CS 423
Operating System Design: OS Security Crash Course

Tianyin Xu

Thanks for Prof. Adam Bates for the slides.

Security Properties

- Confidentiality?
- Integrity?
- Availability?
- Authenticity?

Security Properties

- Confidentiality?
  - Only trusted parties can read data
- Integrity?
  - Only trusted parties have modified data
- Authenticity?
  - Data originates from the correct party
- Availability?
  - Data is available to trusted parties when needed

Security Functions

- Define security functions over *principals* (e.g., users, programs, sysadmins)
- … and also *entities* (e.g., files, network sockets, ipc)
- Authentication
  - How do we determine the identity of the principal?
- Authorization
  - Which principals are permitted to take what actions on which objects?
- Auditing
  - Record of (un) authorized actions that took place on the system for post-hoc diagnostics

Authorization

- <u>Access control matrix</u>
  - For every protected resource, list of who is permitted to do what
  - Example: for each file/directory, a list of permissions
    - Owner, group, world: read, write, execute
    - Setuid: program run with permission of principal who installed it
  - Smartphone: list of permissions granted each app

Question

Access control matrices allow us to specify an arbitrary ***security policy****… what properties should our security policy provide?*

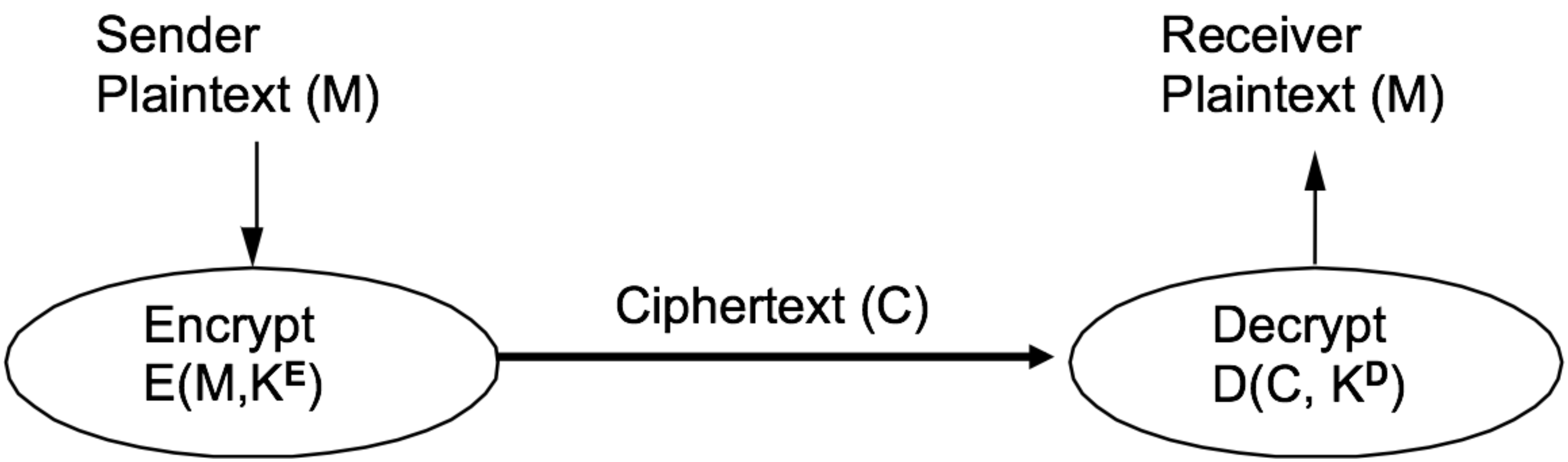
Principle of Least Privilege

- Grant each principal the least permission possible for them to do their assigned work
  - Minimize code running inside kernel
  - Minimize code running as sysadmin
- Practical challenge: hard to know
  - what permissions are needed in advance
  - what permissions should be granted
    - Ex: to smartphone apps
    - Ex: to servers

Authorization w/ Intermediaries

- **Trusted Computing Base (TCB)**: set of software trusted to enforce security policy
- Is it good or bad to have a large TCB?
- Ex: Storage Server is trusted to check user access control list
  - Why? Because server must store/retrieve data on behalf of all users.
  - Implication? security flaw in server allows attacker to take control of system

Encryption



- Cryptographer chooses functions E, D and keys K^E, K^D
  - Suppose everything is known (E, D, M and C), should not be able to determine keys K^E, K^D and/or modify msg
  - provides basis for authentication, privacy and integrity

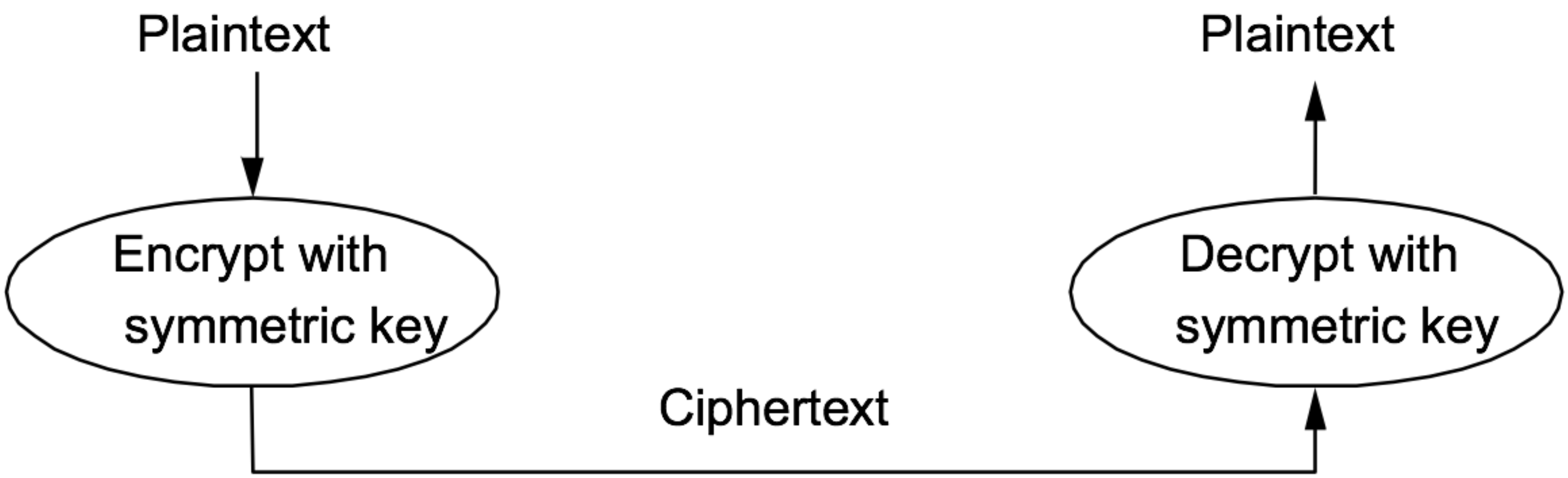
Authentication

- How do we know user is who they say they are?
- Try #1: user types password (something they know)
  - User needs to remember password!
  - Short passwords: easy to remember, easy to guess
  - Long passwords: hard to remember

Question

- Where are passwords stored?
  - Password is a per-user secret
  - In a file?
    - Anyone with sysadmin permission can read file
  - Encrypted in a file?
    - If gain access to file, can check passwords offline
    - If user reuses password, easy to check against other systems
  - Encrypted in a file with a random salt?
    - Hash password and salt before encryption, foils precomputed password table lookup

Symmetric Key (DES, AES)



- Single key (symmetric) is shared between parties, kept secret from everyone else
  - Ciphertext = (M)^K; Plaintext = M = ((M)^K)^K
  - if K kept secret, then both parties know M is authentic and secret

Authentication

- How do we know user is who they say they are?
- Try #2: user has secret key
  - User needs to safely store the secret!
  - Is system configured s.t. it can protect the confidentiality of the secret key?
  - Can the user prove they know the secret without giving it to the other part?

Public Key (RSA, PGP)

Keys come in pairs: public and private

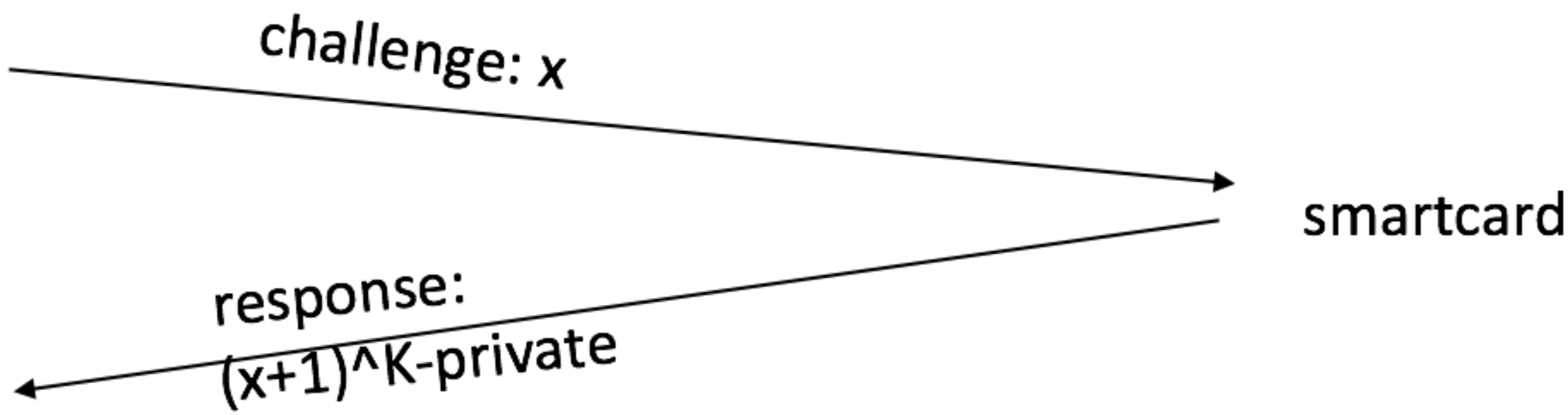
- M = ((M)^K-public)^K-private
- Ensures secrecy: can only be read by receiver

Encryption Summary

- Symmetric key encryption
  - Single key (symmetric) is shared between parties, kept secret from everyone else
  - Ciphertext = (M)^K
- Public Key encryption
  - Keys come in pairs, public and private
  - Secret: (M)^K-public
  - Authentic: (M)^K-private

2-Factor Authentication

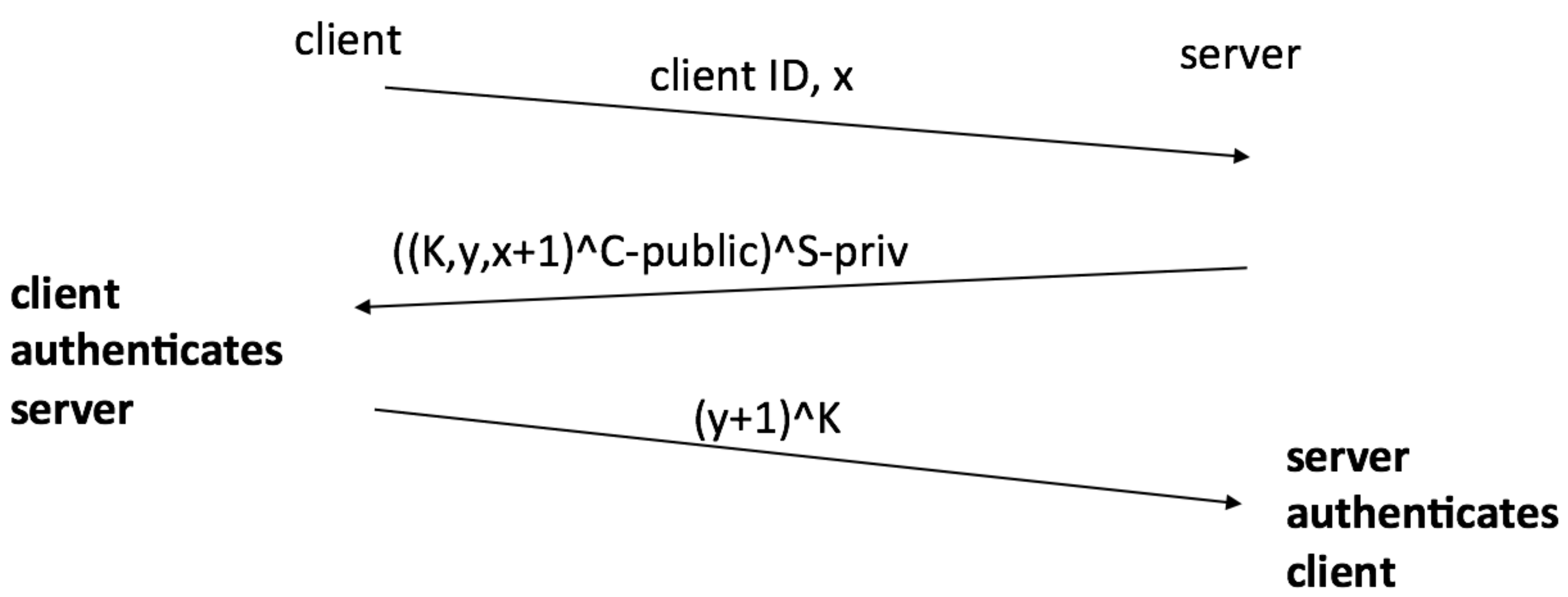
- Can be difficult for people to remember encryption keys and passwords
- Instead, store private key (K-private) inside a chip
  - Use PIN/PW to prove user is authorized (something user knows)
  - Use challenge-response to authenticate smartcard (something user has)



Are there other authentication factors?

Public Key to Session Key

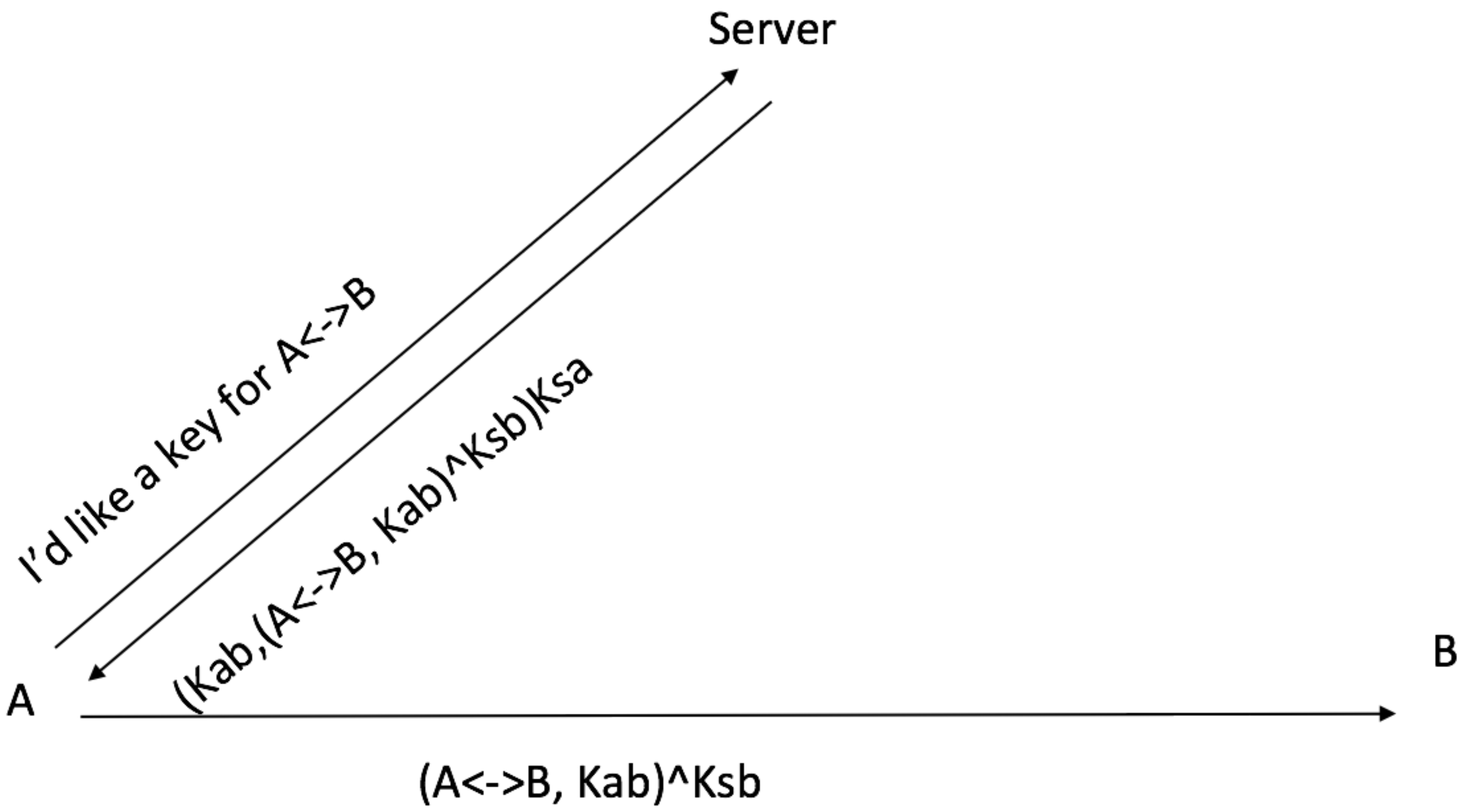
- Public key encryption/decryption is slow; so can use public key to establish (shared) session key
  - assume both sides know each other's public key



Symmetric Key to Session Key

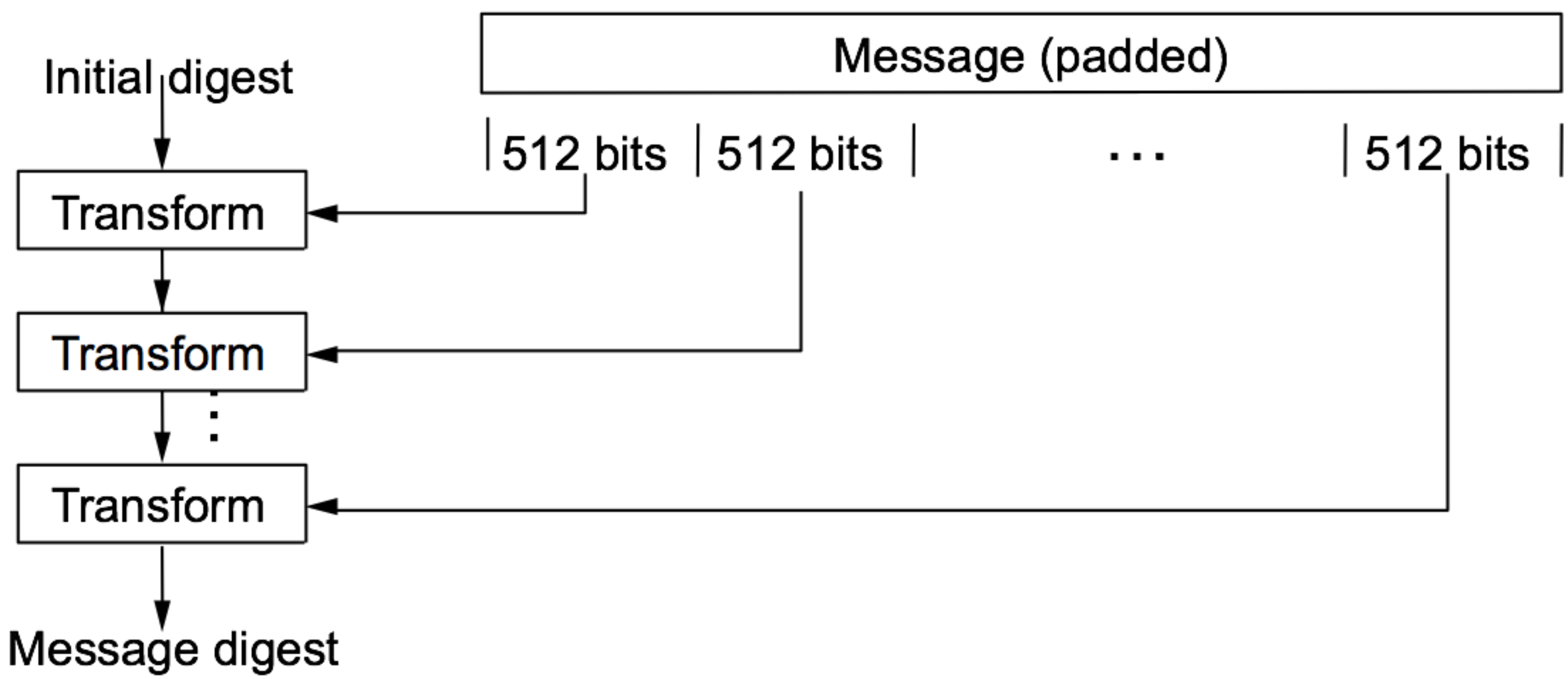
- In symmetric key systems, how do we gain a session key with other side?
  - infeasible for everyone to share a secret with everyone else
  - solution: “authentication server” (Kerberos)
    - everyone shares (a separate) secret with server
    - server provides shared session key for A <-> B
  - everyone trusts authentication server
    - if compromise server, can do anything!

Kerberos Example



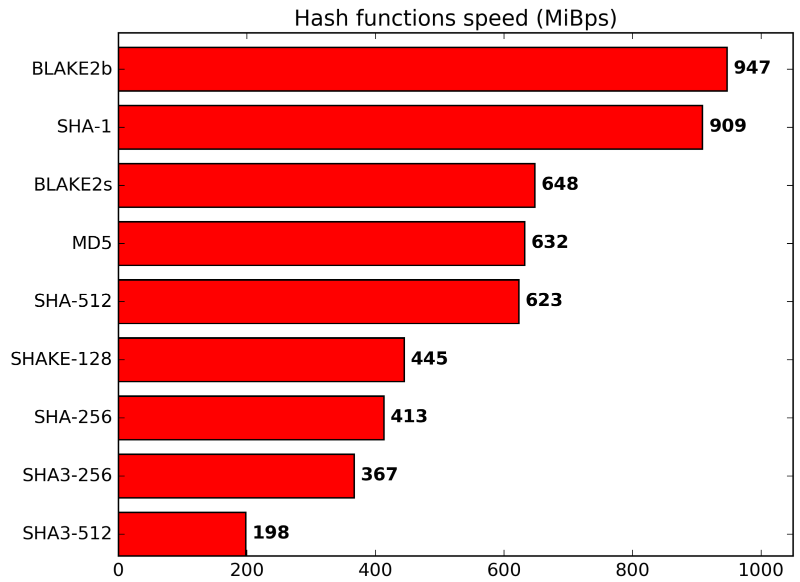
Message Digest (MD5, SHA)

- Cryptographic checksum: message integrity
  - Typically small compared to message (MD5 128 bits)
  - "One-way": infeasible to find two messages with same digest



Cryptographic Hash Functions

- What properties do we need?
  1. Deterministic
  2. Quick
  3. One-Way
  4. "Avalance Effect"
  5. Collision Resistant
  6. Pre-image attack resistant
- How does this compare to non-crypto hashes?



All of these functions were once thought to be cryptographically strong (some are not)… Seahash outperforms BLAKE(2) by 32x!

Security Practice

- In practice, systems are not that secure
  - hackers can go after weakest link
    - any system with bugs is vulnerable
  - vulnerability often not anticipated
    - usually not a brute force attack against encryption system
  - often can't tell if system is compromised
    - hackers can hide their tracks
  - can be hard to resecure systems after a breakin
    - hackers can leave unknown backdoors