Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Zapracování digitálního podpisu pro autentizaci přístupu Oracle Portal

Plzeň, 2003 Josef Steinberger

Abstrakt

Securing Access Authentication to Oracle Portal with Digital Signatures

The goal of the diploma work is to increase security of access authentication into Oracle Portal with digital signatures. User will need, except username and password, his digital certificate for accessing the Oracle Portal. This work solves also authentication to PL/SQL applications, which is demonstrated on a small demo-application. Topics discussed in this work are: encrypting, message digests, digital signatures, certificate authorities, cerificates and their publishing, user authentication, SSL (Secure Socket Layer), Oracle 9i Aplication Server, publishing certificates with OpenSSL, configuring Oracle HTTP Server, securing user authentication in PL/SQL applications and Oracle Portal.

An exteding theme of this diploma work is email signing and signature verifying with language Java.

Obsah

Abstrakt **2**
Obsah **3**
Prohlášení **7**
1. Úvod **8**
2. Digitální podpis **10**
2.1. Úvod 10
2.1.1. Úloha digitálního podpisu 10
2.1.2. Hlavní cíle a zásady bezpečnosti elektronické komunikace 11
2.2. Kryptografie 12
2.3. Šifrování a dešifrování 12
2.3.1. Součásti šifrování 13
2.3.2. Síla šifry 14
2.4. Kryptografické algoritmy 15
2.4.1.1. Přehled systémů s privátním klíčem 16
2.4.2. Přehled systémů s veřejným klíčem 17
2.4.3. RSA a kryptografie s veřejným klíčem 18
2.4.3.1. Jak RSA funguje 19
2.4.3.2. Příklad algoritmu RSA 20
2.4.3.3. Síla algoritmu RSA 22
2.5. Výtahy zpráv a digitální podpisy 23
2.5.1. Výtah zprávy 24
2.5.2. Digitální podpisy 25
2.5.3. Algoritmy pro generování výtahu zprávy 26
3. Certifikáty **29**
3.1. Certifikační autorita a certifikáty 29
3.2. Tvorba a životnost certifikátů 30
3.3. Standardní formáty souborů certifikátů 31
4. Analýza problému autentizace **34**
4.1. Autentizace 34
4.2. Základní autentizace – uživatelské jméno + heslo 34
4.2.1. Volba hesla 35
4.2.2. Hesla na více počítačích 36

4.2.3. Jednorázová hesla ... 36
4.2.4. Shrnutí ... 36
4.3. Autentizace uživatele v internetovém prostředí ... 37
4.3.1. Vyloučení možnosti odposlechu ... 37
4.4. Analýza využití digitálního podpisu pro autentizaci přístupu ... 37
4.5. Secure Socket Layer (SSL) ... 39
4.5.1. Základ SSL ... 39
4.5.2. SSL proces ... 40
4.5.2.1. Fáze 1: „Klientské ahoj“ (Client Hallo) ... 41
4.5.2.2. Fáze 2: „Serverové ahoj“ (Server Hallo) ... 42
4.5.2.3. Fáze 3: „Hlavní klíč klienta“ (Client Master Key) ... 42
4.5.2.4. Fáze 4: „Klient skončil“ (Client Finish) ... 43
4.5.2.5. Fáze 5: „Ověření serveru“ (Server Verify) ... 43
4.5.2.6. Fáze 6: „Požadavek certifikátu“ (Request Certificate) ... 44
4.5.2.7. Fáze 7: „Certifikát klienta“ (Client Certificate) ... 44
4.5.2.8. Fáze 8: „Ověření certifikátu klienta“ (Client Cerificate Verification) ... 45
4.5.2.9. Fáze 9: „Server skončil“ (Server Finish) ... 45
4.6. Závěr ... 46
5. Aplikační server Oracle 9iAS ... 47
5.1. Služby Oracle 9iAS ... 47
5.1.1. Komunikační služby ... 47
5.1.2. Prezentační služby ... 48
5.1.3. Aplikační logika ... 49
5.1.4. Služby pro zvýšení výkonnosti ... 50
5.1.5. Systémové služby ... 50
5.2. Bezpečnostní architektura Oracle 9iAS ... 51
5.2.1. Elementy architektury bezpečnosti AS ... 51
6. Tvorba certifikátů ... 54
6.1. OpenSSL ... 54
6.1.1. Vytvoření certifikační autority ... 55
6.1.2. Vytvoření certifikátu serveru ... 57
6.1.3. Vytvoření certifikátu uživatele ... 58
6.2. Vydávání certifikátů ... 58
7. Konfigurace SSL v Oracle HTTP serveru ... 60

7.1. Zprovoznění certifikátu serveru 60
7.2. Konfigurace SSL v souboru httpd.conf 60
7.3. Instalace klientského certifikátu v prohlížeči 62
8. Zabezpečení autentizace uživatele 63
8.1. Základní autentizace 63
8.1.1. Základní autentizace v PL/SQL aplikacích 63
8.1.2. Základní autentizace v Oracle Portal 63
8.2. Aplikační rozhraní zabezpečení autentizace certifikáty 64
8.2.1. Vytvoření uživatele CERT_AUTH_ADMIN 64
8.2.2. Tabulka uživatelů a jejich certifikátů 65
8.2.3. PL/SQL balík CERT_AUTH_API 66
8.2.4. Konfigurace CERT_AUTH_API 69
8.3. Demonstrační aplikace autentizace certifikáty v PL/SQL 71
8.3.1. Konfigurační tabulka aplikace 71
8.3.2. PL/SQL balík CERT_AUTH_EXAMPLE 72
8.3.3. Konfigurace demonstrační aplikace 73
8.3.4. Vytvoření uživatele 73
8.3.5. Registrace certifikátu uživatele 74
8.3.6. Testování zabezpečení autentizace aplikace 75
8.4. Zabezpečení autentizace certifikáty v Oracle Portal 78
8.4.1. Single Sign-On 78
8.4.1.1. Komponenty Single Sign-On 78
8.4.1.2. Single Sign-On autentizační metody 79
8.4.1.3. Autentizační proces Oracle Portal 80
8.4.2. Způsob zabezpečení přístupu do Oracle Portal 81
8.4.3. Konfigurace Oracle Portal pro použití HTTPS 81
8.4.4. Konfigurace externí autentizace Login serveru 82
8.4.5. Tabulka WWSSO_USERCERT_T 83
8.4.6. Přidání kontroly certifikátu do balíku WWSSO_AUTH_EXTERNAL 84
8.4.7. Testování zabezpečení autentizace portálu 84
9. Podepisování a verifikace emailů 89
9.1. Specifikace S/MIME 89
9.1.1. S/MIME formáty pro podepsané zprávy 90
9.1.2. Podepisování s formátem multipart/signed 90

9.1.3. Příklad multipart/signed zprávy 90
9.2. Práce s emaily v jazyce JAVA 91
9.3. Odeslání emailu 91
9.3.1. Protokol SMTP 91
9.3.2. Demonstrační program odeslání podepsaného emailu 92
9.4. Přijetí emailu 95
9.4.1. Protokoly pro příjem emailů 95
9.4.2. Demonstrační program přijetí a verifikace podpisu emailu 95
10. Závěr 97
Přehled použitého značení 99
Přehled zkratek 100
Literatura 102
Příloha A: Parametry systému 103
Příloha B: Listing cert_auth_api 104
Příloha C: Listing cert_auth_example 108
Příloha D: Listing SendSignedEmail.java 116
Příloha E: Listing ReceiveSignedEmail.java 120
Příloha F: Zákon o elektronickém podpisu 125

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května ……

Josef Steinberger

1. Úvod

Organizace, které používají internet pro obchodní účely nebo provozování dalších neveřejných webových aplikací, potřebují kontrolovat přístup k jejich stránkách. Dnes je autentizace uživatele ve většině webových systémů založena na kontrole uživatelského jména a hesla. Cílem této diplomové práce je zvýšení bezpečnosti procesu autentizace přístupu Oracle Portal zapracováním digitálního podpisu. V rámci aplikace nebude stačit k přihlášení pouze uživatelské jméno a heslo, ale bude také vyžadován jeho digitální podpis.

Digitální podpis je velmi složitý, zašifrovaný číselný kód, který je pro každého uživatele jedinečný obdobně jako otisk prstu, a který je právně ověřitelný. Proces podepisování musí být v souladu se Zákonem o elektronickém podpisu.

Kapitola 2 rozebírá problematiku kryptografie, šifrování, podepisování a následného ověřování podpisů. Třetí kapitola pojednává o certifikačních autoritách, cerifikátech a jejich vydávání. Další část popisuje problém autentizace z hlediska bezpečnosti. Jsou zde prodiskutovány možnosti zabezpečení procesu autentizace digitálním podpisem. Podrobně je zde popsána vrstva SSL (Secure Socket Layer). V páté kapitole je představen aplikační server Oracle 9i. Další kapitola popisuje tvorbu vlastní certifikační autority a cerifikátů. Tyto certifikáty slouží pouze pro demonstraci v rámci této diplomové práce. Pro možnost právní odpovědnosti v souladu se Zákonem o elektronickém podpisu by musely být vydávány kvalifikované certifikáty od akreditovaných certifikačních autorit (např. I. CA). Sedmá část se zabývá nastavením serveru pro použití SSL a certifikátů.

Kapitola 8 tvoří vlastní jádro diplomové práce. Je zde popsáno vytvořené PL/SQL aplikační rozhraní pro kontrolu cerifikátů předtím prošlých vrstvou SSL. Toto API umožňuje zabezpečení autentizace PL/SQL aplikací i Oracle Portal. V rámci diplomové práce byla vytvořena v PL/SQL demonstrační aplikace se zabezpečenou autentizací uživatele certifikáty. V této části je také popsán postup jak nakonfigurovat portál (a změnit jeho autentizační proceduru) k použití SSL + API pro kontrolu certifikátů.

Poslední kapitola představuje rozšíření diplomové práce. Je zde popsáno podepisování a odesílání emailů a následné přijímání a verifikace podpisů v emailech. Programy v této části jsou napsány v jazyce Java. V databázi Oracle (verzi 8.1.7), ve které bylo programové vybavení této diplomové práce vytvořeno, je možné uložit třídy napsané v Javě přímo do databáze. To umožňuje využít funkce napsané v jazyce Java pro generování podepsaných emailů z databáze.

V příloze F lze nalézt kompletní znění Zákona o elektronickém podpisu, který definuje správný proces použití elektronických podpisů, vydávání kvalifikovaných certifikátů, získávání akreditace poskytovatelů certifikačních služeb atd. z právního hlediska.

2. Digitální podpis

2.1. Úvod

2.1.1. Úloha digitálního podpisu

Společným jmenovatelem, který umožňuje dálkový přenos dokumentů a informací je digitalizace, tedy proces, který z klasických "papírových" dokumentů vytvoří dokumenty digitální, dokumenty přeměněné do "řeči" čísel. Informační dálnice, např. městská počítačová nebo globální síť sítí - Internet, dopraví digitalizované dokumenty bezpečně na místo určení, na příslušnou digitální adresu.

Dokumenty, které se touto formou přenášejí mají většinou informativní, osobní či služební charakter. V případě právních dokumentů procházejí elektronickou poštou často návrhy smluv, které účastníci právního řízení vzájemně dolaďují a připravují jejich konečnou verzi. Připravené dokumenty pak ve svém konečném stádiu realizace opouštějí kybernetický prostor, aby na sebe vzaly klasickou papírovou podobu. Ta totiž jako jediná umožní provést poslední akt, kterým je právoplatný podpis dokumentu. A jsme u jádra věci. Dokumenty zatím nedokážeme ošetřit tak, aby je bylo možné právoplatně podepsat a stvrdit svoji právní odpovědnost za obsah dokumentu.

Kvalitativně vyšší úroveň práce s elektronickými dokumenty zajistí poslední zdánlivá maličkost – právně ověřitelný a právně správný **digitální** nebo chcete-li **elektronický podpis**.

Současný stav podepisování elektronických dokumentů či pošty je důvěrně znám. Dopisy včetně těch obchodních a služebních podepisujeme formou textového řetězce, sady znaků, kterou zapíšeme svoje jméno. Dalším způsobem je vložit do dopisů obrázek obsahující skenovaný podpis odesilatele. Ani ten však právní problém neřeší. Takový podpis lze velmi snadno kopírovat a vložit do dokumentu zcela jiného. Samozřejmě, že hlavička elektronické pošty obsahuje informace, ze kterých lze vystopovat odesilatele, ale po stránce právní jsou i takto zdánlivě "ručně" podepsané dopisy naprosto bezcenné. Navíc dáváme všanc grafickou podobu svého podpisu, které může být zneužito mnoha způsoby.

Problém vyřeší až nasazení digitálních podpisů. Digitální podpis je velmi složitý, zašifrovaný číselný kód, který je pro každého uživatele ojedinělý obdobně jako otisk prstu, a který je

právně ověřitelný. Podstata digitálního podpisu spočívá v "označkování" elektronického dokumentu, ze kterého je zřejmá nezpochybnitelná identita autora. K podepisování dokumentu slouží privátní, tajné klíče. Ke čtení dokumentu a ověření podpisů slouží veřejné klíče.

Vidina digitálního podpisu je docela příjemná a lákavá. V rozvinuté informační společnosti lze digitální podpis uplatnit např. v bankách, u lékaře a v mnoha dalších případech. Výhody digitálního podpisu ocení především Ti, kterým se ne vždy a napoprvé podaří vykouzlit stejnou grafickou podobu podpisu. U digitálního podpisu tento problém odpadá, navíc ověření podpisu je nesrovnatelně jednodušší a rychlejší.

2.1.2. Hlavní cíle a zásady bezpečnosti elektronické komunikace

Obvykle si lidé slučují pojem bezpečnost se šifrováním, resp. s nerozluštitelností důvěrných elektronických dat. Ve skutečnosti je problém poněkud, ale ne příliš, složitější. Zpravidla cíle zabezpečení dat při jejich výměně a použití rozdělujeme na tři zásady:

- Zásada důvěrnosti
- Zásada neodmítnutelnosti odpovědnosti
- Zásada integrity

Zásada důvěrnosti vyjadřuje potřebu uložit data tak, aby jejich obsah mohl přečíst jen ten, komu jsou určena, přičemž kdokoliv další nemá šanci obsah rozluštit ani za pomoci nejmodernějších technologií.

Zásada neodmítnutelnosti odpovědnosti vyjadřuje neméně důležitou potřebu možnosti dokázat, kdo je autorem zprávy. Zde nejde o utajení, ale naopak o průkaznost původu dat. Požadavek neodmítnutelnosti odpovědnosti bývá často v praxi splněn digitálním, elektronickým podpisem.

Zásada integrity má na starosti, aby data došla nejen úplná, ale též prokazatelně nezměněná.

Pokud zprávy budou všechny tři zásady respektovat, pak je jejich bezpečnost stoprocentně zaručena.

2.2. Kryptografie

Kryptografie je věda, zabývající se šifrováním – tedy utajením informací[1]. V počítačovém prostředí slouží kryptografie k ochraně dat před neautorizovaným odhalením, může sloužit k autentizaci uživatele nebo procesu, požadujícího nějakou službu, může zabránit neautorizovaným modifikacím.

Kryptografie představuje nezanedbatelnou součást moderní počítačové bezpečnosti.

2.3. Šifrování a dešifrování

Šifrování je proces, při němž se zpráva (nešifrovaný text) transformuje na jinou zprávu (zašifrovaný text) pomocí matematické funkce[2] a speciálního šifrovacího hesla, takzvaného ***klíče***.

Dešifrování je opačný proces: zašifrovaný text se pomocí matematické funkce a klíče převede zpět na text nešifrovaný.

Šifrování může hrát významnou úlohu při každodenní komunikaci a práci s počítačem:

- Pomocí šifrování můžeme chránit informace uložené na našem počítači před neautorizovaným přístupem – a to dokonce i před lidmi, kteří jinak mají k našemu počítačovému systému přístup.
- Šifrováním můžeme chránit informace při přenosu z jednoho počítače na druhý.
- Šifrováním můžeme zabránit či detekovat náhodné nebo úmyslné změny dat.
- Pomocí šifrování je možno ověřit, zda autorem dokumentu je opravdu ten, kdo myslíme.

I přes tyto výhody má šifrování i určitá omezení:

---

[1] ***Kryptoanalýza*** je související disciplína, zabývající se luštěním šifer. ***Kryptologie*** kombinuje dohromady kryptografii a kryptoanalýzu.

[2] I když ne vždy musí být funkce takto vyjádřena.

- Šifrováním nemůžeme útočníkovi zabránit v úplném vymazání našich dat.
- Útočník může porušit samotný šifrovací program. Může jej modifikovat tak, že bude používat jiný klíč než zadáme, nebo může šifrovací klíče někam zaznamenávat pro pozdější použití.
- Útočník může objevit dříve neznámý relativně snadný způsob dekódování zpráv, zašifrovaných naším algoritmem.
- Útočník může získat soubor před nebo po jeho zašifrování nebo dešifrování.

Pro všechny tyto důvody je třeba šifrování chápat pouze jako součást celkové bezpečnosti našeho systému, ne jako náhradu za ostatní metody, například za správné řízení přístupu.

2.3.1. Součásti šifrování

Existuje celá řada způsobů, jak můžeme počítač použít k zašifrování nebo dešifrování dat. Bez ohledu na algoritmus však všechny šifrovací systémy používají shodné základní prvky:

Šifrovací algoritmus

Šifrovací algoritmus je funkce, obvykle sestavená na nějakém matematickém základě, která provádí samotné šifrování a dešifrování dat.

Šifrovací klíč

Šifrovací klíč říká šifrovacímu algoritmu, jak má data šifrovat nebo dešifrovat. Klíče se podobají počítačovým heslům: jakmile informaci zašifrujete, musíte k jejímu dešifrování zadat správný klíč. Na rozdíl od kontroly hesel však šifrovací program neporovnává klíč s dříve zadaným klíčem a nepovoluje přístup tehdy, pokud se klíče shodují. Namísto toho šifrovací algoritmus přímo používá klíč při transformaci zašifrovaného textu zpět do nezašifrované podoby. Zadáte-li správný klíč, dostanete zpět původní zprávu. Budete-li data šifrovat špatným klíčem, dostanete nesmysly[3].

[3] Předpokládejme pochopitelně, že původní zpráva nesmyslná nebyla. Pokud by byla, pak po odšifrování bysme pochopitelně získali opět nesmysly.

Délka klíče

Stejně jako hesla, i klíče mají nějakou předem určenou délku. Delší klíče jsou bezpečnější než kratší klíče, protože při použití útoku hrubou silou[4] skýtají více kombinací. Různé šifrovací systémy umožňují použití klíčů různých délek, některé dovolují použití klíčů s proměnnou délkou.

Nešifrovaný text

Informace, které chceme zašifrovat.

Zašifrovaný text

Informace po zašifrování.

2.3.2. Síla šifry

Různé kryptografické metody nejsou ekvivalentní. Některé systémy se dají velmi snadno obejít či ***prolomit***. Jiné daleko lépe vzdorují i mnohem systematičtějšímu útoku. Schopnost kryptografického systému chránit informace před útokem se označuje jako jeho ***síla***. Síla systému závisí na mnoha okolnostech, například:

- Utajení klíče.
- Obtížnost uhodnutí klíče nebo vyzkoušení všech možných klíčů (tzv. hledání klíče). Obecně platí, že čím delší klíč, tím hůře se dá uhodnout či najít.
- Obtížnost otočení šifrovacího algoritmu bez znalosti klíče (prolomení šifrovacího algoritmu).
- Existence (či neexistence) zadních vrátek, tedy metody, jak je možno data relativně snadno dešifrovat i bez znalosti klíče.
- Možnost dešifrovat celý text v případě, že z části znáte jeho nezašifrovanou podobu (tzv. útok se znalostí textu).
- Vlastnosti nezašifrovaného textu, které jsou známy útočníkovi. Například některé systémy se dají snadno prolomit, pokud všechny jimi šifrované zprávy začínají nebo končí stejným kusem textu[5].

[4] Termínem ***útok hrubou silou*** (*brute force attack*) se označuje útok metodou „pokus-omyl“, kdy se při zjišťování klíče zkoušejí postupně všechny možné hodnoty.
[5] Této pravidelnosti využili Spojenci při rozluštění šifer stroje Enigma, používaného Němci za 2. světové války.

Cílem kryptografického návrhu je vyvinout algoritmus, který se dá bez klíče prolomit tak těžko, že to bude zhruba odpovídat námaze nutné ke zjištění klíče prohledáváním celého prostoru klíče. Tato schopnost by měla zůstat zachována i v případě, že útočním má nějaké informace o zašifrované zprávě. Na návrzích šifer obvykle spolupracují špičkoví matematici.

2.4. Kryptografické algoritmy

V současné době se používají dva hlavní typy šifrovacích algoritmů

- ***Algoritmy s privátní klíčem***, v nichž se pro zašifrování i dešifrování zprávy používá stejný klíč. Těmto algoritmům se rovněž někdy říká šifry se symetrickým klíčem. Symetrické šifry jsou historicky nejstarší. Jejich hlavní nevýhoda spočívá v tom, že dojde-li k prozrazení klíče, pak nelze jednoznačně určit, došlo-li k úniku na straně autora nebo příjemce. Kryptografie s privátním klíčem se obvykle používá k ochraně informací uložených na disku počítače nebo k zašifrování dat při přenosu mezi dvěma systémy.
- ***Algoritmy s veřejným klíčem***, v nichž se pro zašifrování zprávy používá tzv. ***veřejný klíč***, pro dešifrování zprávy slouží ***privátní klíč***. Termín veřejný klíč vychází ze skutečnosti, že šifrovací klíč můžete klidně zveřejnit, aniž by se porušila bezpečnost zprávy nebo dešifrovacího klíče. Systémy s veřejným klíčem se někdy také označují jako kryptografie s asymetrickým klíčem. Kouzlo asymetrických šifer spočívá v tom, že to, co bylo zašifrováno jedním z klíčů privátní/veřejný, lze rozšifrovat právě a pouze druhým klíčem z dané dvojice. Tedy zašifruji-li něco svým klíčem privátním, rozšifruje to někdo pouze, má-li můj klíč veřejný. A obráceně: zašifruji-li něco vaším klíčem veřejným, pak to můžete rozšifrovat jen vaším klíčem privátním. U asymetrické kryptografie je při prozrazení vždy jasné odkud byl klíč prozrazen. Privátní a veřejný klíč tvoří vždy nerozlučnou dvojici. Logicky patří totiž jeden k druhému. Dokonce je známo, že jeden lze vypočítat jednoznačně z hodnoty druhého. Problém je ale v tom, že zatímco veřejný je možno z privátního vypočítat snadno a rychle, obráceně je to prakticky nemožné. Nikoliv ale proto, že by nebylo známo jak, ale proto, že výpočet by byl natolik náročný na kapacitu počítače, že výsledek by nebylo možné získat ani v horizontu let. Privátní klíč si každý musí chránit jako oko v hlavě, ale klíč veřejný by měl naopak zveřejnit. Zveřejnění je vhodné, aby každý

mohl rozluštit soubory zašifrované privátním klíčem a zjistit tak, kdo je opravdu autorem. Zašifrování dokumentu privátním klíčem a následné dešifrování klíčem veřejným za účelem ověření, kdo svým privátním klíčem dokument zašifroval, se nazývá metodou ***elektronického podpisu***.

Kromě toho ještě existuje třetí typ šifrovacího systému, a to

- ***Hybridní veřejné/privátní kryptosystémy***. V těchto systémech se pomalejší algoritmy s veřejným klíčem použijí k předání náhodně generovaného klíče sezení, který se pak použije jako základ pro algoritmy s privátním klíčem. (Tento klíč se používá pouze pro jediné sezení a poté se zruší.) Prakticky většina implementací systémů s veřejným klíčem jsou ve skutečnosti hybridní systémy[6].

2.4.1.1. Přehled systémů s privátním klíčem

Následuje přehled dnes běžně používaných šifrovacích systémů s privátním klíčem.

DES, 3DES

Data Encryption Standard (DES) je šifrovací algoritmus, vyvinutý v 70. letech Národním úřadem pro standardy a technologie (dnes *National Institute of Standards and Technology*, NIST) a firmou IBM. DES používá klíč o délce 56 bitů. Bezpečnost algoritmu DES je možno vylepšit provedením více šifrování, takzvaným ***superšifrováním***. Nejběžnější způsoby jsou dvojí šifrování (*Double DES*) a trojí šifrování (*Triple DES*). I když dvojitý DES zdánlivě výrazně zvyšuje bezpečnost, ukázaly se některé možnosti prolomení této metody a proto odborníci doporučují pro náročnější použití trojitý DES.

RC2, RC4

Blokové šifry původně vyvinuté Ronaldem Rivestem a uchovávané jako obchodní tajemství firmy *RSA Data Security*. Algoritmy byly v roce 1996 anonymně popsány na Usenetu a jsou rozumně bezpečné (i když některé klíče mohou být slabé). RC2 a RC4 se prodávají v implementacích s délkou klíče od 1 do 248 bitů.

[6] Například vrstvu SSL lze zařadit mezi hybridní systémy. Viz kapitola 4.5.

RC5

Bloková šifra vyvinutá Ronaldem Rivestem, publikovaná v roce 1994. RC5 umožňuje uživateli definovat délku klíče, délku bloku dat a počet šifrovacích průchodů.

IDEA

International Data Encryption Algorithm (IDEA), vyvinutý Jamesem L. Massey a Xuejia Lai v Curychu, publikovaný v roce 1990. IDEA používá klíč o délce 128 bitů a je považován za poměrně silný. K šifrování souboru a elektronické pošty jej používá populární program PGP. Širšímu použití algoritmu IDEA bohužel brání řada softwarových patentů, které vlastní Ascom-Tech AG v Solothurnu (Švýcarsko).

Skipjack

Algoritmus s klasifikací Tajný, vyvinutý Národní bezpečnostní agenturou (NSA). Chcete-li vidět zdrojový kód algoritmu a jeho popis, musíte mít oprávnění pro nahlížení do přísně tajných materiálů. Používá klíč o délce 80 bitů.

2.4.2. Přehled systémů s veřejným klíčem

Následuje přehled dnes běžně používaných šifrovacích systémů s veřejným klíčem.

Diffie-Hellman

Systém pro výměnu kryptografických klíčů mezi dvěma stranami. Nejedná se vlastně o šifrovací algoritmus, ale o metodu pro vyvinutí a výměnu sdíleného privátního klíče přes veřejné komunikační kanály. V zásadě se obě strany dohodnou na nějaké společné číselné hodnotě a pak vytvoří klíč. Pak si strany vymění použité matematické transformace. Poté každá strana spočítá třetí klíč, který už není možno jednoduše odvodit ani při znalosti předchozí komunikace.

Existuje několik verzí tohoto protokolu pro různý počet komunikujících stran a s různými matematickými transformacemi. Je nutné pečlivě volit čísla a kalkulace, pro některé kombinace je totiž snadné algoritmus prolomit. Diffie-Hellmanův algoritmus se často používá jako základ při výměně kryptografických klíčů při zakódování komunikační linky. Podle implementace může mít klíč jakoukoliv délku. Obecně platí, že delší klíče jsou bezpečnější.

RSA

Známý kryptografický systém s veřejným klíčem, vyvinutý (tehdejšími) profesory MITu Ronaldem Rivestem a Adi Shamirem a profesorem USC Leonardem Adlemanem. RSA je možno použít jednak jako šifrovací algoritmus a také jako základ pro systém digitálních podpisů. Digitální podpisy slouží k ověření pravosti a původnosti digitálních informací. Podle použité implementace může mít klíč libovolnou délku.

ElGamal

Další algoritmus založený na exponenciální a modulární aritmetice. Podobně jako RSA algoritmus se dá použít k šifrování a digitálním podpisům.

DSA

Digital Signature Algorithm, vyvinutý v NSA a převzatý NISTem jako federální standard pro zpracování informací (FIPS). Přestože algoritmus DSA může používat klíče libovolné délky, podle FIPS je možno použít pouze klíče o délce 512 a 1024 bitů. Jak vyplývá z názvu, DSA slouží pouze pro digitální podpisy, dá se však upravit i pro potřeby šifrování. Tento algoritmus se občas označuje také zkratkou DSS, obdobně jako se algoritmus DEA označuje jako DES.

2.4.3. RSA a kryptografie s veřejným klíčem

RSA je nejznámější kryptografický algoritmus s veřejným klíčem. Je pojmenován po svých objevitelích, Ronaldu Rivestovi, Adi Shamirovi a Leonardu Adlemanovi, kteří jej vymysleli v roce 1977. Na rozdíl od algoritmů s privátním klíčem, používá RSA dva kryptografické klíče: veřejný klíč a privátní klíč. Veřejný klíč slouží k zašifrování zprávy, privátní klíč k jejímu dešifrování. Systém může fungovat i naopak, privátním klíčem se data šifrují, veřejným se dešifrují.

2.4.3.1. Jak RSA funguje

Síla algoritmu RSA je založena na obtížnosti ***faktorizace***[7] velmi velkých čísel. Následující popis neobsahuje matematické nuance algoritmu v plné šíři.

RSA je založen, na z teorie čísel dobře známých, vlastnostech modulární aritmetiky celých čísel. Jedním z použitých prvků je Eulerova funkce (n). Tato funkce je definována jako počet nesoudělných čísel, menších než n[8]. Funkce pro prvočíslo je o jednu menší než toto prvočíslo: každé celé kladné číslo menší než naše prvočíslo je s ním nesoudělné.

Vlastnost využívaná algoritmem RSA byla objevena už Eulerem a říká: pro každé celé číslo i nesoudělné s n platí:

$$i^{\Phi(n)} \bmod n \equiv 1$$

Dále předpokládejme náhodná celá čísla e a d, která vyhovují následující kongruenci:

$$ed \equiv 1 \bmod \Phi(n)$$

Další využívanou vlastnost rovněž objevil Euler. Jeho teorém říká, že pro jakékoliv M nesoudělné s n platí:

$$(M^e)^d \bmod n \equiv M \quad and \quad (M^e)^d \bmod n \equiv M$$

Řečeno kryptografickým jazykem, bude-li M část zprávy, pak ji budeme jednoznačně šifrovat následující funkcí:

$$s \equiv M^e \pmod n$$

Dešifrování se provede další funkcí:

$$M \equiv s^d \pmod n$$

[7] Pozn.: rozklad čísla na prvočinitele
[8] Dvě čísla jsou nesoudělná, pokud nemají žádného společného dělitele (vyjma jedničky), nesoudělná jsou například čísla 8 a 9.

Jak nyní zvolit správné hodnoty pro *n*, *e*, *d*? Nejprve pomocí nějaké vhodné metody zvolíme dvě velká prvočísla *p* a *q* o přibližně stejné velikosti. Tato čísla by měla být veliká – řádově stomístná – a je třeba je uchovávat v tajnosti. Dále vypočítáme Eulerovu funkci $\Phi(pq)$. Je-li číslo *n* součinem dvou prvočísel, pak platí:

$$(pq) = (p - 1)(q - 1) = \Phi(n).$$

Dále zvolíme hodnotu *e*, která je nesoudělná s $\Phi(n)$. Vhodná hodnota leží někde v intervalu $\mathbf{max}(p+1, q+1) < e < \Phi(n)$. Pak vypočítáme *d* tak, aby platilo $ed \ \mathbf{mod}\ \Phi(n) \equiv 1$. Jinými slovy hledáme modulo převrácenou hodnotu k $e \ \mathbf{mod}\ \Phi(n)$. Pokud by *d* vyšlo příliš malé (tedy menší než asi $\log_2(n)$), zvolíme jinou dvojici *e* a *d*.

Teď už tedy známe klíče. Při šifrování zprávy *m* postupujeme tak, že ji rozdělíme na stejná[9] celá čísla *M* menší než *n*. Pro každou část zprávy hledáme hodnotu $(M^e) \ \mathbf{mod}\ n = s$. Tento výpočet je možno provést velmi rychle buď hardwarově, nebo softwarově s použitím speciálních algoritmů. Získané hodnoty uspořádáme do formátu zašifrované zprávy. Při dešifrování zprávu rozdělíme do bloků a každý blok dešifrujeme jako $(s^d) \ \mathbf{mod}\ n = M$.

2.4.3.2. Příklad algoritmu RSA

Předpokládejme například, že jsme zvolili prvočísla *p* a *q* takto:

$$p = 251$$
$$q = 269$$

Číslo *n* tedy bude

$$n = 251 * 269 = 67519$$

Hodnota funkce je

$$\Phi(n) = (251 - 1)\,(269 - 1) = 67000$$

[9] Pozn.: „stejná“ ve smyslu „reprezentovaná na stejném počtu bitů“.

Zvolíme si e = 50253, d potom bude

$$d = e^{-1} \textbf{ mod } 67000 = 27917$$

protože platí

$$50253 * 27917 = 1402913000 = 20939 * 67000 + 1 = 1 (\textbf{mod } 67000)$$

Pomocí čísla n = 67519 můžeme kódovat libovolnou zprávu M mezi 0 a 67518. Tímto systémem tedy můžeme kódovat textovou zprávu po dvojicích znaků. (Dva znaky mají 16 bitů, tedy 65536 kombinací.)

Jako klíč použijeme hodnotu e a zakódujeme zprávu „RSA works!“. ASCII hodnoty zprávy „RSA works!“ a její zakódovanou podobu vidíte v následující tabulce:

ASCII	Dekadická hodnota	Šifrovaná hodnota
„RS“	21075	48467
„A “	16672	14579
„wo“	30575	26195
„rk“	29291	58004
„s!“	29473	30141

Jak vidíte, zakódované hodnoty nijak neodpovídají původní zprávě.

Při dešifrování se každá hodnota umocní na d-tou a s výsledkem se provede operace **mod** n. Po převedení zpět do ASCII máme původní text.

Při použití algoritmu RSA ve skutečných aplikacích se pracuje s čísly, která jsou dlouhá řádově stovky míst. Protože početní operace se stoznakovými řetězci jsou časově velmi náročné, jsou moderní aplikace navrženy tak, aby minimalizovaly počet RSA operací, které bude třeba provést. Namísto použití RSA k šifrování celé zprávy se tímto algoritmem šifruje pouze klíč sezení, a teprve tímto se pak šifruje samotná zpráva pomocí nejakého rychlého algoritmu s privátním klíčem, jako jsou třeba DES nebo IDEA.

2.4.3.3. Síla algoritmu RSA

Čísla *n*, *e* a dokonce i *d* mohou být odhalena, aniž by se tím výrazně snížila bezpečnost šifry. Aby mohl útočník zprávu dekódovat, musel by zjistit hodnotu (*n*), což, podle všech současných vědomostí, vyžaduje faktorizovat číslo *n*.

Faktorizace velkých čísel je velmi obtížná – není dosud známa žádná metoda, kterou by bylo možno faktorizaci efektivně provést. Čas potřebný k faktorizaci může být, podle velikosti faktorizovaného čísla, třeba sto let nebo i několik miliard let, a to na těch nejrychlejších počítačích. Pokud je *n* dostatečně velké, je bez ohledu na vynaložené úsilí nefaktorizovatelné. Šifrovací algoritmus RSA je tedy poměrně velmi silný za předpokladu, že byly zvoleny vhodné hodnoty čísel *n*, *e* a *d*.

Abychom si ukázali, jak obtížná je faktorizace velkého čísla, zkusíme odhadnout, jak dlouho by trvala faktorizace dvousetmístného čísla – což je asi o 70 cifer delší, než doposud nejdelší faktorizovatelné číslo.

Všechna dvousetmístná čísla jsou reprezentovatelná na maximálně 655 bitech.

$\lceil 2^x \rceil$ má $\lfloor x \log_{10} 2 \rfloor + 1$ číslic

Faktorizace čísla o délce 655 bitů si, s použitím nejrychlejšího známého algoritmu, vyžádá přibližně $1{,}2 \times 10^{23}$ operací. Předpokládejme nyní, že máme k dispozici počítač který je schopen provést 10 miliard operací za sekundu[10]. K provedení $1{,}2 \times 10^{23}$ operací pak budeme potřebovat $1{,}2 \times 10^{13}$ sekund, neboli 380,267 let počítačového času. Pokud se vám nezdá dost bezpečná šifra, kterou je možno rozluštit za 380 let, pak použijte číslo o čtyřech stech cifer – jeho faktorizace si vyžádá zhruba $8{,}6 \times 10^{15}$ let. To už by mělo být dost – podle Stručné historie času Stephena Hawkinga má samotný vesmír stáří jen asi 2×10^{10} let.

Ukažme si ještě jiný pohled na velikost takovýchto čísel. Předpokládejme, že se vám (nějak) podaří vypočítat rozklady všech dvousetmístných dekadických čísel . Čistě k uložení samotných rozkladů budete potřebovat $(9 \times 10^{200}) \times 655$ bitů paměťového prostoru

[10] Což je o něco více, než umí dnešní nejrychlejší paralelní počítač.

(bez jakékoliv režie nebo indexace). Tyto hodnoty budete ukládat na disk s kapacitou 100 GB (100×1024^4 nebo přibližně $1{,}1 \times 10^{14}$). Pak byste potřebovali $6{,}12 \times 10^{189}$ takovýchto disků. Dále uvažujme, že každý z těchto disků by vážil pouhou miliontinu gramu. Hmotnost celého potřebného diskového pole by byla $6{,}75 \times 10^{177}$ tun. Planeta Země váží pouhých $6{,}588 \times 10^{21}$ tun. *Chandrasekharova mez*, hmotnost, která stačí hvězdě ke kolapsu do černé díry, je asi 1,5 násobek hmotnosti Slunce, tedy $3{,}29 \times 10^{27}$ tun. Takže vaše diskové pole by svou vlastní vahou spolehlivě zkolabovalo na černou díru[11].

Opakuji tedy, že pokud nedojde k nějakým zásadním objevům na poli teorie čísel, je algoritmus RSA prakticky stoprocentně odolný proti útoku hrubou silou[12].

2.5. Výtahy zpráv a digitální podpisy

Výtahy zpráv (*message digest*, nazývaný také kryptografický kontrolní součet nebo kryptografický hash-kód, *cyptographic checksum*, *cryptographic hashcode*) není nic jiného, než číslo – speciální číslo, vytvořené nějakou funkcí, která se jen velmi obtížně invertuje.

Digitální podpis (*digital signature*) je nejčastěji výtah zprávy zašifrovaný něčím privátní klíčem. Tomuto procesu se říká ***podepsání***. Digitální podpis první dvě funkce, obě jsou pro bezpečnost systému důležité:

- ***Integrita*** – digitální podpis indikuje, zda nedošlo k modifikaci souboru nebo zprávy
- ***Autentizace*** – digitální podpis umožňuje matematicky ověřit, kdo zprávu podepsal

Kromě toho může plnit ještě třetí funkci, která může být někdy velmi důležitá – ***nepopiratelnost***. Nepopiratelnost znamená, že jakmile zprávu podepíšete a odešlete, nemůžete nikdy v budoucnu tvrdit, že nejste autorem této zprávy. Nemůžete svou zprávu zapřít, protože byla podepsána vaším privátním klíčem, o němž se předpokládá, že jej vlastníte pouze vy.

[11] A z té už žádnou faktorizaci nevytlučete! ;-)
[12] Pochopitelně za předpokladu, že příslušné konstanty budou správně zvoleny.

2.5.1. Výtah zprávy

Jednoduchá hashovací funkce dostane nějaký vstup, obvykle proměnné délky, a jejím výsledkem je číslo, výrazně menší než vstup. Funkce má charakter zobrazení „mnoha do jednoho“ – totiž větší množství vstupů (pravděpodobně nekonečné množství) produkuje stejný výstup . Funkce je kromě toho ***deterministická***, což znamená, že stejná vstupní hodnota dává vždy stejný výstup. Hashovací funkce se velmi často používají v aplikacích, které potřebují rychle prohledávat velký objem dat – například u tabulek symbolů v překladačích nebo u spellcheckerů.

Výtah zprávy je též hashovací funkce. Má vstup libovolné délky – obvykle celý diskový soubor – a vytváří malou hodnotu (typicky o délce 128 nebo 512 bitů). Ze stejného vstupu vytváří vždy stejný výstup. Protože prostor výstupu je mnohem menší než prostor potencionálního vstupu, musí pro nejméně jednu výstupní hodnotu existovat více vstupních kombinací, které ji vyprodukují. U dobrého hashovacího algoritmu by to mělo platit pro všechny výstupní hodnoty. Kromě toho by měl dobrý algoritmus pro generování výtahu zprávy další dvě důležité podmínky. Algoritmus by neměl být snadno odvoditelný nebo invertovatelný. To znamená, že pokud máme výstupní hodnotu, neměli bychom být schopni jednoduše vymyslet vstupní text, který by takovýto výstup generoval ať už reverzací hashovací funkce nebo vypozorováním nějakých závislostí mezi vstupem a výstupem. Má-li výstup alespoň 128 bitů, celkem nepřichází v úvahu útok hrubou silou, protože bychom v průměru museli vyzkoušet $1{,}7 \times 10^{38}$ možných vstupů stejné délky, než bychom nalezli ten vstup, z něhož mohl vzniknout známý výstup. Když si to porovnáte s odhady v části „Síla algoritmu RSA“, zjistíte, že je to úloha, kterou se současnou technikou nemůže nikdo nikdy zvládnout. Při použití takto obrovských čísel je dokonce velmi nepravděpodobné, že by za celou historii lidstva mohly vzniknout dva *různé* dokumenty, které by produkovaly stejný 128 bitový výtah. Druhá důležitá podmínka dobrého algoritmu je ta, že malá změna ve vstupu bude mít za následek velké změny ve výstupu. Při změně jediného vstupního bitu by se měla změnit asi polovina výstupních bitů. Je to vlastně důsledek první podmínky, protože nechceme, aby se dala odvodit závislost chování vstupu a výstupu. Nicméně jedná se o vlastnost, která je důležitá i sama o sobě.

2.5.2. Digitální podpisy

Jak už jsme zjistili v předchozí části, funkce výtahu zprávy představuje pouze polovinu řešení spolehlivého digitálního podpisu. Druhá polovina spočívá v šifrování s veřejným klíčem – provozovaném ovšem opačným směrem. Připomeňme si, že když jsme dříve v této kapitole hovořili o kryptografii s veřejným klíčem, řekli jsme si, že je založena na dvou klíčích:

- ***veřejný klíč*** – používá se k zašifrování tajné zprávy. Obvykle je tento klíč široce znám,
- ***privátní klíč*** – je držen v tajnosti a slouží k dešifrování přijaté zprávy.

Pomocí trochy matematické gymnastiky se dá celý postup převrátit. Zprávu zašifrujete svým privátním klíčem a kdokoliv ji bude moci dešifrovat veřejným klíčem. Proč by to někdo dělal? Každému veřejnému klíči vyhovuje pouze jediný privátní klíč. Pokud je možno daným veřejným klíčem zprávu dešifrovat, s jistotou to znamená, že byla zašifrována správným privátním klíčem. No a to je princip činnosti digitálních podpisů.

Jakmile použijete na zprávu svůj privátní klíč, podepisujete ji. Když použijete privátní klíč a funkci výtahu zprávy, můžete vypočítat digitální podpis odesílané zprávy. Principiálně můžete použít algoritmus s veřejným klíčem bez algoritmu výtahu zprávy: je možno zašifrovat privátním klíčem celou zprávu. Každý známý algoritmus s veřejným klíčem však i u středně velkých zpráv trvá poměrně dlouho. Pokud byste tedy použili algoritmus s veřejným klíčem na soubor o velikosti několika megabajtů, mohlo by šifrování trvat také několik hodin či dní.

Namísto toho používáme rychlý algoritmus pro vytvoření výtahu zprávy a potom podepisujeme privátním klíčem tuto malou hodnotu. Když příjemce zašifrovanou hodnotu obdrží, může ji dešifrovat veřejným klíčem. Ze vstupního souboru se rovněž snadno vytvoří hashovaná hodnota. Pokud se obě hodnoty shodují, máte jistotu (téměř), že jste obdrželi stejnou zprávu, která byla odesílána.

V současné době se pro vytváření digitálních podpisů nejčastěji používají kombinace algoritmu pro výtah zprávy MD5 a kryptografického mechanismu s veřejným klíčem RSA. Další možnost spočívá v použití algoritmu SHA (Secure Hash Function) a ElGamalova mechanismu veřejného klíče – tyto algoritmy dohromady vytvářejí algoritmus DSA (Digital Signature Algorithm).

2.5.3. Algoritmy pro generování výtahu zprávy

V současnosti je k dispozici řada algoritmů pro generování výtahu zprávy. Všechny fungují v zásadě na stejném principu, liší se však v rychlosti a dalších vlastnostech.

MD2, MD4 a MD5

Jedním z nejrozšířenějších algoritmů pro generování výtahu zprávy je algoritmus MD5 Ronalda Rivesta, distribuovaný společností *RSA Data Security*, který je možno použít volně bez licenčních poplatků. Je založen na algoritmu MD4, který zase pro změnu vychází z algoritmu MD2. Všechny tyto tři algoritmy generují ze vstupního textu libovolné délky výtah o délce 128 bitů. Všechny algoritmy nejprve text rozšíří na fixní délku a poté provedou sérii matematických operací s celým vstupním blokem. Algoritmus MD2 byl navržen Ronaldem Rivestem a publikován jako RFC 1319[13]. Nemá žádné známé slabiny, je však velmi pomalý. Z toho důvodu vyvinul Rivest algoritmus MD4, který byl publikován jako RFC 1186 a 1320. Tento algoritmus byl navržen jako rychlý, kompaktní a optimalizovaný pro procesory s architekturou *„little-endian“*[14]. V kryptografické literatuře se objevily možnosti potencionálních útoků na algoritmus MD4, a proto dr. Rivest vyvinul algoritmus MD5, publikovaný jako RFC 1321. Jedná se o přepracovaný algoritmus MD4, do kterého byla přidána navíc jedna série interních operací a bylo provedeno několik zásadních změn v algoritmu. Kvůli těmto změnám je MD5 o něco pomalejší než MD4. Je ovšem daleko šířeji používán, než algoritmus MD4. Na počátku roku 1996 byly odhaleny zásadní slabiny algoritmu MD4. Z toho důvodu by tento algoritmus neměl být používán.

SHA

Algoritmus *Secure Hash Algorithm* byl vyvinut NISTem ve spolupráci s NSA. Algoritmus vypadá jako blízce příbuzný algoritmu MD4, produkuje však výstup o délce 160 bitů, nikoliv 128 bitů. Analýza algoritmu ukazuje, že některé změny algoritmu MD4 plní podobnou funkci, jako vylepšení v algoritmu MD5 (i když se jedná o zcela jiné zásahy).

[13] Dokumenty RFC představují způsob publikování otevřených standardů. Je možno si je nahrát nebo posílat poštou a popisují většinu běžných protokolů a datových struktur.
[14] Procesory, u nichž se nejméně významný bajt vícebajtového čísla ukládá na nejnižší adrese, více významný bajt na vyšší atd. Jsou to typicky všechny procesory rodiny Intel.

HAVAL

Algoritmus HAVAL je modifikací algoritmu MD5, vyvinuli jej Yuliang Zheng, Josef Pieprzyk a Jennifer Seberry. Dá se modifikovat tak, že vytváří výstup o délce od 92 do 256 bitů. Rovněž se dá volit počet „kol“ (průchodů interním algoritmem). Výsledkem je, že HAVAL může být rychlejší než MD5, i když z toho plyne jisté snížení bezpečnosti výstupu. Na druhé straně může HAVAL vytvářet i delší a potencionálně bezpečnější hashovací kód[15].

Ostatní kódy

Pro úplnost si ještě popíšeme další dva typy „podpisových funkcí“.

Kontrolní součty

Kontrolní součet je funkce, která se vypočítává ze vstupu ke zjištění, zda nedošlo k porušení vstupu. Nejčastěji se kontrolní součty používají ke kontrole, zda data přenášená modemem nebo sítí nebyla poškozena nějakým prohozením bitů nebo šumem. Často se rovněž používají v diskových zařízeních pro kontrolu dat, zapisovaných a čtených z disku: pokud kontrolní součet dat nesouhlasí, vyskytl se zřejmě nějaký problém na disku nebo na pásce.

Kontrolní součet se obvykle počítá jako jednoduchá lineární nebo polynomická funkce vstupu a výsledkem bývá malá hodnota (16 nebo 32 bitů). Často používanou formou kontrolních součtů jsou CRC – cyklické kontrolní redundantní součty. Kontrolní součty se snadno počítají a dají se rovněž snadno ošidit. Je možno modifikovat soubor tak, aby měl stejný kontrolní součet jako před modifikací. Řada „hackerských utilit“, které kolují v hackerském podsvětí, obsahuje nástroje, jež obnovují hodnotu *sum* součtu systémových příkazů po jejich modifikaci. Z toho důvodu by se kontrolní součty **nikdy** neměly používat jako ochrana proti záměrné modifikaci.

Autentizační kódy zpráv

Autentizační kód zprávy (*Message Athentication Code*, MAC) je v zásadě funkce výtahu zprávy doplněná o heslo. Smyslem je, aby hodnota MAC kódu nebyla obnovitelná osobou, která má k dispozici sice stejný vstup, nezná však příslušný tajný

[15] Je nutné si uvědomit, že pouhé prodloužení hešovací hodnoty nemusí nutně zvýšit bezpečnost algoritmu.

klíč (heslo). Tento algoritmus může a nemusí být bezpečnější než prostá funkce výtahu zprávy – záleží na použitém algoritmu, síle klíče a délce MAC kódu.

Jednoduchá metoda MAC algoritmu přidá ke zprávě klíč a poté generuje výtah zprávy. Protože klíč je součástí vstupu, ovlivňuje hodnotu výsledného kódu neobnovitelným způsobem. Protože dva různé klíče generují pro stejná data velmi rozdílné výstupy, plní tato metoda úspěšně funkci heslem řízeného výtahu zprávy.

Druhá používaná metoda vychází z nějaké proudové šifry, jako je třeba RC4 nebo DES. Klíč je v tomto případě šifrovací heslo a MAC kód je poslední blok bitů šifrovacího algoritmu. Protože výstup šifry závisí na všech bitech vstupu a na hesle, bude poslední blok výstupu různý pro různé vstupy i pro různá hesla. Pokud je ale velikost šifrovacího bloku malá (například 64 bitů), může být MAC kód snáze rozluštitelný hrubou silou než podstatně delší výtahy zpráv.

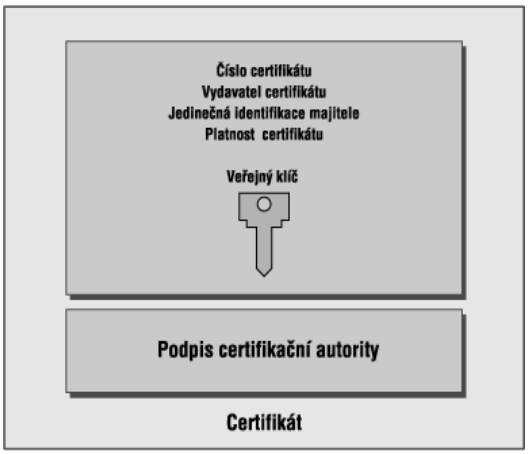
Digitální podpisy s veřejným klíčem mohou být také chápány jako určitá forma MAC kódu, protože závisejí jednak na výtahu zprávy a jednak na tajném klíči. Změna kterékoliv hodnoty má vliv na celkový výsledek funkce.

3. Certifikáty

Problémem asymetrické kryptografie je způsob, jak ověřit pravost zveřejněných veřejných klíčů. K tomu slouží ***digitální či elektronický certifikát***. Digitální certifikát je elektronická obdoba cestovního pasu nebo občanského průkazu. Jedná se v podstatě o uživatelův veřejný klíč plus další údaje popisující držitele certifikátu (jméno, bydliště, fotografie apod.) To vše je zašifrováno (elektronicky podepsáno) privátním klíčem, jehož veřejný klíč je znám a dostupný z nezaměnitelných zdrojů. Pomocí digitálního certifikátu pak lze ochránit nejen elektronickou poštu, ale zajistit bezpečnou komunikaci např. i po Internetu.
Držitelem a vydavatelem privátního klíče je tzv. certifikační autorita (v ČR např. I. CA), tedy instituce nebo útvar, který tyto certifikáty neboli elektronické občanské průkazy vydává. Každý může požádat certifikační autoritu o digitální certifikát.

3.1. Certifikační autorita a certifikáty

Řešením problému správy, distribuce a uchování klíčů je využití služeb ***certifikační autority***. Tyto instituce se podobají státním notářům. certifikační autorita vystupuje při vzájemné komunikaci dvou subjektů jako třetí nezávislý důvěryhodný subjekt, který prostřednictvím jím vydaného certifikátu jednoznačně svazuje identifikaci subjektu s jeho dvojicí klíčů respektive s jeho digitálním podpisem. Certifikáty obsahují ve své nejjednodušší formě veřejný klíč, jméno a další údaje zajišťující nezaměnitelnost subjektů. Běžně používané certifikáty též obsahují datum počátku platnosti, datum ukončení platnosti, jméno certifikační autority, která certifikát vydala, sériové číslo a některé další informace. Certifikační autorita garantuje jedinečnost subjektů podle užité identifikace subjektu. To je zajištěno legislativními a technickými pravidly provozu instituce certifikační autority. Splnění těchto požadavků potvrdí certifikační autorita podepsáním dokumentu svým privátním klíčem a následným vydáním tohoto certifikátu.



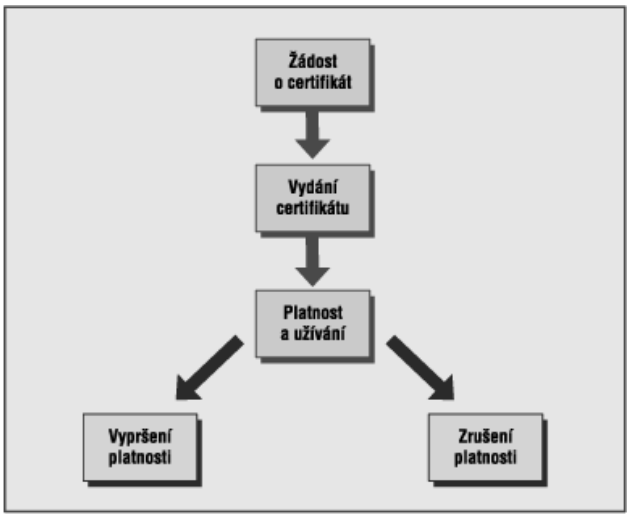
Znamená to, že certifikát je podepsaným dokumentem se všemi důsledky z toho plynoucími, tedy zejména autorizace (certifikační autorita jako garant pravosti dokumentu) a integrita dat (nelze zaměnit klíč nebo identitu klienta). Tím, že certifikační autorita zaručuje správnost jí vydaného certifikátu , odstraňuje nutnost smluvní důvěryhodné výměny klíčů mezi dvěma subjekty navzájem a jejich dohoda spočívá pouze v domluvě o společně uznávané certifikační autoritě. Důležité je, že se utajovaná data na straně klienta redukují pouze na bezpečné uchovávání privátního klíče, protože ostatní je řešeno certifikáty. Ty si můžeme kdykoliv ověřit se znalostí veřejného klíče certifikační autority, repektive jejího certifikátu. Existence certifikační autority také umožňuje důvěryhodnou komunikaci i subjektů, jenž se navzájem fyzicky nikdy nepotkali nebo neabsolvovali složitou proceduru vzájemné důvěryhodné výměny svých klíčů.

3.2. Tvorba a životnost certifikátů

Tvorba certifikátu má 6 kroků:

1. Generování klíčů. Každý potenciální žadatel o certifikát si nejprve sám pomocí dostupného SW vybavení vygeneruje dvojici klíčů pro použití v asymetrické kryptografii.
2. Příprava identifikačních dat. Žadatel o certifikát shromáždí podle požadavků certifikační autority osobní identifikační materiály nutné pro vydání certifikátu, jako IČO, DIČO, resp. číslo OP, rodné číslo a podobně.
3. Předání veřejných klíčů a identifikačních údajů certifikační autoritě. Žadatel předá certifikační autoritě data nutná pro vydání certifikátu spolu s doklady o jejich pravosti.

4. Ověření informací. Certifikační autorita si na příslušných místech ověří, že může vydat žadateli certifikát.
5. Tvorba certifikátu. Certifikační autorita vytvoří digitální dokument příslušného formátu a ten poté podepíše svým privátním klíčem.
6. Předání certifikátu. Podle dohody je certifikát žadateli předán (disketa), zaslán, nebo zveřejněn.



Doba platnosti certifikátů je omezená a je uvedena v každém certifikátu. Tato veličina je velmi důležitá. Pokrok ve zvyšování výkonnosti výpočetní techniky a možnost objevení mezer v protokolech nebo algoritmech by ve velkém časovém horizontu mohl způsobit, že by se certifikáty staly nespolehlivé. Běžné certifikáty jsou proto vydávány s platností 6 měsíců, nejvíce 1 rok. I během této doby je možné zrušit platnost certifikátu. Důvodem pro toto opatření může být například vyzrazení privátního klíče.

3.3. Standardní formáty souborů certifikátů

Do svého prohlížeče můžete importovat a exportovat certifikáty v následujících formátech:

- **Personal Information Exchange (PKCS #12)**

  Personal Information Exchange formát (PFX, také nazývaný PKCS #12) umožňuje přenos certifikátu a jeho odpovídajícího privátního klíče z jednoho počítače na jiný nebo z počítače na vyjímatelné médium.

  PKCS #12 (*Public Key Cryptography Standard #12*) je průmyslový standard, který se hodí pro přenos nebo zálohu a obnovení certifikátu a s ním asociovaného privátního klíče. Přenos může být proveden mezi produkty od různých dodavatelů.

- **Cryptographic Message Syntax Standard (PKCS #7)**

  PKCS #7 umožňuje přenos certifikátu a všech certifikátů v jeho certifikační cestě z jednoho počítače na druhý nebo z počítače na vyjímatelný disk. Soubory PKCS #7 typicky používají příponu `.p7b` a jsou kompatibilní se standardem ITU-T X.509.

- **DER – binárně kódované X.509**

  DER (*Distinguished Encoding Rules*) pro ASN.1, jak je definováno v ITU-T doporučení X.509, je více omezující kódovací standard než alternativní BER (*Basic Encoding Rules*) pro ASN.1, jak je definováno v ITU-T doporučení X.209, na kterém je DER založen. Oba BER i DER poskytují metodu kódování objektů (jako jsou certifikáty a zprávy) pro přenos mezi zařízeními a aplikacemi nezávislou na platformě.

  Při kódování certifikátu většina aplikací užívá DER, protože část certifikátu (info žádosti o certifikát) musí být DER-kódována, aby mohla být podepsána.

  Tento formát může být užíván certifikačními autoritami, které nejsou na serverech s Windows 2000 pro svoji nezávislost na platformě. Soubory s certifikáty DER mají typicky příponu `.der`.

- **Base64 kódované X.509**

  Toto je metoda kódování vyvinutá pro užití s S/MIME (*Secure/Multipurpose Internet Mail Extensions*), což je populární a standardní metoda pro přenos binárních příloh přes internet. ***Base64*** kóduje soubory v ASCII textovém formátu, který snižuje pravděpodobnost znehodnocení souborů, které jsou posílány přes internetové brány,

zatímco S/MIME poskytuje některé kryptografické zabezpečovací služby pro aplikace elektronické pošty včetně nepopiratelnosti původu užitím digitálního podpisu, důvěrnosti a bezpečnosti dat užitím šifrování, autentizace a integrity zprávy.

MIME specifikace (RFC 1341 a následující) definuje mechanismus pro kódování jakýchkoliv binárních informací pro přenos elektronickou poštou.

Protože všechny klientské aplikace podporující MIME umí dekódovat Base64 soubory, tento formát může být užíván certifikačními autoritami, které nejsou na serverech s Windows 2000 pro svoji nezávislost na platformě. Soubory s certifikáty Base64 mají typicky příponu `.cer`.

4. Analýza problému autentizace

4.1. Autentizace

Autentizace je proces, při kterém dochází ke kontrole totožnosti uživatele. Existují tři možné způsoby, jak se můžeme počítačovému systému autentizovat, a při každé autentizaci jeden nebo více z nich používáme:

1. Můžeme počítači říct něco, co víme (například heslo).
2. Můžeme počítači „ukázat“ něco co máme (například identifikační kartu).
3. Necháme počítač, aby si něco našeho zkontroloval (například otisk prstu).

Žádný z těchto systémů není dokonalý. Například tajným odposlechem vaší terminálové linky může někdo získat vaše heslo. Pokud vás přepadnou, mohou vám sebrat vaši identifikační kartu. A pokud bude mít útočník nůž, můžete klidně přijít o prst! Obecně platí, že čím je autentizační metoda spolehlivější, tím obtížněji se používá a narušitel musí být při jejím překonání více agresivní.

4.2. Základní autentizace – uživatelské jméno + heslo

Každý uživatel systému musí mít ***účet***. Účet je identifikován ***uživatelským jménem***. Obvykle má každý účet rovněž tajné ***heslo***, které chrání před neoprávněným přístupem. Uživatelským jménům se občas také říká jména účtů. Abysme se mohli k systému přihlásit musíme znát jak uživatelské jméno, tak i své heslo. Hesla představují nejjednodušší způsob autentizace: je to tajemství, které spolu sdílí uživatel a počítač. Když se přihlašujeme, zadáváme heslo, čímž počítač přesvědčíme, že jsme opravdu tím, za koho se vydáváme. Počítač zkontroluje, zda námi zadané heslo odpovídá zadanému účtu. Pokud se shodují, můžeme začít pracovat v systému. Když heslo zapisujeme, systém jej nezobrazuje. Tím jsme chráněni pro případ, že používáme tiskárnový terminál nebo, že nám někdo hledí přes rameno[16].

Hesla obvykle představují první obranou linii systému před těmi, kteří se chtějí do něj vloupat. I když je možné se do systému dostat nebo ukrást nějaké informace po síti i bez

[16] Nahlížení přes rameno za účelem získání nějakých informací se někdy označuje jako ***shoulder surfing***.

znalosti hesla, řada průniků do počítačů uspěla zejména kvůli špatně zvoleným nebo špatně chráněným heslům.

4.2.1. Volba hesla

Přestože hesla představují základní prvek počítačové bezpečnosti, uživatelé o jejich volbě dostanou často jen velmi základní instrukce.

Špatné heslo je každé, které je možno snadno uhodnout. Nejprimitivnější metoda pro uhodnutí hesla je vyzkoušení všech možností. Když bychom vytvořili program, který by zkoušel všechny šestipísmenné kombinace počínaje od AAAAAA a konče u ZZZZZZ, museli bychom vyzkoušet 308 915 776 různých hesel. Pokud bychom zkoušeli jedno heslo za sekundu, potřebovali bychom k tomu skoro deset let. Skuteční narušitelé postupují daleko chytřeji. Namísto ručního zadávání hesel se svým počítačem připojí telefonicky nebo přes síť a zkoušejí hesla. Když je vzdálený systém odpojí, připojí se znovu a zkoušejí dále. Namísto, aby zkoušeli všechny možné kombinace písmen počínaje od AAAAAA (nebo čehokoliv jiného), zkoušejí pouze obecně často používaná hesla[17]. I nepříliš výkonný domácí počítač s dobrým programem pro hádání hesel může vyzkoušet tisíce různých hesel za necelý den. Některé seznamy hesel, které různí útočníci používají, obsahují i několik set tisíc slov. Takže jakékoliv heslo, které by mohl používat kdokoliv jiný, je pravděpodobně heslo špatné.

Dobrá hesla jsou takové hesla, která se dají jen špatně uhodnout. Nejlepší hesla se hádají špatně z následujících důvodů:

- používají malá i velká písmena
- kromě písmen obsahují i číslice a/nebo interpunkční znaky
- mohou obsahovat nějaké řídící znaky a/nebo mezery
- snadno se pamatují, takže je není potřeba zapisovat
- jsou dlouhá sedm nebo osm znaků
- dají se rychle napsat, takže je nikdo nemůže zjistit tím, že by vám hleděl přes rameno

[17] Tyto útoky bývají často nazývány jako ***slovníkové***.

4.2.2. Hesla na více počítačích

Pokud máme účty na několika počítačích, možná budeme chtít používat na všech stejné heslo, abychom si jich nemuseli pamatovat tolik. Pokud však máme na více počítačích stejné heslo a dojde k průniku do jednoho z nich, jsou potencionálně narušeny naše účty na všech počítačích. Jedna obvyklá metoda, kterou volí uživatelé s účty na více počítačích, spočívá v tom, že mají základní heslo, které se na jednotlivých počítačích lehce modifikuje. Můžeme mít například základní heslo *kxyzzy* následované prvním písmenem jména počítače. Takže na počítači *athena* budeme mít heslo *kxyzzya*, a na počítači *ems* heslo *ksyzzye*. Pochopitelně, že nebudeme pro modifikaci hesel používat přesně tuto metodu.

4.2.3. Jednorázová hesla

Nejúčinnější způsob jak minimalizovat riziko plynoucí z použití špatného hesla je úplně vyloučit požití konvenčních hesel. Namísto toho bychom mohli nainstalovat software a/nebo hardware, který umožní práci s ***jednorázovými hesly***. Jednorázové heslo je přesně to, co napovídá jeho název – heslo určené pro jediné použití.

Jako uživatel můžeme dostat vytištěný seznam hesel. Vždy, když heslo použijeme, na seznamu jej škrtneme a při příštím přihlášení použijeme následující heslo ze seznamu. Nebo můžeme dostat malou kartu, kterou budeme nosit s sebou; na kartě se bude každou minutu objevovat jiné číslo. Nebo dostaneme malou kalkulačku. Když se budeme přihlašovat, počítač nám zobrazí nějaké číslo. Toto číslo napíšeme na kalkulačce, přidáme k němu naše osobní číslo a zobrazený výsledek pak použijeme jako heslo.

Každý z těchto systémů výrazně zvyšuje bezpečnost celého systému. Protože však vyžadují instalaci nějakého speciálního softwaru nebo pořízení speciálního hardwaru, nejsou tyto systémy v současnosti příliš rozšířené.

4.2.4. Shrnutí

Hesla představují jednu ze základních forem autentizace. Uživatel se musí prokázat správným heslem při zřizování spojení z důvodu autorizace přístupu do systému, aplikací nebo

k informacím. Bezpečnostní systémy, které jsou závislé na heslech vyžadují, aby hesla byla udržována vždy v tajnosti. Avšak tato ochrana bývá často napadena krádežemi a následným zneužitím (zvláště v internetovém prostředí).

4.3. Autentizace uživatele v internetovém prostředí

World Wide Web představuje jedno z nejzajímavějších využití internetu. Zároveň však přináší i potencionální problémy s bezpečností. Autentizace uživatele ve webovém prostředí je stále nejčastěji založena na kombinaci uživatelského jména a hesla. Při přenosu těchto důvěrných informací mezi Web serverem a prohlížečem je však možné tyto informace zachytit.

4.3.1. Vyloučení možnosti odposlechu

Riziko odposlechu se týká všech internetových protokolů, je však významné zejména u služeb WWW, kdy se mohou přenášet důvěrné dokumenty a další údaje, například čísla kreditních karet. Existují pouze dva způsoby, jak informace před odposlechem chránit. První možnost je přenášet je pouze po fyzicky bezpečných linkách (což Internet nesplňuje). Druhá možnost je zašifrovat informace tak, aby je mohl dešifrovat pouze autorizovaný přijemce.

Jinou možností odposlechu je ***analýza provozu***. Při tomto odposlechu se útočník dozví o transakcích, které sledovaný objekt provádí, aniž by však znal jejich obsah. Tento typ útoků bývá nejčastěji namířen na logovací soubory Web serverů.

4.4. Analýza využití digitálního podpisu pro autentizaci přístupu

Ve webovém prostředí se objevuje riziko odposlechu citlivých informací. Tedy útočník může uživatelské jméno a heslo odposlouchat. Musíme se tedy zamyslet nad silnějším zabezpečením přihlašování.

Jak již bylo řečeno v úvodu této kapitoly, pro ověření totožnosti uživatele je možno použít něco, co víme (heslo) nebo něco, co vlastníme (identifikační kartu). Majitel digitálního podpisu má také něco, co vlastní pouze on – privátní klíč. Smyslem této podkapitoly je zjistit

možnost získání potřebných údajů k autentizaci digitálním podpisem. Cílem je zvýšení bezpečnosti přístupu k citlivým informacím. To znamená, že nebude stačit pouhé přihlašování uživatelským jménem a heslem, ale bude vyžadován i jeho digitální certifikát.

Digitálním podpisem lze podepsat vše od e-mailů až po databázové položky. To znamená, že by mělo jít podepsat i uživatelské jméno a heslo při přihlašování. Pro získání podpisu by uživatel použil nějaký externí program, do kterého by zadal své uživatelské jméno, heslo a cestu k souboru se svým privátním klíčem. Program by potom vypočítal digitální podpis přihlašovacích položek, který by uživatel schránkou zkopíroval do pole pro digitální podpis v prohlížeči. Toto řešení by určitým způsobem zvýšilo bezpečnost přihlašování. Ovšem pokud útočník odposlouchává komunikaci mezi klientem a serverem, získá uživatelské jméno, heslo a z nich vypočítaný digitální podpis. Pokud se tedy potom bude chtít do systému dostat, jednoduše tyto položky při přihlášení použije[18].

Předchozí případ se tedy musí vylepšit o určitou unikátní informaci při každém přihlášení. Server tedy před přihlášením musí vygenerovat nějakou náhodnou sekvenci, která se navíc zadá do programu pro vypočítání digitálního podpisu. Tedy podpis bude při každém přihlášení unikátní. To znamená, že i kdyby útočník odposlouchal uživatelské jméno, heslo i podpis, není mu to nic platné, protože při novém přihlášení server vygeneruje jinou náhodnou sekvenci. Bez znalosti privátního klíče se útočník do systému nedostane. Tato varianta přihlašování už kritéria bezpečnosti splňuje. Ovšem nyní si ji musíme rozebrat z pohledu řadového uživatele. Ten musí udržovat soubor s privátním klíčem v tajnosti. Dále musí vygenerovat podpis v externí aplikaci, do které musí předtím zadat svoje uživatelské jméno, heslo a serverem vygenerovanou unikátní sekvenci. Nakonec musí vygenerovaný podpis přenést schránkou do prohlížeče. Takto složitou proceduru přihlašování by však málokterý uživatel uvítal!

Dalším možným řešením daného problému je využití služeb vrstvy SSL (*Secure Socket Layer*), která je z části založena na předchozí úvaze. Hlavní roli při ověřování identity zde hraje digitální podpis. Řešení užitím SSL přináší navíc další výhody (viz 4.6 Závěr).

[18] Útočník by se tak dostal do systému bez znalosti privátního klíče!

Zvolené řešení:

Do aplikací se bude uživatel přihlašovat stále systémem uživatelské jméno a heslo, ale k úspěšnému přihlášení však bude potřebovat ještě certifikát nainstalovaný v prohlížeči. SSL zařídí ověření digitálního podpisu. V aplikaci se potom musí ještě zkontrolovat, zda se uživatel prezentoval správným certifikátem. K tomu potřebujeme udržovat databázi o uživatelích a jejich certifikátech. Zamezíme tím útočníkovi, který by vlastnil jakýkoliv certifikát a získal uživatelské jméno a heslo někoho jiného.

4.5. Secure Socket Layer (SSL)

SSL technologie umožňuje dvě funkce: šifruje informační tok mezi klientem a serverem a vytváří základ pro vzájemnou klient/server autentizaci. SSL bylo vyvinuto firmou Netscape Communication v roce 1994. Je podporováno populárními klientskými aplikacemi (*Netscape Navigator*, *Microsoft Internet Explorer*), většinou serverových aplikací (*Netscape*, *Microsoft*, *Apache*, *Oracle*, *NSCA* a dalších) a certifikačními autoritami jako je ve světě *VeriSign* a u nás *I. CA*. Dřívější verze SSL 2.0 umožňovala pouze autentizaci serveru (pouze server potřeboval certifikát). Současná verze SSL 3.0 zajišťuje i autentizaci klienta (server i klient se musí prokázat certifikátem).

4.5.1. Základ SSL

SSL je založeno na vytvoření zabezpečeného kanálu mezi klientem (prohlížečem) a serverem. Tento kanál garantuje důvěrnost každé zprávy, která se jím přenáší. SSL nešifruje žádné informace uložené na klientu ani na serveru. SSL zabezpečuje aplikační protokoly jako jsou HTTP, NNTP a Telnet. SSL umožňuje bezpečnou výměnu informací při inicializaci TCP/IP spojení, při kterém se klient a server domluví na konkrétní užité bezpečnosti a vykonají vzájemnou autentizaci certifikáty. Od tohoto bodu, pokud je šifrování aktivováno (počáteční stav), SSL šifruje a dešifruje proud bytů použitého aplikačního protokolu. To znamená, že všechny informace v HTTP požadavku i v HTTP odpovědi jsou plně zašifrované včetně URL, které klient požaduje, potvrzených obsahů formulářů[19], přístupových autorizačních informací a všech dat vrácených serverem klientovi.

[19] Tedy i jméno a heslo v přihlašovacím formuláři.

Protože HTTP+SSL (nebo „HTTPS“) a HTTP jsou dva různé protokoly, které typicky fungují na různých portech (443 a 80). Na stejném systému může běžet zabezpečený i nezabezpečený HTTP server zároveň. To znamená, že můžeme zpřístupnit některé informace všem uživatelům bez zabezpečení a jiné informace již určitým uživatelům užitím zabezpečení. Například v internetovém obchodu by měl být katalog produktů přístupný nezabezpečeně a objednávání a placení zabezpečeně.

SSL používá kryptografii s veřejným klíčem pro výměnu *klíče sezení* mezi klientem a serverem. Tento klíč je unikátně generován pro každé spojení mezi serverem a klientem a je použit pro šifrování HTTP transakcí (požadavku a odpovědi). Kryptografie s veřejným klíčem je užita jen pro vzájemné ověření a k zašifrování klíče sezení. SSL používá šifrování s privátním klíčem[20] při šifrování další výměny informací. Každá transakce se provádí s různým klíčem sezení, takže i kdyby útočním „ukradl“ jeden klíč, v další komunikaci mu to nijak nepomůže.

Nové webové klientské aplikace mají již zabudovány klíče některých certifikačních autorit jako je VeriSign, další je možno samozřejmě doinstalovat. Tyto zabudované klíče umožňují klientské aplikaci ověřit legitimitu kontrolou identity serveru a identity CA, která danému serveru certifikát vydala. SSL vyžaduje, aby server měl digitální certifikát vydaný důvěryhodnou certifikační autoritou. Tento verifikační proces probíhá transparentně. Pokud klient (prohlížeč neboli uživatel) nedůvěřuje certifikační autoritě (nemá nainstalován její certifikát), která vydala serveru certifikát, klientská aplikace vypíše varovné hlášení.

Přidání každé bezpečnostní vrstvy zpomaluje serverové procesy. SSL není výjimkou. Avšak v aplikacích, kde je nutná zvýšená bezpečnost, SSL vysoce převyšuje risk.

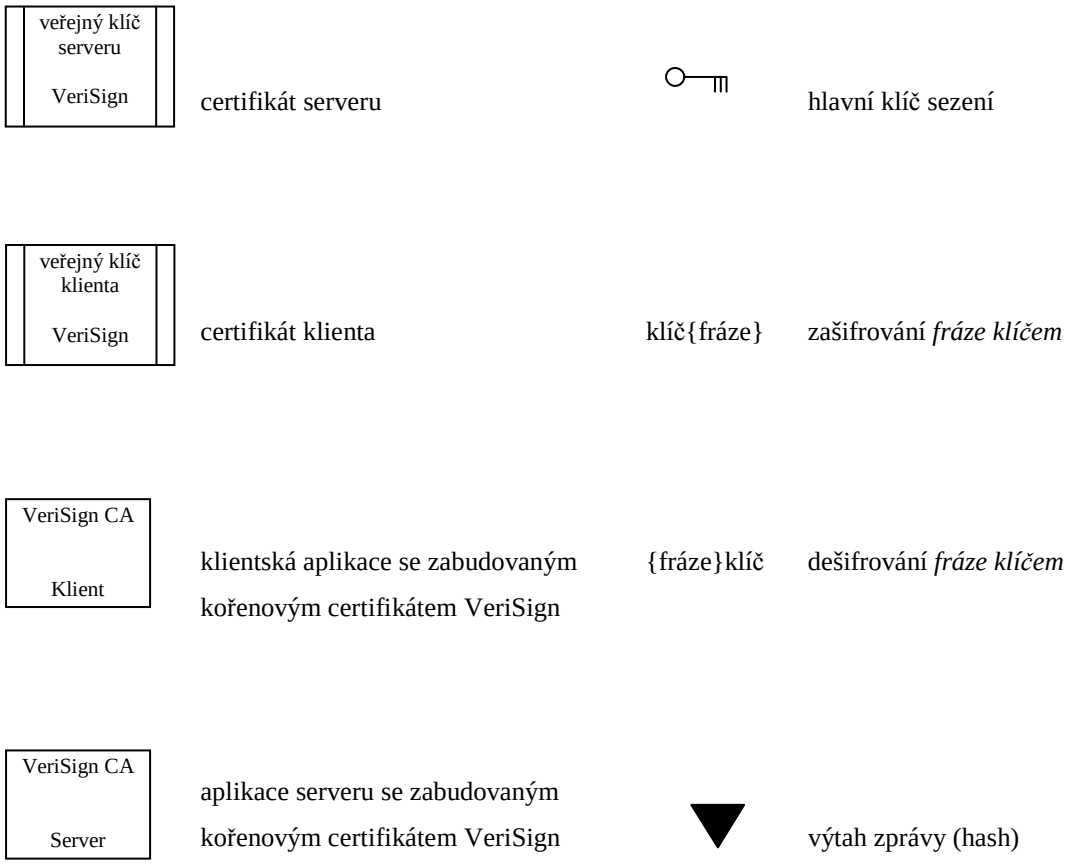
4.5.2. SSL proces

SSL používá sérii klient/server výměn a vytvoření zabezpečeného sezení. Dvě hlavní fáze SSL protokolu jsou:

[20] Které je mnohem rychlejší než šifrování s veřejným klíčem.

1. vytvoření privátní komunikace
2. klientská autentizace

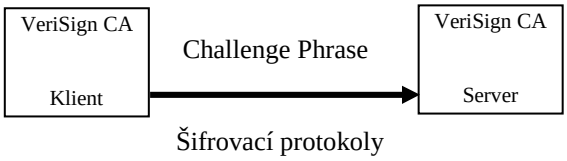
SSL proces se může zdát dlouhý a komplikovaný, ale pracuje konkrétně transparentně k uživateli a probíhá hned po vytvoření spojení mezi klientem a serverem. V následujícím výkladu budeme jako certifikační autoritu jmenovat VeriSign, což je světově uznávaná CA. Grafika užívá následující legendu:



4.5.2.1. Fáze 1: „Klientské ahoj“ (Client Hallo)

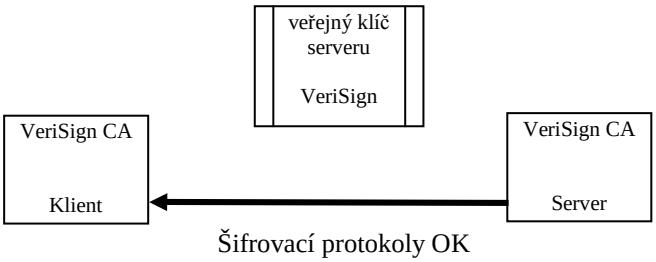
Tento první krok nastane, když se klientská aplikace zkouší připojit k zabezpečené stránce. Aplikace nejprve posílá ***řetězec náhodné výzvy*** (*Random Chalange String* nebo *Challenge Phrase*) serveru, potom vybírá, kterou sadu šifrovacích protokolů použít (závisí na protokolech nainstalovaných v aplikaci). Klientská aplikace musí vybrat algoritmus pro výměnu klíče sezení (fáze autentizace serveru), jako je například DES. Dále musí vybrat algoritmus šifrování privátního klíče (jako je RC2 nebo RC4), a hešovací algoritmus

zajišťující integritu zprávy (jako je MD5 nebo SHA). Tyto algoritmy se použijí při zabezpečeném přenosu. Je mnoho jiných algoritmů, které se mohou použít.



4.5.2.2. Fáze 2: „Serverové ahoj“ (Server Hallo)

Server potvrzuje svoji identitu navrácením svého certifikátu plus prohlášením, že podporuje sadu algoritmů vybranou klientem. Navíc generuje ***náhodný identifikátor spojení***, který bude použit při komunikační fázi.



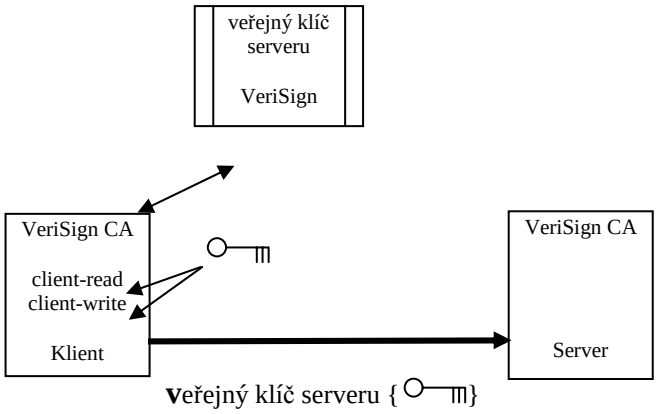
4.5.2.3. Fáze 3: „Hlavní klíč klienta“ (Client Master Key)

Klientská aplikace ověřuje certifikát serveru porovnáním podpisu certifikační autority (CA) v certifikátu serveru s veřejným klíčem CA zabudovaným v klientské aplikaci. Pokud klient nemá klíč CA nebo certifikát CA v klientské aplikaci nesouhlasí s CA serveru, uživatel přijme varovné hlášení, že tento server vlastní certifikát neznámý klientské aplikaci. Uživatel má možnost ukončit spojení, vždy věřit certifikátům nové CA serveru nebo věřit certifikátu pouze při tomto spojení.

Po ověření serveru klient generuje ***hlavní klíč sezení*** (*Master Session Key*). Tento klíč je použit jako semínko k vygenerování komunikačních klíčů klienta i serveru. Dvě symetrické sady párů klíčů jsou vytvořeny: jeden pro příchozí a jeden pro výchozí komunikaci. Protože k vytvoření klíčů byl jako semínko použit pouze jeden klíč, ***zápisový klíč serveru*** (*server write-key*) je shodný jako ***čtecí klíč klienta*** (*client read-key*) a ***čtecí klíč serveru*** (*server read-*

key) odpovídá ***zápisovému klíči klienta*** (*client write-key*). Důležité je, že hlavní klíč sezení je vygenerován klientem a ne serverem, což uživateli zajišťuje další vrstvu bezpečnosti.

Nakonec klient šifruje klíč sezení veřejným klíčem serveru (certifikát serveru jej obsahuje) a posílá jej serveru.

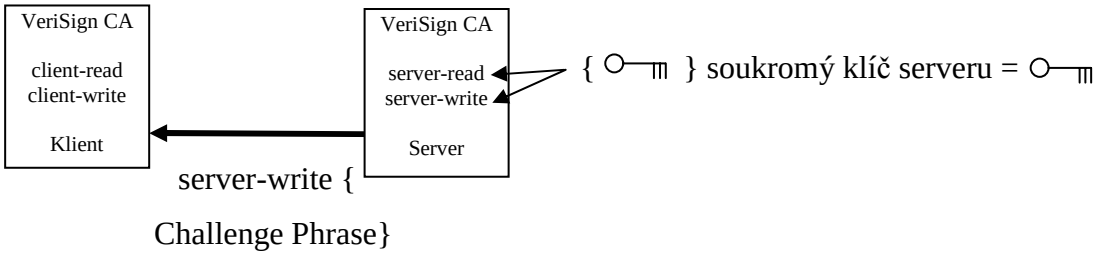


4.5.2.4. Fáze 4: „Klient skončil“ (Client Finish)

Klient končí svou část privátní komunikační fáze zašifrováním identifikátoru spojení serveru klientským zápisovým klíčem. Potom čeká na obdržení zprávy „Server skončil“, aby se ujistil, že zabezpečení kanálu je hotové.

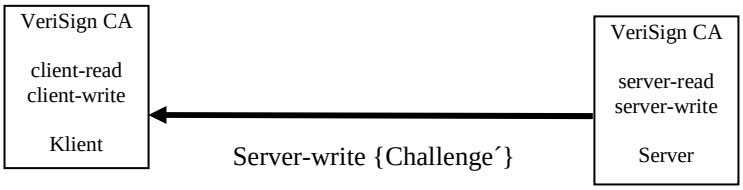
4.5.2.5. Fáze 5: „Ověření serveru“ (Server Verify)

Server dešifruje hlavní klíč sezení svým privátním klíčem. Klíč sezení použije k vytvoření odpovídajícího párů klíčů (*server-read* + *server-write*). Potom server vrací klientovi inicializační frázi (*Challenge Phrase*) zašifrovanou zápisovým klíčem serveru. Toto je potvrzení autenticity serveru, protože pouze hlavní klíč sezení mohl vytvořit klíč, kterým byla zašifrována inicializační fráze.



4.5.2.6. Fáze 6: „Požadavek certifikátu“ (Request Certificate)

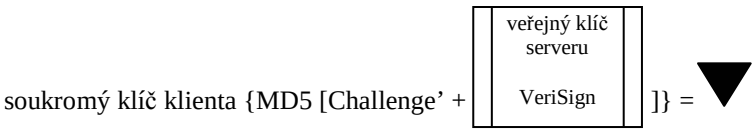
Server nyní požaduje, aby se klient prezentoval platným certifikátem[21] a posílá klientovi novou ***výzvu*** (*Challenge´*) zašifrovanou zápisovým klíčem serveru.



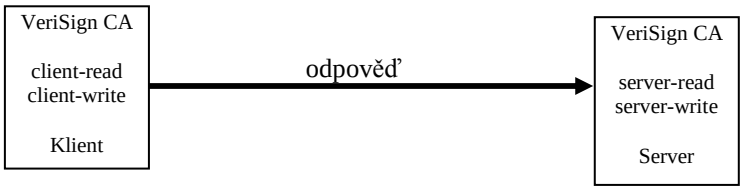
4.5.2.7. Fáze 7: „Certifikát klienta“ (Client Certificate)

Pokud klient nemá certifikát, odpovídá chybovou zprávou. Jinak dešifruje výzvu serveru (svým čtecím klíčem) a vytváří odpověď, která se skládá z výtahu výzvy serveru (např. algoritmem MD5) plus certifikátu serveru. Tato odpověď plus certifikát klienta je digitálně podepsána[22] klientovým soukromým klíčem a potom odeslána serveru.

{Challenge’} client-read = Challenge’



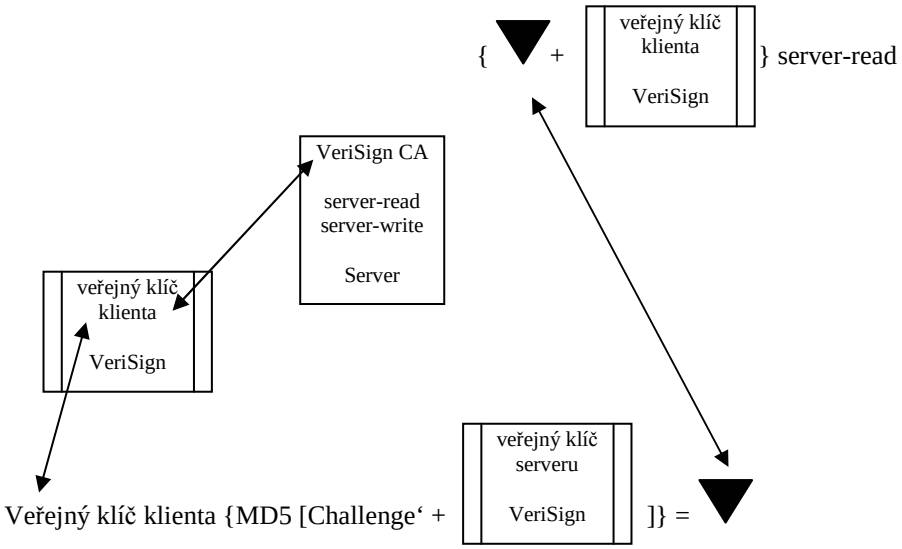
[21] Předpokládáme, že webová stránka je nakonfigurována k akceptování certifikátů.
[22] Zde se uplatňuje digitální podpis, který zajišťuje autenticitu uživatele.



4.5.2.8. Fáze 8: „Ověření certifikátu klienta“ (Client Cerificate Verification)

Server ověřuje autenticitu klienta dvěma způsoby:

- nejprve kontrolou, že certifikát vydala důvěryhodná CA – podpis certifikátu ověří proti zabudovanému seznamu kořenových CA,
- potom vytvořením výtahu výzvy (*Challenge´*) plus certifikátu serveru a porovnáním s výtahem v odpovědi



4.5.2.9. Fáze 9: „Server skončil“ (Server Finish)

Server končí posláním unikátního identifikátoru sezení zašifrovaného zápisovým klíčem serveru. Tento unikátní identifikátor je použit po zbytek sezení, což eliminuje potřebu další výměny, která by dále zpomalovala komunikaci.

4.6. Závěr

Problém autentizace přístupu do webových aplikací není jednoduchý. Přístup nikdy nebude stoprocentně bezpečný. Digitální podpis však přináší další neopomenutelnou vrstvu zabezpečení.

Řešení užitím SSL přináší navíc následující výhody:

- SSL probíhá transparentně – uživatel není ničím zatěžován, pouze musí mít v prohlížeči nainstalován certifikát od certifikační autority, kterou server „uznává“
- autentizace probíhá vzájemně – server se autentizuje klientovi a naopak
- komunikace je šifrována
- SSL podporuje většina moderních prohlížečů i serverů

5. Aplikační server Oracle 9iAS

Aplikační server Oracle9i poskytuje integrované prostředí pro vývoj, distribuci a správu internetových aplikací. Spolu s databází Oracle9i tvoří spolehlivou, škálovatelnou a bezpečnou základnu pro provozování e-business aplikací v prostředí internetu a intranetu, včetně portálů, transakčních aplikací, služeb typu *Business Intelligence*, mobilních aplikací a integrace podnikových aplikací a dat.

Oracle9iAS obsahuje celou řadu modulů potřebných pro provozování aplikací na střední vrstvě a je komplexním řešením pro nasazení našich aplikací na internetové platformě. Sdílením klíčových částí systému společně s Oracle9i je zajištěna spolehlivost, dostupnost a škálovatelnost, nezbytná pro důležité aplikace provozované v prostředí internetu.

5.1. Služby Oracle 9iAS

Oracle Internet Application Server 9i implementuje celou řadu služeb zajišťujících:

- komunikaci
- prezentační logiku
- aplikační logiku
- *kešing* dat
- další systémové služby potřebné k provozování libovolné internetové aplikace

5.1.1. Komunikační služby

Komunikační služby zajišťují zpracování požadavků odesílaných klienty na iAS. Klientem může být buď webovský prohlížeč, aplikace běžící na klientském počítači, mobilní zařízení apod. Oracle HTTP Server je implementován nad Apache HTTP Serverem, který je vstupním bodem Oracle iAS, zpřístupňujícím jak staticky, tak i dynamicky generované informace. Statické informace jsou načítány přímo ze souborů, dynamické informace jsou generovány po zpracování požadavku aplikačním modulem (tzv. “mod”), který spustí a provede příslušnou aplikaci. Tato aplikace může běžet buď přímo na Oracle iAS, nebo v databázi Oracle9i.

Oproti modulům, které jsou standardně dodávany s Apache serverem obsahuje Oracle HTTP Server také moduly vyvinuté a podporované společností Oracle. Klíčové moduly dodávané s Oracle HTTP serverem jsou následující:

- **mod_jserv** - zpracovává HTTP požadavky na spuštění servletů v Oracle iAS servlet engine.

- **mod_perl** - zpracovává HTTP požadavky na provedení Perl skriptů ve standardním Apache Web Server Perl interpreteru.

- **mod_ssl** - zajišťuje služby PKI založené na bezpečnostních certifikátech a SSL, kóduje komunikaci mezi klientem a Apache Serverem.

- **mod_plsql** - modul vyvinutý Oracle, zajišťuje provádění HTTP požadavků na spuštění uložených databázových procedur naprogramovaných v jazycích PL/SQL nebo Java.

Kromě klientů přistupujících prostřednictvím protokolu HTTP, podporuje Oracle iAS také klienty přistupující prostřednictvím protokolů IIOP a Net8.

5.1.2. Prezentační služby

Zajišťují zobrazování dokumentů klientovi. Oracle iAS umožňuje prezentovat data několika různými způsoby:

- **Oracle Portal** - je kompletním řešením otázek spojených s konfigurací a provozováním informačního portálu. Tvoří základnu sloužící k integraci podnikových aplikací a dat, bezpečný přístup k těmto aplikacím a datům, včetně možnosti tvorby osobních konfigurací pro jednotlivé uživatele.

- **Apache JServ** - je Java servlet engine plně kompatibilní se specifikací Sun Microsystems Java Servlet 2.0 API. Servlety jsou jednoduchou a výkonnou základnou pro tvorbu aplikací určených pro WWW.

- **Perl Interpreter** - Oracle HTTP Server Perl Interpreter umožňuje využít potenciál, který skýtá programovací jazyk Perl a v něm vytvořené, vysoce výkonné skripty.

- **Oracle JavaServer Pages (JSP)** - tvoří překladač a runtime engine vyvinutý firmou Oracle pro generování JavaServer Pages. JavaServer Pages umožňují použít kód v jazyce Java přímo v HTML dokumentu a přitom zachovat aplikační a prezentační logiku odděleně.

- **Oracle PL/SQL Server Pages (PSP)** - tvoří překladač a runtime engine vyvinutý firmou Oracle. PSP umožňují použít kód v jazyce PL/SQL přímo v HTML dokumentu a přitom zachovat aplikační a prezentační logiku odděleně. PSP kompilátor převádí a překládá PSP stránky na uložené PL/SQL procedury a pro generování HTML dokumentů používá PL/SQL Web Toolkit.

5.1.3. Aplikační logika

Oracle iAS obsahuje několik nástrojů pro tvorbu a provozování aplikací:

- **Oracle Container for Java (OC4J)** - je server-side Java engine podporující J2EE API (včetně servletů, JSP a Enterprise JavaBeans), CORBA a uložené databázové procedury.

- **Oracle9i PL/SQL engine** - je škálovatelným prostředím pro provozování PL/SQL procedur, PL/SQL Web aplikací a PSP stránek na střední vrstvě. Sdílí společnou infrastrukturu s Oracle9i Cache.

- **Business Components for Java (BC4J)** - obsahuje knihovny tříd a objektů určených pro vývoj aplikací pracujících s databázovými objekty v jazyce Java. Komponenty vytvořené s využitím BC4J mohou být provozovány jako servlety či JSP, nebo mohou být nasazeny ve formě Enterprise Java Beans či objektů CORBA v Oracle9i JVM.

- **Forms Services** - umožňují výkonný provoz rozsáhlých transakčních aplikací vytvořených nástrojem Forms Developer ve 3-vrstvé architektuře

- **Reports Services** - zajišťují bezpečný provoz dynamicky generovaných reportů vytvořených nástrojem Reports Developer.

- **Discoverer Services** - je runtime zajišťující provozování sestav vytvořených nástrojem Discoverer. Umožňuje provádění ad-hoc dotazů a analýzy ve standardním WWW prohlížeči.

5.1.4. Služby pro zvýšení výkonnosti

Slouží k rychlejšímu odbavení uživatelů a k odlehčení zátěže databázového serveru.

- **Oracle Web Cache** - patentované *kešování* dynamických stránek.

- **Oracle Data Cache** - zajišťuje read-only, transparentní kešování dat na střední vrstvě. Přispívá ke zvýšení výkonu aplikací přistupujících k databázovým datům a tím i celého Web serveru. Oracle9i Cache pracuje s libovolným Web serverem komunikujícím s databází prostřednictvím OCI nebo jakéhokoli jiného software přes OCI přistupujícího (např. JDBC, PRO*C, ODBC).

5.1.5. Systémové služby

Oracle9iAS realizuje systémovou správu a zabezpečení následujícími nástroji:

- **Oracle Enterprise Manager** - je nástroj určený pro centrální správu všech produktů firmy Oracle. Sestává z grafické konzole, Oracle Management Serveru a Oracle Intelligent Agents. Poskytuje jednotné komplexní prostředí pro správu Oracle9iAS i Oracle9i.

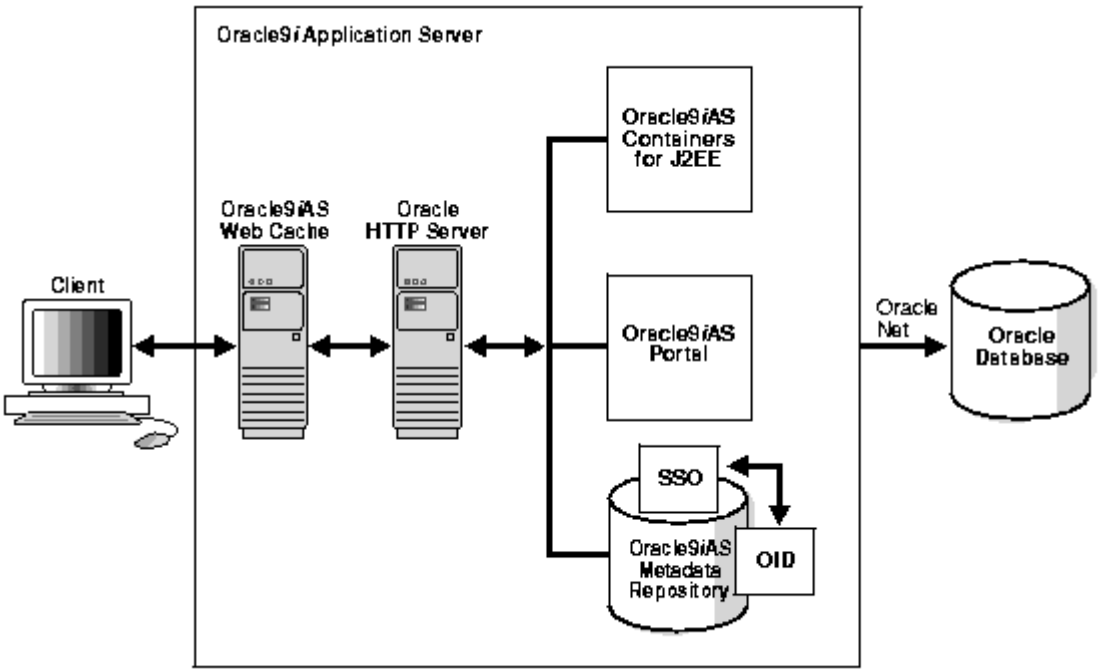
- **Oracle Advanced Security** - implementuje bezpečnostní služby pro Oracle9i Cache, Oracle9i JVM, a Oracle9i PL/SQL. Zajišťuje kódování dat přenášených po síti, autentizaci uživatelů a podporuje celou řadu bezpečnostních protokolů včetně Secure Sockets Layer, RADIUS a Kerberos.

5.2. Bezpečnostní architektura Oracle 9iAS

Oracle 9iAS představuje spolehlivý systém pro vytváření webových aplikací, jehož základem je Oracle HTTP server, který je založený na serveru Apache, Oracle 9iAS Containers for J2EE a Oracle 9iAS Portal. Bezpečnost AS začíná u značně konfigurovatelného http serveru, přidává obsáhlou sadu webových Single Sign-On služeb a rozšiřuje je dále centralizovanou správou uživatelů, kterou umožňuje Oracle Internet Directory, LDAP verze 3. Navíc AS poskytuje implementaci služeb autorizace a autentizace v Javě (JAAS – *Java Authorization and Authentication Services*) pro bezpečnost aplikací J2EE. Oracle9iAS také podporuje bezpečnost přístupu do Oracle databáze – *Oracle Advance Security*.

5.2.1. Elementy architektury bezpečnosti AS

Následující obrázek ukazuje architekturu bezpečnosti Oracle 9iAS. Na levé straně diagramu je klient, který komunikuje s Web Cache, která potom komunikuje s HTTP serverem. HTTP server potom může přímo komunikovat s kontajnery pro J2EE, Single Sign-On nebo portálem. Single Sing-On komunikuje s Oracle Internet Directory, který potom komunikuje s Metadata Repository, kde jsou uloženy informace o uživatelích a skupinách uživatelů.



Oracle 9iAS Web Cache

- Web cache, která může být nakonfigurována na podporu protokolu https, je umístěna vpředu, kde *kešuje* často přistupované webové stránky nebo části stránek.

Oracle HTTP Server

- Web server podporuje protokoly http i https a za pomoci plug-inů směruje požadavky pro autentizaci a autorizaci.

Oracle 9iAS Containers for J2EE a JAAS

- Technologie Oracle 9iAS Containers for J2EE poskytuje prostředí pro běh komponent AS. JAAS zajišťuje bezpečné provádění a přístup do *javovských* aplikací spolu s integrací se Single Sign-On.

Oracle 9iAS Portal

- Portal poskytuje infrastrukturu pro tvorbu a správu webových stránek. Dovoluje uživateli zobrazit více stránek na každé stránce portálu s odkazy na obsah skrze *javovské* aplikace. Portal používá Single Sign-On k poskytování přístupu k obsahu a aplikacím.

Oracle 9iAS Single Sign-On

- Single Sign-On poskytuje jednoduchou, jednotnou autentizační službu komponentám, aplikacím a webovým stránkám AS. Single Sign-On server ukládá a autentizuje uživatele proti Oracle Internet Directory.

Oracle Internet Directory

- Toto LDAP, verze 3, slouží prostředníkovi (např. Single Sign-On server) umožněním autentizace a centralizovaného uživatelského modelu, ve kterém uživatelé mohou být vytvářeni, spravováni a ukládáni.

Oracle 9iAS Metadata Repository

- Toto je databáze, která obsahuje schémata užívaná komponentami AS.

6. Tvorba certifikátů

První možností je získávání certifikátů od autorizovaných certifikačních autorit jako je v ČR 1. Certifikační autorita. Je možno získat testovací certifikát s platností 14 dní, avšak standardní certifikáty s delší platností již bezplatné nejsou.

Další možností je využít software určený pro vytvoření vlastní certifikační autority a jejích certifikátů. Pro realizaci této diplomové práce bylo zvoleno prostředí příkazové řádky OpenSSL.

Výhody generování certifikátů s OpenSSL:

- standardní součást systému Oracle 9iAS
- OpenSSL je volně dostupné na adrese: http://www.openssl.org

Dalším programovým nástrojem určeným pro tvorbu a správu certifikátů je program *keytool*, který je součástí *Java 2 Standard Developer Kit 1.4*. Hojně využívaný je také kryptografický produkt PGP neboli *Pretty Good Privacy*, který se umí úzce integrovat do systému a dílčích programů (např. *Microsoft Outlook*), které mohou šifrování potřebovat.

6.1. OpenSSL

OpenSSL je kryptografický nástroj, který implementuje *Secure Socket Layer* (SSL v2/v3) a *Transport Layer Security* (TLS v1) a k nim přidružené kryptografické standardy.

Openssl je program příkazové řádky pro využití funkcí z knihovny *crypto*, která je napsána v jazyce C. Může být použit pro:

- vytváření RSA, DH a DSA klíčů
- vytváření X.509 certifikátů
- počítání výtahů zpráv
- šifrování a dešifrování
- SSL/TLS – testování klienta a serveru
- Podepisování, šifrování a následná verifikace a dešifrování S/MIME emailů

Použitá verze OpenSSL: 0.9.7a

V serveru Oracle 9iAS se OpenSSL nachází v adresáři: *IAS_HOME*\Apache\open_ssl.

6.1.1. Vytvoření certifikační autority

V adresáři open_ssl vytvoříme adresář pro certifikační autoritu (jméno CCA_CA). V tomto adresáři vytvoříme následující adresářovou strukturu:

- certs – adresář pro vydané certifikáty
- crl – adresář pro vydané *crl* (*Certificate Revocation List* - seznam zrušených certifikátů)
- newcerts – implicitní adresář pro ukládání nově vydaných certifikátů
- requests – pro uchovávání požadavků o certifikáty
- private – pro uchování soukromých dat certifikační autority[23]
    - keys – vygenerované privátní klíče uživatelů
    - certsp12 – certifikáty uživatelů ve formátu PKCS#12

V adresáři certifikační autority (CCA_CA) dále vytvoříme následující dva soubory:

- index.txt – textová databáze vydaných certifikátů (při vytváření prázdný soubor)
- serial – soubor s aktuálním sériovým číslem (při vytváření číslo 01)

Do adresáře private zkopírujeme soubor .rnd[24], který se nachází v adresáři open_ssl. Ve stejném adresáři dále vytvoříme soubor s implicitním heslem pro použití privátního klíče (jméno souboru – keypasswd.nfo[25]).

Potom musíme upravit konfigurační soubor openssl.cnf (sekci *CA_default*) podle následujícího příkladu:

```
dir               = ./CCA_CA
certs             = $dir/certs
crl_dir           = $dir/crl
database          = $dir/index.txt
new_certs_dir     = $dir/newcerts
```

[23] Do tohoto adresáře by měl mít přístup pouze správce certifikační autority.
[24] Tento soubor lze také vygenerovat příkazem *openssl rand*.
[25] Implicitní heslo se zapíše do tohoto souboru – např. *welcome*.

```
certificate       = $dir/cacert.pem
serial            = $dir/serial
crl               = $dir/crl.pem
private_key       = $dir/private/cakey.pem
RANDFILE          = $dir/private/.rnd
```

Tento soubor uložíme do adresáře naší CA.

Dalším krokem je vygenerování privátního klíče CA:

```
openssl genrsa -out ./CCA_CA/private/cakey.key 1024
```

Po spuštění tohoto příkazu nás openssl vyzve k zadání hesla privátního klíče. Toto heslo potom můžeme z klíče RSA vymazat následujícím příkazem[26]:

```
openssl rsa -in ./CCA_CA/private/cakey.key -out ./CCA_CA/private/cakey.key
```

Dále vytvoříme certifikát naší certifikační autority (tento certifikát bude kořenový, tzn. že nebude nikým podepsán), který bude mít platnost 2 roky (730 dní):

```
openssl req -x509 -new -key ./CCA_CA/private/cakey.key -out
./CCA_CA/cacert.pem -config ./config/openssl.cnf –days 730
```

Potom tento certifikát převedeme z formátu PEM do formátu DER (kvůli instalaci do prohlížeče):

```
openssl x509 -in ./CCA_CA/cacert.pem -inform PEM -out ./CCA_CA/cacert.der
-outform DER
```

Nakonec vytvoříme počáteční *crl*:

```
openssl ca -gencrl -config ./CCA_CA/openssl.cnf -out ./CCA_CA/crl/crl.pem
```

Pokud budeme potřebovat zrušit platnost některého *certifikátu* zavoláme příkaz:

```
openssl ca -revoke certifikát -config ./CCA_CA/openssl.cnf
```

[26] Tato akce však snižuje bezpečnost certifikační autority.

a vytvoříme nový *crl*.

6.1.2. Vytvoření certifikátu serveru

Dále musíme pro umožnění SSL vygenerovat certifikát serveru. Nejprve vytvoříme privátní klíč:

```
openssl genrsa -out ./CCA_CA/private/keys/server.key -passout
file:./CCA_CA/private/keypasswd.nfo 1024
```

Argument -passout slouží k automatickému načtení hesla privátního klíče ze souboru. Heslo pak můžeme z privátního klíče vymazat[27]:

```
openssl rsa -in ./CCA_CA/private/keys/server.key -out
./CCA_CA/private/keys/server.key
```

Dále vytvoříme požadavek certifikátu (*Certificate Request*):

```
openssl req -new -key ./CCA_CA/private/keys/server.key -out
./CCA_CA/requests/server.req -config ./CCA_CA/openssl.cnf
```

Po spuštění tohoto příkazu jsme dotázáni na jméno, stát, organizaci atd. Tyto data tvoří DN (*Distingish Name* – rozlišovací jméno) serveru. Zde je důležité aby se zadané jméno shodovalo s doménovým jménem serveru – tedy například *brahma.cca.cz*. Pokud se tyto názvy neshodují, je po připojení na server (v rámci protokolu https) vypsáno varovné hlášení o nesprávnosti certifikátu a uživatel se potom může rozhodnout, zda ukončí spojení se serverem či ne.
Po provedení tohoto příkazu vytvoří openssl v textové databázi CA `index.txt` záznam (jednu řádku), ve kterém je sériové číslo, ukončení platnosti a DN nově vydaného certifikátu.

Tento požadavek musí potom podepsat certifikační autorita:

```
openssl ca -in ./CCA_CA/requests/server.req -out
./CCA_CA/certs/servercert.pem -config ./CCA_CA/openssl.cnf
```

[27] Toto samozřejmě představuje také určité snížení bezpečnosti, ale zde to má výhodu, že nemusíme při každém startu http serveru toto heslo zadávat.

6.1.3. Vytvoření certifikátu uživatele

Opět musíme nejprve vygenerovat privátní klíč a můžeme z něj odstranit heslo:

```
openssl genrsa -out ./CCA_CA/private/keys/user01.key -passout
file:./CCA_CA/private/keypasswd.nfo 1024

openssl rsa -in ./CCA_CA/private/keys/user01.key -out
./CCA_CA/private/keys/user01.key
```

Potom vytvoříme požadavek na certifikát:

```
openssl req -new -key ./CCA_CA/private/keys/user01.key -out
./CCA_CA/requests/user01.req -config ./CCA_CA/openssl.cnf -subj
"/C=CZ/ST=Czech Republic/O=Novak a syn s.r.o/CN=Jan Novak" -batch
```

Parametr `-subj` udává DN uživatele, které musí být v rámci jedné CA jedinečné.

Požadavek se pak musí podepsat certifikační autorita.

```
openssl ca -in ./CCA_CA/requests/user01.req -out ./CCA_CA/certs/user01.pem
-config ./CCA_CA/openssl.cnf -batch
```

Certifikát potom převedeme do formátu PKCS#12, ve kterém lze certifikát importovat do prohlížeče:

```
openssl pkcs12 -export -in ./CCA_CA/certs/user01.pem -inkey
./CCA_CA/private/keys/user01.key -out ./CCA_CA/private/certsp12/user01.p12
```

Po spuštění tohoto příkazu musíme zadat přenosové heslo privátního klíče.

6.2. Vydávání certifikátů

Zde s OpenSSL vybudovaná certifikační autorita byla vytvořena pouze z demonstračních důvodů. Správný proces tvorby a vydávání certifikátů by měl být následující:

1. Uživatel si sám vygeneruje privátní klíč. Využije proto jemu dostupný software (tedy například i OpenSSL).
2. Potom vygeneruje požadavek na certifikát – *Certificate Request* (stále svým softwarem).
3. Tento požadavek odešle na podepsání certifikační autoritě nebo jej vloží do formuláře na webových stránkách autority nebo jej přinese přímo (např. na disketě) na pracoviště CA.
4. CA zkontroluje identitu žadatele o certifikát. Měl by být vyžadován alespoň občanský průkaz. Některé cerifikační autority mohou vyžadovat i další doklady.
5. CA podepíše požadavek certifikátu a vzniklý certifikát předá (nebo odešle) žadateli. Může jej klidně odeslat nezabezpečeným emailem, protože tento certifikát obsahuje pouze veřejný klíč.
6. Pro potřebu instalace certifikátu do prohlížeče (ve formě s privátním klíčem – PKCS#12) si může uživatel opět svým softwarem převést certifikát do této formy.

Výhoda tohoto procesu spočívá v tom, že privátní klíč uživatele má k dispozici pouze on sám a veškerá komunikace s CA se týká pouze certifikátu, který obsahuje veřejný klíč.

7. Konfigurace SSL v Oracle HTTP serveru

Pokud chceme umožnit certifikátovou autentizaci musíme správně nakonfigurovat SSL v Oracle HTTP serveru.

7.1. Zprovoznění certifikátu serveru

Dříve získaný certifikát serveru nakopírujeme do adresáře:
IAS_HOME\Apache\Apache\conf\ssl.crt a změníme jméno souboru na ccacert.crt.

Privátní klíč serveru zkopírujeme do adresáře:
IAS_HOME\Apache\Apache\conf\ssl.key a změníme jeho jméno na ccapriv.key.

Certifikát certifikační autority se musí zkopírovat do adresáře:
*IAS_HOME**Apache**Apache**conf**ssl.crt* a změníme jeho jméno na ccaca.crt.

CRL soubor zkopírujeme do adresáře:
IAS_HOME\Apache\Apache\conf\ssl.crl a pojmenujeme ho ccaca.crl.

7.2. Konfigurace SSL v souboru httpd.conf

Oracle HTTP Server je implementován nad Apache HTTP Serverem, což znamená, že jeho konfigurace je uložena v souboru *IAS_HOME*\Apache\Apache\conf\httpd.conf. V tomto souboru nejprve zapneme poslouchání serveru na portu 443 (port určený pro SSL) – direktiva Listen:

```
Listen 443
```

Další nastavení se týkají virtuálního hostitele vytvořeného pro SSL (implicitní jméno: _default_:443):

Zapnutí SSL:

```
SSLEngine on
```

Certifikát serveru:

```
SSLCertificateFile conf\ssl.crt\ccacert.crt
```

Privátní klíč serveru:

```
SSLCertificateKeyFile conf\ssl.key\ccapriv.key
```

Certifikát certifikační autority:

```
SSLCACertificateFile conf\ssl.crt\ccaca.crt
```
[28]

Soubor s CRL:

```
SSLCARevocationFile conf\ssl.crl\ccaca.crl
```

Typ klientské autentizace – tři možnosti:

- `none` – žádné ověřování certifikátu
- `optional` – ověření proběhne, ale uživatel se dostane k požadovanému dokumentu na serveru, i když bude ověření neúspěšné
- `require` – k zpřístupnění požadovaného dokumentu na serveru je vyžadováno úspešné SSL ověření

příklad: `SSLVerifyClient optional`

Kontrola přístupu – direktiva `SSLRequire` – umožňuje podrobnější zkoumání certifikátu. Syntaxe zápisu je podobná mixu mezi jakyky C a Perl.

příklad: `SSLRequire ( %{SSL_CLIENT_S_DN_CN} eq "Jan Novak" )`
- pro úspěšné ověření musí mít certifikát v poli jméno „Jan Novak“

[28] Tento soubor může obsahovat více certifikátů certifikačních autorit, jejichž certifikáty server uznává.

Direktiva `SSLOptions`

- pro pozdější zpřístupnění proměnných prostředí serveru musíme zapnout jejich exportování[29]
- `StdEnvVars` – standardní proměnné prostředí
- `ExportCertData` – proměnné `SSL_CLIENT_CERT` a `SSL_SERVER_CERT` (certifikát klienta a serveru)

Příklad: `SSLOptions +StdEnvVars +ExportCertData`

7.3. Instalace klientského certifikátu v prohlížeči

Pro správnou funkci klientského certifikátu musíme nejprve ve webovém prohlížeči nainstalovat kořenový certifikát certifikační autority. Importování bude popsáno na nejrozšířenějším prohlížeči *Microsoft Internet Explorer* :

Nástroje – Možnosti Internetu – Obsah – Certifikáty – Importovat a vložíme cestu se jménem certifikátu (v případě námi vygenerované CA v odstavci 5.1.2 soubor `cacert.der`) . V dalším dialogu zvolíme:
Automaticky vybrat úložiště certifikátů na základě typu certifikátu.

Certifikát (a po stisknutí informace o něm) můžeme potom nalézt v záložce:
Důvěryhodné kořenové certifikační úřady.

Potom nainstalujeme vlastní certifikát se zabudovaným privátním klíčem. Postup je stejný jako v předchozím případě. V případě námi vygenerovaného certifikátu v odstavci 5.1.4 importujeme soubor `user01.p12`. Certifikát (a po stisknutí informace o něm) můžeme potom nalézt v záložce: *Osobní*.

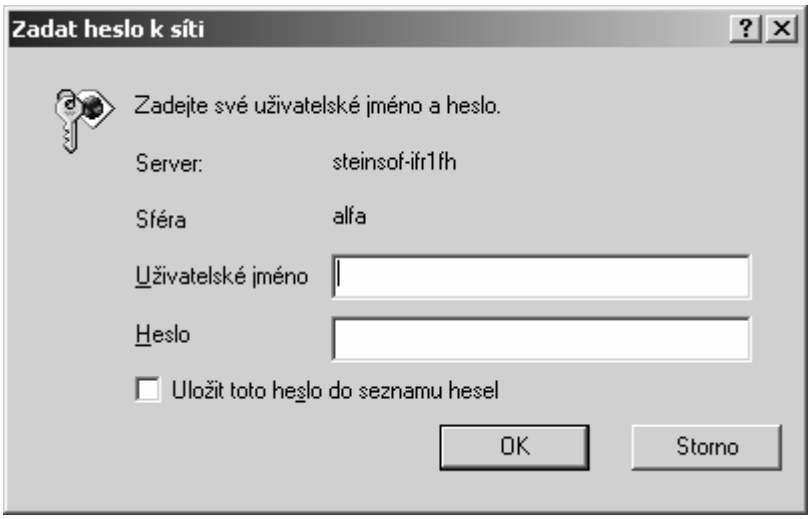
[29] Exportování proměnných prostředí serveru je časově velmi náročná akce. To znamená, že zapnutím této volby se sníží výkon systému.

8. Zabezpečení autentizace uživatele

8.1. Základní autentizace

8.1.1. Základní autentizace v PL/SQL aplikacích

Autentizace v PL/SQL aplikacích je založena na uživatelském jméně a hesle. Při http požadavku na PL/SQL proceduru se objeví následující dialog pro zadání uživatelského jména a hesla pro přístup do databáze (zde `alfa`):



Uživatel má potom tři možnosti pro zadání jména a hesla. Při úspěšném přihlášení se zobrazí stránka generovaná PL/SQL procedurou. Po třetím neúspěšném přihlášení vrátí server chybou stránku (*Authorization Required*).

8.1.2. Základní autentizace v Oracle Portal

Autentizace v portálu je založena také na zadání jména a hesla, ale řešena odlišným způsobem. Po stisknutí odkazu *login* v portálu je uživatel přesměrován na ***Login Server***, který autentizuje uživatele. Při úspěšném přihlášení vytvoří v prohlížeči ***cookie***[30], který umožňuje serveru při doručení dalšího požadavku ověřit uživatele. Uživateli tedy stačí pouze jediné

[30] *Cookie* je informace, kterou prohlížeč uloží pro další použití.

přihlášení k využití všech služeb a aplikací portálu, na jejichž užívání dostal oprávnění. Tato technologie se nazývá ***Single Sign-On*** a podrobněji je popsána v odstavci 8.4.1.

8.2. Aplikační rozhraní zabezpečení autentizace certifikáty

V rámci této diplomové práce bylo vytvořeno aplikační programové rozhraní pro zabezpečení autentizace certifikáty. Toto API podporuje autentizaci uživatele v PL/SQL aplikacích i v Oracle Portal. Tato podkapitola popisuje implementaci a konfigurační možnosti tohoto API, které je napsáno v jazyce PL/SQL.

8.2.1. Vytvoření uživatele CERT_AUTH_ADMIN

Pro potřeby uložení tohoto API bylo vytvořeno uživatelské konto `CERT_AUTH_ADMIN` (správce certifikátové autentizace)[31].

Přihlásíme se do SQL Plus jako uživatel *sys* a vytvoříme konto `CERT_AUTH_ADMIN`:

```
SQL> create user cert_auth_admin identified by cert_auth_admin default
tablespace user_data temporary tablespace temporary_data;
```

Správci certifikátové autentizace musíme potom přidělit potřebná práva:

1. `connect` – připojení do databáze
2. `unlimited tablespace` – prostor pro tabulky
3. `create public synonym` – tvorba veřejných synonym
4. `drop public synonym` – rušení veřejných synonym
5. `create table` – tvorba tabulek
6. `insert any table` – vkládání do tabulek
7. `delete any table` – mazání z tabulek
8. `select any table` – zobrazení tabulek
9. `drop any table` – rušení tabulek
10. `create user` – tvorba uživatelů

[31] API lze samozřejmě vložit do každého schématu, které má potřebná práva (tedy např. schéma *sys*).

11. `drop user` – mazání uživatelů
12. `grant any privilege` – přidělování práv
13. `create procedure` – tvorba procedur
14. `execute any procedure` – spouštění procedur

Přidělení práv `connect`, `select any table` a `execute any procedure` musí mít klauzuli `WITH ADMIN OPTION` pro možnost vytváření uživatelů aplikací.

```
SQL> grant connect, select any table, execute any procedure to
cert_auth_admin with admin option;

SQL> grant unlimited tablespace,  create public synonym,  drop public
synonym, create table, insert any table, delete any table, drop any table,
create user, drop user, grant any privilege, create procedure to
cert_auth_admin;
```

8.2.2. Tabulka uživatelů a jejich certifikátů

Ve schématu `CERT_AUTH_ADMIN` teď musíme vytvořit tabulku uživatelů PL/SQL aplikací se zabezpečenou autentizací (jsme přihlášeni již jako `cert_auth_admin`):

```
SQL> create table cert_auth_user_t(
  user_name varchar(30) not NULL,   -- uživatelské jméno
  app_name varchar(30) not NULL,    -- jméno aplikace
  ssl_client_cert varchar2(2000),   -- certifikát uživatele
  ssl_client_i_dn varchar2(200),    -- DN vydavatele certifikátu
  ssl_client_s_dn varchar2(200),    -- DN vlastníka certifikátu
  constraint pk_cert_auth_user_t primary KEY ( user_name, app_name )
);
```

Primárním klíčem této tabulky je dvojice uživatelské jméno – jméno aplikace. Tato implementace tedy počítá s jednou i více aplikacemi. Pro každou aplikaci, do které chce uživatel přistupovat, musí mít záznam v této tabulce.

Poznámka:

Otázkou je, zda má být u položky certifikátu uživatele klíčové slovo `UNIQUE`. Pokud zde toto slovo bude, tak to znamená, že by uživatel musel mít pro každou aplikaci jiný certifikát. Pokud toto pole nebude muset být unikátní, uživatel může přistupovat do více aplikací se stejným certifikátem. Tato druhá možnost je pohodlnější jak pro uživatele, tak i pro vydavatele certifikátů.

8.2.3. PL/SQL balík CERT_AUTH_API

Do schématu potom `CERT_AUTH_ADMIN` potom vložíme specifikaci a tělo balíku `CERT_AUTH_API`.

```
SQL> @cesta_k_balíku\cert_auth_api.sql;
SQL> @cesta_k_balíku\cert_auth_api.pkb;
```

Tento balík představuje samotné API zabezpečení autentizace certifikáty. Obsahuje funkci `verify_user_cert`, která kontroluje certifikát, který předtím prošel vrstvou SSL. Tato funkce vrací `true`, pokud je získaný certifikát od uživatele zaregistrovaný (má záznam v tabulce `CERT_AUTH_USER_T`) s uživatelským jménem, kterým se uživatel přihlásil do databáze a s aplikací, do které se přihlašuje.

Volání této funkce pro PL/SQL aplikace:

```
result := verify_user_cert('EXAM_APP', '', ernno);
```

Poznámka:

První parametr udává jméno aplikace, druhý jméno uživatele a třetí je výstupní – číslo chyby. Jméno uživatele není pro PL/SQL aplikace důležité, protože si jej funkce sama zjistí z hodnoty systémové proměnné `REMOTE_USER`.

Volání této funkce pro Oracle Portal:

```
result := verify_user_cert('PORTAL', 'jstein', ernno);
```

Poznámka:

První parametr musí být klíčové slovo PORTAL, druhý je jméno uživatele zadané v přihlašovacím dialogu login serveru a třetí udává opět vrácené číslo chyby.

Na začátku si funkce zjistí hodnoty systémových proměnných SSL_CLIENT_CERT, SSL_CLIENT_I_DN, SSL_CLIENT_S_DN a SSL_VERIFY (používá funkci get_cgi_env z balíku owa_util):

```
ssl_cert_data.ssl_client_cert := owa_util.get_cgi_env('SSL_CLIENT_CERT');
ssl_cert_data.ssl_client_i_dn := owa_util.get_cgi_env('SSL_CLIENT_I_DN');
ssl_cert_data.ssl_client_s_dn := owa_util.get_cgi_env('SSL_CLIENT_S_DN');
ssl_verify  := owa_util.get_cgi_env('SSL_CLIENT_VERIFY');
```

Potom načte uživatelské jméno – u portálu přímo z parametru funkce a u PL/SQL aplikací ze systémové proměnné REMOTE_USER:

```
if (compare_varchar2(p_app_name,'PORTAL') = 0) then
      user := p_user_name;
else
      user := owa_util.get_cgi_env('REMOTE_USER');
end if;
```

Dále načte konfiguraci aplikace (metodu kontroly certifikátu) z konfigurační tabulky[32]:

```
Open plist for select * from cert_auth_config_t where (p_app_name=app_name);
fetch plist into config;
if (plist%notfound) then
      errno := 1;
      return (false);
end if;
```

Pak načte záznam certifikátu (pokud se jedná o PLSQL aplikaci, tak z tabulky CERT_AUTH_USER_T, a v případě portálu z tabulky WWSSO_USERCERT_T):

[32] Pokud není nalezen konfigurační záznam, funkce končí a v proměnné errno vrací číslo chyby 1.

```
if (compare_varchar2(p_app_name,'PORTAL') = 0) then
      open plist for select ssl_client_cert, ssl_client_i_dn,
            ssl_client_s_dn from portal30_sso.wwsso_usercert_t where
            user_name = user;
      fetch plist into db_cert_data;
      if (plist%notfound) then
            errno := 2;
            return (false);
      end if;
else
      open plist for select ssl_client_cert, ssl_client_i_dn,
            ssl_client_s_dn from cert_auth_user_t where (user_name = user)
            and (p_app_name = app_name);
      fetch plist into db_cert_data;
      if (plist%notfound) then
        errno := 2;
        return (false);
      end if;
end if;
```

Následuje kontrola, zda verifikace vrstvy SSL proběhla úspěšně – systémová proměnná `SSL_VERIFY` musí mít hodnotu `SUCCESS`. Nakonec se zkontroluje shoda certifikátových proměnných načtených z databáze a proměnných získaných z vrstvy SSL. To, které proměnné se budou kontrolovat, udává konfigurační proměnná `config`.

Kontrola `ssl_client_s_dn` (DN klienta) proběhne vždy:

```
if (compare_varchar2(db_cert_data.ssl_client_s_dn,
            ssl_cert_data.ssl_client_s_dn) = 1) then
      errno := 4;
      return (false);
end if;
```

Kontrola `ssl_client_i_dn` (DN vydavatele certifikátu) proběhne pokud má konfigurační parametr hodnotu `'ALL'` nebo `'CLIENT_AND_ISSUER_DN'`:

```
if ((compare_varchar2(config.compare_type, 'ALL') = 0) or
      (compare_varchar2(config.compare_type, 'CLIENT_AND_ISSUER_DN') = 0))
            then
```

```
      if (compare_varchar2(db_cert_data.ssl_client_i_dn,
                  ssl_cert_data.ssl_client_i_dn) = 1) then
            errno := 5;
            return (false);
      end if;
end if;
```

Kontrola `ssl_client_cert` (vlastního certifikátu) proběhne pokud má konfigurační parametr hodnotu `'ALL'`.

```
if (compare_varchar2(config.compare_type, 'ALL') = 0) then

      if (compare_varchar2(db_cert_data.ssl_client_cert,
                  ssl_cert_data.ssl_client_cert) = 1) then
            errno := 6;
            return (false);
      end if;
end if;
```

V případě úspěchu všech kontrol vrací funkce hodnotu `TRUE`.

8.2.4. Konfigurace CERT_AUTH_API

K uložení konfigurace API slouží tabulka `cert_auth_config_t`, kterou uložíme ve schématu `CERT_AUTH_ADMIN`:

```
SQL > create table cert_auth_config_t(
  app_name varchar(30) not NULL,    -- jméno aplikace
  compare_type varchar2(30),        -- typ porovnávání
  constraint pk_cert_auth_config_t primary key ( app_name )
);
```

Pro každou aplikaci se potom musí nakonfigurovat typ porovnávání. K tomu slouží funkce `configure`, která se nachází v balíku `CERT_AUTH_API`. Vrací `TRUE` v případě úspěšné konfigurace. Tato funkce má 3 parametry – jméno aplikace, typ porovnávání a číslo chyby (výstupní parametr). Jako typ porovnávání můžeme zvolit tři hodnoty:

CLIENT_DN_ONLY

Toto je nejnižší úroveň bezpečnosti, při níž se porovnává pouze DN (*Distinguish Name*) vlastníka certifikátu. Při této konfiguraci se může uživatel prezentovat jakýmkoliv certifikátem s DN vlastníka, které má zaregistrováno v databázi. Výhodou tohoto nastavení je, že pokud vyprší platnost certifikátu a uživateli je vydán následný certifikát, správce aplikace nemusí nic měnit v databázi. Na druhé straně je zde nevýhoda ve snížení úrovně bezpečnosti, protože uživateli stačí obstarat si certifikát s požadovaným DN. Ovšem zde je podmínka, aby tento certifikát vydala certifikační autorita, které server důvěřuje.

CLIENT_AND_ISSUER_DN

Střední úroveň bezpečnosti, při níž se porovnává DN vlastníka certifikátu i DN vydavatele certifikátu. Pokud nastavíme tuto úroveň bezpečnosti, musí se uživatel při přihlášení prezentovat certifikátem, který má DN vlastníka i DN vydavatele stejnou, jako je uložena v databázi. Při této konfiguraci můžeme opět využít výhodu možnosti vydání následného certifikátu po vypršení platnosti bez zásahu do databáze. Útočník by si musel obstarat certifikát od stejné certifikační autority a se stejným DN vlastníka, což většina CA nepovoluje (DN vlastníka musí být v rámci jedné CA unikátní – platí u OpenSSL). Doporučuje se používat tuto konfiguraci.

ALL

Nejvyšší úroveň bezpečnosti, kdy se porovnává i vlastní certifikát (PEM kódovaný). Přístup je umožněn jen tomu, kdo vlastní certifikát, který je zaregistrován v databázi pro uživatelské jméno, kterým se uživatel přihlásil. Vysoká bezpečnost je zde kompenzována nutností zásahu do databáze v případě, že vyprší platnost certifikátu – je nutné starý záznam vymazat a nahradit jej novým. Tato úroveň se doporučuje v aplikacích, kde je opravdu nutná maximální bezpečnost nebo nepopiratelnost (z právního hlediska).

Příklad konfigurace aplikace `'EXAMPLE_APP'` na střední úroveň bezpečnosti (spouští administrátor certifikátové autentizace – `CERT_AUTH_ADMIN`):

```
SQL> declare
  ret_val boolean;
  errno number;
```

```
begin
  ret_val := cert_auth_api.configure('EXAMPLE_APP','CLIENT_AND_ISSUER_DN',
                errno);
  if (ret_val) then
    dbms_output.put_line('CERT_AUTH successfully configured');
  else
    dbms_output.put_line('Error: ' || errno);
  end if;
end;
/
```

8.3. Demonstrační aplikace autentizace certifikáty v PL/SQL

V rámci této diplomové práce byla vytvořena PL/SQL aplikace demonstrující zabezpečení autentizace certifikáty. Tato podkapitola popisuje její instalaci, konfiguraci a testování bezpečnosti.

8.3.1. Konfigurační tabulka aplikace

Pro uložení konfigurace aplikace slouží pomocná tabulka `CERT_AUTH_EX_CONFIG_T`:

```
SQL> create table cert_auth_ex_config_t(
  reg_type varchar2(30),           -- typ registrace
  pls_url_path varchar2(200),      -- url cesta k balíku CERT_AUTH_EXAMPLE
  ps_url_path varchar2(200),       -- url stránky veřejné sekce
  constraint pk_cert_auth_ex_config_t primary key ( reg_type )
);
```

Výhody tohoto řešení:

- nezávislost na URL aplikace (absolutní URL je zadáno až při konfiguraci aplikace – ne v kódu)
- možnost nastavení typu registrace – 2 možnosti
  - `'USER_REG'` – uživatelé si mohou zaregistrovat svůj certifikát v aplikaci
  - `'ADMIN_REG'` – registraci smí provádět pouze administrátor

8.3.2. PL/SQL balík CERT_AUTH_EXAMPLE

Aplikace je uložena v balíku `CERT_AUTH_EXAMPLE`. Jeho deklaraci a tělo vložíme do schématu `CERT_AUTH_ADMIN`:

```
SQL> @cesta_k_balíku\cert_auth_example.sql;
SQL> @cesta_k_balíku\cert_auth_example.pkb;
```

Procedury a funkce v tomto balíku:

- `print_error` – procedura, která vypisuje stránku s chybovým hlášením
- `app_page` – hlavní zabezpečená stránka aplikace
- `html_header` – html hlavička aplikace
- `html_footer` – html patička aplikace
- `html_ref` – html odkaz
- `cli_cert_info` – zabezpečená stránka s informacemi o certifikátu uživatele
- `ser_cert_info` – zabezpečená stránka s informacemi o certifikátu serveru
- `ssl_info` – zabezpečená stránka s informacemi o vrstvě SSL
- `register_cert` – procedura registrující certifikát uživatele s aplikací a jeho uživatelským jménem
- `configure` – konfigurační procedura aplikace

Poznámka:

Zabezpečená stránka aplikace znamená, že pro přístup na tuto stránku musí mít uživatel konto v databázi a certifikát zaregistrovaný se svým uživatelským jménem. Programově to znamená, že na začátku procedury generující tuto stránku je kontrola uživatelova certifikátu funkcí `verify_user_cert` z API `cert_auth_api`.

Pro možnost spouštění aplikace každým uživatelem databáze vytvoří administrátor veřejné synonymum:

```
SQL> create public synonym cert_auth_example for
            cert_auth_admin.cert_auth_example;
```

8.3.3. Konfigurace demonstrační aplikace

Konfiguraci aplikace provede administrátor aplikace následujícím způsobem:

```
SQL> declare
  ret_val boolean;
  errno number;
begin
  ret_val := cert_auth_example.configure('USER_REG',
             'https://steinsof-ifr1fh/pls/alfa', 'http://steinsof-ifr1fh',
             errno);
  if (ret_val) then
    dbms_output.put_line('CERT_AUTH_EXAMPLE successfully configured');
  else
    dbms_output.put_line('Error: ' || errno);
  end if;
end;
/
```

8.3.4. Vytvoření uživatele

Administrátor aplikace (v našem případě `CERT_AUTH_ADMIN`) vytvoří uživatele následujícím způsobem:

Vytvoří konto v databázi:

```
SQL> create user josef_steinberger identified by josef_steinberger
        default tablespace user_data temporary tablespace temporary_data;
```

Přidělení práv:

```
SQL> grant connect, execute any procedure, select any table to
        josef_steinberger;
```

Registrace certifikátu uživatele (data certifikátu – tedy vlastní certifikát, DN vydavatele certifikátu i DN vlastníka – musí být administrátorovi známa):

```
insert into cert_auth_user_t values ('JOSEF_STEINBERGER', 'EXAMPLE_APP',
'-----BEGIN CERTIFICATE-----
MIIDozCCAwygAwIBAgIBAzANBgkqhkiG9w0BAQUFADCBpTELMAkGA1UEBhMCQ1ox
FzAVBgNVBAgTDkN6ZWNoIFJlcHVibGljMQ4wDAYDVQQHEwVQbHplbjEXMBUGA1UE
ChMOQ0NBIEdyb3VwIGEucy4xIzAhBgNVBAsTGm9kZC4gdnl2b2plIGEgcHJvZ3Jh
bW92YW5pMQ8wDQYDVQQDEwZDQ0EgQ0ExHjAcBgkqhkiG9w0BCQEWD2NhX2FkbWlu
QGNjYS5jejAeFw0wMzA0MTYwOTA5MjdaFw0wNDA0MTUwOTA5MjdaMFQxCzAJBgNV
BAYTAkNaMRcwFQYDVQQIEw5DemVjaCBSZXB1YmxpYzEQMA4GA1UEChMHWkNVLUZB
VjEaMBgGA1UEAxMRSm9zZWYgU3RlaW5iZXJnZXIwgZ8wDQYJKoZIhvcNAQEBBQAD
gY0AMIGJAoGBAN7tdkuou/doYBsrdpSQL5jDNubeJd7JAR6AHrAAhUpCbGnAaoZ9
WYwVecjlMZWNIUi9cm+SEaKgTZsCyf9+stdD2E8Cg4PL5kre+KfeduEjG9nWhUOS
sIpJcYZmSmKRBx49GX+apwonzEAuEmXjOuY3wajWkZTVfyI/v0w7ZCnJAgMBAAGj
ggExMIIBLTAJBgNVHRMEAjAAMCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVy
YXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4EFgQUwOGxjFjCtONCJG+wcLtPFd671zcw
gdIGA1UdIwSByjCBx4AUKI/sZ4xtVoXHcZ5Cm1NsaaR4x0GhgaukgagwgaUxCzAJ
BgNVBAYTAkNaMRcwFQYDVQQIEw5DemVjaCBSZXB1YmxpYzEOMAwGA1UEBxMFUGx6
ZW4xFzAVBgNVBAoTDkNDQSBHcm91cCBhLnMuMSMwIQYDVQQLExpvZGQuIHZ5dm9q
ZSBhIHByb2dyYW1vdmFuaTEPMA0GA1UEAxMGQ0NBIENBMR4wHAYJKoZIhvcNAQkB
Fg9jYV9hZG1pbkBjY2EuY3qCAQAwDQYJKoZIhvcNAQEFBQADgYEAIi5FUujk/Ehs
bvE+p2e5Cxgg3G4pIoZZ+sxkoLKe7tz+inBuiC++Sl0ZBVNQq9uJhuO4yfS5IH9j
sQG7z9nxwDC02ukiyzOOijwtrdmTQUbk3KARiXyaw984ADD70GXZA+hOeTDND+by
h52HiCuWuOXtk+GvtQVADrq0qIvgPB8=
-----END CERTIFICATE-----
','/C=CZ/ST=Czech Republic/L=Plzen/O=CCA Group a.s./OU=odd. vyvoje a
programovani/CN=CCA CA/Email=ca_admin@cca.cz','/C=CZ/ST=Czech
Republic/O=ZCU-FAV/CN=Josef Steinberger');
```

8.3.5. Registrace certifikátu uživatele

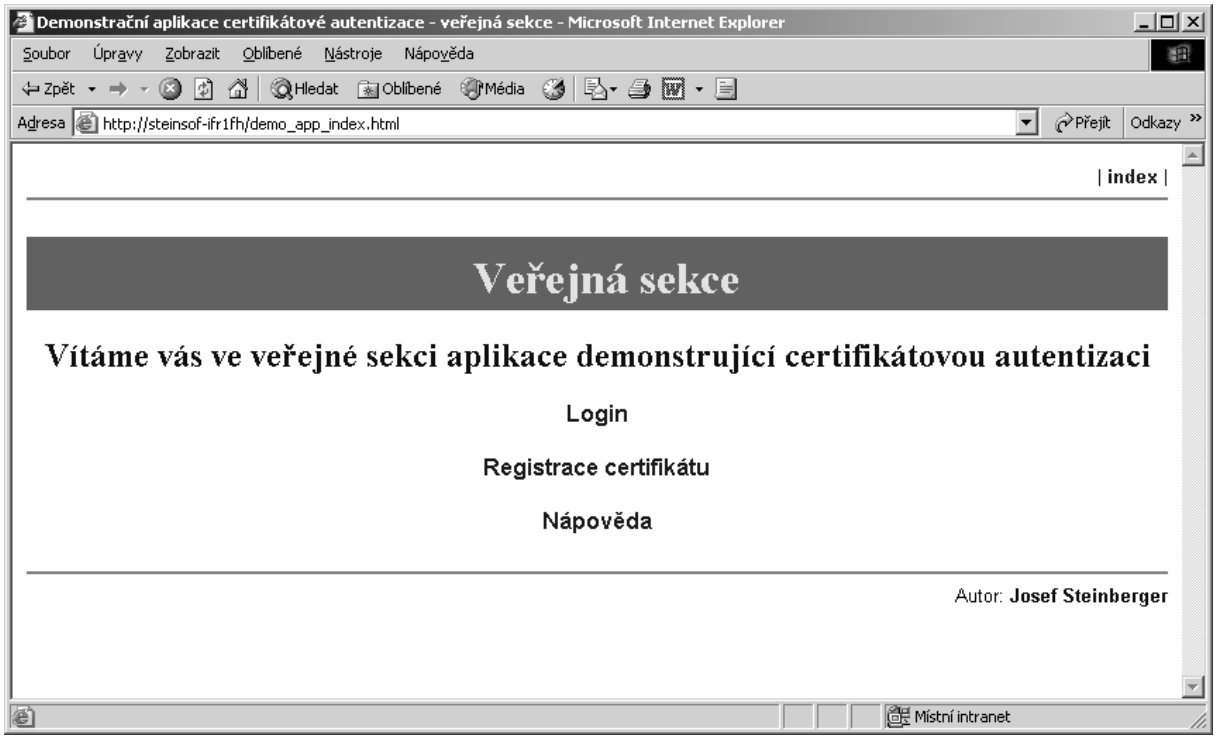
Aplikace může být nakonfigurována na uživatelskou registraci (při spuštění konfigurační procedury bude mít první parametr hodnotu `'USER_REG'`). Znamená to, že uživatelé si mohou svůj certifikát zaregistrovat sami (odkazem v aplikaci). Výhoda tohoto řešení spočívá v tom, že administrátor nemusí zjišťovat data o certifikátu a poté jej registrovat v databázi (viz předchozí podkapitola 8.3.4). Tato výhoda je však kompenzována sníženou úrovní

bezpečnosti, protože na registrovaný certifikát je kladena jediná podmínka, aby byl vydán certifikační autoritou, které server důvěřuje.

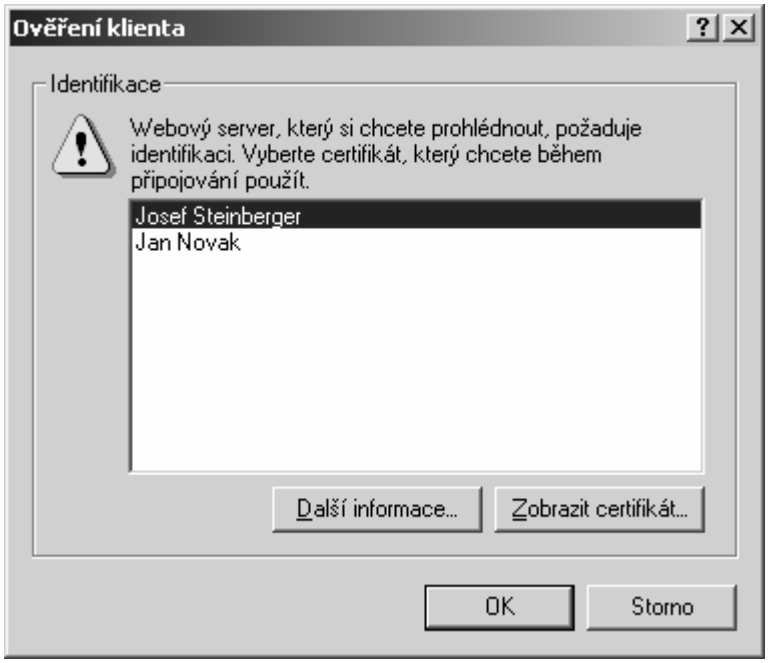
Druhou možností je povolit registaci pouze administrátorem aplikace (při spuštění konfigurační procedury bude mít první parametr hodnotu `'ADMIN_REG'`). Vyšší úroveň bezpečnosti je zde kompenzována nutností získání dat certifikátu.

8.3.6. Testování zabezpečení autentizace aplikace

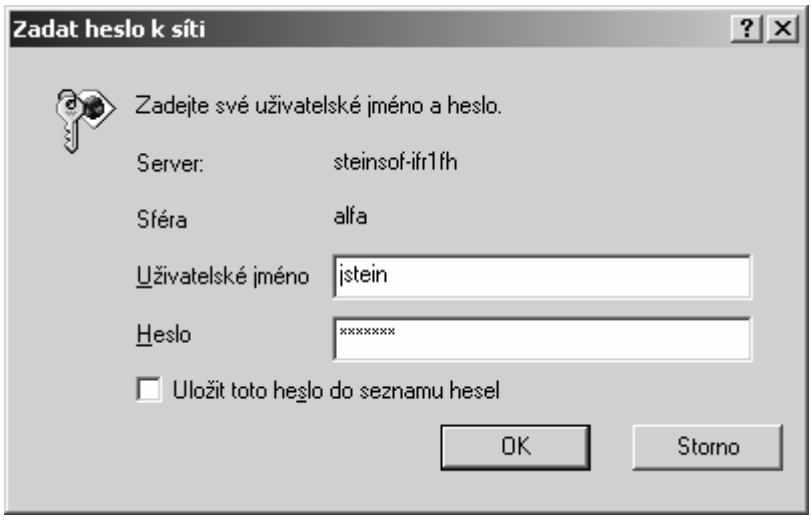
Hlavní stránka aplikace (soubor `demo_app_index.html`), na kterou mají přístup všichni uživatelé:



Po stisknutí odkazu `Login` uživatel vybere certifikát, kterým se bude prezentovat[33] (pro potřeby ověření autentizace byly vytvořeny dva certifikáty):



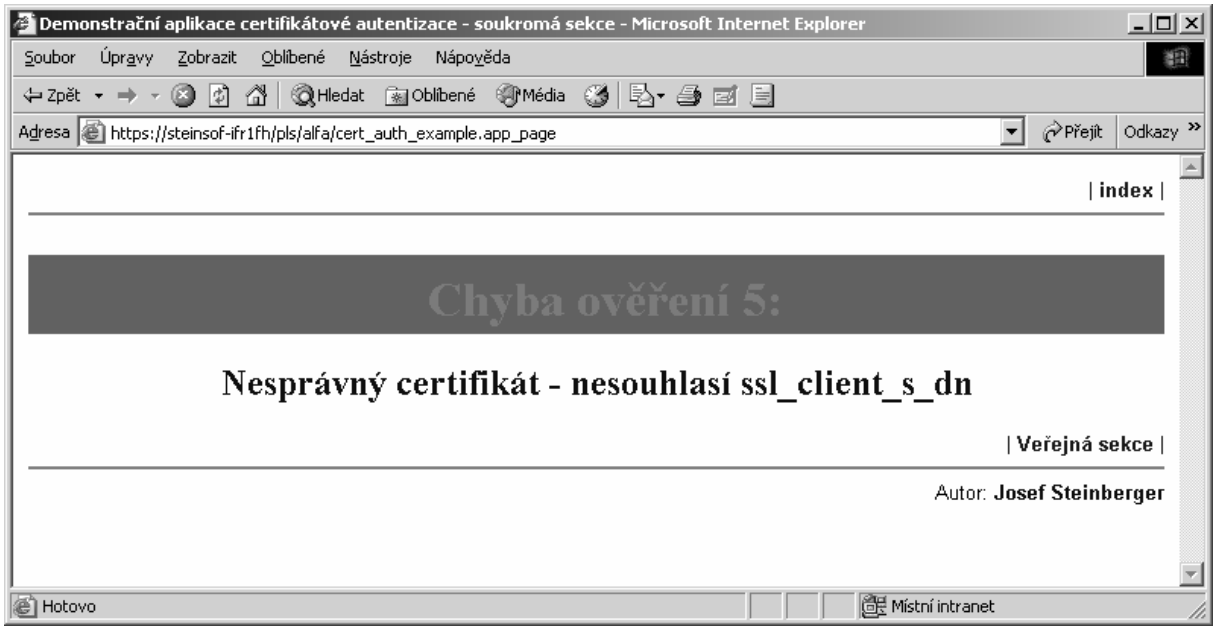
Potom se přihlásí do databáze:



[33] To platí pro uživatele, kteří mají v prohlížeči nainstalováno více certifikátů od certifikačních autorit, kterým server důvěřuje. Pokud mají pouze jeden, je zvolen automaticky.

Pokud proběhne kontrola certifikátu v pořádku, uživatel se dostane do vlastní aplikace (soukromá sekce). V této sekci lze nalézt informace o SSL a použitých certifikátech:

Pokud uživatel nezvolí žádný certifikát (stiskne tlačítko *Storno* v dialogu *Ověření klienta* – chyba ověření 4 – certifikát nebyl ověřen vrstvou SSL) nebo zvolí jiný certifikát, který nemá zaregistrován se svým uživatelským jménem do databáze, aplikace zobrazí stránku s chybou:



Uživatelská registrace:

Po stisknutí odkazu *Registrace certifikátu* z veřejné sekce aplikace, je uživatel vyzván k výběru certifikátu a k zadání jména a hesla do databáze. Pokud zadané konto nemá v databázi (tabulce `cert_auth_user_t`) záznam o certifikátu, jsou data ze zvoleného certifikátu do databáze vložena[34]. Pokud již záznam v databázi existuje, je zobrazena stránka s chybovým hlášením.

8.4. Zabezpečení autentizace certifikáty v Oracle Portal

8.4.1. Single Sign-On

Single sign-on je komponenta Oracle Portal, která uživatelům umožňuje přihlášení do několika webových aplikací najednou jedním uživatelským jménem a heslem.

Single sign-on přináší následující výhody:

- redukování administrativních nákladů spojených s podporou mnohonásobných kont a hesel pro každého uživatele
- „pohodlnost“ z pohledu uživatele – nemusí udržovat uživatelská jména a hesla pro každou aplikaci, ke které přistupují
- zvýšená bezpečnost, protože pokud je heslo požadováno pouze jednou, uživatelé méně raději volí jednoduchá, lehce zapamatovatelná hesla a méně si je někam poznamenávají.

8.4.1.1. Komponenty Single Sign-On

Single Sign-On se skládá z následujících komponent:

- Login Server
- partnerské aplikace
- externí aplikace

[34] Pokud není povolena uživatelská registrace (viz Registrace certifikátu uživatele 8.3.5), je zobrazena stránka s chybou.

Login Server

Login server je jádro single sign-on technologie. Pro partnerské aplikace login server autentizuje uživatele a posílá jejich identitu do aplikací. Pro externí aplikace, které nepoužívají autentizační mechanismus login serveru, umožňuje single sign-on autentizaci přes centralizované úložiště hesel.

Když se uživatel poprvé pokouší přihlásit do single sign-on umožňující aplikace, login server:

- autentizuje uživatele získáním a ověřením uživatelského jména a hesla
- ukládá zašifrovaný login cookie do prohlížeče klienta
- posílá identitu klienta do aplikace

Při následujících přihlášeních login cookie indikuje login serveru, že autentizace již byla provedena. Pokud login cookie není přítomen (například pokud vypršela jeho platnost), login server zobrazí uživateli login stránku. Pro ochranu proti odposlechu může login server poslat login cookie do prohlížeče klienta přes šifrovaný SSL kanál. Platnost končí buď ukončením intervalu specifikovaného administrátorem nebo ukončením prohlížeče. Login cookie se nikdy nezapisuje na disk.

Partnerské aplikace

Partnerské aplikace jsou těsně integrovány s login serverem. Delegují autentizaci do login serveru, kde se uživatelé mohou přihlásit prostřednictvím mechanismu single sign-on.
Příkladem partnerské aplikace je sám Oracle Portal.

Externí aplikace

Externí aplikace mají svou vlastní autentizační logiku – tedy nedelegují autentizaci do login serveru. K umožnění přístupu vyžadují aplikačně-specifická uživatelská jména a hesla. Tyto uživatelská jména mohou být různá od single sign-on uživatelských jmen, na které je login server mapuje.

8.4.1.2. Single Sign-On autentizační metody

Single Sign-On užívá následující autentizační metody:

- lokální autentizaci (*Local User Authentication*)
- externí autentizaci (*External Repository Authentication*)

Lokální autentizace

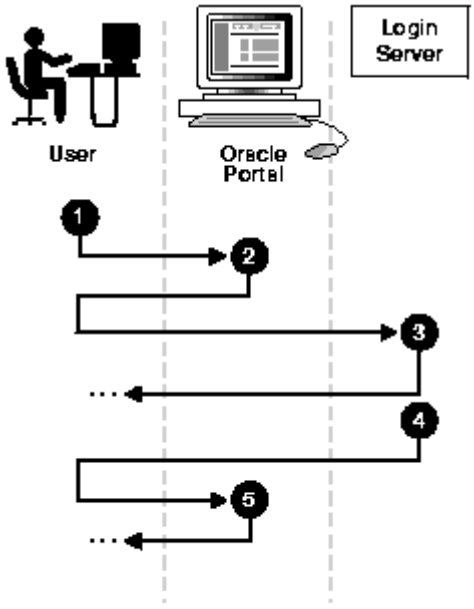
Lokální autentizace používá autentizační tabulku ve schematu login serveru (*portal30_sso*) v databázi asociované s Oracle Portal (*alfa*). Tato tabulka obsahuje jména, hesla a privilegia uživatelů. Zadané heslo v login formuláři je jednosměrně hešováno a porovnáno se záznamem v tabulce.

Externí autentizace

Externí autentizace typicky závisí na LDAP adresáři – specificky *Oracle Internet Directory* (OID). V tomto případě se login server spojí s OID a zjistí existenci záznamu uživatele (uživatelské jméno a správné heslo) v OID.

8.4.1.3. Autentizační proces Oracle Portal

1. Uživatel přistupuje na stránku Oracle Portal.
2. Pokud to je poprvé v sezení, kdy uživatel přistupuje k Oracle Portal, je transparentně přesměrován do login serveru pro získání autentizačních dat (uživatelského jména a hesla).
3. Login server autentizuje uživatele, jak již bylo popsáno v části 8.4.1.1.
4. Login server transparentně přesměruje uživatele do Oracle portal. Používá k tomu URL se zašifrovaným parametrem, který obsahuje identitu uživatele.
5. Oracle Portal:
    - dešifruje parametr
    - identifikuje uživatele
    - vytvoří vlastní management sezení
    - prezentuje uživateli odkazy na externí aplikace



8.4.2. Způsob zabezpečení přístupu do Oracle Portal

`cert_auth_api` popsané v podkapitole 8.2, lze využít i pro zabezpečení přístupu do Oracle Portal. Řešení je jednoduché: nakonfigurujeme login server, aby používal SSL, vytvoříme tabulku certifikátů uživatelů a do funkce, která kontroluje zadané uživatelské jméno a heslo, přidáme kontrolu certifikátu. V přidání kontroly certifikátu je problém. Kontrolu uživatelského jména a hesla řídí v portálu procedura `authenticate_user`, která se nachází v balíku `WWSSO_LS_PRIVATE` (ve schématu `portal30_sso`). Problémem je, že tělo tohoto balíku je ***wrappované***[35]. Toto lze obejít externí autentizací přes OID. V balíku externí autentizace (`WWSSO_AUTH_EXTERNAL`) je také procedura `authenticate_user`. Tělo tohoto balíku společnost Oracle distribuuje ve wrappované ale i v PL/SQL verzi. Musíme tedy nakonfigurovat login server na externí autentizaci a pozměnit tělo procedury `authenticate_user`.

8.4.3. Konfigurace Oracle Portal pro použití HTTPS

Je možné nastavit systém tak, aby jen login server byl nakonfigurován na HTTPS, anebo aby HTTPS používal Oracle Portal i login server. Kapitola 7 této práce popisuje konfiguraci serveru na podporu HTTPS. Po nakonfigurování serveru spustíme skript `ssodatan` (nebo `ssodatax`), který nastaví požadované protokoly a porty. Pro případ zabezpečení autentizace stačí nakonfigurovat pouze login server na HTTPS. Portál může být samozřejmě také nakonfigurován na protokol HTTPS, ale snižuje to jeho výkonnost.
Příklad spuštění skriptu `ssodatan`:

```
ssodatan.cmd -w http://steinsof-ifr1fh/pls/portal30/ -l https://steinsof-
ifr1fh/pls/portal30_sso/ -s portal30 -p portal30 -o portal30_sso -d
portal30_sso -c alfa.cca.cz
```

Parametry skriptu:

- `-w` URL portálu
- `-l` URL login serveru
- `-s` schéma portálu
- `-p` databázové heslo schématu portálu

[35] *Wrappování* znamená převedení PL/SQL kódu do objektové formy, která je nezávislá na platformě a je nečitelná.

`-o` schéma login serveru

`-d` databázové heslo schématu login serveru

`-c` řetězec spojení (*connect string*) pro databázi, ve které je portál nainstalován.

8.4.4. Konfigurace externí autentizace Login serveru

Z důvodu existence newrappovaného balíku `WWSSO_AUTH_EXTERNAL` (diskutováno v části 8.4.2) musíme nakonfigurovat login server na externí autentizaci přes OID. Postup je následující:

1. Přepneme se do adresáře Oracle home databáze, kde je nainstalován login server (v mém případě `OH_RDBMS8I`).
2. Přepneme se do adresáře `rdbms/admin`.
3. Přihlásíme se do SQL*Plus jako `sys`:
   `sqlplus sys@alfa/`*`heslo_sys`*[36]
4. Spustíme skript `catldap.sql`, který nainstaluje potřebné balíky pro LDAP:
   `SQL> @catldap.sql`
5. Ukončíme SQL*Plus a přepneme se do adresáře zdrojových skriptů portálu (v mém případě `C\OH_IAS102\portal30\admin\plsql\sso`).
6. Přihlásíme se do databáze jako schéma login serveru:
   `sqlplus portal30_sso@alfa/`*`heslo_portal30_sso`*
7. Spustíme skript `ssooid`, který nainstaluje vše potřebné pro umožnění externí autentizace (hlavně balík `WWSSO_AUTH_EXTERNAL`):
   Po spuštění musíme zadat následující hodnoty:
   - jméno serveru
   - port OID
   - bázi vyhledávání (*search base*) v adresářové struktuře OID, kde jsou umístěny záznamy uživatelů
   - unikátní atribut (klíč vyhledávání)
   - uživatelské jméno a heslo konta, které má právo hledání v části OID, kde jsou uloženy záznamy uživatelů

   Příklad nastavení:

   ```
   Zadejte hodnotu pro host: steinsof-ifr1fh
   ```

[36] `Alfa` je řetězec připojení (*connect string*) do databáze.

```
Zadejte hodnotu pro port: 389
Zadejte hodnotu pro search_base: cn=Login Server (portal30_sso)
Zadejte hodnotu pro unique_attribute: cn
Zadejte hodnotu pro bind_dn: cn=orcladmin
Zadejte hodnotu pro bind_password: welcome
```

Pro umožnění přihlášení uživatelům, kteří mají konta v portálu, musíme převést jejich konta do OID:

8. Spustíme skript `ssoldif` (stále jsme přihlášeni jako `portal30_sso`). Tento skript vygeneruje soubor `users.ldif` s uživateli portálu[37].

9. Informace ve vygenerovaném souboru vložíme do OID následujícím příkazem:
   ```
   ldapadd -h <host> -p <port> -D <bind_DN> -w <bind_password> -f
         users.ldif
   ```
   Konkrétní příklad:
   ```
   ldapadd -h steinsof-ifr1fh -p 389 –D cn=orcladmin -w welcome -f
         users.ldif
   ```

8.4.5. Tabulka WWSSO_USERCERT_T

Pro kontrolu certifikátu při autentizaci uživatele musíme vytvořit tabulku uživatelů a jejich certifikátů. Úmyslně nejsou certifikáty uživatelů portálu ukládány do stejné tabulky jako v případě PL/SQL aplikací, protože přístup do této tabulky by měl mít pouze administrátor single sign-on (`portal30_sso`).

```
create table wwsso_usercert_t(
  user_name varchar(30) not NULL,
  ssl_client_cert varchar2(2000),
  ssl_client_i_dn varchar2(200),
  ssl_client_s_dn varchar2(200),
  constraint pk_cert_auth_user_t primary KEY ( user_name )
);
```

[37] Po spuštění skriptu jsme dotázáni na adresář (a cestu k němu), kam se má vygenerovaný soubor uložit. Tento adresář musí být v parametru `utl_file_dir` inicializačního souboru databáze (zde `initALFA.ora`), například:

```
utl_file_dir = C:\OH_IAS102\Apache\Apache\utl_file_dir
```

8.4.6. Přidání kontroly certifikátu do balíku WWSSO_AUTH_EXTERNAL

Do funkce `authenticate_user` v balíku `WWSSO_AUTH_EXTERNAL` musíme přidat kontrolu certifikátu[38]. Využijeme zde funkci `verify_user_cert` z balíku `cert_auth_api`:

```
function authenticate_user(p_user in varchar2, p_password in varchar2)
      return pls_integer
is
      ...
begin
      ...
      if (cert_auth_admin.cert_auth_api.verify_user_cert('PORTAL', p_user,
            errno)) then
            l_result := EXT_AUTH_SUCCESS;
      else
            raise EXT_AUTH_FAILURE_EXCEPTION;
      end if;
      ...
end authenticate_user;
```

8.4.7. Testování zabezpečení autentizace portálu

Nejprve se nakonfiguruje `cert_auth_api` pro portál (v databázi jsme přihlášeni jako `cert_auth_admin`):

```
declare
  ret_val boolean;
  errno number;
begin
  ret_val := cert_auth_api.configure('PORTAL','ALL', errno);
  if (ret_val) then
    dbms_output.put_line('CERT_AUTH successfully configured');
  else
    dbms_output.put_line('Error: ' || errno);
  end if;
end;
/
```

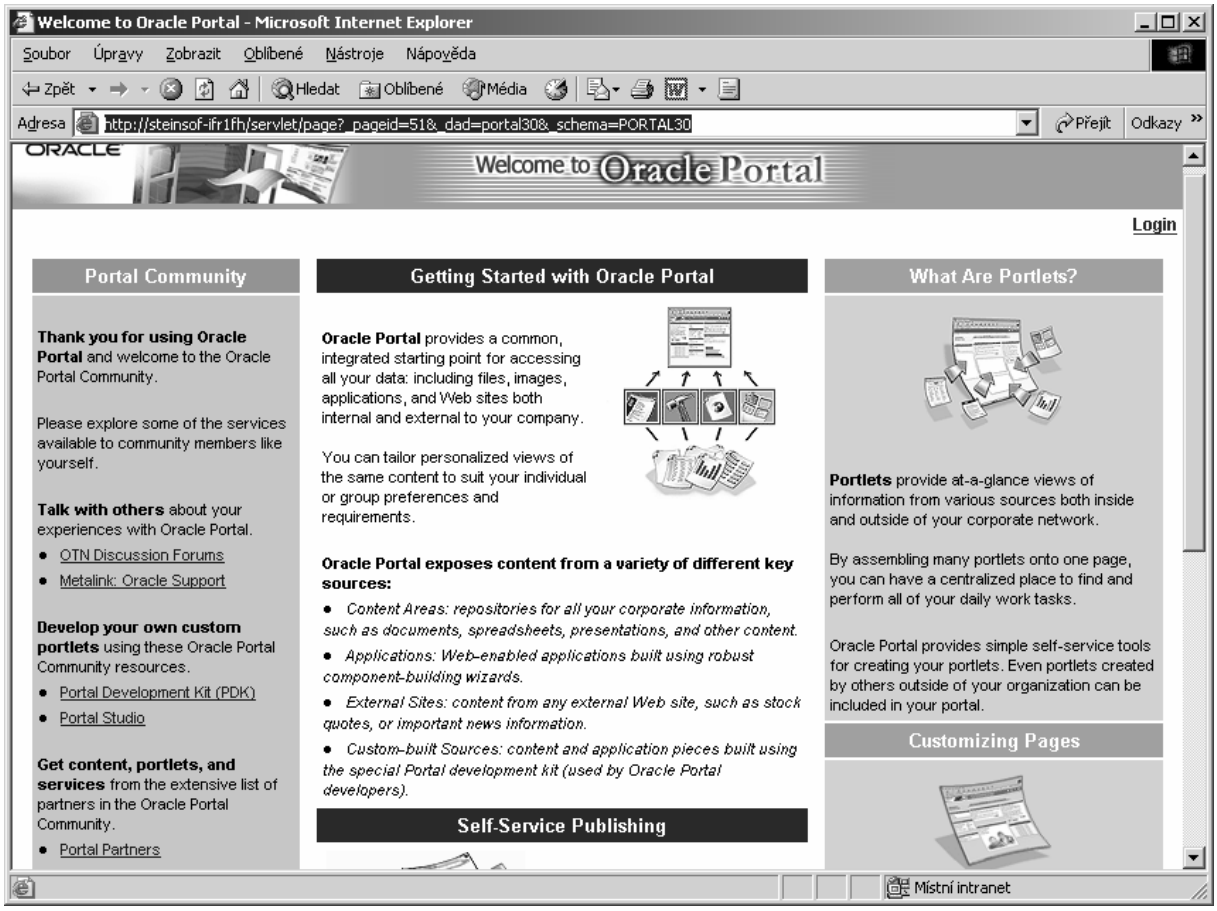
[38] Soubor `ssoxoid.pkb` s tělem tohoto balíku lze nalézt v adresáři *`IAS_HOME`*`\portal30\admin\plsql\sso`.

Do tabulky certifikátů uživatelů portálu (`wwsso_usercert_t`) vloží administrátor login serveru (`portal30_sso`) záznam o certifikátu. Registrace certifikátu je podobná jako administrátorská registrace v PL/SQL aplikacích:

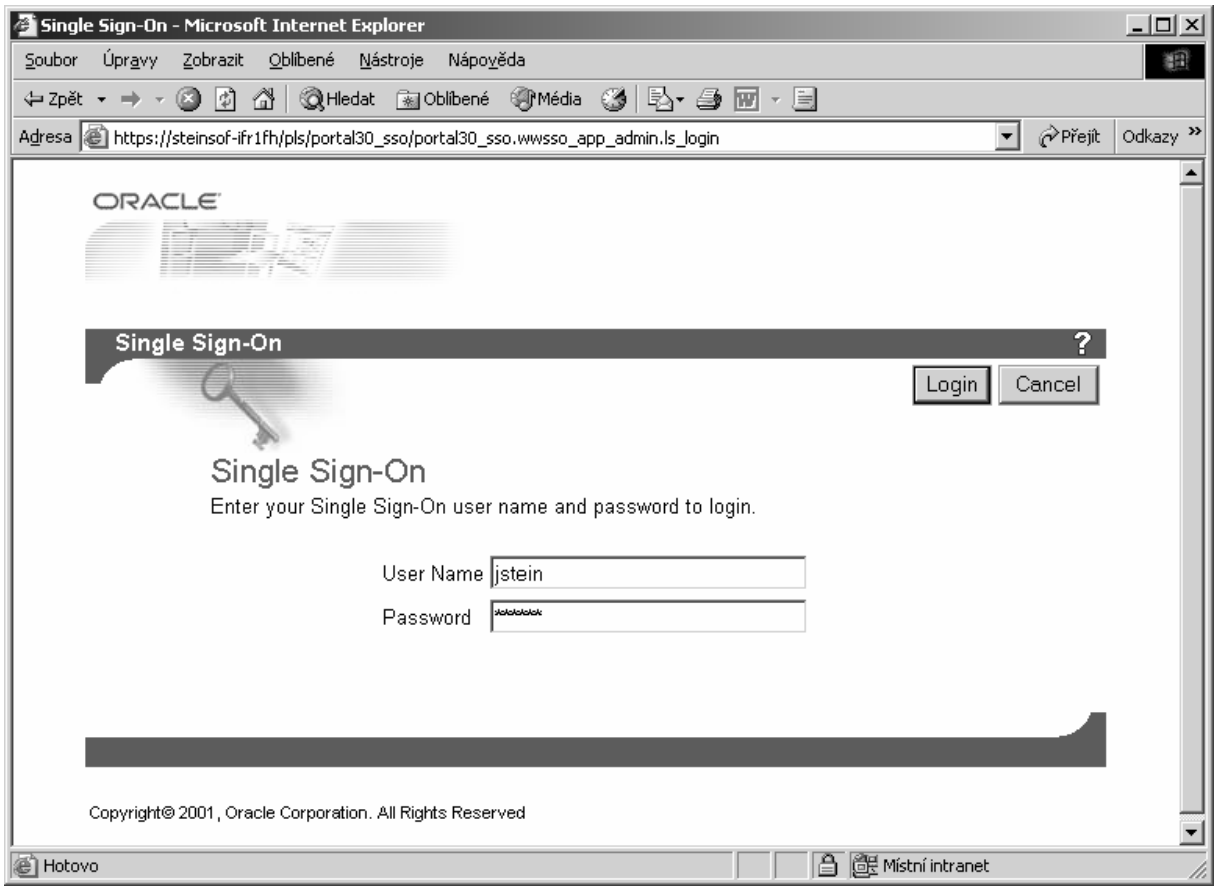
```
insert into wwsso_usercert_t values ('JSTEIN',
'-----BEGIN CERTIFICATE-----
data certifikátu
-----END CERTIFICATE-----
','/C=CZ/ST=Czech Republic/L=Plzen/O=CCA Group a.s./OU=odd. vyvoje a
programovani/CN=CCA CA/Email=ca_admin@cca.cz',
'/C=CZ/ST=Czech Republic/O=ZCU-FAV/CN=Josef Steinberger');
```

Tedy s uživatelským jménem portálu je svázán certifikát vydaný na jméno Josef Steinberger.

V prohlížeči napíšeme URL portálu: `http://steinsof-ifr1fh/pls/portal30`. Objeví se úvodní stránka portálu:

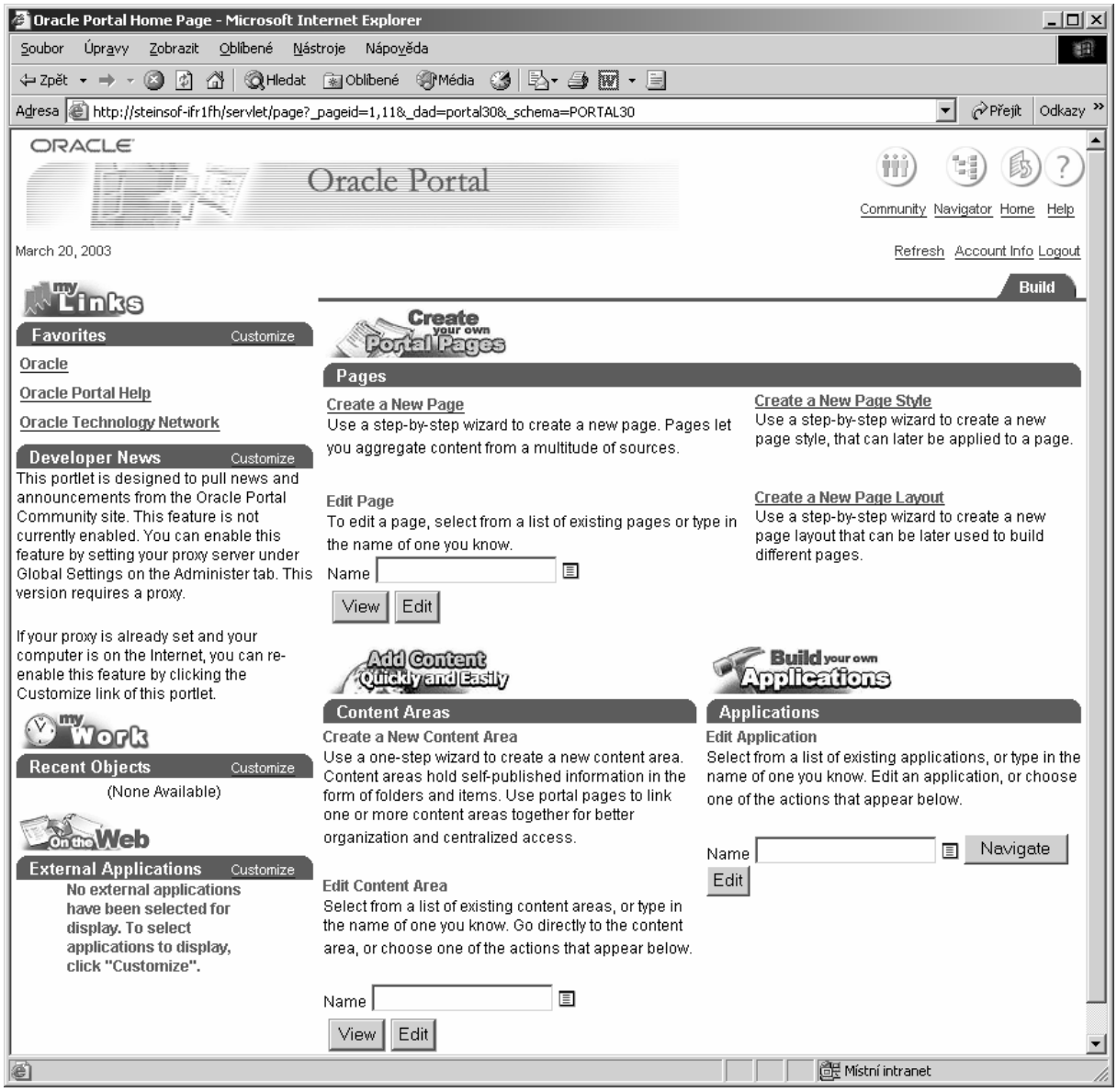


Stiskneme odkaz Login v pravém horním rohu obrazovky. Pokud máme v prohlížeči nainstalováno více certifikátů od certifikačních autorit, kterým server důvěřuje, objeví se dialog pro výběr certifikátu (viz 8.3.6). V našem případě v tomto dialogu zvolíme cerifikát vydaný na jméno Josef Steinberger. V přihlašovacím dialogu vyplníme uživatelské jméno (`jstein`) a heslo:

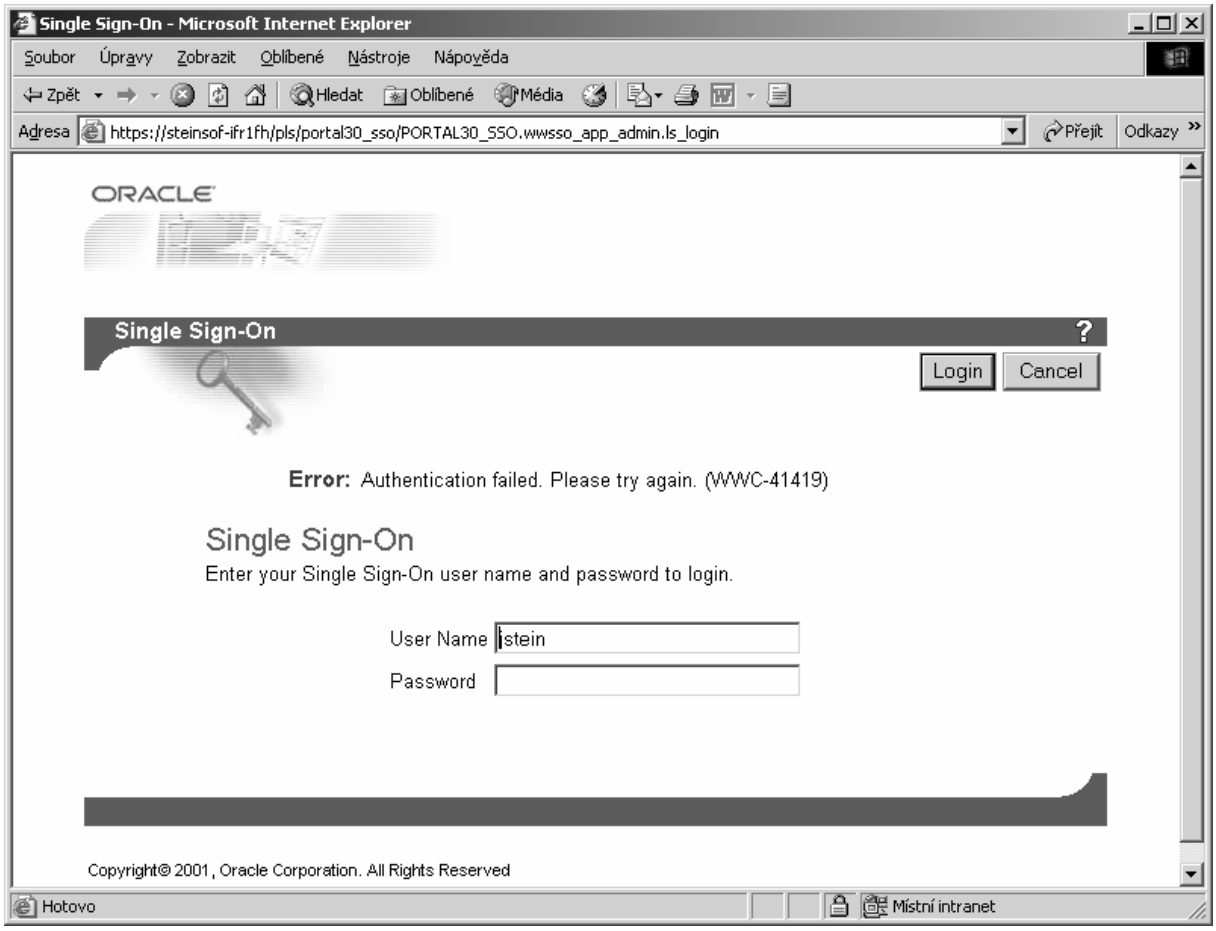


Stiskneme tlačítko *Login*.

Autentizace uživatele proběhla bezchybně, takže se zobrazí úvodní stránka portálu uživatele `jstein`:



Nyní předpokládejme útočníka, který odhalil uživatelské jméno a heslo uživatele `jstein`, avšak nevlastní jeho certifikát (s privátním klíčem). Vlastní pouze svůj certifikát vydaný na jméno (Jan Novák). Po stisku tlačítka *Login* (po vyplnění uživatelského jména a hesla) se zobrazí následující stránka s chybou:



Text chyby – `Authentication failed. Please try again. (WWC-41419)` – je stejný jako v případě zadání chybného hesla (heslo ověřuje OID). Změna textu chyby (například na `Incorrect certificate`) by vyžadovala zásah do wrappovaných balíků login serveru.

9. Podepisování a verifikace emailů

Podepisování emailů nám umožňuje digitálně stvrdit, kdo je odesilatelem. Podpis navíc zaručuje, že zpráva nebyla po jejím podepsání změněna. K podpisu zprávy odesilatel potřebuje svůj privátní klíč, svůj veřejný certifikát a certifikát CA, která mu certifikát vydala. Nejprve se vypočítá výtah zprávy (např. algoritmem SHA), který se potom zašifruje (např. algoritmem RSA) pomocí privátního klíče odesilatele. Takto vzniklý digitální podpis se potom přiloží k emailu jako příloha.

K ověření podpisu musí adresát mít certifikát odesilatele a také příslušný certifikát CA. Tyto certifikáty jsou však veřejné údaje. Naskýtá se tedy možnost zakomponovat je přímo do zprávy. V tomto případě nemusí mít adresát tedy žádný údaj k ověření digitálního podpisu zprávy. Příjemce musí pouze definovat CA, jejichž certifikátům důvěřuje. Pokud dorazí k příjemci podepsaný email, podpis může být automaticky ověřen.

9.1. Specifikace S/MIME

Aby mohl být digitální podpis emailu ověřen v jakémkoliv emailovém klientském programu (tedy i Microsoft Outlook), musí být email ve standardizovaném formátu. Formát, který mimo jiné umožňuje podepisování a šifrování emailů, se nazývá S/MIME (*Secure/Multipurpose Internet Mail Extensions*). Tato specifikace je rozšířením MIME (*Multipurpose Internet Mail Extensions*) o bezpečnost zpráv.

MIME je standard pro popis různých typů informací. Toto doporučení bylo původně určeno pro převod různě kódovaných informací do textu. Toto umožnilo poslat tyto informace emailem. Standard MIME se také používá v dalších typech komunikace, kde je potřeba specifikovat použitý typ informací.

S/MIME umožňuje konzistentní způsob posílání a přijímání bezpečných MIME dat. Tato specifikace poskytuje následující kryptografické bezpečnostní služby: autentizace, integrita a nepopiratelnost původu zpráv (užitím digitálních podpisů) a důvěrnost a bezpečnost dat (užitím šifrování).

9.1.1. S/MIME formáty pro podepsané zprávy

Specifikace S/MIME definuje dva formáty pro podepsané zprávy: `application/pkcs7-mime` a `multipart/signed`. Pro posílání zpráv je preferován formát `multipart/signed`, protože zprávy v tomto formátu mohou být příjemcem zobrazeny, i pokud nemá zabudován S/MIME software (v tomto případě bude zpráva zobrazena, jako kdyby nebyla podepsána). Aplikace, které zprávy přijímají by měly umět pracovat s oběma formáty.

9.1.2. Podepisování s formátem multipart/signed

Tento MIME typ má dvě části. První obsahuje MIME entitu (například text zprávy), která má být podepsána a druhý obsahuje podpis podle kryptografického standardu PKCS #7. `multipart/signed Content type` má dva vyžadované parametry – `protocol` a `micalg`. Parametr `protocol` musí mít hodnotu `"application/pkcs7-signature"` (nebo rozšířený protokol – `"application/x-pkcs7-signature"`, který právě umožňuje zabudování certifikátu odesilatele a CA do zprávy). Druhý parametr - `micalg` - specifikuje algoritmus výtahu zprávy. Povolené hodnoty jsou `md5` a `sha1`.

9.1.3. Příklad multipart/signed zprávy

```
To: nekdo@nekde.cz
From: steinberger@somewhere.cz
Subject: Podepsana S/MIME zprava
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/x-pkcs7-signature";
      micalg=sha1; boundary="----01DF7FDE8C8EA0EE949EDECB66D5E33E"

------01DF7FDE8C8EA0EE949EDECB66D5E33E
Vzorova mzpráva na podepsani
------01DF7FDE8C8EA0EE949EDECB66D5E33E
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
```

```
MIIFrwYJKoZIhvcNAQcCoIIFoDCCBZwCAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3
. . .
Vlastní data podpisu a certifikátů
. . .
pjPMabmFUT0jjlovxXAPttmjow==

------01DF7FDE8C8EA0EE949EDECB66D5E33E—
```

Podrobnosti o komponování podpisu a certifikátu do zprávy lze nalézt v [RFC-2311] .

9.2. Práce s emaily v jazyce JAVA

Pro práci s emaily slouží v jazyce Java ***JavaMail API***. Toto API podporuje odesílání a příjem MIME zpráv. V době psaní této diplomové práce však zatím neumožňuje práci s formátem S/MIME. Existuje však řada poskytovatelů těchto funkcí. Pro podepisování a verifikaci emailů bylo použito knihovny ***JCSI S/MIME*** od firmy Wedgetail (www.wedgetail.com), která poskytuje volné stažení této knihovny. V budoucnu je plánováno zabudování S/MIME funkcí do JavaMail API.

9.3. Odeslání emailu

9.3.1. Protokol SMTP

SMTP (*Simple Mail Transfer Protocol*) je protokol, který se používá pro výměnu emailů na Internetu. Poštovní klient jej používá při komunikaci se serverem SMTP (někdy zjednodušeně nazývaným poštovní emailový server). Co je to odesílací server SMTP? Termín odesílající znamená, že server akceptuje email pro každého příjemce a projde skrze problémy s doručováním emailů. To je jako když ve skutečném světě vhodíme dopis do schránky (odesílající server SMTP) a pošta jej doručí na adresu, která je uvedena na obálce (příjemce emailu). Poskytovatelé služeb sítě Internetu (ISP, *Internet Service Provider*) často umožňují svým zákazníkům používat jako odesílací jejich servery SMTP. Veřejně dostupné servery, které umožňují jejich použití kýmkoliv dnes stěží existují, protože je někteří lidé používají pro odesílání velkého množství nevyžádaných emailů (spamů). To vedlo k odstoupení od veřejně dostupných serverů SMTP.

9.3.2. Demonstrační program odeslání podepsaného emailu

Pro demonstrování postupu vytvoření, podepsání a odeslání zprávy v Javě byla vytvořena konzolová aplikace `SendSignedEmail.java`.

Na začátku programu se importují balíky tříd nutné pro chod programu. V hlavním programu (funkci `main`) se potom načtou následující parametry:

- `SMTP host` – SMTP server (např.: smtp.volny.cz)
- `SMTP port` – SMTP port (SMTP port by měl být 25)
- `Soubor s certifikátem uživatele` – soubor (cesta + jméno souboru) musí být ve formátu DER
- `Soubor s certifikátem CA` – soubor musí být ve formátu DER
- `Soubor s privátním klíčem odesílatele` – soubor musí být v binárním formátu PKCS#8[39]
- `Od` – emailová adresa odesilatele
- `K` – emailová adresa adresáta
- `Předmět` – název zprávy
- `Text zprávy` – vlastní text zprávy

Tyto parametry se předají funkci `sendMessage`, která zařizuje sestavení, podepsání a odeslání zprávy. Na začátku této funkce je dynamické vložení providera DSTC, se kterým pracuje knihovna JCSI S/MIME:

```
Security.insertProviderAt(new com.dstc.security.provider.DSTC(), 2);
Security.addProvider(
                new com.dstc.security.keymanage.keystore.DSTC());
```

Další část této funkce je nastavení SMTP vlastností pro odeslání zprávy:

```
Properties props = System.getProperties();
props.put("mail.smtp.host", smtpHost);
props.put("mail.smtp.port", smtpPort);
```

[39] Privátní klíč lze převést do tohoto formátu s OpenSSL následujícím příkazem:
`openssl pkcs8 -inform PEM -outform DER -nocrypt -in user.key -out userp8.key -topk8`

Dále se vytvoří `session`[40]:

```
session = Session.getInstance(props, null);
```

Potom se načte certifikát uživatele, certifikát CA a privátní klíč odesilatele:

```
senderCert = getCert(senderCertFile);
caCert = getCert(caCertFile);
senderKey = getPrivKey(privKeyFile);
```

Způsob načtení certifikátu a klíče ze souboru (těla funkcí `getCert` a `getPrivKey`) lze nalézt v příloze D.

Pak se vytvoří objekt zprávy:

```
MimeMessage plainMsg = createMessage(from, to, subject, messageText);
```

Funkce `createMessage` nejprve vytvoří objekt třídy `MimeMessage` a potom nastaví jeho atributy:

```
MimeMessage msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(from));
InternetAddress[] to2 = { new InternetAddress(to)};
msg.setRecipients(Message.RecipientType.TO, to);
msg.setSubject(subject);
msg.setSentDate(new Date());
msg.setContent(messageText, "text/plain");
msg.saveChanges();
```

Vytvořenou zprávu potom podepíšeme. K tomu slouží funkce `sign`:

```
MimeMessage tbSent = sign(plainMsg);
```

[40] Třída `Session` poskytuje přístup k poskytovatelům protokolů, kteří implementují třídu `Transport`, která umožňuje odeslání zprávy.

V této metodě se nejprve vytvoří pole certifikátů, které se mají zakomponovat do zprávy (*Certificate Chain*[41]). První v tomto poli musí být certifikát odesilatele. Potom se vytvoří objekt třídy `SMIMESignature` a zinicializuje se jeho funkcí `initSign`. První parametr této inicializační funkce udává jméno algoritmu výtahu zprávy, druhý objekt privátního klíče pro podepsání, třetí předtím vytvořené pole certifikátů. Čtvrtý parametr je booleovská hodnota *opaque*. Hodnota `TRUE` tohoto parametru znamená opaque signing (matné podepisování), což znamená, že podepisovaný text i data podpisu jsou uloženy ve stejné části MIME zprávy. Výhoda této volby spočívá v jednoznačnosti ověření podpisu v rámci této části MIME zprávy. Nevýhodou je, že zpráva je nečitelná pro poštovní klienty, kteří nepodporují S/MIME. Hodnota `FALSE` znamená clear signing (jasné podepisování). Při tomto nastavení se text zprávy uloží do jedné části MIME zprávy a podpis do druhé. Výhody a nevýhody jsou opačné proti opaque podepisování. Z důvodu možnosti přečtení zprávy ve všech poštovních klientech, kteří podporují MIME se doporučuje zde zadat `FALSE`. Funkce `sign`:

```
private static MimeMessage sign(MimeMessage msg) {
    try {
        X509Certificate[] certs = { senderCert, caCert };

        SMIMESignature smail = new SMIMESignature();

        smail.initSign("SHA-1", senderKey, certs, false);
        smail.setMessage(msg);
        return smail.sign();
    }

    catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Nakonec se vytvoří ***transporter*** (objekt třídy `Transport`) zprávy a ta se odešle:

```
Transport smtp = session.getTransport("smtp");
smtp.send(tbSent);
```

[41] Certificate Chain (řetěz certifikátů) je posloupnost certifikátů od certifikátu uživatele ke kořenovému certifikátu.

9.4. Přijetí emailu

9.4.1. Protokoly pro příjem emailů

Nejrozšířenějšími protokoly pro příjem zpráv jsou IMAP (*Internet Message Access Protocol*) a POP3 (*Post Office Protokol*). Přijímání zpráv bude popisováno na protokolu POP3, avšak aplikace protokolu IMAP by nevyžadovala příliš změn.

9.4.2. Demonstrační program přijetí a verifikace podpisu emailu

Pro demonstrování postupu přijetí a verifikace podpisu zprávy v Javě byla vytvořena konzolová aplikace `ReceiveSignedEmail.java`.

Na začátku programu se importují balíky tříd nutné pro chod programu. V hlavním programu (funkci `main`) se potom načtou následující parametry:

- `POP3 host` – POP3 server (např.: pop3.volny.cz)
- `Uživatelské konto` – jméno uživatelského konta
- `Heslo` – heslo pro předtím zadané uživatelské konto
- `Soubory s důvěryhodnými certifikáty`

Tyto parametry se předají funkci `receiveAndVerifyMessages`, která zařizuje přijetí zpráv z POP3 serveru a verifikuje jejich podpis. Na začátku této funkce je opět dynamické vložení providera DSTC, se kterým pracuje knihovna JCSI S/MIME (viz Odeslání Emailu). Další část této funkce zřizuje spojení s POP3 serverem:

```
System.setProperty("mail.store.protocol", "pop3");
Session session = Session.getDefaultInstance(System.getProperties(), null);
Store store = session.getStore();
store.connect(pop3Host, userAccount, userPasswd);
```

Následuje stažení zpráv z POP3 serveru:

```
Folder inbox = store.getFolder("INBOX");
inbox.open(Folder.READ_WRITE);
int messCount = inbox.getMessageCount();
```

```
msgs = inbox.getMessages();
```

Dále následuje cyklus, který pro každou zprávu ověřuje její podpis. Zpráva se nejprve přetypuje z `Message` na `MimeMessage`. Potom se funkcí `isSigned` ze třídy `SMIMEUtil` zjistí, zda byla tato zpráva podepsána. Pokud ano – ověří se podpis funkcí `verify`, která v případě úspěchu vrací MIME zprávu, jinak `null`. Pro pozdější výpis jsou zprávy ukládány do pole `mimeMessages`.

```
for (int i = 0; i < msgs.length; i++) {
    MimeMessage m = (MimeMessage) msgs[i];
    if (SMIMEUtil.isSigned(m))
        m = verify(m);

    mimeMessages[i] = m;
}
```

Funkce `verify` nejprve vytvoří objekt třídy `SMIMESignature`. Potom zavolá jeho inicializační funkci `initVerify`, jejíž prvním parametrem je vektor důvěryhodných certifikátů CA načtených ve funkci `main`. Do vytvořeného objektu třídy `SMIMESignature` se pak vloží ověřovaná zpráva. Nad tímto zinicializovaným objektem se potom zavolá funkce `verify`, která ověří digitální podpis.

```
    private static MimeMessage verify(MimeMessage msg) {
        try {
            SMIMESignature smail2 = new SMIMESignature();
            smail2.initVerify(trusted, null);
            smail2.setMessage(msg);
            VerificationResult res = smail2.verify();
            return res.getMessage();
        }
        catch (Exception e) {
            return null;
        }
    }
```

Získané zprávy jsou na závěr přehledně vypsány funkcí `printResult`, která je volána z funkce `main`.

10. Závěr

Problém autentizace přístupu do webových aplikací není jednoduchý. Přístup nikdy nebude stoprocentně bezpečný. Digitální podpis však přináší další neopomenutelnou vrstvu zabezpečení. Kombinace použití vrstvy SSL a následné kontroly certifikátu prošlého vrstvou SSL proti certifikátu uloženému v databázi zajišťuje soulad procesu podepisování se Zákonem o elektronickém podpisu. Dalšími výhodami SSL je podpora většinou prohlížečů i serverem Oracle, šifrování komunikace mezi klientem a serverem, vzájemnost autentizace (také server se autentizuje klientovi) a transparentnost (uživatel není ničím zatěžován, pouze musí mít v prohlížeči nainstalován certifikát od certifikační autority, kterou server uznává. Nevýhodou řešení založeném na použití SSL je zpomalení systému, které je způsobené exportováním SSL proměnných.

Tato práce se zaobírala zabezpečením autentizace v PL/SQL aplikacích a v Oracle Portal. Bylo vytvořeno API pro kontrolu certifikátů, které se využívá v obou typech aplikací. Konfigurací vytvořeného API pro kontrolu certifikátů lze volit úroveň bezpečnosti autentizace přístupu. Nejnižší úroveň bezpečnosti porovnává pouze DN vlastníka certifikátu. Při této konfiguraci se může uživatel prezentovat jakýmkoliv certifikátem s DN vlastníka, které má zaregistrováno v databázi. Ovšem zde je podmínka, aby tento certifikát vydala certifikační autorita, které server důvěřuje. Při nastavení střední úrovně bezpečnosti se porovnává DN vlastníka i DN vydavatele certifikátu. Nejvyšší úroveň porovnává i vlastní certifikát. Zde je již téměř jistota jedinečnosti. Výhodou nižších dvou úrovní je rychlost porovnání a také možnost vydání následného certifikátu (po vypršení platnosti) bez zásahu do databáze. Pro ukázání zabezpečeného přihlašování byla vytvořena demonstrační PL/SQL aplikace. Je zde popsán i proces vytvoření nového uživatele a možnosti registrace jeho certifikátu.

V přidání kontroly certifikátu do autentizačního procesu Oracle Portal byl problém, protože těla balíků, které obsahují řídící procedury a funkce tohoto procesu, jsou wrappované (převedené z PL/SQL kódu do objektové formy, která je nezávislá na platformě, ale je nečitelná). Jediná možnost vedla přes externí autentizaci portálu, jejíž autentizační procedura se nachází v newrappovaném balíku. Je zde tedy popsána konfigurace portálu na externí autentizaci přes Oracle Internet Directory a změny v autentizační proceduře. Opět je zde možné nalezt způsob registrace certifikátu uživatele portálu.

Pro podepisování a verifikaci podpisů v emailech bylo využito funkcí z knihovny JCSI SMIME. Implementace vlastních funkcí pro podepsání a verifikaci podpisu emailů přesahuje obsah této diplomové práce. Sama tématika podepisování emailů by mohla tvořit základ další DP.

Přehled použitého značení

`if (a == b)`	úsek programu
`if (`*`podmínka`*`)`	podmínka je příklad obecného syntaktického objektu ve vysvětlení konstrukce příkladu
`soubor.java`	jméno souboru
certifikát	výraz, který byl použit poprvé a bude dále vysvětlen
digital signature	anglický výraz
Možnosti	název položky menu, jméno tlačítka nebo odkazu
Poznámka:	začátek poznámky
Výraz	zdůraznění významu
`package`	listing programu
[4.2.1]	tato problematika je rovněž zmiňována v části 4.2.1
[Pošo2]	tato problematika je rovněž zmiňována v literatuře, zkratku tvoří počáteční písmena jmen autorů a letopočet

Přehled zkratek

DP	Diplomová práce
AS	Application Server
SQL	Structured Query Language
PL/SQL	Programming Language/Structured Query Language
CA	Certifikační autorita
DES	Data Encryption Standard
3DES	Tripple Data Encryption Standard
NIST	National Institute of Standards and Technology
NSA	Národní bezpečnostní agentura
IDEA	International Data Encryption Algorithm
DSA	Digital Signature Algorithm
FIPS	Federální standard pro zpracování informací
SHA	Secure Hash Algorithm
CRC	Cyklický kontrolní redundantní součet
MAC	Message Athentication Code
PKCS	Public Key Cryptography Standard
PFX	Personal Information Exchange
DER	Distinguished Encoding Rules
BER	Basic Encoding Rules
MIME	Multipurpose Internet Mail Extensions
S/MIME	Secure/Multipurpose Internet Mail Extensions
WWW	World Wide Web
SSL	Secure Socket Layer
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
PKI	Public Key Infrastructure
JSP	Java Server Pages
PSP	PL/SQL Server Pages
OC4J	Oracle Container for Java
BC4J	Business Components for Java
JVM	Business Components for Java

JAAS	Java Authentication and Authorization Services
J2EE	Java 2 Enterprise Edition
LDAP	Lightweight Directory Access Protocol
PGP	Pretty Good Privacy
OID	Oracle Internet Directory
SSO	Single Sign-On
PEM	Privacy Enhanced Mail
DN	Distinguish Name

Literatura

[Pošmura, V.: **Apache – příručka správce WWW serveru**

[GS98] Garfinkel,S. – Spafford, G.: **Bezpečnost v unixu a internetu v praxi**

[Her00] Herout, P.: **Učebnice jazyka Java**

Oracle dokumentace:

Oracle9i Application Server Security Guide

Oracle9iAS – Using the PL/SQL Gateway

Oracle9iAS – Mod PL/SQL User's Guide

Oracle9iAS – PL/SQL Web Toolkit Reference

Oracle9iAS Portal – Configuration Guide

Oracle9iAS – Single Sign-On Administrator's Guide

Oracle 8i – Java Stored Procedures Developer's Guide

PL/SQL – User's Guide and Reference

Ostatní zdroje z internetu:

Microsoft Technet – Importing and Exporting Certificates

I. CA – Principy bezpečné komunikace

mod_ssl – User Manual

OpenSSL Command Line Tool

RFC2311 – S/MIME Version 2 Message Specification

VeriSign – Implementing Web Site Client Authentification Using Digital IDs

Příloha A: Parametry systému

Systém, na kterém byly vytvořené programy testovány má následující parametry:

Procesor:	Intel Pentium III – 500 MHz
Operační paměť:	256 MB
Operační systém:	Windows 2000
Server:	Oracle 9iAS Release 1
Databáze:	Oracle 8i Release 3 (8.1.7)
OpenSSL:	OpenSSL 0.9.7a
Java:	Java 2 SDK, Standard Edition 1.4.1
	Java Mail API 1.3
	Javabeans Activation Framework 1.0.2
	JCSI S/MIME Extended 3.0

Příloha B: Listing cert_auth_api

Soubor `cert_auth_api.sql`:

```
-- Package CERT_AUTH_API predstavuje API pro zvyseni bezbecnosti autentizaci certifikaty


create or replace package cert_auth_api as

  -- Typ pro data z certifikatu

  type TCert is record (
    ssl_client_cert varchar2(2000),
    ssl_client_i_dn varchar2(200),
    ssl_client_s_dn varchar2(200)
  );


  -- Funkce, ktera porovna 2 retezce (varchar 2)
  --     vstupni parametry:
  --           par1: prvni vstupni retezec
  --           par2: druhy vstupni retezec
  --
  --     vraci:
  --           0: retezce jsou stejne
  --           1: retezce jsou ruzne

  function compare_varchar2(par1 in varchar2, par2 in varchar2) return integer;


  -- Funkce, ktera overuje data z certifikatu proti databazi
  --     vstupni parametry:
  --           p_app_name: jmeno aplikace
  --
  --           p_user_name: uzivatelske jmeno (pouze pro portal)
  --
  --     vystupni parametry:
  --           errnr: vychozi parametr, ktery udava cislo chyby
  --                0: bez chyby
  --                1: cert_auth nenakonfigurovana
  --                2: certifikat uzivatele nezaregistovan
  --                3: SSL neovereno
  --                4: nesouhlasi ssl_client_s_dn
  --                5: nesouhlasi ssl_client_i_dn
  --                6: nesouhlasi ssl_client_cert
  --
  --     vraci TRUE: overeni v poradku
  --           FALSE: overeni selhalo

  function verify_user_cert(p_app_name in varchar2, p_user_name in varchar2, errno out number)
return boolean;


  -- Konfiguracni funkce pro certifikatovou autentizaci
  --     vstupni parametry:
  --                 compare_type: 'ALL' = porovnat client_dn, issuer_dn i certifikat
  --                               'CLIENT_AND_ISSUER_DN' = porovnat client_dn a issuer_dn
  --                               'CLIENT_DN_ONLY' = porovnat pouze client_dn
  --
  --                 p_app_name: jmeno aplikace
  --
  --     vystupni parametry:
  --           errnr: vychozi parametr, ktery udava cislo chyby
  --                0: bez chyby
  --                1: neplatny typ porovnavani
  --
  --     vraci TRUE: konfigurace v poradku
  --           FALSE: konfigurace selhala

  function configure(p_app_name in varchar2, compare_type in varchar2, errno out number)
return boolean;

END;
/
```

Soubor `cert_auth_api.pkb`**:**

```
-- Package CERT_AUTH_API predstavuje API pro zvyseni bezbecnosti autentizaci certifikaty

create or replace package body cert_auth_api as

  -- Funkce, ktera porovna 2 retezce (varchar 2)
  --     vstupni parametry:
  --           par1: prvni vstupni retezec
  --           par2: druhy vstupni retezec
  --
  --     vraci:
  --           0: retezce jsou stejne
  --           1: retezce jsou ruzne

  function compare_varchar2(par1 in varchar2, par2 in varchar2) return integer as

    raw1 RAW(2000);
    raw2 RAW(2000);

  begin
    raw1 := utl_raw.cast_to_raw(par1);
    raw2 := utl_raw.cast_to_raw(par2);

    if (utl_raw.compare(raw1, raw2) = 0) then
      return (0);
    else
      return (1);
    end if;
  end;


  -- Funkce, ktera overuje data z certifikatu proti databazi
  --     vstupni parametry:
  --           p_app_name: jmeno aplikace
  --
  --           p_user_name: uzivatelske jmeno (pouze pro portal)
  --
  --     vystupni parametry:
  --           errnr: vychozi parametr, ktery udava cislo chyby
  --                0: bez chyby
  --                1: cert_auth nenakonfigurovana
  --                2: certifikat uzivatele nezaregistovan
  --                3: SSL neovereno
  --                4: nesouhlasi ssl_client_s_dn
  --                5: nesouhlasi ssl_client_i_dn
  --                6: nesouhlasi ssl_client_cert
  --
  --     vraci TRUE: overeni v poradku
  --           FALSE: overeni selhalo

  function verify_user_cert(p_app_name in varchar2, p_user_name in varchar2, errno out number)
return boolean as

    ssl_cert_data TCert;             -- data ze SSL promennych
    db_cert_data TCert;              -- data z databaze
    user varchar2(30);                      -- uzivatelske jmeno prihlaseneho uzivatele
    ssl_verify varchar(20);          -- hodnota promenne SSL_VERIFY

    type t_crsr is ref cursor;              -- typ kurzoru
    plist t_crsr;                    -- pomocny kurzor

    config cert_auth_config_t%ROWTYPE;  -- konfiguracni data

  begin

    -- nacteni obsahu promennych serveru

    ssl_cert_data.ssl_client_cert := owa_util.get_cgi_env('SSL_CLIENT_CERT');
    ssl_cert_data.ssl_client_i_dn := owa_util.get_cgi_env('SSL_CLIENT_I_DN');
    ssl_cert_data.ssl_client_s_dn := owa_util.get_cgi_env('SSL_CLIENT_S_DN');
    ssl_verify  := owa_util.get_cgi_env('SSL_CLIENT_VERIFY');


    -- nacteni uzivatelskeho jmena podle typu aplikace
```

```
    if (compare_varchar2(p_app_name,'PORTAL') = 0) then
      user := p_user_name;
    else
      user := owa_util.get_cgi_env('REMOTE_USER');
    end if;


    -- nacteni konfigurace cert_auth

    open plist for select * from cert_auth_config_t where (p_app_name = app_name);
    fetch plist into config;
    if (plist%notfound) then
      errno := 1;
      return (false);
    end if;


    -- ziskani dat certifikatu z databaze

    if (compare_varchar2(p_app_name,'PORTAL') = 0) then
      open plist for select ssl_client_cert, ssl_client_i_dn, ssl_client_s_dn from
portal30_sso.wwsso_usercert_t where user_name = user;
      fetch plist into db_cert_data;
      if (plist%notfound) then
        errno := 2;
        return (false);
      end if;
    else
      open plist for select ssl_client_cert, ssl_client_i_dn, ssl_client_s_dn from
cert_auth_user_t where (user_name = user) and (p_app_name = app_name);
      fetch plist into db_cert_data;
      if (plist%notfound) then
        errno := 2;
        return (false);
      end if;
    end if;

    -- kontrola ssl_verify

    if (compare_varchar2(ssl_verify, 'SUCCESS') = 1) then
      errno := 3;
      return (false);
    end if;

    -- kontrola ssl_client_S_dn

    if (compare_varchar2(db_cert_data.ssl_client_s_dn, ssl_cert_data.ssl_client_s_dn) = 1)
then
      errno := 4;
      return (false);
    end if;


    if ((compare_varchar2(config.compare_type, 'ALL') = 0) or
(compare_varchar2(config.compare_type, 'CLIENT_AND_ISSUER_DN') = 0)) then

      -- kontrola ssl_client_i_dn

      if (compare_varchar2(db_cert_data.ssl_client_i_dn, ssl_cert_data.ssl_client_i_dn) = 1)
then
        errno := 5;
        return (false);
      end if;
    end if;


    if (compare_varchar2(config.compare_type, 'ALL') = 0) then

      -- kontrola ssl_client_cert

      if (compare_varchar2(db_cert_data.ssl_client_cert, ssl_cert_data.ssl_client_cert) = 1)
then
        errno := 6;
        return (false);
      end if;
    end if;
```

```
    -- certifikat souhlasi

    errno := 0;
    return (true);

  end;


  -- Konfiguracni funkce pro certifikatovou autentizaci
  --     vstupni parametry:
  --                 compare_type: 'ALL' = porovnat client_dn, issuer_dn i certifikat
  --                               'CLIENT_AND_ISSUER_DN' = porovnat client_dn a issuer_dn
  --                               'CLIENT_DN_ONLY' = porovnat pouze client_dn
  --
  --                 p_app_name: jmeno aplikace
  --
  --     vystupni parametry:
  --           errnr: vychozi parametr, ktery udava cislo chyby
  --                0: bez chyby
  --                1: neplatny typ porovnavani
  --
  --     vraci TRUE: konfigurace v poradku
  --           FALSE: konfigurace selhala

  function configure(p_app_name in varchar2, compare_type in varchar2, errno out number)
return boolean as

  begin

    -- vymazani stare konfigurace

    delete from cert_auth_config_t where (p_app_name = app_name);


    -- kontrola compare_type

    if ((compare_varchar2(compare_type, 'ALL') = 1) and (compare_varchar2(compare_type,
'CLIENT_AND_ISSUER_DN') = 1) and
        (compare_varchar2(compare_type, 'CLIENT_DN_ONLY') = 1)) then
        errno := 1;
        return (false);
    end if;


    -- ulozeni konfigurace

    insert into cert_auth_config_t values (p_app_name, compare_type);


    -- konfigurace v poradku

    errno := 0;
    return (true);
  end;


end;
/

show errors;
```

Příloha C: Listing cert_auth_example

Soubor `cert_auth_example.sql`**:**

```
-- Package CERT_AUTH_EXAMPLE tvori aplikaci demonstrujici zabezpeceni autentizace certifikaty


create or replace package cert_auth_example as

  -- Hlavni stranka soukrome sekce aplikace

  procedure app_page;


  -- Hlavicka HTML stranky
  --     Parametry:
  --                title: nazev stranky

  procedure html_header(title in varchar2);


  -- Paticka HTML stranky
  --     Parametry:
  --                url: url odkazu na nadrazenou stranku
  --                text: text odkazu na nadrazenou stranku

  procedure html_footer(url in varchar2, text in varchar2);


  -- HTML odkaz
  --     Parametry:
  --                url: url odkazu
  --                text: text odkazu

  procedure html_ref(url in varchar2, text in varchar2);


  -- Stranka informaci o certifikatu klienta

  procedure cli_cert_info;


  -- Stranka informaci o certifikatu serveru

  procedure ser_cert_info;


  -- Stranka informaci o ssl

  procedure ssl_info;


  -- Tisk chyby
  --     Parametry:
  --                errno: cislo chyby

  procedure print_error(errno in number);


  -- Stranka uzivatelske registrace

  procedure register_cert;


  -- Konfiguracni funkce pro aplikaci demonstrujici certifikatovou autentizaci
  --     vstupni parametry:
  --                 reg_type: 'USER_REG' = uzivatelska rgistrace
  --                           'ADMIN_REG' = administratorska registrace
  --                 pls_url_path: url cesta k package cert_auth_example
  --                 ps_url_path: url cesta k verejne sekci
  --
  --     vystupni parametry:
  --           errnr: vychozi parametr, ktery udava cislo chyby
  --                0: bez chyby
```

```
  --                1: neplatny typ registrace
  --
  --     vraci TRUE: konfigurace v poradku
  --           FALSE: konfigurace selhala

  function configure(reg_type in varchar2, pls_url_path in varchar2, ps_url_path in varchar2,
errno out number) return boolean;


  -- vraci typ registrace

  function get_reg_type return varchar2;


  -- vraci pls_url_path

  function get_pls_url_path return varchar2;


  -- vraci ps_url

  function get_ps_url_path return varchar2;

END;
/
```

Soubor `cert_auth_example.pkb`:

```
-- Package CERT_AUTH_EXAMPLE tvori aplikaci demonstrujici zabezpeceni autentizace certifikaty

create or replace package body cert_auth_example as

  -- Tisk chyby
  --     Parametry:
  --                errno: cislo chyby

  procedure print_error(errno in number) as

  begin

    if (errno = 1) then
      htp.print('<h1 class="error">Chyba ovìøení 2:</h1>');
      htp.print('<h2>CERT_AUTH nenakonfikurována</h2>');

    elsif (errno = 2) then
      htp.print('<h1 class="error">Chyba ovìøení 3:</h1>');
      htp.print('<h2>Váš certifikát není zaregistrován</h2>');
      if (cert_auth_api.compare_varchar2(get_reg_type(), 'USER_REG') = 0) then
        htp.print('<p>');
        html_ref(get_pls_url_path() ||'/cert_auth_example.register_cert','Zaregistrovat
certifikat');
        htp.print('</p>');
      end if;

    elsif (errno = 3) then
      htp.print('<h1 class="error">Chyba ovìøení 4:</h1>');
      htp.print('<h2>Cerifikát nebyl ovìøen vrstvou SSL</h2>');

    elsif (errno = 4) then
      htp.print('<h1 class="error">Chyba ovìøení 5:</h1>');
      htp.print('<h2>Nesprávný certifikát - nesouhlasí ssl_client_s_dn</h2>');

    elsif (errno = 5) then
      htp.print('<h1 class="error">Chyba ovìøení 6:</h1>');
      htp.print('<h2>Nesprávný certifikát - nesouhlasí ssl_client_i_dn</h2>');

    elsif (errno = 6) then
      htp.print('<h1 class="error">Chyba ovìøení 7:</h1>');
      htp.print('<h2>Nesprávný certifikát - nesouhlasí ssl_client_cert</h2>');
    end if;
  end;
```

```
  -- Hlavni stranka soukrome sekce aplikace

  procedure app_page as

    remote_user varchar2(30);
    errno number(1);

  begin

    html_header('Demonstraèní aplikace certifikátové autentizace - soukromá sekce');

    htp.print('<center>');

    remote_user := owa_util.get_cgi_env('REMOTE_USER');

    if (cert_auth_api.verify_user_cert('EXAMPLE_APP', NULL, errno)) then

      htp.print('<h1>Soukromá sekce</h1>');
      htp.print('<h2>Vítáme vás v sekci pro registrované uživatele</h2>');
      htp.print('Uživatelské jméno: <b> '|| remote_user ||'</b><br><br>');
      htp.print('<p>');
      admin.html_ref(get_pls_url_path() || '/cert_auth_example.cli_cert_info','Certifikát
klienta');
      htp.print('</p>');
      htp.print('<p>');
      admin.html_ref(get_pls_url_path() || '/cert_auth_example.ser_cert_info','Certifikát
serveru');
      htp.print('</p>');
      htp.print('<p>');
      admin.html_ref(get_pls_url_path() || '/cert_auth_example.ssl_info','SSL parametry');
      htp.print('</p>');

    else
      print_error(errno);

    end if;

    htp.print('</center>');

    html_footer(get_ps_url_path() || '/demo_app_index.html','Veøejná sekce');

  end;


  -- Hlavicka HTML stranky
  --     Parametry:
  --                title: nazev stranky

  procedure html_header(title in varchar2) as

  begin
    htp.print('<html>');
    htp.print('<head>');
    htp.print('  <title>'|| title ||'</title>');
    htp.print('  <link rel="stylesheet" type="text/css" href="' || get_ps_url_path() ||
'/demo_app.css">');
    htp.print('</head>');
    htp.print('<body>');
    htp.print(' <div id=page-footer align=right><span class=navig><small>');
    htp.print(' <a href="' || get_ps_url_path() || '/demo_app_index.html"><b>| index
|</b></a>');
    htp.print(' </small></span></div>');
    htp.print(' <hr noshade>');
  end;


  -- Paticka HTML stranky
  --     Parametry:
  --                url: url odkazu na nadrazenou stranku
  --                text: text odkazu na nadrazenou stranku

  procedure html_footer(url in varchar2, text in varchar2) as

  begin
    htp.print('  <div id=page-footer align=right><span class=navig><small>');
    htp.print('<a href="'|| url ||'"><b>| '|| text ||' |</b></a>');
    htp.print('  </small></span></div>');
```

```
    htp.print('  <hr noshade>');
    htp.print('  <div id=page-footer align=right><span class=navig><small>');
    htp.print('  Autor: <a href="mailto:steinberger@volny.cz"><b>Josef Steinberger</b></a>');
    htp.print('  </small></span></div>');
    htp.print('</body>');
    htp.print('</html>');
  end;


  -- HTML odkaz
  --     Parametry:
  --                url: url odkazu
  --                text: text odkazu

  procedure html_ref(url in varchar2, text in varchar2) as

  begin
    htp.print('  <div id=page-footer><span class=navig>');
    htp.print('<a href="'|| url ||'"><b>'|| text ||'</b></a>');
    htp.print('  </span></div>');
  end;


  -- Stranka informaci o certifikatu klienta

  procedure cli_cert_info as

    remote_user varchar2(30);
    errno number(1);

  begin
    html_header('Demonstraèní aplikace certifikátové autentizace - certifikát uživatele');
    htp.print('<center>');

    remote_user := owa_util.get_cgi_env('REMOTE_USER');
    if (cert_auth_api.verify_user_cert('EXAMPLE_APP', NULL, errno)) then
      htp.print('<h1>Certifikát klienta s uživatelským jménem <b>'|| remote_user ||
'</b></h1>');

      htp.print('<p><h2>Držitel certifikátu: </h2>');
      htp.print('<b>Jméno: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_S_DN_CN') || '<br>');
      htp.print('<b>Stát (zkratka): </b>'|| owa_util.get_cgi_env('SSL_CLIENT_S_DN_C') ||
'<br>');
      htp.print('<b>Stát (plný název): </b>'|| owa_util.get_cgi_env('SSL_CLIENT_S_DN_ST') ||
'<br>');
      htp.print('<b>Organizace: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_O') || '</p>');

      htp.print('<p><h2>Vydavatel certifikátu: </h2>');
      htp.print('<b>Jméno: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_CN') || '<br>');
      htp.print('<b>Stát (zkratka): </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_C') ||
'<br>');
      htp.print('<b>Stát (plný název): </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_ST') ||
'<br>');
      htp.print('<b>Lokalita: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_L') || '<br>');
      htp.print('<b>Organizace: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_O') || '<br>');
      htp.print('<b>E-mail: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_I_DN_EMAIL') || '</p>');

      htp.print('<p><h2>Algoritmy: </h2>');
      htp.print('<b>Algoritmus podpisu: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_A_SIG') ||
'<br>');
      htp.print('<b>Algorimus veøejného klíèe: </b>'||
owa_util.get_cgi_env('SSL_CLIENT_A_KEY') || '</p>');

      htp.print('<p><h2>Platnost: </h2>');
      htp.print('<b>Zaèátek: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_V_START') || '<br>');
      htp.print('<b>Konec: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_V_END') || '</p>');

      htp.print('<p><h2>Ostatní parametry: </h2>');
      htp.print('<b>Verze certifikátu: </b>'|| owa_util.get_cgi_env('SSL_CLIENT_M_VERSION') ||
'<br>');
      htp.print('<b>Sériové èíslo certifikátu: </b>'||
owa_util.get_cgi_env('SSL_CLIENT_M_SERIAL') || '</p>');

    else
      print_error(errno);

    end if;
```

```
    htp.print('</center>');
    html_footer(get_pls_url_path() ||'/cert_auth_example.app_page','Soukromá sekce');
  end;


  -- Stranka informaci o certifikatu serveru

  procedure ser_cert_info as

  remote_user varchar2(30);
  errno number(1);

  begin
    html_header('Demonstraèní aplikace certifikátové autentizace - certifikát serveru');
    htp.print('<center>');

    remote_user := owa_util.get_cgi_env('REMOTE_USER');
    if (cert_auth_api.verify_user_cert('EXAMPLE_APP', NULL, errno)) then
      htp.print('<h1>Certifikát serveru</h1>');

      htp.print('<p><h2>Držitel certifikátu: </h2>');
      htp.print('<b>Jméno: </b>'|| owa_util.get_cgi_env('SSL_SERVER_S_DN_CN') || '<br>');
      htp.print('<b>Stát (zkratka): </b>'|| owa_util.get_cgi_env('SSL_SERVER_S_DN_C') ||
'<br>');
      htp.print('<b>Stát (plný název): </b>'|| owa_util.get_cgi_env('SSL_SERVER_S_DN_ST') ||
'<br>');
      htp.print('<b>Organizace: </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_O') || '</p>');

      htp.print('<p><h2>Vydavatel certifikátu: </h2>');
      htp.print('<b>Jméno: </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_CN') || '<br>');
      htp.print('<b>Stát (zkratka): </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_C') ||
'<br>');
      htp.print('<b>Stát (plný název): </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_ST') ||
'<br>');
      htp.print('<b>Lokalita: </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_L') || '<br>');
      htp.print('<b>Organizace: </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_O') || '<br>');
      htp.print('<b>E-mail: </b>'|| owa_util.get_cgi_env('SSL_SERVER_I_DN_EMAIL') || '</p>');

      htp.print('<p><h2>Algoritmy: </h2>');
      htp.print('<b>Algoritmus podpisu: </b>'|| owa_util.get_cgi_env('SSL_SERVER_A_SIG') ||
'<br>');
      htp.print('<b>Algorimus veøejného klíèe: </b>'||
owa_util.get_cgi_env('SSL_SERVER_A_KEY') || '</p>');

      htp.print('<p><h2>Platnost: </h2>');
      htp.print('<b>Zaèátek: </b>'|| owa_util.get_cgi_env('SSL_SERVER_V_START') || '<br>');
      htp.print('<b>Konec: </b>'|| owa_util.get_cgi_env('SSL_SERVER_V_END') || '</p>');

      htp.print('<p><h2>Ostatní parametry: </h2>');
      htp.print('<b>Verze certifikátu: </b>'|| owa_util.get_cgi_env('SSL_SERVER_M_VERSION') ||
'<br>');
      htp.print('<b>Sériové èíslo certifikátu: </b>'||
owa_util.get_cgi_env('SSL_SERVER_M_SERIAL') || '</p>');


    else
      print_error(errno);

    end if;

    htp.print('</center>');
    html_footer(get_pls_url_path() ||'/cert_auth_example.app_page','Soukromá sekce');
  end;


  -- Stranka informaci o ssl

  procedure ssl_info as

    remote_user varchar2(30);
    errno number(1);

  begin
    html_header('Demonstraèní aplikace certifikátové autentizace - SSL info');
    htp.print('<center>');
```

```
    remote_user := owa_util.get_cgi_env('REMOTE_USER');
    if (cert_auth_api.verify_user_cert('EXAMPLE_APP', NULL, errno)) then
      htp.print('<h1>SSL informace</h1>');

      htp.print('<p><h2>Šifrování: </h2>');
      htp.print('<b>Jméno šifry: </b>'|| owa_util.get_cgi_env('SSL_CIPHER') || '<br>');
      htp.print('<b>Export šifry: </b>'|| owa_util.get_cgi_env('SSL_CIPHER_EXPORT') ||
'<br>');
      htp.print('<b>Poèet bitù šifrovacího klíèe (použitého): </b>'||
owa_util.get_cgi_env('SSL_CIPHER_USEKEYSIZE') || '<br>');
      htp.print('<b>Poèet bitù šifrovacího klíèe (možného): </b>'||
owa_util.get_cgi_env('SSL_CIPHER_ALGKEYSIZE') || '</p>');

      htp.print('<p><h2>Ostatní parametry SSL: </h2>');
      htp.print('<b>Verze SSL: </b>'|| owa_util.get_cgi_env('SSL_PROTOCOL') || '<br>');
      htp.print('<b>Verze mod_ssl: </b>'|| owa_util.get_cgi_env('SSL_VERSION_INTERFACE') ||
'<br>');
      htp.print('<b>Verze OpenSSL: </b>'|| owa_util.get_cgi_env('SSL_VERSION_LIBRARY') ||
'<br>');
      htp.print('<b>SSL session id: </b>'|| owa_util.get_cgi_env('SSL_SESSION_ID') || '</p>');

    else
      print_error(errno);

    end if;

    htp.print('</center>');
    html_footer(get_pls_url_path() ||'/cert_auth_example.app_page','Soukromá sekce');
  end;


  -- Stranka uzivatelske registrace

  procedure register_cert as

    ssl_cert_data cert_auth_api.TCert;              -- data ze SSL promennych
    db_cert_data cert_auth_api.TCert;               -- data z databaze

    ssl_verify varchar(20);          -- promenna serveru SSL_VERIFY

    type t_crsr is ref cursor;              -- typ kurzoru
    plist t_crsr;                    -- pomocny kurzor

    user_name varchar2(30) default '-1';-- jmeno uzivatele

    remote_user varchar2(30);  -- prihlaseny uzivatel

  begin
    html_header('Demonstraèní aplikace certifikátové autentizace - registrace certifikátu');
    htp.print('<center>');

    ssl_cert_data.ssl_client_cert := owa_util.get_cgi_env('SSL_CLIENT_CERT');
    ssl_cert_data.ssl_client_i_dn := owa_util.get_cgi_env('SSL_CLIENT_I_DN');
    ssl_cert_data.ssl_client_s_dn := owa_util.get_cgi_env('SSL_CLIENT_S_DN');

    remote_user := owa_util.get_cgi_env('REMOTE_USER');
    ssl_verify  := owa_util.get_cgi_env('SSL_CLIENT_VERIFY');

    if (cert_auth_api.compare_varchar2(get_reg_type(), 'USER_REG') = 0) then

      if (cert_auth_api.compare_varchar2(ssl_verify, 'SUCCESS') <> 0) then
        htp.print('<h1 class="error">Chyba pøi registraci 1: </h1>');
        htp.print('<h2>Cerifikát nebyl ovìøen vrstvou SSL</h2>');

      else

        open plist for select user_name from cert_auth_user_t where user_name = remote_user;
        fetch plist into user_name;

        if (plist%notfound) then
          open plist for select user_name from cert_auth_user_t where ssl_client_cert =
ssl_cert_data.ssl_client_cert;
          fetch plist into user_name;

          if (plist%notfound) then user_name := null;
          end if;
```

```
          if  (user_name is not null) then
            htp.print('<h1 class="error">Chyba pøi registraci 2: </h1>');
            htp.print('<h2>Tento certifikát je již zaregistrován s jiným uživatelským
jménem</h2>');
          else

            insert into cert_auth_user_t values(remote_user, 'EXAMPLE_APP',
ssl_cert_data.ssl_client_cert, ssl_cert_data.ssl_client_i_dn,
                                    ssl_cert_data.ssl_client_s_dn);

            htp.print('<h1>Registrace dokonèena</h1>');
            htp.print('<h2>Certifikát zaregistrován s uživatelským jménem ' || remote_user
||'</h2>');
          end if;
        else
          htp.print('<h1 class="error">Chyba pøi registraci 3: </h1>');
          htp.print('<h2>Uživatel '|| remote_user ||' má již zaregistrovaný certifikát</h2>');
        end if;
      end if;
    else
        htp.print('<h1 class="error">Chyba pøi registraci 4: </h1>');
        htp.print('<h2>Uživatelská registrace certifikátù není povolena</h2>');
    end if;

    htp.print('</center>');
    html_footer(get_ps_url_path() || '/demo_app_index.html','Veøejná sekce');
  end;


  -- Konfiguracni funkce pro aplikaci demonstrujici certifikatovou autentizaci
  --     vstupni parametry:
  --                 reg_type: 'USER_REG' = uzivatelska rgistrace
  --                           'ADMIN_REG' = administratorska registrace
  --                 pls_url_path: url cesta k package cert_auth_example
  --                 ps_url_path: url cesta k verejne sekci
  --
  --     vystupni parametry:
  --           errnr: vychozi parametr, ktery udava cislo chyby
  --                0: bez chyby
  --                1: neplatny typ registrace
  --
  --     vraci TRUE: konfigurace v poradku
  --           FALSE: konfigurace selhala

  function configure(reg_type in varchar2, pls_url_path in varchar2, ps_url_path in varchar2,
errno out number) return boolean as
  begin

    -- vymazani stare konfigurace

    delete from cert_auth_ex_config_t;


    -- kontrola reg_type

    if ((cert_auth_api.compare_varchar2(reg_type, 'USER_REG') = 1) and
(cert_auth_api.compare_varchar2(reg_type, 'ADMIN_REG') = 1)) then
        errno := 1;
        return (false);
    end if;


    -- ulozeni konfigurace

    insert into cert_auth_ex_config_t values (reg_type, pls_url_path, ps_url_path);

    -- konfigurace v poradku

    errno := 0;
    return (true);

  end;


  -- vraci typ registrace
```

```
  function get_reg_type return varchar2 as

    ret_val varchar2(30);

  begin
    select reg_type into ret_val from cert_auth_ex_config_t;
    return (ret_val);
  end;


  -- vraci pls_url_path

  function get_pls_url_path return varchar2 as

    ret_val varchar2(200);

  begin
    select pls_url_path into ret_val from cert_auth_ex_config_t;
    return (ret_val);
  end;


  -- vraci ps_url_path

  function get_ps_url_path return varchar2 as

    ret_val varchar2(200);

  begin
    select ps_url_path into ret_val from cert_auth_ex_config_t;
    return (ret_val);
  end;

end;
/

show errors;
```

Příloha D: Listing SendSignedEmail.java

```
/*****************************************************************************
 * Program SendSignedEmail - demonstrace podepisovani emailu
 *
 * Autor: Josef Steinberger
 *
 * Duben 2003
 *
 * Program byl vytvoren v ramci diplomove prace
 *****************************************************************************/

import java.security.*;
import java.security.spec.*;
import java.security.cert.*;

import javax.activation.*;

import com.dstc.security.smime.*;

import java.io.*;
import java.util.*;
import java.text.*;
import javax.mail.*;
import javax.mail.internet.*;


public class SendSignedEmail {

    private static PrivateKey senderKey = null; // privatni klic odesilatele
    private static X509Certificate caCert = null; // certifikat CA
    private static X509Certificate senderCert = null; // certifikat odesilatele

    private static Session session = null;

    /******************
     * Hlavni program *
     ******************/
    public static void main(String[] args) {

        String smtpHost;
        String smtpPort;
        String senderCertFile;
        String caCertFile;
        String privKeyFile;
        String from;
        String to;
        String subject;
        String messageText;

        System.out.println("*************");
        System.out.println("* SMTP data *");
        System.out.println("*************");
        smtpHost = readParam("SMTP host");
        smtpPort = readParam("SMTP port");

        System.out.println("\n**********************************");
        System.out.println("* Soubory s certifikaty a klicem *");
        System.out.println("**********************************");
        senderCertFile = readParam("Soubor s certifikatem odesilatele");
        caCertFile = readParam("Soubor s certifikatem CA");
        privKeyFile = readParam("Soubor s privatnim klicem odesilatele");

        System.out.println("\n***********");
        System.out.println("* Zprava *");
        System.out.println("***********");
        from = readParam("Od");
        to = readParam("K");
        subject = readParam("Predmet");
        System.out.println("");
        messageText = readParam("Text zpravy");

        if (sendMessage(smtpHost, smtpPort, senderCertFile, caCertFile, privKeyFile,
                    from, to, subject, messageText))
            System.out.println("Zprava byla odeslana");
```

```
        else
            System.out.println("Chyba pri odesilani");
;
    }


    /*****************************
     * Nacteni parametru programu *
     *****************************/
    private static String readParam(String paramName) {

        String sdata;
        byte[] data = new byte[80];

        try {
            System.out.print(paramName + ": ");
            System.in.read(data);
            sdata = new String(data).trim();

            return sdata;
        }

        catch (IOException e) {
            System.out.println("Chyba pri cteni parametru " + paramName);
            return null;
        }
    }


    /******************
     * Poslani zpravy *
     ******************/
    public static boolean sendMessage(String smtpHost, String smtpPort, String senderCertFile,
                                      String caCertFile, String privKeyFile, String from, String
to,
                                      String subject, String messageText) {

        try {
            // Dynamicke pridani providera
            Security.insertProviderAt(new com.dstc.security.provider.DSTC(), 2);
            Security.addProvider(
                new com.dstc.security.keymanage.keystore.DSTC());

            // nastaveni smtp vlastnosti
            Properties props = System.getProperties();
            props.put("mail.smtp.host", smtpHost);
            props.put("mail.smtp.port", smtpPort);

            // vytvoreni session
            session = Session.getInstance(props, null);

            // nacteni certifikatu
            senderCert = getCert(senderCertFile);
            caCert = getCert(caCertFile);

            // nacteni privatniho klice
            senderKey = getPrivKey(privKeyFile);

            // vytvoreni zpravy
            MimeMessage plainMsg = createMessage(from, to, subject, messageText);

            MimeMessage tbSent = sign(plainMsg);

            // vytvoreni transporteru zpravy a odeslani
            Transport smtp = session.getTransport("smtp");
            smtp.send(tbSent);

            return true;
        }

        catch (MessagingException mex) {
           return false;
        }
    }


    /********************
```

```
     * Vytvoreni zpravy *
     ********************/
    private static MimeMessage createMessage(String from, String to, String subject, String
messageText) {

        try {
            MimeMessage msg = new MimeMessage(session);

            msg.setFrom(new InternetAddress(from));
            InternetAddress[] to2 = { new InternetAddress(to)};

            msg.setRecipients(Message.RecipientType.TO, to);
            msg.setSubject(subject);
            msg.setSentDate(new Date());

            msg.setContent(messageText, "text/plain");
            msg.saveChanges();

            return msg;
        }

        catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException("Neocekavana chyba: " + e.toString());
        }
    }


    /********************
     * Podepsani zpravy *
     ********************/
    private static MimeMessage sign(MimeMessage msg) {
        try {
            X509Certificate[] certs = { senderCert, caCert };

            SMIMESignature smail = new SMIMESignature();

            smail.initSign("SHA-1", senderKey, certs, false);
            smail.setMessage(msg);
            return smail.sign();
        }

        catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }


    /*************************************
     * Nacteni privatniho klice ze souboru *
     *************************************/
    private static PrivateKey getPrivKey(String filename) {

        PrivateKey privateKey = null;

        try {
            DataInputStream in = new DataInputStream(
            new FileInputStream(filename));
            byte[] encodedPrivateKey = new byte[1024];
            int noOfBytes = in.read(encodedPrivateKey);
            in.close();

            PKCS8EncodedKeySpec encodedPrivateKeySpec = new
PKCS8EncodedKeySpec(encodedPrivateKey);
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            privateKey = keyFactory.generatePrivate(encodedPrivateKeySpec);

            return privateKey;
        }

        catch (Exception e) {
            System.out.println(e.toString());
            e.printStackTrace();
            return null;
        }
    }
```

```
    /*********************************
     * Nacteni certifikatu ze souboru *
     *********************************/
    private static X509Certificate getCert(String certFile) {

        try {
            FileInputStream fis = new FileInputStream(certFile);
            DataInputStream dis = new DataInputStream(fis);
            byte[] data = new byte[dis.available()];
            dis.readFully(data);
            ByteArrayInputStream bais = new ByteArrayInputStream(data);
            CertificateFactory fact = CertificateFactory.getInstance("X509");
            X509Certificate cert = (X509Certificate) fact.generateCertificate(bais);

            return cert;
        }

        catch (Exception e) {
            System.out.println(e.toString());
            e.printStackTrace();
            return null;
        }
    }
}
```

Příloha E: Listing ReceiveSignedEmail.java

```
/*****************************************************************************
 * Program ReceiveSignedEmail - demonstrace prijimani a verifikace podpisu v emailu
 *
 * Autor: Josef Steinberger
 *
 * Duben 2003
 *
 * Program byl vytvoren v ramci diplomove prace
 *****************************************************************************/

import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import com.dstc.security.smime.*;


/***********************************
 * Pomocna trida pro nacitani hesla *
 ***********************************/

class MaskingThread extends Thread {
   private boolean stop = false; // priznak ukonceni maskovani
   private String prompt; // @prompt se ukazuje misto hesla na obrazovce


   public MaskingThread(String prompt) {
      this.prompt = prompt;
   }

   // zacatek maskovani
   public void run() {
      while(!stop) {
         try {
            // maskovaci frekvence
            this.sleep(1);
         }catch (InterruptedException iex) {
            iex.printStackTrace();
         }
         if (!stop) {
            System.out.print("\r" + prompt + " \r" + prompt);
         }
         System.out.flush();
      }
   }

   // prikaz k ukonceni maskovani
   public void stopMasking() {
      this.stop = true;
   }
}


/*****************************************************
 * Hlavni trida prijimaci a verifikace podpisu v emailu *
 *****************************************************/
public class ReceiveSignedEmail {

    private static int messageCount;  // pocet zprav
    private static MimeMessage[] mimeMessages = new MimeMessage[100]; // buffer MIME zprav
    private static Message[] msgs; // buffer zprav typu Message
    private static Vector trusted = new Vector(); // vektor certifikatu duveryhodnych CA

    /******************
     * Hlavni program *
     ******************/
    public static void main(String[] args) {

        String host;
        String userAccount;
        String userPasswd = "";
```

```
    String trustedPath;
    String addNextTrusted;

    System.out.println("*************");
    System.out.println("* POP3 data *");
    System.out.println("*************");
    host = readParam("POP3 host");
    userAccount = readParam("Uzivatelske konto");
    try {
        userPasswd = getPassword("Heslo: ");
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("\n**************************");
    System.out.println("* Duveryhodne certifikaty *");
    System.out.println("**************************");
    addNextTrusted = "A";
    while (addNextTrusted.compareToIgnoreCase("A") == 0) {
        trustedPath = readParam("Soubor s certifikate duveryhodne CA");
        addTrustedCert(trustedPath);

        System.out.print("Pridat dalsi duveryhodny certifikat [A/N]? ");
        try {
            byte[] data = new byte[200];
            System.in.read(data);
            addNextTrusted = new String(data).trim();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
    receiveAndVerifyMessages(host, userAccount, userPasswd);
    printResult();
}


/*******************************************
 * Funkce pro ziskani (nezobrazovaneho) hesla *
 *******************************************/
private static String getPassword(String prompt) throws IOException {

    String password = ""; // heslo
    MaskingThread maskingthread = new MaskingThread(prompt);
    Thread thread = new Thread(maskingthread);
    thread.start();

    // blokovani dokud se nezmackne enter
    while (true) {
        char c = (char)System.in.read();

        maskingthread.stopMasking();

        if (c == '\r') {
            c = (char)System.in.read();
            if (c == '\n') {
                break;
            }
            else {
                continue;
            }
        }
        else if (c == '\n') {
            break;
        }
        else {
            // ulozeni hesla
            password += c;
        }
    }
    return password;
}


/*************************
 * Vypsani prijatych zprav *
 *************************/
```

```
private static void printResult() {
    String s;

    try {
        System.out.println("\nPocet zprav: " + messageCount);
        for (int i = 0; i < messageCount; i++) {
            MimeMessage m = (MimeMessage) msgs[i];
            System.out.println("\n******************");
            System.out.println("* Zprava cislo " + (i+1) + " *");
            System.out.println("******************");
            if (mimeMessages[i] != null) {
                if (SMIMEUtil.isSigned(m)) {
                    System.out.println("Digitalni podpis v poradku\n");
                }
                else {
                    System.out.println("Zprava nebyla digitalne podepsana\n");
                }
                s = (String) mimeMessages[i].getContent();
            }
            else {
                System.out.println("Digitalni podpis neplatny\n");

            }
            InternetAddress[] ia = (InternetAddress[]) m.getFrom();
            System.out.println("Odesilatel: " + ia[0].getAddress());
            System.out.println("Datum odeslani: " + m.getSentDate().toString());
            System.out.println("Predmet: " + m.getSubject());
            System.out.println("Text Zpravy: \n");
            printMessageText(msgs[i]);
            System.out.println("\n");
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}


/*************************
 * Vypsani obsahu zpravy *
 *************************/
private static void printMessageText(Message m) {

    try {
        // ziskani casti zpravy (nebo zpravy cele)
        Part messagePart=m;
        Object content=messagePart.getContent();

        // pokud je to multipart zprava - ziskej prvni BodyPart
        if (content instanceof Multipart)
            messagePart=((Multipart)content).getBodyPart(0);

        // typ obsahu casti zpravy
        String contentType=messagePart.getContentType();

        // nacteni a vypsani obsahu zpravy
        if (contentType.startsWith("text/plain") || contentType.startsWith("text/html")) {
            InputStream is = messagePart.getInputStream();

            BufferedReader reader = new BufferedReader(new InputStreamReader(is));
            String thisLine=reader.readLine();

            while (thisLine != null) {
                System.out.println(thisLine);
                thisLine=reader.readLine();
            }
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}


/*****************************
 * Nacteni parametru programu *
 *****************************/
```

```
    private static String readParam(String paramName) {

        String sdata;
        byte[] data = new byte[80];

        try {
            System.out.print(paramName + ": ");
            System.in.read(data);
            sdata = new String(data).trim();

            return sdata;
        }

        catch (IOException e) {
            System.out.println("Chyba pri cteni parametru " + paramName);
            return null;
        }
    }


    /*****************************
     * Prijem a verifikace zprav *
     *****************************/
    public static void receiveAndVerifyMessages(String pop3Host, String userAccount, String
userPasswd) {
        try {
            // Pridani DSTC providera pred Sun RSA providera
            Security.insertProviderAt(new com.dstc.security.provider.DSTC(), 2);
            Security.addProvider(
                new com.dstc.security.keymanage.keystore.DSTC());

            // konfigurace na protokol pop3
            System.setProperty("mail.store.protocol", "pop3");

            Session session =
                Session.getDefaultInstance(System.getProperties(), null);

            // vytvoreni spojeni s pop3 serverem
            Store store = session.getStore();
            store.connect(pop3Host, userAccount, userPasswd);

            // otevreni INBOXu
            Folder inbox = store.getFolder("INBOX");
            inbox.open(Folder.READ_WRITE);
            messageCount = inbox.getMessageCount();


            // ziskani zprav z inboxu
            msgs = inbox.getMessages();

            for (int i = 0; i < msgs.length; i++) {
                MimeMessage m = (MimeMessage) msgs[i];

                // pokud je zprava podepsana
                if (SMIMEUtil.isSigned(m)) {
                    // Tisk prijate zpravy do souboru
                    // FileOutputStream fos = new FileOutputStream("message.smi");
                    // m.writeTo(fos);
                    // fos.close();
                    m = verify(m); // verifikace podpisu zpravy
                                   // v pripade uspechu vraci MIME zpravu jinak null
                }
                mimeMessages[i] = m;
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    /************************************************
     * Pridani certifikatu do seznamu duveryhodnych CA *
     ************************************************/
    private static void addTrustedCert(String fileName) {
        X509Certificate cert = getCert(fileName);
        if (cert != null) trusted.add(cert);
    }
```

```
    /*************************
     * Overeni podpisu zpravy *
     *************************/
    private static MimeMessage verify(MimeMessage msg) {
        try {

            SMIMESignature smail2 = new SMIMESignature();
            smail2.initVerify(trusted, null);
            smail2.setMessage(msg);
            VerificationResult res = smail2.verify();
            return res.getMessage();
        }
        catch (Exception e) {
            return null;
        }
    }

    /*********************************
     * Nacteni certifikatu ze souboru *
     *********************************/
    private static X509Certificate getCert(String certFile) {

        try {
            FileInputStream fis = new FileInputStream(certFile);
            DataInputStream dis = new DataInputStream(fis);
            byte[] data = new byte[dis.available()];
            dis.readFully(data);
            ByteArrayInputStream bais = new ByteArrayInputStream(data);
            CertificateFactory fact = CertificateFactory.getInstance("X509");
            X509Certificate cert = (X509Certificate) fact.generateCertificate(bais);

            return cert;
        }

        catch (FileNotFoundException e) {
            System.out.println("    Soubor nenalezen");
            return null;
        }
        catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Příloha F: Zákon o elektronickém podpisu

Profil předpisu:

Titul předpisu:

Zákon o elektronickém podpisu a o změně některých dalších zákonů (zákon o elektronickém podpisu)

Citace: **227/2000 Sb.**

Částka: 68/2000 Sb.

Na straně (od-do): 3290-3297

Rozeslána dne: 26. července 2000

Druh předpisu: Zákon

Autoři předpisu: Parlament

Datum přijetí: 29. června 2000

Datum účinnosti od: 1. října 2000

Platnost předpisu: ANO

Datum účinnosti do:

Hesla rejstříku:

Vydáno na základě:

Předpis mění:

40/1964 Sb.; 337/1992 Sb.; 71/1967 Sb.; 99/1963 Sb.; 141/1961 Sb.; 101/2000 Sb.; 368/1992 Sb.

Předpis ruší:

227

ZÁKON

ze dne 29. června 2000

o elektronickém podpisu a o změně některých dalších zákonů (zákon o elektronickém podpisu)

Parlament se usnesl na tomto zákoně České republiky:

ČÁST PRVNÍ

ELEKTRONICKÝ PODPIS

§ 1

Účel zákona

Tento zákon upravuje používání elektronického podpisu, poskytování souvisejících služeb, kontrolu povinností stanovených tímto zákonem a sankce za porušení povinností stanovených tímto zákonem.

§ 2

Vymezení některých pojmů

Pro účely tohoto zákona se rozumí

a)

elektronickým podpisem údaje v elektronické podobě, které jsou připojené k datové zprávě nebo jsou s ní logicky spojené a které umožňují ověření totožnosti podepsané osoby ve vztahu k datové zprávě,

b)

zaručeným elektronickým podpisem elektronický podpis, který splňuje následující požadavky:

1.
je jednoznačně spojen s podepisující osobou,

2.
umožňuje identifikaci podepisující osoby ve vztahu k datové zprávě,

3.
byl vytvořen a připojen k datové zprávě pomocí prostředků, které podepisující osoba může udržet pod svou výhradní kontrolou,

4.
je k datové zprávě, ke které se vztahuje, připojen takovým způsobem, že je možno zjistit jakoukoliv následnou změnu dat,

c)

datovou zprávou elektronická data, která lze přenášet prostředky pro elektronickou komunikaci a uchovávat na záznamových médiích, používaných při zpracování a přenosu dat elektronickou formou,

d)

podepisující osobou fyzická osoba, která má prostředek pro vytváření podpisu a jedná jménem svým nebo v zastoupení jiné fyzické či právnické osoby,

e)

poskytovatelem certifikačních služeb subjekt, který vydává certifikáty a vede jejich evidenci, případně poskytuje další služby spojené s elektronickými podpisy,

f)

akreditovaným poskytovatelem certifikačních služeb poskytovatel certifikačních služeb, jemuž byla udělena akreditace podle tohoto zákona,

g)

certifikátem datová zpráva, která je vydána poskytovatelem certifikačních služeb, spojuje data pro ověřování podpisů s podepisující osobou a umožňuje ověřit její totožnost,

h)

kvalifikovaným certifikátem certifikát, který má náležitosti stanovené tímto zákonem a byl vydán poskytovatelem certifikačních služeb, splňujícím podmínky, stanovené tímto zákonem pro poskytovatele certifikačních služeb vydávající kvalifikované certifikáty,

i)

daty pro vytváření elektronických podpisů jedinečná data, která podepisující osoba používá k vytváření elektronického podpisu,

j)

daty pro ověřování elektronických podpisů jedinečná data, která se používají pro ověření elektronického podpisu,

k)

prostředkem pro vytváření elektronických podpisů technické zařízení nebo programové vybavení, které se používá k vytváření elektronických podpisů,

l) prostředkem pro ověřování elektronických podpisů technické zařízení nebo programové vybavení, které se používá k ověřování elektronických podpisů,

m) prostředkem pro bezpečné vytváření elektronických podpisů prostředek pro vytváření elektronického podpisu, který splňuje požadavky stanovené tímto zákonem,

n) prostředkem pro bezpečné ověřování elektronických podpisů prostředek pro ověřování podpisu, který splňuje požadavky stanovené tímto zákonem,

o) nástrojem elektronického podpisu technické zařízení nebo programové vybavení, nebo jejich součásti, používané pro zajištění certifikačních služeb nebo pro vytváření nebo ověřování elektronických podpisů,

p) akreditací osvědčení, že poskytovatel certifikačních služeb splňuje podmínky stanovené tímto zákonem pro výkon činnosti akreditovaného poskytovatele certifikačních služeb.

§ 3

Soulad s požadavky na podpis

(1) Datová zpráva je podepsána, pokud je opatřena elektronickým podpisem.

(2) Použití zaručeného elektronického podpisu založeného na kvalifikovaném certifikátu a vytvořeného pomocí prostředku pro bezpečné vytváření podpisu umožňuje ověřit, že datovou zprávu podepsala osoba uvedená na tomto kvalifikovaném certifikátu.

§ 4

Soulad s originálem

Použití zaručeného elektronického podpisu zaručuje, že dojde-li k porušení obsahu datové zprávy od okamžiku, kdy byla podepsána, toto porušení bude možno zjistit.

§ 5

Povinnosti podepisující osoby

(1) Podepisující osoba je povinna

a)

zacházet s prostředky, jakož i s daty pro vytváření zaručeného elektronického podpisu s náležitou péčí tak, aby nemohlo dojít k jejich neoprávněnému použití,

b)

uvědomit neprodleně poskytovatele certifikačních služeb, který jí vydal kvalifikovaný certifikát, o tom, že hrozí nebezpečí zneužití jejích dat pro vytváření zaručeného elektronického podpisu,

c)

podávat přesné, pravdivé a úplné informace poskytovateli certifikačních služeb ve vztahu ke kvalifikovanému certifikátu.

(2) Za škodu způsobenou porušením povinností podle odstavce 1 odpovídá podepisující osoba podle zvláštních právních předpisů.[1)] Odpovědnosti se však zprostí, pokud prokáže, že ten, komu vznikla škoda, neprovedl veškeré úkony potřebné k tomu, aby si ověřil, že zaručený elektronický podpis je platný a jeho kvalifikovaný certifikát nebyl zneplatněn.

§ 6

Povinnosti poskytovatele certifikačních služeb vydávajícího kvalifikované certifikáty

(1) Poskytovatel certifikačních služeb, který vydává kvalifikované certifikáty, je povinen

a) zajistit, aby certifikáty jím vydané jako kvalifikované obsahovaly všechny náležitosti kvalifikovaných certifikátů stanovené tímto zákonem,

b) zajistit, aby údaje uvedené v kvalifikovaných certifikátech byly přesné, pravdivé a úplné,

c) před vydáním kvalifikovaného certifikátu bezpečně ověřit odpovídajícími prostředky totožnost osoby, které kvalifikovaný certifikát vydává, případně i její zvláštní znaky, vyžaduje-li to účel kvalifikovaného certifikátu,

d) zjistit, zda v okamžiku vydání kvalifikovaného certifikátu měla podepisující osoba data pro vytváření elektronických podpisů odpovídající datům pro ověřování elektronických podpisů, která obsahuje kvalifikovaný certifikát,

e) zajistit, aby se každý mohl ujistit o identitě poskytovatele certifikačních služeb a jeho kvalifikovaném certifikátu,

f) zajistit provozování bezpečného a veřejně přístupného seznamu vydaných kvalifikovaných certifikátů, a to i dálkovým přístupem, a údaje v něm obsažené při každé změně okamžitě aktualizovat,

g) zajistit provozování bezpečného a veřejně přístupného seznamu kvalifikovaných certifikátů, které byly zneplatněny, a to i dálkovým přístupem,

h) zajistit, aby datum a čas s uvedením hodiny, minuty a sekundy, kdy je kvalifikovaný certifikát vydán nebo zneplatněn, mohly být přesně určeny a tyto údaje byly dostupné třetím stranám,

i) přijímat do pracovního nebo obdobného poměru osoby, které mají odborné znalosti, zkušenosti a kvalifikaci nezbytnou pro poskytované služby, a které jsou obeznámeny s příslušnými bezpečnostními postupy,

j) používat bezpečné systémy a nástroje elektronického podpisu a zajistit dostatečnou bezpečnost postupů, které tyto systémy a nástroje podporují; nástroj elektronického podpisu je bezpečný, pokud odpovídá požadavkům stanoveným tímto zákonem a prováděcí vyhláškou; toto musí být ověřeno Úřadem pro ochranu osobních údajů (dále jen "Úřad"),

k) přijmout odpovídající opatření proti zneužití a padělání kvalifikovaných certifikátů a zajistit utajení dat pro vytváření zaručených elektronických podpisů v případě, že poskytovatel certifikačních služeb umožňuje podepisující osobě jejich vytvoření v rámci poskytovaných služeb,

l) mít k dispozici dostatečné finanční zdroje na provoz v souladu s požadavky uvedenými v tomto zákoně a s ohledem na riziko odpovědnosti za škody,

m) uchovávat veškeré informace a dokumentaci o vydaných kvalifikovaných certifikátech po dobu nejméně 10 let od ukončení platnosti kvalifikovaného certifikátu; informace a dokumentaci může uchovávat v elektronické podobě,

n)

před uzavřením smluvního vztahu s osobou, která žádá o vydání kvalifikovaného certifikátu, informovat ji písemně o přesných podmínkách pro užívání kvalifikovaného certifikátu, včetně případných omezení pro jeho použití, a o podmínkách reklamací; je rovněž povinen tuto osobu informovat o tom, zda je či není akreditován Úřadem podle § 10; tyto informace lze předat elektronicky; podstatné části těchto informací musí být na vyžádání k dispozici třetím osobám, které se spoléhají na tento kvalifikovaný certifikát,

o)

používat bezpečný systém pro uchovávání kvalifikovaných certifikátů v ověřitelné podobě takovým způsobem, aby záznamy nebo jejich změny mohly provádět pouze pověřené osoby, aby bylo možno kontrolovat správnost záznamů a aby jakékoliv technické nebo programové změny porušující tyto bezpečnostní požadavky byly zjevné.

(2) Poskytovatel certifikačních služeb, který vydává kvalifikované certifikáty, vydává podepisujícím osobám kvalifikované certifikáty na základě smlouvy. Smlouva musí být písemná, jinak je neplatná.

(3) Poskytovatel certifikačních služeb, který vydává kvalifikované certifikáty, nesmí uchovávat a kopírovat data pro vytváření zaručeného elektronického podpisu osob, kterým poskytuje své certifikační služby.

(4) Pokud byla poskytovateli certifikačních služeb, který vydává kvalifikované certifikáty, akreditace Úřadem odňata, je povinen informovat o této skutečnosti subjekty, kterým poskytuje své certifikační služby, a uvést tuto skutečnost v seznamech vedených podle odstavce 1 písm. f) a g).

(5) Není-li poskytovatel certifikačních služeb akreditován Úřadem, je povinen ohlásit Úřadu nejméně 30 dnů před vydáním prvního kvalifikovaného certifikátu, že bude vydávat kvalifikované certifikáty.

(6) Pokud poskytovatel certifikačních služeb, který vydává kvalifikované certifikáty, uvede v

kvalifikovaném certifikátu omezení pro použití tohoto certifikátu včetně omezení hodnoty transakce, pro kterou lze kvalifikovaný certifikát použít, musí být tato omezení rozpoznatelná třetími stranami.

(7) Poskytovatel certifikačních služeb, který vydává kvalifikované certifikáty, musí neprodleně ukončit platnost certifikátu, pokud o to podepisující osoba požádá, nebo v případě, že byl certifikát vydán na základě nepravdivých nebo chybných údajů.

(8) Poskytovatel certifikačních služeb musí rovněž ukončit platnost kvalifikovaného certifikátu, dozví-li se prokazatelně, že podepisující osoba zemřela nebo ji soud způsobilosti k právním úkonům zbavil nebo omezil,[2)] nebo pokud údaje, na základě kterých byl certifikát vydán, přestaly platit.

(9) O veškeré činnosti poskytovatele certifikačních služeb, který vydává kvalifikované certifikáty, musí být vedena provozní dokumentace, která musí obsahovat tyto údaje:

a) smlouvu s podepisující osobou o vydání kvalifikovaného certifikátu,

b) vydaný kvalifikovaný certifikát,

c) kopie předložených osobních dokladů podepisující osoby,

d) potvrzení o převzetí kvalifikovaného certifikátu podepisující osobou,

e) přesné časové určení doby platnosti vydaného kvalifikovaného certifikátu.

(10) Zaměstnanci poskytovatele certifikačních služeb, který vydává kvalifikované certifikáty, případně jiné fyzické osoby, které přicházejí do styku s osobními údaji a daty pro vytváření elektronických podpisů podepisujících osob, jsou povinni zachovávat mlčenlivost o osobních údajích, datech pro vytváření elektronických podpisů a o bezpečnostních opatřeních, jejichž zveřejnění by ohrozilo zabezpečení osobních údajů a dat pro vytváření elektronických

podpisů. Povinnost mlčenlivosti trvá i po skončení zaměstnání nebo příslušných prací.

§ 7

Odpovědnost za škodu

(1) Za škodu způsobenou porušením povinností stanovených tímto zákonem odpovídá poskytovatel certifikačních služeb vydávající kvalifikované certifikáty podle zvláštních právních předpisů.[1)]

(2) Poskytovatel certifikačních služeb neodpovídá za škodu vyplývající z použití kvalifikovaného certifikátu, která vznikla v důsledku nedodržení omezení pro jeho použití.

§ 8

Ochrana osobních údajů

Ochrana osobních údajů se řídí zvláštním právním předpisem.[3)]

§ 9

Akreditace a dozor

(1) Udělování akreditací k působení jako akreditovaný poskytovatel certifikačních služeb, jakož i dozor nad dodržováním tohoto zákona náleží Úřadu.

(2) Úřad

a) uděluje a odnímá akreditace k působení jako akreditovaný poskytovatel certifikačních služeb subjektům působícím na území České republiky,

b)

vykonává dozor nad činností akreditovaných poskytovatelů certifikačních služeb a poskytovatelů certifikačních služeb vydávajících kvalifikované certifikáty, ukládá jim opatření k nápravě a pokuty za porušení povinností podle tohoto zákona,

c)

vede evidenci udělených akreditací a jejich změn a evidenci poskytovatelů certifikačních služeb, kteří Úřadu oznámili, že vydávají kvalifikované certifikáty,

d)

pravidelně uveřejňuje přehled udělených akreditací a přehled poskytovatelů certifikačních služeb vydávajících kvalifikované certifikáty, a to i způsobem umožňujícím dálkový přístup,

e)

vyhodnocuje shodu nástrojů elektronického podpisu s požadavky stanovenými tímto zákonem a prováděcí vyhláškou,

f)

plní další povinnosti stanovené tímto zákonem (například § 10 odst. 7, § 13 odst. 2 a § 16 odst. 2).

(3) Za účelem výkonu dozoru je akreditovaný poskytovatel certifikačních služeb vydávající kvalifikované certifikáty povinen pověřeným zaměstnancům Úřadu umožnit v nezbytně nutném rozsahu vstup do obchodních a provozních prostor, na požádání předložit veškerou dokumentaci, záznamy, doklady, písemnosti a jiné podklady související s jeho činností, umožnit jim v nezbytně nutné míře přístup do svého informačního systému a poskytnout informace a veškerou potřebnou součinnost.

(4) Není-li tímto zákonem stanoveno jinak, postupuje Úřad při výkonu dozoru podle zvláštního právního předpisu.[4)]

§ 10

Podmínky udělení akreditace
pro poskytování certifikačních služeb

(1) Každý poskytovatel certifikačních služeb může požádat Úřad o udělení akreditace pro výkon činnosti akreditovaného poskytovatele certifikačních služeb. Podání žádosti o akreditaci podléhá správnímu poplatku.[5)]

(2) V žádosti o akreditaci podle odstavce 1 musí žadatel doložit

a) obchodní jméno, sídlo a identifikační číslo žadatele,

b) doklad o oprávnění k podnikatelské činnosti a u osoby zapsané do obchodního rejstříku také výpis z obchodního rejstříku ne starší než 3 měsíce,

c) výpis z rejstříku trestů podnikatele - fyzické osoby nebo statutárních představitelů právnické osoby v případě, že žadatelem je právnická osoba, ne starší než 3 měsíce,

d) věcné, personální a organizační předpoklady pro činnost poskytovatele certifikačních služeb vydávajícího kvalifikované certifikáty podle § 6 tohoto zákona,

e) údaj o tom, zda žadatel již vydává nebo hodlá vydávat kvalifikované certifikáty,

f) doklad o zaplacení správního poplatku.

(3) Jestliže žádost neobsahuje všechny požadované údaje, Úřad řízení přeruší a vyzve žadatele, aby ji ve stanovené lhůtě doplnil. Jestliže tak žadatel v této lhůtě neučiní, Úřad řízení zastaví. Správní poplatek se v takovém případě nevrací.

(4) Splňuje-li žadatel všechny podmínky předepsané tímto zákonem pro udělení akreditace, vydá Úřad rozhodnutí, jímž mu akreditaci udělí. V opačném případě žádost o udělení akreditace zamítne.

(5) Akreditovaný poskytovatel certifikačních služeb musí mít sídlo na území České republiky.

(6) Kromě činností uvedených v tomto zákoně může akreditovaný poskytovatel certifikačních služeb bez souhlasu Úřadu působit jen jako advokát, notář nebo znalec.[6)]

(7) Součástí rozhodnutí Úřadu o akreditaci je ověření kvalifikovaného certifikátu poskytovatele certifikačních služeb Úřadem.

§ 11

V oblasti orgánů veřejné moci je možné používat pouze zaručené elektronické podpisy a kvalifikované certifikáty vydávané akreditovanými poskytovateli certifikačních služeb.

§ 12

Náležitosti kvalifikovaného certifikátu

(1) Kvalifikovaný certifikát musí obsahovat

a) označení, že je vydán jako kvalifikovaný certifikát podle tohoto zákona,

b) obchodní jméno poskytovatele certifikačních služeb a jeho sídlo, jakož i údaj, že certifikát byl vydán v České republice,

c)

jméno a příjmení podepisující osoby nebo její pseudonym s příslušným označením, že se jedná o pseudonym,

d)

zvláštní znaky podepisující osoby, vyžaduje-li to účel kvalifikovaného certifikátu,

e)

data pro ověřování podpisu, která odpovídají datům pro vytváření podpisu, jež jsou pod kontrolou podepisující osoby,

f)

zaručený elektronický podpis poskytovatele certifikačních služeb, který kvalifikovaný certifikát vydává,

g)

číslo kvalifikovaného certifikátu unikátní u daného poskytovatele certifikačních služeb,

h)

počátek a konec platnosti kvalifikovaného certifikátu,

i)

případně údaje o tom, zda se používání kvalifikovaného certifikátu omezuje podle povahy a rozsahu jen pro určité použití,

j)

případně omezení hodnot transakcí, pro něž lze kvalifikovaný certifikát použít.

(2) Další osobní údaje smí kvalifikovaný certifikát obsahovat jen se svolením podepisující osoby.

§ 13

Povinnosti akreditovaného poskytovatele certifikačních služeb při ukončení činnosti

(1) Akreditovaný poskytovatel certifikačních služeb musí záměr ukončit svou činnost ohlásit Úřadu nejméně 3 měsíce před plánovaným datem ukončení činnosti a musí vynaložit veškeré možné úsilí na to, aby platné kvalifikované certifikáty byly převzaty jiným akreditovaným poskytovatelem certifikačních služeb. Akreditovaný poskytovatel certifikačních služeb dále musí prokazatelně informovat každou podepisující osobu, které poskytuje své certifikační služby, o svém záměru ukončit svoji činnost nejméně 2 měsíce předem.

(2) Nemůže-li akreditovaný poskytovatel certifikačních služeb zajistit, aby platné kvalifikované certifikáty převzal jiný akreditovaný poskytovatel certifikačních služeb, je povinen na to včas Úřad upozornit. V takovém případě Úřad převezme evidenci vydaných kvalifikovaných certifikátů a oznámí to dotčeným podepisujícím osobám.

(3) Ustanovení odstavců 1 a 2 se použijí přiměřeně také v případě, když akreditovaný poskytovatel certifikačních služeb zanikne, zemře nebo přestane vykonávat svoji činnost, aniž splní ohlašovací povinnost podle odstavce 1.

§ 14

Opatření k nápravě

(1) Zjistí-li Úřad, že akreditovaný poskytovatel certifikačních služeb nebo poskytovatel certifikačních služeb vydávající kvalifikované certifikáty porušuje povinnosti stanovené tímto zákonem, uloží mu, aby ve stanovené lhůtě sjednal nápravu, a případně určí, jaká opatření k odstranění nedostatků je tento poskytovatel certifikačních služeb povinen přijmout.

(2) V případě, že se akreditovaný poskytovatel certifikačních služeb dopustí závažnějšího porušení povinností stanovených tímto zákonem nebo ve stanovené lhůtě neodstraní nedostatky zjištěné Úřadem, je Úřad oprávněn mu udělenou akreditaci odejmout.

(3) Rozhodne-li Úřad o odnětí akreditace, může ukončit současně platnost kvalifikovaných certifikátů vydaných poskytovatelem certifikačních služeb v době platnosti akreditace.

§ 15

Zrušení kvalifikovaného certifikátu

(1) Úřad může nařídit poskytovateli certifikačních služeb jako předběžné opatření[7)] zneplatnění kvalifikovaného certifikátu podepisující osoby, pokud existuje důvodné podezření, že kvalifikovaný certifikát byl padělán, nebo pokud byl vydán na základě nepravdivých údajů. Nařízení o zneplatnění kvalifikovaného certifikátu může být vydáno také v případě, kdy bylo zjištěno, že podepisující osoba používá prostředek pro vytváření podpisu, který vykazuje bezpečnostní nedostatky, které by umožnily padělání zaručených elektronických podpisů nebo změnu podepisovaných údajů.

(2) Seznam certifikátů podle § 6 odst. 1 písm. g) musí obsahovat přesný časový údaj, od kdy byl certifikát zneplatněn. Zneplatněné certifikáty není povoleno opětovně zprovoznit a používat.

§ 16

Uznávání zahraničních certifikátů

(1) Certifikát, který je vydán zahraničním poskytovatelem certifikačních služeb jako kvalifikovaný ve smyslu tohoto zákona, může být používán jako kvalifikovaný certifikát tehdy, je-li uznán poskytovatelem certifikačních služeb, který vydává kvalifikované certifikáty podle tohoto zákona, a za podmínky, že tento poskytovatel certifikačních služeb zaručí ve stejném rozsahu jako u svých kvalifikovaných certifikátů správnost a platnost kvalifikovaného certifikátu vydaného v zahraničí.

(1) Akreditovanému poskytovateli certifikačních služeb nebo poskytovateli certifikačních služeb vydávajícímu kvalifikované certifikáty, který poruší povinnost uloženou mu tímto

zákonem, může Úřad uložit pokutu až do výše 10 000 000 Kč.

(2) Pokud akreditovaný poskytovatel certifikačních služeb nebo poskytovatel certifikačních služeb vydávající kvalifikované certifikáty porušil do jednoho roku ode dne, kdy nabylo rozhodnutí o uložení pokuty právní moci, povinnosti uložené mu tímto zákonem opakovaně, může mu být uložena pokuta do výše 20 000 000 Kč.

(3) Akreditovaný poskytovatel certifikačních služeb nebo poskytovatel certifikačních služeb vydávající kvalifikované certifikáty, který maří kontrolu prováděnou Úřadem, může být potrestán pořádkovou pokutou do výše 1 000 000 Kč, a to i opakovaně.

(4) Osobě, která, byť z nedbalosti, neposkytne Úřadu při výkonu kontroly potřebnou součinnost, může být uložena pokuta do výše 25 000 Kč, a to i opakovaně.

(5) Při rozhodování o výši pokuty se přihlíží zejména ke způsobu jednání, míře zavinění, závažnosti, rozsahu, době trvání a následkům protiprávního jednání.

(6) Pokutu lze uložit do jednoho roku ode dne, kdy příslušný orgán porušení povinnosti zjistil, nejdéle však do tří let ode dne, kdy k porušení povinnosti došlo.

(7) Pokutu vybírá Úřad. Pokutu vymáhá územní finanční orgán podle zvláštního právního předpisu.[8)]

(8) Výnos pokut je příjmem státního rozpočtu České republiky.

§ 19

Není-li v tomto zákoně stanoveno jinak, vztahuje se na řízení podle tohoto zákona zvláštní právní předpis.[9)]

§ 20

Zmocňovací ustanovení

Úřad se zmocňuje vydávat vyhlášky k upřesňování podmínek stanovených v § 6 a 17 a způsobu, jakým se jejich splnění bude dokládat, a k upřesnění požadavků, které musí splňovat nástroje elektronického podpisu, a k náležitostem postupu a způsobu vyhodnocování shody nástrojů elektronického podpisu s těmito požadavky.

ČÁST DRUHÁ

Změna občanského zákoníku

§ 21

Zákon č. 40/1964 Sb., občanský zákoník, ve znění zákona č. 58/1969 Sb., zákona č. 131/1982 Sb., zákona č. 94/1988 Sb., zákona č. 188/1988 Sb., zákona č. 87/1990 Sb., zákona č. 105/1990 Sb., zákona č. 116/1990 Sb., zákona č. 87/1991 Sb., zákona č. 509/1991 Sb., zákona č. 264/1992 Sb., zákona č. 267/1994 Sb., zákona č. 104/1995 Sb., zákona č. 118/1995 Sb., zákona č. 89/1996 Sb., zákona č. 94/1996 Sb., zákona č. 227/1997 Sb., zákona č. 91/1998 Sb., zákona č. 165/1998 Sb., zákona č. 159/1999 Sb., zákona č. 363/1999 Sb., zákona č. 27/2000 Sb. a zákona č. 103/2000 Sb., se mění takto:

V § 40 odst. 3 se doplňuje tato věta: "Je-li právní úkon učiněn elektronickými prostředky, může být podepsán elektronicky podle zvláštních předpisů.".

ČÁST TŘETÍ

Změna zákona č. 337/1992 Sb.,o správě daní a poplatků

§ 22

Zákon č. 337/1992 Sb., o správě daní a poplatků, ve znění zákona č. 35/1993 Sb., zákona č. 157/1993 Sb., zákona č. 302/1993 Sb., zákona č. 315/1993 Sb., zákona č. 323/1993 Sb., zákona č. 85/1994 Sb., zákona č. 255/1994 Sb., zákona č. 59/1995 Sb., zákona č. 118/1995 Sb., zákona č. 323/1996 Sb., zákona č. 61/1997 Sb., zákona č. 242/1997 Sb., zákona č. 91/1998 Sb., zákona č. 168/1998 Sb., zákona č. 29/2000 Sb., zákona č. 159/2000 Sb. a zákona č. 218/2000 Sb., se mění takto:

V § 21 odstavce 2 a 3 znějí:

"(2) Stanoví-li tak tento nebo zvláštní zákon, podávají daňové subjekty o své daňové povinnosti příslušnému správci daně přiznání, hlášení a vyúčtování na předepsaných tiskopisech. Tiskopisy zveřejněné v elektronické podobě lze podepsat elektronicky podle zvláštních předpisů.

(3) Jiná podání v daňových věcech, jako jsou oznámení, žádosti, návrhy, námitky, odvolání apod., lze učinit buď písemně nebo ústně do protokolu nebo elektronicky podepsané podle zvláštních předpisů či za použití jiných přenosových technik (dálnopis, telefax apod.).".

ČÁST ČTVRTÁ

Změna správního řádu

§ 23

Zákon č. 71/1967 Sb., o správním řízení (správní řád), ve znění zákona č. 29/2000 Sb., se mění takto:

V § 19 odstavec 1 zní:

"(1) Podání lze učinit písemně nebo ústně do protokolu nebo v elektronické podobě podepsané elektronicky podle zvláštních předpisů. Lze je též učinit telegraficky; takové podání obsahující návrh ve věci je třeba písemně nebo ústně do protokolu doplnit nejpozději do 3 dnů.".

ČÁST PÁTÁ

Změna občanského soudního řádu

§ 24

Zákon č. 99/1963 Sb., občanský soudní řád, ve znění zákona č. 36/1967 Sb., zákona č. 158/1969 Sb., zákona č. 49/1973 Sb., zákona č. 20/1975 Sb., zákona č. 133/1982 Sb., zákona č. 180/1990 Sb., zákona č. 328/1991 Sb., zákona č. 519/1991 Sb., zákona č. 263/1992 Sb., zákona č. 24/1993 Sb., zákona č. 171/1993 Sb., zákona č. 117/1994 Sb., zákona č. 152/1994 Sb., zákona č. 216/1994 Sb., zákona č. 84/1995 Sb., zákona č. 118/1995 Sb., zákona č. 160/1995 Sb., zákona č. 238/1995 Sb., zákona č. 247/1995 Sb., nálezu Ústavního soudu č. 31/1996 Sb., zákona č. 142/1996 Sb., nálezu Ústavního soudu č. 269/1996 Sb., zákona č. 202/1997 Sb., zákona č. 227/1997 Sb., zákona č. 15/1998 Sb., zákona č. 91/1998 Sb., zákona č. 165/1998 Sb., zákona č. 326/1999 Sb., zákona č. 360/1999 Sb., nálezu Ústavního soudu č. 2/2000 Sb., zákona č. 27/2000 Sb., zákona č. 30/2000 Sb., zákona č. 46/2000 Sb., zákona č. 105/2000 Sb., zákona č. 130/2000 Sb., zákona č. 155/2000 Sb. a zákona č. 220/2000 Sb., se mění takto:

V § 42 odst. 1 věta první zní: "Podání je možno učinit písemně, ústně do protokolu, v elektronické podobě podepsané elektronicky podle zvláštních předpisů, telegraficky nebo telefaxem.".

ČÁST ŠESTÁ

Změna trestního řádu

§ 25

Zákon č. 141/1961 Sb., o trestním řízení soudním (trestní řád), ve znění zákona č. 57/1965 Sb., zákona č. 58/1969 Sb., zákona č. 149/1969 Sb., zákona č. 48/1973 Sb., zákona č. 29/1978 Sb., zákona č. 43/1980 Sb., zákona č. 159/1989 Sb., zákona č. 178/1990 Sb., zákona č. 303/1990 Sb., zákona č. 558/1991 Sb., zákona č. 25/1993 Sb., zákona č. 115/1993 Sb., zákona č. 292/1993 Sb., zákona č. 154/1994 Sb., nálezu Ústavního soudu č. 214/1994 Sb., nálezu Ústavního soudu č. 8/1995 Sb., zákona č. 152/1995 Sb., zákona č. 150/1997 Sb., zákona č. 209/1997 Sb., zákona č. 148/1998 Sb., zákona č. 166/1998 Sb., zákona č. 191/1999 Sb., zákona č. 29/2000 Sb. a zákona č. 30/2000 Sb., se mění takto:

V § 59 odstavec 1 zní:

"(1) Podání se posuzuje vždy podle svého obsahu, i když je nesprávně označeno. Lze je učinit písemně, ústně do protokolu, v elektronické podobě podepsané elektronicky podle zvláštních předpisů, telegraficky, telefaxem nebo dálnopisem.".

ČÁST SEDMÁ

Změna zákona o ochraně osobních údajů

§ 26

Zákon č. 101/2000 Sb., o ochraně osobních údajů a o změně některých zákonů, se mění takto:

V § 29 se doplňuje odstavec 4, který zní:

"(4) Úřad uděluje a odnímá akreditace k působení jako akreditovaný poskytovatel certifikačních služeb a provádí dozor nad dodržováním povinností stanovených zákonem o elektronickém podpisu.".

ČÁST OSMÁ

Změna zákona o správních poplatcích

§ 27

Zákon č. 368/1992 Sb., o správních poplatcích, ve znění zákona č. 10/1993 Sb., zákona č. 72/1994 Sb., zákona č. 85/1994 Sb., zákona č. 273/1994 Sb., zákona č. 36/1995 Sb., zákona č. 118/1995 Sb., zákona č. 160/1995 Sb., zákona č. 301/1995 Sb., zákona č. 151/1997 Sb., zákona č. 305/1997 Sb., zákona č. 149/1998 Sb., zákona č. 157/1998 Sb., zákona č. 167/1998 Sb., zákona č. 63/1999 Sb., zákona č. 166/1999 Sb., zákona č. 167/1999 Sb., zákona č. 223/1999 Sb., zákona č. 326/1999 Sb., zákona č. 352/1999 Sb., zákona č. 357/1999 Sb., zákona č. 360/1999 Sb., zákona č. 363/1999 Sb., zákona č. 46/2000 Sb., zákona č. 62/2000 Sb., zákona č. 117/2000 Sb., zákona č. 133/2000 Sb., zákona č. 151/2000 Sb., zákona č. 153/2000 Sb., zákona č. 154/2000 Sb., zákona č. 156/2000 Sb. a zákona č. 158/2000 Sb., se mění takto:

1. V příloze k zákonu (Sazebník správních poplatků) se doplňuje nová část XII, která zní:

"ČÁST XII

ŘÍZENÍ PODLE ZÁKONA O ELEKTRONICKÉM PODPISU

Položka 162

a)

podání žádosti o akreditaci poskytovatele certifikačních služeb Kč 100 000,-

b)

podání žádosti o vyhodnocení shody nástrojů elektronického podpisu s požadavky Kč 10 000,-.".

2. **REJSTŘÍK K SAZEBNÍKU** se doplňuje o část XII, která zní:

"ČÁST XII

Řízení podle zákona o elektronickém podpisu 162.".

3. Tečka za částí XI se vypouští.

ČÁST DEVÁTÁ

ÚČINNOST

§ 28

Tento zákon nabývá účinnosti prvním dnem třetího kalendářního měsíce po dni jeho vyhlášení.

Klaus v. r.

Havel v. r.

Zeman v. r.

1)

Zákon č. 40/1964 Sb., občanský zákoník, ve znění pozdějších předpisů.

2)

§ 10 zákona č. 40/1964 Sb., ve znění zákona č. 509/1991 Sb.

3)

Zákon č. 101/2000 Sb., o ochraně osobních údajů a o změně některých zákonů.

4)

Zákon č. 552/1991 Sb., o státní kontrole, ve znění pozdějších předpisů.

5)

Zákon č. 368/1992 Sb., o správních poplatcích, ve znění pozdějších předpisů.

6)

Zákon č. 85/1996 Sb., o advokacii, ve znění zákona č. 210/1999 Sb.

Zákon č. 358/1992 Sb., o notářích a jejich činnosti (notářský řád), ve znění pozdějších předpisů.

Zákon č. 36/1967 Sb., o znalcích a tlumočnících.

7)

§ 43 zákona č. 71/1967 Sb., o správním řízení (správní řád).

8)

Zákon č. 337/1992 Sb., o správě daní a poplatků, ve znění pozdějších předpisů.

9)

Zákon č. 71/1967 Sb., ve znění zákona č. 29/2000 Sb.

Soulasím s prezenčním půjčováním práce v Univerzitní knihovně

V Plzni dne 14. května ……

Josef Steinberger