جامعة الامير سلطان
PRINCE SULTAN UNIVERSITY

CS435 Distributed systems

SECURITY IN DIST SYSTEMS

Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

TOPICS

- Confidentiality
- Integrity
- Authentication

CS435 Distributed systems

CIA

Dr. Basit Qureshi

PhD FHEA SMIEEE MACM

www.drbasit.org

CIA

- **C**onfidentiality: Hide data and resources
- Data **I**ntegrity: Data is not modified or destroyed
- **A**vailability : Ability of access data/resources

Turning off a computer provides confidentiality & integrity but hurts availability

DDOS Attack affects availability

CIA

Security is not just adding encryption …

- or using a 512-bit key instead of a 64-bit key …
- or changing passwords …
- or setting up a firewall

- **It is a SYSTEMS ISSUE = Hardware + firmware + OS + app software + networking + people = Processes & procedures, policies, detection, forensics**

"Security is a chain: it's only as secure as the weakest link" – Bruce Schneier

CIA

The OS handles security issues

- User authentication – passwords, etc
- Access control – file permissions, etc
- Resource management – memory limits, etc

- But it can only control resources it owns

Distributed systems

- Use components that belong to different entities
- Programs may:
  - Call remote services – are they trustworthy?
  - Receive requests – are they from a legitimate & authorized user or service?
  - Store data on remote servers – who manages them?
  - Send data over a network – what route do the packets take?

Cryptography is the solution!

CRYPTOGRAPHY

Confidentiality: Others cannot read contents of the message

Authentication: Determine origin of message

Integrity: Verify that message has not been modified

Non-repudiation: Sender should not be able to falsely deny that a message was sent

CS435 Distributed systems

CONFIDENTIALITY

Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

CONFIDENTIALITY

Encryption key terms:

- **Plaintext** (cleartext) message P
- **Cipher** = cryptographic algorithm
- **Encryption** E(P)
- Produces **Ciphertext**, C = E(P)
- **Decryption**, P = D(C)

CONFIDENTIALITY

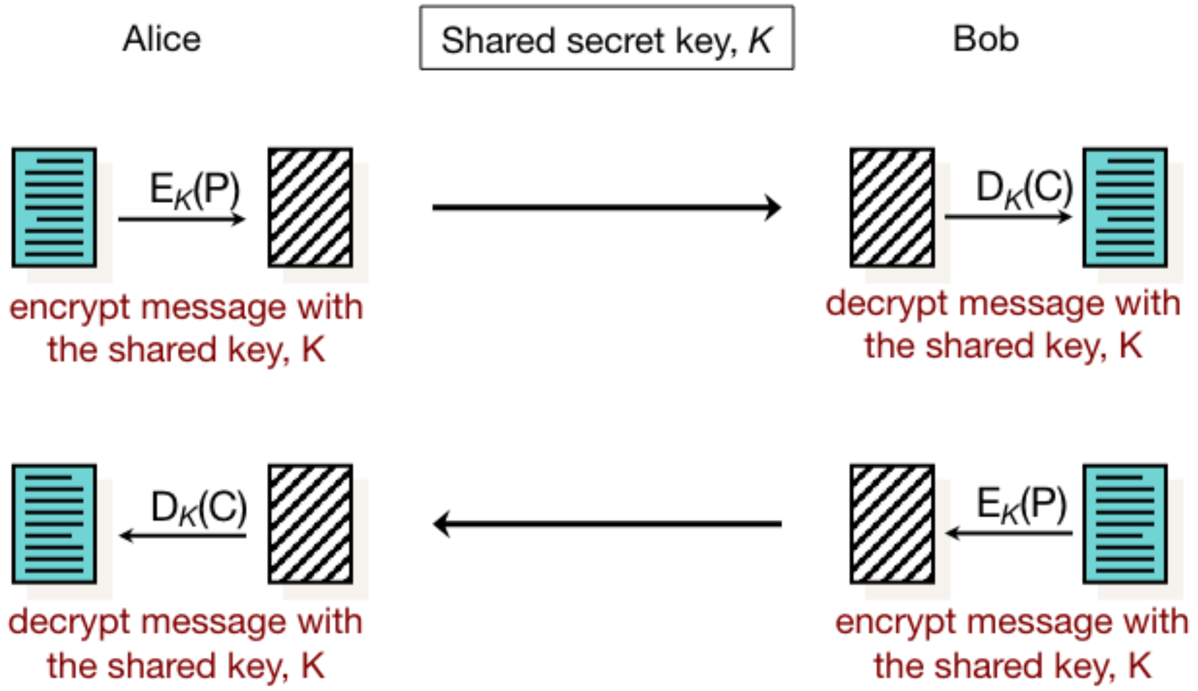
A good crypto system

- Ciphertext should be indistinguishable from random values
- Given ciphertext, there should be no way to extract the original plaintext or the key short of enumerating all possible keys (i.e., a brute force attack)
- The keys should be large enough that a brute force attack is not feasible

CONFIDENTIALITY

Symmetric Key cyphers

Same shared secret key, K, for encryption & decryption **C = $E_K(P)$; P = $D_K(C)$**



Popular symmetric cyphers: AES, DES, 3DES, ChaCha20, IDEA

CONFIDENTIALITY

Public Key cryptography

Two related keys (A, a)

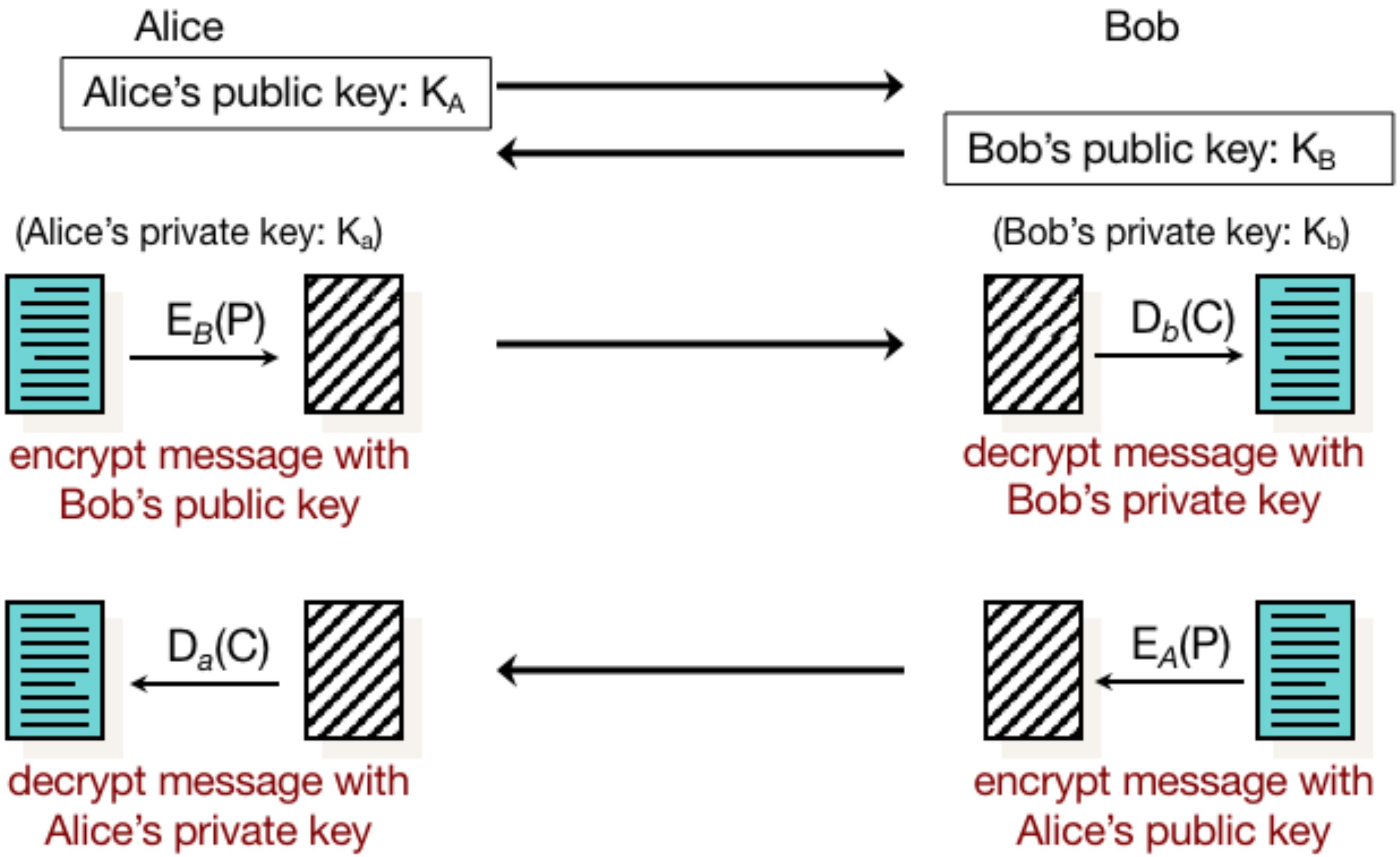
$C = E_A(P)$	**$P=D_a(C)$**	A is a **public** key
$C' = E_a(P)$	**$P=D_A(C')$**	a is a **private** key

Examples: RSA, Elliptic Curve Cryptography (ECC)

Different keys for encrypting and decrypting – No need to worry about secure key distribution

CONFIDENTIALITY

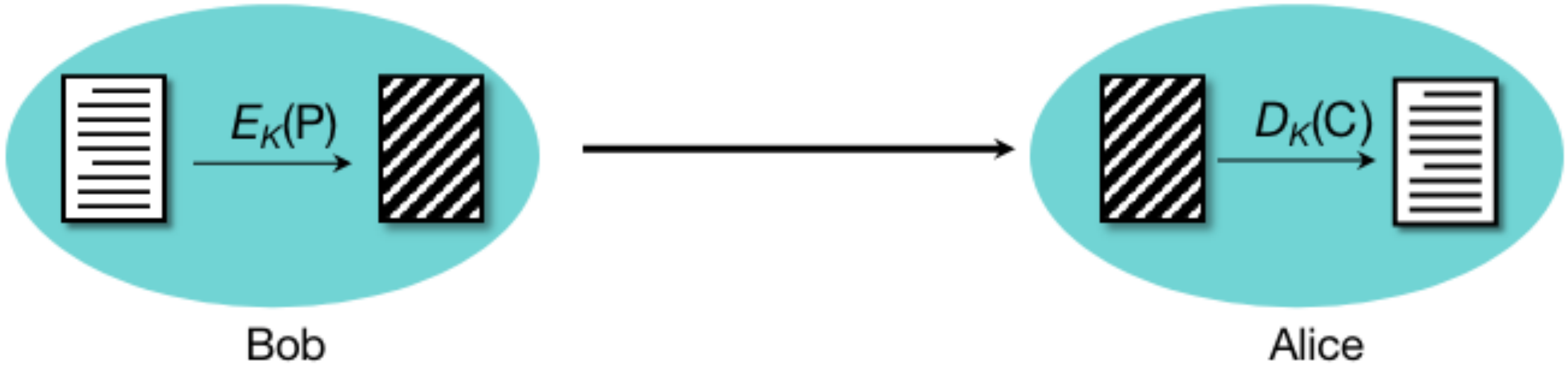
Public Key cryptography



CONFIDENTIALITY

- Both parties must agree on a secret key, K
- Message is encrypted, sent, decrypted at other side

Key distribution must be secret; Otherwise, messages can be decrypted; Users can be impersonated



Secure key distribution is the biggest problem with symmetric cryptography

CONFIDENTIALITY

Sharing Keys:

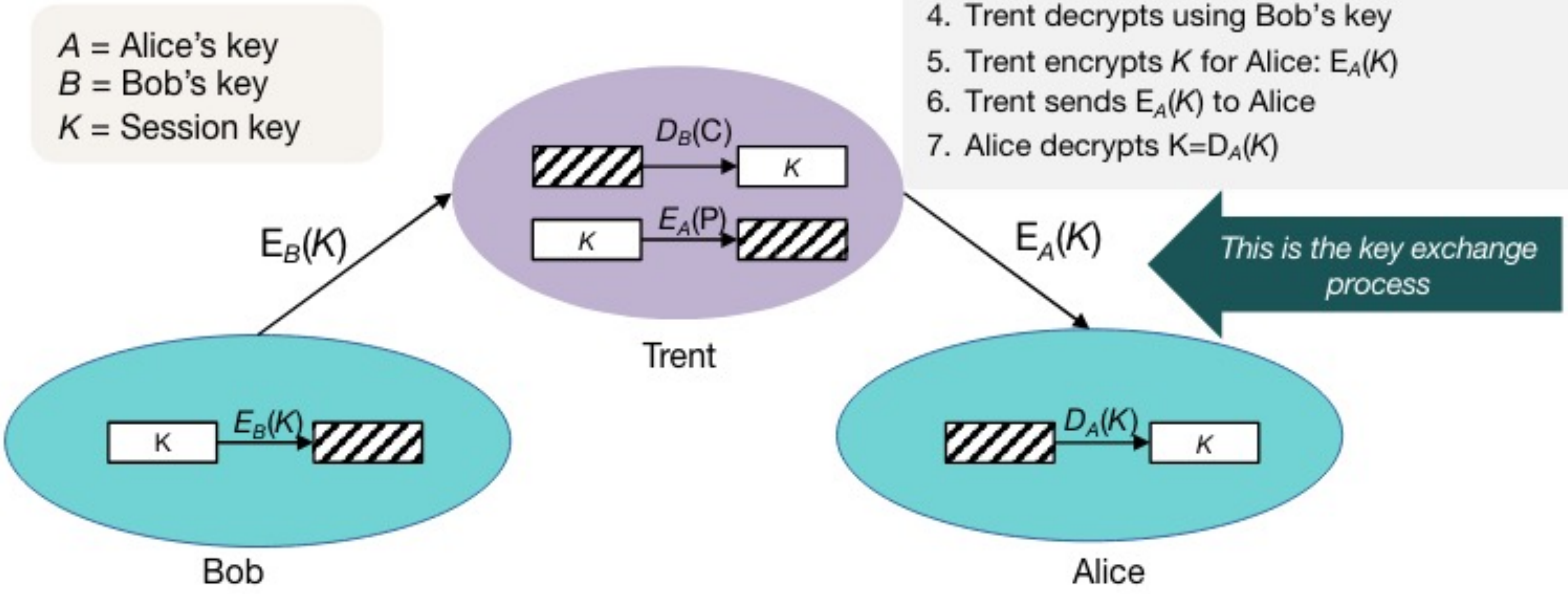
- **Pre-shared keys** – Initial configuration, out of band (send via USB key, recite, …)
- **Trusted third party**
  - Knows all keys
  - Alice creates a temporary key (session key)
  - Encrypts it with her key
  - sends to Trent
  - Trent decrypts it and sends it to Bob
  - Alternatively: Trent creates a session key – encrypts it for Alice & for Bob
- **Public key cryptography**
  - Alice encrypts a message with Bob's public key
  - Only Bob can decrypt

CONFIDENTIALITY

- **Trusted third party**

- Trusted third party, Trent, knows all the keys
- Everyone else only knows their own keys

1. Bob creates a random session key, K
2. Bob encrypts it with his secret key: $E_B(K)$
3. Bob sends $E_B(K)$ to Trent
4. Trent decrypts using Bob's key
5. Trent encrypts K for Alice: $E_A(K)$
6. Trent sends $E_A(K)$ to Alice
7. Alice decrypts $K=D_A(K)$



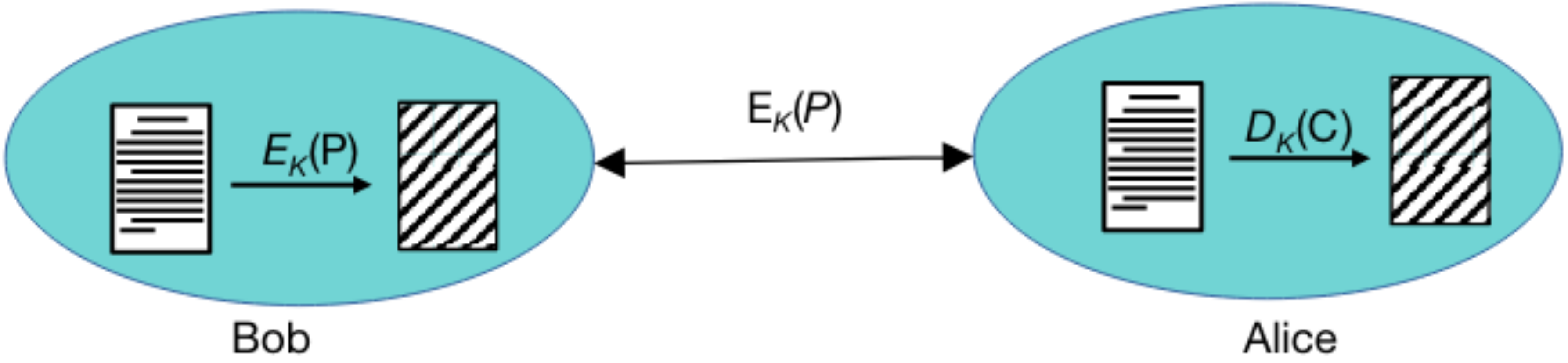
CONFIDENTIALITY

- **Trusted third party**

… continued

A = Alice's key
B = Bob's key
K = Session key

1. Bob creates a random session key, *K*
2. Bob encrypts it with his secret key: $E_B(K)$
3. Bob sends $E_B(K)$ to Trent
4. Trent decrypts using Bob's key
5. Trent encrypts *K* for Alice: $E_A(K)$
6. Trent sends $E_A(K)$ to Alice
7. Alice decrypts $K=D_A(K)$
8. **Alice & Bob communicate, encrypting messages with the session key, *K***



CS435 Distributed systems

INTEGRITY

Dr. Basit Qureshi

PhD FHEA SMIEEE MACM

www.drbasit.org

INTEGRITY

Use cryptographic techniques to detect that data has not been modified

Integrity mechanisms can help to

- Detect data corruption
- Detect malicious data modification
- Prove ownership of data

INTEGRITY

How do we detect that a message has been tampered?

A **hash** is a small, fixed amount of information that lets us have confidence that the data used to create the hash was not modified

- We associate a hash with a message
- We're not encrypting the message
- We're concerned with **integrity**, not **confidentiality**
- If two messages hash to different values, we know the messages are different

INTEGRITY

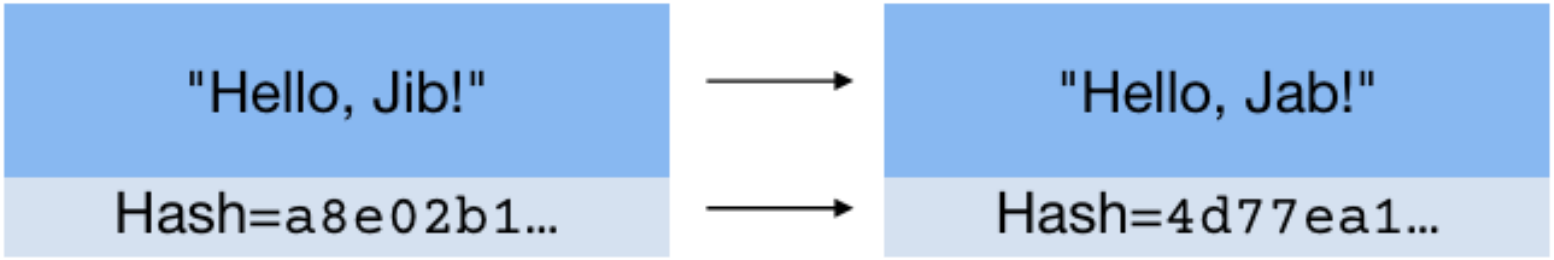
- Hash functions are the basis of integrity
- Not encryption
- Can help us to detect:
  - Masquerading: Insertion of message from a fraudulent source
  - Content modification: Changing the content of a message
  - Sequence modification: Inserting, deleting, or rearranging parts of a message
  - Replay attacks: Replaying valid sessions

INTEGRITY

- Some popular Hash functions
- MD5: 128 bits (Known weaknesses)
- SHA-1: 160-bits
- SHA-2: (Bitcoin uses SHA-256)
- SHA-3: 256 & 512 bit
- Blowfish: Used in OpenBSD
- 3DES: Used for Linux passwords

INTEGRITY

- Message Authentication Codes (MAC)
- We rely on hashes to assert the integrity of messages
- An attacker can create a new message M′ and a new hash and replace H(M) with H(M′)



- So, let's create a checksum that relies on a key for validation:
  Message Authentication Code (MAC) = hash(M, key)
- Hash of message and a symmetric key: An intruder will not be able to replace the hash value. You need to have the key and the message to recreate the hash
- **MACs provide message integrity**

CS435 Distributed systems

AUTHENTICATION

Dr. Basit Qureshi

PhD FHEA SMIEEE MACM

www.drbasit.org

AUTHENTICATION

Three factors of authentication

1. Ownership Something you have	*Key, card*	*Can be stolen*
2. Knowledge Something you know	*Passwords, PINs*	*Can be guessed, shared, stolen*
3. Inherence Something you are	*Biometrics (face, fingerprints)*	*Requires hardware* *May be copied* *Not replaceable if lost or stolen*

AUTHENTICATION

• Multi-factor authentication

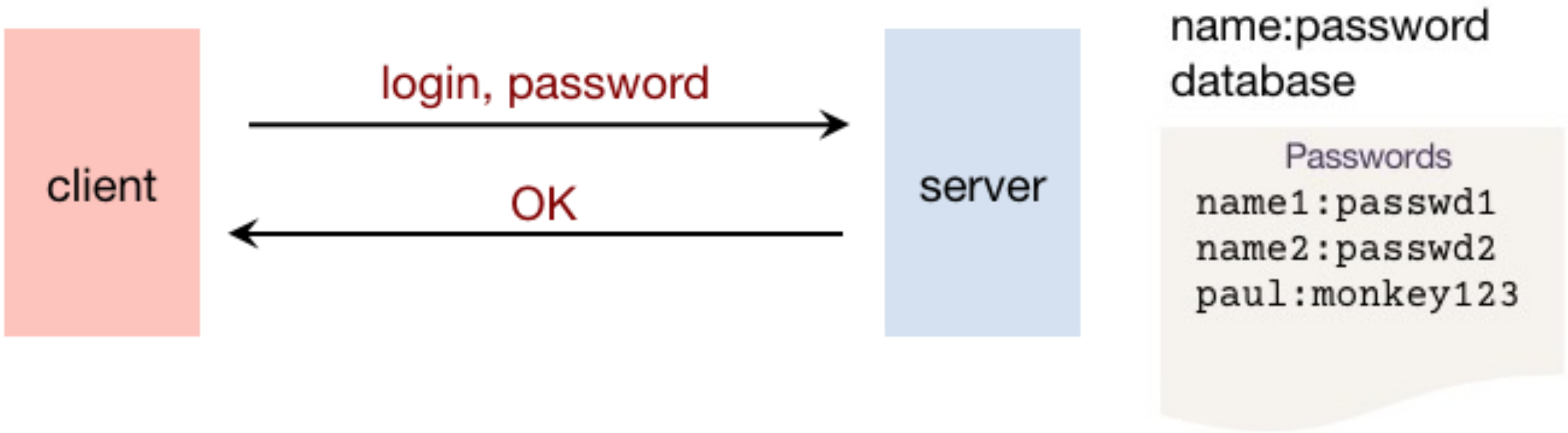
- Factors may be combined
- ATM machine: 2-factor authentication (2FA)
  - ATM card something you have
  - PIN something you know
- Password + code delivered via SMS: 2-factor authentication
  - Password something you know
  - Code something you have: your phone

- Two passwords ≠ Two-factor authentication
- The factors must be different

AUTHENTICATION

• Password Authentication Protocol (PAP)

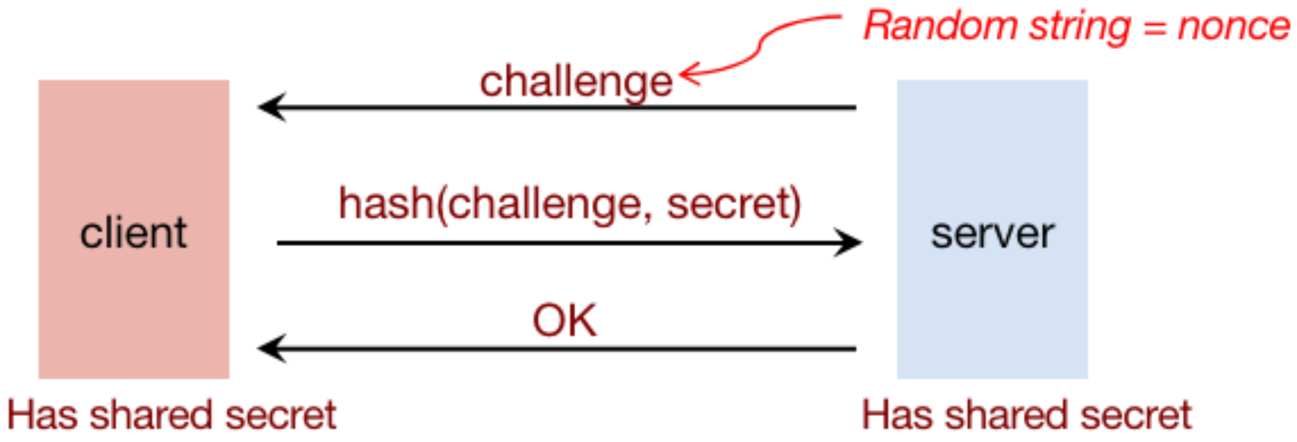
- Unencrypted, reusable passwords
- Insecure on an open network
- Furthermore, the password file must be protected from open access
  - But administrators can still see everyone's passwords



AUTHENTICATION

• Challenge-Handshake Authentication Protocol (CHAP)

- The challenge is a nonce (random bits)
- We create a hash of the nonce and the secret
- An intruder does not have the secret and cannot do this!



AUTHENTICATION

• Time-based One-time Password (TOTP) algorithm

- Both sides share a secret key
- Sometimes sent via a QR code so user can scan it into the TOTP app
- User runs TOTP function to generate a one-time password

one_time_password = hash(secret_key, time)

- User logs in with: Name, password, and one_time_password
- Service generates the same password

one_time_password = hash(secret_key, time)