Table of contents

Authentication

Authentication of Data Origin and Entities

Authentication Methods

Authentication of origin ensures that a message genuinely comes from the claimed sender.

Symmetric Methods:

- **MAC alone**: `A → B: X, MACk(X)` – B verifies using the shared key k
- **MDC + encryption**: `A → B: X, Ek(MDC(X))` or `A → B: Ek(X, MDC(X))`

Asymmetric Method:

- **MDC + signature**: `A → B: X, Sigpriv-A(MDC(X))` – Also provides non-repudiation

Limitations: These simple protocols do not protect against replay attacks or ensure message freshness. Time- or context-aware mechanisms are required.

Original Text

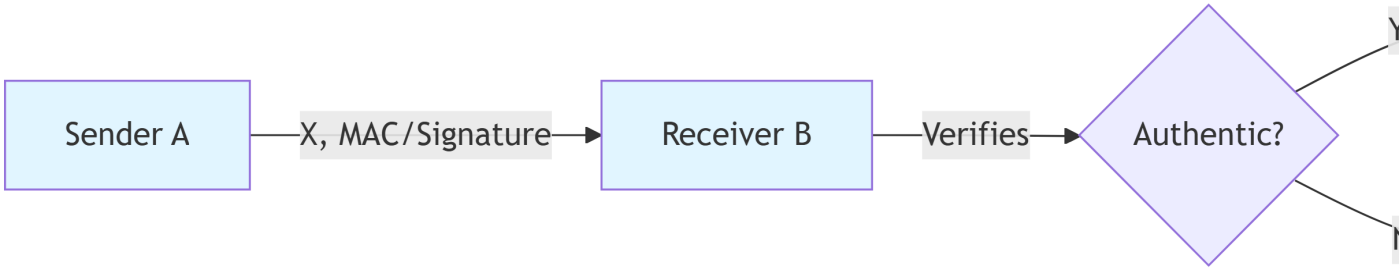
Authentication of Data Origin
1) MAC with a symmetric key *k* known to A and B: `A → B: X, MACk(X)` If B computes `MACk(X)` independently and obtains the same value the message originates from A.
2) MDC + symmetric encryption (key *k* known to A and B): `A → B: X, Ek(MDC(X))` B computes `MDC(X)` then `Ek(MDC(X))`. If equal message comes from A.
3) As 2) with confidentiality of X: `A → B: Ek(X, MDC(X))`
4) MDC + digital signature: `A → B: X, Sigpriv-A(MDC(X))` B computes `MDC(X)` and verifies `Sigpriv-A(MDC(X))` using an authentic copy of *pub-A*. If equal A is the originator. This solution also provides **non-repudiation of origin**.
These simple protocols offer no support for **uniqueness** or **freshness (timeliness)** of received messages and are vulnerable to **replay attacks**. They require mechanisms accounting for time or transaction context.

Quick Review

4 Methods

- MAC alone
- MDC + encryption
- MDC + confidential encryption
- MDC + signature

Warning: Vulnerable to replay attacks without temporal mechanisms



Entity Authentication – Introduction

Objectives of a Robust Protocol

Entity authentication (or identification) aims to prove an entity's identity in real time.

Required Properties:

1. If A and B are honest and A authenticates, B must accept A's identity.
2. B cannot reuse A's information to impersonate A to C.
3. Negligible probability that entity C successfully impersonates A.
4. Property 3 holds even if C has observed or participated in prior protocol instances.

Basic Elements:

- **Something known**: passwords, PINs, keys
- **Something possessed**: smart cards, password generators
- **Something inherent**: biometrics (fingerprints, retina, DNA)

Classification:

- **Weak authentication**: Reveals the secret (userid/password)
- **Strong authentication**: Proves possession of the secret without revealing it
- **Zero-knowledge**: Strong authentication without revealing any information about the secret

Original Text

Entity Authentication (Entity Identification)
Objectives of a Robust Identification Protocol:

1. If A and B are "honest": if A can authenticate to B, B must accept A's identity.
2. B cannot reuse information provided by A to identify as A to C.
3. The probability that a third party C successfully impersonates A to B is negligible.
4. Point 3) remains true even if:
    - C has observed a large (polynomial) number of identification protocol instances between A and B
    - C has participated (possibly impersonating someone else) in prior protocol executions with A or B
    - Multiple protocol instances (possibly initiated by C) may run simultaneously without compromising the identification process

Terminology: The user (A) is called the **claimant** (the one claiming to be A), and the system (B) is the **verifier**.
Basic Authentication Elements:

- **Something known**: passwords, PINs, private or secret keys, etc.
- **Something possessed**: passport, smart card, password generators, etc.
- **Something inherent to the individual**: biometric properties like fingerprints, retina, DNA, etc.

Weak Authentication: The user presents a pair (userid, password) to the system. The userid is the claimed identity, and the password is the corroborating evidence.
Strong Authentication: The secret used to corroborate identity is not explicitly revealed. The user provides proof of possession of the secret.
Zero-Knowledge Authentication: Strong authentication protocols that additionally prove identity without revealing any information (not even a hint) about the secret itself. This involves proving an assertion without disclosing any details.
Weak authentication protocols satisfy points 1) and 3). **Strong authentication protocols** satisfy (at least partially) points 2) and 4) as well.

Quick Review

3 Levels: Weak (reveals secret) < Strong (proof of possession) < Zero-knowledge (no info revealed)
4 Objectives

- Acceptance if honest
- Non-reusability
- Impersonation resistance
- Observation resistance

Attacks and Countermeasures

Dictionary Attacks

Principle and Countermeasures

A dictionary attack systematically tests probable passwords against a cryptographic system.

Attack Methods:

- **Offline**: The attacker obtains the hashed password database or captures exchanges
- **Online**: Direct attempts against the system (typically rate-limited)

Vulnerability Example:

- `A → B: A`
- `A ← B: R` (random challenge)
- `A → B: Ep(R)`

The pair `(R, Ep(R))` enables an offline dictionary attack.

Countermeasures:

- Limit online attempts
- Salting (adding a random element)
- Use slow key derivation functions
- Strong authentication avoiding password transmission

Original Text

Dictionary Attacks
A dictionary attack uses a database containing dictionary words from one or more languages (including variants) as input to an encryption or hashing system to obtain secret keys or passwords.
This attack is highly effective for obtaining poor-quality passwords, even today, with large databases containing word variations and complex mnemonic rules to "crack" higher-entropy passwords.
A dictionary attack can be mounted:

- By obtaining the system's password database (encrypted or hashed)
- From one or more authentication exchange instances following a passive attack (network packet observation). For example:
  - `A → B: A` (A sends its identity)
  - `A ← B: R` (R = random number, challenge)
  - `A → B: Ep(R)` (A encrypts R with its password)

The pair `(R, Ep(R))` enables an **offline** dictionary attack.
Dictionary attacks are typically less effective **online** because operating systems limit the number of failed authentication attempts.

Quick Review

Offline (via DB or capture) > **Online** (system-limited)
Protection: Salting, attempt limiting, strong authentication

Plaintext Equivalence

Concept and Risks

A string is **plaintext-equivalent** to a password if it grants the same access as the password itself.

Vulnerability Example:

If the system stores `H(p)` and the protocol is: `A → B: H(p)`

Then `H(p)` is plaintext-equivalent to `p` because the attacker can use it directly.

Counterexample (Classic UNIX):

The system stores `H(p)` but the protocol transmits `p`. The stored hash is thus not plaintext-equivalent.

Security Principle: Server-stored information must be neither plaintext-equivalent to passwords nor exposed to offline dictionary attacks.

Original Text

Plaintext Equivalence
A data string is **plaintext-equivalent** to a password if it can be used to obtain the same access level as the password.
Example: If system B stores a list of all hashed passwords in the following authentication process: `A → B: H(p)` (A sends B the hash of the password)
The string `H(p)` is **plaintext-equivalent** to the password `p`.
This is equivalent to saying that applying a hash function for password storage provides no additional security for the system.
Counterexample: In the classic UNIX authentication system, the password hash stored in `/etc/passwd` is **not** plaintext-equivalent to the password because it is `p` (not `H(p)`) that is exchanged between client and server.
This property is essential because password databases are typically protected by logical mechanisms that are often compromised by server OS vulnerabilities.
If these central databases contain passwords in cleartext or plaintext-equivalent information, the consequences of an attack are devastating.
The ideal case is that server-stored information is neither plaintext-equivalent to passwords nor exposed to offline dictionary attacks.

Quick Review

Plaintext-equivalent: Data usable like the original password
Danger: If the system transmits `H(p)` and stores `H(p)` → `H(p)` is plaintext-equivalent

Good Design: System transmits `p`, stores `H(p)` → not plaintext-equivalent

---

Weak Authentication

Fixed Passwords

Storage and Protection

Fixed-password systems exhibit significant vulnerabilities.

Storage Techniques:

- **Cleartext**: Protected by OS access control (vulnerable to OS flaws, backups)
- **Encrypted or hashed**: Vulnerable to offline attacks (guessing, dictionary, collisions)

Major Problem: The password can be replayed after observation on an unprotected network.

Protection Techniques:

- Strict creation rules (minimum entropy)
- Rate-limiting and attempt restrictions
- **Salting**: Adding a random element before hashing
- Restrict password file dissemination

Typical Password Entropy: Low (~40 bits for an 8-character random password, much less for common words).

Original Text

Weak Authentication – Fixed Password
Weak authentication systems are divided into two main categories:

- **Fixed password**: The password does not depend on time or the number of protocol executions. This includes systems where the password is changed by user decision or system security measures.
- **Variable password**: Password modification based on time and/or execution count is part of the identification protocol.

Storage Techniques for Fixed-Password Systems:

- **Cleartext password storage** in a file protected by the OS's access control mechanisms.

  - Problems: OS vulnerabilities, "super-user" privileges, backups, etc.

- **Encrypted or hashed password storage** (possibly making the file publicly accessible, cf. UNIX example).
  - Problems: offline attacks, i.e., guessing attacks, brute-force dictionary attacks, collision identification, etc.

Most serious fixed-password problem: It can be replayed after eavesdropping on an unprotected network.
Fixed-Password System Protection Techniques:

- Strict rules for password creation, maintenance, and updates, considering the low entropy of user-chosen passwords
- Slowing the identification process and limiting failed attempts to counter "online brute-force attacks"
- **Salting** (cf. UNIX example)
- Restrict or avoid dissemination of password files, even when encrypted

Quick Review

2 Types: Fixed password (static) vs. Variable password (changes per instance)
Storage: Cleartext (highly vulnerable) vs. Encrypted/Hashed (offline attacks)
Protections: Strict rules, attempt limiting, salting, non-dissemination

---

Variable Passwords

One-Time Passwords and Generators

Variable passwords change with each authentication, reducing replay risk.

Lamport Scheme (S/Key):

```
Initialization:
  A generates secret w, chooses t
  A → B: wt = Ht(w)
  B stores: wstored := wt, n := t-1

Identification (t-n)th:
  A → B: A, n, wn = Hn(w)
```

```
B tests: H(wn) == wstored
If OK: n := n-1, wstored := wn
```

Attacks if B is not authenticated:

- **Pre-play attack**: C obtains `wn` before A and replays it
- **Small `n` attack**: C requests an `n` < current n

Hardware Generators (SecureID):

- Card generates a code every 30–60 seconds
- Based on a secret key shared with the system
- Vulnerable to pre-play but with limited time window

Original Text

Weak Authentication: Variable Password
The two best-known variable-password identification techniques are **one-time passwords** and **hardware random number generators**.
One-Time Passwords – Lamport Scheme (S/Key):
Initialization:

- A generates a secret w
- A constant t (~1000, number of identifications) and a OWF H are chosen
- `A → B: wt = Ht(w)` (H applied t times to w)
- B stores: `wstored := wt`, `n := t-1`

Messages for the (t-n)th identification:

- `A → B: A` (A's identity)
- `A → B: n` (current iteration for A)
- `A → B: wn = Hn(w)`
- B tests: `H(wn) == wstored`. If OK `n := n - 1` and `wstored := wn`

End: When n == 0, A chooses a new w and restarts…
Attacks: B must be authenticated! Otherwise, C impersonates B and:

- obtains the current password wn and can replay it (**pre-play attack**)
- provides an n < current n and can thus generate all Hm>n(wn) (**small n attack**)

Hardware Random Number Generators:

- These are smart cards that periodically (~every 30 or 60 seconds) generate different numbers used to identify (along with a PIN and user identity information) the cardholder.

- Generation is based on a secret key present on the card and known to the system.
- The best-known is **SecureID** by RSA Security.
- It has been adopted by many banks for Internet tele-banking authentication.
- It is also vulnerable to pre-play attacks, but the replay window is limited to the change frequency (30 or 60 seconds).

Conclusions on Weak Authentication:

- Fixed passwords offer very low security.
- Variable passwords are a significant step toward strong authentication but require additional precautions.

Quick Review

Lamport: wn+1 = H(wn), authentication via hash chain verification
Hardware: Synchronized generator (30–60s), limited pre-play window
Warning: Requires B's authentication to prevent pre-play and small-n attacks

---

Strong Authentication

Symmetric

Basic Protocols

Challenge-Response

Strong authentication uses cryptography to prove secret possession without revealing it.

Basic Unilateral Authentication:

```
A → B: A
A ← B: R (random challenge)
A → B: Ek-AB(R)
B verifies by decrypting
```

Session key: `K := R`

Improvements:

- Add B's identity: `Es(B, ra)` for key confirmation
- Add timestamp: `Es(B, ta, ra)` for freshness (requires synchronized clocks)
- Use MAC instead of encryption: `Hk-AB(R)` (faster)

Vulnerabilities:

- Man-in-the-Middle if no mutual authentication
- Chosen-plaintext attacks possible
- Replay if challenges are poorly managed

Original Text

Strong Authentication: Symmetric Solutions
Strong authentication protocols use symmetric or asymmetric cryptographic techniques.
Unilateral Authentication with Shared Symmetric Key:

```
A → B: A (A sends its identity)
A ← B: R (R = random number, challenge)
A → B: Ek-AB(R) (A encrypts R with the shared key)
```

B decrypts `Ek-AB(R)` and identifies A if it finds R.
Remarks:

- B must ensure the challenge R is random and not repeated.
- This protocol is a significant improvement over password authentication because varying challenges prevent Eve from replaying protocol parts.
- Eve can attempt an **offline known-plaintext attack** from a (typically small) number of pairs (R, Ek-AB(R)), but most encryption systems are secure in this regard (DES is vulnerable only after 2^47 pairs).
- C can impersonate B and choose challenges R to mount a **chosen-plaintext attack** (DES vulnerability is also 2^47, but other systems are more sensitive to such attacks).
- C could mount an **Active Man-in-the-Middle** attack by impersonating B since B is not authenticated, but must convince A to start the protocol.
- A **MDC**: `H(k-AB,R)` or a **MAC**: `Hk-AB(R)` can replace `Ek-AB(R)` to speed up identification.
- After initial identification, a secure (at least authenticated) channel must be established using cryptographic protection to prevent C from injecting packets while impersonating A.

Protocols of this type, where one entity must respond based on a challenge from the other, are called **challenge-and-response protocols** and are the most common form of strong authentication.
Unilateral Authentication with Shared Symmetric Key, 2nd Variant:

```
A → B: A, Ek-AB(timestamp)
```

Requires synchronized clocks between A and B.
Advantage: One fewer message and stateless protocol
But:

- Clock synchronization is difficult to achieve in practice, and "drifts" can be exploited by an adversary.
- Moreover, if B's clock is "advanced," some past identification instances may become valid again.

Quick Review

Challenge-Response: B sends challenge R, A responds with E_k(R)
Alternative: MAC instead of encryption (faster)
With Timestamp: One fewer message but requires clock synchronization

---

Mutual Authentication

Robust Protocols and Reflection Attacks

Bilateral authentication requires precautions against reflection attacks.

Vulnerable (Naive) Protocol:

```
A → B: A, R2
A ← B: R1, Ek-AB(R2)
A → B: Ek-AB(R1)
```

Reflection Attack: C can start two instances and use B's response to its own request to complete authentication.

Robust Protocol:

```
(1) A → B: A, R2
(2) A ← B: Ek-AB(R1, R2, A)
(3) A → B: Ek-AB(R2, R1)
```

Protections:

- Inclusion of A's identity in (2) to prevent reflection attacks
- Asymmetry in challenge order (R1,R2) vs. (R2,R1)
- Inclusion of challenges in the encrypted message

Original Text

Strong Authentication: Symmetric Solutions (Mutual Authentication)
Bilateral Authentication with Shared Symmetric Key (Intuitive Solution):

```
A → B: A, R2
A ← B: R1, Ek-AB(R2)
A → B: Ek-AB(R1)
```

At first glance, the protocol seems robust, but observe what an adversary C can do by starting two identification processes:

```
C → B: A, R2 (C pretends to be A)
C ← B: R1, Ek-AB(R2) (B responds)
```

At this point, C starts a second instance:

```
C → B: A, R1
C ← B: R3, Ek-AB(R1) (C cannot proceed further but...)
```

Successfully completes the first identification instance with:

```
C → B: Ek-AB(R1) (and it's done!)
```

Because C returns to B the same R it received from B, such attacks are called **reflection attacks**.
Since the key is shared, C could have achieved the same result (even more discreetly) by executing the second instance with A (pretending to be B).
Bilateral Authentication with Shared Symmetric Key (Robust Solution):

```
(1) A → B: A, R2
(2) A ← B: Ek-AB(R1, R2, A)
(3) A → B: Ek-AB(R2, R1)
```

The presence of **A** in (2) adds extra security in case obvious reflection attacks are not detected by the protocol. Otherwise, if A initiates authentication with what it believes to be B but is actually C:

```
A → C: A, R2 (*)
```

Then C starts a new authentication instance with A using the same R2:

```
C → A: B, R2
```

If A does not see R2 as an obvious reflection, it responds:

```
C ← A: Ek-AB(R1, R2) (As in (2) but without the 'A')
```

Which C uses to complete its protocol (*). However, if A responds with B inside the packet as recommended in the protocol:

```
A → C: Ek-AB(R1, R2, B)
```

This can no longer be used by C to continue (*) because it would require A instead of B. Note also that including R1 in the encrypted part protects against **chosen plaintext attack** risks from the previous solution.

Quick Review

Reflection Attack: Use one session's response to authenticate another
Protection: Include identities + asymmetry in challenges (R1,R2) vs. (R2,R1)

Asymmetric

Public-Key Protocols

Asymmetry avoids secret sharing but requires precautions against chosen-ciphertext attacks.

Vulnerable Protocol:

```
A → B: A
A ← B: Epub-A(R)
A → B: R
```

Problem: B can make A decrypt anything.

Robust Protocol:

```
A → B: A
A ← B: H(R), B, Epub-A(B, R)
A → B: R (after verifying H(R) and B)
```

Protection: Structure the encrypted text and prove plaintext knowledge via `H(R)`.

Mutual Authentication (Needham-Schroeder):

```
(1) A → B: Epub-B(r1, A)
(2) A ← B: Epub-A(r1, r2)
(3) A → B: Epub-B(r2)
```

The presence of A in (1) prevents chosen-ciphertext attacks.

Original Text

Strong Authentication: Asymmetric Solutions
Unilateral Authentication with Asymmetric Key (Intuitive Solution...):

```
A → B: A
A ← B: Epub-A(R) (B encrypts with A's public key)
A → B: R (A returns R after decryption)
```

Remarks:

- B must know A's authentic key to avoid man-in-the-middle attacks.
- But especially: B can mount **chosen-ciphertext attacks** (i.e., B can make A decrypt anything!).

Unilateral Authentication with Asymmetric Key (Robust Solution):
Idea: Structure the text encrypted with pub-A and show that B knows the plaintext:

```
A → B: A
A ← B: H(R), B, Epub-A(B, R) (H(R) proves B knows R)
```

A decrypts `Epub-A(B, R)` and obtains B' and R'. A aborts the protocol if `h(R')  h(R)` or `B'  B`, otherwise:

```
A → B: R
```

B identifies A if it matches the initial R.
A dual protocol can be imagined using A's signature with priv-A (instead of encryption with pub-A), but the same structural precautions apply to prevent A from signing a "malicious" message generated by B.
Bilateral Authentication with Asymmetric Key. Robust Solution by Needham and Schroeder:

```
(1) A → B: Epub-B(r1, A)
(2) A ← B: Epub-A(r1, r2)
(3) A → B: Epub-B(r2)
```

Note that the presence of A in (1) thwarts chosen-ciphertext attacks.
The protocol can be strengthened by adding a "witness" `H(r1)` in (1).
Final Remarks on Classical Authentication:

- Entity authentication is a highly complex process full of unexpected pitfalls.
- Some protocols, like the one proposed by ISO in 1988 for authentication in distributed directories, have flaws very similar to those highlighted here.
- When identification occurs within a session, it is imperative that all session packets be authenticated (e.g., by establishing a secure channel with session key establishment).

Quick Review

Vulnerability: Chosen-ciphertext attacks if no structure
Protection: Include H(R), B's identity in the encrypted message; A verifies before revealing R
Needham-Schroeder: 3 messages with identity inclusion to prevent chosen-ciphertext

---

Zero-Knowledge Proofs

Concepts

Definitions and Principles

Zero-knowledge proofs allow proving possession of a secret without revealing any information about it.

Required Properties:

- **Completeness**: If A and B are honest, B accepts A's proof
- **Soundness**: If C succeeds in deceiving B, then C holds A's secret (or equivalent)
- **Zero-knowledge**: B learns nothing about A's secret

Generic Structure:

```
(1) A → B: witness
(2) A ← B: challenge
(3) A → B: response
```

ZKIP Types:

- **Computational ZKIP**: A polynomial-time observer cannot distinguish a real proof from a simulation
- **Perfect ZKIP**: No probabilistic difference between real proof and simulation (guaranteed by information theory)

Principle:

- A commits to a class of questions (1)
- B chooses a question from this class (2)
- A answers using its secret (3)
- Repeat to reduce guessing probability.

Original Text

Zero-Knowledge Proofs: Definitions
Problem with "classical" authentication methods: B (or even an observer) can obtain information about A's secret:

- In weak authentication methods (password-based), the secret is fully revealed.
- In classical challenge-and-response methods, B can obtain [plaintext/ciphertext] pairs useful for cryptanalysis.

Definition: An interactive protocol is a **proof of knowledge** if it has the following two characteristics:

- **Completeness**: If A and B are "honest," B accepts the proof provided by A.
- **Soundness**: If a "dishonest" entity C can "deceive" B, then C holds A's secret (or polynomially equivalent information). This is equivalent to requiring secret possession for proof success.

A proof of knowledge is called a **zero-knowledge interactive proof (ZKIP)** if it additionally has the property that A can convince B of a fact without revealing any information about its secret.
A protocol is a **computational ZKIP** if an observer capable of probabilistic polynomial-time tests cannot distinguish a genuine proof (where A responds) from a simulated proof (e.g., by a random generator).
A protocol is a **perfect ZKIP** if there is no probabilistic difference between the real proof and the simulated proof. The absence of information in the proof is guaranteed by Shannon's information theory, not computational criteria.
Generic ZKIP Structure:

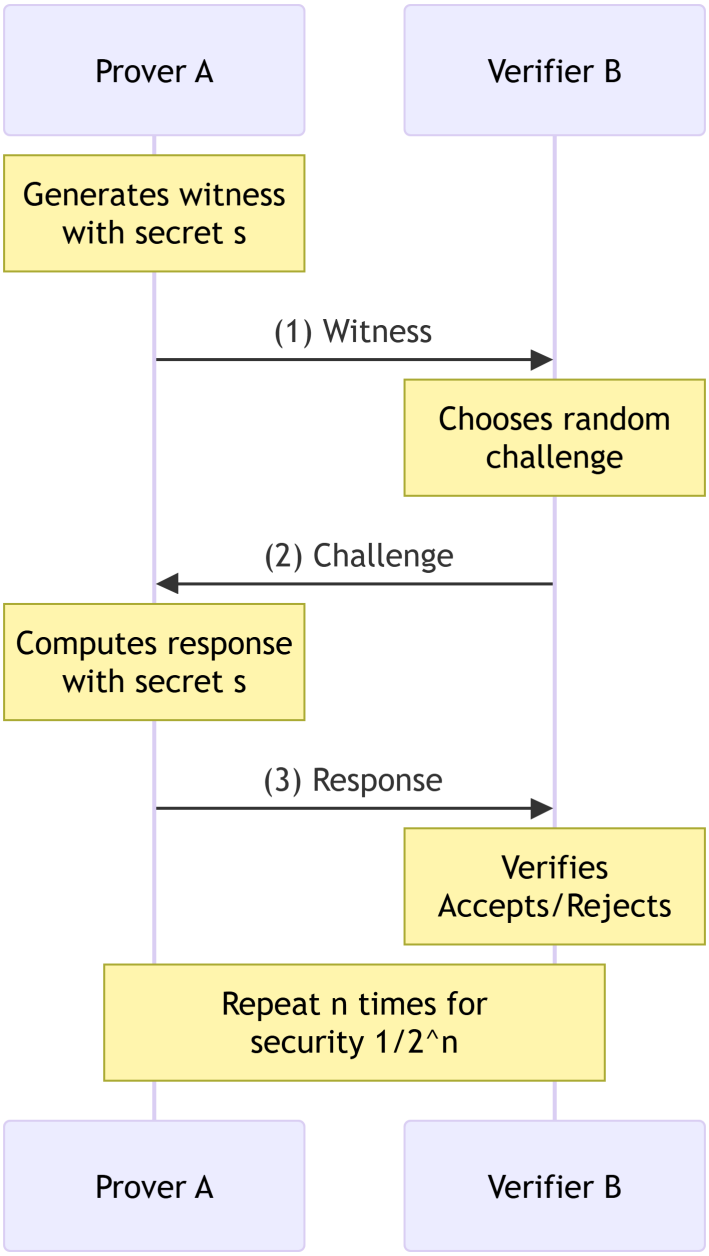
```
(1) A → B: witness
(2) A ← B: challenge
(3) A → B: response
```

- **(1)** A chooses a random secret number and sends B proof of possession of this secret. This constitutes a commitment from A and defines a class of questions to which A claims to know the answers.
- **(2)** The challenge sent by B randomly selects a question from this class.
- **(3)** A responds (using its secret).

If necessary, the protocol is repeated to minimize the probability of an "impostor" guessing the correct answers by chance.

Quick Review

3 Properties: Completeness (accepts if honest), Soundness (requires secret), Zero-knowledge (no info revealed)
Structure: Witness → Challenge → Response (repeat n times)
Perfect ZK: Indistinguishable from simulation even with infinite resources



ZKIP – Intuitive Example (Ali Baba's Cave)

Concept Illustration

This example intuitively illustrates the zero-knowledge principle.

Scenario:

- A knows the secret passage between y and z in a cave
- B wants to verify this knowledge without learning how to traverse

Protocol:

1. B stands at entrance E
2. A chooses to go to y or z (witness)
3. B enters and stops at point x
4. B asks A to return from the right or left (challenge)
5. A uses the secret to comply (if needed)

Repetition: n times. If A does not know the secret: success probability = 2^{-n}

Properties:

- B confirms A can traverse but learns nothing about how
- B cannot convince a third party B' (A and B could have agreed on sequences)
- Inspired by the "cut-and-choose" technique

Original Text

ZKIP: Intuitive Example
This example is described in [Qui89] (Quisquater et al., "How to Explain Zero-Knowledge Protocols to Your Children", Crypto'89). Suppose A knows a passage between y and z (the secret).

(1) B stands at the cave entrance at point E.
(2) A chooses a direction and goes to points y or z (witness choice).
(3) Once A is inside the cave, B enters and stops at point x.
(4) B asks A to return to point x from the right or left (the challenge).
(5) Using the secret to move from y to z (or vice versa) if necessary, A complies with B's instructions.
Repeat steps 1 to 5 n times. If A does not know the secret, it has a 2^{-n} probability of successfully deceiving B (guessing "correctly").
In this example, B sees that A can traverse the yz passage at will but obtains no information on how to do so, even if the protocol is executed millions of times.

Moreover, B cannot convince B' that A knows the secret (as would be the case if A encrypted information using a private key, for example). B' might suspect A and B of agreeing on the sequences (right/left).
Such protocols are inspired by the **"cut-and-choose"** technique, where A and B fairly share a pie as follows: - A cuts the pie. - B chooses a piece. - A takes the remaining piece. The first ZKIP was published in 1985 by S. Goldwasser [Gol85]. The application of the cut-and-choose paradigm to cryptographic protocols is due to Rabin [Rab78].

Quick Review

Cave: A enters randomly (y or z), B asks for exit (left/right)
Cheating Probability: 2^{-n} after n repetitions
ZK: B verifies knowledge but learns no secret, cannot convince third party

---

ZKIP – Graph Isomorphism

ZKIP – Graph Isomorphism

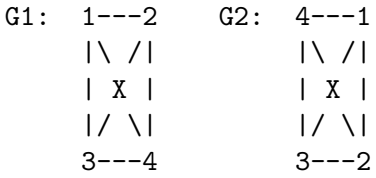
Formal Protocol
Zero-knowledge proofs can be constructed on hard mathematical problems.
Context: Two graphs G_1 and G_2 are isomorphic if there exists a permutation π such that for every edge $\{u, v\} \in E_1$, $\{\pi(u), \pi(v)\} \in E_2$.
Property: Finding the permutation π between two ~1000-node graphs is computationally hard (no known polynomial algorithm).
Protocol:

```
Init: A chooses G1 and creates G2 =  (G1) with   secret
(1) A → B: H (A creates H =  (G2) randomly)
(2) A ← B: i   {1,2}
(3) A → B:   such that H =  (Gi)
    If i=2:   :=
    If i=1:   :=
(4) B verifies H =  (Gi)
(5) Repeat n times
```

Perfect Zero-Knowledge Verification: Protocol transcripts are indistinguishable (probabilistic distribution) from those produced by a simulator.

Original Text

ZKIP: Graph Isomorphism
Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a permutation π such that $\{u, v\} \in E_1$ if and only if $\{\pi(u), \pi(v)\} \in E_2$.
Example: $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $V = \{1, 2, 3, 4\}$, $E_1 = \{12, 13, 23, 24\}$, and $E_2 = \{12, 13, 14, 34\}$ are isomorphic with the permutation $G_1 \to G_2 : \{4, 1, 3, 2\}$:



From a graph G_1, one can easily (in polynomial time) find a permutation π such that $G_2 = \pi(G_1)$.
However, no polynomial-time algorithm is known to determine if two sufficiently large graphs (~1000 nodes) are isomorphic (i.e., find the permutation π from G_1 and G_2).
ZKIP Based on Graph Isomorphism:
(Initialization) A chooses a sufficiently large graph G_1 and invents a permutation π (the secret) to compute a second graph $G_2 = \pi(G_1)$. G_1 and G_2 are made public.

```
(1) A → B: H
```

A chooses a random permutation ϕ such that $H = \phi(G_2)$ and sends H to B (the witness).

```
(2) A ← B: i
```

B chooses an integer $i \in \{1, 2\}$ and sends it to A (the challenge).

```
(3) A → B:
```

A computes ψ such that $H = \psi(G_i)$: - If $i = 2$: $\psi := \phi$ - If $i = 1$: $\psi := \phi \circ \pi$
(4) B checks if $H = \psi(G_i)$ and accepts the step as correct.
(5) Repeat (1) to (4) enough times to minimize "guessing" risks.
Property Verification:

- **Completeness**: The protocol is accepted if A knows the secret (i.e., the permutation π between the two graphs).
- **Soundness**: If C tries to impersonate A without knowing π, it can fix a j and provide a correct permutation $\psi(G_j)$ but cannot find a correct permutation for both graphs. It must guess the challenge provided by B.

- **Zero-Knowledge**: A succeeds in convincing B that the two graphs are isomorphic but learns nothing about π. B only sees a random graph H isomorphic to G_1 and G_2 and a permutation between H and G_1 or between H and G_2.
- **Perfect Zero-Knowledge**: This means B could generate such information alone (using a random generator and polynomial computations). It can be proven that the transcripts provided by the protocol are indistinguishable (from a probabilistic distribution perspective) from those produced by a simulator.

Quick Review

Problem: Finding permutation between two isomorphic graphs = hard
Protocol: A creates random H, B asks for permutation to G1 or G2, A responds
Perfect ZK: Transcripts indistinguishable from a simulator

ZKIP – Fiat-Shamir Algorithm

Practical Protocol

Fiat-Shamir is an efficient and practical ZKIP based on the square root modulo composite problem.

Initialization:

- Trusted third party T chooses $n = pq$ (keeps p,q secret)
- A chooses secret s with $\gcd(s, n) = 1$
- A computes $v = s^2 \bmod n$ and distributes v (certified public key)

Protocol:

```
(1) A → B: x = r² mod n
    (A chooses random r, witness)

(2) A ← B: e  {0,1}
    (B sends challenge)

(3) A → B: y = r·s  mod n
    (A computes response with secret s)

B rejects if y = 0
B accepts if y²  x·v  (mod n)
```

Repetition: Multiple times for security 2^{-nk}

Properties:

- **Soundness**: An impostor can easily answer e=0, but for e=1, it must compute $\sqrt{x} \bmod n$ (hard by SQROOTP)
- **Perfect Zero-Knowledge**: The pairs (x,y) can be simulated by B by choosing y randomly and computing $x = y^2$ or y^2/v
- B cannot impersonate A because it cannot predict challenges

Original Text

ZKIP: Fiat-Shamir Algorithm
Goal: Allow A to identify itself by proving knowledge of a secret s (associated with A via authentic public information) to B without revealing any information about s.
This is a protocol serving as the basis for real and efficient implementations.
Algorithm:
(Initialization):

(a) A trusted third party T chooses and publishes an n such that $n = pq$ and keeps p and q secret.

(b) A chooses a secret s with $1 \leq s \leq n-1$ and $\gcd(s, n) = 1$, computes $v = s^2 \bmod n$, and distributes v as a public key certified by T.

(1) A → B: $x = r^2 \bmod n$

A chooses a random r and sends a witness r^2.

(2) A ← B: e ∈ {0,1}

B sends its challenge.

(3) A → B: $y = r \cdot s \bmod n$

A computes the response using the secret s.
B rejects the proof if $y = 0$ (an impostor could falsify the proof with $r = 0$) and accepts the proof if $y^2 \equiv x \cdot v^e \pmod{n}$.
Steps (1) to (3) are repeated until a sufficient confidence margin is reached.
Property Verification:

- **Completeness**: If A knows s, the protocol accepts the identification proof.
- **Soundness**: In the simple case, an impostor could only answer $e = 0$. Otherwise, it could choose a random r and send $x = r^2/v$ in (1) and respond to the challenge $e = 1$

with a correct answer $y = r$. For $e = 0$, it would need to compute the square root of x mod n (n composite with unknown factorization), which is hard by SQROOTP. Proof success thus requires secret possession.

- **Zero-Knowledge**: B cannot obtain any information about s because when $e = 1$, it is hidden by a random number (blinding factor).
- **Perfect Zero-Knowledge**: The pairs (x,y) obtained from A can also be simulated by B by choosing a random y and computing $x = y^2$ or $y^2/v \bmod n$. It can be proven that these pairs have an identical probabilistic distribution to those provided by A (who computes them differently!).

Note that, despite this last property, B is unable to impersonate A to B' because it cannot predict the challenge values e.

Quick Review

Secret: s such that $v = s^2 \bmod n$ (public key)
Protocol: Witness r^2, challenge $e \in \{0, 1\}$, response $y = r \cdot s^e$
Verification: $y^2 \equiv x \cdot v^e \pmod{n}$
Perfect ZK: Pairs (x,y) simulatable by B

ZKIP – Practical Implementations

Efficient Protocols

Practical implementations improve Fiat-Shamir's efficiency.

Feige-Fiat-Shamir (FSS):

- Uses multiple witnesses and challenges (k values) per iteration
- Cheating probability: 2^{-nk} for n iterations
- Reduces required exchanges

Guillou-Quisquater (GQ):

- Based on Fiat-Shamir but with expanded challenge domain
- Reduces guessing probability without increasing exchanges
- Better efficiency/security trade-off

Schnorr:

- Based on discrete logarithm difficulty (DLP)
- Very large challenge domain
- **Identification in just 3 exchanges**
- Sometimes sacrifices perfect zero-knowledge for efficiency

Advantages: More efficient than RSA, implementable on low-capacity devices (smart cards).

Original Text

ZKIP: Common Implementations
Feige-Fiat-Shamir (FSS):

- Based on the Fiat-Shamir protocol but using multiple witnesses and challenges (sets of k values) per iteration; for n iterations, this gives a probability of 2^{-nk} of guessing all responses.

Guillou-Quisquater (GQ):

- Also based on Fiat-Shamir but increasing the challenge choice, which reduces the guessing probability without increasing the number of transferred instances and protocol steps.

Schnorr:

- Based on the difficulty of computing discrete logarithms (DLP)
- It also uses a very large challenge domain, allowing identification in **just 3 message exchanges**.

These protocols are significantly more efficient than RSA and can be implemented on low-computing-capacity devices (smart cards).
They satisfy completeness and soundness properties, but the zero-knowledge property is sometimes sacrificed (as in Schnorr's case) to increase efficiency.
For a detailed description of these protocols, refer to [Men97] or [Sti95].

Quick Review

FSS: Multiple witnesses/challenges → probability 2^{-nk}
GQ: Expanded challenge domain → fewer exchanges
Schnorr: DLP + large challenges → **3 exchanges only**
All: More efficient than RSA, suitable for smart cards

ZKIP – Mafia Attack and Final Remarks

Vulnerabilities and Countermeasures

Even robust ZKIP protocols can be vulnerable to sophisticated attacks.

Mafia Attack (1989, Adi Shamir):

Scenario: C (attacker) and D (accomplice) collaborate so D impersonates A to B.

```
A   C: ZKIP Instance     D   B: ZKIP Instance
```

C relays A's messages to D (accomplice), who uses them to authenticate to B. The attack is transparent to A and B.

Countermeasures:

- **Faraday cages** (blocking radio communications)
- **Strong synchronization** to prevent side exchanges
- **Distance bounding protocols** limiting response delay

General Recommendations:

- Choose proven solutions over inventing new ones
- Verify objectives are met
- Analyze practically (reflection attacks, redundancy, etc.)
- Analyze formally (BAN logic, model checking)

Original Text

ZKIP: Final Remarks
ZKIPs offer a very high level of cryptographic security. They allow identifications while minimizing the chances of a hypothetical impostor and, most importantly, protecting the secret information of "honest" users.
In 1989 (SECURICOM'89), Adi Shamir said about ZKIPs: "I could go to a Mafia-owned store a million times in a row, and they still wouldn't be able to impersonate me"...
And yet: A participates in a ZKIP with C (Mafia); simultaneously, D (C's accomplice) participates in another ZKIP pretending to be A to B (an "honest" verifier).

```
(1) A → C: t1 (witness that C forwards via radio to D)
(1') D → B: t1
(2') D ← B: d1 (B sends the challenge to D; D forwards it to C...)
(2) A ← C: d1 (C resumes the challenge in its dialogue with A)
(3) A → C: r1 (the response using its secret, which C sends to D)
```

```
(3') D → B: r1 (B accepts r1 and so on!)
```

Solutions:

- Conduct identifications in Faraday cages…
- Use strong synchronization algorithms to prevent side exchanges.

Authentication: Summary – Attacks and Protections

Attack	Description	Protection
replay	replay a previous identification instance	zero-knowledge, challenge-and-response, one-time password (beware of pre-play!)
known/chosen-plaintext	obtain plaintext/ciphertext pairs	zero-knowledge
chosen-ciphertext	make A decrypt (or sign) carefully chosen information	zero-knowledge, ch. & resp. + knowledge witness + structure (redundancy!)
reflection	return the same number received	include target entity in messages, asymmetry in messages
interleaving	use messages from multiple simultaneous protocol instances	include target entity in messages, introduce cryptographic chaining between messages of the same identification instance
collusion	collusion between participants	Faraday cage, strong synchronization

Quick Review

Mafia Attack: Relay messages via accomplice → transparent fraudulent authentication
Protections: Faraday cage, strong synchronization, distance bounding
Attack Table: replay, chosen-plaintext/ciphertext, reflection, interleaving, collusion

Summary – Attacks and Protections

Attack	Description	Protection
replay	replay a previous identification instance	zero-knowledge, challenge-and-response, one-time password (beware of pre-play!)
known/chosen-plaintext	obtain plaintext/ciphertext pairs	zero-knowledge
chosen-ciphertext	make A decrypt (or sign) carefully chosen information	zero-knowledge, ch. & resp. + knowledge witness + structure (redundancy!)
reflection	return the same number received	include target entity in messages, asymmetry in messages
interleaving	use messages from multiple simultaneous protocol instances	include target entity in messages, introduce cryptographic chaining between messages of the same identification instance
collusion	collusion between participants	Faraday cage, strong synchronization