# Zpracované otázky na Aplikovanou kryptografii

Karel Velička

Poslední edit: 15. května 2024

Tento dokument byl sepsán narychlo *(navíc se konalo MS světa v hokeji - občas špetka alkoholu přikrášlila text výrazy typu "OMG" apod.)*, takže za správnost ručit nemůžu. Valná většina materiálu je z pdf přednášek. Doplňkové info jsem čerpal obvykle z Wiki nebo z knihy Real-World Cryptography.

## Obsah

# 1 Protokol HTTP, cookies, Basic autorizace

## 1.1 Protokol HTTP - Hypertext Transfer Protocol

Jedná se v podstatě o protokol pro WWW. Je součástí Aplikační vrstvy.
Tradičně funguje nad TCP. Veškerá data jsou poslána v plaintextu, proto je zapotřebí komunikaci zabezpečit. K tomu se využívá mechanismů TLS. Tedy šifrování komunikace a autentizace komunikujících stran - HTTPS.
HTTP používá pro komunikaci tzv. DNS (*Domain Name System*). Jedná se zjednodušeně jen překlad domén na IP adresy a naopak. *(DNSSEC - zajišťuje i podepisování)*
Fungování: Klient, což je obvykle prohlížeč, pošle požadavek (`request`) na server. Server odpoví (`response`). Jak požadavek, tak odpověď mají hlavičku a tělo, kde hlavička obsahuje metadata a tělo obsahuje data.
Často jsou používány metody: `GET`, `POST`, `HEAD`, `PUT`, `DELETE`, atd.

## 1.2 Cookies

Jedná se ve zkratce o malý soubor, který server pošle prohlížeči a ten ho uloží. Obsah souboru je nastaven serverem. Slouží primárně pro ukládání informací.
Běžný prohlížeč nastavenou cookie pošle zpět při každém dalším požadavku
Cookies jde nastavovat, čisté nastavení se provádí příkazem `Set-Cookie`. Naopak zaslání cookie se provádí příkazem `Cookie`. V případě, že je cookie nastavena bez domény, je dostupná pouze pro doménu, ze které byla vytvořena.
Cookies mohou mít různé vlastnosti: jméno + hodnota; doména; cesta = `path`; expirace (`session`, konkrétní čas); příznaky = `flags`.
Poslední vlastnost, právě `flags`, může mít několik hodnot:

- `Secure` - zasílá se pouze přes HTTPS
- `HttpOnly` - není dostupná z JavaScriptu
- `SameSite` - pravidla pro „cross-origin“ zasílání

## 1.3 Basic autorizace

Jedná se o jednoduchý způsob autentizace klienta na serveru.

### HTTP security headers

Jedná se o nastavení, jak má prohlížeč se stránkou nakládat. Jde o komunikaci server → prohlížeč.

- `Referrer-Policy` - jak má prohlížeč zpracovávat informace o odkud byl uživatel přesměrován
- `X-Frame-Options` - zda je možné vkládat stránku do iframe
- `X-Content-Type-Options` - zda je možné měnit typ obsahu
- *`Content-Security-Policy`* - jaký obsah je možné zobrazit
- *`Strict-Transport-Security`* - zda je možné použít HTTP

### Hlavička Authorization

Slouží pro autentizaci klienta. Má zabudované metody:

- `Basic` - v každém požadavku jméno + heslo
- `Digest` - challenge-response, nepoužívá se
- `NTLM`, `Negotiate` - SSO v MS Active Directory
- `Bearer` - (náhodný) token. Bearer tokeny lze využít i jakoukoli vlastní hlavičku (není třeba podpora ze strany prohlížeče)

Autorizace probíhá následovně: Server pošle klientovi hlavičku *WWW-Authenticate: Basic realm="Přihlas se"* s typem autentizace a realmem.
Klient naopak pošle v každém dalším požadavku serveru hlavičku *Authorization: Basic* $\underbrace{dXNlcjE6SGVzbG8xMjM=}_{base64(user1:Heslo123)}$.

**Base64**

Jedná se o kódování, které převádí binární data na tisknutelné znaky. Konkrétně se jedná o převod trojic bytů na čtveřici tisknutelných znaků `[a-Z][0-9]+/`
Byty zprávy se poskládají za sebe, rozdělí se po šesticích a každá šestice se zakóduje jedním znakem dle převodní tabulky.
V případě, že zakódovaná zpráva není násobkem 3 bytů, je nutné ji doplnit 0-bity a na konec přidat padding `=` podle toho, kolik bylo bitů přidáno - pokud byly přidány 2 bity, tak se přidá `=`; pokud 4, tak `==`. Není to však povinné.

**Origin**

Jedná se o URL, ze které byl požadavek odeslán. Tedy protokol, doména a port.

- *Same origin policy* - protokol, doména, port
- *Cross-origin policy* - povolen zápis (např. odeslání formuláře `<a href>`), povoleno vnoření (embedding - `<script>`, `img`, `<iframe>`), zakázáno čtení
- *Cross-Origin Resource Sharing* - sdílení zdrojů mezi doménami, typické u API
-
  - `Simple` - ´pouze `HEAD`, `GET`, `POST` + bezpečné hlavičky
  - `Preflight` - když nejsou splněny podmínky pro simple

# 2 Formát certifikátů X.509, důvěryhodnost certifikátu

## 2.1 Standard X.509

Jedná se o standard pro certifikáty veřejných klíčů.
Konkrétněji se jedná o propojení vlastnictví privátního asymetrického klíče s reálnou identitou (osoby, serveru, atd.). Využívají se k tomu šifry jako RSA, DSA, ECDSA, EdDSA
Nutnost je využití důvěryhodné třetí strany (*certifikační autority*). *Public key infrastructure (PKI)* - zajišťuje vydávání, správu a odvolávání certifikátů
*Certifikát* obsahuje: verzi, veřejný klíč, vydavatele, elektronický podpis CA, platnost, předmět,...
Dvojice (vydavatel; sériové číslo) jednoznačně identifikuje certifikát.

## 2.2 Základní princip

Alice chce Bobovi poslat zprávu, k tomu potřebuje Bobův veřejný klíč. Chce mít ale jistotu, že používá veřejný klíč, který opravdu patří Bobovi.
Ověření/ verifikace se provádí pomocí certifikátu, který vydala důvěryhodná třetí strana.
Využívá se k ověřování stromovitá struktura.

## 2.3 Ověřování certifikační cesty

Začíná od důvěryhodné kotvy (obráceně než sestavování).
Postupně se ověří následující body:

- *Vydavatel* certifikátu $x$ je *předmětem* certifikátu $x - 1$.
- *digitální podpis* certifikátu $x$ pomocí certifikátu $x - 1$.
- aktuální *čas* v době platnosti certifikátu - jestli náhodou nebyl *odvolán* (revoked).
- *Jména* uvedená v předmětu certifikátu a v rozšíření (Alternativní jména předmětu *odpovídají omezením* vyplývajícím z rozšíření. Omezení jmen v předchozích certifikátech certifikační cesty.)
- Verifikují se rozšíření spojená s *certifikačními politikami.*

Důvěryhodné kotvy jsou často distribuovány už spolu s operačním systémem. Některé prohlížeče, jako třeba Firefox, používají vlastní seznamy důvěryhodných CA.
Lze přidávat dle potřeby uživatele.

## 2.4 Technický pohled na věc

Formáty ve zkratce: ASN.1 → DER → Base64 → PEM.
Pro zápis struktur se používá jazyk *ASN.1* - je to abstraktní jazyk pro popis typů dat.
ASN.1 je struktura zakódovaná ve formátu *DER* a DER je zakódované *base64 + hlavička* a *patička*, neboli *PEM*.
*Struktura DER* je způsob, jak zapsat data definovaná ASN.1 strukturou. Základní schéma *Typ dat | Délka dat | Data*. Obsahuje *SET* a *SEQUENCE*, což je posloupnost dat jiných typů.
*Object identifier* je zápis cesty ve stromové struktuře.

**Jedinečná jména** mohou být relativní nebo spojená dohromady. V relativní formě jsou jednotlivé prvky odděleny čárkami, například *CommonName (CN), Organization (O), Country (C)*. Tento zápis využívá stejnou konvenci jako LDAP

## 2.5 Rozšíření

Certifikáty je možné rozšířit o tzv. *Extention*. V souvislosti s tím musí platit i tzv *Critical extension* - pokud klient rozšíření nezná, musí certifikát odmítnout
*Identifikátor klíče* zajišťuje rozlišení více certifikátů.
Použití klíče

- *digitalSignature* - elektronický podpis (např. pro autentizaci). Neumožňuje podepisovat certifikáty a CRL.
- *nonRepudation* - nepopiratelný elektronický podpis. Neumožňuje podpis certifikátů a CRL.
- *keyEncipherment* - šifrování klíčů. Data šifrována symetricky; asymetricky klíč pro symetrickou šifru.
- *dataEncipherment* - šifrování jiných uživatelských dat, než jsou kryptografické klíče.
- *keyAgreement* - veřejný klíč je určen pro algoritmy založené na výměně klíčů (např. Diffie-Hellman).
- *keyCertSign* - podepisování certifikátů.
- *cRLSign* - podepisování seznamu odvolaných certifikátů (CRL).

# 3 Platnost certifikátu, jeho obnova a odvolání

Časová informace - obsahuje `Validity` a `Time` (UTCTime, GeneralizedTime).

**Životní cyklus certifikátu** Žádost → Vydání → $\begin{cases} \text{Expirace certifikátu} \\ \text{Odvolání certifikátu} \rightarrow \text{Publikace na CRL } \textit{(nelze obnovit)} \end{cases}$

**Obnovení certifikátu** Obnova se provádí příkazy `Renew` a `Rekey` (navíc nový privátní klíč). Dokud je starý certifikát platný, je možné se jím stále identifikovat.

**Odvolání certifikátu** Dojde ke kompromitaci soukromého klíče *(Invalidity date)*, dále CA obdrží požadavek na odvolání *(Revocation date)* a nakonec dojde k Odvolání certifikátu - update na CRL.

## 3.1 CRL - Certificate Revocation List

Je to seznam certifikátů, které byly odvolány. Obvykle je vydáván v pravidelných intervalech.
Existují tři hlavní typy CRL:

1. *Úplné CRL* obsahuje všechny odvolané, ale stále platné certifikáty.
2. *Rozdílové CRL* obsahuje pouze certifikáty, které byly odvolány od posledního vydání úplné CRL.
3. *Omezené CRL* je podmnožina certifikátů podle určitého kritéria, například typu.

Rozdílové CRL mohou být rozšířeny o speciální funkci, kterávyžaduje, aby byly zkoumány i úplné CRL.

## 3.2 OSCP - Online Certificate Status Protocol

Je to protokol pro ověření platnosti digitálních certifikátů online. Je to vlastně alternativa k CRL, která umožňuje přímý dotaz na platnost jednotlivých certifikátů. Díky tomu je přenášeno méně dat a publikace změn je rychlejší, což zlepšuje výkon a efektivitu systému ověřování certifikátů.

**Rozšíření Authority Information Access (AIA)** slouží k poskytnutí informací potřebných k nalezení certifikátu vydávající CA a k získání informací o OCSP.

**OCSP stapling** je technika, která řeší problémy s přetížením a pomalými odpověďmi OCSP responderů. Server hostující web může pravidelně žádat o podepsaný OCSP status a ten rovnou zaslat klientovi spolu s certifikátem. (Platnost OCSP odpovědi je typicky 24 hodin).

# 4 Vystavení certifikátu

## 4.1 Žádost o certifikát

Je zapotřebí *Identifikace* žadatele, *Veřejný klíč*, Důkaz o držení *privátního klíče*, Požadované *použití klíče*
Platformy vydávající certifikáty - Let's Encrypt (certbot), CESNET,

#### Důkaz o vlastnicví privátního klíče a CSR

CSR je zkratkou pro *Certificate Signing Request* a označuje elektronický podpis. Vydaný certifikát je šifrován pomocí *veřejného klíče*. Existuje také alternativní cesta, jako je generování párových dat certifikační autoritou. Atributy v CSR (PKCS #10) zahrnují:

- *challengePassword*, což je jednorázové heslo určené pro odvolání certifikátu
- *extensionRequest*, který umožňuje specifikovat požadovaná rozšíření certifikátu.

CMRF (*Certificate Request Message Format*) je standardní formát zprávy určený k přenosu requestu na vystavení certifikátu. Zjednodušuje komunikaci mezi klientem a CA.

**Self-signed** Je to koncový certifikát, kde *subjekt je totožný s vydavatelem* certifikátu. Obvykle se používá pro *testovací* účely a je velmi obtížné ověřit jeho důvěryhodnost.

**Certificate transparency** je systém udržující seznam všech vydaných certifikátů s cílem hlídat podvodné CA. Pokud není certifikát v seznamu, tak ho moderní prohlížeče považují za neplatný.

**DNS záznam CAA** (*Certification Authority Authorization*) umožňuje doménovým správcům specifikovat, která CA je oprávněna vydávat certifikáty pro danou doménu. Před vydáním certifikátu je CA povinna zkontrolovat existenci CAA záznamu.
V kombinaci s Certificate transparency umožňuje komukoli identifikovat potenciálně podvodné CA.

**Formát PKCS #12** je standardní formát, který slouží ke spojení certifikátu a privátního klíče do jednoho souboru. Často se využívá pro import osobních certifikátů do operačních systémů, prohlížečů a dalších aplikací. Používá se opět ASN.1 struktura. (přípony `.p12`/ `.pfx`).

# 5 TLS - Transport Layer Security

Je to univerzální šifrovaná vrstva využívaná v mnoha protokolech (např. HTTPS, SMTP, POP3, FTPS).
Zajišťuje *důvěrnost* (šifrování), *integritu* (podepisování) a *autentizaci* jedné nebo oubou stran.

## 5.1 Verze a varianty TLS

TLS je až novějším názvem pro staré SSL (Secure Sockets Layer). S tím i souvisí verze: SSL-2, SSL-3/TLS-1.1 jsou prolomitelné a nejdou využít pro moderní šifry. Naopak verze TLS 1.2 a TLS-1.3 jsou bezpečné.
Existují i další varianty TLS, například *DTLS*, což je TLS komunikující prostřednictvím protokolu UDP.

## 5.2 Spojení

Nejprve dojde k TCP handshake, dojde ke spojení a následně dojde k první části šifrování - asymetrického.
Ověří se certifikát (autentizace) a dojde k ustanovení veřejných klíčů. Tím se dostaneme k druhé části šifrování - symetrické - kde už jen dochází k běžnému toku dat.
Nejprove dojde k domluvení veškerých potřebných parametrů - ustanovení šifrovacích klíčů, autentizace - a následně dojde k šifrovanému přenosu.

**PSI - Pre-Shared key** Vyskytuje se v TLS-1.3. Protože výměna klíče a ověření certifikátu může být zbytečně složité, používá se tzv. *PSK*, což je tajemství, které znají obě strany. Z tohoto tajemství se odvodí šifrovací klíče a informace o něm jsou v ClientHello.

**SNI - Server Name Indication** V rámci ClientHello zaslaná informace o HTTP Host hlavičce. Aby server už v okamžiku navazování spojení věděl, s jakou doménou bude spojení navázáno. Podle toho vybere certifikát/ nabídne jiné parametry TLS/ vyžaduje klientskou autentizaci.

### 5.3 Autentizace

Autentizuje se za pomoci ceritifikátu X.509 a autentizuje (v podstatě) vždy serverová strana.
Klientská strana může také, ale není to obvyklé (volitelně).
Pro autentizaci se většinou používají *RSA* (nejčastěji; velikost modulu $\geq 2048b$), *ECDSA/EdDSA* (klíč $\geq 256b$) nebo *DSA* (klíč $\geq 2048b$).

### 5.4 Bezpečný přenos dat

Pro šifrování obsahu se využívá mnoho možností, nejčastěji však *AES-CBC + HMAC*, případnš moderní AEAD šifry (AES-GCM, ChaCha20-Poly1305).

#### AES

Bloková šifra, kde blok má velikost *128b = 16B*, klíč *128/192/256b*.

## 6 Diffie-Hellmanova výměna klíče, HMAC, HKDF

### 6.1 Diffie-Hellman

Jedná se o algoritmus pro bezpečnou výměnu klíče. Je založen na problému diskrétného logaritmu. V dnešní době se často kombinuje s eliptickými křivkami.

#### Popis algoritmu

Mějme velké prvočíslo $p$ a generátor $g$ vhodné grupy (třeba $\mathbb{Z}_p$). Alice vlastní privátní klíč $a$ a veřejný klíč $A = g^a \pmod p$, Bob vlastní privátní klíč $b$ a veřejný klíč $B = g^b \pmod p$. Následně se provede operace:

$$B^a = (g^b)^a = g^{ab} = (g^a)^b = A^b \pmod p.$$

#### Varianty D-F algoritmu

- Prvočíselná grupa $\mathbb{Z}_p$, tzv. *DHE*. Číslo $p \geq 2048b$ nebo se volí náhodně třeba $p = 2q + 1$ (safe prime).
- Grupa prvků eliptické křivky, tzv. *ECDHE*. Stačí $256b$.

### 6.2 Alternativní výměna klíčů

Lze provádět i bez DHE/ ECDHE a to za použití serverového certifikátu.
Klient si vygeneruje secret a pošle ho na server zašifrovaný pomocí klíče ze serverového certifikátu.
Nevýhodou však je, že to v TLS-1.3 možné a že nezajišťuje *forward secrecy* (komunikace zůstane bezpečná i v případě, že je jeden z klíčů kompromitován - nedojde ke kompromitaci předchozích zpráv).

### 6.3 HMAC - Hash Based MAC

Jedná se o symetrický podpis, kde se využívá hashování (hash dvakrát - kvůli length extension attack).
Alice s Bobem si dohodnou klíč $K$. Následně Alice Bobovi pošle zprávu $X$ a $Y$ =*hash*$(X, K)$*)*. Bob obdrží $X, Y$ a provede $Z$ =*hash*$(X, K)$*)*. Pokud $Y = Z$, tak víme, že zpráva byla v pořádku - nebyla pozměněna.

### 6.4 HKDF - HMAC based Key Derivation Function

Pomocí Diffie-Hellmana získáme sdílené tajemství a z něj odvodíme jednotlivé šifrovací klíče (vždy jiné klíče pro handshake, šifrování dat, směr komunikace). Využívá jako základní stavební blok HMAC a pracuje se ve dvou krocích:

1. *HKDF-Extract* - za pomoci HMAC se vstupní klíč spolu se *solí* převede na jiný klíč pevné délky (pseudorandom key - PRK).
2. *HKDF-Expand* - z PRK spolu s vstupní *informací* a hex indexem se opět provede HMAC a výstup se přilepí k následující *informaci*. Tedy: $\underbrace{(PRK;\ info\ \mid 0x01) \rightarrow\ HMAC}_{=Y} \Rightarrow (PRK;\ Y \mid info \mid 0x02) \rightarrow HMAC,\ \ldots$

# 7 Módy blokových šifer, GCM, CCM

## 7.1 ECB - Electronic Code Book

Každý blok se šifruje zvlášť - nezávisle na ostatních stejnou šifrovací operací. Velké úniky dat.

## 7.2 CBC - Cipher Block Chaining

Používá se IV. Každý blok zprávy je xorován s předchozím šifrovaným blokem před zašifrováním. (Na vstupu se IV XORuje s plaintextem a AES. Výstup se opět XORuje s plaitnextem a AES atd...)
Nevýhodou je, že neřeší integritu (řeší se přes HMAC). Je nutné dávat pozor na reuse IV. Složitá implementace (padding oracle)

## 7.3 CTR - CounTeR mode

Převod blokové šifry na proudovou.
Transformuje bloky vstupního textu na šifrované bloky. Šifrovací funkce je aplikována na hodnoty počítadla, které se zvyšují pro každý blok.
Šifrované bloky jsou poté xorovány s odpovídajícími bloky vstupního textu.

## 7.4 GCM - Galois CounTeR mode

Kombinuje *CTR* mód s *GMAC*em. Umožňuje to tak šifrování a ověřování integrity současně.
Využívá se tzv. autentizační tag $\in GF(2^{128})$, který je připojen k zašifrovaným datům a slouží k ověření, zda byla data modifikována během přenosu.

## 7.5 GHASH - Galois HASH

Je to režim šifrování kombinující *CTR* s *GMAC*em pro ověření integrity dat. GCM používá operace nad $GF(2^128)$.
Šifruje bloky dat pomocí CTR a následně vypočítá autentizační tag pomocí funkce GHASH, což umožňuje současně šifrovat a ověřovat integritu dat. Používá se nonce.
Převede vstupní data na několik bloků, které jsou následně XORovány s interními hodnotami klíče a násobeny konstantami, čímž vznikne autentizační tag.
Je AXU (almost XORed universal hash) a je to *skoro* hashovací funkce - méně požadavků (např. ne nutně collision resistant). Je zpravidla rychlejší než běžné hashovací funkce.
Výpočet nad $GF(2^{128})$ definovaným polynomem $x^{128} + x^7 + x^2 + x + 1$. Key $= H = E_K(0^{128})$. GHASH core je modulární násobení binárních polynomů Key a Input.

Jde paralelizovat.
Bloky vstupu $X_1, X_2, \ldots, X_n$. Klíč $H = E_K(0^{128})$. GHASH $= X_1 \cdot H^m \oplus \ldots \oplus X_{m-1} \cdot H^2 \oplus X_m \cdot H$.

## 7.6 GMAC - Galois MAC

Je to způsob generování autentizačního tagu používaný v GCM pro zajištění integrity a autenticity dat.
Využívá funkci GHASH, která kombinuje operace nad $GF(2^128)$ s klíčem pro výpočet autentizačního tagu.

## 7.7 CCM - CbC + Mac

Jedná se o *CTR* spolu s *CBC-MAC*. Vyžaduje dvě šifrování blokovou šifrou pro každý blok - je tedy výpočetně náročnější, pomalejší. (V praxi se v TLS příliš nepoužívá).

## 7.8 CBC-MAC

Je to autentizační kód vytvořený aplikací režimu *CBC* na zprávu s použitím symetrické šifry. Zpráva je rozdělena do bloků, které jsou postupně šifrovány pomocí symetrické šifry a výsledné šifrované bloky jsou XORvány s následujícími bloky, dokud se nedosáhne konečného autentizačního tagu, který slouží k ověření integrity dat.

### 7.9 ChaCha20-Poly1305

Kombinuje proudovou šifru *ChaCha20* (šifrování) s *Poly1305* (autentizace) pro ověření integrity dat. ChaCha20 šifruje zprávu proudem klíčů a Poly1305 generuje autentizační tag, který je připojen k zašifrované zprávě, což poskytuje integritu, autenticitu a důvěrnost dat.
Je rychlá i bez HW akcelerace;
*ChaCha20* je proudová ARX šifra založená na blokové šifře podobné AES (bloky $64B$, přímo zakomponovaný "CTR mode")
*Poly1305* - podobné GMAC

## 8 Druhy elektronického podpisu dle legislativy, časová razítka

### 8.1 Legislativa

- *eIDAS* - electronic IDentification, Authetication and trust Services. Jendá se o nařízení EU
- *Zákon č. 297/2016 Sb.* - Zákon o službách vytvářejících důvěru pro elektronické transakce
- *Zákon č. 250/2017 Sb.* - Zákon o elektronické identifikaci

**Prostý elektronický podpis** Je to forma e-podpisu, která není založena na kvalifikovaném certifikátu, ani na bezpečném zařízení pro podpis, ale může být generována a ověřována pomocí různých elektronických prostředků, jako jsou e-maily, zadání hesla nebo jiné metody...
Není tedy vyžadováno žádné kryptografické propojení s dokumentem.
Příkladem jsou třeba *nakreslený podpis (v editoru), obrázek podpisu, biometrický podpis*, ...
Mělo by jít vždy o něco unikátního a charakteristického pro danou osobu

**Digitální podpis** Je svázán konkrétním dokumentem. Nelze ho vytvořit dopředu a je nepřenosný.
Podepisující osoba rozhoduje o tom, co podepisuje.
Vše je založeno na kryptografii, využívá se typicky X.509 certifikáty.

**Zaručený elektronický podpis** Opět založený na X.509 certifikátu.
Je považován za právně ekvivalentní klasickému ručnímu podpisu a je využíván pro právně závazné transakce a dokumenty v souladu s evropskou legislativou eIDAS.
Je založen na certifikátu vydaném CA a vytvořen pomocí bezpečného podepisovátka (čipová karta s certifikovaným elektronickým podpisem).
Není zde žádná záruka identity, nejsou žádné požadavky na certifikát (lze využít i self-signed).

**Uznávaný elektronický podpis** Je potřeba kvalifikovaný certifikát.
Je uznáváný a akceptováný v souladu s právními předpisy dané země nebo regionu.
Nemusí splňovat všechny nároky na zaručený e-podpis.

**Kvalifikovaný elektronický podpis** Uznávaný elektronický podpis s privátním klíčem uloženým na kvalifikovaném prostředku pro vytváření elektronických podpisů.

**Kvalifikovaný certifikát** Certifikát vydaný kvalifikovanou CA. Podmínky jsou právě v zákoně 297/2016 Sb.
V ČR seznam spravuje DIA.

**Kvalifikovaní poskytovatelé certifikátů** například Pošta, eIdentity, Komerční banka, ...

**Kvalifikovaný prostředek** Obvykle HW zařízení pro bezpečné uchování privátního klíče - USB token, Čipová karta... Definuje eIDAS.

### 8.2 Versus

#### 8.2.1 Elektronický VS Digitální podpis

Elektronický podpis je označení různých metod podepisování elektronických dokumentů.
Digitální podpis je konkrétní typ elektronického podpisu. Je založen na asymetrické kryptografii. Digitální podpis je vytvořen pomocí soukromého klíče podpisovatele a ověřen pomocí odpovídajícího veřejného klíče.

### 8.2.2 Elektronický podpis VS pečeť

Podpis je fyzická osoba, naopak pečeť je právnická osoba (často navíc generováno strojově).

## 8.3 Časová razítka

U podpisu může být podstatné, kdy vznikl. Elektronický podpis platí tak dlouho, dokud platí certifikát.

**Časové razítko** - Ověření od CA, že daný dokument v daný čas existoval.
Podpis od speciální CA, ověřuje existenci dokumentu v daný čas, nic jiného.
Pokud chceme dlouhodobě udržovat elektronický podpis, musíme ho pravidelně "přerazítkovávat", jinak přijdeme o možnost podpis ověřit.

**Technicky** Funguje nad protokolem TSP. Při komunikaci s CA je možné využít i HTTP nebo e-mail.
*ASiC - Associated Signature Containers* - ZIP soubor, metadata v META-INF (např. podpisy)

# 9 PAdES, XAdES, CMS

Nařízení eIDAS definuje *Advanced Electronic Signature* (AdES). S tím úzce souvisí následující:

## 9.1 PAdES

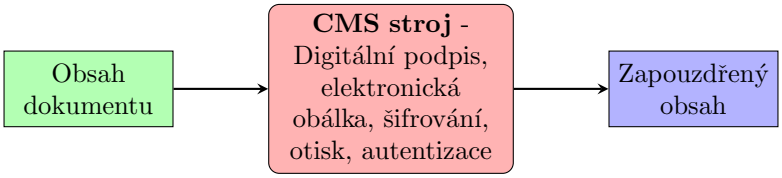
Neboli PDF-AdES. Definuje způsob, jak vložit elektronický podpis do PDF dokumentu tak, aby byl soubor s podpisem plně kompatibilní s právními požadavky, jako je eIDAS.
Nejčastější e-podpis. Je vložený do dokumentu. Podepisuje se vše až na atribut s podpisem.
Používá se Adobe-Reader, JSignPDF. Podpis lze dokonce doplnit i obrázkem.

## 9.2 XAdES

Neboli XML-AdES
Umožňuje vložit elektronický podpis do jakéhokoli typu XML dokumentu. XAdES poskytuje záruky autenticity, integrity a nezpochybnitelnosti pro podepsané XML dokumenty.
Před podpisem je nutná kanonizace dokumentu (odstranění přebytečných mezer). Lze si vybrat, jaká část dokumentu bude podepsaná.
Existují dva druhy podepsaných dat: *enveloping* (uvnitř podpisu) a *enveloped* (mimo tag s podpisem).

## 9.3 CMS (CadES) - Cryptographic Message Syntax

ASN.1 struktura využívající PKCS #7.
Definuje formát zprávy pro digitální podpisy, šifrování a další kryptografické operace. CMS je používán pro vytváření elektronických podpisů a šifrování dat v bezpečných komunikačních protokolech, jako je například S/MIME pro zabezpečené elektronické pošty.




# 10 Možnosti autentizace obecně, vícefaktorová autentizace, SSO

**Autorizace** souhlas/ schválení konkrétní operace daným subjektem (provedení platby)

**Autentizace** ověření identity subjektu (přihlášení do internetového bankovnictví)
Autentizovat se může *user*↔*PC* (přihlášení na web), *PC*↔*PC* (automatické zálohování), *PC*↔*PC+user* (přihlášení k Wi-Fi).

**Uživatelská autentizace** Uživatel musí buď *něco znát* (heslo/PIN), *něco mít* (HW token/ čipovou kartu/ SMS), *něčím být* (biometrie).

**Vícefaktorová autentizace** Pro ověření identity uživatele je využito více prostředků (SMS/ Google Authenticator).

## 10.1 SSO - Single Sign On


Potvrzení identity zajišťuje IdP (Identity provider). Co se týče operačních systémů, tak příkladem je Kerberos nebo NTLM. Co se týče HTTP služeb, tak příkladem je OpenID Connect nebo SAML.

**Webový SSO** Webová stránka *service provider (SeP)* požádá uživatele o autentizaci. Uživatel zašle na IdP požadavek na ověření identity (v případě potřeby se autentizuje). Následně od IdP dostane potvrzení své identity (vše je podepsané a případně i šifrované). Toto potvrzení uživatel předloží SeP

**SAML - Security Assertion Markup Language** Je to otevřený standard pro výměnu autentizačních a autorizačních dat mezi identitami SeP a identity IdP. Uživatel poté nemusí neustále zadávat přihlašovací údaje. Je založený na XML a často se využívá ve firemním prostředí.

**OpenID Connet (OIDC)** Rozšíření OAuth 2.0. Využívá JWT (JSON Web Tokens - podpis symetrická (HMAC)/ asymetrický (RSA/ ECDSA)) a je typickým řešením pro obecné weby. Používá například Google, Facebook, OpenID, MojeID, atd... *(viz. Obr).*

# 11 Přihlášení jménem a heslem, uchovávání hesla na serverové straně

## 11.1 Jméno a heslo

Typická situace na webu. Server je autentizovaný pomocí TLS certifikátu - uživatel se přihlásí - na základě autentizace se vytvoří session (za pomoci cookies)

**Nevýhody hesel** Uživatelé často volí slabá hesla (wordlisty a brute-force), recyklují je pro více služeb.
V případě útoku na server mohou být hesla ukradena.
Často se nacházejí chyby při obnově zapomenutého hesla.

**Bezpečnost hesla** Specifikoval NIST, OWASP, vyhláška o kybernetické bezpečnosti.
OWASP konkrétně specifikoval, jak by měl server přistupovat k uživatelským heslům:

- Požadovat alespoň 12 znaků (žádná max délka)
- Nezkracovat hesla (například nahrazení mezer)
- Povolit všechny speciální znaky (náročnější pro brute-force)
- Umožnit uživatelům změnu hesla - a vyžadovat v takovém případě staré heslo
- Při tvorbě hesla ověřit, jestli heslo není v TOP $X$ nejčastějších hesel.
- Nevynucovat pravidelnou změnu hesla a nevynucovat komplexitu (délka je důležitější)
- Umožnit zobrazovat zadávané heslo
- Nebránit kopírování hesel

Dále se radí nevyužívat bezpečnostní otázky, pro obnovu hesla zaslat jednorázový kód.

## 11.2 Brute-force a ochrana

**Brute-force útok** Máme hash, chceme získat heslo. Potřebujeme výkonný GPU, vhodný slovník a vhodné nástroje (hashcat, John the Ripper).

**Ochrana** Nejlépe prodlužovat intervaly dalšího přihlášení, CAPTCHA. Po úspěšném přihlášení informovat e-mailem. (Nedoporučuje se blokace IP ani zamknutí účtu).

**Password manager** Nejlepší ochrana, ideální řešení pro správu hesel.
Offline správci například *KeePassXC*, *pass*, ... - databáze hesel je lokálně uložená a při přihlášení je dešifrovaná v RAM.
Online správci například *Bitwarden*, *1Password*, ... - synchronizace cloudem, dešifrování databáze je lokální.

## 11.3 Ukládání hesel na serveru

Je třeba co nejvíce znepříjemnit útočníkovi se k heslům dostat. Nejlepším řešením je hesla hashovat a používat ideálně sůl/salt (náhodný řetězec unikátní pro každého uživatele). Tedy *hash(sůl||heslo)*.
Případně je možné použít i pepř/pepper (náhodný řetězec unikátní pro aplikaci - není uložený v DB s hesly).

**Výhody soli** Není poznat, že dva uživatelé mají stejné heslo. Nelze použít efektivně rainbow tables. Hodně zdržuje při brute-force (každý tip hesla je nutné pro každý hash spočítat zvlášť).

## 11.4 Hashování hesel

Běžné hashovací funkce mají jiné užití a jsou moc rychlé. My klademe trošku jiné nároky - co nejnižší kolize, ale může to trvat déle.
Vhodné funkce pro hashování hesel jsou například *PBKDF2* (velice často), *bcrypt, scrypt, Argon2* (vítěz 2015).

**PBKDF2 - Password Based Key Derivation Function 2** Nejen pro hashování, ale i pro odvozování klíče. Jedná se ve zkratce jen o opakované volání hash funkce (mnoho iterací). ´Funkce bere na vstup *PRF* (HMAC-SHA256), *heslo*, *c* (počet iterací) a *dkLen* (počet bitů výstupu).

**bcrypt, scrypt** Funkce navržená speciálně pro hashování hesel. Vyžaduje větší množství paměti a dá se škálovat dle spotřeby výpočetního výkonu a paměti.
Opět je lze využít jako KDF.

**Argon2** Je nejlepší volbou. Existují tři varianty:

- *Argon2d* - odolnější proti crackování, méně odolné proti side-channel
- *Argon2i* - méně odolné proti crackování, více proti side-channel
- *Argon2dd* - kombinace obou variant, takže 50/50

## 11.5 Operační systémy

Každý operační systém využívá jiný mechanismus. Nejlépe vychází BSD, které využívá *bcrypt*, dále macOS (PBKDF2 - HMAC - SHA512), Linux (sha512crypt - nestandardní) a nakonec tradičně katastrofa Windows (NT hash - vlastní mechanismus. Není solený, rychle se počítá, těžko nahraditelný, protože byl implementován dávno)

# 12 Asymetrický PAKE: SRP, OPAQUE

**PAKE** - Password Authenticated Key Agreement
Jedná se o vytvoření šifrovacího klíče z hesla. Konkrétně se asymetrický PAKE zaměřuje na ověření hesla bez toho, aby muselo být zasláno na server.

## 12.1 SRP - Secure Remote Password

Je to v podstatě "Diffie-Hellman doplněný o heslo". V praxi se používá nejčastěji (OpenSSL, TLS, ...).
Má hodně nevýhod - nutnost je pracovat nad okruhem $\mathbb{Z}_p$, protože nelze použít eliptické křivky. Tím se zároveň stává nekompatibilní s mnohými moderními protokoly. Dokonce je nekompatibilní s TLS-1.3. Další nevýhodou je, že vyzrazuje salt.

**Parametry** Volíme $p$ velké prvočíslo ve formátu $p = 2q+1$, kde $q$ je taktéž prvočíslo. Dále nechť $g$ je generátor$\mathbb{Z}_p$, $H$ je hashovací funkce a $k = H(p, q)$ je konstanta.
Uživatel zvolí heslo *pass* a spočte $x = H(salt,\ pass)$, *salt* je vygenerována náhodně. Dále spočte $v = g^x$, což je verifikátor hesla.
Následně se na server odešle *username= ID, salt* a $v$ (předání hodnot protokol neřeší).

**Autentizace** Nechť Alice ≈ User a Bob ≈ Host.
Alice pošle Bobovi *ID* a $A = g^a$, kde $a$ je náhodná hodnota.
Bob pošle Alici *salt* spolu s $B = kv + g^b$, kde $b$ je také náhodná hodnota. Oba následně vypočtou $u = H(A, B)$.
Nyní Alice spočte $x = H(salt,\ pass)$ a $S = (B - k\ g^x)^{a+ux}$ a Bob $S = (A\ v^u)^b$.
Oba nakonec vypočtou *session key* $K = H(S)$.

**Ověření session key** Klíč si sice oba vzájemně vyměnili, musí ale ověřit, že se shoduje. To se dělá následovně:
Alice pošle Bobovi: $M_1 = H\Big(H(p) \oplus H(g),\ H(ID), salt, A, B, K\Big)$.
Bob pošle Alici : $M_2 = H(A, M_1, K)$

**Platnost výpočtu**

Shrnutí: $A = g^a, B = kv + g^b, u = H(A, B), p = 2q + 1, k = H(p, q), v = g^x$

$$\begin{aligned} S_A &= (B - k\ g^x)^{a+ux} = (kv + g^b - k\ g^x)^{a+ux} = (\cancel{k\ g^x} + g^b - \cancel{k\ g^x})^{a+ux} = \\ &= (g^b)^{a+ux} = (g^{a+xu})^b = (g^a(g^x)^u)^b = (A\ v^u)^b = \\ &= S_B \end{aligned}$$

## 12.2 OPAQUE

Je to výherce PAKE soutěže a je založen na OPRF - Oblivious PRF. Aktuálně nejlepší variantou.

**PRF - PseudoRandom Funciton** Na vstupu je *klíč* a *hodnota* (libovolné délky). Na výstupu je naopak *řetězec* (fixní délky).
Neexistuje efektivní algoritmus pro rozlišení PRF od náhodného orakula.
Typickým příkladem je MAC.

**Oblivious PRF** Jedna strana provádí kryptografickou operaci bez znalosti vstupu. V praxi se to provádí takto:

- Zvolí se grupa $G$, kde je problém diskrétního logaritmu těžký
- *Alice:* vstup $x$, náhodný blinding factor $r$, `blinded_input` $= x^r$
- *Bob:* `blinded_output = blinded_input`$^k$, kde $k$ je tajný klíč
- *Alice:* `output = blinded_output`$^{(1/r)} = x^k$, kde $1/r$ je inverzní prvek k $r$ v grupě (výpočet rozšířeným Eukleidovým algoritmem).

# 13 FIDO2, OTP

## 13.1 FIDO2 - Fast IDentity Online 2

Na zařízení (USB) je uložený privátní klíč/ tajemství a pomocí něj lze podepsat challenge.
Využívá standardů WebAuthn, což umožňuje webovým aplikacím a službám komunikovat s autentizačními zařízeními (například biometrickými čtečkami otisků prstů nebo bezpečnostními klíči) bez nutnosti hesel.

**WebAuthn - Web Authentication** protokol pro autentizaci na webových stránkách implementovaný prohlížeči.

## 13.2 OTP - One Time Password

Příkladem je Google Authenticator. Obě strany nejprve nasdílí symetrický klíč (vyfocení QR) a následně spolu s *přidanými daty* vytvoří OTP (obvykle 6 ciferné heslo)

**HOTP - HMAC based OTP** Přidaná data tvoří *counter*. Vše závisí na vzájemné synchronizaci.
Není to moc praktické.

**TOTP - Time based OTP** Přidaná data tvoří *čas*. Výpočet je stejný jako v HOTP, ale místo counteru se použije čas $T = (currentTime - T_0)/X$, kde $X$ je krok (obvykle 30s) a $T_0$ je nějaký posun (default 0).
Při ověření je nutné vyzkoušet současnou i minulou hodnotu.

**OCRA - OAUTH Challange Response Algorithm** Přidaná data tvoří *challange*. Opět využívá vnitřně HOTP.
Challenge je často využívána v mobilních autentizačních aplikacích (Smart Klíč).

# 14 Symetrický PAKE: SAE, cPace

Na obou zařízeních je třeba zadat heslo. Využívá se často DF výměna klíče. Existuje mnoho algoritmů implementující Symetrický PAKE. Dva z nich si uvedeme:

## 14.1 SAE - Simultaneous Authentication of Equals

Jedná se o variantu Dragonfly Key Exchange fungující jako *hash-to-curve algoritmus*. Tedy je dána EC grupa a generátor grupy je odvozen z hesla (heslo → bod křivky).
Následně se provede Diffie-Hellman nad EC.
Nahrazuje PSK u WPA-Personal a používá se v WPA3.

## 14.2 cPace - Composable Password Authenticated Connection Establishment

Je vítězem soutěže z roku 2020 a je doporučený symetrický PAKE (je standardizován jako RFC).
Dvě zařízení si na základě společného hesla odvodí generátor nad nějakou cyklickou grupou.
Potom tyto dvě zařízení tento generátor využijí k provedení rozšířeného Difiie-Hellmana.
Odvodí si tedy generátor $h$ a spočtou $x$ tak, aby splňovalo $g^x = h$ (aby neznali diskrétní logaritmus).
Session key se nakonec odvodí z výstupu DF spolu s veřejnými rozšířenými klíči a UID.
Můžeme si všimnout, že posláním grupy prvků jakožto část handshaku znamená, že posíláme veřejný klíč, který je ovšem asociovaný s privátním klíčem a potřebujeme tedy znát heslo.

# 15 Bezpečnost Wi-Fi: OWE, WEP, WPA, WPA2, WPA3

Zprostředkovává bezdrátové připojení k internetu. Není to zkratka - nic to neznamená. Pracuje se standardy *IEEE 802.11* a spravuje je Wi-Fi Alienace.
Dříve nebylo doslova žádné zabezpečení Wi-Fi, nebylo potřeba. Byla kompletně otevřená (*open*) - kdokoli se mohl připojit, žádné šifrování paketů (kdokoli mohl poslouchat komunikaci)
Až později, s nárůstem zájmu, se začala zabezpečení vyvíjet.

## 15.1 OWE

Jedná se o alternativu *Open AP*.
Je jako SAE, ale bez hesla, tedy jen Diffie-Hellmann. Není zde ale žádná ochrana před MitM - útočník může vytvořit falešný AP.

## 15.2 WEP - Wired Equivalent Privacy

Standard z roku 1997. Pracuje se sdíleným PSK klíčem a používá nekvalitní RC4 šifrování. Navíc klíč často byl jen ASCII znaky (kvůli zmenšení prostoru) a komunikace se všemi zařízeními byla prováděna tím jedním klíčem.

**RC4** Velice jednoduchá a rychlá proudová šifra. Generuje pseudonáhodný keystream a používá následně k němu přiXORuje plaintext. Ke generování keystreamu používá šifra vnitřní stav, který tvoří pole bytů $S$ proměnné $i, j$.

**Autentizace** Jsou dva druhy.

- *Open System authentication*: není autentizace žádná, přesto ale může být komunikace šifrovaná. Kterýkoliv klient se tak může s přístupovým bodem ověřit a pokusit se o spojení, protože nedochází k žádné autentizaci. WEP klíče mohou být následně používány pro zakódování datového rámce, když bude mít klient správný klíč.
- *Shared Key atuhentication*: Vše probíhá v rámce challenge-response (handshake).

  Server pošle klientovi náhodnou challenge (v plaintextu), klient přijatou challenge zašifruje (WEP klíčem) a pošle zpět (response). Na závěr server dešifruje přijatou response a pokud se shoduje s dříve odeslanou challenge, odešle pozitivní odpověď.

  Nevýhodou je fakt, že lze komunikaci odposlechnout - konkrétně je možné odvodit klíč použitý k handshaku ze zachycení response.

  Dokonce jde snáze prolomit než Open Systen authentication.

## 15.3 WPA, WPA2, WPA3

Nejprve dojde k *autentizaci* a výsledkem je PMK (Pairwise Master Key). Dále se provede *4-way handshake* pro ustanovení šifrovacích klíčů a nakonec už dochází k *šifrované komunikaci*.

**WPA-Personal** využívá PSK. Typicky pro domácí použití a obecně méně zabezpečené sítě. Na konci se odvodí 256*b* PMK z *HMAC-SHA1, zakódovaného PSK, SSID,* 4096 *iterací*.

**WPA-Enterprise** drž se standardu 802.1X a autentizuje za pomoci externího serveru RADIUS. Je mnoho možností autentizace (jméno+heslo, certifikát, ...). Na konci PMK.

**4-way handshake** Z klíčů jsou postupně odvozovány další podklíče (šifrování, podepisování). AP pošle PC požadavek, PC odvodí PTK a pošle ho AP, ten ho přijme, odvodí GTK a pošle ho PC, ten nakonec pošle AP potvrzení o přijetí.
*PTK - Pairwise Transient Key* - pro šifrování komunikace klient↔AP.
*GTK - Group Transient Key* - broadcast a multicast komunikace

**WPA2** Od roku 204 a do dnes je běžně používána (nejhojněji). Je zde šifrování i integrita CCMP (AES-CCM, 128*b*).

**WPA3** Od roku 2018 a stále se spíše začíná využívat. Šifrování je AES-GCM 256*b* a HMAC-SHA384. Namísto PSK se nově používá SAE. Umožňuje forward secrecy.

## 15.4 WPS - WiFi Protected Setup

Protože opisování hesla je zdlouhavé a může vést k volbě slabých hesel, vzniklo WPS.
Nejprve se ověří 8 místný PIN, poté se na AP zmáčkne tlačítko (*push button*) a PIN se na krátkou dobu - do autentizace prvního zařízení - nastaví na 0000 0000.
Po ověření se odešlou přístupové údaje.

**Zranitelnost** AP informuje o správnosti každé půlky PINu, protože poslední cifra je vždy kontrolní. Je tedy možné hrubou silou s maximálně 11000 pokusy uhádnout PIN.

**DPP - Easy Connect** Alternativa WPS u WPA3. Je kryptograficky v pořádku - výměna klíče probíhá za pomoci DF a je zapotřebí QR, Bluehooth, NFC, atd.
DF tajemství tedy vytvoří šifrovaný tunel, kterým jsou zaslány přístupové údaje k Wi-Fi. Je to hodně složitý protokol a má mnoho možností.

# 16 Párování v Bluetooth, SAS

## 16.1 Bluetooth

K dispozici nezabezpečený kanál pro komunikaci mezi zařízeními.
Uživatel tvoří důvěryhodný kanál, ale nechce opisovat dlouhé kódy.
Často omezení na straně zařízení (žádná klávesnice/ displej).

**Just Works** Nejjednodušší, rychlá a snadná metoda připojení - pouze se zmáčkne tlačítko, nic se neověřuje. Tzv *TOFU - Trust On First Use*. Příkladem jsou sluchátka.

**Passkey Entry** Je zapotřebí zadání hesla (*passkey*) na jednom zařízení a následně jeho potvrzení na druhém. Jakmile je vše potvrzeno, máme spárováno.
Příkladem je propojení mezi dvěma telefony - obvykle je vyžadováno potvrzení hesla.

**Numeric Comparsion** Umožňuje uživatelům ověřit, že párovaná zařízení jsou autentická a že párování je prováděno bezpečně.
Nejprve se vygeneruje náhodné číslo a zobrazí se na jednom obou zařízeních. Následně je potřeba, po porovnání, potvrdit, že se shodují. Opět je příkladem propojení mezi dvěma telefony.

**OOB - Out of Band** Využívá alternativního komunikačního kanálu k výměně klíčů pro šifrování, například *NFC, QR kód, Wi-Fi Direct...* Umožňuje generovat klíče přímo v zařízeních, přenášet je mimo samotný kanál Bluetooth, což zvyšuje bezpečnost párování (prevence MitM)

### 16.2 SAS - Short authenticated string

Jedná se o bezpečnostní mechanismus používaný při párování zařízení, je prevencí třeba před MitM. Je to krátký řetězec znaků, který je zobrazen na jednom zařízení a potvrzen na druhém zařízení jako součást procesu ověření při párování.
Funguje jako *Numeric Comparsion* s tím rozdílem, že může jít i o string, ne pouze číslo. Klíč se přenese pomocí DH (typicky ECDH).

**Přenos klíče** Naivním způsobem bychom se na to mohli podívat takto:
Alice pošle $PubA$ Bobovi a ten pošle alici $PubB$. Nicméně to je stále náchylné k MitM.
Když si Ema stoupne mezi ně, může zachytit $PubA$ a poslat Bobovi $PubE_1$, ten naopak, s domněním, že posílá klíč Alici pošle $PubB$ Emě a ta pošle $PubE_2$ Alici. Neboli Ema má oba potřebné klíče a může bez problému zasahovat do komunikace.

Řešením je tzv. *commitment* klíče. Tedy:
Alice zašle *hash(PubA)* bobovi, ten zašle zpět $PubB$ a nakonec pošle opět Alice $PubA$. Bob si tak může jednoduše ověřit, že se skutečně jedná o ten samý klíč. Kdyby ne, tak se hash nebude shodovat.

## 17 Popis protokolu NTLM

**LM hash** Dnes se, až na výjimky, nepoužívá, protože je slabý. Maximální délka hesla je 14 znaků a navíc každých 7 znaků se hashuje zvlášť (*ještě k tomu DESem OMG*).

**NT hash** Umožňuje libovolnou délku hesla, není ale solený. Jedná se o lepší hash, ale stále je to rychlý výpočet (používá se MD4)

### 17.1 Protokol NTLM

Jedná se o autentizační protokol vyvinut Microsoftem. Ověřuje uživatele při přihlašování do sítě/ k tiskárnám/ sdílené složky apod.

- *Negociate* - když se chceme autentizovat
- *Challenge* - předáme 64*b* náhodnou hodnotu
- *Authenticate* - podpis challenge za pomoci NT(LM) hashe jako klíče.

Existuje mnoho variant NTLM response: *LM* (pro LM hashe, nepoužívat), *NTLMv1* (pro NT hash, nepoužívat), *NTLMv2* (NT hash, dnes nejlepší), *LMv2* (NT hash, podobné NTLMv2, kvůli kompatibilitě)

Mezi úskalí protokolu NTLM se řadí Pass the Hash a NTLM Relay (o tom později).

**Ověření u lokálního účtu** Cílový server zná NT hash toho, kdo se přihlašuje (SAM), takže si snadno spočítá stejnou odpověď a odpovědi porovná.

**Ověření u doménového účtu** NT hash přihlašovaného zná jen DC (Domain Controller). Využití Netlogon service - Zajistí se Secure Channel serveru s DC, autentizuje se pomocí machine account serveru. Server pošle *uname, challenge, response* a DC vše ověří. Pokud je vše bez problému, tak server pošle i informaci o uživateli (stejné jako v PAC).

## 18 Pass the hash, NTLM relay: princip a způsoby obrany

### 18.1 Pass the Hash

Nastávájí problémy, pokud více strojů ze stejné šablony má stejné heslo lokálního administrátora (stačí používat jiné heslo/ LAPS). Nebo je rizikem ovládnutí privilegovaného účtu.
Prakticky se jedná o command line program `smbclient`.

**Princip** Útočník získá hash hesla uživatele místo samotného hesla. Poté, ho může použít k přihlášení na jiných zařízeních nebo službách, které používají stejné heslo, aniž by potřeboval znát samotné heslo.

**Obrana** Hlavní obranou proti útoku typu Pass the Hash je používání jedinečných a silných hesel, která jsou odolná proti reverznímu inženýrství. Také je důležité minimalizovat práva uživatelů, což omezuje dopady, pokud útočník získá přístup pomocí ukradeného hashe.

### 18.2 NTLM relay

**Princip** Útočník zachytí autentizační provoz mezi dvěma PC a přenese ho na jiný cílový PC. Útočník tedy využívá relay autentizačního provozu k získání přístupu k systému nebo službám.

**Obrana** Používání bezpečnějších autentizačních mechanismů (Kerberos), nastavení firewallů a filtrů.

## 19 Kerberos

### 19.1 SPN - Service Principal Name

Jedná se o označení service v rámci AD (Active Directory). *Service* je služba, kterou mohou využívat ostatní.
Speciální třída HOST Je to skupina mnoha tříd služeb. Lze konfigurovat, jaké služby zahrnuje a typicky zahrnuje služby běžící pod machine account stroje.
Pro privilegované operace je typicky potřeba využít specifickou třídu.

### 19.2 Protokol Kerberos

Vznik v 80. letech na MIT. Na MS Windows je od Windows 2000 defaultní autentizační protokol a jinak je implementován i v mnoha UNIXových prostředích (Linux, BSD, macOS, ...).

#### Jak to funguje

Vše stojí na SSO. Komunikace probíhá mezi *klientem*, *cílovým serverem* a *KDC* (Key Distribution Center, v prostředí Active Directory bývá DC)

**AS - Authentication Service** Ověření uživatele vůči KDC. Získání *Ticket Granting Ticket = TGT*.

- Požadavek: `KRB_AS_REQ` = Kerberos Authentication Service Request - Obsahuje jméno a timestamp zašifrovaný heslem uživatele. Na KDC se vygenruje unikátní session klíč.
- Odpověď: `KRB_AS_REP` = Kerberos Authentication Service Reply - Session key zašifrovaný heslem uživatele. TGT je šifrováno heslem. TGT obsahuje (uname, platnost, session key, PAC)

**TGS - Ticket Granting Service** Zajišťuje získání *ticketu* pro připojení ke službě. Služba označená pomocí svého SPN. Je to opět komunikace uživatele s KDC.
Je šifrován heslem služby, obsahuje *username, timestamp, PAC, session key pro službu.*

- Požadavek: `KRB_TGS_REQ` - TGT, SPN cíle, Authenticator (uname + timestamp, šifrováno session key). *Uživatel ověřuje znalost session key, ne TGT*
- Odpověď: `KRB_TGS_REP` - nový session key pro službu

**AP - Application Request** Je to autentizace vůči službě. Komunikace uživatel ↔ služba
Jedná se o stejný princip jako při autentizaci pomocí TGT, takže:

- Požadavek: `KRB_AP_REQ` - TGS, Authenticator využívající session klíč z TGS. Ověření jako u TGS, uživstele ověřuje znalost session key pro službu
- Odpověď: `KRB_AP_REP`

**PAC - Privilege Attribute Certificate** Rozšíření Kerberosu od Microsoftu, které se stará o user mangement. Služba často rozhoduje, zda má uživatel právo přístupu, jen na základě PAC.

#### Únik hesla

Můžeme si TGT/TGS sestavit sami. Heslo účtu, pod kterým běží služba (TGS) → *Silver ticket.*
Heslo účtu krbtgt (TGT) → *Golden ticket*
Do PAC pak lze uvést cokoli (třeba, že uživatel je administrátor).

**Dvojitý podpis PAC** Podpisy pomocí hesla *účtu*, pod kterým běží služba (v TGS), nebo *krbtgt*.
Služba nemůže sama zkontrolovat podpis krbtgt (typicky ho nekontroluje a Silver tickety fungují)

# 20 Kerberoasting, AS_REP roasting, Kerberos delegation

Jedná se o Bruteforce útoky na Kerberos.

## 20.1 AS_REP Roasting

V `KRB_AS_REQ` nevyžadovat (pre)autentizaci (šifrovaný timestamp). Uživatel nezná heslo, takže nedokáže dešifrovat session key patřící k TGT.
Pravděpodobně kvůli kompatibilitě se staršímy UNIXy.

Důsledek: kdokoli může získat data šifrovaná uživatelovým heslem, tedy lze crackovat heslo

## 20.2 Kerberoasting

Každý uživatel může zažádat o TGS pro libovolnou službu. Jestli k ní má přístup, rozhoduje až služba na základě PAC. Dokonce stačí, aby bylo v AD registrováno SPN, služba už nemusí existovat.

Důsledek: lze získat data šifrovaná jakýmkoli účtem s registrovaným SPN, tedy lze crackovat heslo.

**Obrana** Na rozdíl od `AS_REP` roasting je to otázka designu protokolu. V principu nelze zabránit.
Proto: služby spouštět pouze pod účty se silnými hesly a hesla pravidelně měnit. A nespouštět služby pod velmi privilegovanými účty (i z jiných důvodů).

## 20.3 Kerberos delegation

Občas se hodí, aby služba mohla přistupovat k dalším službám jménem uživatele - třeba přístup na sdílený disk. Tedy potřeba SSO.
Příklad delegation webové rozhraní pro přístup k fileserveru.

**Unconstrained Delegation** Aby bylo možné:
- delegující server musí mít nastaven UAC flag na `TRUSTED_FOR_DELEGATION`.
- uživatel nesmí mít nastaven UAC flag na `NOT_DELEGATED`.
Pak je do TGS přidán TGT a přislušný session key.
Server si zapamatuje TGT + klíč a s nimi může žádat o libovolné TGS tickety pro libovolné služby.
Uživatel tedy nemá vůbec žádnou kontrolu, jak s jeho TGT server nakládá (je proto nutná důvěra v server)
Je to v podstatě analogické NTLM.

**Constrained Delegation** Server smí impersonovat uživatele pouze pro specifokované SPN.
Uživatel se chce autentizovat ke službě $A$, ta ho chce impersonovat při připojení ke službě $B$.
Autentizace pro službu $A$ klasicky pomocí TGS.
Služba $A$ si pak vyžádá od KDC TGS ticket pro službu $B$ pomocí rozšíření S4U2Proxy, vyžadován TGS ticket pro službu $A$.
KDC (DC) spravuje seznam povolených delegací a uživatel nesmí mít flag `NOT_DELEGATED`

**Resource Based Constrained Delegation (RBCD)** Změna, kdo hlídá omezení. Nyní má služba seznam účtů, které se mohou připojit a identifikovat jako někdo jiný.
Důsledek za předpokladum že znám machine account stroje $A$: Pokud ovládnu machine account $B$, mohu u něj nastavit atribut pro RBCD a se znalostí machine accountu $A$ získat ticket administrátora na stroji $B$.
Vhodná kombinace s NTLM relay na LDAP.

**S4U2Self** Je třeba jako additional-tickets poslat TGS ticket uživatele pro službu $A$.
Pokud se uživatel přihlásil pomocí NTLM/ přes web/ ..., tak umožní službě $A$ si vyžádat TGS pro sebe pro libovolného uživatele.
Je nutný flag `TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION`
Důsledek S4U2Self
- kompromitován účet s nastaveným flagem `TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION` - kompromitovány všechny služby běžící pod tímto účtem (útočník si může vytvořit TGS pro administrátora každé ze služeb)

# 21 SMTP, ochrana před podvržením e-mailu

## 21.1 SMTP - Simple Mail Transfer Protocol

Jedná se o textový protokol pro zasílání e-mailů.
Základní příkazy `HELO/EHLO`, `MAIL FROM`, `RCPT TO`, `DATA`, `QUIT`, ...
Existuje i rozšíření *ESMTP*, které ještě obstarává autentizaci a umožňuje navázat TLS spojení.

**E-mailová zpráva** Skládá se z hlavičky + zprávy. V hlavičce se opakuje informace `From` a `To` z protokolu SMTP (pro doručení jsou zásadní ty z SMTP, ale klient obvykle ukazuje ty z e-mailu).
Hlavičky: `From`, `To`, `Bc` (příjemci kopie), `Reply-To` (adresa pro odpověď), `Subject`, `Date`, ...

## 21.2 MIME - Multipurpose Internet Mail Extensions

Přidává podporu znaků mimo ASCII a různá kódování, dále třeba příloh nebo HTML e-mailů.
Má vlastní hlavičku obsaující verzi, typ (text/plain), kódování (Base64), disposition (inline/ příloha).

### S/MIME - Secure/MIME

Obstarává E2E šifrování. (alternativou by mohlo být třeba PGP).
Zajišťuje podpisy, šifrování a je založeno na CMS. Využívá X.509 certifkátů vystavených pro osobu. Často se využívá ve firemním prostředí.

## 21.3 Zabezpečení

Protokol nenabízí žádnou kontrolu, kdo zprávu odeslal a podepisování zpráv není příliš rozšířené.

**SPF - Sender Policy Framework** Je řešením, jak zabezpečit e-mail před podvrhnutím zprávy.
Určuje, jaké stroje (IP) mohou za doménu posílat e-maily. Můžeme nastavit libovolný počet pravidel s modifikátory (`+PASS` - zpráva je důvěryhodná, `-FAIL` - zprávu zahodí, ?NONE, ∼`SOFT FAIL` - zprávu doručí, ale označí za podezřelou).
Můžeme si zvolit z mnoha pravidel - IPV4/6, vše, atd.

**DKIM = DomainKeys Identified Mail** Alternativa SPF. Také má vlastní hlavičku. Přidá elektronický podpis připojený serverem (MSA). Veřejný klíč je jako DNS TXT záznam v ASN.1 struktuře.
Pro ověření podpisu je nutnost DKIM DNS záznam.

**DMARC - Domain-based Message Authentication, Reporting and Conformance** Jak jsou využívány SPF a DKIM (bez DMARC nelze zjistit, že doména využívá DKIM). Definuje akce při selhání hlaviček.

**IMAP a POP3** Jedná se o textové protokoly zajišťující stahování pošty z MDA.
Je k dispozici volitelné šifrování pomocí TLS (buď od začátku, nebo příkazem `STARTTLS`
*IMAP* je modernější a dnes častěji využívaný.

**Open relay** Pokud MTA přeposílá poštu na všechny domény, lze ho zneužít pro posílání spamu. Proto dnes MTA typicky doručuje e-maily pouze pro své domény.
Odeslání pošty pro libovolnou doménu je možné obvykle jen přes MSA s využitím autentizace.

# 22 PGP, web of trust

## 22.1 PGP - Pretty Good Privacy

Jedná se o první více rozšířený algoritmus (standarde OpenPGP) pro E2EE
Dnes se nejčastěji používá svobodná implementace *GPG* (GNU Privacy Guard).
Stále se jedná o rozšířený standard. Má hojné využití v e-mailu a podepisování softwaru.

#### Šifrování pomocí PGP

1. uživatel vytvoří zprávu, komprese
2. náhodný symetrický klíč, zašifrovat zprávu
3. symetrický klíč asymetricky zašifrován pro každého příjemce jeho veřejným klíčem
4. vše spojit do jedné zprávy, rozeslat
5. příjemce najde zašifrovanou variantu klíče pro sebe, dešifruje symetrický klíč, dešifruje zprávu

**Podepisování pomocí PGP** Probíhá ještě před šifrováním. Standardně má každý uživatel dva asymetrické klíče - jeden využívaný ostatními pro šifrování symetrických klíčů a druhý na podepisování zpráv.

**PGP a e-mail** Standardně nejsou šifrovány hlavičky (existuje rozšíření pro šifrování předmětu). Je podporováno nativně řadou e-mailových klientů (Thubnderbird), ale případně lze data šifrovat/dešifrovat i mimo klienta.

**PGP technicky** Typicky se využívá *RSA, AES* v základních módech. Data jsou v binární podobě a využívá se *base64* kódování + hlavičky.

**Chyby PGP** Protože je PGP už poněkud zastaralý, není ve všech ohledech ideální.
Nepodporuje moderní algoritmy, nemá *authenticated encryption* (proto raději využívat s podpisem) a není forward secrecy.
Velkou nevýhodou je princip *sign than encrypt*. Lze totiž vzít podepsanou zprávu a přeposlat někomu dalšímu. Ten bude mít pocit, že je původním adresátem.

### 22.2 WOT - Web of trust

Jedná se o decentralizovanou síť ověřující věrohodnost veřejného klíče.
Stará se o distribuci veřejných klíčů v PGP. Existují veřejné databáze PGP klíčů (*Key registers*), kam každý může nahrát svůj klíč a důvěru zprostředkuje WoT.

**Key Revocation Certificate** Slouží k revokování klíče v key registers pro případ, že ztratíme přístup k privátnímu klíči (smazání; zapomenutí hesla, kterým je šifrován)

#### Saltpack

Jedná se o alternativu PGP. Principiálně podobné PGP, ale řeší řadu jeho nedostatků.
Je založen na modernější kryptografii (vždy authenticated encryption) a řeší problém s přeposláním zprávy.
Využívá se aktuálně například v keybase.io nebo v keys.pub.

#### OTR = Off-The-Record

Opět alternativa PGP. Slouží pro XMPP (Jabber) a podporuje forward secrecy.

## 23 Signal protocol

Aplikace Signal vznikla v roce 2010 pod názvem *TextSecure*, později došlo k přejmenování na Signal.
Dodnes je jediným oficiálním klientem Signal protokolu. (Jinak je ale protokol využíván i ve službách jako je WhatsApp, (volitelně) Messenger, Skype, atd.)
Jako KDF se využívá *HKDF, HMAC*, jako hash *SHA256, SHA512*.

### 23.1 TOFU - trust on first use

V signal protokolu se namísto WoT využívá TOFU - je uživatelsky přívětivé *(mohu psát rovnou každému)*.
Při prvním použití věříme veřejnému klíči druhého uživatele. Příště se klíč ověří dle svého záznamu. Klíč lze zpětně ověřit (např. při osobním setkání).

### 23.2 Handshake: X3DH - Extended Triple Diffie-Hellman

Standardně je *forward secrecy* pomocí DF algoritmu. DF ale vyžaduje interaktivnost - tedy Alice nemůže použít DH při zahájení komunikace s Bobem, který je offline. Proto Signal zavádí vlastní neinteraktivní výměnu klíče - X3DH.

**Klíče v X3DH**

- *Identity key* - je dlouhodobý klíč reprezentující uživatele (jako PGP, S/MIME, ...)
- *One-time prekeys* - předpřipravené klíče na jedno použití, uživatel nahraje na server dopředu
- *Signed prekey* - klíč pro případ, že one-time prekeys dojdou (forward secrecy alespoň do jejich poslední rotace). Je podepsaný pomocí Identity key.

Sending key Alice = receiving key Boba.

**X3DH** Jedná se o 3 nebo 4 DH výměny klíče. Využijí se všechny klíče Boba (identity, signed, one-time), identity klíč Alice a dočasný klíč vygenerovaný Alicí.
Alice ověří podpis Bobova signed prekey za pomocu identity key a dočasného klíče apod.
Výsledkem je po KDF Session key.
Význam handshaků: proběhne 2×DH s identity key, což zajistí vzájemnou autentizaci. A zbylé 2×DH zajišťují forward secrecy.
Když Alice Bobovi pošle veřejnou část dočasného klíče, tak Bob může udělat úplně stejné odvození session klíče.

**Double ratchet** Klíč odvozený z *X3DH* je využíván, dokud není konverzace smazaná, proto je přidaná forward secrecy i během posílání zpráv.
Navíc je zde snaha o zotavení, pokud byl klíč ukraden.

**Symmetric ratchet** Před každým odesláním zprávy Alice zavolá KDF a odvodí nový klíč. Analogicky Bob s receiving chain v opačném směru. To nám řeší forward secrecy.

### 23.3 PCS - Post Compromise security

Jedná se o zotavení z kompromitace klíče. Nový veřejný klíč s každou zprávou. Je potřeba přidat novou entropii. Využívá se k tomu DH.

**DH ratchet** Je to takový "ping-pong" s klíči. Slouží jako ochrana před útočníkem, který jednorázově získá private keys jedné strany (pro udržování platného klíče je potřeba sledovat celou komunikaci + aktivní MitM, pokud útočník chvíli nedokáže komunikaci sledovat/upravovat, o platný klíč přijde)
S každou zprávou nový veřejný klíč pro DH.

**DH ratchet schematicky (1)** Ve zkratce - výstup z DH je využit jako vstup do KDF.

- Alice zná Bobův $PubB_1$, vygeneruje nový DH klíč $PubA_1$ a ten pošle se zprávou Bobovi.
- Bob dostane $PubA_1$, vygeneruje nový DH klíč $PubB_2$ a ten pošle se zprávou Alici.
- Alice opět dostane Bobův public key $PubB_2$, vygeneruje nový DH klíč $PubA_2$ a ten pošle se zprávou Bobovi
- ...

## 24 Group chat v Signalu a MLS

**Skupinové konverzace tradičně**

Například *iMessage* problém řeší tak, že aplikace zašifruje zprávu pro každého uživatele ve skupině zvlášť.
Jinak u *Signal, WhatsApp, Keybase* má každý uživatel vlastní "sender key", kterým šifruje zprávy pro celou skupinu.

Ovšem obvykle má jak příjemce tak vlastník více zařízení, což způsobuje další problémy.
Keybase to řeší tak, že klíč uživatele je sdílený mezi jeho zařízeními. Signal tak, že zašifruje zprávu pro každé zařízení (vlastníka i příjemce) zvlášť a zvolá se jedno hlavní zařízení - mobil (jinak by mohly nastávat problémy s ověřením TOFU klíče)

*Existuje lepší varianta TOFU, která se skupinovými chaty pracuje. Vyvíjí ji Google pod názvem* Key Transparency. *Je to podobné certificate transparency a je stále ve vývoji.*

**Matrix** Matrix se od Signalz, WhatsAppu a Keybase liší tím, že je decentralizovaný.
Šifrování je inspirováno Signal protokolem, používá *identity* a *one-time* klíče a využívá i *Double ratchet*.

## 24.1 MLS - Messaging Layer Security

Je standardem a je inspirovaný Signal protokolem,
Slouží pro skupinovou konverzaci FS i PCS. Základní jednotkou je skupina *(ne One-2-One chat)* a je reprentována stromem klíčů, v jehož listech jsou klíče jednotlivých členů.
To umožňuje lepší škálovatelnost pro velké skupiny.

**Princip** Pro 1To1 kanály je třeba každou zprávu zašifrovat $N-1$ krát.
1To1 kanálem zašle každý každému svůj "sender key" (máme tedy $\mathcal{O}(N^2)$ zpráv).
Pak je zpráva šifrována jen jednou. Nicméně je tu problém s odebíráním ze skupiny, protože je třeba změnit všechny sender keys. Řeší se to za pomoci tzv. *ratchet tree*.

**MLS ratchet tree** Všechny sender keys jsou uspořádány do binárního stromu, kde uživatelé jsou v listech a každý uživatel zná privátní klíče svých předků.
Změna členství ve skupině tak proběhne v $\mathcal{O}(\log(N))$: U grafu se všemi listy je složitost zjevně logaritmická. Když některé listy chybějí, nejhůře lineální.

# 25 Bitcoin

Bitcoin vznikl v roce 2008. Založila jej anonymní skupina pod pseudonymem „Satoshi Nakamoto“. Jedná se o první rozšířenou kryptoměnu, která dosáhla nejvyšší tržní kapitalizace a stala se významným hráčem na poli digitálních platidel.

**Bitcoin adresa** Je odvozena z veřejného klíče uživatele. Každý uživatel má svůj *ECDSA klíč*. Adresa je vypočítána jako SHA256 hash veřejného klíče, který je poté zakódován pomocí *base56Check* (speciální kódování vylučující některé znaky kvůli podobnosti - $0OIl+/$, a zahrnuje navíc $4B$ kontrolní cifry.

## 25.1 Transakce

Transakce v Bitcoinu jsou zpravidla vyjádřeny jako „posílám $X$ BTC na adresu $Y$“ a jsou podepsány soukromým klíčem odesílatele. Ve skutečnosti jsou transakce reprezentovány skriptem (velmi omezeným), který určuje podmínky, za kterých se platba uskuteční.
(Na rozdíl od Bitcoinu používá Ethereum jazyk s větší flexibilitou pro definici podmínek transakcí a kontraktů $\implies$ inteligentní smlouvy, složitější scénáře provádění transakcí - tzv. *smart contracts*.)

- Transakce obecně může mít více zdrojových i cílových adres.
- Za každou transakci je účtován poplatek - platí ten, kdo transakci potvrdí.
- Veřejný klíč je třeba zveřejnit až ve chvíli, kdy má z adresy odejít platba (kvůli ověření podpisu) - (dříve se nezveřejňuje, aby ECDSA nepoložil celý systém)
- *(UTXOs)*: Neexistuje databáze zůstatků pro jednotlivé adresy, ale jen historie transakcí

**Těžba bloku** spočívá v nalezení takové hodnoty nonce, která zajistí, že výsledný hash bloku (2xSHA256) začíná dostatečným počtem nul. Tato podmínka je dynamicky upravována podle výkonu celé sítě, aby se udržovala průměrná doba mezi nalezením bloku na přibližně 10 minutách.
Těžba vyžaduje velké množství výpočetní síly, protože to umíme jen hrubou silou. Lze využít specializovaný hardware, například ASIC, který je efektivnější než běžné počítačové procesory.

## 25.2 Blockchain

je veřejný seznam vytěžených bloků, který obsahuje všechny provedené transakce v síti. Jedná se tedy o neustále rostoucí seznam transakcí (aktuálně $\sim 400GB$), který je strukturován do bloků. Je navržen tak, aby byl "append only", nelze tedy měnit již existující bloky.

**Řešení kolizí** v blockchainu se týká situací, kdy dva těžaři vytěží nový blok téměř současně. V takovém případě může vzniknout více než jeden možný řetězec bloků, který by mohl vést k rozštěpení blockchainu. Řešením je princip známý jako *"longest chain with the most proof of work"*.

**51% attack** nastává, pokud někdo vlastní 51% výpočetní kapacity. V takovém případě má vždy nejdelší chain a získává absolutní kontrolu nad sítí (může přepisovat historii.)
To umožňuje útočníkovi provádět *double spend* - může použít stejné prostředky dvakrát tím, že nejprve pošle platbu a poté přepíše historii tak, aby transakce nebyla zahrnuta v blockchainu.

**Merkle tree** je stromová datová struktura, která je používána v blockchainu k efektivnímu uložení (šetří místu a čas) a ověření transakcí. Není tak potřeba vždy stahovat všechna data.
Obsahuje pouze jeden (kořenový) hash. Pokud se nějaká transakce v bloku změní, změní se i kořenový hash Merkleova stromu. To rychle detekovuje jakékoli změny v obsahu bloku.

**Proof of Work (PoW)** Zajišťuje *distribuci kryptoměny* (vytěžené mince jsou spravedlivě rozděleny těm, kteří vynaložili práci na těžbě nového bloku) a *limitaci zápisů do blockchainu*, neboli omezuje počet transakcí.
Nevýhodou tohoto mechanismu jsou *Ekologické problémy.* Těžba bloků vyžaduje obrovské množství výpočetní síly (hodně hardwaru - třeba ASIC - výdrž 15 měsíců ) a energie.

**Consensus algoritmus** pro kryptoměny zajišťuje spolehlivé potvrzení plateb.
Jedním z konceptů, který byl zkoumán, jsou algoritmy pro toleranci byzantinských chyb - *BFT* (část uzlů může selhat/ být úmyslně zmanipulována).
Jedním z přístupů je volba lídra, který je zodpovědný za potvrzování výsledku a koordinaci transakcí.

**Proof of Stake (PoS)** je (ekologičtější) alternativa k PoW. Princip: náhodně určený těžař vytvoří nový blok a ostatní uzly v síti potvrzují správnost tohoto bloku. Těžař musí ručit svou zálohou (stake) za každý vytěžený blok, přičemž tato záloha je obvykle množství kryptoměny, kterou vlastní.
Základní myšlenka PoS je taková, že čím více kryptoměny jednotlivec vlastní a čím déle ji drží, tím větší je jeho pravděpodobnost, že bude vybrán k vytěžení nového bloku.
Nevýhodou jsou obavy z větší centralizace (vlastníci více kryptoměny ji drží déle a mají větší šanci na vytěžení bloku).

# 26 Ethereum

Druhá největší kryptoměna. Funguje na principu PoS a je decentralizovaně udržován její stav - EVM - kde lze spouštět i libovolný kód.

## 26.1 Proof of Stake

Validátorem *(dohlíží na tvorbu bloků)* se může stát každý, kdo složí zástavu 32 ETH. Navíc pokud jedná čestně, získává za svou práci odměny. Pokud čestně nejedná, tak naopak část své zástavy ztrácí.

**Epochy** Čas je dělený na epochy, kde každá epocha má 32 slotů a každý slot trvá $12s$. V každém slotu je přidán (maximálně) jeden blok.
Na začátku epochy je pro každý slot určen náhodně navrhovatel *(proposer)* bloku a jeho ověřovatelé *(committee)*.
Ověřovatelé pro daný slot hlasují, jaký znají poslední blok *(LMD GHOST vote)*. Musí se shodnout 2/3.
*Checkpoint* je aktuální blok v prvním slotu epochy. Hlasují o něm úplně všichni ověřovatelé s váhou hlasu dle vložených ETH. Opět jsou třeba 2/3 shody. *(FFG vote)*

Transakce považována za schválenou, pokud jsou za jejím blokem dva odsouhlasené checkpointy.
Zdržení je zde proto, aby byl čas objevit případné podvody nebo se vypořádat s nefunkčností některých validátorů.

## 26.2 Platby v Ethereu - Gas

Platí se za každou transakci. Poplatek odpovídá náročnosti pro EVM. Podle vytíženosti sítě se dynamicky mění.
Poplatek je spálen *(burned)* a ether zaniká.
Jsou zde odměny pro navrhovatele a validátory bloku a pak volitelné odměny na navrhovatele (tip).

## 26.3 Účty v Ethereu

Účty se dělí do dvou kategorií: *EOA (Externally owned Accounts)*, což jsou běžní lidé a pak na *Contracts*, které nemají privátní adresu (nemohou tedy podepsat transakci), ale mohou běžně reagovat na transakci o EOA.

**Contracts** Na blokchain může být zapsán program. Ten program pak kdokoli může zavolat *(má svou adresu)*.
Problémem však je, že programy obsahují chyby a zde není možnost opravy.
Smart contracts se obvykle využívá v decentralizovaných aplikacích; pro vytvoření další měny (stablecoins); jako NFT (non-fungible tokens - doklad o vlastnictví digitálního (uměleckého) díla); automatizované kryptoměnové burzy; bankovní služby (úschovna); uzavírání smluv.

## 26.4 Problém s množstvím transakcí

Na blokchain je možné zapsat pouze omezené množství transakcí a kapacita nestačí. Roste tedy proto cena za každou transakci.
Řešením je zvětšit blok (např. Bitcoin Cash), to ovšem nelze dělat do nekonečna.
Dalším řešením se nabízí komprese a optimalizace dat, nebo lightning network.

# 27 Idea Lightning Network, anonymita v Bitcoinu a Moneru

## 27.1 Lightning Network

Je to platební vrstva nad Bitcoinem (obecně i jinou kryptoměnou) umožňující levné mikrotransakce.
Umožňuje dost transakcí pro běžné používání.
Stále se jedná o velmi novou technologii, ale už je reálně využitelná (např. na alza.cz).
Řeší kapacitu sítě, rychlost platby (není třeba čekat několik bloků), cenu za platbu (jednotky haléřů), přispívá k anonymitě.

## 27.2 Idea

Alice a Bob spolu chtějí obchodovat v BTC.
Otevřou si společný účet (kanál, kam převedou BTC) a vedou si seznam provedených plateb.
Každý nový stav je podepsán Alicí i Bobem a každý z nich může kdykoli kanál uzavřít a BTC jsou zapsány na blokchain dle posledního stavu.

*("Tranzitivita za poplatek")* Alice chce obchodovat s Charliem, ale nemá s ním otevřený kanál. Má ale otevřený kanál s Bobem a ten má kanál s Charliem.
Pro provedení platby stačí upravit stavy existujících kanálů a Bob si za zprostředkování platby může říct o (malý) poplatek.

**Otevření kanálu** Neboli převedení peněz na tzv "2-of-2 multisignature adresu" - tedy 2 účastníci a výběr musí potvrdit oba.
Oba účastníci si vymění předem podepsaný pokyn k výběru peněz dle toho, kolik kdo vložil.

**Transakce** Stav účtu může být veden kdekoli. Při každé změně si účastníci vymění podepsaný pokyn pro výběr peněz dle nového poměru. Je zajištěno, aby nešlo využít starý pokyn (o to se stará RSMC)

**RSMC - Revocable Sequence Maturity Contracts** Své peníze mohu vždy vybrat až se zpožděním. Musí se nasdílet privátní klíče pro předchozí kontrakty. KKdyž někdo zkusí použít starý kontrakt, jeho peníze budou zablokovány a druhá strana má příslušný privátní klíč - tedy může vybrat vše.

**Hashed Timelock Contract (HTLC)** Vyplacení peněz za kontrakt podmíněno znalostí hesla. Příjemce platby vytvoří heslo, jeho hash pošle odesílateli a ten ho přidá jako podmínku kontraktu.

## 27.3 Anonymita a Monero

Blockchain je veřejný a lze akorát skrývat, kdo je reálným vlastníkem jednotlivých adres.
Možnosti u Bitcoinu jsou nevyužívat stále stejnou adresu a nebo *CoinJoin* služby.

**CoinJoin** Vezme se mnoho vstupů od různých uživatelů a uvnitř se "zmixují" tak, že jednotlivé výstupy jsou nevysledovatelné.

### Monero (XMR)

Jedná se o nejrozšířenější anonymní (anonymní platba) kryptoměnu. Architektura je podobná Bitcoinu. Pracuje s PoW (hash RandomX neumožňuje využití ASIC ani GPU).
Velikost bloku není pevně dána - může být až dvojnásobek mediánu posledních 100 bloků. Za větší blok je menší odměna (tedy: čím, vyšší poplatky, tím se vyplatí větší bloky). Navíc se odměna pro těžaře postupně zmenšuje.

**Stealth address** Zajišťuje anonymitu příjemci platby.
Každý uživatel má dva *EdDSA* klíče *view key* (private $a$, public $A = aG$) a *spend key* (private $b$, public $B = bG$). Veřejnou adresu tvoří $A, B$. Odesílatel vytvoří náhodnou hodnotu $r$, z ní se vytvoří jednorázová adresa a proběhne DH s $A$.
Příjemce využije *private view key* pro detekování platby (zkouší všechny transakce v blokchainu).

Pokud chce odesílatel dokázat, že zaslal platbu, může zveřejnit $r$ (nebo využít jakýkoli zero knowledge proof o držení $r$ - např, pomocí $r$ podepsat platbu)
Pokud chce příjemce, aby ostatní viděli jeho příchozí platby může zveřejnit *private view key.*

**Ring signature** Zajišťuje anonymitu odesílateli platby.
Pro každou platbu je náhodně vybráno 9 dalších potenciálních odesílatelů. Platba je vždy podepsána tak, aby nebylo možné zjistit, kdo z 10 osob ji podepsal.
Key image slouží jako ochrana před double spend: $I = xH(P)$, ověření, že stejný $I$ nebyl využit v minulosti, (vždy nutno utratit celou částku na adrese)

**RingCT = Ring Confidential Transactions** Zajišťuje anonymitu zaplacené částky. Technicky je to složité, ale jedná se o vylepšení Ring signature.

## 27.4 Stablecoin

Protože kurzy kryptoměn vůči *EUR/USD* nejsou stabilní, což není úplně šikovné při placení, vznikly tzv. *stablecoiny.* Tedy kryptoměna svázána s tradiční měnou (typicky USD), komoditou, kryptoměnou, předepsaným algoritmem Je třeba nějaká rezerva na výkyvy.
Příkladem je USDT, USDC.