Authentication Overview

Authentication refers to the process of ‘*verifying the identity of the communicating principals to one another*’

Usually sub-divided into

- *Entity authentication*

  Authentication in ‘real-time’ between two parties about ready to engage in communication

- *Message* (or *data origin*) authentication

  Relates to data such as email that may later need to be authenticated by the receiver as to the origin and time sent

Note that authentication of the origin of data also addresses **data integrity**, i.e., whether the message has been tampered with by unauthorized parties

This is true b/c unauthorized changes imply the data has a *new source*

Authentication is typically carried out between

- A **prover** *A*, e.g., a hardware token such as a smart card, and
- A **verifier** *B*, e.g., a secure server operated by your bank

Authentication Overview

The verifier *B* either

- Confirms or *accepts* the prover's identity as authentic or
- Terminates without acceptance, i.e., *rejects*

Note that the information exchanged with verifier *B* **must be designed to prevent reuse** by *B*, otherwise it could impersonate *A* to a third party *C*

Authentication can be used for security objectives including:

- Access control
- Entity and message authentication
- Data integrity
- Non-repudiation
- Key authentication

Authentication can be carried out using

- *Symmetric encryption techniques*, e.g., via message authentication codes or **MACs**
- *Public/private encryption schemes* via **digital signatures**
- Through *authenticated key establishment* methods

Authentication Overview

The most common usage models include **access control** to a resource, e.g., to computer accounts, ATMs, to software, to a building, etc.

The ***capabilities*** provided in the authentication protocol depend on the *security requirements*

- An authentication protocol may be **unilateral**, i.e., from prover to verifier, or it may be **mutual**
- Some protocols may **preserve privacy**, to prevent malicious adversaries from tracking instances of authentications between the prover and verifier over time
- Others may be **symmetric** in nature, requiring the use of a shared secret between the prover and verifier (a *trusted third party* (TTP) may be used as a liaison here)
- Or may be **asymmetric** with the prover and verifier maintaining their own private secrets

The *computational and communication overheads* of the protocols will depend on:

- The type of protocol
- Its security requirements
- The security properties that must be guaranteed

Entity Authentication

Entity authentication techniques can be divided into 3 categories:

- *Something you know*: Passwords, PINs and secret or private keys whose knowledge is demonstrated in challenge-response protocols
- *Something you possess*: Physical accessory, resembling a passport in function

  Magnetic-striped cards, smart-cards and hand-held customized calculators (password generators) which provide time-variant passwords
- *Something inherent*: Biometrics, e.g., human physical characteristics such as fingerprints, voice, retinal patterns and signatures

Passwords represent the most widely used form of authentication, but are considered **weak authentication protocols**

Passwords provide *unilateral* and *time-invariant* authentication

Here, the *userid* serves as the claim of identity and the password serves as **evidence** supporting the claim

Attacks include

- *Eavesdropping* to enable replay
- *Password guessing* such as dictionary attacks

Entity Authentication

On most systems, the passwords are **encrypted** using a *one-way function (OWF)* before being stored on disk

A technique called *salting* is also commonly used to make dictionary attacks more difficult by expanding the search space for the adversary

Two-stage authentication and **password-derived keys** address the *insufficient entropy* issue associated with human chosen passwords

- An *n*-digit PIN verifies the *user to the token*, e.g., smart card, in the first stage
- The token typically embeds additional secrets for use in stage two between the token and the system

A variant uses **passkeys** to map a user password to a cryptographic key using a OWF

The most secure of the weak authentication schemes uses **one-time passwords**, which addresses eavesdropping and replay attacks on password schemes

Entity Authentication

Challenge-Response protocols fall in the class of **strong authentication protocols**

Here, authentication requires the *prover* to demonstrate knowledge of a secret **without revealing the secret itself** to the *verifier*

The prover provides a *response* to a *time-variant challenge*, with the response **insep-arably** bound to both the secret and the challenge

The challenge can be a ***random number***, called a *nonce* (for ‘used only once’), a *sequence number* or a *timestamp*

Time-variant parameters are **countermeasures** to replay attacks and certain types of chosen-text attacks

This is true b/c the uniqueness and timeliness guarantees allow one protocol instance to be distinguished from another

Note that *challenge-response protocols* requires some type of computing device and secure storage for long-term keying material

Entity Authentication: Challenge-Response by Symmetric-key

Each pair of communicating parties share a secret key

In large communities, a TTP can provide session keys in real time to circumvent the need to distribute n^2 key pairs

A common form of unilateral authentication uses random number(s) (RN).

$A \leftarrow B : r_B$ (B generates random nonce r_B)

$A \rightarrow B : E_K(r_B, B^*)$

A. J. Menezes, et al. text

- B generates random nonce r_B and transmits it to A (over an unsecured channel)
- A encrypts the nonce and the identifier B using a shared secret key K and transmits the encrypted message back to B
- B then decrypts and 1) checks that the r_B received matches the r_B sent and 2) verifies B* is equal to his own B

The shared secret K must be securely transmitted to A and B beforehand, typically using a mechanism involving a TTP, in order for this scheme to work

Entity Authentication: Challenge-Response by Symmetric-key

Mutual authentication requires a second nonce r_A and a third message:

$A \leftarrow B: r_B$ (B generates nonce r_B)

$A \rightarrow B: E_K(r_A, r_B, B^*)$ (A generates nonce r_A)

$A \leftarrow B: E_K(r_B, r_A)$

A. J. Menezes, et al. text

Encryption ensures the nonces and identifiers are ‘inseparably’ bound as discussed above

Challenge-Response using Keyed One-Way Functions

Encryption is considered a ‘heavy weight’ cryptographic primitive, and may be replaced in resource-constrained devices by:

- A *one-way function* (OWF) or a non-reversible function with shared key, and
- A challenge

The encryption algorithm E_K is replaced by a MAC algorithm h_K, i.e., a keyed hash function

The receiver also computes the MAC and compares it with the received MAC

Entity Authentication: Challenge-Response using Keyed One-Way Functions

These protocols require an additional cleartext field r_A to be transmitted

$A \leftarrow B: r_B$ (B generates nonce r_B)

$A \rightarrow B: r_A, h_K(r_A, r_B, B)$ (A generates nonce r_A)

$A \leftarrow B: h_K(r_B, r_A, A)$

A. J. Menezes, et al. text

B confirms that the hash value received, designated as $h_k(r_A, r_B, B)$, is equal to the value he/she computes locally using the same hash function and shared secret K

A performs a similar validation using the transmitted hash $h_K(r_B, r_A, A)$ from B

The *computational infeasibility* of finding a second input to h_K that produces the same hash provides the security guarantee in this mutual authentication protocol

Challenge-Response by Public-Key

In this case, the prover **decrypts** a challenge using its secret key component of the public-private pair, which is **encrypted** by the verifier under its public key P_A

Alternatively, the prover can *digitally sign* a challenge

Entity Authentication: Challenge-Response by Public-Key

B chooses nonce *r*, computes the **witness** $x = h(r)$ (*h* is a OWF)

$$A \leftarrow B: h(r), B, P_A(r, B)$$
$$A \rightarrow B: r$$

A. J. Menezes, et al. text

Here, *x* demonstrates knowledge of *r* without disclosing it

B computes challenge $e = P_A(r, B)$

A decrypts *e* to recover *r'* and *B'*, computes $x' = h(r')$ and rejects if *x'* does not equal *x* or if *B'* does not equal *B*

Otherwise *A* sends $r = r'$ to *B*

B succeeds with unilateral entity authentication of *A* upon verifying the received *r* agrees with his *r*

The *witness* prevents chosen-text attacks