Information systems security

Lorenzo Di Maio

October 2024

Contents

Chapter 1

Authentication techniques protocols, and architectures

Authentication refers to the process of verifying the identity of an entity (whether it's a human, software component, or hardware element) before granting access to resources in a system. Authentication can be applied to various type of "actors", such as:

- **Human being**
- **Software component**
- **Hardware element**

Authentication vs Authorization

- **Authentication (authC/authN)**: established the identity of an entity.
- **Authorization (authZ)**: determines where a authenticated entity has permission to access.

1.1 Authentication factors

Authentication can be based on 3 primary factors:

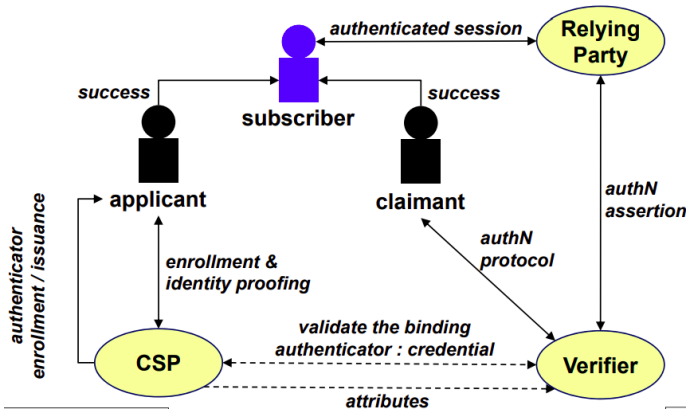
- **Knowledge**: Information that only the user knows and can provides as proof of their identity.
- **Ownership**: Physical object or device that only the user has access to.
- **Inherence**: This factor relies on unique biological traits of the user (e.g fingerprint).

N.B Authentication can be applied not just to human user, but also to processes and devices.

1.1.1 Risks

- **Knowledge:**
  - Storage → if passwords are stored improperly, they are vulnerable to thefts.
  - Demonstration → user might inadvertently reveal their password through social engineering.
  - Transmission → if passwords are sent over insecure channel, they can be intercepted by attackers.
- **Ownership:**
  - Authentication thieft
  - Cloning
  - Unathorized usage
- **Inherence:**
  - Counterfeiting → biometric data can be spoofed or replicated by attackers using sophisticated techniques.
  - Privacy → the use of biometric data raises the risk of biometric information being exposed.
  - Irreversibility → biometric traits cannot be replaced if compromised.

1.2 Digital Authentication model (NIST SP800.63B)



Entities:

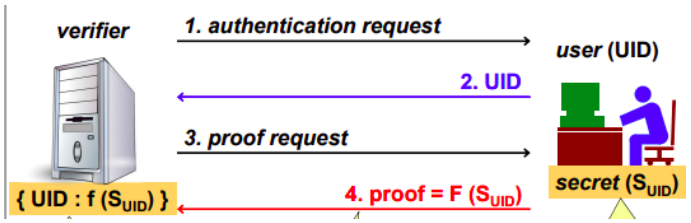
- **Subscriber**: applicant who has successfully completed identity proofing.
- **Applicant**: an individual applying to established a digital identity.
- **Claimant**: the user trying to prove their identity to access a system or service.
- **Relying Party**: entity (e.g service provider, website) that requests/receives an authN assertion from the verifier to asses user identity (and attributes).
- **Verifier**: validates the user's credential during each authentication event.
- **CSP**: an entity that issues, manages, and maintains **credentials** used by individuals to authenticate themselves.

Applicant vs Claimer

An applicant needs to enroll in the system for the first time to establish their identity. Instead, a claimant asserts their identity to gain access to the system after enrollment.

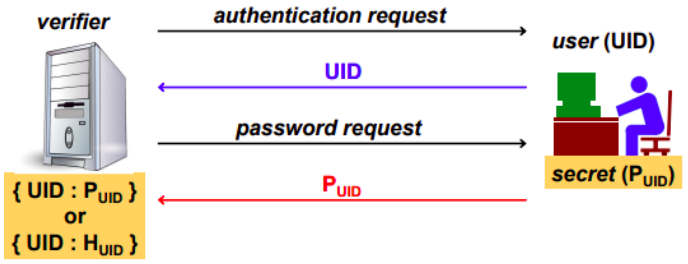
1.3 Generic authentication protocol

1. The user initiates an authentication request by sending their UID.
2. The user generates a proof based on their secret, using a secure function $F(S_{UID})$, and send this proof to the verifier.
3. The verifier checks if the received proof matches the stored representation of the secret.
4. If it matches, the user is successfully authenticated.



1.4 Password base authentication

1. The user sends their UID and P_{UID} (= Password) to the verifier.
2. The server verifies the proof:
   - If password are stored in cleartext, it directly compares the proof with the stored password.
   - If password are stored in hashes, it hashes the proof and compares it to the store hash H_{UID}.



Problems of reusable Passowrds

- **PWD Sniffing** (attackers intercept password during transmission)
- **PWD Database attack** (if DB contains plaintext or obfuscated PWD)
- **PWD Guessing** (very dangerous if it can be done offline, e.g against a list of PWD hashes)
- **PWD Enumeration** (PWD brute force attack)
  - If PWD is limited in length and/or character type.
  - If authN protocol does not block repeated failures.
- **PWD Duplications** (using the same PWD for one service against another one, due to user PWD reuse)
- **Cryptographic Aging** (as computing power grows, older cryptographic methods become vulnerable to new attacks)
- **PWD capture via server spoofing and phishing** (attackers deceive user into giving away their PWD by pretending to be legitimate service)

Password best practies

Suggestion to reduce password risks:

- Use alphabetical characters (upper case + lower case), digits and special characters
- Make passwords long (at least 8 character)
- Never use dictionary words
- Change password regularly, but not too frequently
- Do not reuse passwords across different services

Password storage

- **Server Side:**
  - Passwords should never be stored in cleartext.
  - Encrypted passwords aren't ideal since the server would need to know the encryption key.
  - Better to store a password digest (hashed password), though vulnerable to dictionary attacks.
  - Rainbow tables can speed up these attacks, so it's important to add a "salt" (random variation) to each password.
- **Client-side:**
  - Ideally, passwords are memorized by the user, but having many passwords makes this difficult.
  - People may resort to writing them down or using simple passwords, which is risky.
  - Using a password manager or encrypted file is a safer alternative.

1.5 The "dictionary" attack

- **Hypothesis:** The attacker knows the hash algorithm and the hashed password values.
- **Pre-computation:** For each word in a dictionary, compute and store its hash $store(DB, Word, hash(World))$
- **Attack process:**
  - Let HP (=hash password) to be the hash of an unknown password.
  - Lookup HP in the precomputed dictionary (DB) to find a matching password.
  - If found, output the password; if not, indicate it's "not in dictionary".

1.6 Rainbow Table attack

Rainbow Table is a **space-time trade-off technique** that reduces storage needs for exhaustive hash tables, making certain brute-force attacks feasible within limited space. It uses a reduction function $r : h \to p$ (which is NOT h^{-1}) to generate chains of hashes.
Example:

- For a 12-digit password, an exhaustive hash table would require $10^{12} rows(P_i : HP_i)$
- rainbow = 10^9 rows, each representing 1000 possible passwords.

Attack

```
for (k=HP, n=0; n<1000; n++)
 ■ p = r(k)
 ■ if lookup( DB, x, p ) then exit ( "chain found, rooted at x" )
 ■ k = h(p)
exit ( "HP is not in any chain of mine" )
```

1.7 Salting Passwords: A Defense Against Dictionary and Rainbow Table Attacks

Salting passwords is a security technique used to protect stored passwords from dictionary attacks and rainbow table attacks. A salt is a unique, random string added to each password before hashing. This ensures that even if two users have the same password, their hashes will be different due to the unique salt.

Steps for each user (UID):

- Generate or ask for the user's password.
- Create a unique, random slat for each user.
- Compute the salted hash: $SHP = hash(password \mid\mid salt)$
- Store the triplet $\{UID,\ SHP,\ salt\}$

Password Verification with Salt

- **Claimant:** Provides their user ID (UID) and password (PWD).
- **Verifier:**
  - Uses the UID to find the stored salted hash (SHP) and salt.
  - Computes $SHP' = hash(PWD \mid\mid salt)$.

The Linkedin attack

In 2012, LinkedIn was breached, exposing 6.5 million unsalted SHA-1 password hashes. The lack of salting allowed attackers to crack at least 236,578 passwords through crowdsources efforts before restrictions halted the exposure.

1.8 Strong authentication definitions

The concept of strong authentication (authN) is crucial in ensuring secure identity verification, but it has never been formally defined with a universal definition. Various definitions exist depending on the context, such as the European Central Bank (ECB) and PCI-DSS.

ECB definition

The ECB defines strong authentication as a process that involves at least two independent elements from **knowledge** (e.g. password), **ownership** (e.g. smartcard), and **inherence** (e.g. biometrics). The key requirement is that these elements must be mutually independent, so compromising one should not affect the others. Furthermore, at least one element should be **non-reusable** or **non-replicable** (except for inherence), with the entire process safeguarding the confidentiality of the authentication data.

PCI-DSS Definition

PCI-DSS mandates **multi-factor authentication (MFA)** for access to cardholder data, particularly for administrators and remote access from untrusted networks. Since version 3.2, MFA has become compulsory for remote access, and the use of the same factor twice (e.g., two passwords) does not qualify as MFA.

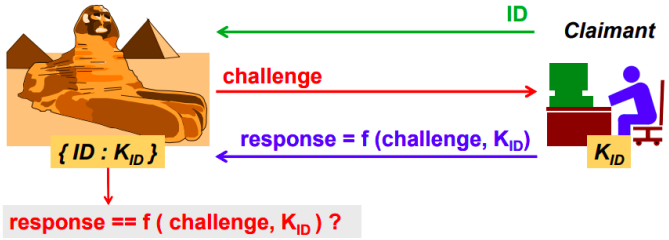
1.9 Challenge-Response Authentication (CRA)

Challenge-response authentication (CRA) is a widely used technique where a challenge is issued, and the claimant responds by solving it with a secret (shared or private). The challenge must be **non-repeatable** (usually a random nonce) to avoid replay attacks. The function used to compute the response must be **non-invertible**, otherwise, a listener can record the traffic and easily find the shared secret:

$$if(\exists f^{-1})\ then\ K_c\ =\ f^{-1}(response, challenge)$$

1.9.1 Symmetric CRA

In symmetric CRA, both the client and the server share a secret key that is used to verify the authenticity of a user or system This method is fast, often utilizing hash functions (e.g., SHA1, SHA2, SHA3).



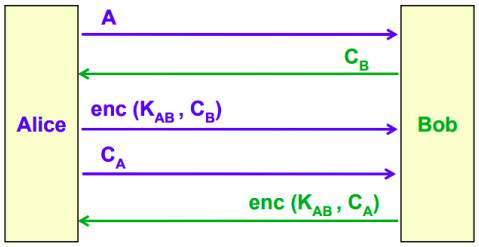
1.9.2 Mutual symmetric CRA

Mutual symmetric CRA requires both parties to authenticate each other. However, it's a old protocols so it has many vulnerabilities.

Version 1: Basic Exchange

In this case, the initiator explicitly provides its claimed identity (This version is considered outdated and insecure).
Process:

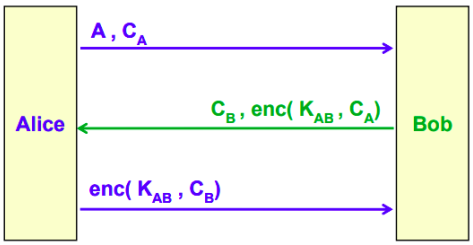
- Alice sends an encrypted challenge (C_B) to Bob using the shared key K_{AB}.
- Bob responds with an encrypted challenge (C_A) for Alice, also using K_{AB}.



Version 2: Improved Performance

Optimized by reducing the number of messages, which improves performance without compromising security.
Process:

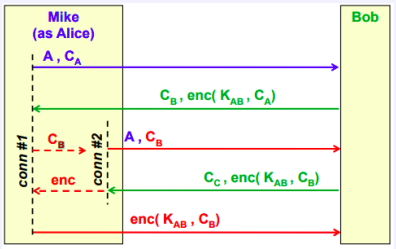
- Alice includes her identity (C_A) and sends an encrypted challenge (C_B) in the same message.
- Bob responds with his encrypted challenge C_A to complete the exchange.



Attack on Mutual Symmetric CRA

A potential attacker, "Mike" (posing as Alice), exploits the protocol by mimicking responses:

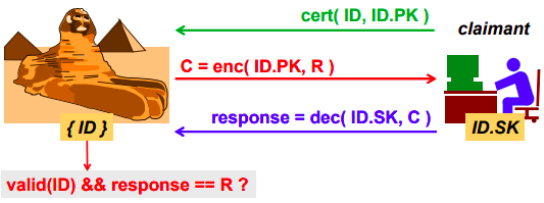
- The attacker intercepts Alice's identity (C_A) and Bob's challenge (C_B).
- The attacker uses the shared key K_{AB} to manipulate responses and mimic both parties.



1.9.3 Asymmetric CRA

Process:

- A **random nonce (R)** is generated by the Verifier.
- The verifier encrypts R using the user's public key ($ID.PK$) and sends it to the Claimant: $C = enc(ID.PK, R)$
- The Claimant decrypts C using their private key ($ID.SK$) and sends R back in cleartext: $response = dec(ID.SK, C)$
- The Verifier validates: $valid(ID)$ && $(response == R)$.



Applications

- Widely implemented in secure communication protocols like IPsec, SSH, and TLS.
- Fundamental in modern authentication frameworks such as FIDO.

Asymmetric CRA analysis

Security:

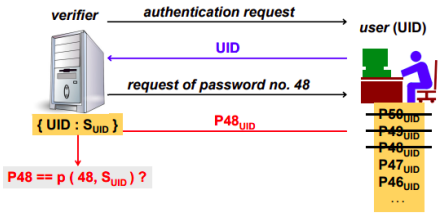
- It's the strongest mechanism.
- Does not require the Verifier to store any shared secret, reducing potential attack vectors.

Problems:

- It **slower** compared to symmetric methods.
- If designed inaccurately may lead to an involuntary signature by the Claimant.
- Trust issues managing root certificates, name constraint, and certificate revocation.

1.10 One-Time Password(OTP)

One-Time Passwords are temporary and valid for a single use in an authentication session. They mitigate risks like password reuse and passive sniffing but can still be vulnerable to man-in-the-middle (MITM) attacks. These passwords are often designed with random characters to prevent guessing, but this can make password insertion difficult for users.



1.10.1 S/KEY System

S/KEY System was the first OTP implementation by Bell Labs (1981). It pre-computes a sequence of passwords derived from a user's secret. Each password is validated and replaced with its predecessor, ensuring security without storing the secret:

$$Secret = S_{ID}$$

$$P_1 = h(S_{ID}), P_2 = h(P_1), ..., P_N = h(P_{N-1})$$

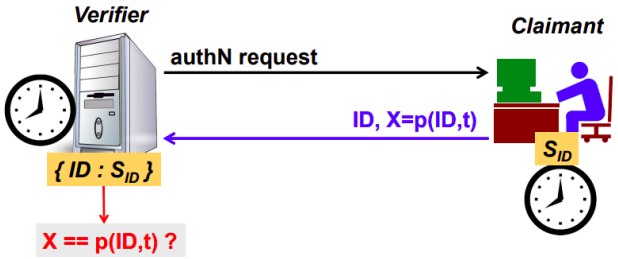
This approach minimizes verifier storage needs and offers robust protection, with users solely responsible for password retention.

One-time generation with S/KEY
In the S/KEY system, the user creates a secret passphrase (PP), which is combined with a server-provided seed to generate a 64-bit password. The passphrase is concatenated with the seed, and an MD4 hash is used to produce the password. The result is presented as six short words from a shared dictionary, making it easy to remember. This method allows secure password generation while using the same passphrase across multiple servers with different seeds. If the passphrase is compromised, security is at risk.

1.10.2 Time-based OTP (TOPT)

TOTP systems generate passwords based on the user's secret and the current time, requiring synchronization between the user and the verifier:

$$p(ID, t) = h(t, S_{ID})$$



Authentication Process:

- The claimant sends an authentication request with ID and the generated OTP x.
- The verifier checks if X matches the computed OTP for the corresponding ID and t.

Requirements:

- Local computation of OTPby the subscriber.
- Clock synchronization (or keeping track of time-shift for each subscribers).

Limitations:

- Only one authentication is allowed per time-slot, typically 30s or 60s.
- This time limit may not suit all services.

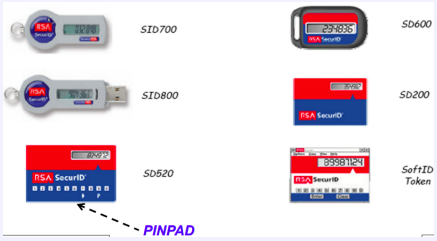
Vulnerabilities:

- Potential attacks on the subscriber and verifier:
  - Fake NTP servers or compromised mobile network femtocells.
  - Sensitive database storage at the verifier (e.g., the RSA SecurID attack).

Example: RSA SecurID

Authentication process:

- **The claimant sends to the verifier:**
  - Without PIN Pad: User ID, PIN, and Token Code (computed using seed and time).
  - With PIN Pad: User ID and a combined Token Code* (includes seed, time, and PIN).



1.10.3 Out-of-Band (OOB) OTP

Out-of-Band OTP requires a **secure channel** with server authentication to prevent MITM (Man-In-The-Middle) attacks. Traditionally, it uses text or SMS as the communication channel, but this method is increasingly vulnerable due to weaknesses in VoIP, mobile user identification, and the SS7 protocol. Nowadays, a push mechanism over a **TLS-secured channel** to a registered device is recommended for enhanced security.

1.10.4 Two-/Multi-Factor Authentication (2FA/MFA)

MFA enhances authentication (authN) by requiring multiple factors, such as a PIN, OTP, or biometrics, to verify identity. These factors can include something you know (like a PIN or password), something you have (like a token or phone), and something you are (like biometric data). MFA also protects the authenticator, for example, by using a PIN to safeguard it, but risks arise if the lock mechanism is weak or if there's no protection against multiple unlock attempts.

Importance of MFA: The iPhone Ransomware (2014)

In 2014, iCloud accounts with 1FA were hacked, allowing attackers to lock devices remotely. Victims were extorted for $100, but paying didn't help as the PayPal account was fake. This incident underscores the need for MFA to secure devices and prevent such attacks.

1.11 Authentication of human beings

To verify whether we're interacting with a human rather than a machine, there are two common approaches:

- **CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart):** A method where users must solve challenges like distorted characters in images to prove they are human.
- **Biometric Techniques:** These involve verifying human characteristics such as fingerprints, voice, retinal scans, iris scans, blood vein patterns in hands, heart rate, and hand geometry.

Problems of biometric systems

There are several issues with biometric systems:

- **FAR (False Acceptance Rate)** and **FRR (False Rejection Rate)**: These rates depend on the system's cost and can be adjusted, but biological factors like injuries or emotional changes can affect accuracy.

- **Psychological Acceptance:** Many people fear the "Big Brother" scenario—personal data collection and potential privacy invasions.
- **Irreplaceability:** Once compromised, biometric data cannot be changed, unlike a password or PIN. Thus, biometrics are primarily useful for local authentication but unsuitable for global identity systems.
- **Lack of Standardization:** High development costs and dependency on specific vendors are significant drawbacks in current biometric systems.

1.12 Kerberos Authentication System

Kerberos is a widely-used authentication protocol based on a Trusted Third Party (TTP) model. It's designed to ensure that user passwords are never transmitted over the network. Instead, the password is used locally for encryption.

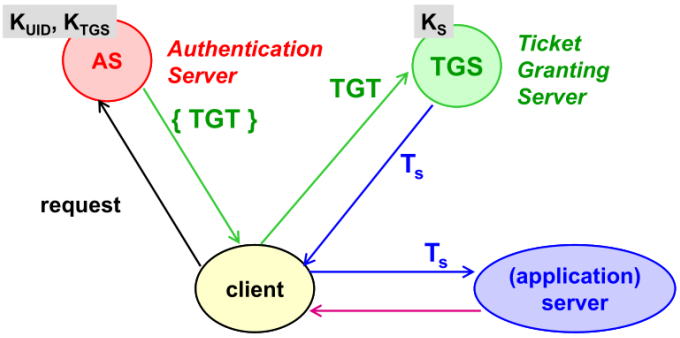
- **Realm:** Refers to a Kerberos domain, grouping together all systems that use Kerberos for authentication.
- **Credential:** A unique identifier for a user, typically in the format **user.instance@realm**.

1.12.1 Players

There are 4 key players:

- **Client (User/Application:)** the person or application trying to access a resource.
- **Authentication Server (AS):** Confirms who you are and issues a Ticket Granting Ticket (TGT).
- **Ticket Granting Server (TGS):** Issues tickets for specific services after seeing the TGT.
- **Service (Server):** The resource you want to access.

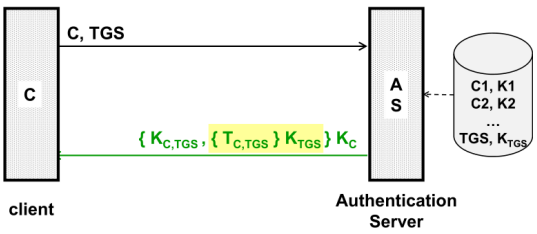
1.12.2 How it Works?



1. **TGT Request**: Client authenticates with the Authentication Server (AS) to obtain a Ticket Granting Ticket (TGT).

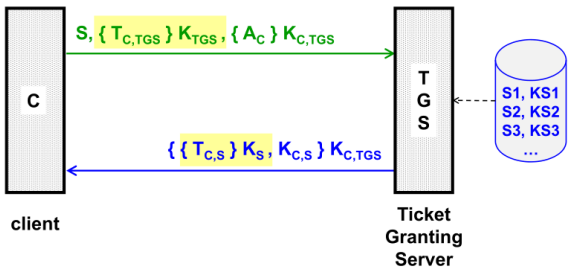
   The expression $KC, TGS, TC, TGS\ KTGSKC$ represents a **TGT**:

   - The enitre structure is encrypted using the client's secret key (KC) (This ensure that only the intended client can decrupt and use the TGT).
   - $TC, TGS\ KTGS$ is the **TGS** that contains the client's identity and other information. It's encrypted using the TGS's secret key ($KTGS$), ensuring only TGS can read and verify the ticket.



2. **Service Ticket Request**: TGT is sent to the Ticket Granting Server (TGS) to request a service-specific ticket.

   - The client sends: (S) $\rightarrow$ the indetifier of the target device, $TG, TGS\ \ KTGS$ $\rightarrow$ TGS ticket and ($AC\ KC, TGS$) $\rightarrow$ the authenticator.
   - TGS Verifies and Responds:
     - The TGS decrypts the TGS ticket using its secret key (KTGS).
     - It verifies the client's identity using the authenticator.
     - If successful, the TGS generates a Service Ticket ($TC, S\ KS$) encrypted with the secret key (KS), ensuring only the service can read it; and a new session key (KC, S), shared between the client and the service.

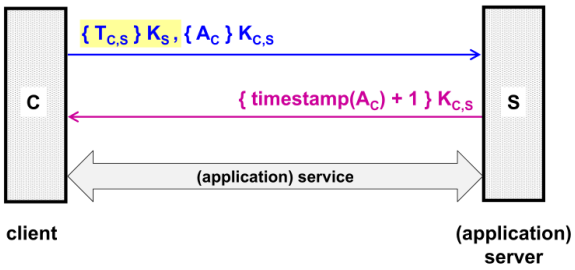


3. **Access Service**: Client uses the service ticket to authenticate and access the resource.

   The client send:

   - The service ticket encrypted using the service's secret key (KS), so only the service can decrypt and validate it.
   - The authenticator encrypted using the session key (KS, S) shared bewteen the client and the service.

   The service responds with a response in which confirms successfully authentication. It encrypts the message with the session key (KC, S) to ensure it's secure and readable only by the client.



1.12.3 Single Sign-On (SSO)

SSO allows users to authenticate once and access multiple services without repeated logins. Types of SSO:

- **Fictitious SSO:**
  - Relies on tools like password synchronization or management (e.g., password wallets).
  - Limited to specific applications.
- **Integral SSO:** Uses advanced multi-application methods like Asymmetric Challenge-Response Authentication (CRA) or Kerberos. Often requires application changes.
- **Multi-Domain SSO:** Expands SSO across domains using technologies like SAML tokens, which generalize Kerberos tickets.

N.B. Single Sign-On (SSO) is not exclusive to Kerberos, but Kerberos is one of the prominent technologies that implements SSO capabilities.

1.13 Authentication Interoperability

Authentication interoperability define methods, standard, and protocol for performing authentication securely and efficiently. Let's start to analyze some framework.

1.13.1 OATH

The Open Authentication (OATH) framework provides standards for one-time password (OTP) and symmetric key management.

- **HOTP:** Uses a shared secret key (K) and a counter (C) to generate an OTP.
  Function:
  $$HOTP(K, C) = sel(HMAC - h(K, C)) \ \& \ 0x7FFFFFFF$$
  The result is truncated and transformed into an N-digit code (e.g., 6 digits)[1].
- **TOTP:** Similar to HOTP but uses time intervals (TS) instead of counters.
  Function:
  $$C = (T - T0)/TS$$
  With default values: $T0$ -> Unix epoch, TS -> 30-second intervals, T -> unixtime (now).
- **OCRA** (OATH Challenge-Response Algorithm)
- **PSKC** (Portable Symmetric Key Container): XML-based format for symmetric key transport.
- **DSKPP** (Dynamic Symmetric Key Provisioning Protocol): A client-server protocol for securely provisioning symmetric keys.

1.13.2 Google Authenticator

Supports HOTP and TOTP with adjustments for usability:

- **K:** Base-32 encoded.
- **C:** 64-bit unsigned integer.
- **sel(X):** Uses the 4 least-significant bits of X to locate a portion of the result.
- **Defaults:** TS = 30 seconds, N = 6 digits (zero-padded if necessary).

[1] K-> shared key, C -> counter, sel -> function to select 4 bytes out of a byte string

1.13.3 FIDO (Fast Identity Online)

FIDO, developed by the FIDO Alliance, improves authentication by offering secure, password-less, and multi-factor methods. It uses biometric data locally to unlock cryptographic keys and employs asymmetric cryptography for signing challenges or transactions. FIDO prevents phishing by ensuring that authentication responses cannot be reused. Each response is a unique signature created over various data, including the Relying Party (RP) identity, making it specific to the service being accessed. Additionally, a new key pair is generated during each registration, which prevents the association of a user's identity across different services or accounts.

FIDO's frameworks include:

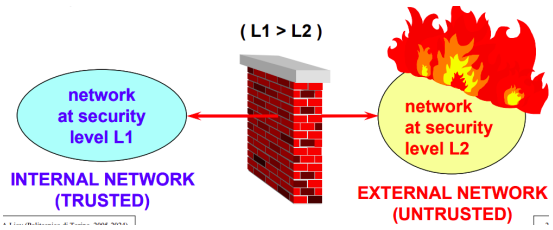
- **UAF** (The Universal Authentication Framework): for password-less login.
- **U2F** (Universal 2nd Factor) for device-based two-factor authentication.
- **FIDO2**: integrates U2F with WebAuthn for robust web authentication.

Chapter 2

Firewall and IDS/IPS

2.1 What is a Firewall?

A firewall acts as a protective barrier, much like a physical wall against fire. It is primarily a controlled connection point between networks with differing security levels. Its key functions include:



- **Boundary Protection:** Serving as a network filter between trusted (internal) and untrusted (external) networks.
- **Compartmentalization:** Dividing network zones based on security levels to enforce stricter control.

2.2 Ingress vs Egress firewall

Firewalls can be classified based on the direction of traffic they manage:

- An **ingress firewall** focuses on incoming connections, typically regulating access to public services provided by the network or supporting exchanges initiated by internal users.
- An **egress firewall** monitors outgoing connections. It's typically used to check the activity of internal personnel [Works well for channel-based services (e.g., TCP applications) but faces challenges with stateless, message-based services (e.g., ICMP, UDP)].

2.3 The three principles of the firewall

1. The firewall should be the only connection between the internal and external networks.
2. Only the "authorized" traffic is allowed to pass the firewall.
3. The firewall itself must be secure against potential vulnerabilities.

These principles were outlined by *D.Cheswick, S.Bellovin.*

2.4 Authorization policies

- **Permitlist/allowlist**: All that is not explicitly permitted, is forbidden.
  - It offers higher security but it's difficult to manage.
- **Blocklist/denylist**: All that is not explicitly forbidden, is permitted.
  - It's less secure but it's more easy to manage.

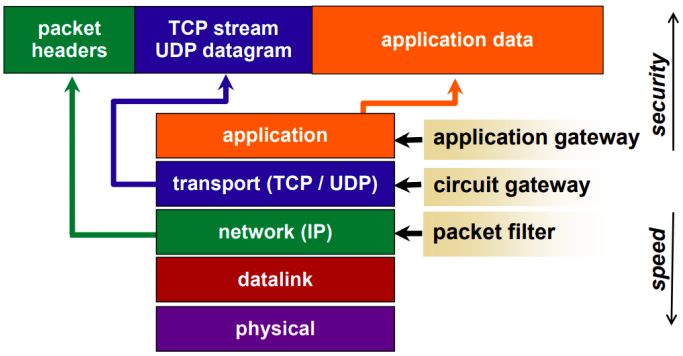
2.5 Basic components

Firewalls include several key components that work together to provide security:

- **Packet Filter/ Screening Router / Choke**: Filters traffic at the network level based on packet attributes such as IP headers and transport headers.
- **Bastion Host**: A secure system with auditing capabilities, often positioned to handle critical traffic.
- **Application Gateway (Proxy)**: Acts on behalf of an application, managing access control and providing detailed packet inspection at the application level.
- **Dual-Homed Gateway**: A system with two network interfaces and disabled routing, isolating internal and external networks. (In the way we can decide which packets are sent from a network to another).

2.6 A which level the controls are made?

Firewalls at higher levels provide more security but tend to be slower, while firewalls at lower levels offer less security but greater speed.



To undersand:

security is better up → speed is better down
security is better down → speed is better up

2.7 Network-Level Controls

Firewalls operate at various levels of the network stack applying different controls based on the type of firewall used:

- **Packet Filters**: Operate at the network level, inspecting packet headers.
- **Stateful Packet Filters**: Track connection states for more dynamic filtering.ù
- **Circuit-Level Gateways / Proxies**: Work at the transport level, ensuring secure connection establishment.
- **Application-Level Gateways / Proxies**: Provide inspection and control at the application layer, often with more granular rules.

2.7.1 Packet filter

A packet filter inspects packets based on headers, such as IP and transport headers. These filters are traditionally available on routers and now in most OS. They allow for rules like permitting incoming connections to specific services (e.g., a web server) or limiting DNS queries from specific internal servers.

- **Pro:**
  - Independent of applications, scalable, and cost-effective (available in many OS and routers).
  - Good performance and low cost.
- **Cons:**
  - Vulnerable to attacks like IP spoofing or fragmented packets.
  - Difficult to support services using dynamically allocated ports (e.g. FTP).
  - Complex configuration and hard to implement user authentication.

2.7.2 Circuit-Level Gateway

A circuit-level gateway acts as a transport-level proxy, creating secure communication channels between client and server without inspecting the payload. It protects against Layer 3/Layer 4 attacks like IP fragmentation or TCP handshake exploits.

- **Pro:**
  - Provides protection by isolating the server from attacks.
  - Offers client authentication and eliminates many low-level attack vectors.
- **Cons:**
  - Still shares many limitations of packet filters and requires modifications to the application for full functionality.

2.7.3 Application-Level Gateway

An application-level gateway (or proxy) operates at the application layer, inspecting the payload of packets. These proxies often require modifications to client applications and can enhance security by checking the semantics of application data (e.g., HTTP methods) or performing peer authentication.

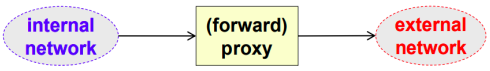
- **Pro:**
  - Strong security against application vulnerabilities (e.g., buffer overflow attacks).
  - Fine-grained access controls and the ability to mask internal IP addresses.
  - May provide protection against attacks like buffer overflows.
  - Not transparent to the client and may break the client-server model.
- **Cons:**
  - Requires specific proxies for each application and can introduce delays when supporting new applications.
  - High resource usage and lower performance due to user-mode operations.

Variants of application-level proxies include **transparent proxies**, which are less intrusive to clients, and **strong application proxies**, which focus on checking data semantics, not just syntax.

2.7.4 HTTP Proxies

An HTTP forward proxy is a server that acts as an intermediary or front-end for client requests. It receives requests from internal users and forwards them to the real external server (So it's a egress control). **Benefits:**

- **Shared Cache**: External web pages are cached, reducing load times for all internal users.
- **Authentication and Authorization**: Enforces user authentication and controls access for internal users.



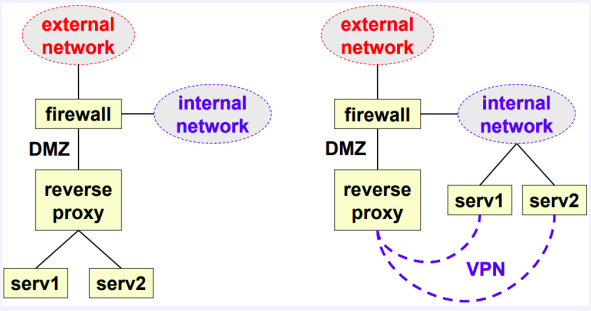
Type of Proxies

- **Forward Proxy (HTTP Proxy)**: This type of proxy controls outgoing traffic from the internal network to the external one, enforcing access controls and caching content. It provides features like user authentication and authorization, content filtering, and bandwidth control.
- **Reverse Proxy (HTTP Reverse Proxy)**: A reverse proxy sits in front of web servers, providing additional services like load balancing, content inspection, TLS acceleration, and caching. It can obfuscate the internal server structure, improving security, and support performance optimizations like dynamic page feeding based on client speed.

Reverse proxy: possibile configuration

Key components:

- **External Network (Red Cloud)**
- **Firewall**
- **DMZ (Demilitarized Zone)**: A buffer zone between external and internal networks where systems that interact with the external network are placed for added security.
- **Reverse Proxy**
- **Internal Network**: The secure area where your actual servers (e.g., serv1, serv2) reside.



How configurations work?

Left configuration: Direct reverse proxy setup

- **Flow:** External user → Reverse Proxy → Internal Servers (serv1, serv2).
- **Benefits:** Internal servers are hidden and protected, while performance and security are improved.

Right Configuration: Reverse Proxy with VPN

- **Flow**: External user → Reverse Proxy → Encrypted VPN connection → Internal Servers (serv1, serv2).
- **Benefit**: Adds an extra layer of security through encryption for communication between the reverse proxy and internal servers.

2.7.5 WAF (Web Application Firewall)

A WAF is a module installed at a proxy (forward and/or reverse) to filter the application traffic. Filters the followings types of traffic:

- HTTP commands.
- HTTP request/response headers.
- HTTP request/response content.

Popular WAF example: ModSecurity

ModSecurity is a widely-used WAF plugin for web servers like Apache and NGINX, offering protection through predefined rules like the OWASP ModSecurity Core Rule Set (CRS).

2.8 Firewall's Architectures

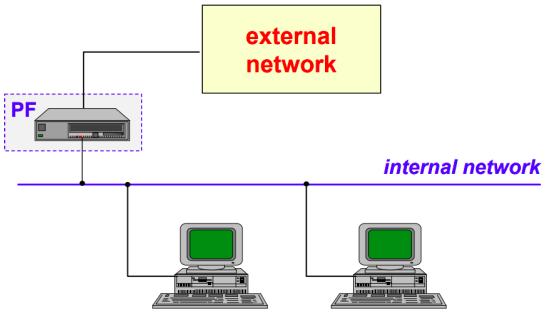
2.8.1 "Packet Filter" architecture

What it does?

This architecture uses a simple packet filter to screen traffic at both the IP and higher protocol levels (like TCP)

Key Points:

- If implemented with a router then it's a "screening router" and there's **no need for extra dedicated hardware**.
- The packet filter element represents a **single point of failure**.
- **There's no need for proxies** or application modifications.



Beware

Simple, cost-effective, but insecure!

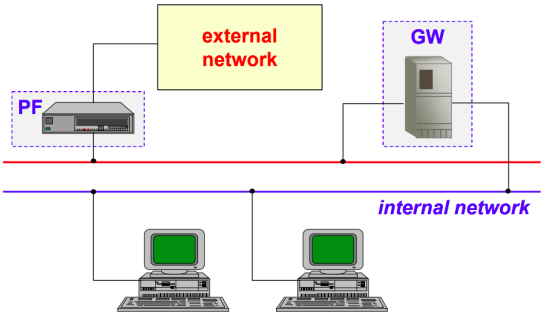
2.8.2 "Dual-homed Gateway" architecture

What it does?

Uses a system with two network interfaces (hence "dual-homed") to provide a basic firewall between external and internal networks.

Key Points:

- Easy to implement with small hardware requirements.
- The internal network can be masquerade.



Beware

Inflexible, with higher management overhead and reduced flexibility in handling complex network setups.

2.8.3 "Screened host" architecture

What it does?

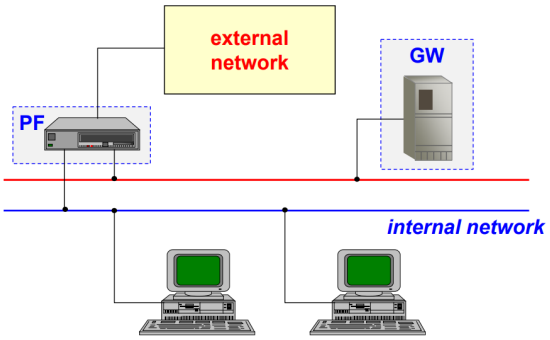
Uses two primary components:

- **Router**
- **Bastion host**

This setup aims to provide enhanced security by controlling the flow of traffic between the internal and external networks.

Key Points:

- Traffic from the internal network (INT) to external (EXT) is blocked unless it's from the bastion host. Similarly, external traffic is blocked unless directed to the bastion.
- More flexible (skip control over some services / hosts)



Beware

- **More Expensive and Complex** to manage: Requires two systems (router + bastion host) instead of just one.
- **Limited Masking**: Only the host and protocols that go through the bastion host can be masked for security (such as hiding internal IP address or data). However, if the packet filter (PF) uses NAT, it can mask additional traffic and hide internal network details.

2.8.4 "Screened subnet" architecture

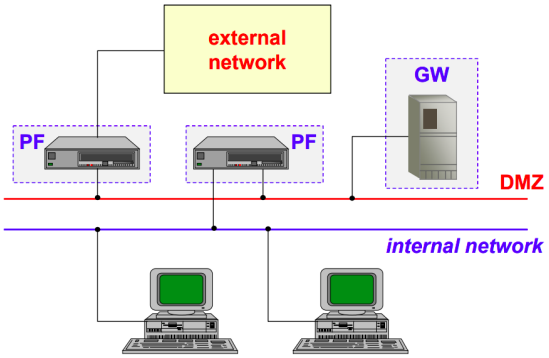
What it does?

A DMZ (De-Militarized Zone) is created between an internal network and a external network. The DMZ acts as a buffer zone to isolate external entities from the internal network, such as web servers or remote access points.

Traffic flows:

- The first firewall controls access from the external network to the DMZ, ensuring only authorized traffic can reach the public-facing systems.
- The second firewall restricts traffic from the DMZ to the internal network, allowing only specific and necessary connections.

It the most secure solution because this setup hides the internal network from external users, reducing exposure to attacks.



Beware

- **Higher cost** due to the additional hardware.
- **The DMZ is home not only to the gateway** but also to other host (typically the public servers).

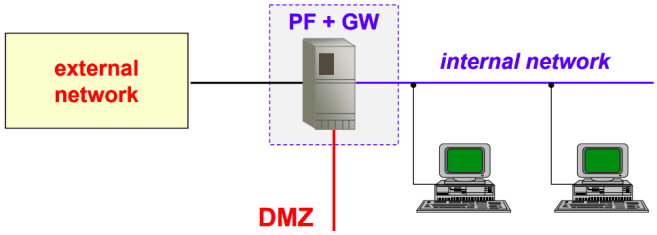
2.8.5 "Screened subnet" architecture (version 2)

What it does?

In this version of the Screened Subnet architecture, the packet filters (**PF**) and gateway (**GW**) are merged into a single device that is called **AKA** ("three-legged firewall").

Three-Legged design:

- One interface connecting to the **External network** (untrusted).
- One interface connecting to the **Internal network** (trusted).
- One interface connecting to the DMZ (where public-facing services reside).

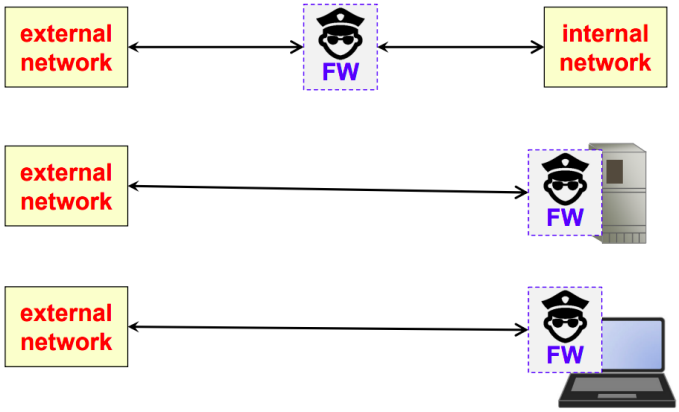


Beware

It's **more simple** than the previous version but presents a **single point of failure**.

2.9 Local/Personal Firewall

This image represents three different firewall configurations: **Network firewall**, **Local firewall** and **Personal firewall**. In the previous pages we have considered the network firewall, now let's start to analyze the Local/Personal firewall.



A local or personal firewall is installed directly on a device to protect it from unauthorized access or attacks. Unlike a normal network firewall, it may limit the PROCESSES that are permitted:

- To open network channels towards other nodes (client behavior).
- To answer network requests (server behavior).

These features make local firewalls essential for containing malware, blocking trojans, or preventing accidental misconfigurations. **However, in order to be effective, the firewall management MUST be separated from system management.**

2.10 Protection offered by a firewall

A firewall is 100% effective only for attacks over/against blocked channels. The other channels require other protection technique:

- **VPNs**
- **intrusion detection systems (IDS)**
- **application-level protections**

2.11 Intrusion Detection System (IDS)

Definition

An **Intrusion Detection System (IDS)**text is a security mechanism designed to:

- identify actors using a system or a network without authorization (and their actions).
- identify authorized actors who violate their privileges.

Hypothesis

- The behavior "pattern" of non-authorized users differs from that of the authorized ones

2.11.1 IDS functional features

- **Passive IDS:** Observes and detects issues but doesn't act to stop them.
  Techniques:
  - Cryptographic checksums to detect changes in files.
  - Pattern matching to identify attack signatures (e.g., malware signatures).
- **Active IDS:** Monitors activity dynamically and reacts when thresholds are exceeded.
  Techniques:
  - Tracks normal behavior to identify anomalies (**"learning"**).
  - Gathers detailed statistics on traffic and actions (**text"monitoring"**).
  - Triggers alerts or responses when unusual patterns occur (**"reaction**).

2.11.2 IDS topological features

- **HIDS (Host-Based IDS):** Monitors individual devices (hosts).
  - Analyzes logs from the OS, services, or applications.
  - Uses built-in OS tools to track internal activity.
- **NIDS (Network-Based IDS):** Monitors traffic at the network level.
  - Uses network traffic monitoring tools.

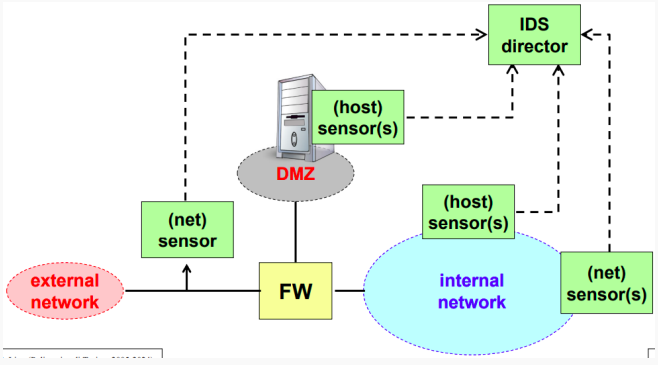
NIDS Architecture

Components:

- **Sensor:** The frontline tool that analyzes traffic or logs looking for suspect patterns. Generates security alerts and can modify access controls (e.g., block traffic).
- **Director:** Coordinates sensors and manages a central database.
- **Message System:** Ensures secure communication among IDS components.

Workflow:

1. Traffic from the external network enters through the firewall.
2. (Net) sensors monitor traffic before it reaches the DMZ or internal network.
3. Within the DMZ, host sensors monitor server activities.
4. Traffic entering the internal network is further monitored by both host and network sensors.
5. All sensors report suspicious activities to the IDS director.
6. The IDS director consolidates this data and raises alerts or takes action if any patterns of intrusion or policy violations are detected.



2.12 Intrusion Prevention System (IPS)

Definition

An **Intrusion Prevention System (IPS)** is an advanced security technology (not a product) that identifies unauthorized activities like an IDS but actively blocks or mitigates them in real time (= IDS + distributed dynamic firewall).
Often, Intrusion Prevention Systems (IPS) are integrated with Intrusion Detection Systems (IDS) into a unified solution called **Intrusion Detection and Prevention System (IDPS)**

Why is it useful to merge IPS with IDS?

IPS already has some of IDS's features, such as detection capabilities. But merging IPS with IDS brings many advantages because:

- **IDS** excels in providing deep visibility into all network activity, including suspicious behavior that might not warrant immediate action.
- **IPS** actively blocks threats, but in doing so, it may not focus as much on monitoring benign-but-suspicious activities.

N.B. IPS requires careful configuration and monitoring to balance its protective capabilities with the risk of blocking innocent traffic.

2.13 Next-Generation Firewall (NGFW)

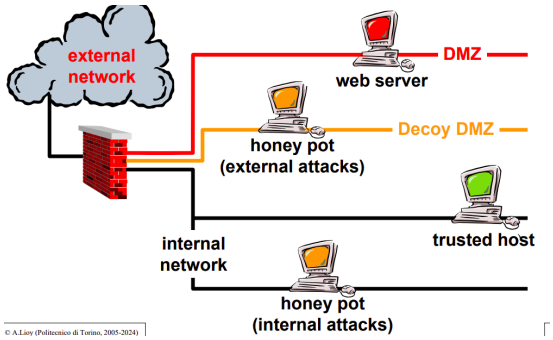
A Next-Generation Firewall (NGFW) enhances traditional firewalls by offering features like **application identification**, regardless of the network port used, and traffic decryption. It can integrate with **authentication systems** such as Active Directory, enabling per-user and per-application security policies. Additionally, NGFWs provide advanced protection by **filtering traffic based on known vulnerabilities**, threats, and malware.

2.14 Unified Threat Management (UTM)

Unified Threat Management (UTM) integrates multiple security features such as firewalls, VPNs, anti-malware, content inspection, and IDPS into a single device. The goal is to simplify security management by consolidating various solutions, reducing complexity and costs, though the specific capabilities depend on the manufacturer.

2.15 Honey pot / Honey net

A honeypot is a decoy system designed to attract attackers and analyze their behavior, while a honeynet is a network of these decoy systems used to study both external and internal attacks. These tools help in identifying and understanding attack methods.



Chapter 3

Security of network applications

3.1 Standard situation

The standard situation for an ordinary network application is very **negative**, since most of the systems implement very **weak authN mechanism** that typically rely on `username` and `password`, which can lead to:

- Password Snooping
- IP Spoofing

Even if **stronger authN mechanism** is implemented, there are still problems with data:

- Data Snooping/Forging
- Shadow Server/MITM
- Replay and Filtering attacks

So to resolve these problems, we can use two different approaches: **Channel Security** and **Message/Data Security**.

3.2 Channel Security

Channel Security implements a secure connection between two nodes. To achieve this, before the start of the communication, the two nodes need to negotiate the `algorithms`, `parameters` and `keys` to protect the whole traffic that will be sent through the communication channel.

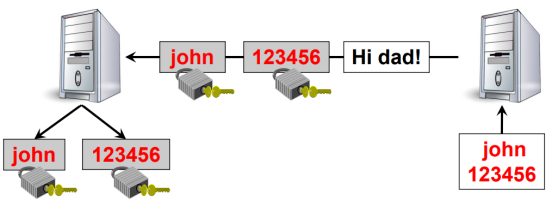
Since all this features are negotiated **before transmitting data**, we ensure:

- **Single** or **mutual authN**
- **Data integrity**
- **Data confidentiality**

Channel Security is very easy to implement because it requires no (or small) modification of applications. Since this features are also negotiated **automatically** we cannot have **non-repudation**.

N.B. The **main issue** with Channel Security is that its security properties are provided **only** during the transit **inside** the communication channel. So data in **not** protected when it's located at the end-user.

3.3 Message/Data security



It applies protection **only** when it's needed. This means that data is **individually** protected by wrapping it into a **secure container**. Data (not the channel) ensures the following security properties:

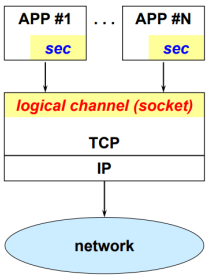
- **Single authN, not mutual** because features are **not negotiated**
- **Data integrity**
- **Data confidentiality**

If protection is applied **voluntary** and **explicity** by the user, we can have **non-repudation**. Message/Data Security requires modification to application.

3.4 Different implementation

`Channel Security` and `Message/Data Security` can be **combined** in order to get the **benefits** of both. These security concepts can be implemented in two different way:

Security Internal to Applications

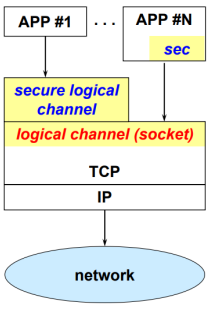


In this approach **each application implements security internally**. The common part is limited to the communication channels (socket). They could be:

- **Possible implementation errors** (implementing security protocols is not simple)
- Does not guarantee interoperability (-> security specification could be different between applications)

N.B. **Communication channels** are the **only** common part between the various application application and are used to exchange information.

Security External to Applications



A **Secure Socket** is created on top of the already existing "normal" socket, effectively creating a **new level** called **Secure Session Level (SSL)**. The Secure Socket implements all the security functions and can be used by any application that wants to communicate in a secure way. Thanks to **SSL** we can:

- Simplify the work of application developer
- Avoid implementation errors

3.5 TLS (Transport Layer Security)

Originally known as **SSL**, **TLS** is a **network/session level protocol** which is capable of creating **secure transport channles** that grants:

- peer authentication
- message confidentiality
- message authentication and integrity
- protection against replay and filtering attacks

It's easily applicable to all protocols based on **TCP**, such as HTTP, SMPT, NNTP, FTP, TELNET, ... (e.g. HTTP (https://....) = 443/TCP).

Beware

HTTPS is not a protocol, it's a combination of HTTP over TLS.

Beware

The current version of TLS is **TLS-1.3** and nowadays everything that uses a version **older than TLS-1.2** is considered **insecure and deprecated**.

```
nsiiops      261/tcp # IIOP Name Service over TLS/SSL
https        443/tcp # http protocol over TLS/SSL
smtps        465/tcp # smtp protocol over TLS/SSL (was ssmtp)
nntps        563/tcp # nntp protocol over TLS/SSL (was snntp)
imap4-ssl    585/tcp # IMAP4+SSL (use 993 instead)
sshell       614/tcp # SSLshell
ldaps        636/tcp # ldap protocol over TLS/SSL (was sldap)
ftps-data    989/tcp # ftp protocol, data, over TLS/SSL
ftps         990/tcp # ftp protocol, control, over TLS/SSL
telnets      992/tcp # telnet protocol over TLS/SSL
imaps        993/tcp # imap4 protocol over TLS/SSL
ircs         994/tcp # irc protocol over TLS/SSL
pop3s        995/tcp # pop3 protocol over TLS/SSL (was spop3)
msft-gc-ssl  3269/tcp # MS Global Catalog with LDAP/SSL
```

Figure 3.1: Official Ports for TLS/SSL Applications

3.5.1 TLS - AuthN and Integrity

Peer AuthN is performed **always at channel setup**:

- The server must be authenticated (mandatory). Sends its public key (X.509 certificate) and responds to an implicit asymmetric challenge.
- The client can authenticate itself (optional): with public key, X.509 certificate and explicit challenge.

For **authN** and **integrity** of **data**, TLS uses:

- A Keyed-Digest (SHA-1 or better).
- An implicit MID to **avoid** Replay and Filtering attacks.

3.5.2 TLS - Confidentiality

Data confidentiality is granted by TLS in the following way:

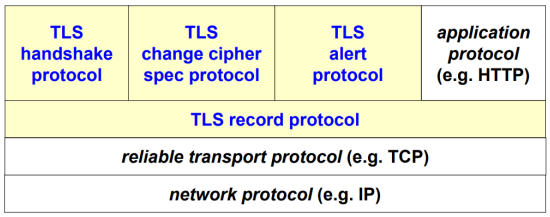
1. The client generates a session key used for symmetric encryption of data (using RC4, 3DES, IDEA, AES, or ChaCha20).
2. The session key is **exchanged** with the server via Asymmetric cryptography (using RSA or DH).

N.B. Authentication is available in TLS 1.2, while is mandatory in TLS 1.3.

3.5.3 TLS Architecture

TLS is positioned **on top** of the transport layer (e.g TCP) and network layer (e.g. IP).

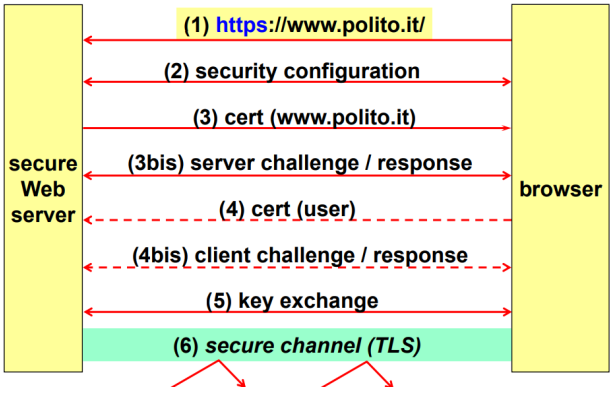
- **Record protocol**: is positioned between TCP and the other TLS protocols and the various application protocol (e.g HTTP).
- **Handshake protocol**
- **Change Cipher Spec Protocol**: used to change the `algorithms`, `parameters` or `keys` on an already established TLS channel.
- **Alert Protocol**: used to send error information on the TLS channel before permanently closing it.



3.5.4 TLS Handshake Protocol

The **Handshake Protocol** is used by TLS to perform **channel setup**, let's see an example:

1. The browser (**client C**) initiates a connection to the **web server** (**S**) by requesting the website (www.polito.it) over HTTPS.
2. **C** and **S** discuss about the security configuration. They should both agree to use the **strongest common algorithms**.
3. The **S** sends its certificate to **C** to prove its identity. The certificate includes the server's public key and the domain name.
   3b) **S** will then respond to an Asymmetric CRA sent by the **C** (This is part of the handshake to confirm (implicitly) the server's identity.)
4. (Optional) **S** may require (explicitly) a client certificate to verify the user's identity.
5. **Key exchange** is performed. During this phase **C** and **S** exchange **master keys** that are linked to the session-ID. Every time a new **TLS Connection** is opened with that **session-ID**, a new connection oriented key must be generated by combining the master key and one of the previously exchanged random numbers.
6. Finally the **Secure TLS Channel** is opened and available to exchange data.

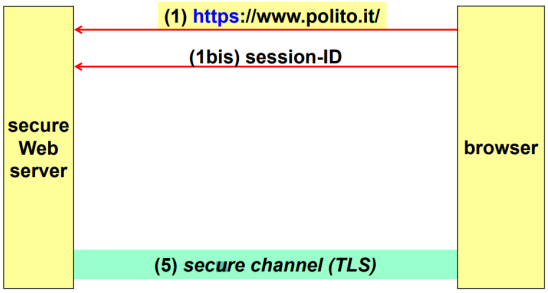


After the Secure TLS Channel has been established, client and server will:

- Negotiate the **Session-ID**
- Exchange **random numbers** to be used for the subsequence generation of keys.

3.5.5 TLS Session ID

A typical web transaction involves the transmission of multiple elements between the client (e.g., a web browser) and the server. To avoid the overhead of negotiating a new session for each element (i.e. Many connections can be part of the same logical session), TLS uses a session ID. **The session ID is a unique identifier that allows the client and server to resume a previous session.** If the client, when opening the TLS connection, sends a valid session-id, the negotiation phase is skipped, and data can be immediately exchanged over the secure channel.
N.B. However, the client must start communicating in encrypted form; otherwise, the server will reject the connection.



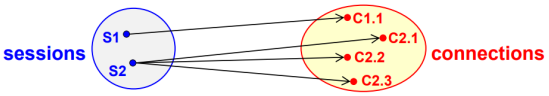
Beware

The server can reject the use of session-id (always or after a time passed after its issuance)

3.5.6 TLS Session & Connections

In TLS we have to distinguish the two:

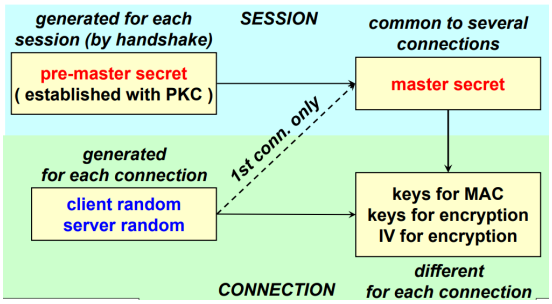
- **TLS Session**: it's a **logical association between client and server** created by the Handshake protocol. It defines a set of cryptographic parameters which are used during communication and can be shared by one or more TLS Connections (1:N).
- **TLS Connection**: it's a **transient** TLS Channel between client and server that is associated to one specific TLS Session (1:1).



3.5.7 Relationship among Keys and Sessions

Every time we create a TLS session: we generate (with PKC) a **pre-master secret**. By generating the client random/server random value, we can mix them with the pre-master secret to generate the **master secret**, which is common to **all** the Connections of the Session. Each time we create a TLS Connection: we generate a **unique** client random/server random value. Combining them with the master secret we create **for this specific connetction and direction**:

- Keys for MAC
- Keys for encryption
- IV for encryption



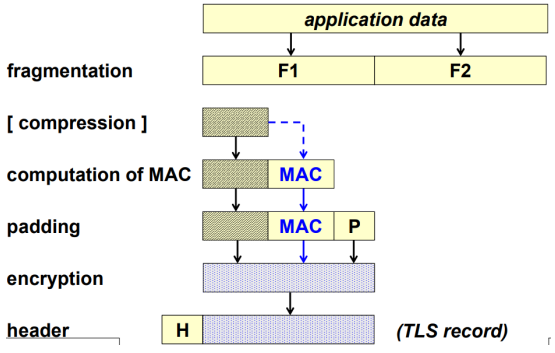
3.5.8 TLS Record Protocol (authenticate-then-encrypt)

The TLS Record Protocol secures application data during transmission.The steps:

1. **Fragmentation**: Divides the data (from the application layer or higher-level TLS protocols) into smaller blocks or "records" that can be transmitted efficiently.
2. The fragmented data is **compressed** (**Optionally**).
3. Ma MAC is computed over the data (→ to verify integrity and authenticity of the data).

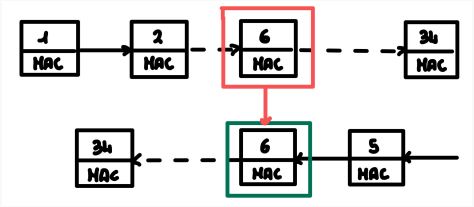
$$MAC = message\_digest(key, seq_number \,||\, type \,||\, version \,||\, length \,||\, fragment)$$

There are two different keys: the sender-write-key and the receiver-write-key. This separation prevents replay attack.

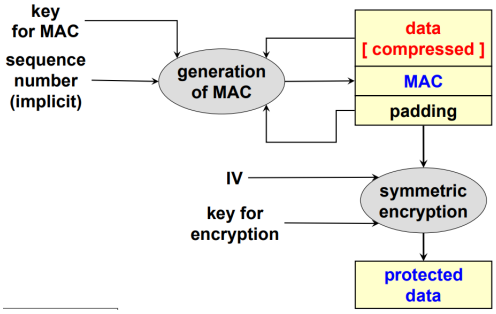


Replay attack

- The sender (Client) sends an encrypted command, like Transfer $100 to the receiver (Server).
- An attacker intercepts this message and replays the exact encrypted data back to the sender (Client).
- Since the sender and receiver share the same key, the sender would decrypt the message and potentially act on it (depending on the system design), causing confusion or security breaches.



4. **Padding** is added to make data encryptable.
5. Data is **encrypted**. Steps:
   - **Generation of MAC**: the (compressed) data is combined with the MAC key and sequence of number as explained before.
   - **Symmetric Encryption**: takes (data || MAC || padding), and, by using the IV and the encryption key, generates the protected data.



6. The encrypted data is packaged into a TLS record (header+payload).

3.5.9 Perfect Forward Secrecy

If a server has a certificate valid for both signature and encryption, it can be used bot for authN (via Signature) and Key Exchange (Asymmetric Encryption of the Session Key). If an attacker copies all the encrypted traffic and later discovers the server's SK, the attacker can decrypt all traffic (**past**, **present** and **future**).
Perfect Forward Secrecy is a technique in which: if a PK/SK pair is compromised, only the **present** and **future** traffic can be decrypted.
This mechanism (implemented in TLS) is achieved through the use of **ephemeral keys** (temporary keys) that are generated for each session and are not stored. For **authenticity**, the one-time key must be signed (the server's SK is only used for signing, granting server authN). However, it cannot have an associated X.509 certificate, as the CA process is slow and often not available online. "Ephemeral" mechanisms are:

- Suitable for DH (Diffie-Helmann).
- Slow for RSA → a compromised would be to re-use N times the same key.

This means that with Perfect Forward Secrecy:

- If the **short term SK** is compromised → the attacker can decrypt **only the present and future** (typically only for the session) traffic.
- If the **long term SK** is compromised → the server can **no longer perform authN** but at least the attacker cannot decrypt any traffic.

Example

ECDHE (Elliptic Curve Diffie-Helmann Ephemeral) is a key exchange protocol that provides PFS.

3.5.10 TLS Downgrade Attack

When the client negotiates with the server which version of TLS to use:

- The client always send (in `Client Hello`) the **highest supported version**.
- The server notifies (in `Server Hello`) the version to be used, which must be the highest in common.

An attacker could send fake server response, forcing a version downgrade until a vulnerable one is reached, and then execute an attack.

Beware

Initial messages, before the setup of the secure channel, are not protected by the security protocol.

This behaviour does not (always) mean that we are under attack, it could be simply a **Insecure downgrade** (→ Some servers do not send the correct response, rather they close the connection. Then the client has no choice to try again with a lower protocol version).

3.6 Virtual Servers and TLS

Problem: the **Virtual Servers problem** is very frequent with **web hosting**: different logical names are associated to the **same IP address** (e.g.: `home.myweb.it=1.2.3.4` and `food.myweb.it=1.2.3.4`).
This problem is easily solved in **HTTP/1.1**: the client uses **Host header** to specify the Logical Name of the server it want to connect to.
But it is a problem for HTTPS, where the domain name is encrypted (because TLS, transport layer security protocol, is activated before HTTP, application layer protocol).

Solutions:

- Collective (wildcard) certificates: the private key is shared by all servers. e.g. `*.myweb.it`. Different browsers may react differently to this solution.
- Certificate with a list of servers in the SAN (subjectAltName) field: The private key is shared among all servers, and the certificate needs to be reissued whenever a server is added or removed.
- The SNI (Server Name Indication) extension: The client sends the domain name in the initial message. The server can then choose the correct certificate. Limited support by browsers and servers.

3.7 ALPN extension (Application-Layer Protocol Negotiation)

The **ALPN extension** was created to **speed up** connection creation by avoiding negotiation `round-trips`. ALPN works in the following way:

- (ClientHello) ALPN = true + list of supported application protocols;
- (ServerHello) ALPN = true + selected application protocols;

N.B. It is particularly important to negotiate HTTP/2 and QUIC, since (for example) Chrome and Firefox support HTTP/2 only over TLS.

> ALPN is useful also for those servers that use different certificates for the different application protocols.

3.8 TLS Fallback - Signaling Cipher Suite Value

The **SCSV estension** is used to prevent protocol downgrade attacks. SCSV works in the following way:

- When a connection to a server is closed during the Handshake protocol negotiation phase, the client (upon opening downgraded connection) should send to the server a new (dummy) ciphersuite called **TLS_FALLBACK_SCSV**;
- Upon receiving TLS_FALLBACK_SCSV and a version lower then the highest one supported, the server must send a new Fatal Alert value: "inappropriate_fallback".

  Immediately after sending this value, the **TLS channel is closed** → the client should retry with its highest protocol version;

N.B. However, many servers do **not** support SCSV, but at least **most** of them have fixed this bad behaviour → browsers can **disable insecure downgrade**.

3.9 DTLS (Datagram Transport Layer Security)

DTLS tries to apply the concepts of TLS to **datagram security** (e.g. UDP). It does not offer the same security properties as TLS and is in competition with IPsec and Application Security.
For example, if we use **SIP** (**Session Initiation Protocol**) we could implement security with:

- IPsec
- TLS (SIP over TCP)
- DTLS (SIP over UDP)
- Secure SIP

3.10 HTTP Security

In**HTTP/1.0** some **bad security mechanisms** are defined:

- **Address-based Access Control**: performed by the server on the IP address of client.
- **Password-based Access Control**: **username** and **password** are sent encoded (base64).

Both of them are **highly insecure** since HTTP assumes we are using a Secure Channel.

> From HTTP/1.1 **digest authN** based on Symmetric CRA was introduce.

3.10.1 HTTP Basic Authentication

1. After having created the channel using **HTTP/1.0**, the server automatically responds with an "authentication failed" message but does **not** close the channel, because the client (maybe) did not know that authN was needed → the servers sends an **authN request** to the client;
2. The client's browser will open a pop-up that asks the client to enter its credentials in order to perform authN.
3. The client submits its credentials encoded in Base64 (**not encrypted → no secure**).

```
#encode
echo username:password | openssl enc -a

#decode
echo YWRtaW46cGFzczEyMw== | openssl enc -a -d
```

4. The server can **verify** the credentials and (eventually) send the information requested by the client.

```
C>S: GET /path/to/protected/page HTTP/1.0
S>C: HTTP/1.0 401 Unauthorized - authentication failed
    WWW-Authenticate: Basic realm="POLITO - didattica"
C>S: Basic czEyMzQ1NjpTZWdyZXRpc3NpbWE=
S>C: HTTP/1.0 200 OK
    Server: Apache/1.3
    Content-type: text/html
    <html> ... protected page ... </html>
```

3.10.2 HTTP Digest Authentication

The keyed-digest introduced in **HTTP/1.1** is computed in the following way:

$$HA1 = \text{md5 (A1)} = \text{md5(user":"realm":"pwd)}$$
$$HA2 = \text{md5 (A2)} = \text{md5(method":"URI)}$$
$$response = \text{md5 (HA1":"nonce":"HA2)}$$

- The sever may even insert an **"opaque"** field to send some state information to the client.
- The server uses a nonce to prevent replay attacks.

1. After having created the channel using **HTPP/1.1**, the server **automatically** responds with a `"Authentication failed"` message but **not** close the channel, because the client maybe did **not** know that authN was needed, so the server sends an **authN request** to the client containing the **nonce** and the **opaque** values encoded in Base64.
2. The clint's browser will open a pop-up that asks the client to perform authN → the client computes the keyed-digest and sends it, together with the **opaque**, to the server.
3. The server can then **verify** the credentials and (eventually) send the information request by the client.

```
C>S: GET /private/index.html HTTP/1.1
S>C: HTTP/1.0 401 Unauthorized - authentication failed
    WWW-Authenticate: Digest realm="POLITO",
    nonce = "dcd98b7102dd2f0e8b11d0f600bfb0c093",
    opaque = "5ccc069c403ebaf9f0171e9517f40e41"
C>S: Authorization: Digest username="lioy",
    realm="POLITO",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    uri="/private/index.html",
    response="4c9d010fac372e048297160bff991883",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
S>C: HTTP/1.0 200 OK
    Server: NCSA/1.3
    Content-type: text/html
    <html> ... protected page ... </html>
```

3.10.3 HTTP & TLS/SSL

We can use both HTTP and TLS/SSL by activating them in one of the following orders:

- **TLS then HTTP**: the client connects to the server using TLS and then sends the HTTP request (→ Content inspection is not possible!).
- **HTTP then TLS**: the client connect to the server using HTTP and then upgrades the connection to TLS (→ We see the HTTP methods until the upgrade!).

Beware

They are **not** equivalent as the activation order has an impact over `applications`, `firewall` and IDS.

In general the most used approach is "TLS then <proto>" (e.g. HTTPS).

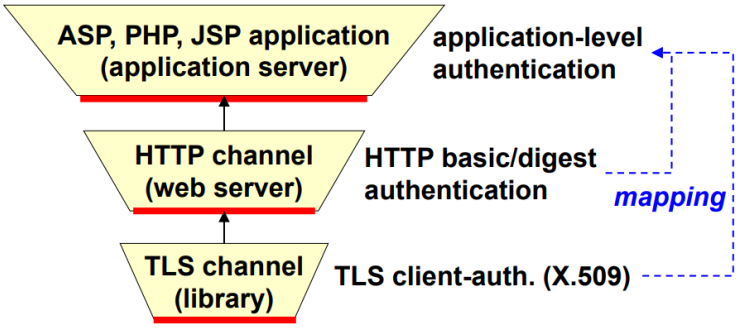
TSL AuthN at the Application Layer

Via client authentication it's possible to identify the user that opened the channel (without asking for his username and password).
The identification process of a client can be done at the application level, but it is not a standard feature of TLS.

AuthN in Web applications

The **earlier** access control is performed, the **smaller** is the attack surface.

Moreover, there is no need to repeat authentication across different parts of the application if the user's identity is securely propagated throughout the application. Some web servers support a (semi-)automatic mapping between the credential extracted from the X.509 certificate and the users of the HTTP service and/or the OS.



Web Form Security

Technically speaking, the security of the page containing the form is not important (e.g. http://www.ecomm.it/login.html).
The actual security depends on the URI of the method used to send the data to the server (e.g <form . . . action="https://www...>).
Yet, as a form of psychological help for the users, we should also look to implement security in the page containing the form

3.11 HTTP Strict Transport Security (HSTS)

The **HSTS extension** is used by HTTP server to declare that its interaction with **User Agent** (**UA** → e.g. browser, etc...) **must only be via HTTPS**, effectively preventing Protocol Downgrade and Cookie Hijacking attacks.
HSTS is applied **only** upon receiving a **valid HTTPS response**. Its validity is renewed at every access and it may:

- Include subdomains (recommended).
- Be pre-loaded into the servers.

Syntax

The syntax is:

```
Strict-Transport-Security:
   max-age = <expire-time-in-seconds>
   [ ; includeSubDomains ]
   [ ; preload ]
```

Example:

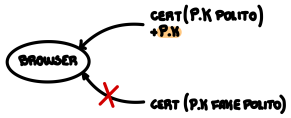
```
$ curl -s -D- https://www.paypal.com/ | fgrep -i strict
strict-transport-security: max-age=63072000
$ curl -s -D- https://accounts.google.com/ | grep -i
strict
strict-transport-security: max-age=31536000;
includeSubDomains
```

Upon receiving an answer from the server that will contain the **HSTS Header**, inside there will be specified:

- The expiration data in seconds.
- (Optionally) the subdomains.
- (Optionally) the option of being added to the client's pre-load list.

3.11.1 HTTP Public Key Pinning (HPKT)

The **HPKP extension** is used by HTTP servers to declare that it is using a certain **PK** by specifying the Digest of the PK and/or one or more CAs in its chain (except the root CA). The User Agent (UA) caches this key and will refuse to connect to a site presenting a different key.



- HPKP is a **TOFU** (**Trust On First USe**) technique.
- HPKP is dangerous if we lose control of the key.
- HPKP can cause problems with key updates → always include a backup key.

Syntax

The syntax is:

```
Public-Key-Pins:
   pin-sha256 = " <base64-sha256-of-public-key> ";
   max-age = <expireTime-in-seconds>
   [; includeSubDomains]
   [; report-uri = " <reportURI> "]
```

```
Public-Key-Pins-Report-Only:
   pin-sha256 = " <base64-sha256-of-public-key> ";
   max-age = <expireTime-in-seconds>
   [; includeSubDomains]
   [; report-uri = " <reportURI> "]
```

Header includes:

- The digest of the PK computed with SHA-256.
- The expiration date in seconds.
- (Optionally) the subdomains.
- (Optionally) the report URI that is used to report violations.

N.B. HPKP can work in **enforcing** or **report-only mode**.

Example

```
$ $ curl -s -D - https://scotthelme.co.uk/
    | fgrep -i public-key
public-key-pins:
pin-sha256="9dNiZZueNZmyaf3pTkXxDgOzLkjKvI+Nza0ACF5IDwg=";
pin-sha256="X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="V+J+7lHvE6X0pqGKVqLtxuvk+0f+xowyr3obtq8tbSw=";
pin-sha256="9lBW+k9EF6yyG9413/fPiHhQy5Ok4UI5sBpBTuOaa/U=";
pin-sha256="ipMu2Xu72A086/35thucbjLfrPaSjuw4HIjSWsxqkb8=";
pin-sha256="+5JdLySIa9rS6xJM+2KHN9CatGKln78GjnDpf4WmI3g=";
pin-sha256="MWfCxyqG2b5RBmYFQuLllhQvYZ3mjZghXTRn9BL9q10=";
includeSubDomains; max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
   enforce"
public-key-pins-report-only:
pin-sha256="X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="Vjs8r4z+80wjNcr1YKepWQboSIRi63WsWXhIMN+eWys=";
pin-sha256="IQBnNBEiFuhj+8x6X8XLgh01V9Ic5/V3IRQLNFFc7v4=";
max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
   reportOnly"
```

The **HPKP**