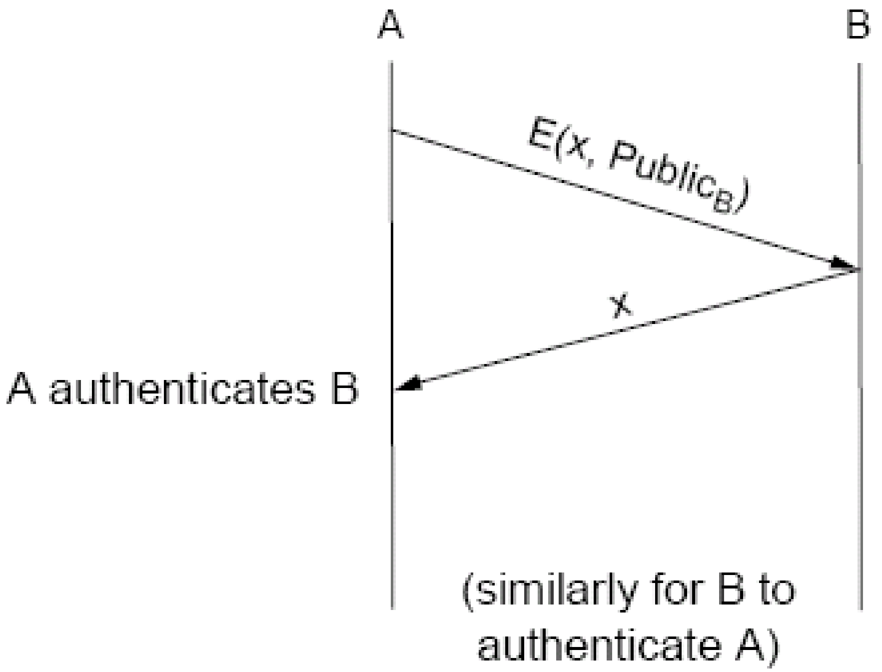


Authentication w/ Shared Secret

- Three-way handshake for mutual authentication
  - Client and server share secrets, e.g., login password

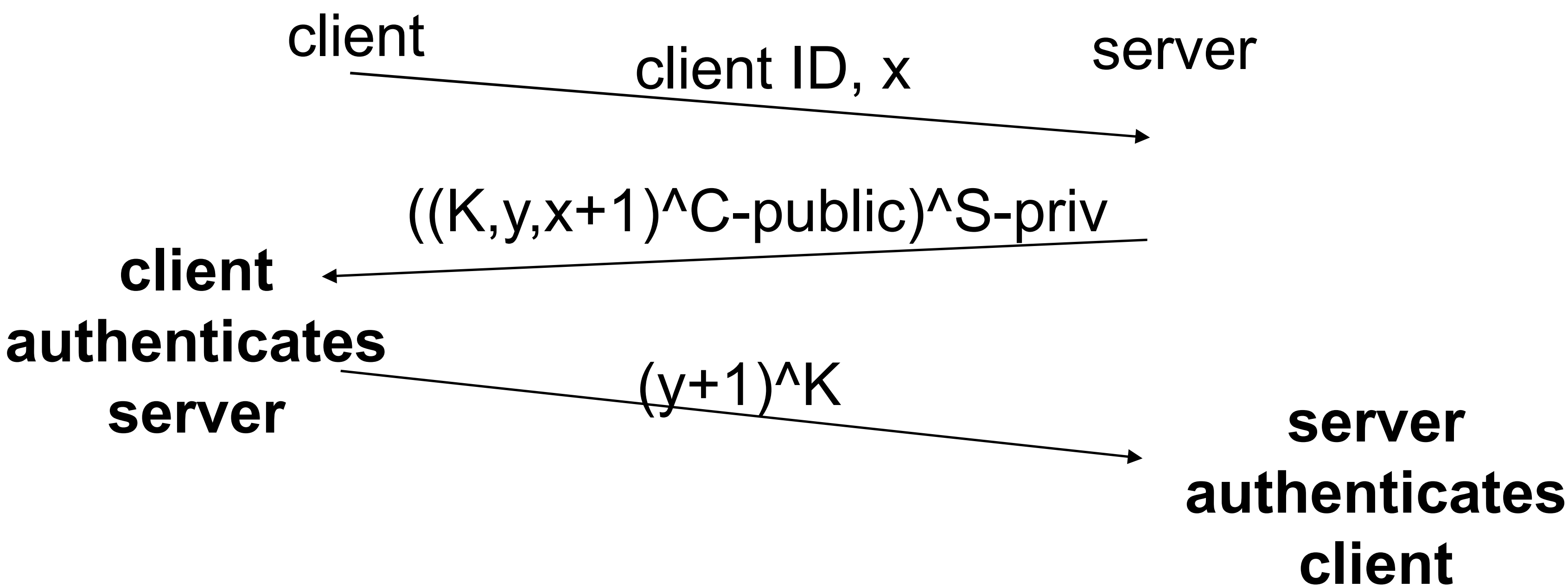


Public Key Authentication



Public Key → Session Key

- Ask other side to decrypt/sign to prove they hold the keys and use public keys to establish (shared) session key



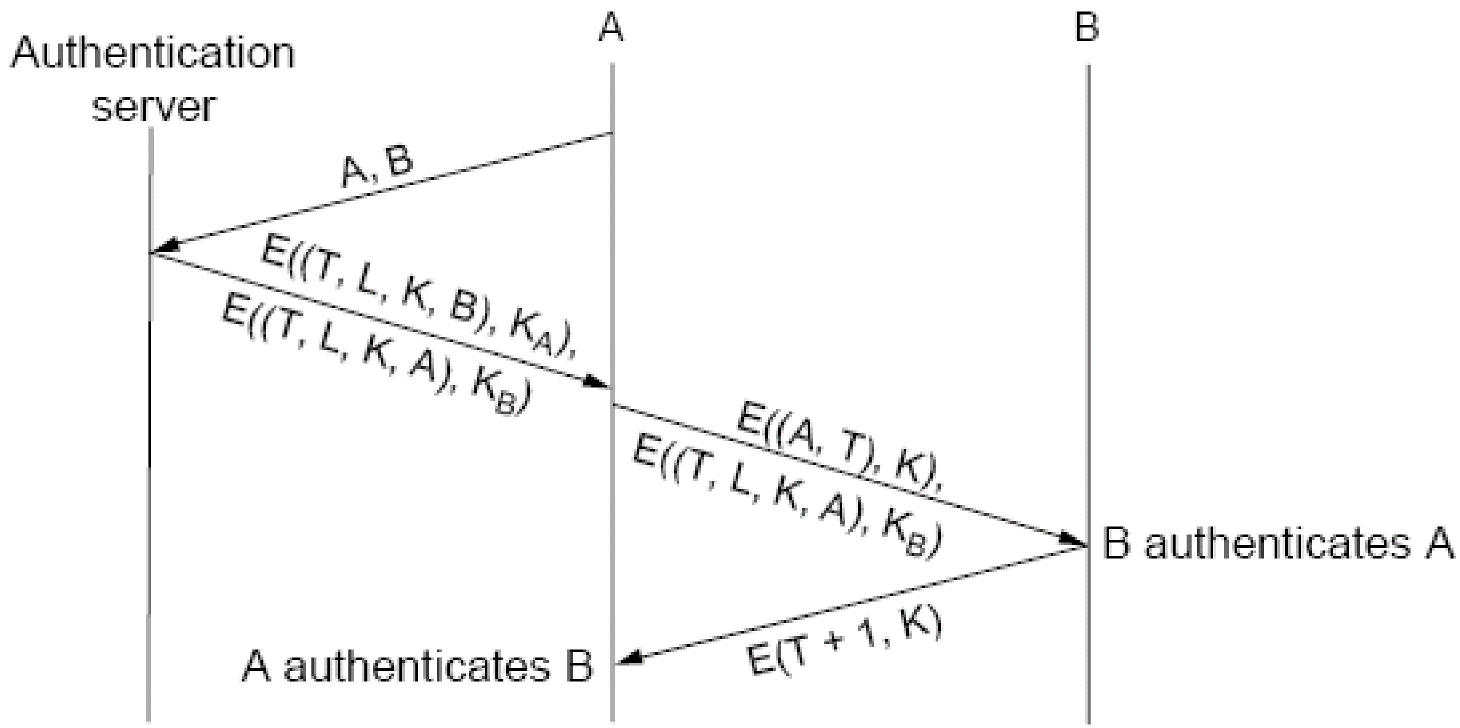
Part IV: Key Distribution

- These keys need to come from somewhere … Achilles heel
- In a large network, we're going to need to trust someone to either
  - 1) establish new shared secrets (session keys), or
  - 2) vouch for public keys.

Kerberos

- Have network with n entities
- Add one more
  - Must generate n new keys
  - Each other entity must securely get its new key
  - Big headache managing n^2 keys!
- Kerberos solution: use a central keyserver
  - Needs n secret keys between entities and keyserver
  - Generates session keys as needed
  - Downsides
    - Only scales to single organization level
    - Single point of failure

Kerberos as Trusted Third Party

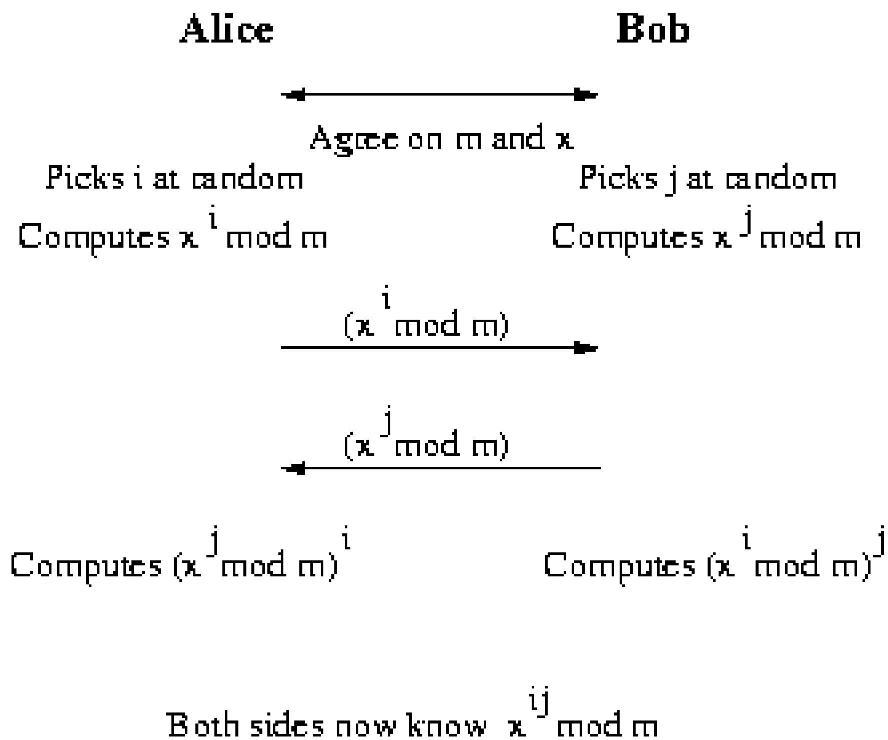


Diffie-Hellman Key Agreement

- History
  - Developed by Whitfield Diffie, Martin Hellman
  - Published in 1976 paper "New Directions in Cryptography"
- Allows negotiation of secret key over insecure network
- Algorithm
  - Public parameters
    - Prime p
    - Generator $g < p$ with property: ✂n: 1⭘n⭘p-1, 👓k: $n = g^k \bmod p$
  - Alice chooses random secret a, sends Bob g^a
  - Bob chooses random secret b, sends Alice g^b
  - Alice computes $(g^b)^a$, Bob computes $(g^a)^b$ – this is the key
  - Difficult for eavesdropper Eve to compute g^{ab}

Diffie-Hellman Key Exchange

- Problem: agree on a session key with no prior information exchanged



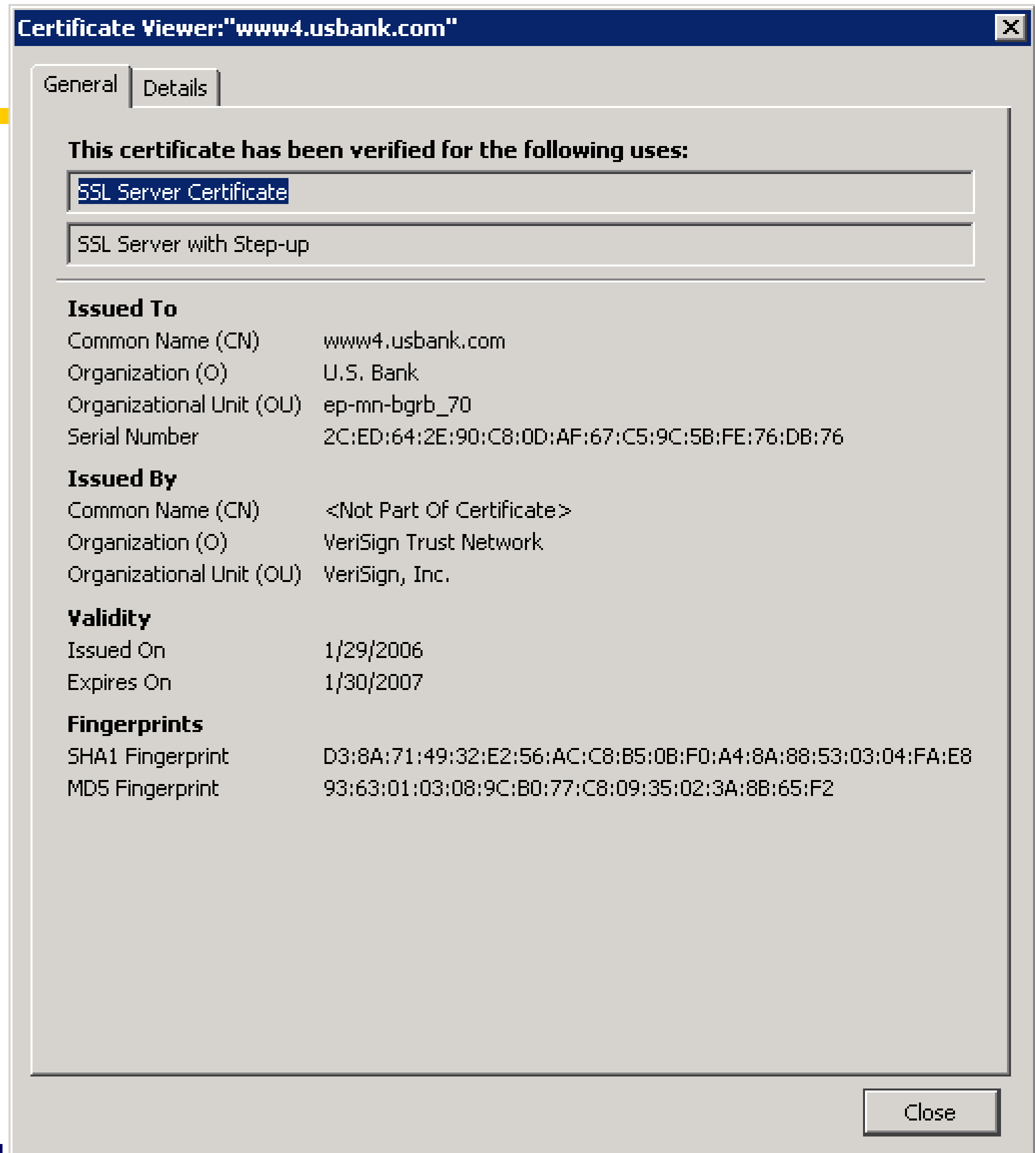
Diffie-Hellman Weakness

- Man-in-the-Middle attack
  - Assume Eve can intercept and modify packets
  - Eve intercepts g^a and g^b, then sends Alice and Bob g^c
  - Now Alice uses g^{ac}, Bob uses g^{bc}, and Eve knows both
- Defense requires mutual authentication
  - Back to key distribution problem

Public Key Authentication Chains

- How do you trust an unknown entity?
- Trust hierarchies ("CA says public key for X is K")
  - Certificates issued by Certificate Authorities (CAs)
    - Certificates are signed by only one CA
    - Trees are usually shallow and broad
    - Clients only need a small number of root CAs
      - Roots don't change frequently
      - Can be distributed with OS, browser
    - Problem
      - Root CAs have a lot of power
      - Initial distribution of root CA certificates
  - X.509
    - Certificate format standard
    - Global namespace: Distinguished Names (DNs)
      - Not very tightly specified – usually includes an email address or domain name

X.509 Certificates



Public Key Revocation

- What if a private key is compromised?
  - Hope it never happens?

- Need certificate revocation list (CRL)
  - and a CRL authority for serving the list
  - everyone using a certificate is responsible for checking to see if it is on CRL
  - ex: certificate can have two timestamps
    - one long term, when certificate times out
    - one short term, when CRL must be checked
    - CRL is online, CA can be offline