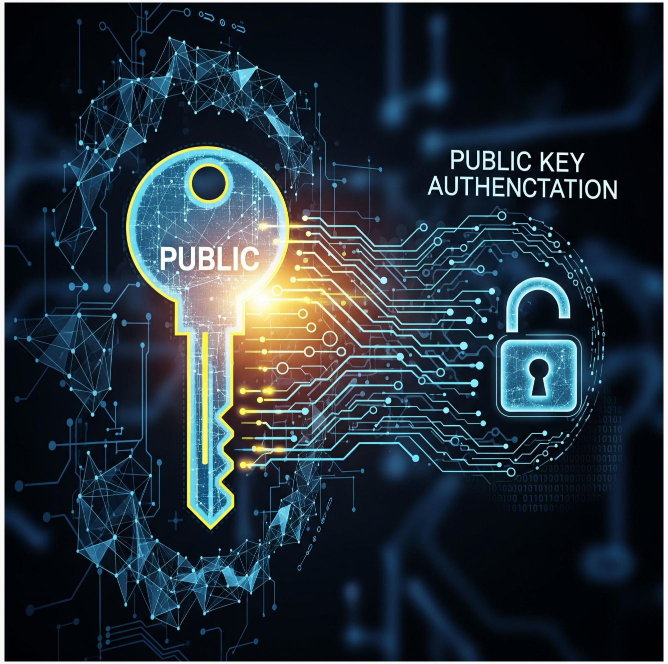
Authentication based on public keys

An introduction



Public key cryptography — recap

- Key pair: private (secret) + public (shared)
- Properties
  - Verify signatures with the public key
  - Infeasible to derive private from public
- **Auth = prove possession of private key without revealing it**

Public-key authentication flow (challenge–response)

- Steps
  - Verifier sends fresh random challenge
  - Prover signs challenge with private key
  - Verifier checks signature using public key
- No shared secret between prover and verifier required

The fake public key problem

- Public-key auth is only as strong as the authenticity of the public key
- Attack
  - Adversary supplies their own key as if it were the victim’s
  - Verifier then accepts signatures from the attacker
- Result: full impersonation despite strong cryptography

Ensuring key authenticity

- Certificates
    - Digitally signed statements binding identity ↔ public key
    - Verifier checks the signer's signature before trusting the key
- Alt/OOB checks
    - Manual key fingerprint/QR verification
    - TOFU (Trust On First Use) with later verification (context dependent)
- **Key authenticity** is the **critical prerequisite** for public-key auth

N-S based on pub key

K_{PX}: public key of X, Sig_C digital signature of C

Mutual authentication (Needham-Schroeder)

1. A to C: <this is A, want to talk to B>
2. C to A: <B, K_{PB}, $Sig_C(K_{PB}$,B)>
3. A checks digital signature of C, generates nonce N and sends to B: K_{PB}(N, A)
4. B decrypts (now wants to check A's identity) and sends to C: <B, A>
5. C to B: <A, K_{PA}, $Sig_C(K_{PA}$, A) >
6. B checks C's digital signature, retrieves K_{PA}, generates nonce N' and sends to A: K_{PA}(N, N')
7. A decrypts, checks N, and sends to B: K_{PB}(N')

Attack to N-S pub key

- Trudy is a system user that can talk (being authenticated) to A, B & C
- Two interleaved excerptions of the protocol
    - R1: authentication between A and T
    - R2: authentication between T (like A) with B (T (like A) is denoted by T(A))
- MITM
- T must be able to induce A to start an authentication session with T
- Steps 1, 2, 4, 5 allow to obtain public keys
- Steps 3, 6, 7 perform authentication

Attack to N-S pub key: steps

Steps 1, 2, 4 e 5 allow to know public keys
We focus on steps 3, 6, 7 of R1 and R2:

a) $A \longrightarrow T$: step 3 of R1 sends $K_{PT}(N, A)$
b) $T(A) \longrightarrow B$: step 3 of R2 sends $K_{PB}(N, A)$
c) $B \longrightarrow T(A)$: step 6 of R2 sends $K_{PA}(N', N)$
d) $T \longrightarrow A$: step 6 of R1 sends $K_{PA}(N', N)$
e) $A \longrightarrow T$: step 7 of R1 sends $K_{PT}(N')$
f) $T(A) \longrightarrow B$: step 7 of R2 sends $K_{PB}(N')$

B thinks that he is talking to A by sharing secret nonces

N-S based on pub key (fixed)

1. A to C: <this is A, want to talk to B>
2. C to A: <B, K_{PB}, $Sig_C(K_{PB}, B)$>
3. A checks digital signature of C, generates nonce N and sends to B: $K_{PB}(N, A)$
4. B decrypts (now wants to check A's identity) and sends to C: <B, A>
5. C to B: <A, K_{PA}, $Sig_C(K_{PA}, A)$ >
6. B checks C's digital signature, retrieves K_{PA}, generates nonce N' and sends to A: K_{PA}(B, N, N')
7. A decrypts, checks N, and sends to B: $K_{PB}(N')$

Why the previous attack fails

We focus on steps 3,6,7 of R1 and R2:

a) A$\longrightarrow$T : step 3 of R1 sends K_{PT}(N,A)
b) T(A)$\longrightarrow$B: step 3 of R2 sends K_{PB}(N,A)
c) B$\longrightarrow$T(A): step 6 of R2 sends K_{PA}(B, N′,N)
d) T$\longrightarrow$A: EARLIER in step 6 of R1 T sends K_{PA}(N′, N); NOW T **CANNOT** send K_{PA}(B, N′, N) to A
e) A$\longrightarrow$T: step 7 of R1 sends K_{PT}(N′)
f) T(A)$\longrightarrow$B: step 7 of R2 sends K_{PB}(N′)

X.509 Authentication standard

- Part of the standard family originally known as CCITT X.500 (now ITU-T)
- Introduced in 1988, revised multiple times
    - current stable version: 2012 – version 3
- Defines a directory of public keys signed by Certification Authorities (CAs)
- Provides the basis for authentication protocols (see e.g. Stallings 2005 or the ITU-T specification)
    - Official specification https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201210-S!!PDF-E&type=items
- Supports several authentication modes
    - One-way authentication (unilateral)
    - Two-way authentication (mutual)
    - Three-way authentication (mutual + anti-replay protection)
- Relies on public-key cryptography (PKC) and digital signatures
    - Cryptographic algorithms are not defined by the standard (X.509 is algorithm-agnostic)

X.509 (one-way) authentication

- Timestamp t_A
- Session key K_{AB}
- B's public key P_B
- $cert_A$: certificate of A's public key, signed by certification authority

- A □ B : $cert_A$, D_A, $Sig_A(D_A)$
  $D_A = <t_A, B, P_B(K_{AB})>$

X.509 (two-ways) mutual authentication

1. A □ B: $cert_A$, D_A, $Sig_A(D_A)$
   $[D_A = <t_A, N, B, P_B(k)>]$
2. B □ A: $cert_B$, D_B, $Sig_B(D_B)$
   $[D_B = <t_B, N', A, N, P_A(k')>]$

- t_A, t_B = timestamps, to prevent delayed delivery of messages; k, k′ session keys proposed by A and B; use of nonces avoids replay attacks
- Criticism: in D_A there is no identity of A

X.509 (three-ways) mutual authentication

Mutual authentication based on nonces, useful for unsynchronised clocks (0 denotes timestamp, optional)

1. A □ B: $<cert_A, D_A, Sig_A(D_A)>$
   $[D_A = <0, N, B, P_B(k)>]$
2. B □ A: $<cert_B, D_B, Sig_B(D_B)>$
   $[D_B = <0, N, A, N', P_A(k)>]$
3. A □ B: $<B, Sig_A(N, N', B)>$

Note: step 3 requires digital signature of nonces, making them tied (no replay attacks)

ISO/IEC 9798-3 (mutual authentication, 1990s)

- Part of the ISO/IEC 9798 family of authentication standards
- Uses public-key signatures and nonces for mutual authentication
- Goal: A and B prove identity to each other without a trusted third party
- Early version (pre-2000s) had a design flaw → vulnerable

9798-3 Early mutual authentication (bugged version)

1. B→A: N_B
2. A→B: $cert_A$, N_A, N_B, B, $Sig_A(N_A, N_B, B)$
3. B→A: $cert_B$, $N1_B$, N_A, A, $Sig_B(N1_B, N_A, A)$

Features

- Nonces (N_A, N_B) prevent replay
- Each party signs a challenge to prove identity
- At first glance: seems secure

Design flaw

- In step 3, B sends back $N1_B$ (fresh, unpredictable by A)
- A cannot verify correct linkage of identities/nonces
- Signed data does not bind peer identities strongly enough
- Similar to N-S flaw

“Canadian Attack” (Boyd & Mathuria, 2003)

a) $T(B) \rightarrow A: N_T$
b) $A \rightarrow T(B): cert_A, N_A, N_T, B, Sig_A(N_A, N_T, B)$
c) $T(A) \rightarrow B: N_A$
d) $B \rightarrow T(A): cert_B, N_B, N_A, A, Sig_B(N_B, N_A, A)$
e) $T(B) \rightarrow A: cert_B, N_B, N_A, A, Sig_B(N_B, N_A, A)$

- A accepts B’s valid signature, but it was relayed by T
- The missing binding of identities allows T to impersonate

Need of a PKI

1. Who owns the key?
   - Without PKI, anyone can present any public key
2. Certificates = Identity + Key
   - Bind an entity's identity to its public key
3. Trusted Authorities
   - Certification Authorities (CAs) act as global trust anchors
4. Scalability
   - Works across millions of users without pre-shared secrets

PKI ensures *this public key really belongs to that entity*

Passkeys

- A new paradigm that links biometric auth to public key auth

Passkeys

- Passkeys: beyond passwords
- How asymmetric cryptography replaces shared secrets in everyday logins
- Focus on protocol flows, security properties, and deployment
    - don't need PKI!
- Use PKC and biometric unlock
- Based on FIDO2
    - Fast IDentity Online (FIDO Alliance)

Cryptographic foundations

- Asymmetric crypto: (private, public) key pair
- Algorithms: modern versions of DSA based on elliptic curves (more advanced)
    - ECDSA (NIST standardized)
    - EdDSA (based on Edwards curves)
- Security: based on discrete log problem hardness

Key generation

- Keys generated inside a secure authenticator
    - enclave, secure, isolated execution environment inside a CPU or chip
    - TPM, Trusted Platform Module, dedicated chip (or firmware module) standardized by the Trusted Computing Group
    - HSM, Hardware Security Module, specialized external device (often a PCI card or network appliance) for high-assurance key management
- Private key never leaves the device
- One key pair per domain

Registration

- User registers on site
- Server sends challenge and id (called RP ID)
- Authenticator generates key pair
- Authenticator returns to server
    - Public key
    - Credential ID (identifier to retrieve key inside authenticator)
    - Attestation signature (proves the authenticator and key are genuine)
- Server stores only: (public key, RP ID, credential ID)

Authentication flow

- User tries to log in
- Server generates a fresh challenge (nonce)
- Server sends: challenge + credential ID
- Authenticator finds the private key for RP ID
- Authenticator signs
    - Challenge
    - RP ID (origin binding)
    - User handle
    - Metadata (e.g., counter to prevent replay)
- Server verifies signature with stored public key
    - If valid → user authenticated
- Replay prevented: challenge is unique & bound to session

Origin binding

- Credentials are scoped to RP ID (domain)
    - Example: key for bank.com cannot be used at evil.com
- Enforced by browser and authenticator
- Cryptographic protection against phishing

Server storage model

- Server keeps only
    - Public key
    - Credential ID
    - Metadata

- No password hashes → nothing to brute-force offline.

Cryptographic guarantees

- Authenticity: only holder of private key can sign challenge
- Integrity: challenge, RP ID, and context are signed together
- Replay protection: fresh challenges, counters
- Phishing resistance: domain binding in signed payload
    - It does not stop DNS poisoning if the attacker can also present a valid TLS certificate for the RP domain

Cross-device authentication

Logging into site with a passkey stored on a smartphone

1. Visit site on laptop → site requests passkey
2. Browser checks for local credentials
   - If none found → offers “use passkey from another device”
3. Cross-device channel established
   - Laptop shows QR / uses Bluetooth → phone connects
   - Secure ephemeral channel is created
4. Authentication on the phone
   - Retrieve passkey for site
   - Unlock with biometrics / PIN
   - Private key signs challenge
5. Signed response sent back to browser → login succeeds

Why Passkeys Don’t Need a PKI

- Passkeys generate per-site key pairs (one key pair per RP)
- The RP stores the user’s public key during registration
- No need to prove the public key’s identity to third parties
- Authentication is proof of possession, not certificate-based trust
- Domain binding is enforced via RP ID, not by CAs
- Result: no certificates, no CA chain, no PKI validation required

Comparison with passwords

Passwords	Passkeys
Shared secret	Asymmetric key pair
Server stores hashes	Server stores only public keys
Susceptible to phishing	Bound to origin
Offline cracking possible	No secret to crack
Reuse across sites	Unique key per domain