

## Autentikace, autorizace

- Autentikace (autentizace, autentifikace) je proces ověření identity subjektu. Autorizace je vymezení rozsahu služeb pro identifikovaný subjekt.
- Lokálně lze autentikovat pomocí:
  - znalostí (heslo, PIN)
  - technických prostředků (klíč, HW token, ...)
  - biometrie (otisky prstů, ...)
- Vzdálená autentikace:
  - ochrana proti odposlechu: systém jednorázových hesel (OTP), kryptografie
  - přenos dat v protokolu: např. pomocí SASL (obecný model, zařazený do protokolů na základě *profilů*, např. v SMTP)
  - možnost použití autentikačního serveru a autentikačního protokolu (LDAP, RADIUS, NTLM, Kerberos, SAML)

Než budeme mluvit o bezpečnosti, měli bychom si ujasnit dva pojmy, které se často zaměňují:

- Autentikace je proces, který používají subjekty (obvykle uživatelé) k prokázání své totožnosti, tj. je to odpověď na otázku „kdo jsem“.
- Autorizace je proces, kterým autorita (obvykle server) přiřazuje oprávnění subjektu, který již uspěl v procesu ověřování, tj. je to odpověď na otázku „co mohu dělat“.

Obecně existují tři základní způsoby prokázání totožnosti:

- „Co znám.“ Nejjednodušší způsob je otestovat znalost nějakého faktu, hesla (nebo v jednodušší verzi pouze čísla, PIN), odpovědi na jednu ze sady předem definovaných otázek atd. Velkou nevýhodou těchto metod je, že důvěrný fakt může být prozrazen. Nicméně mají i určité výhody, zejména pokud se používají k přístupu k méně důležitému zdroji: snadná implementace, jednoduché sdílení, snadné použití pro vzdálené zdroje.
- "Co mám". Je trochu bezpečnější prokázat totožnost tím, že vlastníte nějaký předmět. Může se jednat buď o fyzický objekt (hardwarový token), nebo o data (elektronický klíč). Ukrást nebo zkopírovat předmět je poněkud složitější.
- Nejbezpečnějším důkazem totožnosti je biometrie – otisk prstu, sken sítnice, rozpoznání hlasu atd. – pokud však potřebujete prokázat totožnost pro vzdálený zdroj, mají tyto metod potíž v tom, jak propojit zařízení, které biometrická data získá, a server, který je ověří.

Používá se také kombinace různých metod.

Chceme-li ověřovat totožnost u vzdálené služby (serveru), musíme mít na paměti několik dalších faktorů:

Pokud použijeme znalostní test, existuje riziko odposlouchávání v případě, že používáme nezabezpečený kanál. Nejstarším způsobem ochrany proti opakovanému použití odposlechnutých informací je využití speciálně upravených informací, které jsou platné pouze pro jeden přístup (jednorázové heslo). Dalším způsobem je odstranit problém s nezabezpečeným kanálem a pomocí kryptografie vytvořit bezpečný kanál.

Dalším problémem je, že většina protokolů nemá žádné prostředky pro bezpečné ověření a musejí být rozšířeny, aby vůbec mohly předávat potřebné informace. Existuje framework nazývaný Simple Authentication and Security Layer (SASL), který zahrnuje různé metody ověřování a dokáže je včlenit do většiny důležitých protokolů. Pro každý protokol existuje speciální *profil*, který určuje, jak autentikaci v daném protokolu používat.

Častým požadavkem je také centralizace ověřování pro celou sadu prostředků, tj. vytvoření centrálního ověřovacího serveru (*poskytovatele identity*), který používá speciální protokol ke komunikaci buď s klientem (aby od něj získal data a poté mu poskytl výsledek ověření, jež může být použit k prokázání identity vůči serveru), anebo se serverem (*poskytovatelem služby*), který komunikuje s klientem sám.

### One-Time Password (OTP)

- Obecné označení pro mechanismy umožňující nereplikovatelnou plain-textovou autentikaci uživatele
- Původní off-line varianta:  
Vytisknutý seznam jednorázových hesel.
- Starší systém *challenge-response*:  
Server vyšle jedinečný náhodný kód, uživatel na klientovi z něj určeným způsobem vyrobí odpověď (např. pomocí speciální HW či SW kalkulačky, kam zadá přijatý kód a svoje heslo a dostane odpověď) a klient ji pošle serveru.
- Modernější řešení:  
Uživatel dostane speciální autentikační předmět (*token*), který na základě přesné časové synchronizace se serverem generuje jednorázový časově omezený kód.

V minulosti se používal velmi jednoduchý způsob jednorázové autentikace na nešifrovaném spojení. Uživatel si na serveru nechal vygenerovat a vytisknout seznam hesel, která pak jedno po druhém zadával při následných přihlášeních.

Později byla vyvinuta metoda *challenge-response*. Spočívá v tom, že server uživateli v okamžiku přihlašování pošle jedinečný náhodný řetězec, tzv. *výzvu*. Tu uživatel zadá spolu se svým heslem do speciální (HW nebo SW) kalkulačky a výsledek z kalkulačky napíše serveru jako odpověď. Tato metoda se používá v různých komunikačních schématech. Podobnou koncepci používají také často různé webové aplikace, které provedení nějaké operace vážou na potvrzení pomocí navštívení speciálního URL zaslaného uživateli emailem.

Novější implementace používají speciální malá HW zařízení, tzv. *tokeny*, která jsou přesně synchronizovaná se serverem a generují kód pro identifikaci. Platnost kódu je omezena na několik vteřin a jedno použití, takže nemůže být zcizen a zneužit.

### Kryptografie – symetrické šifrování

- Historie: aditivní, transpoziční, substituční šifry, šifrovací mřížky a tabulky atd.
- Současnost: analogické principy převedené do digitální podoby a podložené matematickou teorií
- Podstata: pro šifrování a dešifrování se používá stejný klíč
- Příklady: DES, Blowfish, AES, RC4
- Výhoda: rychlé, vhodné na velká data
- Nevýhoda: partneři si musí klíč předat bezpečnou cestou

Kryptografie hraje v současných komunikačních metodách důležitou roli. Pro šifrování a elektronický podpis se využívají kombinace tří základních typů algoritmů.

Prvním typem jsou algoritmy **symetrického šifrování**.

Historie symetrických šifrovacích metod je velmi dlouhá. Od pradávna potřebovali lidé ochránit důvěrnost přenášených zpráv. Existuje mnoho metod kódování založených na substituci nebo transpozici znaků. Jiným přístupem jsou šifrovací mřížky – je to destička s vyřezanými otvory, které odkrývají vždy čtvrtinu písmen, takže postupným otáčením odkryjeme celý text. Společnou vlastností těchto metod je, že oba komunikující partneři potřebují nějakou sdílenou informaci (metodu substituce či transpozice, mřížku apod.) pro zakódování i dekodování. Dnes se samozřejmě používají sofistikovanější metody založené na matematických principech.

Velkou výhodou současných metod symetrického šifrování je jejich rychlost, díky níž jsou vhodné pro šifrování velkých objemů dat. Háček je ale v tom, že pomocí symetrické kryptografie jsme nevyřešili náš problém – partneři si chtěli zabezpečeně doručit určitou tajnou informaci, teď si ji mohou zašifrovat, ale pořád si potřebují zabezpečeně doručit jinou tajnou informaci (klíč). Vzhledem k jeho velikosti mohou pro doručení klíče použít jiný způsob, ale podstatou je, že symetrická kryptografie pouze **redukuje velikost** problému, ale neřeší ho.

### Kryptografie – asymetrické šifrování

- Podstata: pro šifrování a dešifrování se používá pár navzájem neodvoditelných klíčů
- Matematický základ: jednocestné funkce
  - násobení vs. rozklad na prvočinitele
  - diskretní logaritmus  $m = p^k \bmod q$
- Příklady: RSA, DSA, ECDSA
- Výhoda: odpadá problém sdíleného tajemství - jeden klíč (veřejný) lze šířit, druhý (tajný) pečlivě uschovat
- Nevýhoda: pomalé algoritmy, lze šifrovat jen malá data
- Zásadní problém: veřejný klíč je třeba **pečlivě ověřovat**

Řešení problému, jak přenést bezpečně zašifrovaná data přes nezabezpečený kanál, přichází s **asymetrickou kryptografií**.

Základní myšlenkou je použít dva různé klíče, jeden pro šifrování a jeden pro dešifrování. Samozřejmě nesmí být možné odvodit jeden klíč z druhého. Pak můžete jednoduše publikovat jeden (veřejný) klíč z páru, zatímco druhý (tajný) uložíte na bezpečném místě.

Pro základní představu, na čem je asymetrické šifrování založeno si představte běžné násobení – snadno můžete vynásobit dvě čísla, ale pokud vám někdo dá pouze součin, může být obtížné najít patřičné činitele (jsou-li to velká prvočísla). Takové funkce se nazývají **jednocestné**. Příkladem sofistikovanější jednocestné funkce je diskretní logaritmus.

Výhody oproti symetrické kryptografii jsou zřejmé – není třeba si předávat *sdílené tajemství*. Velkou komplikací však je, že tyto metody jsou extrémně pomalé, a tudíž vhodné pouze pro malé objemy dat. Ale problém zmenšení velikosti transportovaných dat už jsme vyřešili pomocí symetrické kryptografie!

Obešli jsme tedy problém nutnosti sdílet tajemství, ale objevil se nám jiný problém: musíme **věřit** veřejnému klíči, tj. věřit, že je správný a že odpovídající tajný klíč skutečně patří osobě nebo službě, s níž chceme komunikovat.

### Kryptografie - hashovací funkce

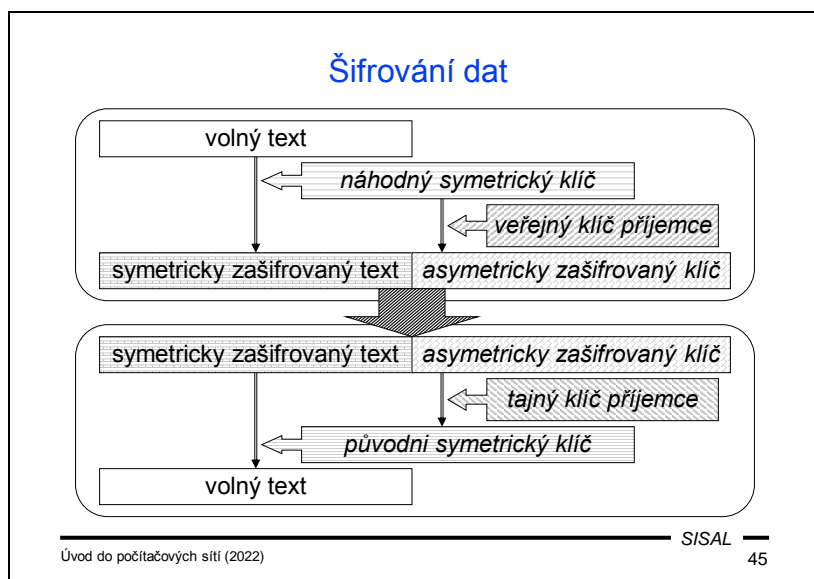
- Hashovací funkce
  - vytvoření pevného „kódu“ z daného textu
  - široké uplatnění (kontroly shody, výběr z tabulky,...)
  - příklady: CRC, MD5
- Použití v kryptografii
  - doplnění požadavků na algoritmus o bezpečnostní prvky
  - malá změna textu = velká změna hashe, „skoro jednoznačný“
  - jednocestnost, text je z hashe „neodvoditelný“
  - nalezení textu se shodným hashem je obtížné
  - příklad: SHA

Posledním typem kryptografického algoritmu, o kterém budeme mluvit, jsou **hashovací funkce**.

Obecně je *hashovací funkce* taková funkce, která vytváří z jakýchkoliv dat kód krátké pevné délky. Tyto funkce se hojně používají při programování, např. pro rychlé prohledávání tabulky, rychlé porovnávání textů apod. Princip je takový, že místo práce s úplnými daty pracujeme pouze s jejich hashem (kódem). Kód samozřejmě nemůže být jednoznačný, takže buď akceptujeme určité riziko (např. když porovnáme dvě verze souboru zdrojového kódu, pravděpodobnost, že se liší, ačkoliv jejich kódy jsou stejné, je tak nízká, že to můžeme ignorovat), nebo hash použijeme pouze ke snížení počtu prvků, se kterými se musíme vypořádat a následně je kompletně zkontrolovat.

Pro použití v kryptografii však takto slabé algoritmy nejsou přijatelné; pro dobrý kryptografický algoritmus musíme přidat další podmínky:

- Je důležité, aby i malá změna původních dat způsobila zásadní změnu hodnoty kódu, takže kód je „téměř jednoznačný“.
- Funkce musí být jednocestná; najít text, který odpovídá danému kódu, musí být „obtížné“, stejně jako najít jiný text, který má stejný kód jako daný text.



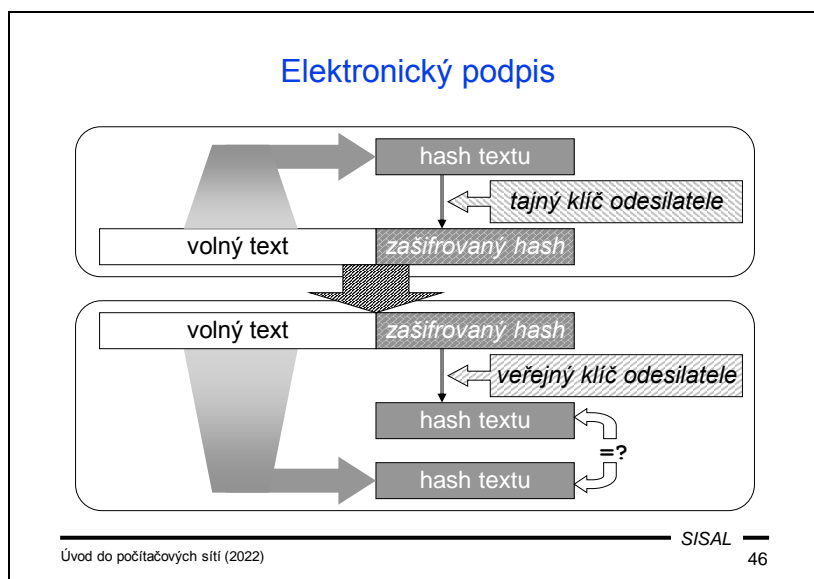
Nyní se můžeme podívat na dvě základní operace používané v kryptografii, šifrování dat elektronický podpis.

Pro efektivní **šifrování dat** používáme kombinaci symetrických a asymetrických algoritmů.

Předpokládejme, že máme text, který by měl být odeslán zašifrovaný přes nezabezpečený kanál (např. e-mail). Vygenerujeme náhodný klíč  $key_{sym}$  a symetricky zašifrujeme text. Text může být velký, protože používáme symetrickou kryptografii, která je dostatečně rychlá. Teď musíme předat  $key_{sym}$  příjemci, takže ho musíme zašifrovat. Klíč je však krátký, takže nyní můžeme použít asymetrickou kryptografii (rychlost zde není problémem) a zašifrovat  $key_{sym}$  **veřejným klíčem příjemce**. Tento zašifrovaný klíč je pak připojen k zašifrovanému textu.

Po úspěšném doručení příjemce vezme zašifrovaný klíč, k jeho dešifrování použije **svůj soukromý klíč** a dostane  $key_{sym}$ , kterým poté symetricky dešifruje text.

*Poznámka:* Zejména u e-mailů má tento přístup další velkou výhodu – pokud máte více než jednoho příjemce, zašifrujete data **pouze jednou**, poté zašifrujete  $key_{sym}$  zvlášť pro každého příjemce jeho veřejným klíčem a všechny takto zašifrované klíče připojíte k zašifrovaným datům.



Pro **digitální podpisy** používáme kombinaci asymetrické kryptografie a hashovací funkce.

Odesílatel zvolí hashovací funkci, vezme text (buď nešifrovaný nebo zašifrovaný) a vypočítá jeho hash. Pak vezme **svůj soukromý klíč** (klíč **odesílatele**) a zašifruje hash. Zašifrovaný hash je pak připojen k původnímu textu (spolu s parametry hashovací funkce, které byly použity).

Příjemce převezme text a použije zadanou hashovací funkci a její parametry, aby si sám vypočítal hodnotu hashe. Pak vezme **veřejný klíč odesílatele** a dešifruje hash, který původně vypočítal a zašifroval odesílatel. Pokud jsou obě hodnoty hashe shodné, lze věřit, že text byl skutečně odeslán osobou, která měla přístup k soukromému klíči odesílatele, a že zpráva nebyla změněna.

Všimněte si, že jsem neřekl "pošta byla poslána správnou osobou", víme jen, že odesílatel byl schopen pracovat se soukromým klíčem, a nikoliv to, že to byla skutečně ta správná osoba...!

Pozor na vlastnosti těchto dvou kryptografických operací: šifrování dat chrání data před přečtením kýmkoliv, zatímco digitální podpis zabraňuje komukoliv měnit data odeslaná původním odesílatelem nebo odesílat zcela odlišná data, aniž by to příjemce odhalil.




### Diffie-Hellmanův algoritmus

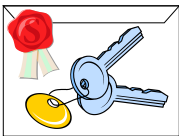
- Způsob výměny informací mezi dvěma partnery posílanými nezabezpečeným kanálem tak, aby oba získali sdílenou tajnou informaci (např. symetrický šifrovací klíč)
- Základ řady protokolů založených na symetrické kryptografii
- Postup:
  1. Alice vygeneruje tajné číslo  $a$  a veřejná (prvo)čísla  $p$  a  $q$ .
  2. Spočítá číslo  $A = p^a \bmod q$  a pošle  $p$ ,  $q$ , a  $A$  Bobovi.
  3. Bob zvolí tajné číslo  $b$ , spočte  $B = p^b \bmod q$  a pošle  $B$  Alici.
  4. Alice spočítá  $s = B^a \bmod q$  a Bob totéž  $s = A^b \bmod q$ .
- Princip:
  - $A^b = (p^a)^b = p^{ab} = p^{ba} = (p^b)^a = B^a$
  - Bez znalosti tajných čísel  $a$  a  $b$  a při volbě dostatečně velkých prvočísel  $p$  a  $q$  je i při odchyzení čísel  $A$  a  $B$  spočítání čísla  $s$  považováno za neřešitelnou úlohu.

Existuje další široce používaná matematická metoda, která umožňuje dvěma partnerům dohodnout se na společném tajemství pomocí komunikace přes otevřený nezašifrovaný kanál, tzv. **Diffie-Hellmanův algoritmus**. Algoritmus je založen na jednocestných funkcích a základní princip je takový, že obě strany si drží své tajné číslo a kanálem posílají pouze výsledky diskretních logaritmu. Díky jednocestnosti funkcí nikdo, kdo nezná tajnou část informace, nemůže vypočítat konečné sdílené tajemství.

### Autenticita veřejných klíčů



- Je třeba ověřit, že jmenovka patří ke klíči
  - Mezi lidmi lze obvykle snadno ověřit, že komunikují se správným partnerem dřív, než sdělím tajné informace
  - Klíč lze ověřit z více nezávislých zdrojů
  - Mezi komponentami SW je nutno nějak automatizovat



- Autenticitu ověří třetí strana a připojí svůj podpis; je to buďto
  - někdo, koho já osobně znám a mám jeho resp. její klíč ověřený („pavučina důvěry“)
  - veřejně uznávaná certifikační autorita; jejich seznam je např. v prohlížečích, ale věrohodnost takového seznamu není zcela stoprocentní

Úvod do počítačových sítí (2022)
SISAL
48

Jak již bylo zmíněno, asymetrická kryptografie má jeden zásadní problém: jak zajistit resp. ověřit, zda (veřejný) klíč patří konkrétní identitě (osoby nebo služby)? Pokud s někým známým komunikujeme osobně, obvykle poznáme, zda je daná osoba skutečně naším partnerem – podle pozdravu, stylu řeči, odkazu na nějaký společný zážitek atd. Pokud však potřebujeme prokázat identitu při elektronické komunikaci programů, musíme použít nějaké technické prostředky.

V reálném životě nastává podobná situace při používání obyčejných klíčů. Máte-li mnoho klíčů, pravděpodobně vytvoříte štítky pro jejich identifikaci a ty připojíte ke klíčům. A protože jste označení udělali sami, pravděpodobně budete věřit, že štítky odpovídají klíčům. Pokud však máte co do činění s klíčem obdrženým od někoho jiného, musíte uvěřit, že jeho štítek je správný. Jednou z možností je, aby klíč se značkou byl vložen do obálky a zapečetěn někým, komu důvěřujete.

Stejný princip se používá i pro softwarové klíče. Ke klíči můžete připojit **identifikační značku** a nechat někoho **potvrdit**, že toto propojení je správné. K tomuto způsobu dokazování autenticity existují různé přístupy.

- Například u PGP klíčů pro podepisování e-mailů se používá metoda *web of trust*. Uživatel vytvoří klíč a identifikační značku a požádá další uživatele, aby svým podpisem potvrdili, že značka a klíč k sobě patří. Tento podepsaný klíč je pak publikován a všichni uživatelé, kteří důvěřují některému z podepisujících klíčů, se mohou rozhodnout, že budou důvěřovat i podepsanému klíči. A když poté tento uživatel někomu dalšímu podepíše jeho klíč, web of trust se zase rozroste.
- Public Key Infrastructure místo toho používá pro podepisování speciální organizace nazývané Certification Authorities (CA). CA podepíše dvojici klíč+značka a ten, kdo důvěřuje CA, bude považovat za důvěryhodný i podepsaný klíč.

### Certifikát

- Certifikát je klíč doplněný o identifikaci vlastníka a podepsaný vydavatelem, např. certifikační autoritou (CA)
- Pokud důvěřujeme vydavateli, můžeme věřit klíči vlastníka (**ověřovat věrohodnost CA!**)
- Struktura certifikátu podle X.509 (RFC 3280, SSL, ne SSH):
  - certifikát
    - verze certifikátu
    - sériové číslo certifikátu
    - vydavatel
    - doba platnosti
    - vlastník veřejného klíče
    - informace o veřejném klíči vlastníka (algoritmus a klíč)
  - algoritmus pro elektronický podpis
  - elektronický podpis

**Certifikát** je veřejný klíč s odpovídající identifikační značkou a podepsaný certifikační autoritou. Uživatel nebo klient, který potřebuje ověřit pravost klíče, stáhne jeho certifikát a zkontroluje podpis pomocí veřejného klíče CA, která ho podepsala (označme ji  $CA_1$ ). Aplikace provádějící ověření musí tento klíč považovat za důvěryhodný. Pokud tomu tak není, je nutné stáhnout certifikát  $CA_1$  (pro urychlení může být také připojen k původnímu certifikátu). Certifikát  $CA_1$  je podepsán certifikační autoritou  $CA_2$ . Nyní musíme stejným způsobem ověřit podpis  $CA_2$ . Toto je takzvaný „řetězec důvěry“, který končí buď nalezením nějaké známé CA, nebo neúspěchem.

Seznam „důvěryhodných“ CA je distribuován společně s operačním systémem nebo se softwarem, který jej bude používat. Pokud například nainstalujete webový prohlížeč, je současně nainstalován seznam CA, kterým věří výrobce prohlížeče. Čas od času v seznamu dojde k určitým změnám a prohlížeč seznam automaticky aktualizuje, aby byl aktuální. Problém je v tom, že čas od času se nějaká CA se špatnou pověstí (např. takové, jež dostatečně neověřují skutečné vlastnictví klíče) dostane na seznam důvěryhodných CA. Jakmile se to zjistí, všechny klíče jí podepsané musí být zneplatněny.

## SSL, TLS

- Secure Socket Layer 3.0 ~ Transport Layer Security 1.0, **dnes již nedoporučovaná**, novější verze: 1.1, 1.2 a 1.3
- Mezivrstva mezi transportní a aplikační vrstvou umožňující autentikaci a šifrování
- Využívá řada protokolů (např. HTTPS na portu 443)
- Princip:
  1. Klient pošle požadavek na SSL spojení + parametry.
  2. Server pošle odpověď + parametry + certifikát serveru.
  3. Klient ověří server a vygeneruje základ šifrovacího klíče, zašifruje ho veřejným klíčem serveru a pošle mu ho.
  4. Server rozšifruje základ šifrovacího klíče. Z tohoto základu vygenerují jak server, tak klient hlavní šifrovací klíč.
  5. Klient a server si navzájem potvrdí, že od teď bude jejich komunikace šifrovaná tímto klíčem.

Nyní máme dostatek informací, abychom vysvětlili, jak jsou staré protokoly TCP/IP používány způsobem, který zajišťuje jejich bezpečnost. Obecnou metodou je vložit speciální mezivrstvu mezi transportní a aplikační vrstvu; tato vrstva má na starosti identifikaci koncových bodů a šifrování dat. Vrstva se původně jmenovala Secure Socket Layer a ke zkratkám protokolů, které ji používaly, bylo připojeno písmeno „s“. Například pokud URL obsahuje schéma „https“, znamená to „HTTP provozované přes SSL“. Důležitým faktem je, že **nejde o nový protokol**; schéma pouze říká, že se používá SSL mezivrstva. Ve verzi 3.0 byl SSL po drobných úpravách přejmenován na Transport Layer Security 1.0, ale termíny SSL a TLS se stále používají zaměnitelně (kromě případů, kdy se mluví o číslech jejich verzí). Vývoj TLS stále pokračuje a nyní se nedoporučuje používat nejstarší verzi 1.0, která vycházela z SSL.

### Aplikační vrstva TCP/IP

- Spojuje funkce OSI 5, 6 a 7
  - pravidla komunikace mezi klientem a serverem
  - stav dialogu
  - interpretaci dat
- Protokol na aplikační vrstvě definuje
  - průběh dialogu na obou stranách
  - formát zpráv (textový/binární, struktura,...)
  - typy zpráv (požadavků a odpovědí)
  - sémantiku zpráv, sémantiku informačních polí
  - interakci s transportní vrstvou

Úvod do počítačových sítí (2022)
SISAL 51

Po několika úvodních kapitolách se můžeme začít věnovat nejdůležitějším protokolům na **aplikační vrstvě**. Jak již víme, v TCP/IP tato vrstva pokrývá náplň práce horních tří vrstev OSI. To znamená, že aplikační protokol v TCP/IP musí určovat:

- Průběh dialogu. Zejména informace jako: kdo iniciuje spojení, kdo zahájí vlastní dialog, jaké operace může klient požadovat a jaké mohou být reakce na jednotlivé požadavky atd.
- Formát zpráv. Obecně platí, že protokoly jsou buď
  - textové - tj. můžeme si prohlédnout komunikaci z protokolu v obyčejném textovém editoru; např. celočíselné hodnoty jsou odesílány v jejich textové interpretaci
  - binární - tj. data jsou strukturována do proudu bloků, bytů nebo dokonce bitů, takže abychom jim porozuměli, musíme použít nějaký software, který je převede do textové podoby; např. celá čísla jsou odesílána jako posloupnost bitů představujících danou hodnotu
- Typy zpráv. Seznam typů zpráv pro různé požadavky a seznam možných odpovědí na ně.
- Sémantika dat. Každé datové pole (textové i binární) musí být popsáno tak, aby existovala pouze jediná možná interpretace.
- Jaký transportní protokol bude použit ve které situaci a jak. Např. pro UDP může být definována maximální velikost zprávy, zatímco v případě TCP bude protokol definovat sdružení nebo naopak rozdělení zpráv do bloků odesílaných jako celek.

## Domain Name System

- Klient-sever aplikace pro překlad jmen na adresy a naopak
- Binární protokol nad UDP i TCP, port 53, RFC 1034, 1035
  - Běžné dotazy (do 512B v non EDNS) se vyřizují pomocí UDP
  - Větší datové výměny probíhají v TCP
- Klient se obrací na servery zadané v konfiguraci a postupně získává informace o dalších, dokud nedostane odpověď
- Jednotkou dat je „záznam“ (resource record - RR), př.:
 

```
ns.cuni.cz. 3600 IN A 195.113.19.78
```

  - jméno záznamu
  - doba platnosti (TTL)
  - typ a data

Domain Name System (DNS) je služba pro překlad doménových jmen na adresy a naopak. Ve skutečnosti poskytuje ještě další informace o uložených doménových jménech, ale dotazy na IP adresy jsou nejfrekventovanější.

Služba je realizována stejnojmenným protokolem, který využívá oba transportní protokoly.

- Běžné dotazy jsou vyřizovány v UDP. Dotaz i odpověď jsou totiž krátké a režie s ustanovením TCP spojení je zbytečná. Navíc UDP umožňuje klientovi rychleji reagovat na případnou pomalou nebo chybějící reakci serverů. Původně byl limit pro UDP zprávu 512 B, ale dnes je standardem rozšířený protokol (EDNS), který umožňuje klientovi a serveru dohodnout se na větší velikosti.
- Pokud odpověď přesahuje daný limit, server pošle pouze takovou část, která se do zprávy vejde a nastaví příznak TC (truncated). Klient pak může dotaz zopakovat pomocí TCP a tím získat kompletní odpověď. Některé výměny dat (např. aktualizace databáze mezi servery) probíhají rovnou v TCP.

Běžná implementace klienta je taková, že postupně posílá dotazy na servery, které má ve své konfiguraci (rozmyslete si, proč tam má **adresy** a ne jména těch serverů), přičemž postupně zvyšuje timeout, který má na příchod odpovědi (obvykle začíná na 0,5 sec), dokud odpověď nedostane. Pokud odpověď neobsahuje potřebné informace, měla by obsahovat odkaz na další servery, kterých je třeba se ptát dál.

Protokol je binární, každá zpráva obsahuje hlavičku a pak určitý počet záznamů nazývaných *resource record* (RR). Každý záznam obsahuje jméno, dobu platnosti v sekundách (time-to-live, TTL), typ záznamu a odpovídající data.

DNS záznamy		
Typ	Jméno	Data
SOA	jméno domény	obecné informace o doméně
NS	jméno domény	jméno nameserveru domény
A	jméno počítače	IPv4 adresa počítače
AAAA	jméno počítače	IPv6 adresa počítače
PTR	reverzní jméno (např. pro IP adresu 1.2.3.4 je to 4.3.2.1.in-addr.arpa, pro ::1 je to 1.0...0.ip6.arpa)	doménové jméno počítače
CNAME	jméno aliasu	kanonické jméno počítače
MX	jméno domény/počítače	jméno poštovního serveru a jeho priorita

Přehled nejdůležitějších typů záznamů:

- SOA (Start Of Authority) je úvodním záznamem o každé doméně a obsahuje informace jako číslo verze dat, adresu správce apod.
- NS jsou záznamy, které definují nameservery, jež udržují databázi záznamů dané domény (obvykle jich má každá doména více).
- A je záznam obsahující IPv4 adresu pro dané jméno.
- AAAA je záznam pro IPv6 adresu. Jméno záznamu je možná poněkud zarážející a člověk by čekal spíše něco jako A6. Skutečně, takový typ záznamu byl zaveden jako první návrh, ale neosvědčil se, a proto byl nahrazen. Jméno AAAA naznačuje, že záznam obsahuje 4x delší data, než záznam A...
- PTR je reverzní záznam, který slouží pro převod adres na jména. Vzhledem k tomu, že na levé straně každého RR musí být **jméno** a nikoliv adresa, je nutné nějak adresu převést do formátu jména. Pro IPv4 se používá převod  $IP_1.IP_2.IP_3.IP_4 \rightarrow IP_4.IP_3.IP_2.IP_1.in-addr.arpa$ . Důvodem pro obrácené pořadí bajtů je odlišná hierarchie jmen (zprava doleva) a adres (zleva doprava). To např. umožňuje umístit server zodpovídající za síť  $IP_1.IP_2.IP_3.*$ , a tedy doménu  $IP_3.IP_2.IP_1.in-addr.arpa$ , přímo do této sítě. V případě IPv6 se adresa rozdělí na půlbajty, jejich hexadecimální hodnoty se oddělí tečkami a připojí se doména ip6.arpa. Každý počítač zapojený do internetu by měl mít reverzní záznam, protože některé servery vyžadují, aby klienti, co se na ně připojují, takový záznam měli.
- CNAME (Canonical NAME) je záznam pro tvorbu aliasů. Na levé straně stojí jméno aliasu, na pravé straně jeho kanonické neboli **skutečné** jméno počítače (pozor, v chápání levé a pravé strany se u CNAME dělají chyby). Název záznamu i popis v RFC jasně definují, že alias už by neměl vést na jiný alias. Důvodem je redukce dodatečných dotazů. Bohužel ale toto omezení je z provozního hlediska nepřijemné a toto pravidlo se běžně porušuje.

- MX (Mail eXchanger) je záznam definující, který server přijímá pro danou doménu (příp. počítač) poštu (obvykle bývá těchto záznamů více). Je to hezký příklad, jak oddělit záznam o doméně od záznamu o určité službě v dané doméně. U pošty funguje velmi dobře a je jen škoda, že podobný jednoduchý mechanismus nebyl zaveden i pro WWW. To vede k tomu, že se dnes běžně definují pro **doménu** záznamy, které jsou určeny pro **počítač** (A resp. AAAA), aby v prohlížečích uživatelé mohli vynechávat „www“. Řešení pomocí speciálního záznamu „web server“ by bylo koncepčnější.



## Servery DNS

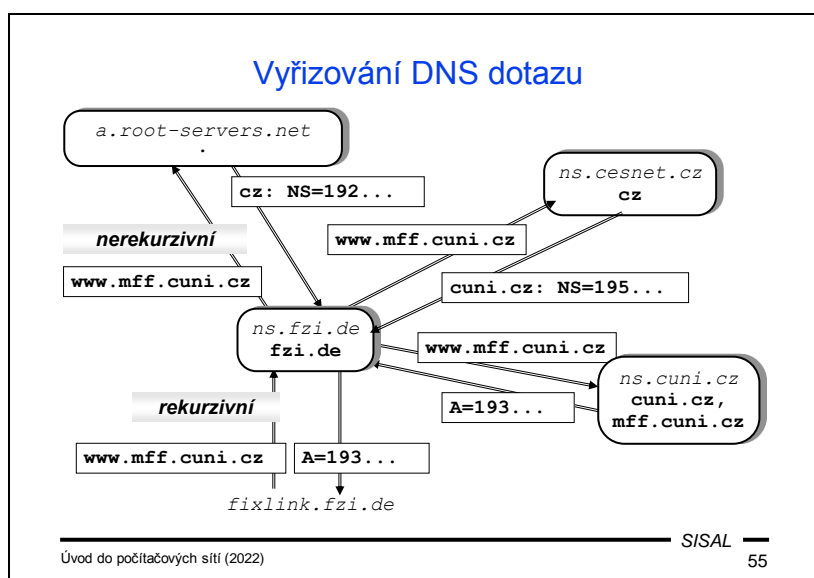
- Typy serverů:
  - primární: spravuje záznamy o doméně
  - sekundární: stahuje a uchovává kopii dat o doméně
  - caching-only: udržuje jen (ne)vyřešené dotazy po dobu platnosti
- Každá doména (zóna) musí mít alespoň jeden, ale raději více *autoritativních* (primárních nebo sekundárních) nameserverů
- Aktualizaci zónové databáze vyvolává sekundární server, je ale možné z primárního serveru signalizovat její potřebu
- Pro výměnu dat se používá TCP, ale normální formát dotazu a odpovědi (data se posílají jako DNS RR)

Z hlediska vyřizování dotazů spadajících do určité domény rozdělujeme servery na tři skupiny:

- Nejdůležitějším serverem je tzv. primární (nebo „master“) server. Ten spravuje databázi záznamů domény.
- Jako záloha se většinou zřizuje několik sekundárních (nebo „slave“) serverů. Ty si periodicky stahují z primárního serveru aktuální obsah databáze.
- Všechny ostatní servery, pokud se k nějakým záznamům pro cizí doménu dostanou, obvykle si je po nějakou dobu uloží do své *cache*, a proto jim říkáme *caching-only*.

První dva typy se dohromady označují jako **autoritativní** servery, jelikož jejich data pocházejí z ověřeného zdroje. Všechny autoritativní servery vracejí odpovědi označené příznakem autoritativnosti, klienti si mohou vybrat libovolný z nich pro položení dotazu.

Obnovu databáze iniciuje sekundární server v závislosti na periodě definované v SOA záznamu. Mimo tyto periodické časy může primární server informovat servery sekundární, že došlo k takové změně dat, že je vhodné udělat aktualizaci mimo pořadí. Tento mechanismus šetří objem komunikace mezi servery, důsledkem ale je, že i autoritativní server může po krátkou dobu po změně dat vracet klientům neaktuální údaje.



Podívejme se nyní na příklad průběhu dotazu.

- Řekněme, že uživatel na nějakém počítači v nějaké doméně napíše do adresního řádku browseru adresu `www.mff.cuni.cz`. Dle obvyklé implementace klienta na pracovní stanici se vygeneruje dotaz směřovaný nameserveru v doméně, kde dotaz vznikl. Dotaz bude **rekurzivní**, což znamená, že server převezme zodpovědnost za vyřízení dotazu a odeslání odpovědi. Pokud zvolený server nemá momentálně ve své cache žádnou relevantní informaci a bude ji muset začít hledat.
- Server nemá žádnou informaci o doménách `mff.cuni.cz`, `cuni.cz` ani `cz`. Musí se proto obrátit na některý z tzv. **kořenových** nameserverů, jejichž adresy má ve své konfiguraci. Tyto servery ale neposkytují rekurzivní styl odpovědi – na položený dotaz najdou ve své databázi co nejrelevantnější odpověď a tu pošlou. V tomto případě to bude odpověď typu „pošli svůj dotaz nameserveru jménem ...cz, jehož adresa je ...“.
- Server si tuto informaci uloží do cache a pokračuje v dotazování směrem k navrženému serveru. Tak se mu postupně dostanou do cache všechny mezilehlé servery, až dotaz dorazí k nějakému autoritativnímu serveru, který mu pošle konečnou odpověď.
- Odpověď si rovněž uloží do cache a konečně ji může odeslat klientovi, který se ptal.

Možná vás bude zajímat, proč kořenovému serveru posíláme celý dotaz, když nám s největší pravděpodobností odpoví pouze záznamem pro `cz`. Má to několik důvodů:

- Základní problém je, že DNS je navrženo tak, že součástí **jména** může být i **tečka**! Je to jako kdybyste v souborovém systému mohli ve jménech souborů používat lomítka resp. zpětná lomítka. Toto rozhodnutí přináší jisté výhody (mohu založit web `www.oddeleni.firma.cz`, aniž existuje doména `oddeleni.firma.cz`), ale rovněž

řadu nevýhod. Klient DNS zkrátka nemůže rozhodnout, které tečky ve jméně jsou opravdu oddělovači domén a které nikoliv. Proto se dotaz musí posílat celý – někomu, kdo tuto informaci má.

- V příkladu na obrázku vidíme, že jsme ušetřili jeden krok tím, že jsme serveru pro cuni.cz poslali celý dotaz a ne jen částečný dotaz „kde je server pro mff.cuni.cz“. Server pro cuni.cz je totiž sekundárním serverem pro mff.cuni.cz, takže nám mohl rovnou poslat požadovanou finální autoritativní odpověď.

### DNS dotaz a odpověď

• **Dotaz:**

ID: *n*

FLAGS: Recursion Desired

QUERY: www.cuni.cz. IN A

• **Odpověď:**

ID: *n*

FLAGS: Authoritative Answer

QUERY: www.cuni.cz. IN A

ANSWER: www.cuni.cz. IN CNAME tarantula

tarantula IN A 195.113.89.35

AUTHORITY: cuni.cz. IN NS golias

ADDITIONAL: golias IN A 195.113.0.2

Úvod do počítačových sítí (2022)
S/SAL 56

Před diskusí o bezpečnosti DNS se podívejme na to, jak vypadá dotaz a odpověď.

Dotaz v DNS se skládá ze záhlaví, které mj. obsahuje 2B náhodné číslo (identifikátor) dotazu a příznaky (např. požadavek na rekurzivní odpověď), a tzv. QUERY sekce s jediným RR obsahujícím dotazované jméno a typ (tento záznam je výjimečný tím, že nemá datovou část).

Odpověď obsahuje v záhlaví opět identifikátor dotazu a příznaky (např. autoritativnost odpovědi) a v sekci QUERY zopakovaný dotaz. Následuje sekce ANSWER obsahující RR s odpověďmi, sekce AUTHORITY obsahující seznam nameserverů, které mohou dát autoritativní odpověď nebo alespoň autoritativní informaci o některé z nadřazených domén obsažených v dotazu, a sekce ADDITIONAL, která obsahuje dodatečné informace (adresy nameserverů ze sekce AUTHORITY, adresy MX serverů uvedených v ANSWER sekci apod.).

## Bezpečnost DNS

- Problém útočníka: jak se dostat ke znění dotazu?
  - volba náhodného zdrojového portu
  - volba náhodného ID
- Příklad útoku („cache poisoning“): Do korektní odpovědi může server do sekce `AUTHORITY` a `ADDITIONAL` přidat falešné údaje o jiné doméně.
- Možné řešení: postupovat od root serverů a ptát se pouze autoritativních serverů.
- Komplexní řešení:
  - podpisy zabezpečené DNS (DNSSEC) - delegující doména obsahuje hash podepisovacího klíče, který je uložen na autoritativním serveru domény
  - je ale komplikované a rozšiřuje se pomalu.

Bezpečnost DNS je založena na tom, že pro potenciálního útočníka není úplně snadné se dostat ke znění dotazu, aby mohl podvrhnout odpověď. Pokud dokáže napadnout a odposlouchávat lokální síť, je útok možný. Pokud ale dotaz opustí lokální síť, je směrován mezi síťovými routery až k cílovému serveru, a tam se již k dotazu dostane obtížně. Druhou variantou je dotaz očekávat, odhadnout jeho znění a poslat falešnou odpověď. Tenhle postup je ztížen náhodnou volbou zdrojového portu UDP dotazu a náhodným ID – to dává řádově miliardy možností.

Je tedy třeba volit jiné cesty. Jako ukázkou slabiny protokolu můžeme uvést tzv. *cache poisoning* útok. Útočník legálně provozuje server pro nějakou doménu a donutí klienta, aby mu poslal DNS dotaz (např. vábivou reklamou ve spamu). Dotaz obsahuje korektní odpověď (tím je útok záluďnější), ale v sekci `AUTHORITY` útočníkův server podvrhne třeba informaci o tom, že server pro doménu `cz` je rovněž tento útočníkův server. Pokud je klient naivní, uloží si tuto falešnou informaci do cache a od té chvíle získá útočník kompletní kontrolu nad dotazy směřujícími do domény `cz` z daného klienta (nebo hůř – z celé sítě).

Řešením je samozřejmě postupovat při dotazech od kořenových serverů a do cache ukládat pouze informace, ke kterým vedou autoritativní odpovědi.

Slabá bezpečnost DNS vedla ke vzniku dalšího rozšíření DNSSEC, jehož podstatou je podepisování všech záznamů v doméně klíčem, který je uložen u nadřazené domény. Útočník, který nezná klíč, nemůže záznamy zfalšovat, a zfalšovat klíč nemůže bez přístupu k serveru nadřazené domény (pro zfalšování údajů o doméně `cuni.cz` by musel napadnout server pro doménu `cz`). Problém DNSSEC je, že pro zajištění dostatečné bezpečnosti se protokol stal extrémně komplikovaným, stejně jako správa podepsaných domén, a díky tomu se rozšiřuje velmi pomalu.

## Diagnostika DNS

- Program **nslookup**
  - podpříkazy: **set type, server, name, IPadr, ls, exit**
  - ```
> set type=ns
> cuni.cz
Server:          195.113.19.71
Address:         195.113.19.71#53

Non-authoritative answer:
cuni.cz nameserver = golias.ruk.cuni.cz.
cuni.cz nameserver = ns.ces.net.

Authoritative answers can be found from:
```
- Program **dig**
  - **dig** [*@server*] *jméno* [*typ\_RR*] [*options*]

Pokud potřebujeme zkontrolovat chování *resolveru* (komponenty operačního systému zodpovědné za využívání DNS služby), máme k dispozici program **nslookup**. Po spuštění program vypíše prompt a můžeme zadávat jednak příkazy pro nastavení serveru, jehož se ptáme, a parametrů dotazů a jednak jména, která chceme dohledat. Program lze volat i z příkazové řádky a dotaz zadat pomocí parametrů.

Na UNIXových počítačích máme obvykle k dispozici ještě program s podobnou funkcí jménem **dig** resp. **drill**.

### Souhrn 3

- Popište tři základní skupiny kryptografických algoritmů a rozdíly mezi nimi.
- Popište princip šifrování a elektronického podpisu.
- Popište účel a podstatu Diffie-Hellmanova algoritmu.
- Jakým způsobem se ověřuje pravost veřejného klíče?
- Jaký je rozdíl mezi protokoly HTTP a HTTPS?
  
- Kdy se v DNS používá UDP a kdy TCP?
- Popište účel a chování sekundárního nameserveru domény.
- Popište postup řešení DNS dotazu.
- Popište bezpečnostní rizika DNS.