VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

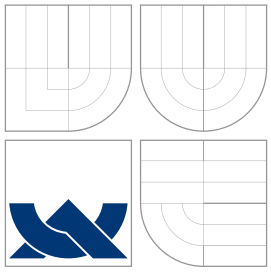
ONLINE VALIDACE ZÁZNAMŮ DNSSEC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

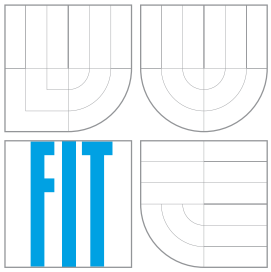
AUTOR PRÁCE
AUTHOR

Bc. MARTIN BACHTÍK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ONLINE VALIDACE ZÁZNAMŮ DNSSEC

ONLINE DNSSEC RECORDS VALIDATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN BACHTÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR LAMPA

BRNO 2011

Zadání

1. Prostudujte standardy RFC týkající se zabezpečeného DNS a metody konfigurace zabezpečených DNS zón.
2. Seznamte se s rekurzivním algoritmem validace záznamů DNSSEC a dostupnými nástroji pro ruční validaci.
3. Navrhněte a implementujte nástroj, který provede postupnou validaci zadaného záznamu od nejvyšší zóny a detailně zdokumentuje celý řetězec důvěry. Nástroj implementujte v jazyce PHP s minimálními externími závislostmi.
4. Zhodnoťte dosažený výsledek v porovnání s dostupnými nástroji.

Abstrakt

Diplomová práce studuje rozšíření, které zabezpečuje systém doménových jmen zavedením ověřitelnosti pravosti dat, známé jako DNSSEC. Tvůrčím výstupem je návrh aplikace a její následná implementace, která v každé etapě při zdolávání cesty jmenným prostorem ke zvolenému doménovému jménu kontroluje náležitosti tohoto rozšíření a detailně eviduje řetězec důvěry.

Abstract

Master's thesis is studying an extension that secures the domain name system by introducing the verifiability of authenticity of data, known as DNSSEC. Productive output is proposal of application and its subsequent implementation that at each stage of browse the namespace to the selected domain name checks the appropriatenesses of this extension and in detail reports the trusted chain.

Klíčová slova

DNS, DNSSEC, RRSIG, DNSKEY, DS, NSEC, NSEC3, NSEC3PARAM, RR, RSA, MD5, SHA-1, SHA-256, SHA-512, DER, PEM, PHP, SQL, AJAX, XHTML, HTTP, OpenSSL, resolver, počítačová bezpečnost, validace, řetězec důvěry, veřejný klíč, elektronický podpis, asymetrická kryptografie, ověřování pravosti, ověřování absence

Keywords

DNS, DNSSEC, RRSIG, DNSKEY, DS, NSEC, NSEC3, NSEC3PARAM, RR, RSA, MD5, SHA-1, SHA-256, SHA-512, DER, PEM, PHP, SQL, AJAX, XHTML, HTTP, OpenSSL, resolver, computer security, validation, trusted chain, public key, electronic signature, asymmetric cryptography, authentication, absence validation

Citace

Martin Bachtík: Online validace záznamů DNSSEC, diplomová práce, Brno, FIT VUT v Brně, 2011

Online validace záznamů DNSSEC

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně pod vedením Ing. Petra Lampy. Uvedl jsem všechna literární díla, ze kterých jsem při tvorbě této práce vycházel.

.....
Martin Bachtík
23. května 2011

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Petra Lampy za zajímavé téma a jeho odbornou pomoc v souvislosti s řešením této práce.

Obsah

Seznam obrázků

1 Úvod

Kdysi dávno, před mnoha a mnoha lety, internet neposkytoval příliš důležité informace, tudíž útočníci nebyli motivováni k provádění útoků; a ani nebyl příliš rozšířen, tudíž útočníci nebyli schopni k provádění útoků. Každý systém pouze plnil svoje poslání, a takovým byl i systém doménových jmen.

Systém doménových jmen je databáze rozprostřená po celém světě poskytující informace o doménových jménech. Takovou informací je zejména adresa IP. Vzdálený počítač se zaměřuje pouze pomocí adresy IP, ale uživatel zná pouze jeho doménové jméno. Pro převod doménového jména na adresu IP se používá právě systém doménových jmen. Tento mechanismus zároveň přináší další schopnosti. Vzdálenému počítači dovoluje, aby si mohl změnit svoji adresu IP, aniž by musel změnit i svoje doménové jméno. Díky textové reprezentaci doménového jména je zapamatování a manipulace s ním snadnější než s číselnou reprezentací adresy IP. Systém doménových jmen nabývá i dalších vlastností a funkcí.

V současném moderním světě internet již poskytuje příliš důležité informace, tudíž útočníci mohou být motivováni k provádění útoků; a je již široce rozšířen, tudíž útočníci mohou být schopni k provádění útoků. Do internetu je připojena spousta bankovních aplikací a internet se pomalu dostává do každého zařízení. Z velkého množství připojení k internetu vyplývá velké množství adres IP a doménových jmen. Tímto trendem důležitost systému doménových jmen stále stoupá. Bylo objeveno již několik typů útoků na systém doménových jmen. Při takovém útoku dochází k podvržení informace poskytované systémem doménových jmen. Když by si uživatel vyžádal adresu IP, tak obdrží adresu IP vzdáleného počítače útočníka. Uživatel si přijatou informaci nemůže žádným způsobem ověřit a slepě věří tomu, co dostal. Na vzdáleném počítači útočníka může běžet služba podobná službě původní, která z uživatele může získat důležité informace. Jsou to útoky spíše experimentální, než-li ojedinělé, neboť útočníci zatím zpracovávají útočiště jinými protokoly a způsoby. Přesto bylo třeba vyvinout mechanismus zabezpečení dříve, než by se rozpoutala případná vlna útoků. Takových mechanismů bylo vyvinuto několik, ale nejdiskutovanějším z nich je ten, který byl nasazen v systému doménových jmen, a jehož jméno se usadilo v názvu této práce.

Zabezpečený systém doménových jmen je takový systém doménových jmen, který zejména následuje bezpečnostní cíle. Vůči původnímu systému doménových jmen zachovává silnou zpětnou kompatibilitu. S jeho asistencí je ověřitelná nejen pravost informace, ale i její neexistence. Pro ověřitelnost informace zavádí princip elektronického podpisu z oboru asymetrické kryptografie. Pro ověřitelnost neexistence využívá již zavedenou ověřitelnost informace, kde touto informací je oznámení o tom, že daná informace neexistuje. S využitím informačního potenciálu systému doménových jmen distribuuje veřejné klíče stejným způsobem jako poskytuje informace. Veřejné klíče se stávají důvěryhodnými tím, že se podepisují důvěryhodnými soukromými klíči. Prvním soukromým klíčem je soukromý klíč autority. Systém doménových jmen má pak jistou podobnost s certifikační autoritou.

1.1 Struktura práce

Čtenáři se doporučuje text číst tak, jak jde za sebou, pro důkladné pochopení souvislostí. Kapitoly byly napsány ve snaze zachovat logický sled, vyjadřovací přesnost a jednotnou terminologii.

Druhá kapitola vysvětluje principy systému doménových jmen. Uvedené principy sahají více do hloubky, protože jejich povaha vyžaduje manipulaci s daty přímo na úrovni bitů a zároveň jejich důkladné pochopení je důležité pro jakýkoli vývoj v oblasti těchto systémů.

Třetí kapitola vysvětluje principy zabezpečeného systému doménových jmen. Jelikož je toto rozšíření závislé na spoustě dalších mechanismů, mezi které patří i tradiční systém doménových jmen, tak je jim věnován prostor v předchozí kapitole.

Čtvrtá kapitola představuje celou řadu technologií, pomocí kterých byla implementace vytvořena. Zaměřuje se zejména na jejich vzájemnou ovlivnitelnost, výhody, nevýhody a důvody pro jejich výběr. Neodmyslitelnou komponentou je protokol HTTP. Jazykovými prostředky byly XHTML, PHP, SQL, JavaScript a AJAX. Technologie kryptografické zaujímali hašovací algoritmus, asymetriká kryptografie a knihovna OpenSSL.

Pátá kapitola pojednává o společných rysech imlementace, neboť je pro svou nezávislost a rozsáhlost rozdělena na dvě následující kapitoly. Kapitola dále shrnuje vývoj a testování implementace.

Šestá kapitola vysvětluje první část imlementace, kterou je knihovna. Provází implementací protokolu, kódování, dekódování, algoritmů a dalších.

Sedmá kapitola vysvětluje druhou část implementace, kterou je aplikace. Provází implementací vyrovnávací paměti, grafického rozhraní, řešení dotazu, ověřování informací a dalších.

Osmá kapitola shrnuje dosavadní vývoj, získaný přínos a diskutuje další vývoj.

1.2 Konvence pro tento dokument

V rámci komunikačního modelu „dotaz – odpověď“ se nebude moci používat pojem dotaz z důvodu kolize se stejnojmenným datovým elementem. Namísto pojmu dotaz se bude používat pojem **odchozí zpráva** a pro konzistenci se namísto pojmu odpověď bude používat pojem **příchozí zpráva**.

Vzhledem k tomu, že specifikace pro tvorbu jmen polí a jmen jejich hodnot nemají žádnou konvenci, tak jsou mnohdy nekonzistentních tvarů. Ačkoli by bylo užitečné pole a jejich hodnoty pojmenovat lépe, tak jsou přesto zachována dle specifikace.

Veškerá data jsou zalomena po 8 nebo 16 bitech. Pole s pevnou délkou mají světle šedé pozadí; a počet jejich řádků a sloupců odpovídá jejich velikosti. Pole s proměnnou délkou mají tmavě šedé pozadí; a počet jejich řádků a sloupců neodpovídá jejich velikosti.

2 Systém doménových jmen

Námětem pro vznik doménového jména (dále jen „jméno“) byl fakt, že si člověk většinou hůře pamatuje číslo než text. Číslem je právě adresa **IP**[1] serveru a textem je jméno s ní spojené. U této příležitosti byl tvar jména navržen tak, aby zároveň umožňoval dobrou orientaci na internetu. Ve volbě jména je poskytována jistá volnost, avšak jméno musí být v rámci internetu jedinečné a v zájmu jeho majitele by mělo být dobře zapamatovatelné. Je-li hůře zapamatovatelné, pak jméno postrádá význam. Nicméně, existují jména, která se svou složitostí nepodobají ani číslům.

Příčinou vzniku systému doménových jmen byla nutnost převodů jmen na adresy IP, přičemž v současné době s přibývajícími požadavky nabývá systém doménových jmen i dalších funkcí. **Systém doménových jmen** (**DNS**[2]) je označením pro poměrně rozsáhlý systém, který zahrnuje jmenný prostor a technické vybavení. Technické vybavení je tvořeno sítí fyzických serverů, na kterých běží aplikační servery DNS (dále jen „servery“). Servery mezi sebou komunikují protokolem DNS aplikační vrstvy modelu **OSI**[3] vnořeným do protokolu **UDP**[4] či **TCP**[5] transportní vrsty modelu OSI s adresou portu **53**.

Jak ve světě, tak i v oblasti počítačů má princip hierarchie nepředstavitelný význam. Význam je u systému doménových jmen rovnou několikanásobný. Jmenný prostor, serverový prostor, zónový prostor a data jsou hierarchické; umístění jmen, převody jmen a rozložení zátěže jsou distribuované.

2.1 Jména a hierarchie jmen

V grafickém vyjádření je jmenný prostor uspořádán do stromu. Kořenový uzel stromu se podílí na tvorbě **kořenové domény**, který je tvořen prázdným textem. Uzly v nižší úrovni se podílejí na tvorbě domén nejvyšší úrovně nebo též **domén první úrovně**, které jsou tvořeny pokud možno co nejkratším textem. **Domény nejvyšší úrovně** (**TLD**[6]) jsou buď tematické nebo státní. Uzly v ještě nižší úrovni se podílejí na tvorbě **domén druhé úrovně**.

Jméno je tedy tvořeno cestou mezi uzly do kořenového uzlu. V textovém vyjádření je jméno uspořádáno do řetězce zapisovaného od nejnižšího uzlu a uzly jsou oddělené tečkou. V některých aplikacích je kořenový uzel zbytečné uvádět. Jak bude ukázáno v dalších odstavcích, tak uzel může být tvořen až 63 znaky, jméno může být tvořeno až 127 uzly a zároveň až 255 znaky.

[1] IP – Internet Protocol
[2] DNS – Domain Name System
[3] OSI – Open Systems Interconnection
[4] UDP – User Datagram Protocol
[5] TCP – Transmission Control Protocol
[6] TLD – Top-Level Domain

2.2 Zóny a hierarchie zón

Jmenný prostor je rozdělen do zón. **Zóna** je část jmenného prostoru, která je delegována na samostatný server. Zóna obvykle ve vyšších úrovních obsahuje jen jeden uzel a v nižších úrovních uzlů více. Data zóny jsou uložena jako **zónový soubor**.

2.3 Servery a hierarchie serverů

Servery mohou mít různé chování z pohledu zajištění dostupnosti. **Primární server** obsahuje data původní a poskytuje **autoritativní odpovědi**. Každá zóna má právě jeden primární server. **Sekundární server** obsahuje data zkopírovaná z primárního serveru; nahrazuje primární server při jeho výpadku, obnovuje data při jeho poškození, distribuuje zátěž při jeho přetížení a poskytuje též autoritativní odpovědi. Zóna by měla mít alespoň jeden sekundární server. **Vyrovnávací server** obsahuje odpovědi vytvořené již dříve; snižuje zátěž všech serverů a poskytuje **neautoritativní odpovědi**.

Server kořenové domény je vlastně pojmem pro 13 serverů kořenové domény, z nichž každý je spravován jiným operátorem a i tento může být pojmem pro více serverů kořenové domény, které se nacházejí na různých místech. Tři z nich se dokonce nacházejí v Praze [21].

Server kořenové domény potřebuje znát pouze adresy serverů svých domén nejvyšší úrovně, server domény nejvyšší úrovně potřebuje znát pouze adresy serverů svých domén druhé úrovně a tak dále.

2.4 Dotazování

Aby počítač mohl systém doménových jmen využívat, musí obsahovat klienta DNS (dále jen „klient“) nebo též **resolver**. **Klient** odesílá odchozí zprávy serverům a přijímá od nich příchozí zprávy. Klient by měl znát adresu IP lokálního serveru, kterou má buď pevně nastavenou, anebo získanou ze serveru **DHCP**[7]. Každý server zná adresy IP serverů kořenové domény.

Servery mohou mít různé chování z pohledu zajištění odpovědí. **Rekurzivní server**, když nezná platnou odpověď, stává se zároveň klientem a dotaz přeposílá serveru, který by platnou odpověď mohl znát. **Autoritativní server**, když nezná platnou odpověď, odpoví seznamem adres serverů, které by platnou odpověď mohly znát. Oba servery, když znají platnou odpověď, odpoví platnou odpovědí. Každý server kořenové domény a každý server domény nejvyšší úrovně je autoritativní kvůli snížení zátěže a každý lokální server by měl být rekurzivní.

Klient nemusí pokládat dotaz, jestliže ten samý dotaz položil již dříve a zná na něj odpověď, jejíž živost ještě nevypršela.

2.5 Data a hierarchie dat

Veškerá data se skládají z elementů, které mají hierarchickou skladbu. Na nejvyšší úrovni se vyjímá zpráva. **Zpráva** se skládá z hlavičky a sekcí. **Sekce** se skládá z dotazů nebo

[7] DHCP – Dynamic Host Configuration Protocol

záznamů. **Hlavička**, **dotaz** a **záznam** (**RR**[8]) se skládají z polí. **Pole** se může skládat z dalších polí nebo příznaků. Na nejnižší úrovni se tedy vyskytují **příznaky**.

Záznamy se mohou sdružovat so sad. **Sada** (**RRset**[9]) se skládá ze záznamů stejného vlastníka, stejné třídy a stejného typu, přičemž musejí mít stejnou živost.

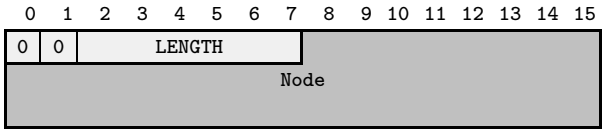
Pole je vlastně pojmenovaná pozice ve zprávě, která může nabývat obecných nebo pojmenovaných hodnot. **Pojmenovaná hodnota** je hodnota, která má jméno a **obecná hodnota** je hodnota, která nemá jméno.

2.6 Pole

Pole některých typů jsou obsažena v záznamech více typů. Pro bližší popis bylo vybráno pole typu `Name`.

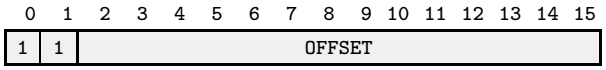
2.6.1 Pole typu `Name`

Jelikož může být zpráva nad protokolem UDP tvořena pouze 512 bajty, tak byl vyvinut kompresní mechanismus, ve kterém může být každý další výskyt stejného jména nahrazen ukazatelem na toto jméno. Kořenový uzel je nutné též uvažovat. Pokud následuje uzel, pak je znázorněn na obrázku (2.1).



Obrázek 2.1: Uzel v poli typu `Name` [30, str. 30]

Pole `LENGTH` určuje délku následujícího uzlu. Jelikož je délka uzlu tvořena 6 bity, tak uzel může být tvořen pouze 63 znaky. Pole `Node` určuje uzel jména. Pokud následuje ukazatel, pak je znázorněn na obrázku (2.2).



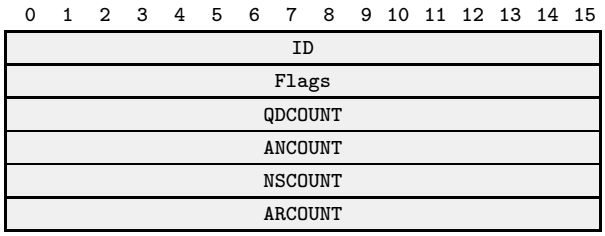
Obrázek 2.2: Ukazatel v poli typu `Name` [30, str. 30]

Pole `OFFSET` určuje pozici jména; `0` pro první bajt zprávy apod.

2.7 Hlavička

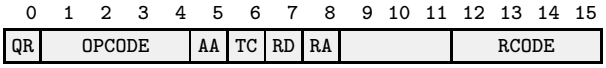
Hlavička je znázorněna na obrázku (2.3).

[8] RR – Resource Record
[9] RRset – Resource Record Set



Obrázek 2.3: DNS hlavička [30, str. 26]

Pole `ID` nastavuje klient a kopíruje server a určuje identifikátor zprávy. Pole `Flags` nastavuje klient i server a skládá se z příznaků; přičemž doposud podporované příznaky jsou znázorněny na obrázku (2.4). Pole `QDCOUNT` nastavuje klient a kopíruje server a určuje počet dotazů v sekci `QUESTION`; nabývá obvykle hodnoty `1`. Pole `ANCOUNT` nastavuje server a určuje počet záznamů v sekci `ANSWER`. Pole `NSCOUNT` nastavuje server a určuje počet záznamů typu `NS` v sekci `AUTHORITY`. Pole `ARCOUNT` nastavuje server a určuje počet záznamů v sekci `ADDITIONAL`.



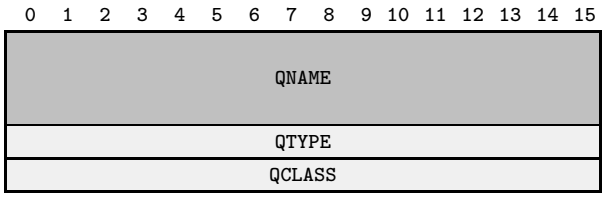
Obrázek 2.4: Příznaky v DNS hlavičce [30, str. 26]

Příznak `QR`[10] nastavuje klient i server a určuje typ zprávy; a nabývá hodnoty `QUERY/0` pro odchozí zprávu nebo `RESPONSE/1` pro příchozí zprávu. Příznak `OPCODE` nastavuje klient a kopíruje server a určuje typ dotazu; a nabývá hodnoty `QUERY/0` pro standardní dotaz, `IQUERY/1` pro inverzní dotaz nebo `STATUS/3` pro dotaz na stav serveru. Příznak `AA`[11] nastavuje server a určuje, zda je server autoritativní; a nabývá hodnoty `OFF/0` pro neautoritativní server nebo `ON/1` pro autoritativní server. Příznak `TC`[12] nastavuje server a určuje, zda je příchozí zpráva zkrácena; a nabývá hodnoty `OFF/0` pro nezkrácenou příchozí zprávu nebo `ON/1` pro zkrácenou příchozí zprávu. V takovém případě by měl klient dotaz odeslat znovu raději nad protokolem TCP. Příznak `RD`[13] nastavuje klient a kopíruje server a určuje, zda má být server rekurzivní; a nabývá hodnoty `OFF/0` pro nerekurzivní server nebo `ON/1` pro rekurzivní server. Příznak `RA`[14] nastavuje server a určuje, zda je server rekurzivní; a nabývá hodnoty `OFF/0` pro nerekurzivní server nebo `ON/1` pro rekurzivní server. Pole `RCODE`[15] nastavuje server a určuje návratový kód příchozí zprávy; a nabývá hodnoty `NOERROR/0` pro žádnou chybu, `FORMERR/1` pro nepodporvaný formát dotazu, `SERVFAIL/2` pro chybu serveru, `NXDOMAIN/3` pro absenci jména, `NOTIMP/4` pro nepodporovaný typ dotazu nebo `REFUSED/5` pro odmítnutý dotaz.

2.8 Dotaz

Dotaz je znázorněn na obrázku (2.5).

[10] QR – Query/Response
[11] AA – Authoritative Answer
[12] TC – TrunCation
[13] RD – Recursion Desired
[14] RA – Recursion Available
[15] RCODE – Response Code



Obrázek 2.5: Dotaz

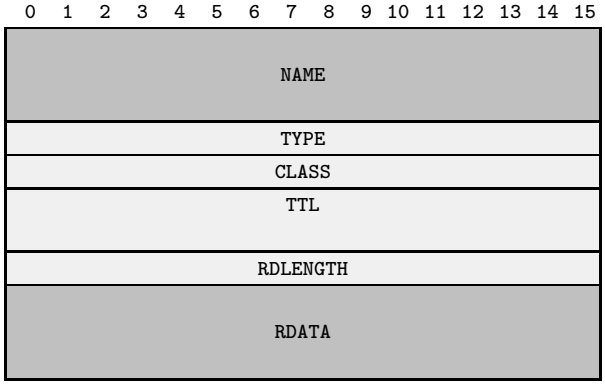
Pole `QNAME`[16] nastavuje klient a kopíruje server a určuje jméno dotazu; a vytváří se podle odstavce (2.6.1) o poli typu `Name`. Pole `QTYPE`[17] nastavuje klient a kopíruje server a určuje typ dotazu; nabývá hodnot z nadmnožiny hodnot pole `TYPE`. Pole `QCLASS`[18] nastavuje klient a kopíruje server a určuje třídu dotazu; nabývá hodnot z nadmnožiny hodnot pole `CLASS`.

2.9 Záznamy

Pro bližší popis byly vybrány záznamy typů `SOA`, `A`, `AAAA`, `CNAME`, `MX`, `NS` a `PTR`.

2.9.1 Záznam obecného typu

Záznam je znázorněn na obrázku (2.6).



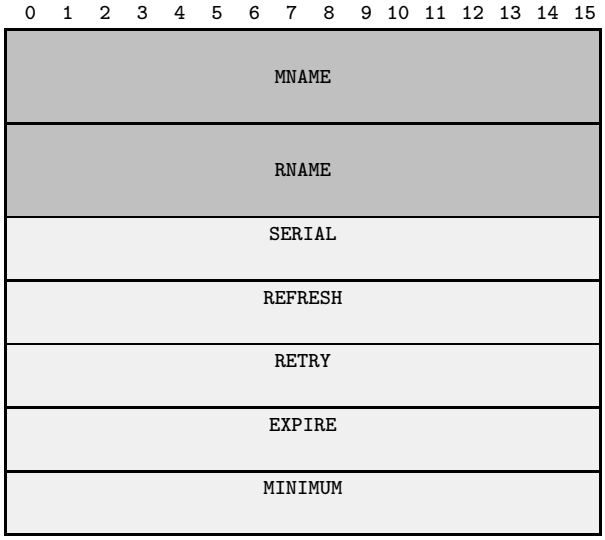
Obrázek 2.6: Záznam obecného typu

Pole `NAME` určuje vlastníka záznamu; a vytváří se podle odstavce (2.6.1) o poli typu `Name`. Pole `TYPE` určuje typ záznamu; a nabývá hodnot z podmnožiny hodnot pole `QTYPES`. Pole `CLASS` určuje třídu záznamu; a nabývá hodnot z podmnožiny hodnot pole `QCLASS`. Pole `TTL`[19] určuje živost záznamu. Pole `RDLENGTH`[20] určuje velikost pole `RDATA`. Pole `RDATA`[21] určuje data záznamu, která se liší od typu a někdy i třídy záznamu.

[16] QNAME – Query Name
[17] QTYPE – Query Type
[18] QCLASS – Query Class
[19] TTL – Time to Live
[20] RDLENGTH – Resource Data Length
[21] RDATA – Resource Data

2.9.2 Záznam typu SOA

Záznam typu SOA[22] uchovává informace o zóně. Musí se nacházet na začátku zónového souboru. Pole RDATA je znázorněno na obrázku (2.7).

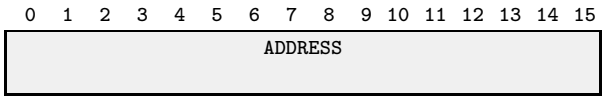


Obrázek 2.7: Pole RDATA v záznamu typu SOA [30, str. 19]

Pole MNAME určuje jméno zóny. Pole RNAME určuje e-mailovou adresu správce zóny, s tečkou namísto zavináče. Pole SERIAL určuje sériové číslo změny v zónovém souboru, často ve formátu relativního nebo absolutního času. Pole REFRESH určuje dobu, po které se sekundární server dotazuje primárního serveru na sériové číslo. Pole RETRY určuje dobu, po které se sekundární server dotazuje primárního serveru na sériové číslo, pokud komunikace s primárním serverem selhala. Pole EXPIRE určuje dobu, po které sekundární server zneplatní záznamy, pokud komunikace s primárním serverem selhala. Pole MINIMUM určuje výchozí živost záznamu.

2.9.3 Záznam typu A

Záznam typu A[23] uchovává adresu IPv4. Pole RDATA je znázorněno na obrázku (2.8).



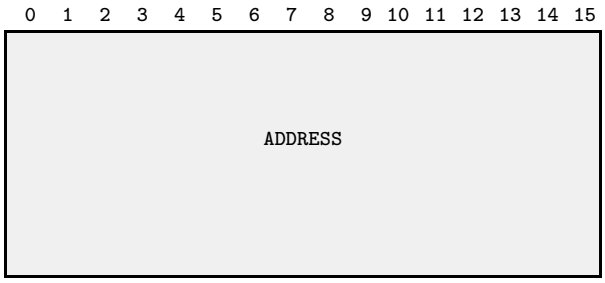
Obrázek 2.8: Pole RDATA v záznamu typu A [30, str. 20]

Pole ADDRESS určuje adresu IPv4.

2.9.4 Záznam typu AAAA

Záznam typu AAAA[24] uchovává adresu IPv6. Pole RDATA je znázorněno na obrázku (2.9).

[22] SOA – Start Of Authority
[23] A – Address
[24] AAAA - Address

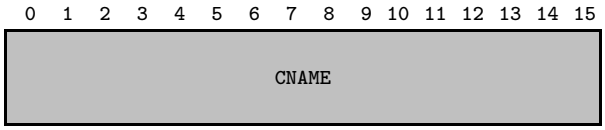


Obrázek 2.9: Pole `RDATA` v záznamu typu `AAAA` [41]

Pole `ADDRESS` určuje adresu IPv6.

2.9.5 Záznam typu `CNAME`

Záznam typu `CNAME`[25] uchovává alias. Pole `RDATA` je znázorněno na obrázku (2.10).

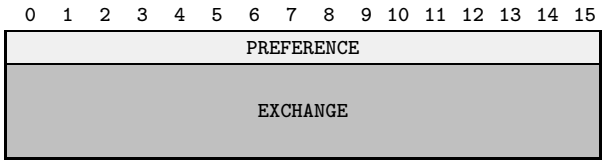


Obrázek 2.10: Pole `RDATA` v záznamu typu `CNAME` [30, str. 14]

Pole `CNAME` určuje alias.

2.9.6 Záznam typu `MX`

Záznam typu `MX`[26] uchovává jméno poštovního serveru a jeho prioritu. Pole `RDATA` je znázorněno na obrázku (2.11).

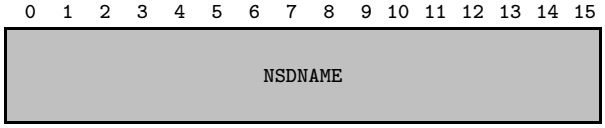


Obrázek 2.11: Pole `RDATA` v záznamu typu `MX` [30, str. 17]

Pole `PREFERENCE` určuje prioritu; přičemž nízká hodnota má vyšší prioritu. Pole `EXCHANGE` určuje poštovní server.

2.9.7 Záznam typu `NS`

Záznam typu `NS`[27] uchovává jméno serveru. Pole `RDATA` je znázorněno na obrázku (2.12).



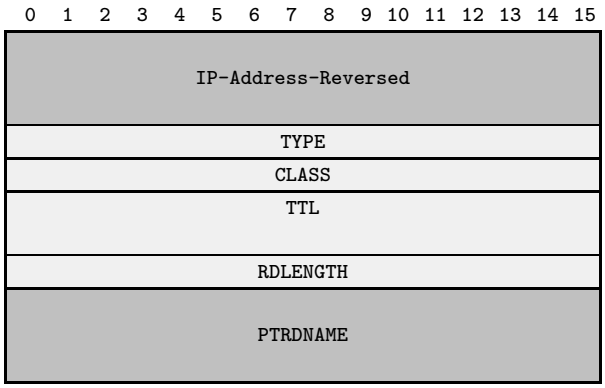
Obrázek 2.12: Pole `RDATA` v záznamu typu `NS` [30, str. 18]

[25] CNAME – Canonical Name
[26] MX – Mail Exchange
[27] NS – Name Server

Pole `NSDNAME` určuje jméno serveru.

2.9.8 Záznam typu `PTR`

Záznam typu `PTR`[28] převádí adresu IP na jméno. Záznam je znázorněn na obrázku (2.13).

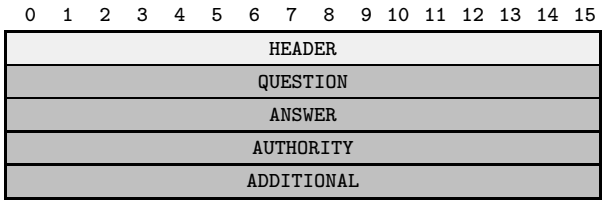


Obrázek 2.13: Záznam typu `PTR` [30, str. 18]

Pole `IP-Address-Reversed` určuje obrácenou adresu IP verze 4 i 6 ve formátu jména. Adresa IP verze 4 je zakončena `.in-addr.arpa.` a adresa IP verze 6 je zakončena `.ip6.arpa.` Pole `PTRDNAME` určuje jméno.

2.10 Zpráva

Zpráva je bitového charakteru, či přesněji řečeno, bajtového charakteru. Totiž, v případě potřeby bitových polí jsou bitová pole skládána do polí bajtových. Zpráva se v nejvyšší úrovni se skládá ze sekcí. Zpráva je znázorněna na obrázku (2.14).



Obrázek 2.14: Zpráva [30, str. 25]

Sekce `HEADER` určuje hlavičku. Sekce `QUESTION` se skládá z dotazů, sekce `ANSWER` z odpovědních záznamů, sekce `AUTHORITY` z autoritativních záznamů a sekce `ADDITIONAL` z přídavných záznamů.

2.11 Rizika

Na systému doménových jmen závisí spousta služeb. Systém doménových jmen umožní například přihlášení do bankovní aplikace, přihlášení do e-mailové schránky nebo zaslání zapomenutého hesla na e-mailovou adresu. Podle uvedených příkladů pak napadený systém doménových jmen umožní přihlášení do podvržené bankovní aplikace, přihlášení do

[28] PTR – Pointer

podvržené e-mailové schránky nebo zaslání zapomenutého hesla na podvrženou e-mailovou adresu. Z těchto rizik je zřejmé, že důležitost systému doménových jmen je značná.

První typ útoku využívá přijetí podvržené adresy serveru ve zprávě DHCP v případě lokální sítě. Jiný typ útoku využívá přijetí podvrženého pole `ID` v sekci `HEADER`. Další typ útoku, který byl představen teprve nedávno, využívá přijetí podvržených **glue** záznamů v sekci `ADDITIONAL` [24].

2.12 Standardizace

Systém doménových jmen je definován v četném počtu dokumentů **RFC**[29]. Samotná existence DNS sahá až do roku 1983. Dokument **RFC 882** [27] popisuje principy DNS, dokument **RFC 883** [28] popisuje implementaci DNS a definuje 14 typů záznamů, 18 typů dotazů, 2 třídy záznamů a 3 třídy dotazů.

Dokument **RFC 1034** [29] přepisuje RFC 882, dokument **RFC 1035** [30] přepisuje RFC 883 a navíc definuje 2 typy záznamů, 2 třídy záznamů a 2 třídy dotazů. Další dokumenty popisují nové typy záznamů.

[29] RFC – Request for Comments

3 Zabezpečený systém doménových jmen

Bezpečnostní rozšíření pro systém doménových jmen (**DNSSEC**[1]) je navrženo s ohledem na maximální kompatibilitu s DNS, tudíž novou funkcionalitu přidává vytvořením nových příznaků v hlavičce, vytvořením nových typů záznamů a mírnou modifikací protokolu DNS. Narozdíl od jiných zabezpečených protokolů je adresa portu ponechána beze změny. Obecně se k ověřování pravosti dat používá elektronický podpis, který vychází z asymetrického šifrování s potřebou soukromého a veřejného klíče. V případě rozšíření DNSSEC je držitelem soukromého klíče zóna. Pravost záznamů podepsaných soukromým klíčem lze ověřit klíčem veřejným. Veřejný klíč a podpis jsou právě součástí nových typů záznamů.

3.1 Veřejné klíče

Prolomení veřejného klíče či prozrazení soukromého klíče není stále vyloučeno. Obecně je prevence proti těmto neduhům obvykle realizována jako pravidelná výměna starého páru klíčů párem klíčů novým o velikosti odpovídající jeho důležitosti a době jeho vystavení. V otázce ohledně ověřování záznamu typu `DNSKEY` byla zavedena konvence koexistence dvou různých párů klíčů.

Veřejný klíč **ZSK**[2] (dále jen „zónový“) je určen pro ověření pravosti záznamu, a to včetně záznamu typu `DNSKEY`. Jelikož musí být podepsání záznamu a ověření pravosti záznamu s ním výpočetně nenáročné, tak musí být pár zónových klíčů krátký a jednoduchý, a tudíž musí být vystavován krátkou dobu.

Veřejný klíč **KSK**[3] (dále jen „klíčový“) je určen pro ověření pravosti záznamu typu `DNSKEY`. Jelikož je pár klíčových klíčů vystavován delší dobu, tak musí být větší a složitější, tudíž podepsání záznamu a ověření pravosti záznamu s ním je výpočetně náročnější.

Díky této koexistenci je možné pár zónových klíčů vyměnit bez obtěžování vyšší zóny. Nutno poznamenat, že může existovat i jen jeden pár klíčů bez újmy na správné funkci.

3.2 Schopnosti

Mezi schopnosti rozšíření DNSSEC patří **ověřování pravosti záznamu**, **ověřování absence záznamu** a **distribuování veřejného klíče**. Nutno poznamenat, že **utajení obsahu záznamu** mezi jeho schopnosti nepatří, neboť systém DNS poskytuje veřejně dostupné informace.

[1] DNSSEC – DNS Security Extensions
[2] ZSK – Zone Signing Key
[3] KSK – Key Signing Key

3.2.1 Distribuování veřejného klíče

Veřejný klíč lze získat několika způsoby. Prvním způsobem je získání z osobního rozhovoru, z e-mailu, z poštovní zásilky, z telefonního hovoru apod. Tento způsob je značně nepohodlný a má smysl ho používat jen v ojedinělých případech.

Pro druhý způsob slouží právě záznam typu `DNSKEY`. Ve všech případech je nutné ověřit pravost získaného veřejného klíče.

3.2.2 Ověřování pravosti záznamu

Řetězec důvěry je proces ověření pravosti záznamu a všech těch prostředků, které k němu vedou. Pravost záznamu je prokázána, pokud je podepsán soukromým zónovým klíčem. Pak pravost záznamu typu `DNSKEY` s veřejným zónovým klíčem je prokázána, pokud je podepsán soukromým klíčovým klíčem. Ověřit pravost záznamu typu `DNSKEY` s veřejným klíčovým klíčem lze několika způsoby.

Prvním způsobem je ověření osobním rozhovorem se správcem zóny, ověření zabezpečenou webovou stránkou zóny, ověření elektronicky podepsaným e-mailem od správce zóny apod.

Pro druhý způsob slouží právě záznam typu `DS`. Pravost záznamu typu `DNSKEY` s veřejným klíčovým klíčem je prokázána, pokud se záznam typu `DS` s jeho otiskem nachází v nadřazené zóně. V nadřazené zóně se proces opakuje. Pravost záznamu typu `DS` je prokázána, pokud je podepsán soukromým zónovým klíčem; a tak dále. V důsledku toho se pravost záznamu typu `DNSKEY` s veřejným klíčovým klíčem v kořenové zóně považuje za prokázanou. Tento způsob je možný pouze v případě, že nadřazená zóna podporuje rozšíření DNSSEC.

Třetím způsobem je **postranní validace domén** (**DLV**[4]). Pravost záznamu typu `DNSKEY` s veřejným klíčovým klíčem je prokázána, pokud se záznam typu `DLV` s patřičným obsahem nachází ve speciální zóně. Tento způsob je dočasným řešením v situaci, kdy nadřazená zóna rozšíření DNSSEC ještě nepodporuje.

Klient nemusí ověřovat pravost záznamu, pokud jeho pravost již ověřoval a jeho živost ještě nevypršela.

3.2.3 Ověřování absence záznamu

Absence záznamu je prokázána buď, pokud není jeho typ obsažen v parametru `Type-Bit-Maps` záznamu typu `NSEC` nebo `NSEC3` stejného vlastníka.

Anebo pokud existuje záznam typu `NSEC` vlastníka předcházejícího a bezprostředně za ním záznam typu `NSEC` nebo `NSEC3` vlastníka následujícího. V důsledku toho musejí být vlastníci záznamů v zónovém souboru uspořádáni abecedně.

3.3 Kanonická forma a pořadí záznamů

Používanou technikou pro rozložení zátěže na více serverů je rozhodnutí v náhodný nebo jinak zvolený výběr jednoho z nich. Teoreticky je možné toto rozhodnutí provést na straně klienta náhodným výběrem záznamu ze sady, pomocí které tyto adresy již obdržel; nebo na straně serveru nahodným seřazením záznamů v sadě, pomocí které tyto adresy teprve poskytne. Jistota takového rozhodnutí je pochopitelně vyšší na straně serveru, tudíž se

[4] DLV – Domain Lookaside Validation

náhodné seřazení záznamů v sadě používá výhradně. Jelikož není žádoucí, aby pro každou sadu jako kombinaci existoval podpis nebo se tato sada jako kombinace pokaždé před odesláním podepisovala, tak se podepisuje pouze sada, jejíž záznamy jsou seřazené jedním předem dohodnutným způsobem. Takto dohodnutá řazení se nazývají kanonická, a patří mezi ně **kanonická forma záznamů**, **kanonické pořadí záznamů v zóně** a **kanonické pořadí záznamů v sadě**.

3.3.1 Kanonická forma záznamu

Kanonická forma záznamu je taková forma záznamu, která splňuje následující podmínky [3, str. 19].

- Pole typu `Name` je plně rozvinuto a není kompresované.
- V poli `NAME` jsou všechna velká písmena nahrazena korespondujícími malými písmeny.
- V poli `RDATA` v polích typu `Name` jsou všechna velká písmena nahrazena korespondujícími malými písmeny, pokud je záznam typu `NS`, `MD`, `MF`, `CNAME`, `SOA`, `MB`, `MG`, `MR`, `PTR`, `HINFO`, `MINFO`, `MX`, `HINFO`, `RP`, `AFSDB`, `RT`, `SIG`, `PX`, `NXT`, `NAPTR`, `KX`, `SRV`, `DNAME`, `A6`, `RRSIG`, `NSEC` nebo `NSEC3`.
- V poli `NAME` je uzel `*` ponechán.
- Pole `TTL` je nastaveno na pole `Minimum` ze záznamu typu `SOA` nebo na pole `Original-TTL` ze záznamu typu `RRSIG`.

Kanonická forma záznamu je vyžadována pro záznam typu `RRSIG`.

3.3.2 Kanonické pořadí záznamů v zóně

Záznamy jsou řazeny podle pole `NAME` tak, že se porovnává uzel s uzlem na stejné pozici zprava a v další úrovni se porovnává bajt s bajtem na stejné pozici v uzlu zleva, přičemž chybějící bajt řadí záznam vpřed a velká písmena se vyhodnocují jako malá [3, str. 18]. Kanonické pořadí záznamů v zóně je vyžadováno pro záznamy typu `NSEC` a `NSEC3`.

3.3.3 Kanonické pořadí záznamů v sadě

Sada aneb záznamy se stejným polem `NAME`, `CLASS` a `TYPE` jsou řazeny podle pole `RDATA` tak, že se porovnává bajt s bajtem na stejné pozici v poli `RDATA` zleva, přičemž chybějící bajt řadí záznam vpřed [3, str. 20]. Sada nesmí obsahovat duplicitní záznamy, jinak nesmějí být použity pro ověřování nebo mohou být ignorovány. Kanonické pořadí záznamů v sadě je vyžadováno pro záznam typu `RRSIG`.

3.4 Rozšířený systém doménových jmen

Vzhledem k navýšení počtu záznamů a jejich velikosti může příchozí zpráva převýšit maximální velikost zprávy nad protokolem UDP. Proto je nutné systém DNS rozšířit ještě více.

Mechanismus rozšíření pro DNS (**EDNS**[5]) je též navržen s ohledem na maximální kompatibilitu s DNS, tudíž novou funkcionalitu přidává pouhým vytvořením nového záznamu typu `OPT`. Jeho pole ale nabývají naprosto odlišných významů a vyskytuje se výhradně v sekci `ADDITIONAL` podle odstavce (3.7.1) o záznamu typu `OPT`. Implementace by tento záznam měly chápat jako rozšíření DNS hlavičky a ne jako plnohodnotný záznam.

[5] EDNS – Extension Mechanisms for DNS

3.5 Pole

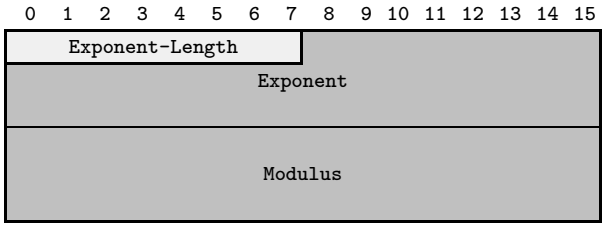
Rozšíření DNSSEC specifikuje záznamy nových typů a s nimi zároveň i pole nových typů. Pole některých typů jsou obsažena v záznamech více typů. Pro bližší popis byly vybrány pole typů `Key-Tag`, `Public-Key`, `Digest-Type`, `Algorithm`, `Type-Bit-Maps` a `Hash-Algorithm`.

3.5.1 Pole typu `Key-Tag`

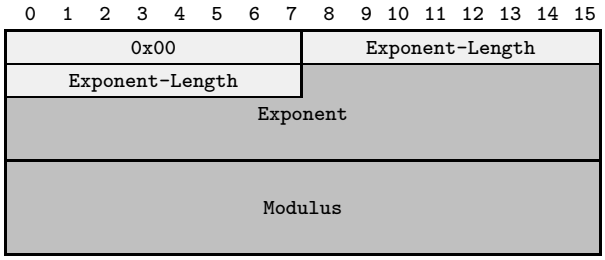
Pole typu `Key-Tag` určuje značku veřejného klíče. Je obsaženo v záznamech typu `RRSIG` a `DS`. Spolu s poli typu `Name` a `Algorithm` slouží pro zúžení výběru, nikoli pro identifikaci, záznamu typu `DNSKEY` [3, str. 25]. V mnoha případech zúží výběr právě na jeden záznam typu `DNSKEY`. Teoreticky je ale možné, aby dva různé záznamy typu `DNSKEY` měli stejná pole typu `Name`, `Algoritmus` a `Key-Tag`. Implementace nesmějí předpokládat, že se jedná o unikátní identifikátor. Vytváří se podle odstavce (3.8.1) o značkovacím algoritmu.

3.5.2 Pole typu `Public-Key`

Pole typu `Public-Key` určuje veřejný klíč. Je obsaženo v záznamu typu `DNSKEY`. Pro podepisovací algoritmy rodiny `RSA` je znázorněn na obrázku (3.1) pro malý veřejný exponent a na obrázku (3.2) pro velký veřejný exponent.



Obrázek 3.1: Pole typu `Public-Key` pro malý veřejný exponent [12, 14, str. 2, 3]



Obrázek 3.2: Pole typu `Public-Key` pro velký veřejný exponent [12, 14, str. 2, 3]

Pole `Exponent-Length` určuje délku pole `Exponent`. Pole `Exponent` určuje veřejný exponent veřejného klíče; o délce až 4096 bitů. Pole `Modulus` určuje modulus veřejného klíče; o délce až 4096 bitů.

Veřejný exponent by měl být malý. Jestliže má být veřejný klíč použit pouze pro autentizaci, jako tomu je v rozšíření DNSSEC, pak délka veřejného exponentu o 3 bajtech je přijatelná [12, 14, str. 3, 4]. Pro podepisovací algoritmy typu `RSAMD5`, `RSASHA1` a `RSASHA256` se délka modulu pohybuje mezi 512 až 4096 bity [12, 14, str. 2, 3]. Pro podepisovací algoritmus typu `RSASHA512` se délka modulu pohybuje mezi 1024 až 4096 bity [23, str. 3].

3.5.3 Pole typu `Digest-Type`

Pole typu `Digest-Type` určuje typ otiskového algoritmu. Je obsaženo v záznamu typu `DS`. Nabývá doposud podporovaných hodnot podle tabulky (3.1). Otiskový algoritmus podrobně vystihuje odstavec (3.8.2) o otiskovém algoritmu.

číslo	jméno	význam	stav
0	*rezervováno*		
1	`SHA1`	SHA-1	**povinný**
2	`SHA256`	SHA-256	**povinný**
3	`GOST`	GOST R 34.11-94	volitelný
4 – 255	*nepřiřazeno*		

Tabulka 3.1: Hodnoty pole typu `Digest-Type` [18]

3.5.4 Pole typu `Algorithm`

Pole typu `Algorithm` určuje typ podpisového algoritmu. Je obsaženo v záznamech typu `DNSKEY`, `RRSIG` a `DS` v případě rozšíření DNSSEC; ale též v záznamech typu `SIG` a `KEY` v případě rozšíření **TSIG**[6] [3, str. 24]. Nabývá doposud podporovaných hodnot podle tabulky (3.2). Podpisový algoritmus podrobně vystihuje odstavec (3.8.5) o podpisovém algoritmu.

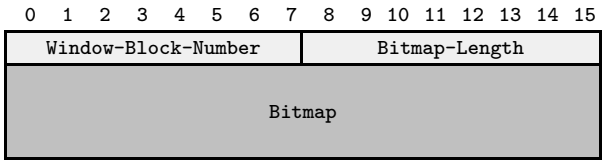
číslo	jméno	význam	stav
0	*rezervováno*		
1	`RSAMD5`	RSA/MD5	nedoporučený
2	`DH`	Diffie – Hellman	
3	`DSA`	DSA/SHA-1	volitelný
4	`ECC`	Elliptic Curve	
5	`RSASHA1`	RSA/SHA-1	**povinný**
6	`DSA-NSEC3-SHA1`	DSA pro záznam typu `NSEC3`	**doporučený**
7	`RSASHA1-NSEC3-SHA1`	RSA/SHA-1 pro záznam typu `NSEC3`	**doporučený**
8	`RSASHA256`	RSA/SHA-256	**doporučený**
9	*nepřiřazeno*		
10	`RSASHA512`	RSA/SHA-512	**doporučený**
11	*nepřiřazeno*		
12	`ECC-GOST`	GOST R 34.10-2001	volitelný
13 – 122	*nepřiřazeno*		
123 – 251	*rezervováno*		
252	`INDIRECT`	Indirect	
253	`PRIVATEDNS`	Private – Domain Name System	volitelný
254	`PRIVATEOID`	Private – OID	volitelný
255	*rezervováno*		

Tabulka 3.2: Hodnoty pole typu `Algorithm` [20]

3.5.5 Pole typu `Type-Bit-Maps`

Pole typu `Type-Bit-Maps` určuje seznam typů záznamů. Je obsaženo v záznamech typu `NSEC` a `NSEC3`. Skládá se z oken; kde okno je znázorněno na obrázku (3.3).

[6] TSIG – Transaction Signature



Obrázek 3.3: Okno v poli typu `Type-Bit-Maps` [3, str. 13]

Pole `Window-Block-Number` určuje rozsah čísel typů záznamů pro pole bitů; `0` pro 0 – 255, `1` pro 256 – 511, … a `255` pro 65280 – 65535. Pole `Bitmap-Length` určuje délku pole `Bitmap`; nabývá hodnot `1` až `32`. Pole `Bitmap` určuje pole bitů.

V poli bitů pozice bitu koresponduje s číslem typu záznamu. Bit s hodnotou `1` indikuje přítomnost typu záznamu a bit s hodnotou `0` indikuje absenci typu záznamu. Okno k pozicím bitu přidává násobky 256. Například pro okno `0` bit o pozici `1` koresponduje s typem záznamu `A/1`, bit o pozici `2` s typem záznamu `NS/2`; pro okno `1` bit o pozici `1` koresponduje s typem záznamu `257` a bit o pozici `2` s typem záznamu `258`. Prázdná okna nesmějí být přítomna.

3.5.6 Pole typu `Hash-Algorithm`

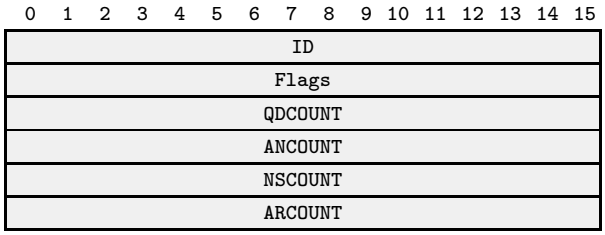
Pole typu `Hash-Algorithm` určuje typ iterovaného otiskového algoritmu. Je obsaženo v záznamu typu `NSEC3`. Nabývá doposud podporovaných hodnot podle tabulky (3.3). Iterovaný otiskový algoritmus podrobně vystihuje odstavec (3.8.11) o iterovaném otiskovém algoritmu.

číslo	jméno	význam	stav
0	*rezervováno*		
1	`SHA1`	SHA-1	**povinný**
2 – 255	*nepřiřazeno*		

Tabulka 3.3: Hodnoty pole typu `Hash-Algorithm` [17]

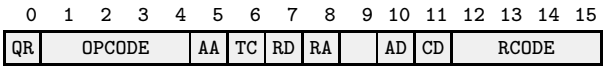
3.6 Hlavička

Rozšíření DNSSEC mírně modifikuje formát hlavičky. Zpráva je znázorněna na obrázku (3.4).



Obrázek 3.4: DNSSEC hlavička [30, str. 26]

Pole `Flags` se skládá z příznaků; přičemž doposud podporované příznaky jsou znázorněny na obrázku (3.5). Pole `ARCOUNT` nastavuje server a navíc i klient a určuje počet záznamů v sekci `ADDITIONAL`.



Obrázek 3.5: Příznaky v DNSSEC hlavičce [19]

Rozšíření DNSSEC do hlavičky přidává dva příznaky. Příznak `AD`[7] nastavuje server a určuje, zda je server validující; a nabývá hodnoty `OFF/0` pro nevalidující server nebo `ON/1` pro validující server. Příznak `CD`[8] nastavuje klient a kopíruje server a určuje, zda má být server validující; a nabývá hodnoty `OFF/0` pro nevalidující server nebo `ON/1` pro validující server. Pole `RCODE` nastavuje server a určuje návratový kód odpovědi; a nabývá hodnoty `SERVFAIL/2` navíc i pro neplatnou odpověď nebo pro podpis, jehož životnost již vypršela.

3.7 Záznamy

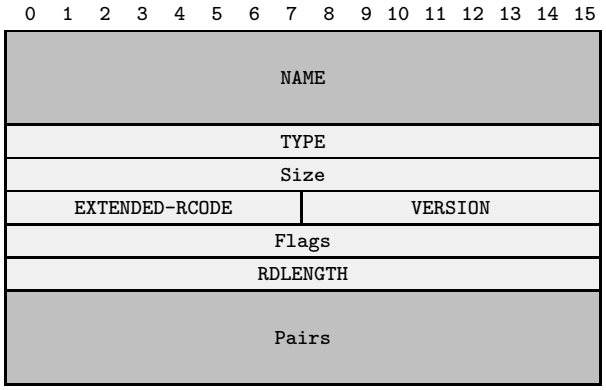
Pro bližší popis byly vybrány záznamy typů `OPT`, `DS`, `RRSIG`, `NSEC` a `NSEC3`. Hodnoty pole typu `TYPE` se nacházejí v tabulce (3.4).

číslo	jméno	význam
41	OPT	
43	DS	Delegation Signer
46	RRSIG	Resource Record Signature
47	NSEC	Next Secure
48	DNSKEY	DNS Key
50	NSEC3	Next Secure 3
51	NSEC3PARAM	Next Secure 3 Parameters

Tabulka 3.4: Hodnoty pole typu `TYPE` [3]

3.7.1 Záznam typu `Opt`

Záznam typu `OPT` je určen pro rozšíření hlavičky systému DNS. Záznam je znázorněn na obrázku (3.6).



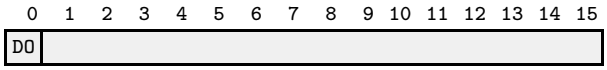
Obrázek 3.6: Záznam typu `OPT` [7, str. 3]

Pole `NAME` nastavuje klient a kopíruje server a neurčuje nic; a nabývá pouze prázdného textu. Pole `TYPE` nastavuje klient a kopíruje server a určuje typ záznamu, tedy `OPT`. Pole

[7] AD – Authenticated Data
[8] CD – Checking Disabled

`Size` nastavuje klient i server a určuje maximální velikost zprávy nad protokolem UDP. Pole `EXTENDED-RCODE`[9] nastavuje server a určuje rozšířený návratový kód odpovědi. Pole `VERSION` nastavuje klient i server a určuje verzi rozšíření EDNS; a nabývá hodnoty `EDNS0/0` pro verzi `EDNS0`. Pole `Flags` se skládá z příznaků; přičemž doposud podporované příznaky jsou znázorněny na obrázku (3.7). Pole `RDLENGTH` nastavuje klient i server a určuje velikost sekce `Pairs`. Sekce `Pairs` zastupuje rozšířená data a skládá se z voleb.

Je zřejmé, že pole `CLASS` je nahrazeno polem `Size` a pole `TTL` je nahrazeno poli `VERSION`, `Flags` a `EXTENDED-RCODE`.

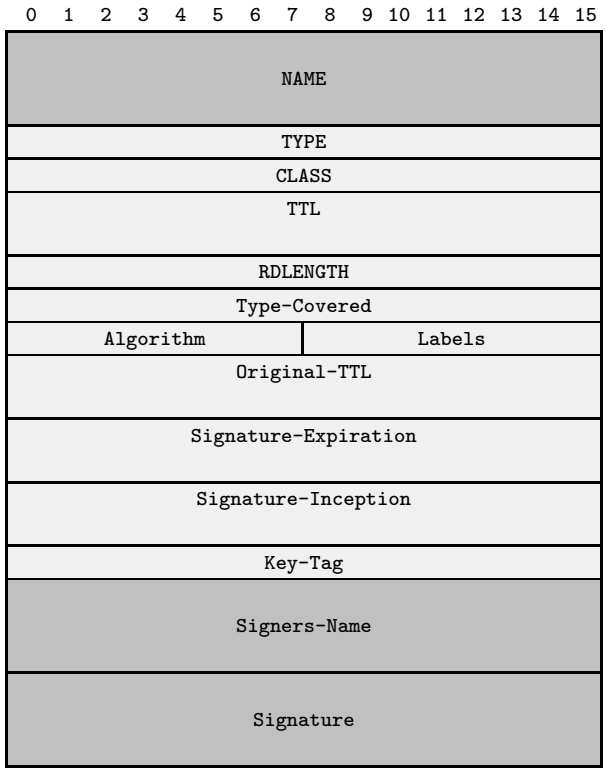


Obrázek 3.7: Příznaky v záznamu typu `OPT` [19]

Rozšířený příznak `DO`[10] nastavuje klient a kopíruje server a určuje, zda klient podporuje rozšíření DNSSEC a zda má server k sadě připojit i záznam typu `RRSIG` s podpisem této sady; a nabývá hodnoty `OFF/0` pro nepodporované rozšíření DNSSEC nebo `ON/1` pro podporované rozšíření DNSSEC.

3.7.2 Záznam typu `RRSIG`

Záznam typu `RRSIG`[11] je určen pro uchování podpisu sady. Není určen k uložení libovolného podpisu bez vztahu k rozšíření DNSSEC. Záznam je znázorněn na obrázku (3.8).



Obrázek 3.8: Záznam typu `RRSIG` [3, str. 7]

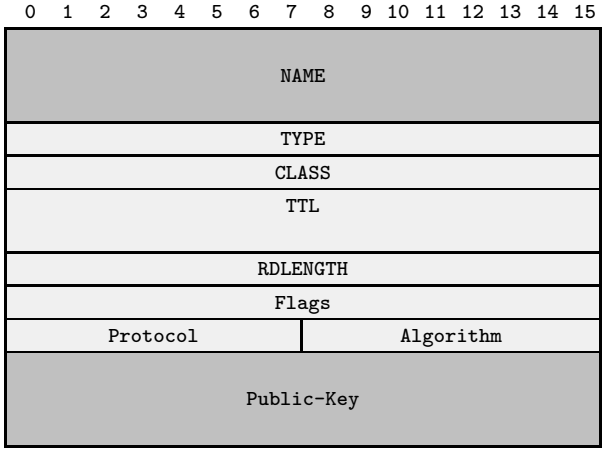
[9] EXTENDED-RCODE – Extended Response Code
[10] DO – DNSSEC OK
[11] RRSIG – DNSSEC Signature

Pole `CLASS` je vůči poli `TYPE` nezávislé; a musí být stejné jako pole `CLASS` v sadě. Pole `TTL` musí být stejné jako pole `TTL` v sadě. Pole `Type-Covered` určuje typ sady, přesněji pole `TYPE` v sadě; a musí být stejné jako pole `TYPE` v sadě. Pole `Algorithm` určuje typ podpisového algoritmu a formát pole `Signature`; vytváří se podle odstavce (3.5.4) o poli typu `Algorithm`; a nabývá doposud podporovaných hodnot podle výše uvedené tabulky (3.2). Pole `Labels` určuje počet uzlů, kromě kořenového a zástupného, v poli `NAME`; a nabývá hodnot menších nebo rovných počtu uzlů v poli `NAME`. Existuje kvůli tomu, že server může zástupný uzel v poli `NAME` nahradit. Pole `Original-TTL` určuje původní živost, vlastně i původní živost sady; a musí být stejné jako pole `TTL` v sadě. Existuje kvůli tomu, že ji cache klient může změnit. Pole `Signature-Expiration` určuje datum a čas vypršení živosti záznamu. Pokud je větší než aktuální čas, tak záznam nesmí být pro ověřování použit. Pole `Signature-Inception` určuje datum a čas zahájení živosti záznamu. Pokud je menší než aktuální čas, tak záznam nesmí být pro ověřování použit. Pole `Key-Tag` určuje značku veřejného klíče, přesněji záznamu typu `DNSKEY`; a vytváří se podle odstavce (3.8.1) o značkovacím algoritmu. Pole `Signers-Name` určuje vlastníka veřejného klíče, přesněji pole `NAME` v záznamu typu `DNSKEY`; a musí být stejné jako pole `NAME` v záznamu `SOA`; a musí být v kanonické formě podle odstavce (3.3.1) o kanonické formě záznamu. Pole `Signature` určuje podpis sady; jeho formát závisí na poli `Algorithm`; a vytváří se podle odstavce (3.8.5) o podpisovém algoritmu.

Navzdory odlišným principům systému DNS musí i tento záznam sekundovat záznamům typu `CNAME`. Kombinace polí `Signers-Name`, `Algorithm` a `Key-Tag` identifikuje záznam typu `DNSKEY`. Kombinace polí `NAME`, `CLASS` a `Type-Covered` identifikuje sadu.

3.7.3 Záznam typu `DNSKEY`

Záznam typu `DNSKEY`[12] je určen pro ověření pravosti záznamu. Není určen k uložení libovolného veřejného klíče či certifikátu bez vztahu k rozšíření DNSSEC. Záznam je znázorněn na obrázku (3.9).

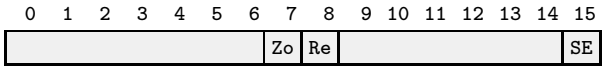


Obrázek 3.9: Záznam typu `DNSKEY` [3, str. 4]

Pole `CLASS` je vůči poli `TYPE` nezávislé. Na pole `TTL` nejsou kladeny speciální požadavky. Pole `Flags` se skládá z příznaků; přičemž doposud podporované příznaky jsou znázorněny na obrázku (3.10). Pole `Protocol` určuje typ protokolu; nabývá pouze hodnoty `DNSSEC/3`; a musí být nastaven jako `DNSSEC` kvůli zpětné kompatibilitě se záznamem typu `KEY`. Pole

[12] DNSKEY – DNS Key

`Algorithm` určuje typ podpisového algoritmu a formát pole `Public-Key`; vytváří se podle odstavce (3.5.4) o poli typu `Algorithm`; a nabývá doposud podporovaných hodnot podle výše uvedené tabulky (3.2). Pole `Public-Key` určuje veřejný klíč; jeho formát závisí na poli `Algorithm`; a vytváří se podle odstavce (3.5.2) o poli typu `Public-Key`.



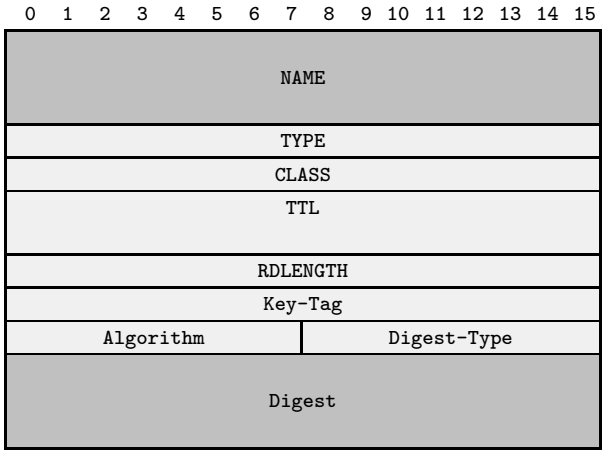
Obrázek 3.10: Příznaky v záznamu typu `DNSKEY` [16]

Příznak `Zone-Key` určuje zónový klíč; nabývá hodnoty `ON/1` pro zónový klíč, přičemž pole `NAME` musí být totožné s polem `NAME` v záznamu typu `SOA`; nebo `OFF/0` pro veřejný klíč jiného druhu, přičemž nesmí být pro ověřování použit. Příznak `Revoke-Key` určuje odvolaný klíč; a nabývá hodnoty `OFF/0` pro neodvolaný klíč nebo `ON/1` pro odvolaný klíč. Příznak `SEP`[13] určuje vstupní bezpečnostní bod; a nabývá hodnoty `OFF/0` pro ZSK nebo `ON/1` pro KSK. Slouží pouze jako nápověda pro podepisování či ladění.

Pokud je příznak `SEP` nastaven na hodnotu `ON` a příznak `Zone-Key` na hodnotu `OFF`, tak záznam nesmí být pro ověřování použit.

3.7.4 Záznam typu `DS`

Záznam typu `DS`[14] je určen pro ověření pravosti záznamu typu `DNSKEY` v podřízené zóně. Spolu se záznamem typu `DNSKEY` mají sice stejné pole `NAME`, ale nacházejí se v různých zónách. Záznam je znázorněn na obrázku (3.11).



Obrázek 3.11: Záznam typu `DS` [3, str. 16]

Pole `CLASS` je vůči poli `TYPE` nezávislé. Na pole `TTL` nejsou kladeny speciální požadavky. Pole `Key-Tag` určuje značku veřejného klíče v podřízené zóně, přesněji záznamu typu `DNSKEY` v podřízené zóně; a vytváří se podle odstavce (3.8.1) o značkovacím algoritmu. Pole `Algorithm` určuje typ podpisového algoritmu veřejného klíče v podřízené zóně, přesněji pole `Algorithm` v záznamu typu `DNSKEY` v podřízené zóně; vytváří se podle odstavce (3.5.4) o poli typu `Algorithm`; a nabývá doposud podporovaných hodnot podle výše uvedené tabulky (3.2). Pole `Digest-Type` určuje typ otiskového algoritmu; vytváří se podle odstavce (3.5.3) o poli typu `Digest-Type`; a nabývá doposud podporovaných hodnot podle výše uvedené tabulky (3.1). Pole `Digest` určuje otisk veřejného klíče v podřízené zóně,

[13] SEP – Secure Entry Point
[14] DS – Delegation Signer

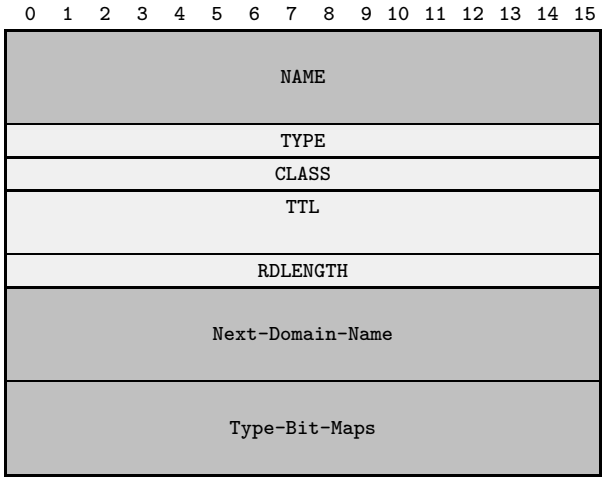
přesněji záznamu DNSKEY v podřízené zóně; a vytváří se podle odstavce (3.8.2) o otiskovém algoritmu.

Pokud je příznak Zone-Key v záznamu typu DNSKEY nastaven na hodnotu OFF, tak záznam typu DS nesmí být pro ověřování použit.

Kombinace polí NAME, Algorithm a Key-Tag identifikuje záznam typu DNSKEY.

3.7.5 Záznam typu NSEC

Záznam typu NSEC[15] je určen pro ověření absence záznamu. Ukazuje na následujícího vlastníka a seznam typů záznamů stejného vlastníka. Navzdory odlišným principům systému DNS musí i tento záznam sekundovat záznamům typu CNAME. Záznam je znázorněn na obrázku (3.12).



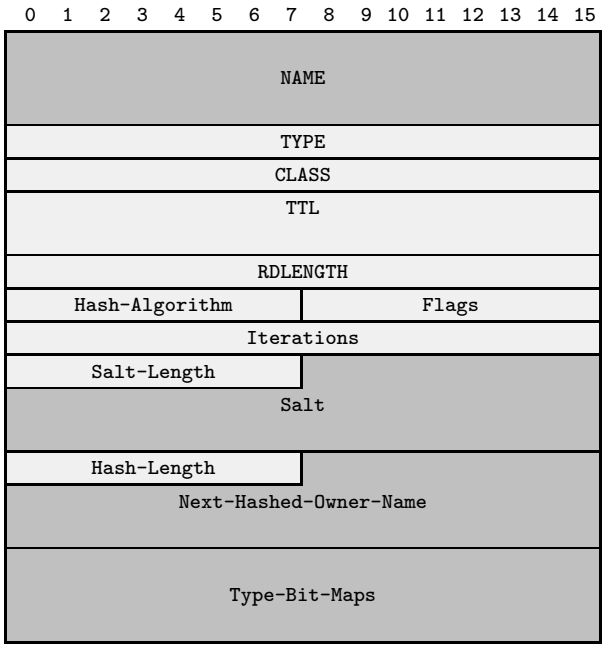
Obrázek 3.12: Záznam typu NSEC [3, str. 13]

Pole CLASS je vůči poli TYPE nezávislé. Pole TTL by mělo být stejné jako pole Minimum v záznamu typu SOA. Pole Next-Domain-Name určuje následujícího vlastníka, přesněji pole NAME; a musí být v kanonické formě podle odstavce (3.3.1) o kanonické formě záznamu a zóna musí být kanonicky seřazena podle odstavce (3.3.2) o kanonickém pořadí záznamů v zóně. Na konci zóny musí být stejné jako pole NAME v záznamu typu SOA. Nesmí být stejné jako pole NAME v sadě, která není pro zónu autoritativní. Pole Type-Bit-Maps určuje seznam typů záznamů vlastníka tohoto záznamu; a vytváří se podle odstavce (3.5.5) o poli typu Type-Bit-Maps.

3.7.6 Záznam typu NSEC3

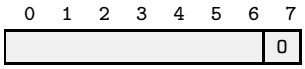
Záznam typu NSEC3 je určen pro ověření absence záznamu s ochranou proti vyčtení zóny. Záznam je znázorněn na obrázku (3.13).

[15] NSEC – Next Secure



Obrázek 3.13: Záznam typu `NSEC3`

Pole `Hash-Algorithm` určuje typ iterovaného otiskového algoritmu; vytváří se podle odstavce (3.5.6) o poli typu `Hash-Algorithm`; a nabývá doposud podporovaných hodnot podle výše uvedené tabulky (3.3). Pole `Flags` se skládá z příznaků; přičemž doposud podporované příznaky jsou znázorněny na obrázku (3.14). Pole `Iterations` určuje počet iterací otiskového algoritmu. Pole `Salt-Length` určuje délku pole `Salt`. Pole `Salt` určuje sůl. Pole `Hash-Length` určuje délku pole `Next-Hashed-Owner-Name`. Pole `Next-Hashed-Owner-Name` určuje iterovaný otisk následujícího vlastníka; a vytváří se podle odstavce (3.8.11) o iterovasném otiskovém algoritmu. Pole `Type-Bit-Maps` určuje seznam typů záznamů vlastníka tohoto záznamu; a vytváří se podle odstavce (3.5.5) o poli typu `Type-Bit-Maps`.



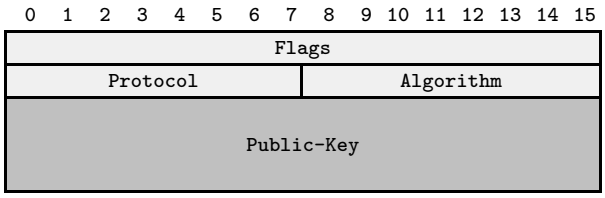
Obrázek 3.14: Příznaky záznamu v typu `NSEC3` [17]

3.8 Algoritmy

V rozšíření DNSSEC vznikla potřeba transformovat data na data jiná, která se dějí pomocí algoritmů. Rozšíření DNSSEC bylo navrženo tak, aby bylo nezávislé vůči těmto algoritmům [3, str. 24]. Implementace musejí implementovat všechny povinné algoritmy [3, str. 24]. Algoritmy jsou pro přehlednost pojmenovány jako **značkovací**, **otiskové**, **podpisové** a **iterované otiskové**.

3.8.1 Značkovací algoritmus

Je užíván záznamy typu `RRSIG` a `DS`. Pro podepisovací algoritmus typu `RSAMD5` je vstup definován jako modulus veřejného klíče [3, str. 27]. Pro podepisovací algoritmy ostatních typů je vstup znázorněn na obrázku (3.15).



Obrázek 3.15: Vstup pro značkovací algoritmus [3, str. 26]

Všechna pole pocházejí ze záznamu typu `DNSKEY`. Výstupem je značka veřejného klíče, která přísluší poli typu `Key-Tag`. Pro podepisovací algoritmus typu `RSAMD5` je značkovací algoritmus definován jako bitový součet odzadu čtvrtého a odzadu třetího bajtu vstupu [3, str. 27]. Pro podepisovací algoritmy ostatních typů je značkovací algoritmus podobný, ale nikoli identický, jako mnohé příbuzné součtové algoritmy používané v jiných internetových protokolech [3, str. 26]. Specifikace pro jistotu nabízí referenční implementaci značkovacího algoritmu v jazyce **ANSI**[16] C [3, str. 26]:

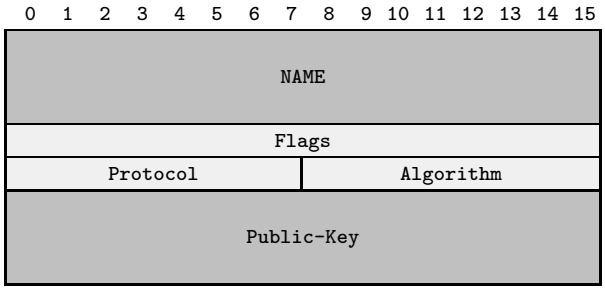
```
unsigned int keytag ( unsigned char key[], unsigned int keysize )
{
  unsigned long ac ;
  int i ;

  for ( ac = 0, i = 0 ; i < keysize ; ++ i )
    ac += ( i & 1 ) ? key[i] : key[i] << 8 ;
  ac += ( ac >> 16 ) & 0xFFFF ;

  return ac & 0xFFFF ;
}
```

3.8.2 Otiskový algoritmus

Je užíván záznamem typu `DS`. Vstup je společný pro podpisové algoritmy všech typů a je znázorněn na obrázku (3.16).



Obrázek 3.16: Vstup pro otiskový algoritmus [3, str. 17]

Všechna pole pocházejí ze záznamu typu `DNSKEY`. Výstupem je otisk veřejného klíče, který přísluší poli typu `Digest`. Typ otiskového algoritmu nese pole typu `Digest-Type` podle odstavce (3.5.3). Seznam doposud podporovaných otiskových algoritmů se nachází ve výše uvedené tabulce (3.1). Pro bližší popis byly vybrány otiskové algoritmy povinných typů `SHA1` a `SHA256`.

[16] ANSI – American National Standards Institute

3.8.3 Otiskový algoritmus typu `SHA1`

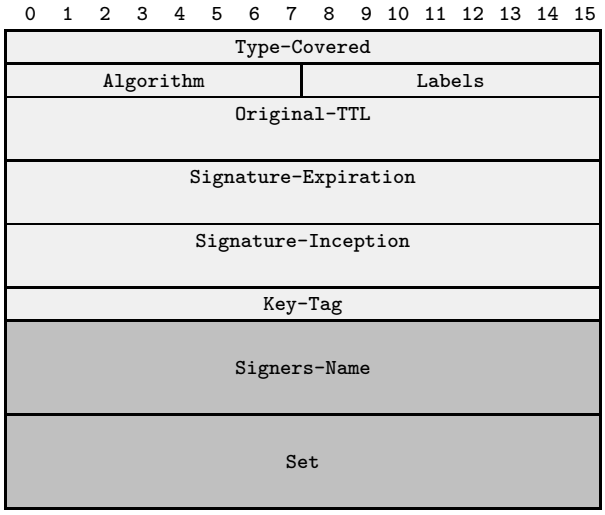
Otiskový algoritmus je definován jako algoritmus SHA-1 z externí specifikace **FIPS PUB 180-1** [43]. Velikost otisku je vždy 160 bitů.

3.8.4 Otiskový algoritmus typu `SHA256`

Otiskový algoritmus je definován jako algoritmus SHA-256 z externí specifikace **FIPS PUB 180-3** [44]. Velikost otisku je vždy 256 bitů.

3.8.5 Podpisový algoritmus

Je užíván záznamem typu `RRSIG`. Vstup je společný pro podpisové algoritmy všech typů a je znázorněn na obrázku (3.17).



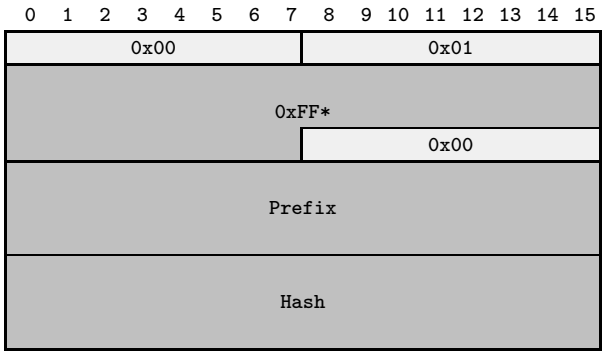
Obrázek 3.17: Vstup pro podpisový algoritmus [3, str. 9]

Pole `Type-Covered` až `Signers-Name` pocházejí ze záznamu typu `RRSIG`. Pole `Set` určuje sadu k podepsání s jejím kanonickým seřazením podle odstavce (3.3.3) o kanonickém pořadí záznamů v sadě.

Výstupem je podpis, který přísluší poli typu `Signature`. Typ podpisového algoritmu nese pole typu `Algorithm` podle odstavce (3.5.4). Seznam doposud podporovaných podpisových algoritmů se nachází ve výše uvedené tabulce (3.2). Pro bližší popis byl vybrán podpisový algoritmus rodiny `RSA`.

3.8.6 Podpisový algoritmus rodiny `RSA`

Podpisovému algoritmu rodiny `RSA` ještě předchází otiskový algoritmus. Výše uvedený vstup (3.17) se stává vstupem pro otiskový algoritmus. Výstupem je otisk. Vstup pro podpisový algoritmus je znázorněn na obrázku (3.18).



Obrázek 3.18: Vstup pro podpisový algoritmus rodiny RSA

Pole `0xFF*` určuje zarovnání vstupu. Pole `Prefix` určuje prefix otiskového algoritmu. Pole `Hash` určuje již vytvořený otisk.

Pole `0xFF*` se skládá z hodnot `0xFF`. Velikost pole `0xFF*` musí být taková, aby velikost vstupu byla totožná s velikostí modulu.

Podpisový algoritmus je dvou podob, a to podepisování a ověřování. Podepisování je definováno jako [12, 14, 23, str. 2, 3, 4]:

$$s = m^e \mod n \tag{3.1}$$

Kde s je podpis, m je vstup, e je soukromý exponent a n je modulus. Soukromý exponent a modulus dávají dohromady soukromý klíč. Ověřování je analogicky definováno jako:

$$m = s^d \mod n \tag{3.2}$$

Kde s je podpis, m je vstup, d je veřejný exponent a n je modulus. Veřejný exponent a veřejný dávají dohromady veřejný klíč.

Pro bližší popis byly vybrány podpisové algoritmy doporučených typů `RSA256` a `RSA512`, povinných typů `RSASHA1` a `RSASHA1-NSEC3-SHA1` a i nedoporučeného typu `RSAMD5` pro úplnost podpisových algoritmů rodiny `RSA`. Ačkoli název typu podpisového algoritmu většinou nese název typu otiskového algoritmu, tak to není samozřejmostí. Proto jsou uvedeny i typy otiskových algoritmů.

3.8.7 Podpisový algoritmus typu `RSAMD5`

Otiskový algoritmus je definován jako algoritmus MD5 ze specifikace **RFC 1321** [33]. Velikost otisku je vždy 128 bitů. Pro tento podpisový algoritmus je prefix definován jako:

`0x3020300c06082a864886f70d020505000410`.

Jelikož je velikost polí `0x00`, `0x01`, `0x00`, `Prefix` a `Hash` celkem 296 bitů, tak lze použít modulus o velikosti minimálně 296 bitů. V úvahu ale připadají velikosti modulu o 512, 1024, 2048 a 4096 bitech.

3.8.8 Podpisový algoritmus typu `RSASHA1` a `RSASHA1-NSEC3-SHA1`

Podpisový algoritmus typu `RSASHA1-NSEC3-SHA1` je aliasem pro podpisový algoritmus typu `RSASHA1`. Otiskový algoritmus je definován jako algoritmus SHA-1 ze specifikace FIPS PUB 180-1 [43]. Velikost otisku je vždy 160 bitů. Pro tento podpisový algoritmus je prefix definován jako:

0x3021300906052b0e03021a05000414.

Jelikož je velikost polí 0x00, 0x01, 0x00, Prefix a Hash celkem 304 bitů, tak lze použít modulus o velikosti minimálně 304 bitů. V úvahu ale připadají velikosti modulu o 512, 1024, 2048 a 4096 bitech.

3.8.9 Podpisový algoritmus typu RSASHA256

Otiskový algoritmus je definován jako algoritmus SHA-256 z externí specifikace FIPS PUB 180-3 [44]. Velikost otisku je vždy 256 bitů. Pro tento podpisový algoritmus je prefix definován jako:

0x3031300d060960864801650304020105000420.

Jelikož je velikost polí 0x00, 0x01, 0x00, Prefix a Hash celkem 432 bitů, tak lze použít modulus o velikosti minimálně 432 bitů. V úvahu ale připadají velikosti modulu o 512, 1024, 2048 a 4096 bitech.

3.8.10 Podpisový algoritmus typu RSASHA512

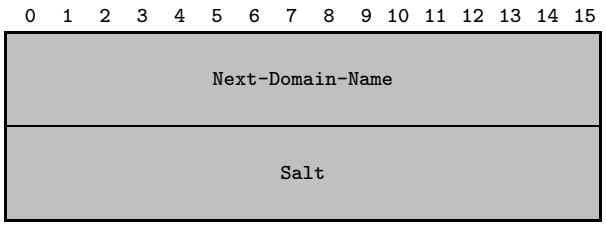
Otiskový algoritmus je definován jako algoritmus SHA-512 z externí specifikace FIPS PUB 180-3 [44]. Velikost otisku je vždy 512 bitů. Pro tento podpisový algoritmus je prefix definován jako:

0x3051300d060960864801650304020305000440.

Jelikož je velikost polí 0x00, 0x01, 0x00, Prefix a Hash celkem 688 bitů, tak lze použít modulus o velikosti minimálně 688 bitů. V úvahu ale připadají velikosti veřejného klíče o 1024, 2048 a 4096 bitech. Velikost modulu o 512 bitech tedy není možné použít.

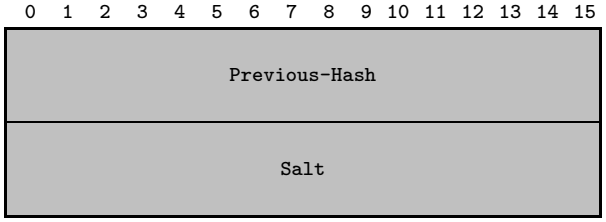
3.8.11 Iterovaný otiskový algoritmus

Je užíván záznamem typu NSEC3. Počáteční vstup pro iterovaný otiskový algoritmus je znázorněn na obrázku (3.19).



Obrázek 3.19: Počáteční vstup pro iterovaný otiskový algoritmus

Pole Next-Domain-Name určuje následujícího vlastníka. Pole Salt určuje sůl. Další vstup pro iterovaný otiskový algoritmus je znázorněn na obrázku (3.20).



Obrázek 3.20: Další vstup pro iterovaný otiskový algoritmus

Pole `Previous-Hash` určuje iterovaný otisk z předchozí iterace.

Výstupem je iterovaný otisk, který přísluší poli typu `Next-Hashed-Owner-Name`. Typ iterovaného otiskového algoritmu nese pole typu `Hash-Algorithm` podle odstavce (3.5.6). Seznam doposud podporovaných iterovaných otiskových algoritmů se nachází ve výše uvedené tabulce (3.3). Iterovaný otiskový algoritmus je definován jako [26, str. 14]:

$$I(m, s, 0) = H(m \mid s) \tag{3.3}$$

$$I(m, s, k) = H(I(m, s, k-1) \mid s), \quad \text{když } k > 0 \tag{3.4}$$

Kde I je iterovaný otiskový algoritmus, H je otiskový algoritmus, m je vstup, s je sůl, k je počet iterací a $\mid$ je konkatenace.

3.9 Standardizace

První zmínka o zabezpečení systému doménových jmen se objevila v roce 1997, ve kterém dokument **RFC 2065** [13] definuje tři nové typy záznamů – `KEY`, `SIG` a `NXT`. Samotná existence rozšíření DNSSEC sahá až do roku 1999, ve kterém dokument **RFC 2535** [11] popisuje první specifikaci rozšíření DNSSEC. Mezitím dokument **RFC 2671** [45] popisuje rozšíření EDNS a definuje jeden nový typ záznamu – `OPT` a dokument **RFC 3225** [7] pro tento typ záznamu definuje jeden nový rozšířený příznak – `DO`.

V roce 2001 je ovšem dokončena studie, která první specifikaci rozšíření DNSSEC označila za nepoužitelnou. A tak se začalo pracovat na specifikaci nové, pracovně označené jako **DNSSECbis**. Po šesti letech v roce 2005 vyšla druhá specifikace rozšíření DNSSEC rozprostřená hned do třech dokumentů. Dokument **RFC 4033** [2] popisuje jeho principy; dokument **RFC 4034** [3] definuje čtyři nové typy záznamů – `DNSKEY`, `SIG`, `DS` a `NSEC` a uspořádání záznamů v zónovém souboru; a dokument **RFC 4035** [4] popisuje změny v protokolu DNS a definuje dva nové příznaky – `AD` a `CD`. Nespokojenost se záznamen typu `NSEC` vyústila v dokument **RFC 5155** [26], který přichází s novým záznamem typu `NSEC3`.

4 Implementační technologie

*I*mplementace je vystavěna hned na několika technologiích, které důkladně plní svoje poslání a vzájemně spolu spolupracují. Samozřejmostí je protokol HTTP(S) pro přenos dat. Zapomínat by se nemělo ani na jazyky (X)HTML a CSS pro popis struktury a vzhledu dokumentů. Převážný podíl zaujímá jazyk a interpret PHP pro popis univerzálních algoritmů na straně serveru. Nezbytností je jazyk SQL pro popis akcí s relační databází MySQL. Intuitivnost pak doplňují jazyk JavaScript a technika AJAX pro popis univerzálních algoritmů na straně klienta. Velkou důležitost pak představuje hašovací algoritmus, asymetrická kryptografie a knihovna OpenSSL pro její podporu.

4.1 Transportní protokol

HTTP[1] je protokol pro přenos dat. Hnízdí v aplikační vrstvě modelu OSI a v transportní vrstvě ho zapouzdřuje protokol TCP, který mu přiděluje port 80. Ačkoli měl původně sloužit pro přenos dokumentů (X)HTML, tak nyní slouží pro přenos dat libovolného typu s pomocí rozšíření **MIME**[2]. Komunikace počítá s modelem klient – server. Protokol je velice jednoduchý, ve kterém klient naváže spojení se serverem, klient odešle požadavek, následně server odešle odpověď a spojení s klientem ukončí. Klient ani server si spojení po jeho ukončení nepamatují, tudíž je protokol „bezstavový.“ Verze protokolu na třetím řádku zprávy je vítaným prostředkem pro zpětnou kompatibilitu, který v jiných protokolech mnohdy bohužel chybí. Adresování v protokolu HTTP je řešeno pomocí **URI**[3].

HTTPS[4] je zabezpečená varianta protokolu HTTP. V transportní vrstvě ho zapouzdřuje protokol TCP, který mu přiděluje port 433.

4.1.1 Historie

Verze **HTTP 0.9** z roku 1991 přidává metodu `GET`. Verze **HTTP 1.0** z roku 1996 přidává hlavičky, metody `HEAD` a `POST` a MIME [36]. Verze **HTTP 1.1** z roku 1999 přidává udržované spojení „keep-alive“ nesoucí více dotazů, přičemž je stále bezstavový; dále metody `OPTIONS`, `PUT`, `DELETE`, `TRACE` a `CONNECT`; a je verzí současnou [32, 31].

[1] HTTP – Hypertext Transfer Protocol
[2] MIME – Multipurpose Internet Mail Extensions
[3] URI – Uniform Resource Identifier
[4] HTTPS – Hypertext Transfer Protocol Secure

4.2 Zobrazovací jazyk

HTML[5] je značkovací jazyk pro popis struktury dokumentů. Vychází z jazyka **SGML**[6]. Pro kód v jazyce HTML se používá pojem „dokument.“ Uplatňuje se zejména na internetu.

4.2.1 Historie

Jazyk vyvinul Tim Berners-Lee v roce 1990 pro potřeby informačního systému pro CERN. Jazyk by implementován webovým prohlížečem **WorldWideWeb**. Později byl implementován webovým prohlížečem **Mosaic** s grafickým rozhraním, který vyvinuli Marc Andreessen a Eric Bin v roce 1993.

Verze **HTML 0.9** z roku 1991 byla první verzí, která ještě nepodporovala grafický režim. Verze **HTML 2** z roku 1994 je již kompatibilní s jazykem SGML; přidává formuláře a grafické elementy; a byla přijata jako standard **IETF**[7] [35]. Následující verze byly přijaty jako standard **W3C**[8]. Verze **HTML 3.2** z roku 1996 přidává tabulky, zarovnávání textu a jazyk CSS [8]. Verze **HTML 4** z roku 1997 rozšiřuje tabulky a formuláře; a přidává rámce [9]. Verze **HTML 4.01** z roku 1999 odstraňuje nedostatky předchozích verzí; a měla být poslední, jelikož nástupcem měl být jazyk XHTML [10]. A verze **HTML 5** z roku 2007 je verzí současnou.

4.2.2 XHTML

XHTML[9] je rozšířená varianta jazyka HTML. Syntaxe je inspirována jazykem **XML**[10], z čehož vyplývá několik změn [25]. Tagy se nesmějí překrývat. Název tagu a název atributu musí být tvořen malými písmeny. U nepárového elementu (`meta`, `link`, `br`, `hr`, `img` a další) musí být přítomen koncový tag. Hodnota atributu, byť jen číslo, musí být uzavřena do uvozovek. U osamoceného atributu (`compact`, `checked` a další) musí být přítomna hodnota atributu. U prázdného elementu musí být přítomen koncový tag nebo počáteční tag ukončený `/>`. Atribut `name` není schválen a bude odstraněn v následujích verzích. U výčtového atributu (`type` a další) musí být hodnota atributu tvořena malými písmeny. Pro dokument s rámy je určena deklarace `XTHML 1.0 Frameset` a pro ostatní `XHTML 1.0 Transitional` nebo `XHTML 1.0 Strict`. Jazyk je aplikován v odstavci (7.9) o grafickém rozhraní pro výstup.

4.2.3 CSS

CSS[11] je definiční jazyk pro popis vzhledu dokumentu v jazyce (X)HTML či XML. Jeho účelem je oddělení obsahu dokumentu a jeho vzhledu. Pro kód v jazyce CSS se používá pojem „styl.“ Závádí vlastnosti alternativní k některým atributům jazyka (X)HTML a řadu vlastností úplně nových. Styl může být interní v tomtéž dokumentu nebo externí v samostatném souboru. Takový přístup přináší hlavně výhody. Externí styl může být společný pro více dokumentů. Externí styl může být načten do vyrovnávací paměti. Externích stylů může

[5] HTML – Hypertext Markup Language
[6] SGML – Standard Generalized Markup Language
[7] IETF – Internet Engineering Task Force
[8] W3C – World Wide Web Consortium
[9] XHTML – Extensible Hypertext Markup Language
[10] XML – Extensible Markup Language
[11] CSS – Cascading Style Sheets

být více a mohou se přepínat v závislosti na výrobci prohlížeče apod. Jazyk je aplikován v odstavci (7.9) o grafickém rozhraní pro výstup.

4.2.4 Historie

Všechny následující úrovně byly přijaty jako standard W3C. Úroveň **CSS 1** z roku 1996 přidává rodinu písma, zdůraznění písma podobné kurzívě; barvu písma, pozadí a jiných elementů; mezeru mezi slovy, písmeny a řádky; zarovnání textu, obrázků, tabulek a jiných elementů; vnější a vnitřní okraje, ohraničení; a poziciování [15]. Úroveň **CSS 2** z roku 1998 přidává absolutní, relativní, pevné a prostorové poziciování; multimédia; směr textu; a stín písma [5]. Úroveň **CSS 2.1** odstraňuje nedostatky předchozí úrovně [6]. Úroveň **CSS 3** počala roku 1999 a skládá se z desítek modulů.

4.2.5 Webový prohlížeč

Jazyky (X)HTML a CSS jsou interpretovány webovými prohlížeči, které dokumenty a styly převádějí na grafickou reprezentaci. Mezi významné webové prohlížeče se řadí Microsoft Internet Explorer, Opera, Mozilla Firefox, Apple Safari, Google Chrome a další.

Webové prohlížeče jsou vůči standardu nekonzistentní a mezi sebou nekompatibilní. Zásadní chybu předvedl webový prohlížeč Microsoft Internet Explorer, který vlastnost `width` interpretuje tím způsobem, že do ní započítává i vlastnosti `border` a `padding`. Dokument pak může v několika různých prohlížečích nabýt úplně jiného a nežádoucího vzhledu. To je důvodem proč webové prohlížeče podporují dva režimy, standardní režim implementující standard a „quirk“ režim implementující zpětnou kompatibilitu. Dále zavádějí podmíněné vlastnosti či komentáře viditelné pro cílené webové prohlížeče.

4.3 Serverový jazyk

PHP[12] je programovací imperativní jazyk pro popis univerzálních algoritmů, který má v sobě velký potenciál ve vytváření výstupu. Pro kód v jazyce PHP se používá pojem „skript.“ Syntaxe jazyka je inspirována výběrem těch nejlepších jazykových konstrukcí z jazyků Perl, C, C++, Pascal a Java. Datový typ proměnné je stanoven implicitně po přiřazení hodnoty do proměnné. Indexy, klíče a prvky v poli mohou být různého datového typu. Podporuje reference, které se hodí zejména ve výstupních parametrech funkce. Objektově orientované programování je podporováno již od verze 3, ale až od verze 5 na přijatelné úrovni. O přehledné dokumentaci není pochyb. Díky těmto pohodlným vlastnostem se jazyk PHP stal poměrně mocným a oblíbeným nástrojem. Jazyk je aplikován v kapitolách (6) o knihovně a (7) o aplikaci.

4.3.1 Interpret

Je implementován stejnojmenným interpretem, který je jediný. Skládá z několika desítek modulů. Spolu se všemi moduly nabízí nepřeberné množství funkcí pro práci s textem, poli, obrázky, soubory, dokumenty **PDF**[13], dokumenty XML, kompresí, přenosy, databázemi a dalšími. Podporuje komunikaci s databázovými servery MySQL, Oracle, PostgreSQL a Microsoft SQL.

[12] PHP – PHP: Hypertext Preprocessor, dříve Personal Home Page
[13] PDF – Portable Document Format

4.3.2 Historie

U úspěšných technologií je zajímavé dozvědět se jak vznikly [38]. Vše začalo skriptem v jazyce Perl, který sledoval návštěvnost a provoz webových stránek. Napsal ho Rasmus Lerdorf v roce 1994. Tento skript přepsal do jazyka C a šířil jako „Personal Home Page Tools.“ Nástroj si získal celou komunitu lidí, kteří pak přispěli k jeho vývoji. Z nich se nástroje ujali Zeev Suraski a Andi Gutmans a přejmenovali ho na „PHP: Hypertext Preprocesor“ verze 3 v roce 1997.

Verze **PHP 1** z roku 1995 se jmenovala „Personal Home Page Tools.“ Verze **PHP 2** z roku 1997 zrychluje interpretaci. Verze **PHP 3** z roku 1998 vylepšuje syntaxi; a přidává moduly, „cookies“ a objektově orientované programování. Verze **PHP 4** z let 2000 až 2008 představuje jádro **Zend Engine**; rozšiřuje podporu pro web servery; přidává HTTP(S) relace, superglobální proměnné; a odstraňuje přímý přístup ke globálním proměnným. A verze **PHP 5** z let 2004 až 2011 představuje jádro **Zend Engine II**; rozšiřuje objektově orientované programování; a přidává jmenný prostor a uvolňování paměti; a je verzí současnou.

4.3.3 Aplikace

Mohou v něm být napsány konzolové i webové aplikace. Konzolová aplikace se může s využitím grafických knihoven stát běžnou desktopovou aplikací.

Webové aplikace jsou častější už z toho důvodu, že pro ty byl původně určen. Výstupem mohou být data libovolného typu stejně jako tomu je u HTTP(S), ale nejčastěji je to právě dokument (X)HTML. Kód v jazyce (X)HTML lze prolínat kódem v jazyce PHP, jelikož je kód v jazyce PHP vymezen vlastním elementem v jazyce (X)HTML. Tato dovednost z něj činí jazykem pro dynamické webové stránky.

4.3.4 Webová aplikace

Nutno poznamenat, že webová aplikace má specifické chování, které se liší od běžných konzolových aplikací. Vstup a výstup je přenášen aplikačním protokolem HTTP(S). Taková aplikace pak skýtá některé přednosti, ale i omezení. Aplikace se může vystavit na internetu a být tak dostupná „online“ všude tam, kde je internet a webový prohlížeč. Aplikace běží jak na serveru, tak svým způsobem i u klienta. Protokol HTTP(S) svou bezstavovostí k výhodám zrovna moc nepřispívá. Jedinou možností jak předat data aplikaci je formou dotazu; a jediným okamžikem pro předání dotazu je při spouštění aplikace. Pak již s aplikací nelze komunikovat a aplikace vůbec neinteraguje s uživatelem. Jedinou možností jak předat data klientovi je formou odpovědi; a jediným okamžikem pro předání odpovědi je při skončení aplikace. Výstup se tedy nezobrazuje průběžně. Z tohoto důvodu musí být aplikace konečného trvání, tudíž nemůže sestávat z nekonečné smyčky čekající na události. Aby aplikace zakryla bezstavovost protokolu HTTP(S), využívá soubory „cookies“ nebo relace. Aby aplikace působila dojmem nepřetržitého provozu, musí si udržovat kontext tím, že bude vhodně generovat posloupnost odkazů. Pro zdokonalení této iluze se prolíná s jazykem **JavaSript** a technikou **AJAX**[14].

[14] AJAX – Asynchronous JavaScript and XML

4.4 Dotazovací jazyk

SQL[15] je deklarativní jazyk pro popis akcí s relační databází. Pro kód v jazyce SQL se používá pojem „dotaz.“ Manipuluje s řádky tabulek tím, že je přidává, aktualizuje či odstraňuje. Jelikož jsou schémata tabulek nebo přístupová práva reprezentována též jako tabulky, tak manipuluje i se schématy a přístupovými právy. Syntaxe jazyka je inspirována přirozeným jazykem – angličtinou. Pro modifikovaný jazyk SQL se užívá pojem, který má velice příhodné jméno – „dialekt.“ Jazyk je aplikován v odstavci (7.4) o vyrovnávací paměti.

4.4.1 Historie

Vše začalo jazykem **SEQUEL**[16], který vyvinuli Donald D. Chamberlin a Raymond F. Boyce ze společnosti IBM v letech sedmdesátých. Byl implementován kvazi-relačním databázovým systémem „System R“ ve stejné společnosti ve stejných letech. Jazyk SEQUEL byl později přejmenován na SQL z důvodu kolize s ochrannou známkou jedné letecké společnosti. Souběžným jazykem byl **QUEL**, který byl implementován databázovým systémem Ingres na univerzitě v Berkeley v roce 1974. Jazyk QUEL byl později přesunut též do SQL. Jazyk SQL byl implementován databázovým systémem **Oracle V2** ve společnosti Relational Software, Inc. v roce 1979.

Verze **SQL-86/SQL-87** byla přijata jako standard **ANSI**[17] a následně jako standard **ISO**[18]. Verze **SQL-98/FIPS 127-1** a **SQL-92/SQL2/FIPS 127-2** odstraňují nedostatky předchozích verzí [42]. Verze **SQL:1999/SQL3** přidává regulární výrazy, rekurzivní dotazy, triggery, procedury, neskalární typy a objektově orientovaný přístup. Verze **SQL:2003** představuje možnosti jazyka XML; a přidává sloupce s automaticky generovanými hodnotami. Verze **SQL:2006** přidává podporu jazyků XML a **XQuery**. Verze **SQL:2008** povoluje klausuli `ORDER BY` vně definici kurzorů; přidává trigger `INSTEAD OF` a příkaz `TRUNCATE`.

4.4.2 Databázový systém

Jazyk SQL je interpretován databázovými systémy, které provádějí požadované chování. Mezi významné databázové systémy se řadí MySQL, Oracle, PostgreSQL, Microsoft SQL a další.

Databázové systémy jsou vůči standardu nekonzistentní a mezi sebou nekompatibilní. Zvláště v syntaxi orientované na datum či čas, spojování řetězců, `NULL` konstantě a v porovnávání citlivém na velikost písmen. Například databázový systém Oracle klíčové slovo `DATE` interpretuje jako `DATETIME` a postrádá `TIME`. Standard je natolik rozsáhlý, že databázové systémy ho neimplementují celý. Nepodporují některé standardní jazykové konstrukce nebo přidávají vlastní jazykové konstrukce. Standard popisuje sémantiku natolik nepřesně, že si ji databázové systémy vyloží po svém. Syntaxi standard specifikuje již precizně. Na druhou stranu mohou být databázové systémy mezi sebou nekompatibilní záměrně, čímž si podněcují věrnost zákazníků.

[15] SQL – Structured Query Language
[16] SEQUEL - Structured English Query Language
[17] ANSI – American National Standards Institute
[18] ISO – International Organization for Standardization

4.4.3 MySQL

MySQL je implementace databázového systému a vlastního dialektu jazyka SQL. Je založen na relačním modelu. Podporuje uživatelské účty a pro ně spravuje řízení přístupu. Často je nasazován k webovým aplikacím napsaných v jazyce PHP.

4.5 Interaktivní jazyk

JavaScript je programovací imperativní jazyk pro popis univerzálních algoritmů. Pro kód v jazyce JavaScript se používá pojem „skript.“ Syntaxe je inspirována jazyky C, C++ a Java. Jeho prefix s jazykem **Java** spojuje pouze podobnost v syntaxi. Jazyk je aplikován v odstavci (7.9.1) o interaktivitě v grafickém rozhraní pro výstup.

Mohou v něm být napsány skriptovací i webové aplikace. Webové aplikace jsou častější už z toho důvodu, že pro ty byl původně určen. Kód v jazyce (X)HTML lze prolínat kódem v jazyce JavaScript, jelikož je kód v jazyce JavaScript vymezen vlastním elementem v jazyce (X)HTML.

Jelikož má interpret přímý přístup k elementům (X)HTML dokumentu, může se tento (X)HTML dokument stát snadno interaktivním. Interpret nemá přístup vně webového prohlížeče z důvodů bezpečnostních.

4.5.1 Historie

Vyvinul ho Brendan Eich ze společnosti Netscape. Jazyk byl přijat jak standard **ECMA**[19] roku 1997 a jako standard ISO roku 1998. Dostal spoustu dalších jmen, jako je **ECMAScript** či **ActionScript**. Ve společnosti Netscape byl pojmenován jako **Mocha**, poté **LiveScript** a později jako JavaScript. Společnosti Netscape a Sun Microsystems ho připojili k jazykům HTML a Java roku 1995. Společnost Microsoft ho pojmenovala jako **JScript**.

4.6 Asynchronní načítání

AJAX[20] je technika pro načtení a zobrazení již natečných a zobrazených webových aplikací na pozadí. Technika je aplikována v odstavci (7.7.1) o načítání v grafickém rozhraní pro vstup.

Tato technika vyzdvihuje některé přednosti. Nenačítá se celý dokument, ale pouze obsah elementu, který má být přepsán. Nevykresluje se celý dokument, ale pouze jeden element, který má být překreslen. A tedy nedochází k načítání a zobrazování elementů, které by zůstaly nezměněné. Uživatel je ušetřen různých deformací dokumentu při jeho vykreslování. Dokument se neposune na jeho úplný vrchol. Práce s dokumentem je plynulá. Zatížení webového prohlížeče, webového serveru, databázového serveru či celé sítě je výrazně nižší.

Tato technika ale ukrývá i některá omezení. Výhodné snížení zátěže však může podnítit jeho užívání v nadměrném množství. Navigace pro posun na předchozí nebo následující dokument se stává nepoužitelnou. Adresa dokumentu nekoresponduje s obsahem dokumentu, tudíž ji nelze uchovat. Načítání není explicitně signalizováno, takže v průběhu načítání může dokument nabýt dojmu, že přestal reagovat. Webové prohlížeče ji musí podporovat. Nicméně pro všechna tato omezení existují techniky, které je řeší.

[19] Europen Computer Manufacturers Association
[20] AJAX – Asynchronous JavaScript and XML

4.6.1 Historie

Interpret Java přidává komunikaci se vzdáleným serverem v roce 1995, která v dokumentu (X)HTML může být přítomna jako aplet. Webový prohlížeč Internet Explorer přidává element `iframe` v roce 1996. Interpret **ActiveX** ve webovém prohlížeči Internet Explorer verze 5 přidává prvek `XMLHTTP` v roce 1999. Interpret JavaScript v ostatních webových prohlížečích přidává třídu `XMLHttpRequest`. Interpret JavaScript ve webovém prohlížeči Internet Explorer verze 7 ho přidává taktéž. Interpret **Flash** verze 4 přidává komunikaci se vzdáleným serverem, která v dokumentu (X)HTML může být přítomna jako objekt.

4.6.2 Třída `XMLHttpRequest`

Třída `XMLHttpRequest` zprostředkovává HTTP(S) spojení v asynchronním okamžiku. Výstupem mohou být data libovolného typu stejně jako tomu je u HTTP(S), ale nejčastěji je to právě dokument XML jak napovídá samotný název třídy. Třída je přijata jako koncept W3C roku 2006 a později jako kandidát na standard W3C roku 2010 [1]. Stav objektu je promítán do atributu `readyState`.

Stav `0/UNSENT` indikuje vytvoření objektu. Stav `1/OPENED` indikuje úspěšnost metody `open`, během kterého mohou být volány metody `setRequestHeader` a `send`. Stav `2/HEADERS_RECEIVED` indikuje přijetí všech hlaviček HTTP(S) odpovědi, během kterého může být čten atribut `status`. Stav `3/LOADING` indikuje přijímání těla HTTP(S) odpovědi. Stav `4/DONE` indikuje ukončení HTTP(S) spojení. Funkci, jejíž ukazatel je předán atributu `onreadystate\-change`, objekt volá při změně stavu. Načtení dokumentu XML znázorňuje následující kód v jazyce JavaScript.

```
function get ( address )
{
  var client = new XMLHttpRequest ( ) ;                                          // vytvořit objekt
  client.onreadystatechange = function ( )              // definovat chování při změně stavu objektu
                              {
                                if ( this.readyState != 4 )           // očekávat stav objektu "DONE"
                                  return ;

                                if ( this.status != 200 )              // očekávat stav odpovědi "OK"
                                  return ;

                                show ( this.responseXML.getElementById('test').firstChild.data ) ;
                                                                  // zpracovat data HTTP(S) odpovědi
                              }
  client.open ( "GET", address ) ;               // zadat metodu "GET" a adresu do HTTP(S) požadavku
  client.send ( ) ;                                                 // odeslat HTTP(S) požadavek
}
```

Odeslání dat znázorňuje následující kód v jazyce JavaScript.

```
function post ( address, message )
{
 var client = new XMLHttpRequest ( ) ;                                           // vytvořit objekt
 client.open ( "POST", address ) ;               // zadat metodu "POST" a adresu do HTTP(S) požadavku
 client.setRequestHeader ( "Content-Type", "text/plain; charset=UTF-8" ) ;
                                                        // přidat hlavičku do HTTP(S) požadavku
 client.send ( message ) ;                                          // odeslat HTTP(S) požadavek
}
```

Načtení pouze HTTP(S) hlavičky znázorňuje následující kód v jazyce JavaScript.

```
function check ( address )
{
  var client = new XMLHttpRequest ( ) ;                                    // vytvořit objekt
  client.onreadystatechange = function ( )              // definovat chování při změně stavu objektu
                              {
                                if ( this.readyState != 4 )           // očekávat stav objektu "DONE"
                                  return ;

                                show ( this.status ) ;              // zpracovat stav HTTP(S) odpovědi
                              }
  client.open ( "HEAD", address ) ;              // zadat metodu "HEAD" a adresu do HTTP(S) požadavku
  client.send ( ) ;                                                    // odeslat HTTP(S) požadavek
}
```

4.7 Hašovací algoritmus

Hašovací algoritmus je takový algoritmus, pro který platí:

- Vstup je libovolné délky.
- Výstup je konstantní délky.
- Malá změna vstupu reflektuje velkou změnu výstupu.
- Z výstupu není možné zrekonstruovat vstup.
- Dva různé vstupy nemají stejný výstup.

Vstup se označuje jako **text** a výstup se označuje jako **hash**, **miniatura**, **fingerprint**, **otisk** či nesprávně **kontrolní součet**. Hašovací algoritmus je často používán ve spojitosti s ukládáním hesel, integritou dat či podpisem. Mezi hašovací algoritmy patří zejména MD a SHA. Hašovací algoritmy jsou aplikovány v odstavci (3.8.2) o otiskovém algoritmu, (3.8.5) o podpisovém algoritmu a (3.8.11) o iterovaném otiskovém algoritmu.

4.7.1 MD

MD[21] je rodina hašovacích algoritmů **MD2**, **MD4**, **MD5** a **MD6**. Z nich je používaným zejména algoritmus MD5. Otisk vytvořený algoritmem MD5 je vždy velikosti 128 bitů.

4.7.2 SHA

SHA[22] je rodina hašovacích algoritmů **SHA-1**, **SHA-224**, **SHA-256**, **SHA-384** a **SHA-512**. Algoritmy jiné než SHA-1 se souhrnně uvádějí jako **SHA-2**. Otisk vytvořený algoritmem SHA-1 je vždy velikosti 160 bitů. Otisk vytvořený algoritmem jiným než SHA-1 je vždy velikosti odpovídající číslu v jeho názvu.

4.8 Asymetrická kryptografie

Asymetrická kryptografie je taková kryptografie, při které není možné šifrování a dešifrování provádět tím samým klíčem.

Tím se liší od **symetrické kryptografie**, při které je nutné šifrování a dešifrování provádět tím samým klíčem. Odesilatel a příjemce si mezi sebou nemusejí vyměňovat tajný klíč.

[21] MD – Message Digest
[22] SHA – Secure Hash Algorithm

Seskupení těchto klíčů se označuje jako **pár klíčů**. **Veřejný klíč** je přístupný veřejnosti a jemu komplementární **soukromý klíč** musí být držen v utajení. Data zašifrovaná jedním klíčem by měla být dešifrovatelná pouze klíčem jemu komplementárním. Klíče mají mezi sebou jistou matematickou souvislost, avšak z veřejného klíče by nemělo být možné odvodit klíč soukromý.

Vstup pro šifrování a výstup po dešifrování se označuje jako **text** a vstup pro dešifrování a výstup po šifrování se označuje jako **šifra**.

Šifrování veřejným klíčem se označuje jako **utajování**, neboť šifru může zašifrovat více odesilatelů a dešifrovat pouze jeden příjemce. Původní text je držen v utajení a šifra může být přístupná veřejnosti.

Šifrování soukromým klíčem se označuje jako **podepisování** a dešifrování veřejným klíčem se označuje jako **ověřování**, neboť šifru může zašifrovat pouze jeden odesilatel a dešifrovat více příjemců. Vzniklá šifra se označuje jako **podpis**. Původní text i podpis jsou přístupné veřejnosti. Pro konstantní velikost podpisu se používá otisková funkce.

Aby zpráva byla ověřitelná a utajitelná zároveň, musí se podepsat soukromým klíčem odesilatele a pak utajit veřejným klíčem příjemce; nebo se musí utajit veřejným klíčem příjemce a pak podepsat soukromým klíčem odesilatele.

Šifrovací a dešifrovací algoritmy se od sebe často liší. Šifrovací algoritmus je definován jako

$$c = f(m, e), \tag{4.1}$$

a dešifrovací algoritmus je definován jako

$$m = g(c, d), \tag{4.2}$$

kde m je zpráva, c je šifra, f je šifrovací funkce, g je dešifrovací funkce, e je jeden klíč a d je druhý klíč.

Jádrem asymetrických algoritmů jsou jednocestné funkce **faktorizace čísel** či **diskrétní algoritmus**. Jejich neprolomitelnost spočívá v pomalu stoupajícím trendu výpočetního výkonu, při kterém by rozluštění dosahovalo nepřijatelného času. V případě faktorizace čísel by rozluštění znamenalo rozklad velkého čísla na součin provočísel. Mezi asymetrické algoritmy patří zejména RSA, **ElGamal**, **DH**[23], **DSA**[24]/**DSS**[25] a algoritmy založené na eliptických křivkách.

4.8.1 RSA

RSA[26] je asymetrický algoritmus prvně vhodný pro podepisování a je považován za dostatečně bezpečný při použití dostatečně dlouhých klíčů. Podle specifikace **PKCS #1 2.1**[27] [34] je postup tvorby páru klíčů následující.

Zvolí se dvě různá velká dokonale náhodná prvočísla p a q. Vypočítá se **modulus** n jako

$$n = p\,q. \tag{4.3}$$

Vypočítá se Eulerova funkce $\varphi(n)$ jako

$$\varphi(n) = (p - 1)(q - 1). \tag{4.4}$$

[23] DH – Diffie-Hellman
[24] DSA – Digital Signature Algorithm
[25] DSA – Digital Signature Standard
[26] RSA – Rivest, Shamir, Adleman
[27] PKCS – Public Key Cryptography Standards

Zvolí se **veřejný exponent** e tak, aby platilo

$$1 < e < \varphi(n), \tag{4.5}$$

$$\mathrm{NSD}(e, \varphi(n)) = 1, \tag{4.6}$$

kde NSD je největší společný dělitel, tj. aby veřejný exponent e a Eulerova funkce $\varphi(n)$ byly nesoudělné. Vypočítá se **soukromý exponent** d jako

$$d = e^{-1} \mod \varphi(n) \tag{4.7}$$

nebo ještě lépe tak, aby platilo

$$de = 1 \mod \lambda, \tag{4.8}$$

kde

$$\lambda = \mathrm{NSN}(p - 1, q - 1), \tag{4.9}$$

kde NSN je nejmenší společný násobek, tj. aby soukromý exponent d byl násobně inverzní vůči

$$e \mod \varphi(n). \tag{4.10}$$

Zpráva m se pak utají jako

$$c = m^e \mod n \tag{4.11}$$

a šifra c se dešifruje jako

$$m = c^d \mod n; \tag{4.12}$$

nebo se zpráva m podepíše jako

$$s = m^d \mod n \tag{4.13}$$

a podpis s se ověří jako

$$m = s^e \mod n. \tag{4.14}$$

Dvojice modulu a veřejného exponentu (n, e) představuje veřejný klíč; a dvojice modulu a soukromého exponentu (n, d) představuje soukromý klíč. Podle odstavce (4.8) by měl rozklad čísla n na součin prvočísel p a q dosahovat nepřijatelného času.

Délka čísel p a q by měla být stejná či alespoň podobná. Délka modulu bývá v řádech set či tisíců bitů. Velikostí klíče se má na mysli právě velikost modulu. Zatímco délka veřejného exponentu může být jen 24 bitová, přičemž veřejný exponent je často roven `0x10001`. Kryptografické knihovny implementují podporu pro extrémně velká čísla a implementují opatření proti známým útokům kvůli tvorbě silných klíčů. Algoritmus RSA je aplikován v odstavci (6.5.8) o třídě `RSA_public_key`.

4.9 OpenSSL

OpenSSL[28] je knihovna a soubor nástrojů pro kryptografii. Implementuje protokoly **SSL**[29] a **TLS**[30]. Je založena na knihovně **SSLeay**, kterou vyvinuli Eric A. Young a Tim J. Hudson. Projekt je spravován světovou komunitou dobrovolníků. Knihovna je aplikována v odstavci (6.5.8) o třídě `RSA_public_key`.

[28] OpenSSL – Open Secure Sockets Layer
[29] SSL – Secure Sockets Layer
[30] TLS – Transport Layer Security

4.9.1 Modul do PHP

Knihovna existuje jako modul do interpretu PHP. Knihovna samotná nabízí nespočet funkcí, nicméně modul je všechny nepodporuje. Možná proto, že jeho vývoj se pozastavil před 3 roky vzhledem k tomu, že poslední verze 0.9.8e byla vydána 1. července 2008. Interpret PHP je nutné zkompilovat s argumentem `--with-openssl[=DIR]`. Interpret PHP o verzi 4.0.5 až 4.3.1 vyžaduje modul o verzi 0.9.5 nebo vyšší; a interpret PHP o verzi 4.0.4 až 4.3.2 vyžaduje modul o verzi 0.9.6 nebo vyšší. Knihovna vyžaduje přístup k náhodnému nebo pseudonáhodnému generátoru čísel, v Linuxu jimi jsou zařízení `/dev/urandom` a `/dev/random` [39].

Drtivá většina funkcí jako argument vyžaduje certifikát nebo klíč [40]. V případě certifikátu může být argumentem zdroj získaný funkcí `OpenSSL_X509_read`, cesta k souboru ve formátu PEM nebo řetězec ve formátu PEM. V případě veřejného klíče může být argumentem zdroj získaný funkcí `OpenSSL_get_PublicKey`, zdroj získaný funkcí `OpenSSL_X509_read` nebo řetězec ve formátu PEM. V případě soukromého klíče může být argumentem zdroj získaný funkcí `OpenSSL_get_PrivateKey`, cesta k souboru ve formátu PEM nebo řetězec ve formátu PEM.

5 Návrh a implementace

V životním cyklu byly etapy návrhu a imlementace blízko sebe natolik, že návrh a implementace probíhaly v podstatě současně.

Implementace je napsána převážně v jazyce PHP. Hlavními důvody pro volbu jazyka PHP jsou volnost v datových typech proměnných a indexů, klíčů, prvků v polích a zkušenosti podle odstavce (4.3) o PHP. Bez těchto vlastností by byla implementace velice komplikovaná a časově nestíhatelná. V implementaci jsou s výhodou používany třídy a dědičnost tříd. Proměnné a funkce nad tentýž objektem jsou soustředěny do tříd jako metody a atributy. Každá třída se nachází ve vlastním souboru se stejnojmenným názvem. Ostatní proměnné a funkce zůstaly osamoceny.

Kvůli účelnosti a přehlednosti byla implementace rozdělena do několika modulů. Kód svázaný s rozšířením DNSSEC je soustředěn do samostatného modulu nazvaného jako `protocol` a kód svázaný s ověřováním je soustředěn do samostatného modulu nazvaného jako `validator`. Modul `validator` používá modulu `protocol`, ale nikoli obráceně, tudíž modul `protocol` je možné označit i za knihovnu a modul `validator` za aplikaci. Pro každý modul je vyhrazena samostatná kapitola (6) o knihovně `protocol` a (7) o aplikaci `validator`.

5.1 Vývoj a testování

Implementace byla vyvíjena a testována na dedikovaném serveru napojeném na páteřní počítačovou síť. Běh umožňoval operační systém o distribuci **CentOS** na bázi GNU/Linux verze 2.6.18-194.26.1.el5 vydané 9. listopadu 2010. Instrukce zpracovával procesor **Intel Celeron** o frekvenci 2,53 GHz. Kód v jazyce PHP interpretoval interpret **PHP** o verzi 5.3.6 vydané 19. dubna 2011. Kód v jazyce SQL interpretoval databázový systém **MySQL** o verzi 5.1.57 vydané 5. května 2011. Komunikaci zprostředkovával webový server **Apache** o verzi `2.2.3` vydané 27. července 2006.

Pro účely testování byla podepsána jedna doména z vlastních třech. Při testování se modifikoval zónový soubor následovně:

- Porušoval se veřejný exponent v poli `Public-Key` v záznamu typu `DNSKEY`.
- Porušoval se modulus v poli `Public-Key` v záznamu typu `DNSKEY`.
- Odstraňoval se příznak `SEP` z pole `Flags` v záznamu typu `DNSKEY`.
- Přidával se příznak `SEP` z pole `Flags` v záznamu typu `DNSKEY`.
- Odstraňoval se příznak `Zone` z pole `Flags` v záznamu typu `DNSKEY`.
- Přidával se příznak `Zone` z pole `Flags` v záznamu typu `DNSKEY`.
- Odstraňoval se záznam typu `DNSKEY`.
- Porušovalo se pole `Signature` v záznamu typu `RRSIG`.
- Porušovalo se pole `Key-Tag` v záznamu typu `RRSIG`.

- Odstraňoval se záznam typu `RRSIG`.
- Odstraňoval se typ z pole `Type-Bit-Maps` v záznamu typu `NSEC`.
- Přidával se typ do pole `Type-Bit-Maps` v záznamu typu `NSEC`.
- Odstraňoval se záznam typu `NSEC`.
- A spousta dalšího...

Knihovna se testovala testovacími skripty a skutečnou komunikací. Testovací skript obsahoval testovací kód pro každou třídu. Aplikace se testovala dotazy na vlastní doménu a na dómeny:

- „“
- „`cz.`“
- „`com.`“
- „`net.`“
- „`org.`“
- „`google.com.`“
- „`vutbr.cz.`“
- A několik dalších...

6 Knihovna `protocol`

Knihovna `protocol`, jak už název napovídá, implementuje protokol DNSSEC. Skládá se z tříd pro vytváření, kontrolování, kódování a dekódování dat, jakými jsou pole, hlavičky, dotazy, záznamy a zprávy. Dále z tříd pro kódování, dekódování, značkování a hashování vstupů. A nakonec z tříd pro dešifrování podpisů. Samozřejmostí je třída pro přenos dat mezi klientem a serverem. Neposkytuje žádné uživatelské rozhraní, neočekává data na vstupu a negeneruje data na výstup.

6.1 Existující knihovny

Pro systém doménových jmen bylo v repositáři **PEAR**[1] jazyka PHP nalezeno několik knihoven. Pro rozšíření DNSSEC to byla pouze knihovna `Net/DNS2` verze `1.0.1`. Analyzovaná knihovna na první pohled vypadala slibně a stala se kandidátem pro připojení k aplikaci, ale na pohled druhý není komplexní a je vhodná především pro zobrazování dat. Ačkoli analyzovaná knihovna podporuje rozšíření DNSSEC, tak její autor zřejmě nezaznamenal ani hlavní principy rozšíření DNSSEC a implementoval analyzovanou knihovnu v podobném duchu jako systém DNS. Bylo minimálně zjištěno, že analyzovaná knihovna nepředpokládá, aby záznam typu `OPT` mohl být součástí odchozí zprávy; obsahuje závažné chyby v kódování a dekódování; a možná další nedostatky.

Rozhodnutí pro vlastní knihovnu přináší hlavně přednosti. Orientace v kódu, která by ve vlastním měla být lepší. Implementace vlastní knihovny dovoluje protokol poznat na té nejnižší úrovni, čehož lze využít v implementaci aplikace. Vlastní knihovnu lze přizpůsobit potřebám aplikace. Řešení se bude více blížit požadavku minimálních externích závislostí. Principy rozšíření DNSSEC naznačují, že knihovna a aplikace jsou na sobě tak závislé, že propojení cizí knihovny a vlastní aplikace by bylo komplikované. Struktura a principy analyzované knihovny nemusejí vyhovovat aplikaci. Všechny tyto aspekty se přiklánějí pro implementaci knihovny vlastní.

6.2 Kódování a dekódování

Ve specifikacích je pro převod z čitelné struktury do zprávy používán pojem, který má velice příhodné jméno *wiring*; obdobně pro převod ze zprávy do čitelné struktury je používán pojem *unwiring*. Navzdory nevhodnosti jakéhokoli překladu těchto pojmů budou zastoupeny vhodnějšími pojmy **kódování** a **dekódování**. Dekódování je známé též jako *parsing*.

Pro kódování a dekódování se obvykle používají bitové operátory `<<` pro posun doleva, `>>` pro posun doprava, `|` pro inkluzivní součet, `^` pro exkluzivní součet, `&` pro součin a `~` pro

[1] PEAR – PHP Extension and Application Repository

negaci. Binární data jsou v ideálním případě reprezentována jako pole znaků. Číslo menší než jeden bajt se kóduje tak, že se posune doleva na příslušnou pozici a přičte se k bajtu, jak znázorňuje následující kód.

```
function wire ( $bits_0, $bits_2, $bits_5, $bits_7, & $char )
{
  $number8 = ( ( $bits_0 << 7 ) | ( $bits_2 << 5 ) | ( $bits_6 << 1 ) | $bits_7 ) ;
  $char = Chr ( $number8 ) ;
}
```

Číslo větší než jeden bajt se kóduje tak, že se každý bajt posune úplně doprava a připojí k výstupu, jak znázorňuje následující kód.

```
function wire ( $number32, & $chars )
{
  $chars .= Chr ( $number32 >> 24 ) ;
  $chars .= Chr ( $number32 >> 16 ) ;
  $chars .= Chr ( $number32 >> 8 ) ;
  $chars .= Chr ( $number32 ) ;
}
```

Čísla větší a ostatní data se kódují funkcí `pack`, která se očekávají v hexadecimálním formátu, jak znázorňuje následující kód.

```
function wire ( $number512, & $chars )
{
  $chars .= pack ( "H*", $number512 ) ;
}
```

Číslo menší než jeden bajt se dekóduje tak, že se posune úplně doprava a vynásobí se s maskou pro ponechání požadovaných bitů a vynulování nechtěných bitů, jak znázorňuje následující kód.

```
function unwire ( $chars, $offset, & $bits_0 )
{
  $number8 = Ord ( $chars[$offset ++] ) ;
  $bits_0 = ( ( $number8 >> 6 ) & 0x02 ) ;
}
```

Číslo větší než jeden bajt se dekóduje tak, že se každý bajt posune doleva na příslušnou pozici a přičte se k výstupu, jak znázorňuje následující kód.

```
function unwire ( $chars, $offset, $variable32 )
{
  $number8 = Ord ( $chars[$offset ++] ) ;
  $variable32 = ( $number8 << 24 ) ;

  $number8 = Ord ( $chars[$offset ++] ) ;
  $variable32 |= ( $number8 << 16 ) ;

  $number8 = Ord ( $chars[$offset ++] ) ;
  $variable32 |= ( $number8 << 8 ) ;

  $number8 = Ord ( $chars[$offset ++] ) ;
  $variable32 |= $number8 ;
}
```

Čísla větší a ostatní data se dekódují funkcí `unpack`, která se očekávají v hexadecimálním formátu, jak znázorňuje následující kód.

```
function unwire ( $chars, $offset, & $number512 )
{
  $chars = SubStr ( $chars, $offset, 64 ) ;
  $number512 = unpack ( "H*", $chars ) ;
  $number512 = $number512[0] ;
}
```

6.3 Třídy základní

Pro bližší popis byly vybrány třídy `base`, `assignments` a `stream`.

6.3.1 Třída `base`

Třída `base` spravuje chybová hlášení. Její přístup je definován jako abstraktní, protože její instance by neměly význam a pro veškeré třídy je vrcholem v hiearchii dědičnosti tříd. Metoda `add_error` přidává chybové hlášení. Metoda `add_errors` přidává seznam chybových hlášení, která se používá zejména při předávání seznamu chybových hlášení z vnořeného objektu.

6.3.2 Třída `assignments`

Třída `assignments` uchovává jména pro pojmenované hodnoty. Její přístup je definován jako statický, protože její instance by neměly význam a obsahuje pouze konstanty. Pojmenované hodnoty byly čerpány z aktuálních registrů [19, 20, 16, 18, 17].

6.3.3 Třída `stream`

Třída `stream` navazuje, odesílá, přijímá a uzavírá spojení. Zapouzdřujícím protokolem z transportní vrstvy modelu OSI může být UDP nebo TCP. Lokální port může být zvolen ručně nebo automaticky operačním systémem. Vzdálený port může být zvolen ručně nebo automaticky na 53. Při použití protokolu UDP je maximální délka zprávy stanovena na 65535. Při použití protokolu TCP je před zprávou umístěno 16 bitové číslo s velikostí zprávy. Metoda `open` otevírá spojení se serverem a metoda `close` uzavírá spojení se serverem.

Metoda `write` odesílá odchozí zprávu. Před odesláním vyčkává na prázdný soket po dobu časového limitu. Pro kontrolu porovnává počet odesílaných znaků s počtem odeslaných znaků.

Metoda `read` přijímá příchozí zprávu. Před příjetím vyčkává na neprázdný soket po dobu časového limitu. Pro kontrolu porovnává počet očekávaných znaků s počtem přijatých znaků, ale pochopitelně jen při použití protokolu TCP.

6.4 Třídy dat

Hierarchie datových tříd striktně koresponduje s hierarchií dat podle odstavce (2.5) o datech a hierarchii dat. S poli korespondují třídy prefixu `field`, s dotazem třída `question`, se záznamy třídy prefixu `record`, s hlavičkou `header` a se zprávou `message`. Pro zjednodušení jsou příznaky zahrnuty již ve třídách prefixu `field`.

V následujích třídách se nachází metoda `set` pro ukládání jednotlivých polí, metoda `get` pro načítání jednotlivých polí, metoda `wire` pro kódování a metoda `unwire` pro dekódování. Metoda `wire` má jako první argument `data`. Metoda `unwire` má jako první argument `data`,

jako druhý `offset` a někdy jako třetí `length`. Po kódování se data automaticky připojují k hodnotě v argumentu `data` a po dekódování se hodnota v argumentu `offset` automaticky posunuje. To má význam při řetězovém kódování nebo dekódování několika třídami. Před dekódováním se kontroluje dostatek dat určených pro dekódování. Ve třídách se mohou nacházet další pro ně specifické metody.

6.4.1 Třídy prefixu `field`

Třídy prefixu `field` vytvářejí, kontrolují, kódují a dekódují pole podle odstavců (2.6, 3.5) o polích. V následujích třídách se navíc nachází metoda `check` pro kontrolu formátu pole. Tato metoda je volána před kódováním a po dekódování. Třídy zahrnují přichystané třídy pro pole, která nabývají obecných hodnot; jako jsou obecná čísla, binární data či prosté texty. Zahrnují přichystané třídy i pro pole, která nabývají pojmenovaných hodnot. Třída `field_name` před kódováním převádí jméno do kanonické formy podle odstavce (3.3.1) o kanonické formě záznamu. Implementované třídy znázorňuje následující seznam.

- `field_number8` pro pole, jehož hodnoty nabývají 8 bitových čísel.
- `field_number16` pro pole, jehož hodnoty nabývají 16 bitových čísel.
- `field_number32` pro pole, jehož hodnoty nabývají 32 bitových čísel.
- `field_string` pro pole, jehož hodnoty nabývají binárních dat.
- `field_text` pro pole, jehož hodnoty nabývají prostých textů.
- `field_name` pro pole typu `Name` podle (2.6.1).
- `field_type` pro pole typu `Type`.
- `field_class` pro pole typu `Class`.
- `field_IPv4` pro pole typu `Address` v záznamu typu `A` podle (2.9.3).
- `field_IPv6` pro pole typu `Address` v záznamu typu `AAAA` podle (2.9.4).
- `field_time` pro pole typu `Time`.
- `field_protocol` pro pole typu `Protocol` podle (3.7.3).
- `field_algorithm` pro pole typu `Algorithm` podle (3.5.4).
- `field_digest_type` pro pole typu `Digest-Type` podle (3.5.3).
- `field_hash_algorithm` pro pole typu `Hash-Algorithm`.
- `field_type_bit_maps` pro pole typu `Type-Bit-Maps` podle (3.5.5).
- `field_flags_header` pro pole typu `Flags` v hlavičce podle (3.6).
- `field_flags_Opt` pro pole typu `Flags` v záznamu typu `OPT` podle (3.7.1).
- `field_flags_DNSKey` pro pole typu `Flags` v záznamu typu `DNSKEY` podle (3.7.3).
- `field_flags_NSec3` pro pole typu `Flags` v záznamu typu `NSEC3` podle (3.7.6).

6.4.2 Třída `header`

Třída `header` vytváří, kóduje a dekóduje hlavičku podle odstavce (3.6) o hlavičce. Pro operace se svými poli používá třídy prefixu `field`.

6.4.3 Třída `question`

Třída `question` vytváří, kóduje a dekóduje dotaz podle odstavce (2.8) o dotazu. Pro operace se svými poli používá třídy prefixu `field`.

6.4.4 Třídy prefixu `record`

Třída `record` implementuje záznam obecného typu podle odstavce (2.9.1) o záznamu. Uplatňuje se pouze při dekódování a dekóduje společná pole záznamu všech typů, čímž zjistí typ záznamu a následně předá štafetu třídě, která dekóduje záznam zjištěného typu.

Třídy prefixu `record` vytvářejí, kódují a dekódují záznamy podle odstavců (2.9, 3.7) o záznamech. Pro operace se svými poli používají třídy prefixu `field`. Implementované třídy znázorňuje následující seznam.

- `record_SOA` pro záznam typu `SOA` podle (2.9.2).
- `record_A` pro záznam typu `A` podle (2.9.3).
- `record_AAAA` pro záznam typu `AAAA` podle (2.9.4).
- `record_NS` pro záznam typu `NS` podle (2.9.7).
- `record_TXT` pro záznam typu `TXT`.
- `record_Opt` pro záznam typu `OPT` podle (3.7.1).
- `record_RRSig` pro záznam typu `RRSIG` podle (3.7.2).
- `record_DNSKey` pro záznam typu `DNSKEY` podle (3.7.3).
- `record_DS` pro záznam typu `DS` podle (3.7.4).
- `record_NSec` pro záznam typu `NSEC` podle (3.7.5).
- `record_NSec3` pro záznam typu `NSEC3` podle (3.7.6).
- `record_NSec3Param` pro záznam typu `NSEC3PARAM`.

6.4.5 Třída `message`

Třída `message` vytváří, kóduje a dekóduje zprávu podle odstavce (2.10) o zprávě. Pro operace s hlavičkou používá třídu `header`, pro operace s dotazy používá třídu `question` a pro operace se záznamy používá třídy prefixu `record`.

6.5 Třídy algoritmů

Následující třídy implementují algoritmy a vstupy pro algoritmy podle odstavce (3.8) o algoritmech.

6.5.1 Třída `key-tag-enter`

Třída `key_tag_enter` vytváří, kóduje a dekóduje vstup pro značkovací algoritmus podle odstavce (3.8.1) o značkovacím algoritmu. Pro operace se svými poli používá třídy prefixu `field`.

6.5.2 Třída `key-tag`

Třída `key_tag` implementuje značkovací algoritmus podle odstavce (3.8.1) o značkovacím algoritmu. Pro operaci se svým vstupem používá třídu `key_tag_enter`.

6.5.3 Třída `digest-enter`

Třída `digest_enter` vytváří, kóduje a dekóduje vstup pro otiskový algoritmus podle odstavce (3.8.2) o otiskovém algoritmu. Pro operace se svými poli používá třídy prefixu `field`.

6.5.4 Třídy prefixu `digest-hash`

Třídy prefixu `digest_hash` implementují otiskový algoritmus podle odstavce (3.8.2) o otiskovém algoritmu. Pro operaci se svým vstupem používá třídu `digest_enter`. Implementované třídy znázorňuje následující seznam.

- `digest_SHA1` pro otiskový algoritmus typu `SHA1` podle (3.8.3).
- `digest_SHA256` pro otiskový algoritmus typu `SHA256` podle (3.8.4).

6.5.5 Třída `signature-enter`

Třída `signature_enter` vytváří, kóduje a dekóduje vstup pro podpisový algoritmus podle odstavce (3.8.5) o podpisovém algoritmu. Pro operace se svými poli používá třídy prefixu `field`. Před kódováním kanonicky seřadí sadu podle odstavce (3.3.3) o kanonickém pořadí záznamů v sadě.

6.5.6 Třídy prefixu `signature-hash`

Třídy prefixu `signature_hash` implementují otiskové algoritmy podpisových algoritmů podle odstavce (3.8.5) o podpisovém algoritmu. Pro operaci se svým vstupem používá třídu `signature_enter`. Implementované třídy znázorňuje následující seznam.

- `signature_DSA_hash` pro podpisový algoritmus typu `DSA`.
- `signature_RSA_MD5_hash` pro podpisový algoritmus typu `RSAMD5` podle (3.8.7).
- `signature_RSA_SHA1_hash` pro podpisový algoritmus typu `RSASHA1` podle (3.8.8).
- `signature_RSA_SHA256_hash` pro podpisový algoritmus typu `RSASHA256` podle (3.8.9).
- `signature_RSA_SHA512_hash` pro podpisový algoritmus typu `RSASHA512` podle (3.8.10).

6.5.7 Třídy prefixu `signature-plaintext`

Třídy prefixu `signature_plaintext` vytvářejí, kódují a dekódují vstupy pro podpisové algoritmy konkrétních typů podle odstavce (3.8.5) o podpisovém algoritmu. Implementované třídy znázorňuje následující seznam.

- `signature_DSA_plaintext` pro podpisový algoritmus typu `DSA`.
- `signature_RSA_MD5_plaintext` pro podpisový algoritmus typu `RSAMD5` podle (3.8.7).
- `signature_RSA_SHA1_plaintext` pro podpisový algoritmus typu `RSASHA1` podle (3.8.8).
- `signature_RSA_SHA256_plaintext` pro podpisový alg. typu `RSASHA256` podle (3.8.9).
- `signature_RSA_SHA512_plaintext` pro podpisový alg. typu `RSASHA512` podle (3.8.10).

6.5.8 Třída `RSA-public-key`

Třída `RSA_public_key` dešifruje podpis algoritmu rodiny `RSA`. Tato třída je stěžejní částí implementace, která si zde zaslouží více pozornosti. Využívá k tomu knihovnu **OpenSSL** jako rozšíření knihovny PHP. Třída je testována knihovnou OpenSSL verze 0.9.8e vydané 1. července 2008. Propojení s knihovnou OpenSSL není přímočaré. Jelikož záznam typu `DNSKEY` poskytuje veřejný klíč RSA jako veřejný exponent a modulus a funkce knihovny OpenSSL podporují pouze předávání certifikátu či klíče ve formátu **PEM**[2] podle odstavce (4.9) o OpenSSL, tak je nezbytných několik pomocných metod.

Privátní metoda `export` přetváří veřejný exponent a modulus na veřejný klíč ve formátu PEM. Metoda je znázorněna v následujícím zkráceném kódu v jazyce PHP.

[2] PEM – Privacy Enhanced Mail

```php
define ( "DER_INTEGER",    0x02 ) ;
define ( "DER_STRING",     0x03 ) ;
define ( "DER_SEQUENCE",   0x30 ) ;
define ( "DER_PREFIX_RSA", "06092a864886f70d0101010500" ) ;

private function export ( $public_exponent, $modulus, & $public_key_PEM )
{
  $public_exponent = pack ( "H*", $public_exponent ) ;                  // zakódovat veřejný exponent
  $modulus         = pack ( "H*", $modulus ) ;                                  // zakódovat modulus
  $prefix          = pack ( "H*", DER_PREFIX_RSA ) ;                          // zakódovat RSA prefix

  // zakódovat složky do formátu DER
  segment ( DER_SEQUENCE, $prefix ) ;                        // zakódovat RSA prefix jako DER sekvenci
  segment ( DER_INTEGER, $public_exponent ) ;             // zakódovat veřejný exponent jako DER číslo
  segment ( DER_INTEGER, $modulus ) ;                            // zakódovat modulus jako DER číslo
  $public_key_DER = $modulus . $public_exponent ;                // spojit modulus a veřejný exponent
  segment ( DER_SEQUENCE, $public_key_DER ) ;                // zakódovat sekvenci jako DER sekvenci
  segment ( DER_STRING, $public_key_DER ) ;               // zakódovat sekvenci jako DER bitový řetězec
  $public_key_DER = $prefix . $public_key_DER ;                  // před sekvenci připojit RSA prefix
  segment ( DER_SEQUENCE, $public_key_DER ) ;                 // zakódovat sekvenci jako DER sekvenci

  // zakódovat veřejný klíč do formátu PEM
  $public_key = base64_encode ( $public_key_DER ) ;        // zakódovat veřejný klíč do formátu base64
  $public_key_PEM = "-----BEGIN PUBLIC KEY-----\n" ;     // před veřejný klíč připojit počáteční okraj
  $length = StrLen ( $public_key ) ;
  for ( $i = 0 ; $i < $length ; $i += 64 )                       // zalomit veřejný klíč po 64 znacích
  {
    $block = SubStr ( $public_key, $i, 64 ) ;
    $public_key_PEM .= "$block\n" ;
  }
  $public_key_PEM .= "-----END PUBLIC KEY-----\n" ;          // za veřejný klíč připojit koncový okraj
}
```

Veškeré komponenty jsou kódovány do formátu **DER**[3] podle specifikace **X.680** [37, str. 12] o **ASN.1**[4]. Prexix, který je pro algoritmus RSA definován jako

0x06092a864886f70d0101010500,

je zakódován jako DER sekvence. Veřejný exponent a modulus jsou zakódovány jako DER číslo a jsou spojeny v tomto pořadí. Tento řetězec je zakódován jako DER sekvence a následně jako DER bitový řetězec. Tomuto řetězci předchází již dříve zakódovaný prefix RSA. Nakonec je tento řetězec zakódován jako DER sekvence. Právě vznikl veřejný klíč ve formátu DER.

Veřejný klíč ve formátu DER je následně překódován do formátu PEM podle specifikace **RFC 1421** [22]. Je tedy zakódován do formátu base64, zalomen po 64 znacích a z obou stran obklopen okraji. Až nyní vznikl veřejný klíč ve formátu PEM.

Privátní metoda `segment` vytváří DER segment pro metodu `export`. Metoda je znázorněna v následujícím zkráceném kódu v jazyce PHP.

```php
private function segment ( $type, & $segment )
{
  if ( ( ( $type === DER_INTEGER ) && ( Ord ( $segment ) > 0x7F ) ) || ( $type === DER_STRING ) )
    $segment = "\0$segment" ;

  $length = StrLen ( $segment ) ;
  if ( $length < 128 )                                 // zakódovat segment v závislosti na jeho velikosti
    $segment = SPrintF ( "%c%c%s", $type, $length, $segment ) ;
  else if ( $length < 0x0100 )
    $segment = SPrintF ( "%c%c%c%s", $type, 0x81, $length, $segment ) ;
  else if ( $length < 0x010000 )
```

[3] DER – Distinguished Encoding Rules
[4] ASN – Abstract Syntax Notation

```
12      $segment = SPrintF ( "%c%c%c%c%s", $type, 0x82, $length / 0x0100, $length % 0x0100, $segment ) ;
13    else
14      $segment = NULL ;
15  }
```

Konečně, veřejná metoda `decrypt` dešifruje podpis algoritmu rodiny `RSA`. Metoda je znázorněna v následujícím zkráceném kódu v jazyce PHP.

```
1  public function decrypt ( $public_key_PEM, $signature, & $plaintext )
2  {
3    $signature = pack ( "H*", $signature ) ;                                    // zakódovat podpis
4
5    $public_key = OpenSSL_PKey_get_public ( $public_key_PEM ) ; // převést klíč na OpenSSL veřejný klíč
6    OpenSSL_public_decrypt ( $signature, $plaintext, $public_key, OPENSSL_NO_PADDING ) ;
7
8    $plaintext = unpack ( "H*", $plaintext ) ;                                   // dekódovat vstup
9    $plaintext = $plaintext[0] ;
10 }
```

Výše uvedená funkce `OpenSSL_public_decrypt` má jako čtvrtý argument uvedenou konstantu `OPENSSL_NO_PADDING`, pomocí níž vrací argument `plaintext` bez zarovnání, neboť třídy prefixu `signature_plaintext` očekávají nezarovnaný vstup do podpisového algoritmu.

7 Aplikace `validator`

Aplikace `validator`, jak už název napovídá, implementuje nástroj pro ověřování rozšíření DNSSEC. Skládá se z tříd pro ukládání a načítání záznamů a sad. Samozřejmostí jsou funkce pro ověřování záznamů. Poskytuje uživatelské rozhraní, očekává data na vstupu a generuje data na výstup.

Pro zvládnutí problému je využit celý výčet technologií, jakými jsou PHP, (My)SQL, XHTML, CSS, JavaScript a AJAX. Všechny tyto technologie jsou blíže vysvětleny v kapitole (4). Využívá vlastní knihovny `protocol`, kterou podrobně vysvětluje kapitola (6).

7.1 Data a hierarchie dat

Pro účely ukládání a ověřování je hierarchie dat z odstavce (2.5) o datech a hierarchii dat mírně rozšířena. **Tabulka** se skládá z řádků a rozšiřuje sadu. **Řádek** se skládá z buněk a rozšiřuje záznam. **Buňka** může přebírat pole.

7.2 Buňky

Buňky některých typů jsou obsaženy v řádcích více typů. Pro bližší popis byly vybrány buňky typu `valid` a `verifying`.

7.2.1 Buňka `valid`

Buňka typu `valid` je datového typu `VARCHAR(5)` a určuje výsledek z ověřování řádku. Nabývá hodnoty `FALSE` pro neplatný řádek nebo `TRUE` pro platný řádek.

7.2.2 Buňka `verifying`

Buňka typu `verifying` je datového typu `TEXT` a určuje výpis z ověřování řádku; a nabývá textu po vzoru následujícího kódu.

```
1  // ověřuje se řádek typu DNSKEY KSK
2  trying row DS Id 1                          // zkouší se první řádek typu DS s˜buňkou Id o˜hodnotě 1
3  {
4    digest matched ;                                                    // buňky Digest se shodují
5    valid ;                                   // na základě výše uvedeného je řádek typu DNSKEY platný
6  }
7  valid ;                                     // na základě výše uvedeného je řádek typu DNSKEY platný
8  // ověřuje se ten samý řádek typu DNSKEY a˜řádky typu RRSIG
9  trying row RRSig                                                  // zkouší se první řádek typu RRSIG
10 {
11   trying row DNSKey Id 5                 // zkouší se první řádek typu DNSKEY s˜buňkou Ido hodnotě 5
12   {
```

```
13      signature decrypted ;                                  // dešifrování buňky Signature bylo úspěšné
14      hash matched ;                                                          // buňky Hash se shodují
15      valid ;                                  // na základě výše uvedeného je řádek typu RRSIG platný
16    }
17    signature live ;                                                  // řádek typu RRSig je časově platný
18  }
19  trying row RRSig                                                     // zkouší se druhý řádek typu RRSIG
20  {
21    ZSK ignored ;                  // řádek typu RRSIG směřující na řádek typu DNSKEY ZSK byl ignorován
22    signature live ;                                                  // řádek typu RRSSIG je časově platný
23    not valid ;                              // na základě výše uvedeného není řádek typu RRSIG platný
24  }
25  valid ;                                   // na základě výše uvedeného je řádek typu DNSKEY platný
```

Nutno poznamenat, že kód není v syntaxi žádného známého jazyka, avšak může mít s některými jistou podobnost. Jazyk je určen pro popis ověřovacího procesu. Jelikož vznikla nutnost vyznačit celek, tak se v kódu může vyskytnout blok, který se uzavírá do složených závorek.

Uvedený kód je vyňat z řádku typu `DNSKEY` KSK. Indikuje, že byl ověřován řádky typu `DS` a poté řádky typu `RRSIG`. Řádek typu `RRSIG` byl ověřován řádky typu `DNSKEY`.

7.3 Řádek

Řádek oproti záznamu navíc obsahuje buňky typu `Id`, `saved`, `valid` a `verifying`. Řádek obecného typu znázorňuje obrázek (7.1).

Obrázek 7.1: Řádek obecného typu

Buňka typu `Id` je datového typu `INT(11)` a určuje primární index řádku v rámci tabulky; a nabývá unikátní hodnoty. Buňka typu `saved` je datového typu `TIMESTAMP` a určuje čas uložení řádku; a nabývá času ve formátu `dd.mm.yy hh:mm:ss`. Spolu se sloupcem `TTL` určuje živost záznamu. Buňka typu `valid` je datového typu `VARCHAR(5)` a určuje výsledek z ověřování řádku; a nabývá hodnoty `FALSE` pro neplatný řádek nebo `TRUE` pro platný řádek. Buňka typu `verifying` je datového typu `TEXT` a určuje výpis z ověřování řádku; a vytváří se podle odstavce (7.2.2) o buňce typu `verifying`. V řádku typu `DNSKey` buňka typu `key_tag` je datového typu `INT(11)` a emuluje značku veřejného klíče, protože záznam typu `DNSKEY` ji nemá z pochopitelných důvodů.

7.4 Vyrovnávací paměť

Jako vyrovnávací paměť pro záznamy posloužil databázový systém MySQL. Hlavními důvody pro jeho volbu jsou zkušenosti a existence databázového správce **phpMyAdmin**.

Jelikož se každý typ řádku skládá z různých typů buněk, tak je typ řádku reprezentován samostatnou tabulkou v databázi. Řádky se ukládají do a načítají z tabulek. Vyrovnávací paměť je kvůli zjednodušení kódu vyžadována i kdyby se měly požadovat pouze nové řádky. Ve výchozím stavu se ukládají veškeré řádky. V případě požadavku na nové řádky se databáze může vyprázdnit před nebo po každém běhu aplikace. Implementované tabulky znázorňuje následující seznam.

- `cache_SOA` pro záznam typu `SOA`.
- `cache_A` pro záznam typu `A`.

- `cache_AAAA` pro záznam typu `AAAA`.
- `cache_NS` pro záznam typu `NS`.
- `cache_TXT` pro záznam typu `TXT`.
- `cache_RRSig` pro záznam typu `RRSIG`.
- `cache_DNSKey` pro záznam typu `DNSKEY`.
- `cache_DS` pro záznam typu `DS`.
- `cache_NSec` pro záznam typu `NSEC`.
- `cache_NSec3` pro záznam typu `NSEC3`.
- `cache_NSec3Param` pro záznam typu `NSEC3PARAM`.

Tabulka pro záznam typu `OPT` by neměla smysl. Z vyrovnávací paměti se vybírají pouze řádky s platnou živostí po vzoru následujícího kódu v jazce SQL.

```
SELECT
  `column1`,
  `column2`,
  ...
FROM
  `table`
WHERE
(
  ( ( `saved` + ( INTERVAL `TTL` SECOND ) ) > NOW( ) )
  AND ( `column1` = 'cell1' )
  AND ( `column2` = 'cell2' )
  AND ...
) ;
```

7.5 Třídy dat

Hierarchie datových tříd koresponduje s hierarchií dat podle odstavce (7.1) o datech a hierarchii dat. S řádky korespondují třídy prefixu `row` a s tabulkami korespondují třídy prefixu `cache`. V následujících třídách se nachází metoda `save`, která ukládá a metoda `load`, která načítá.

7.5.1 Třídy prefixu `row`

Třídy prefixu `row` ukládají a načítají řádky podle odstavce (7.3) o řádku. Pokud řádek již existuje, tak se neuloží. Dědí z tříd prefixu `record`. Implementované třídy znázorňuje následující seznam.

- `row_SOA` pro záznam typu `SOA`.
- `row_A` pro záznam typu `A`.
- `row_AAAA` pro záznam typu `AAAA`.
- `row_NS` pro záznam typu `NS`.
- `row_TXT` pro záznam typu `TXT`.
- `row_RRSig` pro záznam typu `RRSIG`.
- `row_DNSKey` pro záznam typu `DNSKEY`.
- `row_DS` pro záznam typu `DS`.
- `row_NSec` pro záznam typu `NSEC`.
- `row_NSec3` pro záznam typu `NSEC3`.
- `row_NSec3Param` pro záznam typu `NSEC3PARAM`.

7.5.2 Třídy prefixu `cache`

Třídy prefixu `cache` ukládají a načítají tabulky podle odstavce (7.1) o tabulce. Implementované třídy znázorňuje následující seznam.

- `cache_SOA` pro záznam typu `SOA`.
- `cache_A` pro záznam typu `A`.
- `cache_AAAA` pro záznam typu `AAAA`.
- `cache_NS` pro záznam typu `NS`.
- `cache_TXT` pro záznam typu `TXT`.
- `cache_RRSig` pro záznam typu `RRSIG`.
- `cache_DNSKey` pro záznam typu `DNSKEY`.
- `cache_DS` pro záznam typu `DS`.
- `cache_NSec` pro záznam typu `NSEC`.
- `cache_NSec3` pro záznam typu `NSEC3`.
- `cache_NSec3Param` pro záznam typu `NSEC3PARAM`.

7.6 Schopnosti

Dotaz je žádost o sadu. Dotaz je součástí odchozí zprávy a sada je součástí příchozí zprávy. Dotazování je proces, ve kterém se prochází serverový strom z vyšší úrovně do nižší za účelem nalezení této sady. Přechod z vyšší úrovně do nižší je dán přesměrováním pomocí sady typu `NS`. Aplikace s přesměrováním samozřejmě počítá.

Pro účely dotazování je nutné v nejvyšší úrovni znát adresy serverů – kořenových serverů. Adresy kořenových serverů se dotazováním zjistit nedají, neboť nejsou k dispozici adresy kořenových serverů. Vzhledem k této smyčce se adresy kořenových serverů musejí zjistit jiným způsobem. Aplikace s kořenovými servery počítá tím způsobem, že autonomně stahuje seznam kořenových serverů z autoritativních webových stránek.

Během dotazování může dojít k vyvolání vnořeného dotazu, neboť některé sady nebo záznamy mohou záviset na sadách jiných. Kupříkladu na sadě typu `DNSKEY` jsou závislé záznamy typu `RRSIG` v rámci ověřování, nebo na sadě typu `DS` je závislá sada typu `DNSKEY` v rámci ověřování, nebo na sadě typu `A` je závislá sada typu `NS` v rámci přesměrovávání. Pro eliminování vnořených dotazů může rafinovaný server předpovědět sady, na kterých závisejí jiné sady, a zahrnout je do sekcí `AUTHORITY` nebo `ADDITIONAL` příchozí zprávy v tutéž nebo předchozí úrovni. Aplikace s predikcí počítá tím způsobem, že nejprve prochází sekce `ADDITINAL`, `AUTHORITY` a `ANSWER` v tomto pořadí a poté se rozhoduje o vyvolání vnořeného dotazu.

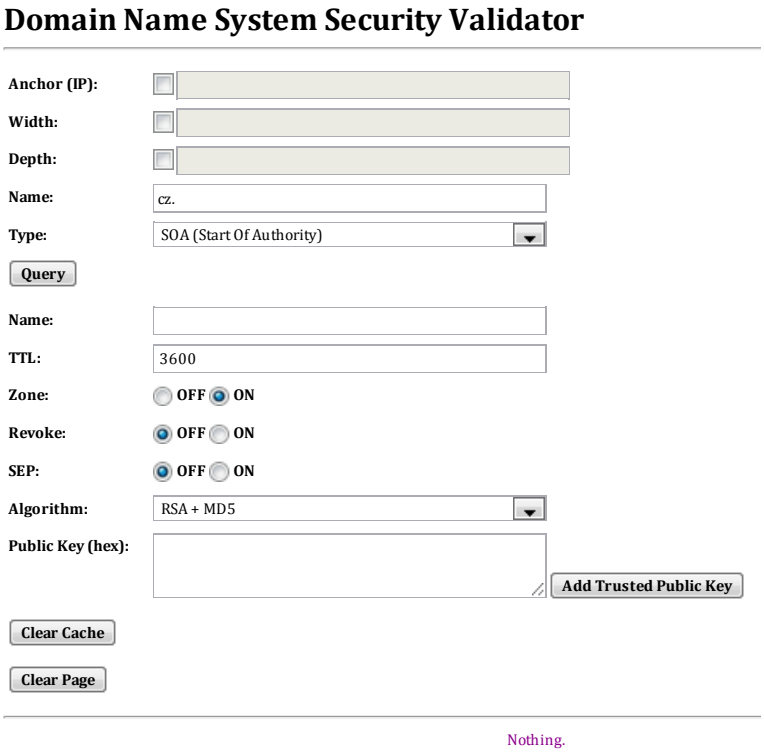
Velikost serverového stromu lze určit hloubkou a šířkou. **Hloubka** stromu je maximání počet úrovní. **Šířka** stromu je maximální počet serverů v jedné úrovni. Aplikace s hloubkou a šířkou počítá, avšak kvůli jednoduchosti jsou společné jak pro dotaz, tak i pro vnořený dotaz. Proto je při volbě hloubky opatrnost na snadě.

Kořenový klíčový veřejný klíč se pomocí záznamu typu `DS` ověřit nedá, neboť vyšší úroveň již neexistuje. Aplikace s kořenovým klíčovým veřejným klíčem počítá tím způsobem, že obsahuje formulář pro jeho vložení do vyrovnávací paměti.

7.7 Grafické rozhraní pro vstup

Hlavními důvody pro volbu jazyka XHTML je náklonnost k jazyku XML a držení kroku v technologiích podle odstavce (4.2.2) o XHTML. A důvodem pro volbu jazyka CSS je sepa-

rovaný kód podle odstavce (4.2.3) o CSS. Pro zadávání hodnot postačily klasické formuláře. Grafické rozhraní pro vstup je znázorněno na obrázku (7.2).



Obrázek 7.2: Grafické rozhraní aplikace `validator` pro vstup

Horní formulář je určen pro dotazování. Některé prvky formuláře korespondují s poli dotazu. Počáteční server lze ovlivnit kolonkou **Anchor (IP)**, která může obsahovat jeho IP adresu. Pokud není vyplněna, použijí se kořenové servery. Maximální počet serverů ve stejné úrovni lze ovlivnit kolonkou **Width**. Pokud není vyplněna, počet je neomezený. Maximální počet úrovní lze ovlivnit kolonkou **Depth**. Pokud není vyplněna, počet je neomezený. Kolonka **Name** koresponduje s polem `QNAME` v dotazu a menu **Type** koresponduje s polem `QTYPE` v dotazu. Tlačítkem **Query** počne dotazování.

Další formulář je určen pro přidání důvěryhodného veřejného klíče do vyrovnávací paměti. Prvky formuláře korespondují s poli záznamu typu `DNSKEY`. V kolonce **Name** je však tolerována tečka na jeho konci. V kolonce **Public Key (hex)** musí být hexadecimální formát, jsou však tolerovány písmena různých velikostí a bílé znaky. Veškeré hodnoty lze zkopírovat z předchozího výsledku. Tlačítko **Add Trusted Public Key** přidá důvěryhodný veřejný klíč do vyrovnávací paměti.

Tlačítko **Clear Cache** vyprázdní vyrovnávací paměť a tlačítko **Clear Page** vymaže výsledek. Zbývající část dokumentu je určena pro výsledek.

7.7.1 Načítání

Aplikace má poněkud specifické chování než běžná webová aplikace. Dotazování může zabrat i několik minut v závislosti na zadané šířce a hloubce stromu, počtu vnořených dotazů, odezvě serverů, rychlosti linek, počtu přesměrování a dalších faktorech. Jak plyne z odstavce (4.3.4) o webových aplikacích, tak aplikace musí být jak konečného, tak i přijatelného trvání. Proto webové servery dovolují aplikacím běžet pouze po určený časový limit. Proto

se doporučuje dotazy sahající do větších hloubek rozdělit do dotazů sahajících do hloubek menších. Nicméně, po takto dlouhou dobu by se zobrazovala pouze prázdná stránka, což může v uživateli vyvolat dojem, že aplikace nevyhodnotila dotaz nebo přestala reagovat. Z tohoto důvodu se sáhlo po technologii AJAX, které je věnován odstavec (4.6) o AJAXu.

Technologie byla aplikována na formulář pro dotazování. Během dotazování se u tlačítka zobrazuje pohyblivý ukazatel indikující dotazování. Umístění pohyblivého ukazatele je záměrné, neboť je v obzoru uživatele, když sleduje tlačítko při kliknutí. Poté se výsledek zobrazí v místě určeném pro výsledek. Při této příležitosti byla technologie aplikována i na ostatní formuláře. To mimojiné přináší i výhodu v ponechání vyplněných hodnot v prvcích formulářů.

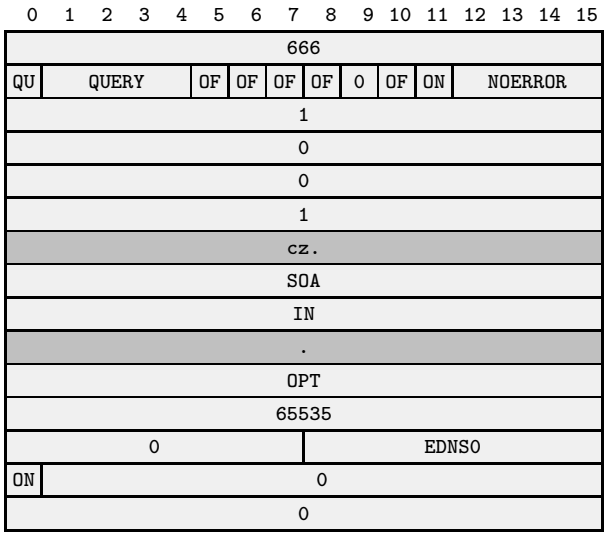
Nicméně, vyskytl se problém jiný. Odstraňování kódu z XHTML elementu může zabrat i několik desítek sekund v závislosti na množství kódu v jazyce XHTML a použitém webovém prohlížeči. To se týká zejména místa určeném pro výsledek. Tento problém je způsoben webovým prohlížečem a není v rámci aplikace řešen. Dočasným rešením může být znovunačtení stránky, avšak se ztrátou vyplněných hodnot v prvcích formulářů.

7.8 Proces

Celý proces zahrnuje inicializování aplikace, načítání vstupu, vytvoření odchozí zprávy, hledání sady a řešení dotazu, analyzování příchozí zprávy, ověřování příchozí zprávy a zobrazování výstupu. Aplikace při spuštění kontroluje přítomnost potřebných modulů, tříd, fukncí a knihoven.

7.8.1 Odchozí zpráva

S asistencí knihovny `protocol` se vytvoří odchozí zpráva. Vyplněnou odchozí zprávu znázorňuje obrázek (7.3).



Obrázek 7.3: Odchozí zpráva

Odchozí zpráva se vyplní podle odstavce (2.10) o zprávě. Odchozí zpráva obsahuje hlavičku, dotaz a v sekci `ADDITIONAL` jeden záznam typu `OPT`.

Hlavička se vyplní podle odstavce (3.6) o hlavičce. Příznak `QR` je nastaven jako `QUERY`, neboť se jedná o zprávu odchozí. Příznak `CD` je nastaven jako `ON`, neboť není žádoucí,

aby server ověřoval, když má ověřovat aplikace. Pole `QDCOUNT` je nastaveno jako `1`, neboť sekce `QUESTION` sestává z jednoho dotazu. Pole `ARCOUNT` je nastaveno jako `1`, neboť sekce `ADDITIONAL` sestává z jednoho záznamu typu `OPT`.

Dotaz se vyplní podle odstavce (2.8) o dotazu. Pole `QNAME` se vyplní hodnotou z kolonky **Name**, pole `QTYPE` se vyplní hodnotou z pole **Type** z formuláře pro dotazování a pole `QCLASS` se vyplní hodnotou `IN`.

Záznam typu `OPT` se vyplní podle odstavce (3.7.1) o záznamu typu `OPT`. Pole `Size` je nastaveno na `65535`, neboť aplikace podporuje maximální velikost zprávy. Příznak `DO` je nastaven jako `ON`, neboť aplikace zajisté podporuje rozšíření DNSSEC. Odchozí zpráva se předá instanci třídy `message` a zakóduje podle odstavce (6.4.5) o třídě `message`.

7.8.2 Hledání sady

Hledání sady je znázorněno v následujícím zkráceném pseudokódu.

```
function find ( NAME, TYPE, CLASS, ... )                                    // najít sadu
  set = load ( NAME, TYPE, CLASS, ... ) ;                                   // pokusit se načíst sadu
  if not set                                                  // není-li sada nalezena, řešit dotaz
    resolve ( NAME, TYPE, CLASS ) ;
  set = load ( NAME, TYPE, CLASS, ... ) ;                                   // pokusit se načíst sadu znovu
  show ( set ) ;                                                            // zobrazit sadu
```

Nejprve dojde k pokusu o načtení sady z vyrovnávací paměti. Pokud se sadu načíst nepodaří, ať už z jakýchkoli příčin, je vyvoláno řešení dotazu. Řešení dotazu by mělo sadu dodat do vyrovnávací paměti. Poté dojde k dalšímu pokusu o načtení sady z vyrovnávací paměti. Pokud se sadu načíst nepodaří ani nyní, sada neexistuje. Řešení dotazu je znázorněno v následujícím zkráceném pseudokódu.

```
function resolve ( QNAME, QTYPE = "SOA", QCLASS = "IN" )                    // vyřešit dotaz
  outgoing_message = create_message ( QNAME, QTYPE, QCLASS ) ;              // vytvořit odchozí zprávu
  if global[anchors] == NULL                                                // není-li zadán server
    anchors = get_roots ( ) ;                                               // použít kořenové servery
  else
    anchors = global[anchors] ;                                             // jinak, použít zadaný server
  destinations = anchors ;
  i = 0 ;
  while ( true )                                                            // procházet úrovně
    if i == global[depth]                                                   // je-li úroveň poslední, skončit
      break ;
    next_destinations = FALSE ;
    for j from 0 to count ( destinations )                                  // procházet servery v úrovni
      destination = destinations[j] ;
      if j == global[width]                                                 // je-li server v úrovni poslední, skončit
        break ;
      send ( destination, outgoing_message ) ;                              // odeslat odchozí zprávu
      incoming_message = receive ( destination ) ;                          // přijmout příchozí zprávu
      separate ( incoming_message ) ;                                       // separovat příchozí zprávu
      validate( incoming_message, QNAME, QTYPE ) ;                          // ověřit příchozí zprávu
      show ( incoming_message ) ;                                           // zobrazit příchozí zprávu
      next_destinations[] = trace ( incoming_message[AUTHORITY] ) ;
                                                // získat servery v další úrovni ze sekce AUTHORITY
    destinations = next_destinations ;
    i ++ ;
```

Proměnná `global[anchors]` se vyplní hodnotou z kolonky **Anchor (IP)**, proměnná `global[width]` se vyplní hodnotou z kolonky **Width** a proměnná `global[depth]` se vyplní z kolonky **Depth** z formuláře pro dotazování.

7.8.3 Analyzování příchozí zprávy

S asistencí knihovny `protocol` se příchozí zpráva přijme a dekóduje. V každé sekci jsou záznamy separovány do shluků, kde **shluk** se skládá ze sady a jejíž odpovídajících záznamů typu `RRSIG`. Záznam typu `OPT` nepodstupuje další procedury. V tomto okamžiku jsou záznamy převedeny na řádky.

7.8.4 Ověřování příchozí zprávy

Ověřování příchozí zprávy je znázorněno v následujícím zkráceném pseudokódu.

```
function validate ( message, NAME, TYPE )                                    // ověřit zprávu
  validate_section ( message[ADDITIONAL], NAME, TYPE ) ;                   // ověřit sekci ADDITIONAL
  validate_section ( message[AUTHORITY], NAME, TYPE ) ;                     // ověřit sekci AUTHORITY
  validate_section ( message[ANSWER], NAME, TYPE ) ;                           // ověřit sekci ANSWER
```

Jelikož mohou být sady v sekci `AUTHORITY` závislé na sadách v sekci `ADDITIONAL`, tak se ověří sekce `ADDITIONAL` jako první; a jelikož mohou být sady v sekci `ANSWER` závislé na sadách v sekcích `AUTHORITY` a `ADDITIONAL`, tak se ověří sekce `AUTHORITY` jako druhá. Ověřování sekce je znázorněno v následujícím zkráceném pseudokódu.

```
function validate_section ( section, NAME, TYPE )                                // ověřit sekci
  validate_DNSKEY_KSK ( section ) ;                          // ověřit pravost sad typu DNSKEY KSK
  validate_set ( section ) ;                               // ověřit pravost sad a záznamů typu RRSIG
  validate_NSEC ( section, NAME, TYPE ) ;                              // ověřit absenci sad typu NSEC
```

Jelikož může být záznam typu `RRSIG` závislý na záznamu typu `DNSKEY` KSK, tak se ověří záznam typu `DNSKEY` KSK dříve. Absence sady typu `NSEC` se ověří po ověření pravosti této sady, neboť podvržená absence by mohla vyřadit ověřování pravosti. Ověřování pravosti záznamů typu `DNSKEY` KSK je znázorněno v následujícím zkráceném pseudokódu.

```
function validate_DNSKEY_KSK ( section )                          // ověřit pravost sad typu DNSKEY KSK
  for i from 0 to count ( section )                                                 // procházet shluky
    set = section[i][set] ;
    if not set[0] is_instance DNSKEY
      break ;
    for j from 0 to count ( set )                                      // procházet záznamy typu DNSKEY
      DNSKEY = set[j] ;
      if not DNSKEY[KSK]                                  // není-li záznam typu DNSKEY KSK, přeskočit
        continue ;
      if DNSKEY[NAME] == ""                                        // je-li záznam typu DNSKEY kořenový
        trusted_DNSKEYs = get_trusted_keys ( ) ;
        valid = FALSE ;
        for k from 0 to count ( trusted_DNSKEYs )             // procházet ověřené záznamy typu DNSKEY
          trusted_DNSKEY = trusted_DNSKEYs[k] ;
          if DNSKEY[NAME]       == trusted_DNSKEY[NAME]
          && DNSKEY[Flags][SEP] == trusted_DNSKEY[Flags][SEP]
          && DNSKEY[Public_Key] == trusted_DNSKEY[Public_Key]
               // je-li alespoň jeden ověřený záznam typu DNSKEY stejný, je záznam typu DNSKEY platný
            valid = TRUE ;
      else if not DNSKEY[NAME] == ""                               // není-li záznam typu DNSKEY kořenový
        DNSKEY[Key_Tag] = wire_key_tag ( DNSKEY ) ;
                                                  // dopočítat pole Key-Tag do záznamu typu DNSKEY
        DSs = find ( DNSKEY[NAME], "DS", "IN", DNSKEY[Algorithm], DNSKEY[Key_Tag] ) ;
                                       // hledat záznamy typu DS korespondující k záznamu typu DNSKEY
        valid = FALSE ;
        for k from 0 to count ( DSs )                                        // procházet záznamy typu DS
          DS = DSs[k] ;
          if not DS[valid]                                  // není-li záznam typu DS platný, přeskočit
            continue ;
```

```
30          reconstructed_digest = wire_digest ( DS[Digest_Type], DNSKEY ) ;
31                                                  // zrekonstruovat pole Digest ze záznamu typu DNSKEY
32          if reconstructed_digest == DS[Digest]
33                                  // je-li alespoň jedno pole Digest stejné, je záznam typu DNSKEY platný
34            valid = TRUE ;
35      DNSKEY[valid] = valid ;
36      DNSKEY->save ( ) ;                              // uložit záznam typu DNSKEY do vyrovnávací paměti
```

Algoritmus sleduje odstavec (3.2.2) o ověřování pravosti záznamu. Procházejí se záznamy typu `DNSKEY` a vybírá se pouze záznam typu `DNSKEY` KSK. Pokud je záznam typu `DNSKEY` kořenový, tak se jeho pravost ověří tím, že se porovná s **platným** kořenovým záznamem typu `DNSKEY`. Hledají se **platné** kořenové záznamy typu `DNSKEY` se stejnými poli v lokální databázi. Pokud se najde alespoň jeden kořenový záznam typu `DNSKEY`, tak je záznam typu `DNSKEY` též **platný**.

Pokud záznam typu `DNSKEY` není kořenový, tak se jeho pravost ověří **platným** záznamem typu `DS`. Hledají se **platné** záznamy typu `DS` se stejnými poli `NAME`, `Algorithm` a `Key-Tag` podle odstavce (3.7.4) o záznamu typu `DS`. Pokud se najde alespoň jeden záznam typu `DS`, jehož pole `Digest` se shoduje s otiskem záznamu typu `DNSKEY`, tak je záznam typu `DNSKEY` též **platný**. Ověřování pravosti sady a záznamů typu `RRSIG` je znázorněno v následujícím zkráceném pseudokódu.

```
1  function validate_set ( section )                          // ověřit pravost sad a záznamů typu RRSIG
2    for i from 0 to count ( section )                                          // procházet shluky
3      set    = section[i][set] ;
4      RRSIGs = section[i][RRSIGs] ;
5      valid = FALSE ;
6      for j from 0 to count ( RRSIGs )                          // procházet záznamy typu RRSIG
7        RRSIG = RRSIGs[j] ;
8        DNSKEYs = find ( RRSIG[Signers_Name], "DNSKEY", "IN", RRSIG[Algorithm], RRSIG[Key_Tag] ) ;
9                                    // hledat záznamy typu DNSKEY korespondující k záznamu typu RRSIG
10       for k from 0 to count ( DNSKEYs )                        // procházet záznamy typu DNSKEY
11         DNSKEY = DNSKEYs[k] ;
12         if not DNSKEY[valid]                          // není-li záznam typu DNSKEY platný, přeskočit
13           continue ;
14         deciphered_plaintext = unwire_plaintext ( decipher_signature ( RRSIG[Signature], DNSKEY ) ) ;
15                                     // dešifrovat a dekódovat pole Signature ze záznamu typu RRSIG
16         if not deciphered_plaintext                       // není-li pole Signature platné, přeskočit
17           continue ;
18         reconstructed_plaintext = make_plaintext ( set, RRSIG ) ;
19                                                  // zrekonstruovat vstup pro podpisový algoritmus
20         if deciphered_plaintext[Hash] == reconstructed_plaintext[Hash]
21           valid = TRUE ;             // je-li alespoň jeden otisk stejný, je záznam typu RRSIG platný
22       if not validate_time ( RRSIG[Signature_Expiration], RRSIG[Signature_Inception] )
23         continue ;                                  // není-li záznam typu RRSIG časově platný, přeskočit
24       RRSIG[valid] = valid ;
25       RRSIG->save ( ) ;                                                  // uložit záznam typu RRSIG
26     for j from 0 to count ( set )                                        // procházet záznamy v sadě
27       record = set[j] ;
28       record[valid] = valid ;       // je-li alespoň jeden záznam typu RRSIG platný, je záznam platný
29       record->save ( ) ;                                                   // uložit záznam ze sady
```

Algoritmus sleduje odstavec (3.2.2) o ověřování pravosti záznamu. Procházejí se záznamy typu `RRSIG`. Pravost záznamu `RRSIG` se ověří **platným** záznamem typu `DNSKEY`. Hledají se **platné** záznamy typu `DNSKEY` se stejnými poli `Signature-Name`, `Algorithm` a `Key-Tag` podle odstavce (3.7.2) o záznamu typu `RRSIG`. Pokud se najde alespoň jeden záznam typu `DNSKEY`, při kterém se pole `Hash` shoduje s otiskem sady, tak je záznam typu `RRSIG` též **platný**. Živost záznamu typu `RRSIG` se ověří po ověření pravosti tohoto záznamu, neboť podvržená živost by mohla vyřadit ověřování pravosti.

Pravost sady se ověří **platným** záznamem typu `RRSIG`. Pokud se najde alespoň jeden **platný** záznam typu `RRSIG`, tak je sada též **platná**. Ověřování absence sady typu `NSEC` je znázorněno v následujícím zkráceném pseudokódu.

```
function validate_NSEC ( section, NAME, TYPE )                    // ověřit absenci sad typu NSEC
  for i from 0 to count ( section )                                        // procházet shluky
    set = section[i][set] ;
    if not set[0] is_instance NSEC
      break ;
    valid = FALSE ;
    for j from 0 to count ( set )                                  // procházet záznamy typu NSEC
      NSEC = set[j] ;
      if NSEC[NAME] == NAME                        // je-li záznam typu NSEC korespondující se jménem
        if not TYPE in NSEC[Type-Bit-Maps]              // a není-li typ obsažen v poli Type-Bit-Maps
          valid = TRUE ;                                            // je záznam typu NSEC platný
      else if not NSEC[NAME] == NAME             // není-li záznam typu NSEC korespondující se jménem
        if NSEC[NAME] < NAME and NAME < NSEC[Next_Domain_Name]
                                                 // a je-li jméno mezi poli NAME a Next_Domain_Name
          valid = TRUE ;                                            // je záznam typu NSEC platný
      NSEC[valid] = valid ;
      NSEC->save ( ) ;                                                 // uložit záznam typu NSEC
```

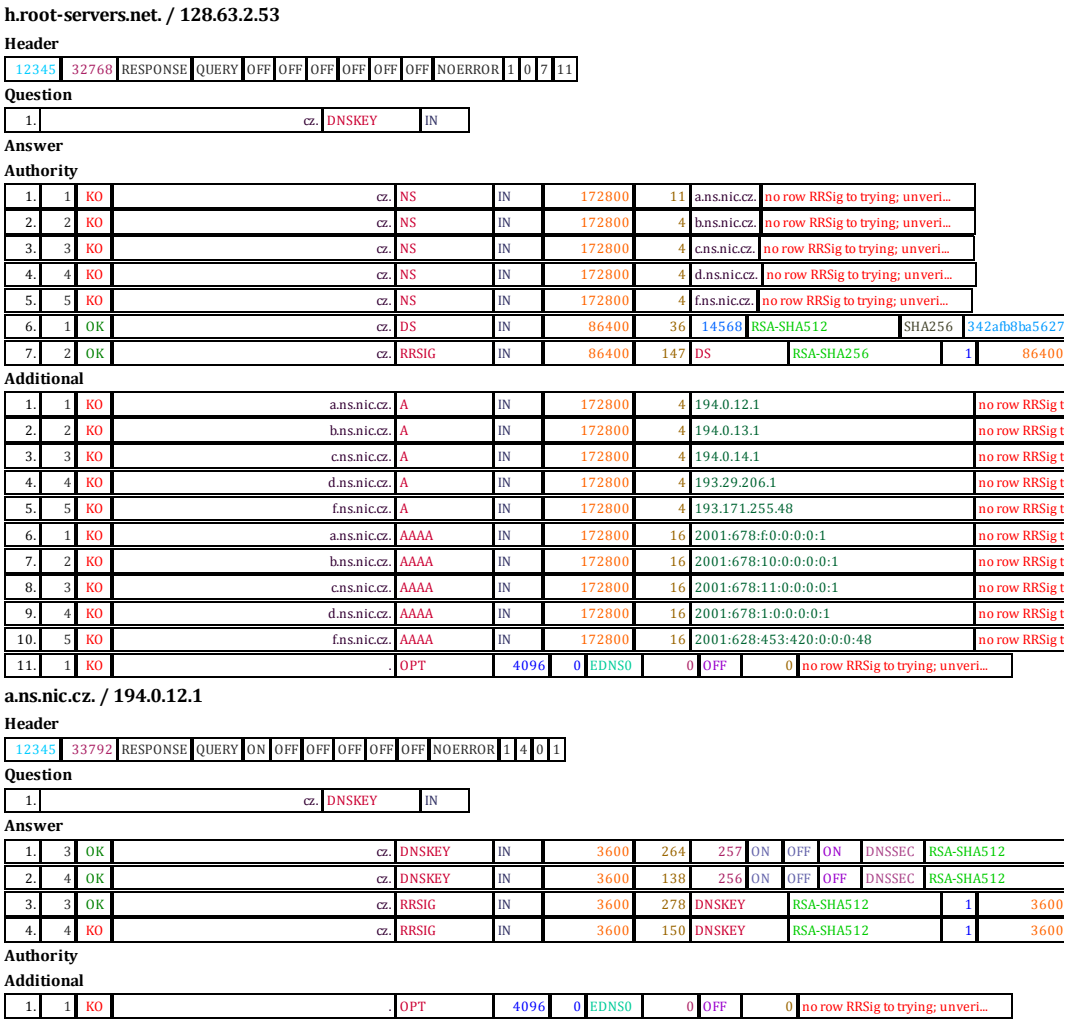
Algoritmus sleduje odstavec (3.2.3) o ověřování absence záznamu. Procházejí se záznamy typu `NSEC`. Pokud je pole `NAME` stejné jako hledané jméno a v poli `Type-Bit-Maps` není obsažen hledaný typ, tak je záznam typu `NSEC` platný.

Pokud není pole `NAME` stejné jako hledané jméno a hledané jméno se nachází mezi poli `NAME` a `Next-Domain-Name`, tak je záznam typu `NSEC` platný. Ověřování absence sady typu `NSEC3` je znázorněno v následujícím zkráceném pseudokódu.

```
function validate_NSEC3 ( section, NAME, TYPE )                  // ověřit absenci sad typu NSEC3
  for i from 0 to count ( section )                                        // procházet shluky
    set = section[i][set] ;
    if not set[0] is_instance NSEC3
      break ;
    valid = FALSE ;
    for j from 0 to count ( set )                                 // procházet záznamy typu NSEC3
      NSEC3 = set[j] ;
      reconstructed_NAME = make_digest ( NAME ) ;
      if NSEC[NAME] == reconstructed_NAME        // je-li záznam typu NSEC3 korespondující se jménem
        if not TYPE in NSEC[Type-Bit-Maps]              // a není-li typ obsažen v poli Type-Bit-Maps
          valid = TRUE ;                                           // je záznam typu NSEC3 platný
      else if not NSEC[NAME] == reconstructed_NAME
                                               // není-li záznam typu NSEC3 korespondující se jménem
        if ! reconstructed_NAME == NSEC[Next-Hashed-Owner-Name]
                                                 // a je-li jméno mezi poli NAME a Next_Domain_Name
          valid = TRUE ;                                           // je záznam typu NSEC3 platný
      NSEC[valid] = valid ;
      NSEC->save ( ) ;                                                // uložit záznam typu NSEC3
```

7.9 Grafické rozhraní pro výstup

Vzhledem k tabulkovém charakteru zobrazovaných dat je použit systém tabulek, a to tabulek složených z plovoucích objektů. Tabulky složené z plovoucích objektů jsou moderní alternativou pro klasické tabulky. Hlavním důvodem pro volbu jazyka CSS je právě podpora plovoucích objektů podle odstavce (4.2.3) o CSS. Grafické rozhraní pro výstup je znázorněno na obrázku (7.4).



Obrázek 7.4: Grafické rozhraní aplikace `validator` pro výstup

Hlavička a každá sekce v příchozí zprávě je promítnuta do tabulky. Téměř každý typ buňky je odlišen barvou, ale v případě potřeby je typ buňky zobrazen v bublině při najetí myši. Buňky nabývající pojmenovaných nebo číselných hodnot mají pevnou velikost. Vyjímkou je buňka `NAME` kvůli jednotnému zarovnání zbylých buněk řádku. Je vidět, že se řádek zleva skládá z buňky s indexem v rámci sekce, buňky typu `Id`, buňky typu `valid` a zprava buňky typu `verifying`, přesně podle odstavce (7.3) o řádku. U buňky typu `valid` je hodnota `TRUE` převedena na hodnotu `OK` a hodnota `FALSE` na hodnotu `KO`.

7.9.1 Interaktivita

U příležitosti použitého jazyka JavaScript kvůli technologii AJAX proběhla v tomto jazyce dodatečná vylepšení v interaktivitě. Jazyku JavaScript je věnován odstavec (4.5). Obsah buňky, jehož důležitost není vysoká a velikost není přiměřená, čímž by rušil design dokumentu, je zkrácen a zobrazen s výpustkou „. . . “ na jeho konci. To se týká zejména buněk s obsahem v hexadecimálním formátu. V případě potřeby lze mezi krátkou a dlouhou variantou obsahu buňky přepínat klinutím myši na buňku.

Nicméně, vyskytl se problém podobný tomu předchozímu při načítání. I modifikace kódu

v XHTML elementu může zabrat i několik desítek vteřin v závislosti na množství kódu v jazyce XHTML a použitém webovém prohlížeči. Tento problém je způsoben webovým prohlížečem a není v rámci aplikace řešen.

8 Závěr

Byla napsána knihovna, která implementuje platné standardy v oblasti systému doménových jmen, v oblasti rozšíření DNSSEC a dalších. Knihovna sestává z mnoha tříd. Z tříd pro navázání spojení se serverem; pro vytváření, kontrolování, kódování a dekódování zpráv, hlaviček, dotazů, záznamů a polí; pro kódování, dekódování, značkování a hashování vstupů do algoritmů; a pro dešifrování podpisů. Záznamy jsou nejpoužívanějších typů zavedených v systému doménových jmen a všech typů zavedených v rozšíření DNSSEC.

S asistencí této knihovny byla napsána aplikace, která implementuje nástroj pro ověřování rozšíření DNSSEC. Aplikace sestává z tříd pro ukládání a načítání záznamů a sad; z funkcí pro ověřování záznamů; a pro zobrazování zpráv.

8.1 Získaný přínos

Největším přínosem bylo osobní překonání složitosti rozšíření DNSSEC a pochopení jeho hlavních principů. Sekundárním přínosem bylo překonání představivosti víceúrovňové asymetrické kryptografie. Všiml jsem si tvrdých zásahů do filosofie systému doménových jmen, které jsou zavedeny jeho rozšířeními. Jsou to mechanismy, které degradují původní systém kvůli zachování zpětné kompatibility. Zachovávání kompatibility obecně brzdí veškerý vývoj ve všech oblastech. Myslím, že pouhé uvádění verze protokolu ve zprávě by vedlo k odstranění zpětné kompatibility protokolu. Například, klient by se dotázal seznamem verzí, které podporuje a server by odpověděl nejvyšší verzí, kterou podporují oba.

Naučil jsem se efektivně manipulovat s binárním protokolem, tedy s daty na nejnižší úrovni, pomocí bitových operátorů. Nevýhody webové aplikace nebyly zdaleka vyrovnány výhodami jazyka PHP. Mezi nevýhody webové aplikace patřila zejména nemožnost interakce mezi uživatelem a aplikací; a velmi obtížné ladění. Mezi výhody jazyka PHP patřilo zejména určování datového typu proměnné po přiřazení hodnoty; a volnost v určování datového typu indexu, klíče a prvku pole. Pole umožňovaly vytvářet libovolné struktury. Implementace v jazyce s otočenými vlastnostmi by byla výrazně náročnější, ale v kombinaci s terminálovou nebo desktopovou aplikací prospěšnější. Webová aplikace dává velmi silnou představu k vytvoření totožné terminálové nebo desktopové aplikace.

8.2 Další vývoj

Dalším směrem vývoje by mohlo být autonomní získávání kořenového klíčového veřejného klíče. Pro komplexnější použití bych volil aplikaci spíše terminálovou nebo desktopovou. Aplikace by poskytovala lepší přehled a uživatel by mohl krokovat dotazování, ve kterém by mohl vybírat konkrétní záznamy, konkrétní servery a konkrétní úrovně.

Literatura

[1] Anne van KESTEREN: *XMLHttpRequest*. World Wide Web Consortium, 2010-08-03, citováno 2011-05-17.
URL <http://www.w3.org/TR/2010/CR-XMLHttpRequest-20100803/>

[2] ARENDS, R.; AUSTEIN, R.; LARSON, M.; aj.: *RFC 4033: DNS Security Introduction and Requirements*. Network Working Group, 2005-03-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc4033.pdf>

[3] ARENDS, R.; AUSTEIN, R.; LARSON, M.; aj.: *RFC 4034: Resource Records for the DNS Security Extensions*. Network Working Group, 2005-03-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc4034.pdf>

[4] ARENDS, R.; AUSTEIN, R.; LARSON, M.; aj.: *RFC 4035: Protocol Modifications for the DNS Security Extensions*. Network Working Group, 2005-03-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc4035.pdf>

[5] Bert BOS and Hakon Wium LIE and Chris LILLEY and Ian JACOBS: *Cascading Style Sheets, level 2: CSS2 Specification*. World Wide Web Consortium, 2008-04-11, citováno 2011-05-17.
URL <http://www.w3.org/TR/2008/REC-CSS2-20080411/>

[6] Bert BOS and Tantek ÇELIK and Ian HICKSON and Hakon Wium LIE: *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. World Wide Web Consortium, 2011-04-12, citováno 2011-05-17.
URL <http://www.w3.org/TR/2011/PR-CSS2-20110412/css2.pdf>

[7] CONRAD, D.: *RFC 3225: Indicating Resolver Support of DNSSEC*. Network Working Group, 2001-12-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc3225.pdf>

[8] Dave RAGGETT: *HTML 3.2 Reference Specification*. World Wide Web Consortium, 1997-01-14, citováno 2011-05-17.
URL <http://www.w3.org/TR/REC-html32>

[9] Dave RAGGETT and Arnaud Le HORS and Ian JACOBS: *HTML 4.0 Specification*. World Wide Web Consortium, 1997-12-18, citováno 2011-05-17.
URL <http://www.w3.org/TR/REC-html40-971218/>

[10] Dave RAGGETT and Arnaud Le HORS and Ian JACOBS: *HTML 4.01 Specification*. World Wide Web Consortium, 1999-12-24, citováno 2011-05-17.
URL <http://www.w3.org/TR/html401/html40.pdf.gz>

[11] EASTLAKE, D.: *RFC 2525: Domain Name System Security Extensions*. Network Working Group, 1999-03-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc2535.pdf>

[12] EASTLAKE, D.: *RFC 2537: RSA/MD5 KEYs AND SIGs in the Domain Name System*. Network Working Group, 1999-03-01, citováno 2011-04-29.
URL <http://tools.ietf.org/pdf/rfc2537.pdf>

[13] EASTLAKE, D. E.; KAUFMAN, C. W.: *RFC 2065: Domain Name System Security Extensions*. Network Working Group, 1997-01-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc2065.pdf>

[14] EASTLAKE 3rd, D.: *RFC 3110: RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System*. Network Working Group, 2001-05-01, citováno 2011-04-29.
URL <http://tools.ietf.org/pdf/rfc3110.pdf>

[15] Hakon Wium LIE and Bert BOS: *Cascading Style Sheets, level 1*. World Wide Web Consortium, 2008-04-11, citováno 2011-05-17.
URL <http://www.w3.org/TR/2008/REC-CSS1-20080411/>

[16] Internet Assigned Numbers Authority: DNSKEY FLAGS [online]. 2007-09-05, citováno 2011-04-28.
URL <http://www.iana.org/assignments/dnskey-flags/>

[17] Internet Assigned Numbers Authority: Domain Name System Security NextSECure3 Parameters [online]. 2008-03-05, citováno 2011-05-20.
URL <http://www.iana.org/assignments/dnssec-nsec3-parameters/>

[18] Internet Assigned Numbers Authority: Delegation Signer Resource Record Type Digest Algorithms [online]. 2010-07-12, citováno 2011-04-28.
URL <http://www.iana.org/assignments/ds-rr-types/>

[19] Internet Assigned Numbers Authority: Domain Name System Parameters [online]. 2011-04-07, citováno 2011-04-28.
URL <http://www.iana.org/assignments/dns-parameters/>

[20] Internet Assigned Numbers Authority: Domain Name System Security Algorithm Numbers [online]. 2011-04-20, citováno 2011-04-28.
URL <http://www.iana.org/assignments/dns-sec-alg-numbers/>

[21] Internet Corporation for Assigned Names and Numbers: Root Name Servers [online]. 2010-07-15, citováno 2010-12-30.
URL <http://www.root-servers.org>

[22] J. LINN: *RFC1421: Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*. Network Working Group, 1993-02-01, citováno 2011-05-12.
URL <http://tools.ietf.org/pdf/rfc1421.pdf>

[23] JANSEN, J.: *RFC 5702: Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC*. Network Working Group, 2009-10-01, citováno 2011-04-29.
URL <http://tools.ietf.org/pdf/rfc5702.pdf>

[24] KAMINSKY, D.: Black Ops 2008: It's The End Of The Cache As We Know It [online]. 2008, citováno 2010-12-29.
URL <http://s3.amazonaws.com/dmk/DMK_BO2K8.ppt>

[25] Kolektiv autorů: *HTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. World Wide Web Consortium, 2000-01-26, citováno 2011-05-17.
URL <http://www.w3.org/TR/2002/REC-xhtml1-20020801/xhtml1.pdf>

[26] LAURIE, B.; SISSON, G.; ARENDS, R.; aj.: *RFC 5155: DNS Security Hashed Authenticated Denial of Existence*. Network Working Group, 2008-02-01, citováno 2011-04-28.
URL <http://tools.ietf.org/pdf/rfc5155.pdf>

[27] MOCKAPETRIS, P. V.: *RFC 882: Domain Names - Concepts and Facilities*. Network Working Group, 1983-11-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc882.pdf>

[28] MOCKAPETRIS, P. V.: *RFC 883: Domain Names - Implementation and Specification*. Network Working Group, 1983-11-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc883.pdf>

[29] MOCKAPETRIS, P. V.: *RFC 1034: Domain Names - Concepts and Facilities*. Network Working Group, 1987-11-01, citováno 2010-12-20.
URL <http://tools.ietf.org/pdf/rfc1034.pdf>

[30] MOCKAPETRIS, P. V.: *RFC 1035: Domain Names - Implementation and Specification*. Network Working Group, 1987-11-01, citováno 2010-12-20.
URL <http://tools.ietf.org/pdf/rfc1035.pdf>

[31] R. FIELDING and J. GETTYS and J. MOGUL and H. FRYSTYK and L. MASINTER and T. BERNERS-LEE: *Hypertext Transfer Protocol – HTTP/1.1*. Network Working Group, 1999-06-01, citováno 2011-05-17.
URL <http://tools.ietf.org/pdf/rfc2616.pdf>

[32] R. FIELDING and J. GETTYS and J. MOGUL and H. FRYSTYK and T. BERNERS-LEE: *Hypertext Transfer Protocol – HTTP/1.1*. Network Working Group, 1997-01-01, citováno 2011-05-17.
URL <http://tools.ietf.org/pdf/rfc2068.pdf>

[33] RIVEST, R.: *RFC 1321: The MD5 Message-Digest Algorithm*. Network Working Group, 1992-04-01, citováno 2011-04-30.
URL <http://tools.ietf.org/pdf/rfc1321.pdf>

[34] RSA Laboratories: *PKCS #1 v2.1: RSA Cryptography Standard*. RSA Laboratories, 2002-June-14, citováno 2011-05-22.
URL <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf>

[35] T. BERNERS-LEE and D. CONNOLLY: *Hypertext Markup Language - 2.0*. Network Working Group, 1995-11-01, citováno 2011-05-17.
URL <http://tools.ietf.org/pdf/rfc1866.pdf>

[36] T. BERNERS-LEE and R. FIELDING and H. FRYSTYK: *Hypertext Transfer Protocol – HTTP/1.0*. Network Working Group, 1996-05-01, citováno 2011-05-17.
URL <http://tools.ietf.org/pdf/rfc1945.pdf>

[37] Telecommunication Standardization Sector of ITU: *X.680: Abstract Syntax Notation One (ASN.1): Specification of basic notation*. International Telecommunication Union, 2002-07, citováno 2011-05-12.
URL
<http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf>

[38] The PHP Group: History of PHP [online]. 2011-05-13, citováno 2011-05-17.
URL <http://cz.php.net/manual/en/history.php.php>

[39] The PHP Group: OpenSSL Installation [online]. 2011-05-13, citováno 2011-05-17.
URL <http://php.net/manual/en/openssl.installation.php>

[40] The PHP Group: OpenSSL Key/Certificate parameters [online]. 2011-05-13, citováno 2011-05-11.
URL <http://php.net/manual/en/openssl.certparams.php>

[41] THOMSON, S.; HUITEMA, C.; KSINANT, V.; aj.: *RFC 3596: DNS Extensions to Support IP Version 6*. Network Working Group, 2003-10-01, citováno 2011-05-23.
URL <http://tools.ietf.org/pdf/rfc3596.pdf>

[42] U.S. Department of Commerce: *Database Language SQL*. National Institute of Standards and Technology, 1993-06-02, citováno 2011-05-17.
URL <http://www.itl.nist.gov/fipspubs/fip127-2.htm>

[43] U.S. Department of Commerce: *FIPS PUB 180-1: Secure Hash Standard*. National Institute of Standards and Technology, 1995-04-17, citováno 2011-04-30.

[44] U.S. Department of Commerce: *FIPS PUB 180-3: Secure Hash Standard*. National Institute of Standards and Technology, 2008-10-01, citováno 2011-04-30.
URL
<http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>

[45] VIXIE, P.: *RFC 2671: Extension Mechanisms for DNS*. Network Working Group, 1999-08-01, citováno 2010-12-27.
URL <http://tools.ietf.org/pdf/rfc2671.pdf>

A Obsah přenosného média

```
CD
+--> /DNSSEC
   +--> /report
   |  +--> /report_source.tar.gz
   |  +--> /report_print.tar.gz
   +--> /literature
   |  +--> /literature.tar.gz
   +--> /implementation
      +--> /protocol_source.tar.gz
      +--> /validator_source.tar.gz
```