Anatomy of a Modern Web3 Infrastructure

How Web3 combines decentralization and centralized components in hybrid architectures.

Alessandro Magnosi

https://github.com/TMPest -uniroma2/web3sec

What is Web3?

Decentralized Evolution

Internet evolves toward decentralized models. Users gain more control over their digital presence.

Direct Ownership

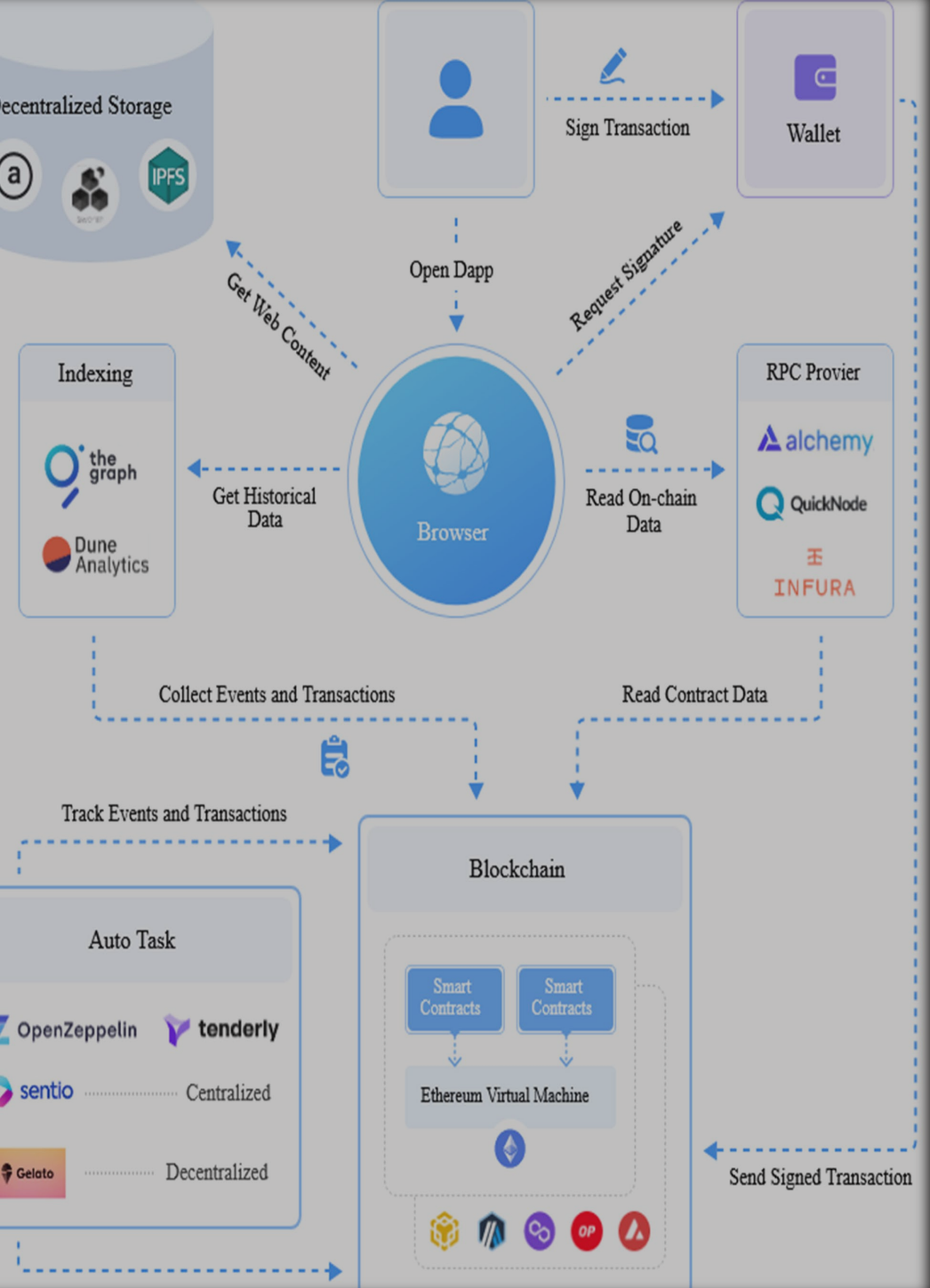
Users control digital assets and identities. Blockchain enables verifiable ownership records.

Trustless Logic

Smart contracts replace trusted intermediaries. Code execution happens transparently on-chain.

Interoperability

Emphasis on transparency and censorship resistance. Systems communicate across chains.



Core Components of a dApp

Smart Contract

On-chain executable code, immutable by design. Written in Solidity for EVM compatibility.

Frontend

React/Vue applications that interact with the blockchain via wallet connections.

Wallet

Tools for key management and digital signatures. Examples include MetaMask and Ledger.

Blockchain

Decentralized ledger that records all transactions and smart contract executions.

Backend

Manage off-chain data storage, complex computations, and integration with external APIs.

Token/Currencies

Digital assets that can represent ownership, access rights, or value within a dApp.

Common Layers in a Web3 Stack

Layer 1/2

Ethereum, Polygon, Arbitrum for contract execution. Base layer for transactions.

Indexing Layer

The Graph enables efficient on-chain data querying. Creates accessible subgraphs.

Storage Layer

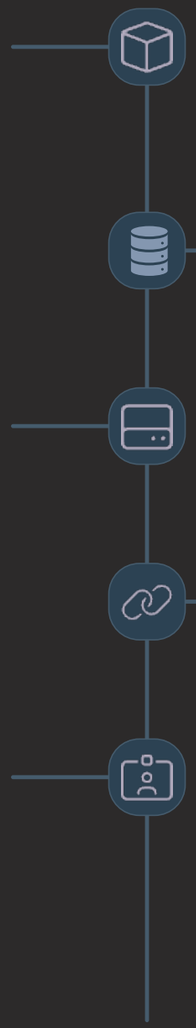
IPFS, Arweave for metadata, files, and frontend hosting. Distributed content networks.

Oracle Layer

Chainlink, Pyth inject off-chain data into smart contracts. Bridge to real world.

Identity Layer

ENS, SIWE provide decentralized naming and authentication. Human-readable addresses.



Hybrid Architecture: The Reality

Why dApps are hybrid:

- High gas costs for complex logic
- Blockchain's latency issues
- On-chain storage is expensive
- Need for responsive UX

Real-world Examples:

- OpenSea: Web2 frontend, NFT metadata on IPFS
- Uniswap: React frontend hosted via Cloudflare
- Most DeFi protocols use similar patterns

Common Hybrid Patterns

Centralized Backend

Used for push notifications, analytics, and rate limiting. Improves user experience.

Mixed Authentication

OAuth2 (Google/Facebook) combined with Sign-In with Ethereum (SIWE). Balances convenience with security.

Hybrid Frontend

Hosted on Web2 platforms, backed up on IPFS/Arweave. Ensures availability and speed.

External RPC Providers

Centralized nodes like Infura serve as critical dependencies. Gateway to blockchain.

Web Security Lab: PKI and Smart Contract Security

Welcome to our comprehensive security lab focused on Public Key Infrastructure (PKI) and smart contract security. This hands-on experience will bridge traditional Web2 security concepts with emerging Web3 paradigms, providing you with practical skills to identify, analyze, and mitigate security vulnerabilities across both environments.

Throughout this lab, you'll explore the fundamentals of PKI, execute common attack simulations, analyze smart contract code for vulnerabilities, apply threat modeling methodologies, and investigate anomaly detection techniques for distributed attacks.

Alessandro Magnosi

Learning Objectives

Understand PKI and Its Role

Explain how public key infrastructure enables trust, authentication, and encryption across Web2 (HTTPS) and Web3 (wallet-based authentication and smart contract signing).

Comprehend Common Attacks

Execute practical attack simulations in Web2 (XSS, CSRF, SQLi) and analyze vulnerabilities in Web3 contexts (reentrancy, signature replay).

Analyze Smart Contract Code

Review and identify security flaws in Hyperledger chaincode and Solidity contracts, applying secure coding practices and tools.

Apply Threat Modeling

Use methodologies like STRIDE or DFDs to map components and threats, highlighting attack surfaces in decentralized applications.

Prerequisites for a Secure Web System

Confidentiality & Integrity

Ensuring sensitive information is accessible only to authorized users and guaranteeing that data remains accurate and untampered throughout its lifecycle.

Availability & Authentication

Maintaining system accessibility for authorized users when needed while verifying the identity of all parties to prevent unauthorized access.

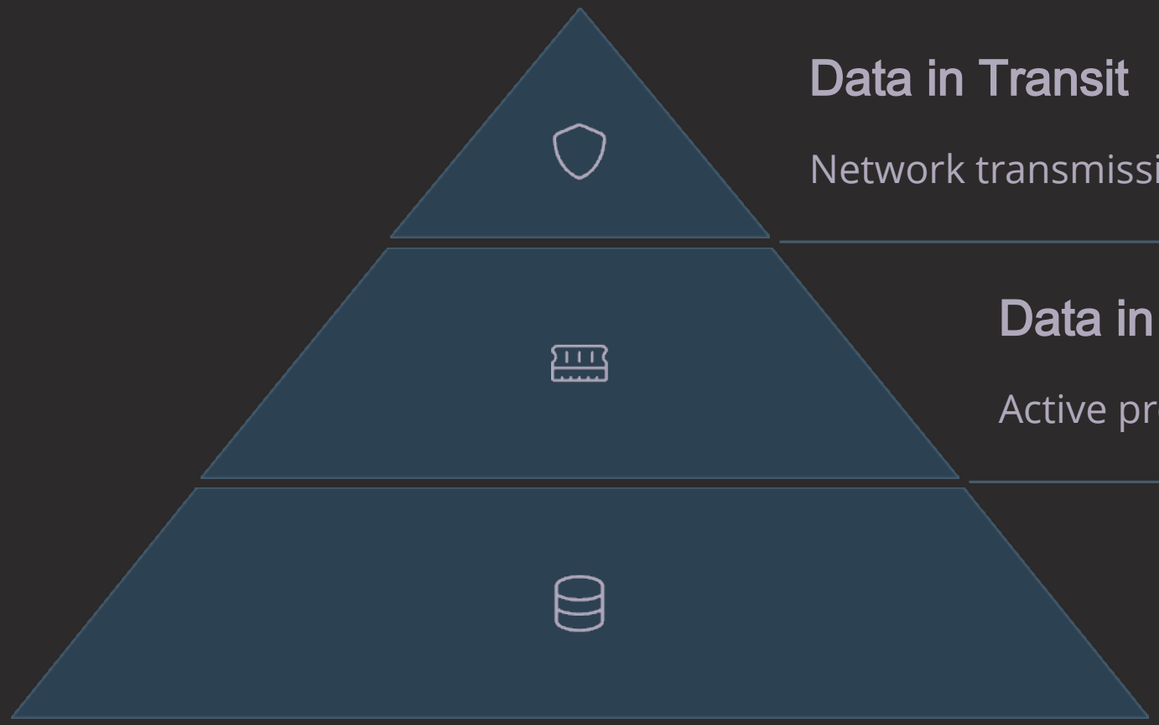
Encryption & Key Management

Implementing strong SSL/TLS protocols while minimizing the number of cryptographic keys to simplify management and reduce security risks.

Responsibility & Non -repudiation

Dividing responsibilities among different roles to minimize risk while providing proof of data origin and integrity to prevent denial of authenticity.

Data Security Fundamentals



Data in Transit

Network transmission security

Data in Use

Active processing protection

Data at Rest

Storage security measures

Security professionals must address three distinct states of data vulnerability. Data at rest, typically stored in databases and file systems, requires encryption solutions like AES to protect against unauthorized access. Data in use faces sophisticated threats like side-channel attacks, often mitigated through Trusted Execution Environments such as Intel SGX.

Data in transit represents perhaps the most visible security challenge, as information traverses multiple network points where interception is possible. This vulnerability is where PKI plays a critical role in establishing secure communication channels through encryption and authentication mechanisms.

Symmetric Encryption Approach

How It Works

Symmetric encryption uses the same key for both encryption and decryption processes. This shared key must be known by both the sender and recipient to successfully encrypt and decrypt messages.

The method is computationally efficient, making it ideal for encrypting large volumes of data quickly without significant processing overhead.

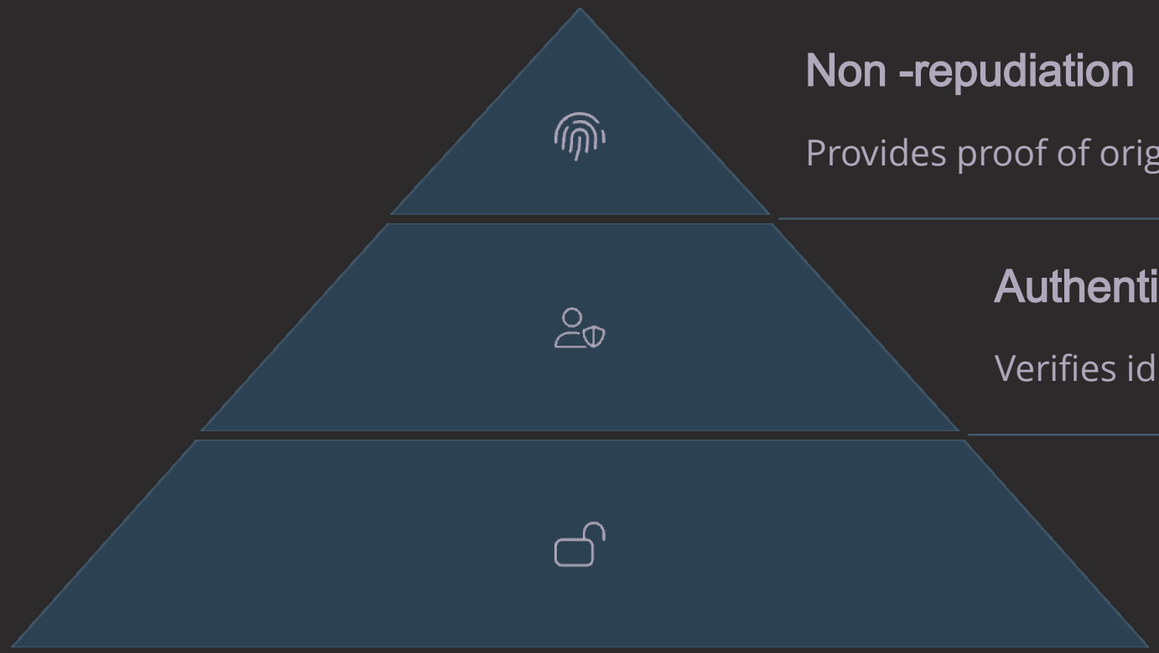
Strengths

- Addresses confidentiality effectively
- Maintains data integrity
- Provides robust SSL implementation
- Offers superior processing speed

Limitations

- Requires secure key exchange
- Lacks inherent authentication
- Doesn't support non-repudiation
- Limited segregation of responsibilities

Asymmetric Encryption Approach



Non -repudiation

Provides proof of origin and authenticity

Authentication

Verifies identity through key ownership

Confidentiality & Integrity

Protects data with public/private key pairs

Asymmetric encryption uses a pair of mathematically related keys - public and private. The public key can be freely shared, while the private key remains secret. Data encrypted with the public key can only be decrypted with the corresponding private key.

While this approach effectively addresses confidentiality, integrity, authentication, and non-repudiation, it struggles with key management complexity and computational demands that affect speed. The need for numerous key pairs creates additional overhead compared to symmetric methods.

Public Key Infrastructure (PKI)

Certificate Issuance

Trusted authorities verify and certify identities

Non -repudiation

Ensures accountability through digital signatures

Key Management

Simplified distribution without pre-established secure channels

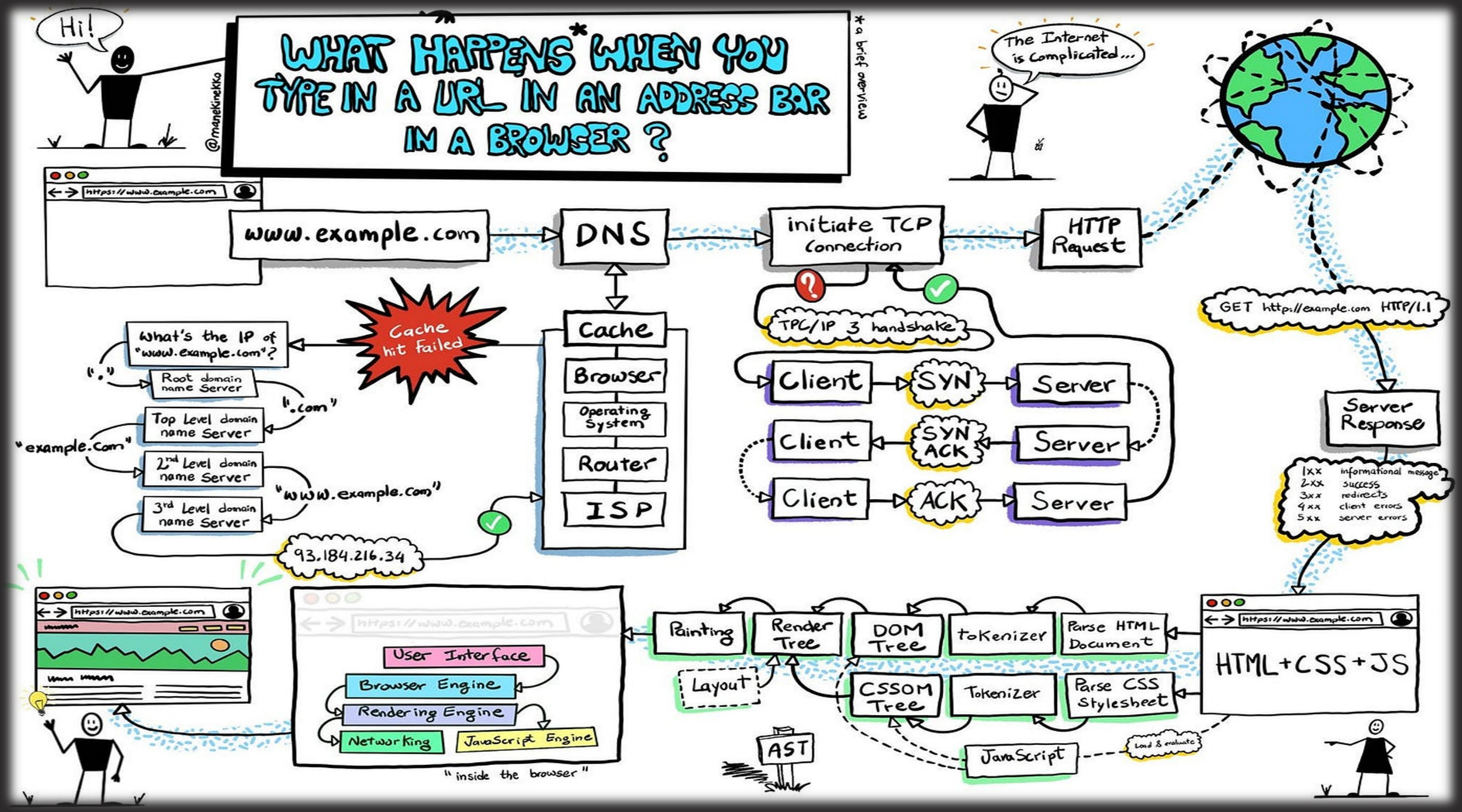
Authentication

Verifies parties through digital certificates

PKI overcomes the limitations of both symmetric and asymmetric encryption by introducing trusted certificate authorities that validate the association between public keys and identities. This comprehensive approach addresses all security prerequisites while providing an efficient framework for key management.

By leveraging certificate authorities and registration authorities, PKI effectively segregates responsibilities and establishes a chain of trust that eliminates the need for pre-established secure channels for key exchange.

Standard Web Navigation



Domain Name System (DNS): The Internet's Directory

DNS is the critical system that translates human-readable domain names (like example.com) into machine-readable IP addresses (like 192.0.2.1) that computers use to identify each other on the network.

Hierarchical Structure

DNS operates as a distributed hierarchical database with different levels of domain authorities (root, top-level domains, authoritative nameservers) working together to resolve queries.

Resolution Process

When a user enters a URL, their device queries a recursive resolver, which then navigates the DNS hierarchy to find the correct IP address, often using cached results to improve performance.

Critical for Web Security

DNS forms a foundational layer of web security, as it determines which servers receive connection requests. Compromised DNS can lead to users being directed to malicious sites without their knowledge.

Without DNS, users would need to memorize numeric IP addresses for every website they visit, making the internet practically unusable for most purposes. This translation service is so fundamental that it's often called the "phone book of the internet."

Evolution of Secure Communication

TCP/IP Stack

The fundamental networking protocol stack lacks inherent security features, offering no built-in confidentiality or authentication mechanisms. Data packets traverse multiple intermediaries, creating numerous opportunities for man-in-the-middle attacks.

HTTP Protocol

As a stateless application layer protocol, HTTP facilitates web communication but was designed without security considerations. All data travels in plaintext, and there's no guarantee of server identity, making it vulnerable to various attacks.

HTTPS/TLS

The addition of Transport Layer Security transformed web security by encrypting communications and verifying server identity through certificates, establishing the foundation for trusted online interactions.

The evolution of secure communication protocols reflects our growing understanding of network security threats. Each advancement has built upon previous technologies to address emerging vulnerabilities while maintaining compatibility with existing systems.

Transport Layer Security (TLS)

Client Hello

Client initiates with supported cipher suites

Server Certificate

Server responds with X.509 certificate

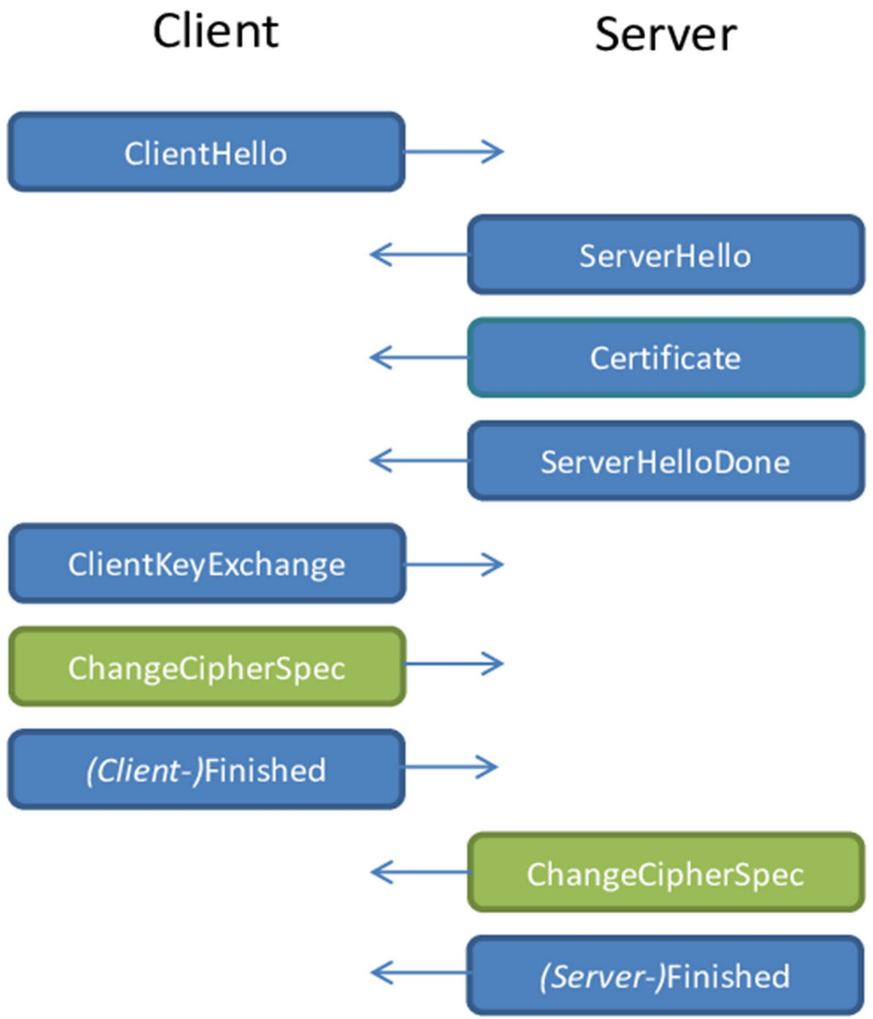
Key Exchange

Asymmetric cryptography establishes symmetric key

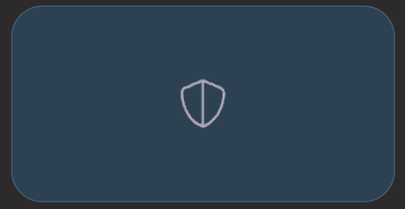
Secure Communication

Encrypted data flows with integrity checks

TLS transforms standard HTTP into secure HTTPS by providing three critical security properties: confidentiality through symmetric encryption, integrity via message authentication codes (MACs), and authentication through X.509 certificates. This layered approach ensures that data remains protected throughout transmission.

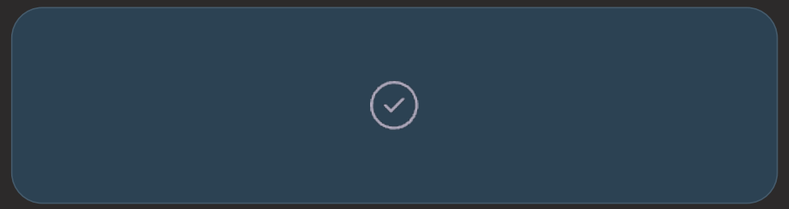


HTTPS Workflow: Secure Data Transmission



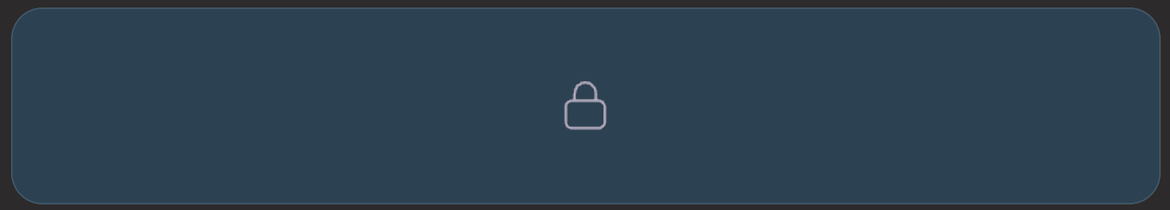
Symmetric Encryption

Fast, efficient encryption of all data using session keys



Data Integrity

Protection against tampering using MAC or AEAD



Secure Transmission

Protection from eavesdropping and man-in-the-middle attacks

Once the TLS handshake completes and session keys are established, all subsequent data transmitted between client and server is protected using symmetric encryption. This ensures both confidentiality and efficiency, as symmetric encryption is well-suited for high-volume data transfer.

Mechanisms like Message Authentication Code (MAC) or Authenticated Encryption with Associated Data (AEAD) work alongside encryption to verify data integrity and authenticity. When the session ends, keys are discarded, ensuring each session has unique encryption that cannot be reused.

SSL/TLS Certificate Validation

Certificate Validity Checks

Browsers verify that certificates are within their validity period and match the domain being accessed. They also confirm the certificate was issued for server authentication (OID 1.3.6.1.5.5.7.3.1).

Trust Verification

Browsers check if the certificate was issued by a trusted Certificate Authority by comparing against pre-installed CA certificates. The entire certificate chain is validated up to the root CA.

Revocation & Signature Checks

Browsers verify the certificate hasn't been revoked using CRL or OCSP protocols. Digital signatures are validated using the issuer's public key to ensure the certificate hasn't been tampered with.

Cryptographic Verification

The browser uses the public key in the server's certificate to encrypt information that only the legitimate server with the corresponding private key can decrypt, proving certificate ownership.

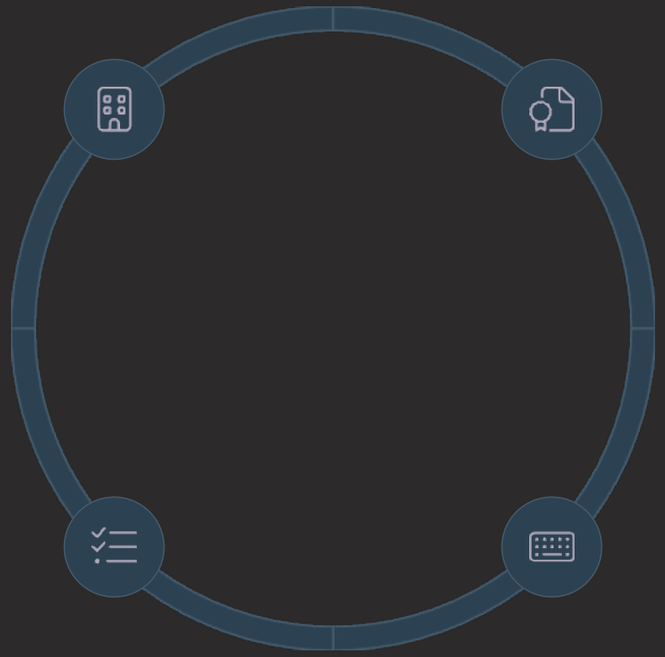
Public Key Infrastructure (PKI)

Certificate Authorities

Trusted entities that issue and verify digital certificates within the PKI ecosystem

CRLs/OCSP

Mechanisms for checking certificate validity and managing revocation



X.509 Certificates

Digital documents binding public keys to entities, containing identity information and digital signatures

Key Management

Processes for generating, storing, distributing, and revoking cryptographic keys

PKI forms the core trust framework for secure digital communications, enabling authentication by ensuring that parties in a transaction are who they claim to be. This infrastructure facilitates secure encryption key distribution without requiring pre-shared secrets, a critical requirement for wide-scale secure communications.

The scalability of PKI through hierarchical or decentralized trust models makes it adaptable to various deployment scenarios, from enterprise environments to global internet security. These properties have made PKI the foundation of trust for virtually all secure online transactions.

Lab Time – Validate a URL

A systematic approach to verify domain security and certificate validity

DNS Resolution Check

Use nslookup to confirm proper DNS name resolution and verify the domain points to the expected IP address.

```
nslookup example.com
```

Port Scanning

Employ nmap to check open ports and available services, focusing on common web ports (80/HTTP, 443/HTTPS).

```
nmap -p 80,443
example.com
```

SSL/TLS Configuration Analysis

Use sslscan to evaluate supported cipher suites, protocols, and potential TLS vulnerabilities.

```
sslscan
example.com:443
```

Certificate Validation

Extract and verify the SSL certificate using openssl to confirm issuer validity, expiration date, and domain match.

```
openssl s_client -
connect
example.com:443 -
showcerts
```

Certificate Authorities

Certificate Authorities (CAs) are trusted entities that issue digital certificates, connecting cryptographic keys with verified identities. These certificates are essential for authenticating web content and establishing secure connections.

CAs serve as trust anchors in the Public Key Infrastructure (PKI), verifying the authenticity of domains and organizations. When a CA issues a certificate, users can be confident they're connecting to a legitimate website rather than a fraudulent one.

Commercial CA Solutions

Paid services that handle the entire certificate lifecycle management process.

Pros:

- Extended validation certificates available
- Dedicated support services
- Warranty protection
- Longer certificate validity periods

Cons:

- Significant recurring costs
- Potential vendor lock-in
- Varying levels of automation

Let's Encrypt

Free, automated CA requiring Certificate Signing Request (CSR) generation and domain validation.

Pros:

- No financial cost
- ACME protocol supports automation
- Widely trusted by browsers
- Promotes HTTPS adoption

Cons:

- 90-day certificate validity
- No extended validation options
- Limited support channels
- Requires renewal automation

Custom Internal CA

Self-operated CA infrastructure for issuing certificates within an organization.

Pros:

- Complete control over issuance policies
- No external dependencies
- Unlimited certificates at no cost
- Customizable validity periods

Cons:

- Not publicly trusted without complex setup
- Requires significant technical expertise
- High operational security requirements
- Added management overhead

The Interdependence of PKI and DNS

Public Key Infrastructure (PKI) and Domain Name System (DNS) are fundamentally coupled, creating a circular dependency that forms the backbone of secure web communications.

Certificate Validation Relies on DNS

SSL/TLS certificates contain domain names that must be verified against DNS records. Without reliable DNS, certificate validation fails as browsers cannot confirm the certificate matches the intended domain.

DNS Security Depends on PKI

DNSSEC (DNS Security Extensions) uses digital signatures and public key cryptography from PKI to authenticate DNS responses, protecting against DNS poisoning and spoofing attacks.

Certificate Authentication Requirements

Certificate Authorities must verify domain ownership before issuing certificates, typically through DNS-based validation methods like adding specific TXT records that prove administrative control.

Common Attack Surface

Compromising either system affects the other: DNS hijacking can redirect to fraudulent sites with valid certificates, while compromised CAs can issue certificates for domains they don't control.

This mutual reliance creates both strength and vulnerability: while providing multiple layers of verification, it also means that security issues in either system can undermine the entire trust framework of secure web communications.

PKI in Web2 vs Web3

Web2 (Traditional Internet)

- Centralized root CAs trusted by browsers
- Hierarchical trust model
- Single points of failure
- Certificate transparency logs
- Vendor-controlled trust stores

Web3 (Blockchain Infrastructure)

- Decentralized trust mechanisms
- Self-sovereign identity models
- Smart contract verification
- Cryptographic wallet authentication
- Decentralized Identifiers (DIDs)

Traditional Web2 internet security relies on a centralized PKI model where root Certificate Authorities are trusted by operating system and browser vendors. This approach creates potential single points of failure, as demonstrated by incidents like the DigiNotar breach, though certificate transparency and monitoring have improved auditability.

In contrast, Web3 infrastructure marks a paradigm shift toward decentralized trust. While still relying on asymmetric cryptography, Web3 systems often bypass traditional CAs in favor of blockchain-native verification mechanisms. Technologies like ENS/DNS with IPFS, DIDs, and zero-knowledge proofs are emerging as alternatives to hierarchical trust models.

Creating SSL Certificates

The process of creating SSL certificates involves several key steps to ensure proper security and authentication. These certificates are essential for establishing encrypted connections between clients and servers.

Generate Key Pair

Create public and private keys using cryptographic algorithms like RSA.

Create CSR

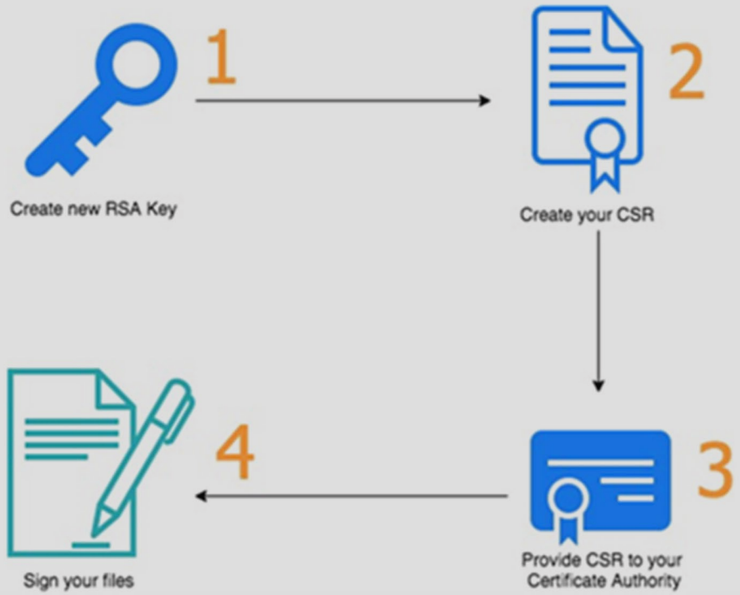
Generate a Certificate Signing Request containing domain information.

Certificate Issuance

Submit CSR to a Certificate Authority for validation and signing.

Installation

Install the signed certificate on the web server to enable HTTPS.



Lab Time – Creating a Self -Signed Certificate with OpenSSL

A step-by-step guide to generate your own certificate for testing and development

Generate a Private Key

Create a 2048-bit RSA private key that will be used to sign your certificate.

```
openssl genrsa -out private.key 2048
```

Create a Certificate Signing Request (CSR)

Generate a CSR containing your identity information. You'll be prompted to enter details like Country, State, Organization, and Common Name (domain).

```
openssl req -new -key private.key -out request.csr
```

Generate the Self -Signed Certificate

Sign the CSR with your own private key to create a self-signed certificate valid for 365 days.

```
openssl x509 -req -days 365 -in request.csr -signkey private.key -out certificate.crt
```

Verify Certificate Content

Inspect the certificate to ensure all information is correct.

```
openssl x509 -text -noout -in certificate.crt
```

Note: Self-signed certificates will trigger browser security warnings as they aren't issued by a trusted Certificate Authority. Use for development and testing environments only.

Linux Certificate Generation Script

Our Linux script automates the process of generating X.509 certificates for both Certificate Authorities (CAs) and server/client entities. Written in Bash, it leverages OpenSSL to handle key generation, Certificate Signing Request (CSR) creation, and certificate signing.

Script Components

- Initial configuration and command-line parsing
- Dynamic OpenSSL configuration generation
- Certificate generation functions
- Main logic for execution flow

Key Features

- Customizable certificate details via command-line flags
- Support for both CA and server certificate generation
- Dynamic configuration based on certificate type
- Secure passphrase handling for private keys

Security Considerations

- CA passphrase stored temporarily and should be deleted
- 4096-bit RSA keys for CAs, 2048-bit for servers
- SHA-256 hashing algorithm for signatures
- Proper certificate extensions for intended use

Windows Certificate Generation Script

The Windows PowerShell script provides similar functionality to the Linux version but is tailored for Windows environments. It interacts directly with the Windows certificate store, leveraging PowerShell cmdlets for certificate management.

Key Differences from Linux

Unlike the Linux script that uses OpenSSL and stores certificates as files, the Windows script:

- Interacts with the Windows certificate store
- Uses PowerShell cmdlets like New-SelfSignedCertificate
- Requires administrator privileges
- Includes functions for certificate store cleanup

Script Structure

The PowerShell script is organized into several key functions:

- SSInvoke-PreCheck: Validates administrator privileges
- SSInvoke-Clean: Removes certificates from the store
- SSInvoke-GenerateCACertificate: Creates CA certificates
- SSInvoke-GenerateCertificate: Creates server certificates

- SSInvoke-Generator: Main wrapper function

Certificate Generation Example: CA Certificate

Creating a Certificate Authority (CA) is the first step in establishing your PKI. The CA certificate serves as the root of trust for all server and client certificates you'll generate. Here's how to create a CA certificate using our scripts.

Linux Command

Run the bash script with the 'ca' type parameter and required certificate details:

```
./linux.sh -t 'ca' \
-e 'admin@test.com' \
-d 'myca.com' \
-p 'MyCASecurePass123' \
-c 'IT' -s 'Rome' -n 'My CA' \
-l 'Rome' -o 'SSI' -u 'IT Dep'
```

Windows Command

Execute the PowerShell function with similar parameters:

```
ipmo ‘.\windows.ps1’; SSInvoke-Generator
-Name "My CA“ -DNSName "myca.com" `
-Passphrase "MyCASecurePass123" `
-OutputDir "C:\Certs" -Type "CA" `
-Subject "CN=My CA, OU=IT Dep, O=SSI, `
L=Rome, C=IT, E=admin@test.com"
```

Lab Time – Configure camanager

```
mkdir $HOME/.pyenv
virtualenv $HOME/.pyenv/camanager
source $HOME/.pyenv/camanager/bin/activate
pip install camanager@git+https://github.com/klezVirus/camanager
```

```
[CA]
COUNTRY_NAME = IT
STATE_OR_PROVINCE_NAME = Rome
LOCALITY_NAME = Rome
ORGANIZATION_NAME = TMPEST ORG
COMMON_NAME = TMPEST-ROOT-CA

[IntermediateCA]
COUNTRY_NAME = IT
STATE_OR_PROVINCE_NAME = Rome
LOCALITY_NAME = Rome
ORGANIZATION_NAME = TMPEST ORG
COMMON_NAME = TMPSET-INT-CA

[example.domain.com]
COUNTRY_NAME = IT
STATE_OR_PROVINCE_NAME = Rome
LOCALITY_NAME = Rome
ORGANIZATION_NAME = TMPEST ORG
COMMON_NAME = example.domain.com
```

```
usage: camanager [-h] -o OUTPUT [-n NAME] -c CONFIG [-cap CA_PASSWORD] [-icap INTERM_CA_PASSWORD]
                 {bash,powershell,native}

Generate a certificate using specified method.

positional arguments:
  {bash,powershell,native}
                        Specify the generator to use: 'bash', 'powershell', or 'native'.

options:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Specify the output directory for the generated certificates
  -n NAME, --name NAME  Specify the name of the certificate to be generated (MUST BE IN CONFIG.INI)
  -c CONFIG, --config CONFIG
                        Specify the path to the configuration file (config.ini).
  -cap CA_PASSWORD, --ca-password CA_PASSWORD
                        Specify the CA Password
  -icap INTERMEDIATE_CA_PASSWORD, --intermediate-ca-password INTERMEDIATE_CA_PASSWORD
                        Specify the Intermediate CA Password
```

Python CA Manager

The high-level design of our certificate management system illustrates the relationship between different components and the flow of certificate creation, validation, and usage. This architecture ensures proper security measures are implemented throughout the process.

Request Initiation

User or system requests certificate generation with required parameters

Validation & Usage

Certificates validated during connections and used for secure communications

Certificate Processing

System generates keys, creates CSR, and processes certificate signing

Storage & Management

Certificates and keys stored securely with proper access controls

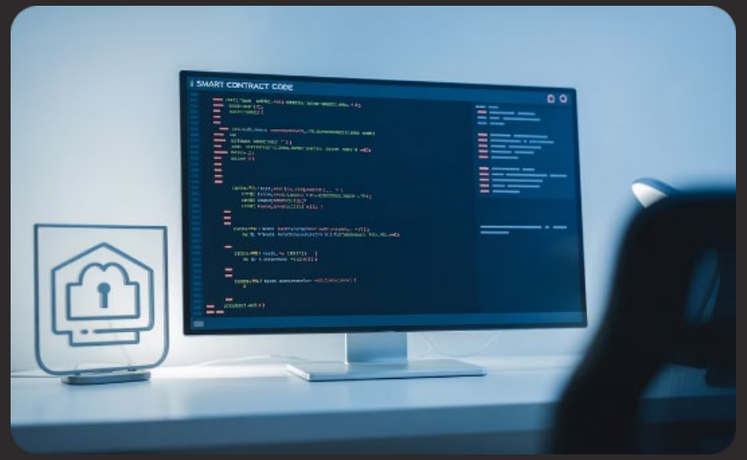
Common Attack Vectors Against Secure Web Systems

Attack	Vector	Privilege Level	IDS Detectability
ARP Spoofing	Network	User	Medium
DNS Spoofing	Network	User	Medium/High
BGP Rerouting	Network	ISP/Admin	Medium
Local DNS Modification	Local	Administrator/ Root	Low
Hosts File Modification	Local	Administrator/ Root	Low
HTTP Proxy Installation	Local/Network	User	High

These attacks target different aspects of secure web communications. ARP and DNS spoofing operate at the network level to redirect traffic, while local modifications require higher privileges but are harder to detect. HTTP proxies with fake SSL certificates can provide detailed control over HTTPS traffic but are more easily detected through certificate fingerprinting.

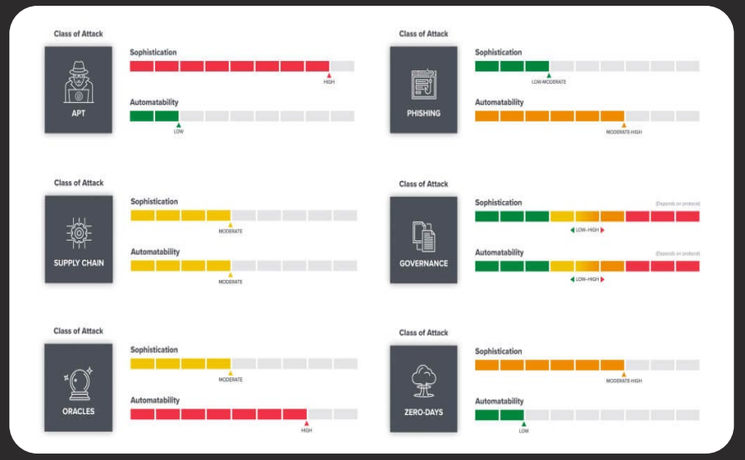
Next Steps: Web3 Security

Now that you understand PKI fundamentals and certificate generation, you're ready to explore how these concepts translate to Web3 environments. The principles of cryptographic trust, authentication, and secure communication remain essential but are implemented differently in blockchain and smart contract contexts.



Smart Contract Security

Apply your understanding of cryptographic principles to analyze and secure smart contract code in Hyperledger and Solidity, identifying vulnerabilities like reentrancy attacks.



Threat Modeling

Apply threat modeling methodologies to identify unique security challenges in decentralized applications and blockchain environments.

Web3 Attack Simulations

Simulate decentralized attacks and understand how these attacks work and how it might be possible to detect them.