MI-PB-21

Tvorba a správa šifrovacích klíčů a certifikátů.

Autentizace

Autentizace:

- **Dat:** MAC, PKI
- **Entit:**
  - **Uživatelů:** hesla, certifikáty, tokeny, biometrické údaje
  - **Systémů:** certifikáty, PUF
  - **Organizací:** PKI

Identifikace: subjekt poskytne identifikační údaje
Autentizace: ověření identifikačních údajů
Autorizace: určení, zda má subjekt právo na přístup k objektu

Cíle autentizačních protokolů:

- Strana A je schopna se plně autentizovat straně B
- *Nepřenositelnost:* B nemůže použít identifikační údaje A získané při předchozí autentizaci, aby se autentizovala u strany C
- *Imitace:* Je zanedbatelná šance, že strana C napodobující A přiměje stranu B, aby si myslela, že jde o A
- Předchozí body platí i pokud bylo pozorováno velké množství předchozích autentizací mezi A a B a útončník C se sám účastnil autentizací s A i B

Autentizační faktory:

- Tradiční:
  - co vím: heslo, PIN, bezpečnostní otázka
  - co mám: čip, OTP token, telefon
  - co jsem: biometrika
- Přídavné faktory:
  - jak se chovám: jak často se přihlašuji, jaké činnosti provádím

- odkud pocházím: fyzicky (GPS), virtuálně (VPN)

Hesla

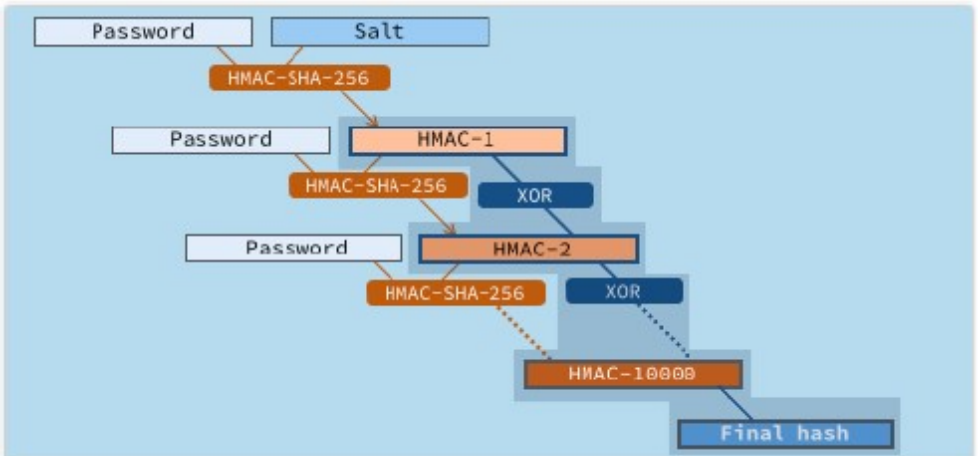
Čím vyšší **entropie**, tím déle trvá cracknout heslo

Problémy:

- Jednoduché sdílení
- Jednoduše zapomenutelné
- Často jednoduše uhodnutelné
- Lze si ho zapsat (co vím $\Rightarrow$ kde to nalézt)
- Může zůstat v paměti/cache

Bezpečné úložiště hesel:

- Heslo musí kvůli autentizaci být uloženo na serveru
- Nejlepší přístup: databáze položek $\mathrm{hash}(\mathrm{heslo}, \mathrm{sůl}), \mathrm{sůl}$
- **Solení:**
  - Generování soli kryptograficky bezpečným RNG
  - Spočítat $\mathrm{hash}(\mathrm{heslo}, \mathrm{sůl})$
  - Uložit hash a sůl do DB
- **Password stretching:**
  - Náročnější bruteforce kvůli opakovanému hashování (znásobení r-krát)
  - $x_0 = 0$
    Pro $i = 1...r : x_i = \mathrm{hash}(x_{i-1}||p||s)$
    $H = x_r$
  - **PBKDF2:** Password-Based Key Derivation Function 2
    - Původně odvozování klíčů, dnes i ukládání hesel



PBKDF2 using XOR to combine 10,000 successive HMAC-SHA-256 outputs into a final hash

Přenos hesel:

- Hashovat vždy na straně **serveru**
  - Hash na straně klienta:
    - Útočník odposlechne hash $\Rightarrow$ může ho použít i bez znalosti hesla
    - Útočník kompromituje DB hesel na serveru $\Rightarrow$ autentizace bez nutnosti crackování
    - Server by klientovi nejdříve musel poslat sůl
- **Bezpečný přenos:**
  - Šifrování: náchylné na MITM
  - Hash: náchylné na slovníkové útoky
  - **Challenge-response protocol:**
    - Klient $\rightarrow$ server: Identita klienta
      Server $\rightarrow$ klient: výzva C
      Klient $\rightarrow$ server: $\mathrm{hash}(C||\mathrm{heslo})$
  - **Secure Remote Password Protocol:**
    - Klient prokáže znalost tajemství, aniž by samo tajemství odhalil
    - Server ukládá pětice
      $(\mathrm{username}, \mathrm{pass\ verifier}, \mathrm{sůl}, g, N)$, kde:
      - $\mathrm{sůl} = \mathrm{random}()$
      - $x = \mathrm{SHA}(\mathrm{sůl}|\mathrm{SHA}(\mathrm{username})|":"|\mathrm{heslo})$
      - $\mathrm{pass\ verifier} = v = g^x \pmod N$
      - N prvočíselný modul, g generátor

One-time password:

- Jednorázová heslo významně posílí autentizaci
- Synchronizované generátory:
  - **Counter based**
    - Heslo je výstup funkce vnitřního čítače a jiných vnitřních dat
    - $\mathrm{HOTP}(K, C) = \mathrm{Truncate}(\mathrm{HMAC\text{-}SHA\text{-}1(K,C)})$
      K: sdílený klíč klientem a serverem
      C: hodnota čítače
    - Hodnota serveru inkrementována pouze po úspěšné autentizaci, hodnota tokenu inkrementována vždy: nutná resynchronizace (look-ahead window -- server počítá a porovnává několik hodnot dopředu)
  - **Clock based:**
    - Uživatel má token, který zobrazuje časově závislý kód

        - Přístup k tokenu nezbytný pro znalost aktuálního kódu
        - $\mathrm{TOTP}(K) = \mathrm{HOTP}(K, T)$, kde T je timestamp
        - Možný časový rozdíl mezi serverem a tokenem: resynchronizace (server ověřuje s více hodnotami dopředu i dozadu v čase, po ověření si server může pamatovat časový posun)
    - **Lamportovo schéma:**
        - Klient počítá řetězec hashů $S_1 = \mathrm{seed}, S_2 = \mathrm{hash}(S_1), S_3 = \mathrm{hash}(S_2), ..., S_n = \mathrm{hash}(S_{n-1})$
        - Inicializace: klient pošle serveru S_n,
        - Při autentizaci jsou hesla používána odzadu: Klient se přihlásí heslem S_i, server spočítá $\mathrm{hash}(S_i)$ a ověří, jestli výsledek sedí se zapamatovaným S_{i+1}
    - **Challenge-Response OTP:**
        - Symetrické, jednosměrná autentizace:
          $A \leftarrow B : r_B$
          $A \rightarrow B : E_K(r_B, B)$
        - Symetrické, vzájemná autentizace:
          $A \leftarrow B : r_B$
          $A \rightarrow B : E_K(r_A, r_B, B)$
          $A \leftarrow B : E_K(r_B, r_A)$

Biometrické údaje

Ne-vizuální: Dynamika úderů na klávesnici, podpis, biorytmy (srdce), řeč, gesta
Vizuální: duhovka, sítnice, otisk prstu, obličej, žíly

Autentizace = pattern regognition

Módy:

- **Verifikační mód:** "je uživatel tím, kdo tvrdí?" (porovnání 1:1)
- **Identifikační mód:** "kdo je uživatel?" (porovnání 1:N)

Fáze:

- **První fáze:** registrace (enrolment)
    - Uživatelé se registrují, jejich biometrické údaje uloženy do systému jako vektory zajímavých vlastností
    - Kompletní naměřené hodnoty mohou být zašifrovány a uloženy, mohou pomoct s pozdější

    alternativní feature extraction
- **Druhá fáze:** autentizace
    - Potvrzena/určena identita uživatele

False Acceptance Rate (FAR): poměr shod, které jsou nesprávné, vůči všem shodám

False Rejection Rate (FRR): poměr nelegitimních shod, které jsou ve skutečnosti správné, vůči všem shodám

Kryptografie veřejného klíče

Řeší problém distribuce klíčů přes nezabezpečený kanál

Lze použít pro autentizaci -- challenge-response protokoly

Distribuce veřejných klíčů bez kontaktu s třetím důvěryhodným subjektem

Certifikát: Struktura obsahující:

- Veřejný klíč držitele
- Identifikační údaje držitele
- Dobu platnosti certifikátu
- Další údaje vytvořené certifikační autoritou

Certifikát **podepsán soukromým klíčem CA** -- každý účastník komunikace může ověřit veřejným klíčem CA

Trust anchor: entita, jejíž důvěryhodnost není odvozena, ale předpokládána

Certifikční autorita: Důvěryhodná třetí strana, která na základě žádostí vydává a aktualizuje certifikáty

Certifikační strom: Struktura vzájemně propojených CA, reprezentován kořenovou CA s kořenovým certifikátem

- **Řetězec certifikátů:** Posloupnost certifikátů od certifikátu uživatele ke kořenovému certifikátu
- Certifikát platný $\Leftrightarrow$ platné všechny certifikáty v řetězci certifikátů
- Kořenový certifikát ověřen jinou bezpečnou cestou, např. křížovou certifikací (2 kořenové CA si vzájemně ověří certifikáty)
- Komunikace mezi A a B v různých stromech: A pošle B certifikát A podepsaný CA1, certifikát

CA1 podepsaný CA1, certifikát CA1 podepsaný CA2

Autentizace založená na asymetrické kryptografii:

- Jednosměrná s timestampy:
  - $A \rightarrow B : \mathrm{cert}_A, t_A, B, E_{S_A}(t_A, B)$
- Jednosměrná s náhodnými čísly:
  - $A \leftarrow B : r_B$
    $A \rightarrow B : \mathrm{cert}_A, r_A, B, E_{S_A}(r_A, r_B, B)$
- Vzájemná s náhodnými čísly:
  - $A \leftarrow B : r_B$
    $A \rightarrow B : \mathrm{cert}_A, r_A, B, E_{S_A}(r_A, r_B, B)$
    $A \leftarrow B : \mathrm{cert}_B, A, E_{S_B}(r_B, r_A, A)$

Bezpečné ukládání privátních klíčů:

- Klíč by nikdy neměl opustit zabezpečené úložiště
- Řešení:
  - **čipová karta** (koncový uživatel)
  - **Hardware Security Module** (server)

Hardware Security Module

Důvěryhodné zařízení

Logická a fyzická tamper-proof ochrana:

Funkce:

- Bezpečné generování kryptografických klíčů
- Bezpečné úložiště kryptografických klíčů
- Bezpečné používání kryptografických klíčů
- Akcelerace kryptografických operací
- Nepovolí export klíčů v plain textu

Podoby: PCI karta, USB zařízení, samostatné síťové zařízení

Využití:

- PKI infrastruktura (certifikace, registrace autorit)

- Platební systémy
- Generování elektronických podpisů
- SSL

Typická architektura: kominukace přes API (custom / PKCS#11 / Microsoft CNG / Java Cryptographic Engine)

Fyzická bezpečnost:

- Detekce neoprávněného přístupu (pečetě, nálepky, senzory teploty, světla, pohybu, energie)
- Automatická akce po detekci neoprávněného přístupu (bezpečné vymazání)
- Pryskyřice, nestandardní šrouby,...

Logická bezpečnost:

- SW/FW updaty (autentizace, integrita)
- Periodické kontroly integrity SW
- Access control -- limity na čas a počet přístupů
- Přesné hodiny
- Autentizace před každou komunikací
- Statistické testy náhodného generátoru

Tamper Security:

- **Tamper Evidence:** neoprávněný přístup dokumentován, snadno ho lze zjistit (např. audit log, samolepky)
- **Tamper Resistance:** odolnost proti manipulaci normálních uživatelů (např. speciální šrouby)
- **Tamper Detection:** automatická detekce manipulace samotným objektem
- **Tamper Responsivness:** po detekci manipulace promazání

Správa klíčů:

- Mimo hranice HSM se klíč nesmí objevit v plain textu
- **Kategorie klíčů:**
  - **Storage keys:** např. Host Master Key -- šifrování ostatních klíčů na úložišti
  - **Transport keys:** klíče šifrující klíče během jejich výměny
  - **Functional keys:** klíče pro specifické kryptografické operace
- **Ochrana klíčů:**
  - Všechny klíče v chráněné paměti
    - Pokud HSM manipulován, všechny klíče nutné znovu nahrát
  - Jeden klíč v HSM (Host Master Key), ostatní v centrální šifrované databázi

- Kompromitace HMK: ostatní kompromitovány taky

Centralizace správy hesel

Pokud v síti N serverů a M uživatelů, každý server autentizuje všech M uživatelů

Centralizace: jeden autentizační server

Needham-Schroeder protokol: (KDC - key distribution center)

1. $A \rightarrow KDC : ID_A||ID_B||n_1$
2. $KCD \rightarrow A : E_{K_A}(K_S||ID_A||ID_B||n_1)||E_{K_B}(K_S||ID_A)$
3. $A \rightarrow B : E_{K_B}(K_S||ID_A)$
4. $B \rightarrow A : E_{K_S}(n_2)$
5. $A \rightarrow B : E_{K_S}(f(n_2))$

- Riziko:
  - odposlech a replay kroku 3, B pak může používat starý klíč, který útočník kompromitoval
  - odposlech kroku 4 -- útočník může napodobit krok 5

Denning-Sacco protokol:

- Jako Needham-Schroeder, ale používá timestamp místo n

Životnost session key:

- Častější výměna session key:
  - Vyšší bezpečnost
  - Větší zátěž na síť

Kerberos:

- Založen na Needham-Schroederovi
- Umožňuje Single-Sign-On (uživatel se autentizuje jednou a má přístup k více aplikacím)
- **Model důvěry:** všichni věří Kerberos serveru, neexistuje výchozí důvěra mezi jinými servery, síť není důvěryhodná, uživatel věří lokálnímu stroji
- **Průběh:**
  LM - lokální stroj
  AS - autentizační server
  TGS - ticket granting server

AD - síťová adresa

t - timestamp

server - server, se kterým chce uživatel pracovat

1. Uživatel se přihlásí na lokálním stroji, vyžádá službu na serveru
   - $LM \rightarrow AS : ID_{LM} || ID_{TGS} || t_1$
2. Autentizační server ověří oprávnění, tvoří **ticket-granting ticket** a **session key** a zašifruje je klíčem vytvořeným z uživatelova hesla
   - $AS \rightarrow LM : E_{K_{LM}}(K_{LM,TGS} || ID_{TGS} || t_2 || \mathrm{životnost}_1 || \mathrm{ticket}_{TGS})$
   - $\mathrm{ticket}_{TGS} = E_{K_{TGS}}(K_{LM,TGS} || ID_{LM} || AD_{LM} || ID_{TGS} || t_2 || \mathrm{životnost}_2)$
3. Lokální stroj dešifruje zprávu, odešle ticket a autentizátor (jméno, síť, IP, čas) ticket-granting serveru
   - $LM \rightarrow TGS : ID_{\mathrm{server}} || \mathrm{ticket}_{TGS} || \mathrm{auth}_{LM}$
   - $\mathrm{auth}_{LM} = E_{K_{LM,TGS}}(ID_{LM} || AD_{LM} || t_3)$
4. Ticket-granting server dešifruje ticket a autentizátor, ověří požadavek, vytvoří ticket pro žádaný server (**service-granting ticket**)
   - $TGS \rightarrow LM : E_{K_{LM,TGS}}(K_{LM,\mathrm{server}} || ID_{\mathrm{server}} || t_4 || \mathrm{ticket}_{\mathrm{server}})$
   - $\mathrm{ticket}_{\mathrm{server}} = E_{\mathrm{server}}(K_{LM,\mathrm{server}} || ID_{LM} || AD_{LM} || ID_{\mathrm{server}} || t_4 || \mathrm{životnost}_4)$
5. Lokální stroj pošle ticket a autentizátor žádanému serveru
   - $LM \rightarrow \mathrm{server} : \mathrm{ticket}_{\mathrm{server}} || \mathrm{auth}_{LM}$
6. Server ověří shodu ticketu a autentizátoru, povolí přístup
   - $\mathrm{server} \rightarrow LM : E_{K_{\mathrm{server}}}(t_5 + 1)$ (vzájemná autentizace)

- **Kerberos Realm:** několik uzlů sdílející stejnou Kerberos databázi
  - Spolupracující realmy -- jejich Kerberos servery sdílí tajný klíč, uživatel může zažádat o ticket-granting ticket jiný server (do jiného realmu)
- V praxi: Windows Active Directory
- **Golden Ticket:** podvržený ticket-granting ticket (potřeba hash hesla KRBTGT účtu, jméno domény a SID, user ID)
- **Silver Ticket:** podvržený service-granting ticket (omezenější dosah, ale jednodušší získat -- místo KRBTGT účtu stačí hash hesla service účtu)
- **Kerberoasting:** získávání údajů service účtů z Active Directory (crackování ticketů šifrovaných heslem uživatele)