Secure Software Design and Engineering
(CY-321)

# Authentication Protocols

Dr. Zubair Ahmad

# Challenge-Response

Alice $\rightarrow$ Bob : "Hi, I' m Alice."

Bob $\rightarrow$ Alice : "Hi Alice, please encrypt 0x67f810a762df5e."

Alice $\rightarrow$ Bob : $\{0x67f810a762df5e\}_k$

Or, more formally,

Alice $\rightarrow$ Bob : Alice

Bob $\rightarrow$ Alice : R where R is a random challenge.

Alice $\rightarrow$ Bob : $\{R\}_k$

# Problems with C-R

Its one-sided: Bob knows about Alice, but not vice versa.

Somehow Bob needs to maintain a database of secrets and keep it secure. In practice, thats **bloody** difficult.

Trudy could hijack the connection after the initial exchange.

If K is derived from a password (that only Alice needs to know), then Eve could mount an offline password-guessing attack.

# Variation 1

Alice $\rightarrow$ Bob : Alice

Alice identifies herself.
Bob now knows who she claims to be.

Bob $\rightarrow$ Alice : $\{R\}_k$

Bob encrypts a random value **R** using the **shared secret key K** associated with Alice.
He sends the **ciphertext** $\{R\}_k$ to Alice.

This requires **reversible encryption** (i.e., symmetric encryption like AES), because Alice will need to decrypt it.

Alice $\rightarrow$ Bob : R,

Alice decrypts $\{R\}_k$ using K to get R.
She sends R back to Bob in **plaintext** to prove she knows the key.

where R is a random challenge.

# Variation 1

Alice $\rightarrow$ Bob : Alice
Bob $\rightarrow$ Alice : $\{R\}_k$
Alice $\rightarrow$ Bob : R,

where R is a random challenge.

- Requires reversible cryptography.

- If K is derived from password, and if R is distinguishable from random bits, Eve can mount a password-guessing attack without snooping, by initiating the protocol as Alice

- Authentication is mutual if R is a recognizable quantity with a limited lifetime.

# Variation 1

Alice → Bob : Alice, $\{t\}_k$

where t is a timestamp.

Where:

- t is the **current timestamp**
- $\{t\}_k$ is the timestamp **encrypted** with a shared secret key K (symmetric encryption)
- Bob knows K (pre-shared secret) and decrypts $\{t\}_k$ to check if the timestamp is valid and recent

# Variation 1

$$\text{Alice} \rightarrow \text{Bob} : \text{Alice}, \{t\}_k$$

where t is a timestamp.

Alice claims her identity and proves knowledge of the shared key K by encrypting a fresh timestamp.

Bob decrypts the message using the shared K and verifies:

1. The decryption worked (so Alice must know K)
2. The timestamp t is **within an acceptable time window**

# Variation 1

$$\text{Alice} \longrightarrow \text{Bob} : \text{Alice}, \{t\}_k$$

where t is a timestamp.

- One-sided (Bob authenticates Alice, not vice versa).

- Requires clocks to be reasonably synchronized.

- When using the same secret K for multiple servers, Eve can impersonate Alice at the other servers (if she's fast enough).

- Replay possible if Eve can cause Bob's clock to be turned back.

- Time setting and login are now coupled.

# Mutual Authentication

$\text{Alice} \rightarrow \text{Bob} : \text{Alice}$

Alice claims her identity.

$\text{Bob} \rightarrow \text{Alice} : \text{R1}$

Bob sends a **random challenge** R1 to Alice.
Bos
**Goal**: make Alice prove she knows the shared secret key K

$\text{Alice} \rightarrow \text{Bob} : \{\text{R1}\}_k, R2$

Alice **encrypts R1** using K to prove her identity. She also generates her own challenge R2 and sends it in plaintext.
**Purpose of R2**: now **Bob has to prove he knows K** by correctly handling Alice' s challenge.

$\text{Bob} \rightarrow \text{Alice} : \{\text{R2}\}_k$

Bob **encrypts Alice challenge R2** using the shared key K.
Alice decrypts it to verify that Bob indeed knows K.

# Mutual Authentication “Optimized”

We attempt to optimize this protocol:

$$\text{Alice} \rightarrow \text{Bob} : \text{Alice}, \text{R2}$$

$$\text{Bob} \rightarrow \text{Alice} : \{\text{R1}\}_k, R1$$

$$\text{Alice} \rightarrow \text{Bob} : \{\text{R1}\}_k$$

We eliminated 25% of all messages. Not bad!

**Whats wrong with this protocol?**

# Reflection Attack

Trudy → Bob : Alice, R2

Trudy pretends to be Alice and sends a fake challenge R2R2R2 to Bob.
Bob thinks he's talking to the real Alice

Bob → Trudy : $\{R2\}_k, R1$

Bob responds as usual:

- He proves he knows K by encrypting R2
- He issues his own challenge R1R1R1 for "Alice" (really Trudy) to answer.

Trudy → Bob : Alice, R1

Now here's the trick: Trudy opens **a second session** with Bob!
This time, she replays Bob earlier challenge R1 as if *she* generated it, pretending again to be Alice.

# Reflection Attack

$\text{Bob} \rightarrow \text{Trudy}: \{R1\}_k, R3$

$\text{Trudy} \rightarrow \text{Bob}: \{R1\}_k$

Bob again responds:

- Encrypts R1 using K — this is what Trudy wanted!
- Sends a new challenge R3R3R3 (not relevant here).

**Result: Bob is fooled!**

Bob:
Sent Trudy a challenge R1 in step 2

Received {R1}K in step 5

Believes this must be **Alice**, since only someone who knows K could produce {R1}K
But it was Bob himself who generated {R1}K — **Trudy just reflected it back** to him!

# Rules

- Dont use the same key K for Alice and Bob. Instead, use K + 1, K $\oplus$ 0x0F0F0F0F, $\neg$K, or something like this

- Different challenges. Either remember past challenges and decline to encrypt known challenges, or insist that the challenges must be different for Alice and Bob (see exercises).

- Let the initiator of a protocol be the first to prove his identity.

# Authentication With Public Key

$\text{Alice} \rightarrow \text{Bob} : \text{Alice}$

Alice initiates contact and says, "Hi, I' m Alice."

$\text{Bob} \rightarrow \text{Alice} : \text{R}$

Bob responds with a **random nonce** RRR, which acts as a challenge.
**His goal:** ensure the responder is really Alice (not someone pretending to be her).

$\text{Alice} \rightarrow \text{Bob} : \{\text{R}\}_{Alice}$

Alice signs the random challenge RRR with her **key**.
She sends the **digital signature** [R]Alice back to Bob.

# Authentication With Public Key

Alice $\rightarrow$ Bob : Alice

Bob $\rightarrow$ Alice : R

Alice $\rightarrow$ Bob : $\{R\}_{Alice}$

**What happens on Bob end?**

Bob knows Alice **public key**. He:

- Uses it to verify the signature [R]Alice.
- If the signature is valid, he knows:
  - The responder is **in possession of Alice private key**
  - So this must be **Alice**.

**Authentication achieved!**

# Authentication With Public Key

$$\text{Alice} \rightarrow \text{Bob} : \text{Alice}$$

$$\text{Bob} \rightarrow \text{Alice} : \text{R}$$

$$\text{Alice} \rightarrow \text{Bob} : \{\text{R}\}_{Alice}$$

**Why this works**

Digital signatures are like handwritten signatures but cryptographically secure:

- Only Alice can create [R]Alice because only she knows her **private key**.
- But **anyone** (including Bob) can verify the signature using Alice's **public key**.

So if Bob verifies the signature on R, he knows Alice must have signed it.

# Authentication With Public Key

$$\text{Alice} \longrightarrow \text{Bob} : \text{Alice}$$

$$\text{Bob} \longrightarrow \text{Alice} : \text{R}$$

$$\text{Alice} \longrightarrow \text{Bob} : \{\text{R}\}_{Alice}$$

- Bobs database doesnt contain secrets anymore ⇒ need not be protected against theft

. • Database must still be protected against modification

# Variation and Criticism

$$\text{Alice} \rightarrow \text{Bob} : \text{Alice}$$

$$\text{Bob} \rightarrow \text{Alice} : \text{R}$$

$$\text{Alice} \rightarrow \text{Bob} : \{\text{R}\}_{Alice}$$

- Needs encryption in addition to signature.
- Both protocols have the flaw that if Eve can impersonate Bob, she can get arbitrary values signed (or encrypted).
- This is a serious flaw if the Alice key pair is used for things other than authentication (e.g., for signing bank transfers).

# Criticism

This problem can be solved if we stipulate that

- keys are never reused for different applications; or

- the system is coordinated that it’ s not possible to use one protocol to break another (for example by formatting the R values differently for different applications).

Also note what this means:

**By combining two protocols that are secure in themselves, you get a system that is not secure at all; and you can design protocols whose deployment threatens the security of a system that is already in place!**

For people who like to sound clever, we can also say that security isnt closed under composition.

# Mediated Authentication

Mediated authentication happened when Alice first asks a trusted intermediary, Trent, to introduce her to Bob.

Because Trent is trusted by both Alice and Bob, authentication is mutual.

Does not need public key!

Alice → Trent : Alice wants Bob

Trent : Invents $\{K\}_{AB}$

Trent → Alice : $\{\text{Use } \{K\}_{AB} \text{ for Bob}\}_{Alice}$

Trent → Bob : $\{\text{Use } \{K\}_{AB} \text{ for Alice}\}_{Bob}$

# Needham-Schroeder

- Its a classic mediated authentication protocol with mutual authentication.

- Its been a model for many other protocols.

- Its used in Kerberos and Kerberos is used in Active Directory ⇒ huge installed base.

- We ll analyze this protocol in some detail in order to understand its strengths and weaknesses.

# Needham-Schroeder

Alice $\longrightarrow$ Trent : $N_1$, Alice wants Bob

Trent : Invents $K_{AB}$

Trent $\longrightarrow$ Alice : $\{N_1$, Bob, $K_{AB}$ , $\{K_{AB}$ , Alice$\}_{\text{Bob}}$ $\}_{\text{Alice}}$

Alice : Verifies $N_1$, extracts $K_{AB}$ and ticket

Alice $\longrightarrow$ Bob : $\{K_{AB}$ , Alice$\}_{\text{Bob}}$, $\{N_2\}AB$

Bob : Extracts $K_{AB}$ from ticket

Bob $\longrightarrow$ Alice : $\{N_2 - 1$, $N_3\}$AB

Alice $\longrightarrow$ Bob : $\{N_3 - 1\}$AB

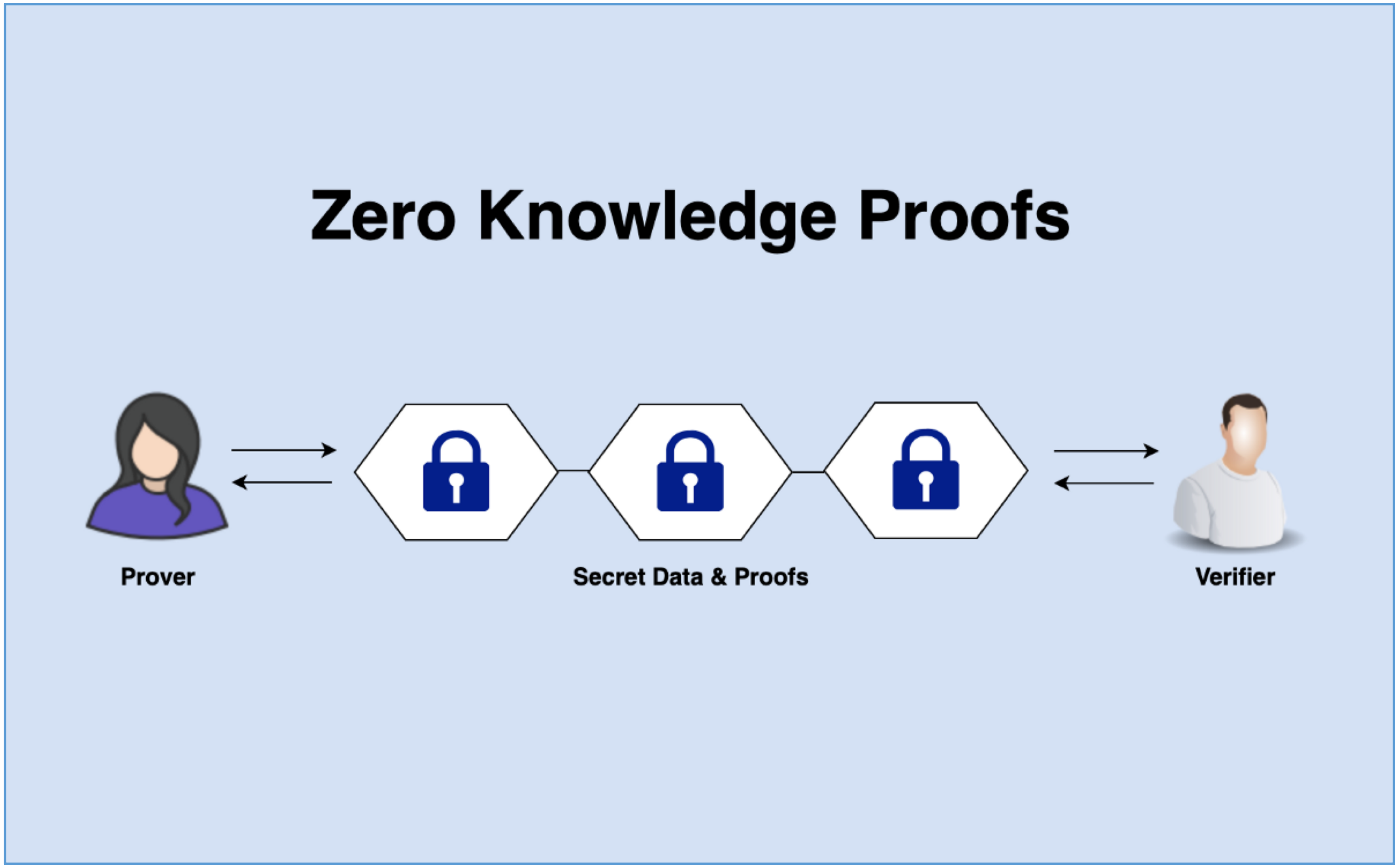
where $\{K_{AB}$ , Alice$\}_{\text{Bob}}$ is Trent ticket for Alice conversation with Bob and the Ni are nonces, i.e., quantities used only once.

# Zero-Knowledge Proofs (ZKPs)

Think of a magic door that opens only with the correct password. You want to prove you know the password without telling anyone the password.

Zero-Knowledge Proofs (ZKPs) are cryptographic protocols that allow one party (the **prover**) to prove to another party (the **verifier**) that they know a value (or that a statement is true), **without revealing any information** about the value itself.

# Zero-Knowledge Proofs (ZKPs)



# Zero-Knowledge Proofs (ZKPs)

```
from random import randint
from sympy import isprime, mod_inverse

# Setup
p = 23                  # a small prime number
g = 5                   # a generator of the group
x = 6                   # secret known only to prover
h = pow(g, x, p)        # public key
```

# Zero-Knowledge Proofs (ZKPs)

```
# === Prover Step 1: Commit ===
r = randint(1, p-2)          # random nonce
t = pow(g, r, p)             # commitment
print(f"Prover sends t={t} to verifier")
```

# Zero-Knowledge Proofs (ZKPs)

```
# === Verifier Step 2: Challenge ===
c = randint(1, p-2)         # random challenge
print(f"Verifier sends challenge c={c}")
```

# Zero-Knowledge Proofs (ZKPs)

```
# === Prover Step 3: Response ===
s = (r + c * x) % (p - 1)
print(f"Prover sends response s={s}")
```

# Zero-Knowledge Proofs (ZKPs)

```
# === Verifier Step 4: Verify ===
left = pow(g, s, p)
right = (t * pow(h, c, p)) % p
print(f"Verifier checks: g^s mod p =? t * h^c mod p")
print(f"{left} =? {right}")

if left == right:
    print("Verifier accepts the proof.")
else:
    print(" Verifier rejects the proof.")
```

## Questions??

zubair.ahmad@giki.edu.pk

Office: G14 FCSE lobby