

Modul pro asymetrickou kryptografii

Using of the Enterprise Java Bean Module for Asymmetric
Cryptography

Bc. Petr Šimkovič

Diplomová práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Petr ŠIMKOVIČ**
Osobní číslo: **A10444**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Bezpečnostní technologie, systémy a management**

Téma práce: **Modul pro asymetrickou kryptografii**

Zásady pro vypracování:

1. Analyzujte současný stav dostupných modulů Enterprise Java Beans (EJB), implementujících funkce asymetrické kryptografie (Public Key Infrastructure – PKI).
2. Navrhněte API vlastního modulu EJB, který bude umožňovat podepsání dat pomocí PKI a ověření podpisu.
3. Analyzujte problém ověření pravosti podepsaných dat po vypršení certifikátu pomocí mechanismu časových známek a implementujte jej.
4. Implementujte ukázkovou aplikaci, využívající modulu EJB, na příkladu následujících funkcí: odeslání emailu s S/MIME, ověření S/MIME, podepsání PDF souboru.
5. Implementujte rozhraní pro PKI modul pomocí BPEL s využitím dalších EJB modulů, využívajících např. JAX-WS.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. VASILIEV, Juri. *Biginning Database-Driven Application Development in JAVA EE using GlassFish*. 1. vyd. Apress, 2008, 409 s. ISBN 978-1-4302-0963-8.
2. SALTER, David a JENNINGS, Frank. *Building SOA-Based Composite Applications Using Netbeans IDE 6*. 1. vyd. Birgmingham: Packt publishing Ltd., 2008, 287 s. ISBN 978-1-847192-62-2.
3. DOSTÁLEK, Libor a VOHNOUTOVÁ, Marta. *Velký průvodce infrastrukturou PKI a technologií elektronického*. 1. vyd. Computer Press, 2008, 536 s. ISBN 9788025108284.
4. *Security Features in Java SE*. The Java Tutorials [online]. 1995. Dostupné z: <http://docs.oracle.com/javase/tutorial/security/index.html>.
5. *Java XML Digital Signatures*. ORACLE Sun Developer Network [online]. 2010. Dostupné z: http://java.sun.com/developer/technicalArticles/xml/dig_signatures/.
6. KEITH, Mike a Schincariol, Merrick. *Pro EJB 3*. Apress, 2006, 452s. ISBN 978-1-59059-645-6.
7. KNUDSEN, Jonathan B. *Java Cryptography*. 1.vydání OReilly, 1998, 362s. ISBN 1-56592-402-9.

Vedoucí diplomové práce:

Ing. Tomáš Dulík

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

15. května 2012

Ve Zlíně dne 24. února 2012

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. RNDr. Vojtěch Křesálek, CSc.
ředitel ústavu

ABSTRAKT

Práce se zabývá digitální podpisem, časovým razítkem a zabezpečením e-mailů. Za svůj cíl si klade implementování PKI EJB modulu, jehož funkcí by vyžívala webová aplikace a rozhraní implementované pomocí jazyka BPEL. V teoretické části je zmíněna problematika certifikátu, včetně jeho ověření. Dále je zmíněna problematika vytvoření a ověření digitálního podpisu, včetně časového razítka. Práce ve své teoretické části popisuje, jak je toto zakotveno v legislativě a popisuje současný stav EJB modulů se zaměřením na PKI. V praktické části se popisuje implementace PKI EJB modulu, webové aplikace a rozhraní postavené nad BPEL.

Klíčová slova: XML, PDF, S/MIME, BPEL, JBI, EJB, PKI, certifikát, webová služba, Java EE, digitální podpis, časové razítko, JAX-WS, BPEL procesy

ABSTRACT

This thesis deals with the digital signature, digital time stamp and e-mail security. The target is to implement the PKI EJB module for the web application and interface implemented in BPEL language. In theory part the thesis describes the issues of the certificate including its verification, the creating and verification of digital signatures and digital time stamp. It describes the law quality in this area and the current EJB modules for PKI. In practical part the thesis describe the implementation of the PKI EJB module, web application and BPEL interface.

Keywords: XML, PDF, S/MIME, BPEL, JBI, EJB, PKI, certificate, web service, Java EE, digital signature, digital time stamp, JAX-WS, BPEL processes

Rád bych poděkoval ing. Tomáši Dulíkovi za ochotu vést tuto práci.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 PKI	11
1.1 CERTIFIKÁT	11
1.1.1 Verze	12
1.1.2 Pořadové číslo	12
1.1.3 Algoritmus podpisu	12
1.1.4 Platnost	12
1.1.5 Vydavatel a předmět.....	12
1.1.6 Veřejný klíč	13
1.1.7 Rozšíření	13
1.2 ŽIVOTNÍ CYKLUS CERTIFIKÁTU	14
1.3 OVĚŘENÍ CERTIFIKÁTU	15
2 DIGITÁLNÍ PODPIS	17
2.1 XML PODPIS	19
2.1.1 Struktura.....	19
2.1.2 Typy.....	20
2.1.3 Vytvoření podpisu	21
2.1.4 Ověření podpisu	22
2.2 PDF PODPIS	23
2.3 S/MIME.....	24
3 DIGITÁLNÍ ČASOVÉ RAZÍTKO	29
3.1 SPECIFIKA FORMÁTŮ.....	30
4 LEGISLATIVA	32
4.1 KVALIFIKOVANÝ CERTIFIKÁT	32
4.2 KVALIFIKOVANÝ SYSTÉMOVÝ CERTIFIKÁT	32
4.3 KOMERČNÍ CERTIFIKÁT.....	33
4.4 ELEKTRONICKÝ PODPIS.....	34
4.5 ZARUČENÝ ELEKTRONICKÝ PODPIS.....	34
4.6 ELEKTRONICKÁ ZNAČKA	35
4.7 LEGISLATIVA V PRAXI ANEB CO Z TOHO VYPLÝVÁ.....	35
5 SOUČASNÉ EJB MODULY PRO PKI	37
II PRAKTICKÁ ČÁST	40
6 POHLED NA CELÝ SYSTÉM	41
6.1 BASE64	42
7 PKI EJB MODUL	44

7.1	VYTVÁŘENÍ ROZŠÍŘENÉHO DIGITÁLNÍHO PODPISU	45
7.1.1	PDF	46
7.1.2	XML	46
7.2	OVĚŘOVÁNÍ ROZŠÍŘENÉHO DIGITÁLNÍHO PODPISU.....	47
7.3	ZASÍLÁNÍ A OVĚŘOVÁNÍ E-MAILŮ	50
7.4	PRÁCE S CERTIFIKÁTY	53
7.5	WEBOVÉ SLUŽBY.....	55
7.5.1	Implemetace	56
7.5.2	Zabezpečení.....	59
8	DATABÁZE.....	61
9	BPEL PROCESY	63
9.1	PŘÍJEM ZPRÁVY Z VNĚJŠÍHO OKOLÍ.....	64
9.2	ZPRACOVÁNÍ POŽADAVKU.....	65
9.3	ODPOVĚĎ A OŠETŘENÍ CHYB BĚHEM ZPRACOVÁNÍ.....	68
10	WEBOVÁ APLIKACE.....	70
10.1	SPOLEČNÝ MODUL.....	70
10.2	EJB MODUL PRO KOMUNIKACI S PKI EJB MODULEM	70
10.3	WEBOVÝ MODUL.....	72
10.3.1	Vytvoření a ověření XML podpisu.....	73
10.3.2	Vytvoření a ověření PDF podpisu	74
10.3.3	Poslání a ověření e-mailu	75
11	LOGOVÁNÍ.....	77
	ZÁVĚR	78
	ZÁVĚR V ANGLIČTINĚ.....	79
	SEZNAM POUŽITÉ LITERATURY	80
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	82
	JAVA ARCHITECTURE FOR XML BINDING – TECHNOLOGIE MAPOVÁNÍ OBJEKTŮ DO XML	83
	SEZNAM OBRÁZKŮ	84
	SEZNAM TABULEK.....	86

ÚVOD

Dřívější způsob komunikace, stejně tak jako podávání různých žádostí nebo obchodování, se prováděl výhradně pomocí papírových dokumentů. Dnes mnoho společností nebo organizací definuje k zajištění svých potřeb tzv. business procesy. A protože většina těchto procesů závisí na elektronických dokumentech, nabízí se otázka jak tyto dokumenty zabezpečit, aby nemohlo dojít např. k jejich zneužití či změně během jejich přenosu použitým komunikačním kanálem.

Jako dobré řešení se dnes jeví opatřit dokument bezpečnostními parametry a prvky, které jsou v něm samy obsaženy. Mezi ně například patří nepopiratelnost, integrita a autenticita. Všechny tyto vlastnosti mohou být pak zajištěny použitím digitálního podpisu, který lze ještě rozšířit o časové razítko.

Je tedy zřejmé, že informační systém společnosti nebo organizace by v sobě měl zahrnovat infrastrukturu, která všechny tyto bezpečnostní prvky, zejména vytvoření a ověření digitálního podpisu, bude nabízet.

I. TEORETICKÁ ČÁST

1 PKI

PKI (Public Key Infrastructure) umožňuje uživatelům bezpečnou výměnu dat pomocí veřejného a soukromého kryptografického klíče, které vydává certifikační autorita (dále jen CA). Těmito klíči se myslí digitální certifikáty identifikující jednotlivé fyzické nebo právnické osoby. PKI lze definovat i jako soustavu technických a především organizačních opatření spojených s vydáváním, správou, používáním a odvoláváním platnosti kryptografických klíčů a certifikátů. [3]

Veřejné a soukromé klíče jsou vytvářeny současně s použitím stejného algoritmu (dnes zřejmě nejpoužívanějším je RSA¹). Soukromý klíč je dán pouze žadateli a neměl by být nikdy sdílen s jinou osobou. Veřejný klíč jako část digitálního certifikátu je, jak už z názvu vyplývá, určen ke sdílení s ostatními uživateli.

1.1 Certifikát

Certifikát se často přirovnává k občanskému průkazu nebo pasu. Zatímco občanský průkaz se vydává v tištěné podobě, certifikát je digitálně podepsanou datovou strukturou, jejíž základní součástí je veřejný klíč držitele certifikátu. V mnoha evropských zemích se dokonce již vydávají občanské průkazy ve tvaru čipové karty, na které jsou uloženy certifikáty držitele této karty.

Existuje několik norem definujících strukturu certifikátu (X.509, EDI, WAP apod.). V Internetu se vychází ze standardu X.509 verze 3, který vydal ITU². Pro potřeby Internetu je vytvořen internetový profil standardu v příslušném RFC, dnes je aktuální RFC-3280³. V následujících sekcích si stručně popíšeme jednotlivé položky certifikátu pomocí citací z knihy. [3] Názvy psané kurzívou znamenají používané anglické názvy.

¹ <http://www.rsa.com/rsalabs/node.asp?id=2125>

² <http://www.itu.int/rec/T-REC-X.509>

³ <http://www.ietf.org/rfc/rfc3280.txt>

1.1.1 Verze

Verze certifikátu souvisí s tím, je-li certifikát odvozen od normy X.509 verze 1, 2 nebo 3. Položka *Version* má v případě verze 1 hodnotu nula, v případě verze 2 hodnotu jedna a v případě verze 3 hodnotu dvě. Dnes se zásadně používají pouze certifikáty verze 3.

1.1.2 Pořadové číslo

Pořadové číslo certifikátu *Serial Number* je definováno jako celé kladné číslo, které musí být jednoznačné v rámci jedné certifikační autority. Tj. autorita nesmí vydat dva certifikáty mající stejné pořadové číslo. Dvojice položek *Serial Number* a *Issuer* (viz dále) je pak jednoznačná identifikace certifikátu.

1.1.3 Algoritmus podpisu

Položka *Signature Algorithm* specifikuje algoritmy použité CA pro vytvoření elektronického podpisu certifikátu. Přesněji vždy specifikuje dvojici algoritmů:

1. Jeden pro výpočet otisku.
2. Druhým algoritmem je asymetrický algoritmus, kterým je otisk šifrován.

1.1.4 Platnost

Validity určuje platnost certifikátu od *Not Before* do *Not After*. Důvody omezení jsou dva:

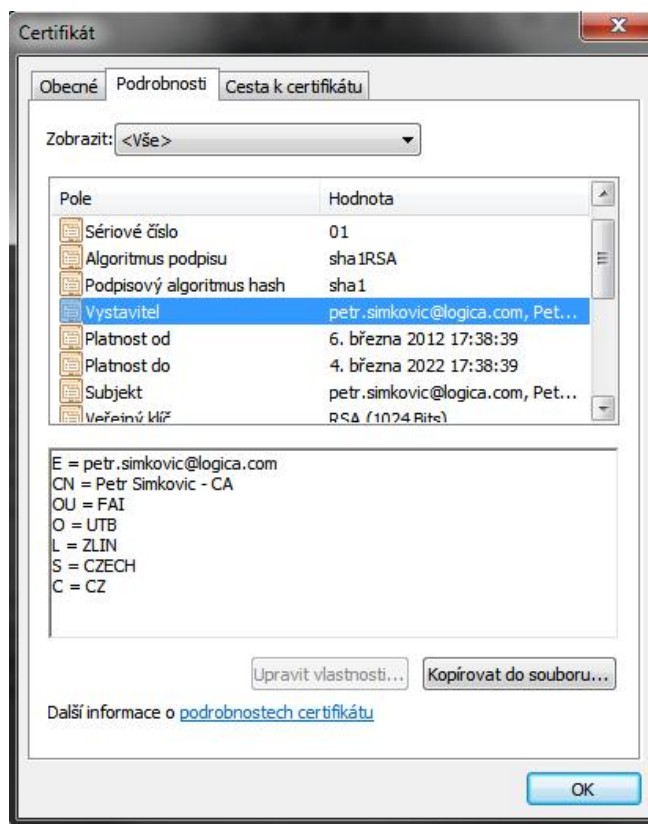
1. Organizační – aplikace má určitou životnost.
2. Bezpečnostní – pádnější důvod. Životnost certifikátu by měla být výrazně kratší než doba nutná k prolomení certifikovaného veřejného klíče.

I po vypršení doby platnosti je certifikát stále potřebný. Např. pro ověření elektronického podpisu nebo dešifrování.

1.1.5 Vydavatel a předmět

Položka *Issuer* obsahuje jedinečné jméno certifikační autority, která certifikát vydala, v rámci všech CA. Podobně položka *Subject* identifikuje držitele certifikátu a musí být jedinečná v rámci certifikátu vydaných jednou CA (přesněji CA nesmí vydat dvěma

různým osobám certifikát se stejným předmětem). Obě používají stejný formát označovaný jako jedinečné jméno (*Distinguished Name*).



Obr. 1.1.1. Vydavatel certifikátu

1.1.6 Veřejný klíč

Položka *Subject Public Key* je sekvencí dvou informací: identifikátorem algoritmu, pro nějž je veřejný klíč určen, a samotným veřejným klíčem.

1.1.7 Rozšíření

Mezi rozšíření certifikátu např. patří platnost soukromého klíče nebo alternativní jméno předmětu. Důležité a používané je rozšíření použití klíče. To se skládá z bitového řetězce, kde každý bit nabývá hodnot pravda nebo nepravda. Následuje přehled a význam jednotlivých bitů.

- Digitální podpis (*Digital Signature*) – certifikát je určen k elektronickému podpisu dat. Netýká se ověřování pravosti, autentizace ani ověřování integrity dat.

- Neodvolatelnost (*Non Repudiation*) – týká se ověření pravosti nebo jinak řečeno nepopiratelnosti.
- Zakódování klíče (*Key Encipherment*) – šifrování klíčů. Např. elektronická obálka, kdy data jsou šifrována náhodným symetrickým klíčem, který je ve zprávě přiložen a zašifrován právě veřejným klíčem z takto označeného certifikátu.
- Zakódování dat (*Data Encipherment*) – veřejný klíč je určen k šifrování dat jiných než šifrovacích klíčů.
- *Key Agreement* – výměna klíčů. Např. DH (Deffie-Hellman) výměna klíčů.
- Podepisování certifikátu (*Key Certificate Sign*) – veřejný klíč je určen pro verifikaci certifikátů (např. pro certifikát CA – kořenový certifikát podepsaný sám sebou).
- Podepisování CRL (*CRL Sign*) – veřejný klíč je určen pro k verifikaci CRL (tj. seznam zneplatněných certifikátů).
- *Encipher Only* – používá se ve spojení s bitem *Key Agreement*. Výsledný symetrický klíč může být použit jen k šifrování.
- *Decipher Only* – používá se ve spojení s bitem *Key Agreement*. Výsledný symetrický klíč může být použit jen k dešifrování.

1.2 Životní cyklus certifikátu

Certifikát v průběhu času prochází několika fázemi tvořícími životní cyklus certifikátu, který je popsán v několika bodech níže.

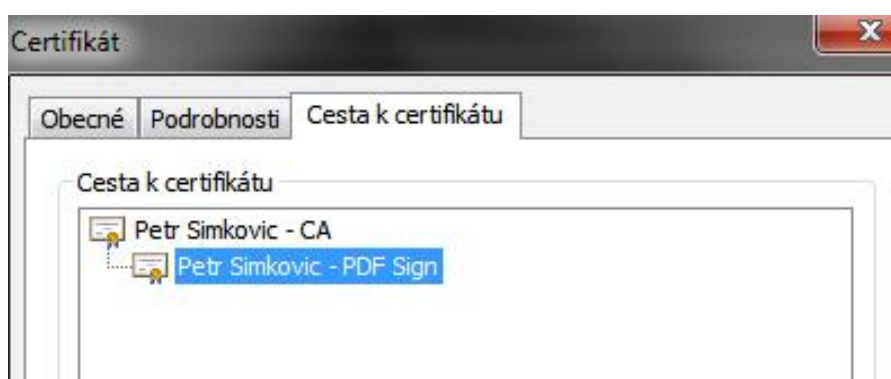
1. Vytvoření žádosti – tomu může, ale i nemusí předcházet generování párových dat. Generování může zajistit až samotná certifikační autorita např. po písemné žádosti.
2. Vydání - zajišťuje CA.
3. Platnost – ta začíná v době uvedené v položce „od“ (popsané v části výše Platnost) a končí buď vypršením platnosti certifikátu, nebo jeho odvoláním.
4. Vypršení – ukončení platnosti nastane po uplynutí doby „do“.

5. Odvolání – nastává v době před vypršením. To provádí sama certifikační autorita buď ze svého rozhodnutí (např. později zjistila, že uvedené údaje v certifikátu nejsou nadále platné), nebo na žádost držitele (např. uživatel ztratil soukromý klíč, který by tak mohl být zneužit).

1.3 Ověření certifikátu

Certifikát ověřujeme proto, abychom zjistili, zda veřejný klíč uvedený v certifikátu je platný a opravdu náleží držiteli uvedenému v předmětu certifikátu. Ověření certifikátu z hlediska času může být rozděleno na dva případy. První případ nastane, pokud chceme ověřit platnost certifikátu k nějakému okamžiku v minulosti. Tento problém nastane v případě ověřování pravosti archivovaných digitálně podepsaných dokumentů. Tímto se budeme zabývat až později v souvislosti s časovými razítky.

Druhým případem je ověřování platnosti certifikátu k aktuálnímu času. Nejdříve je nutné stanovení tzv. certifikační cesty. Jak už bylo zmíněno výše, certifikáty vydává certifikační autorita. Ta ale může vydat kromě uživatelských koncových certifikátů i certifikáty podřízeným autoritám. Ty dále samy mohou opět vydávat certifikáty jak uživatelům, tak i dalším podřízeným autoritám. Výsledkem tedy je stromová struktura certifikačních autorit. Ukázka cesty je znázorněna na obrázku níže.



Obr. 1.3.1. Certifikační cesta

Dalším problémem může být podvržený kořenový certifikát. Útočníkovi stačí vygenerovat libovolnou dvojici soukromého a veřejného klíče, který je podepsán tímto soukromým klíčem a který obsahuje stejné hodnoty položek jako kořenový certifikát pravé autority. Proto kořenové certifikáty musí být distribuovány důvěryhodnou cestou. Uvažujme následující případ. Bohumil pošle Alici podepsaný dokument. K podpisu přidá

i svůj certifikát a navíc i podvržený kořenový certifikát, kterým je podepsán jeho certifikát. Pokud Alice použije k ověření Bohumilova certifikátu zaslaný podvržený kořenový certifikát, bude samozřejmě ověření úspěšné. Proto je nutné používat kořenové certifikáty uložené ve vlastním důvěryhodném úložišti.

Jak bylo výše zmíněno, začínáme stanovením certifikační cesty. Na vrcholu stojí kořenový certifikát, kterému budeme důvěřovat v případě, že jej máme uložený v důvěryhodném úložišti a jeho platnost spadá do kontrolovaného času - v tomto případě jde aktuální čas. Dále postupujeme ke koncovému certifikátu a vždy ověříme zda:

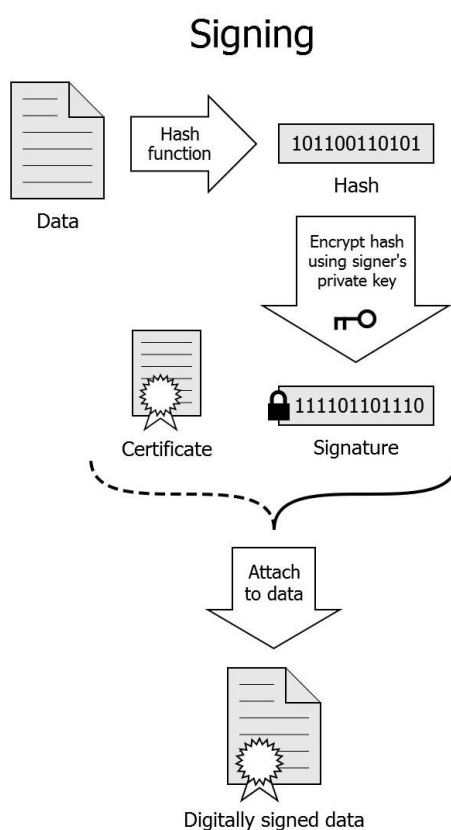
- Vydavatel certifikátu je shodný s předmětem předcházejícího certifikátu.
- Digitální podpis je v pořádku.
- Kontrolované datum spadá do platnosti certifikátu.
- Certifikát nebyl odvolán.
- Dále můžeme nepovinně ověřit rozšíření certifikátu. Např. omezení certifikátu, jeho účel apod.

Pokud není shledán žádný problém, ověření certifikátu může být považováno za úspěšné.

2 DIGITÁLNÍ PODPIS

Digitální podpis je mechanismus, kterým se zajišťuje důkaz nepopiratelnosti dat (pravosti dokumentů). Obecně se vytváří ve dvou krocích:

1. Spočte se otisk (*hash*) zprávy/dokumentu pomocí hashovací funkce.
2. Výsledný otisk se šifruje soukromým klíčem uživatele, který podpis vytváří. Tento šifrovaný otisk dokumentu se nazývá digitální podpis zprávy/dokumentu.



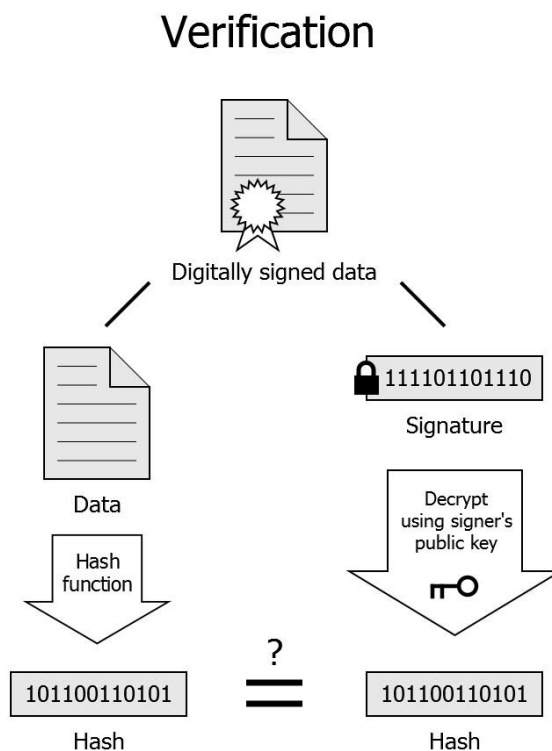
Obr. 1.3.1. Proces digitálního podpisu dokumentu⁴

Proces ověření podpisu se pak provádí ve třech krocích:

1. Příjemce samostatně spočte otisk z přijaté zprávy/dokumentu.
2. Následně dešifruje přijatý digitální podpis veřejným klíčem odesílatele.

⁴ http://en.wikipedia.org/wiki/File:Digital_Signature_diagram.svg

3. A nakonec porovná otisk z bodu 1 s dešifrovaným podpisem (otiskem) z bodu 2. Pokud jsou shodné, pak mohl digitální podpis vytvořit pouze ten, kdo vlastní soukromý klíč (odesílatel zprávy, autor dokumentu). Navíc tato skutečnost prokazuje, že zpráva/dokument nebyla během přenosu změněna.



If the hashes are equal, the signature is valid.

Obr. 1.3.2. Proces ověření digitálního podpisu⁵

Často ještě k tomuto procesu patří ověření certifikátu, kterým byl dokument podepsán. Ověření certifikátu se provádí postupem popsáním v části Ověření certifikátu. Digitální podpis provádí důkaz pravosti na základě vlastnictví soukromého klíče. [3] Je tedy nutné soukromé klíče držet na bezpečném místě a neposkytovat je cizím osobám.

⁵ http://en.wikipedia.org/wiki/File:Digital_Signature_diagram.svg

Pro úplnost bych rád zmínil, že proces šifrování zpráv funguje opačně oproti šifrování otisků při vytváření podpisu. První krok – šifrování se provádí veřejným klíčem a dešifrování pak soukromým klíčem. To znamená, že dešifrovat zprávu smí pouze majitel soukromého klíče (šifrovat a dešifrovat zprávu pouze jedním klíčem např. soukromým není možné). Z toho vyplývá, že operace šifrování a dešifrování jsou zaměnitelné. Vše je zajištěno díky kryptografickým algoritmům, např. již výše zmíněného RSA. Digitálního podpisu se využívá v široké míře. V praktické části práce se budu zabývat digitálním podpisem XML a PDF dokumentů, rovněž i standardem S/MIME.

2.1 XML podpis

XML podpis lze využít pro různé typy dokumentů, ale např. i pro binární data. Specifikem XML podpisu je možnost podepsání jen určité části dokumentu, tedy každá část smí mít svůj vlastní podpis [7]. Změna jen některé části dokumentu způsobí, že podpis celého dokumentu by se stal neplatným. To může mít uplatnění, pokud na jednom dokumentu pracuje více lidí a každý je za svoji část zodpovědný.

2.1.1 Struktura

Základním prvkem každého XML podpisu je element *Signature*. Ten obsahuje všechny ostatní elementy, které jsou nutné zejména pro ověření. Strukturu podpisu lze znázornit na následujícím schématu [11].

```
<Signature ID??>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI? >
      (<Transforms>)?
      <DigestMethod>
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo>)?
  (<Object ID??>)*
</Signature>
```

- *SignedInfo* – obsahuje seznam podepsaných elementů a použité algoritmy pro kanonizaci a podpis

- *CanonicalizationMethod* – informace o kanonizaci elementu *SignedInfo*, tj. způsob úpravy před výpočtem otisku (stejný dokument smí mít různou strukturu, např. bílé znaky)
- *SignatureMethod* – specifikuje použitý algoritmus pro výpočet otisku
- *Reference* – identifikuje podepsaná data
- *Transforms* – seznam transformací provedených na datech před výpočtem otisku
- *DigestMethod* – určuje použitý algoritmus pro výpočet otisku podepsaných dat
- *DigestValue* – obsahuje spočítanou hodnotu otisku
- *SignatureValue* – hodnota digitálního podpisu
- *KeyInfo* – informace o soukromém klíči, může obsahovat odpovídající veřejný klíč
- *Object* – doplňující informace, někdy obsahuje samotná podepsaná data

2.1.2 Typy

V praxi se objevují tři typy XML podpisů [7]:

1. Enveloped signature – podpis je vložen jako dítě kořenového elementu podepsaného elementu. Nejčastější forma XML podpisu.

```
<root>
  <signedElement id="1">
    ...
  </signedElement >
  <Signature>
    <SignedInfo>
      <Reference URI="#1"/>
    </SignedInfo>
  </Signature>
</root>
```

2. Enveloping signature – podepisovaná data jsou součástí elementu *Object*

```
<signedElement id="1">
  ...
</signedElement >
<Signature>
  <SignedInfo>
    <Reference URI="#1"/>
  </SignedInfo>
  <Object id="1">
    ...
  </Object>
</Signature>
```

3. Detached signature – podpis není součástí dokumentu nebo je sousedem podepisovaných dat

```
<signedElement id="1">
  ...
</signedElement >
<Signature>
  <SignedInfo>
    <Reference URI="#1"/>
    <Reference URI="http://www.ietf.org/rfc/rfc3075.txt"/>
  </SignedInfo>
</Signature>
```

2.1.3 Vytvoření podpisu

Tento proces se skládá z následujících kroků:

1. Vytvoří se seznam elementů, které mají být podepsány (element *Reference*).
2. Určí se hashovací funkce (element *DigestMethod*) a spočítají se hodnoty otisků všech objektů odkazovaných pomocí URI.
3. Všechny elementy *Reference* se sdruží spolu s vypočítanými otisky v elementu *SignedInfo*. Nepovinně lze aplikovat další transformace, které se uvedou pomocí elementu *Transform*. Následuje povinná kanonizace, element *CanonicalizationMethod*.
4. Vypočte se otisk elementu *SignedInfo* a podepíše se. Podpis bude pak součástí elementu *SignatureValue*.
5. Do elementu *KeyInfo* se pak může volitelně uložit hodnota veřejného klíče.
6. Elementy *SignedInfo*, *SignatureValue* a *KeyInfo* se pak uvedou do elementu *Signature*, který představuje výsledný podpis. Ukázkou podepsaného XML souboru lze vidět na obrázku níže.

```

<?xml version="1.0" encoding="UTF-8" ?>
- <note id="1">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body id="2">Don't forget me this weekend!</body>
- <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103">
- <ds:SignedInfo>
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
  <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
- <ds:Reference Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-ref0" URI="">
- <ds:Transforms>
  <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
</ds:Transforms>
  <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#sha256" />
  <ds:DigestValue>EDdT9uvOU1SgootsU7renv+JrzVdtb81CGO6wQHAnRk=</ds:DigestValue>
</ds:Reference>
+ <ds:Reference Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-ref1" URI="#2">
+ <ds:Reference Type="http://uri.etsi.org/01903#SignedProperties" URI="#xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-signedprops">
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-sigvalue">AnD3uXKMgAGtKkruEEs1EfnAfjHHLN0nWI9ka5L8RbiWI/DUjmadSHI5nsFjyOxyigbR1a7BrtCI1qkDppf8WdKM542+yuHYANgjG+FAwQsw+m1Y/zGiVFrhxYeAwOxKyA0wEHkCh3/zNeezCwuiJp tP7++D4/1sldkhh9TPI=</ds:SignatureValue>
+ <ds:KeyInfo>
+ <ds:Object>
</ds:KeyInfo>
</ds:Signature>
</note>

```

Obr. 2.1.1. XML podpis

2.1.4 Ověření podpisu

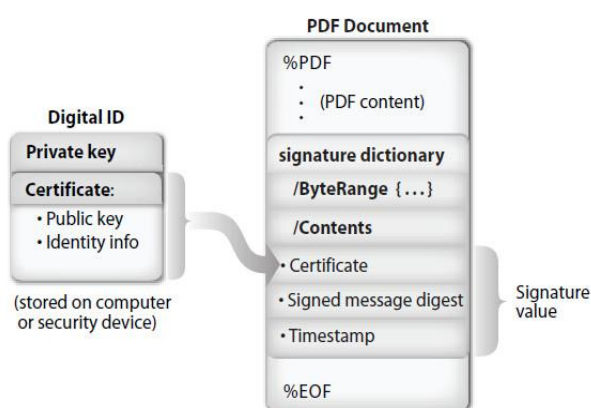
Podobně jako vytvoření se ověření podpisu skládá z několika kroků.

1. Element *SignedInfo* se upraví do kanonického tvaru. Použije se metoda specifikovaná v elementu *CanonicalizationMethod*.
2. Spočte se otisk elementu *SignedInfo* a oproti výsledné hodnotě se ověří podpis dokumentu. Hodnotu veřejného klíče je třeba znát z kontextu nebo jej získat z elementu *KeyInfo*.
3. Pokud je podpis v pořádku, ověří se hodnoty *DigestValue* jednotlivých elementů *Reference*. Objekty uvedené v attributech URI musí být pro ověření dostupné.
4. Pokud všechna ověření dopadnou v pořádku, lze podpis považovat za platný.

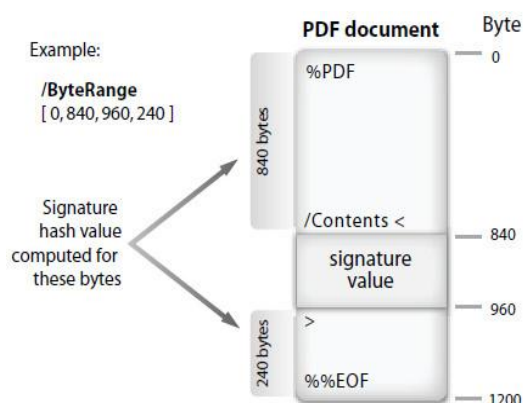
2.2 PDF podpis

Na rozdíl od XML formátu PDF v sobě zahrnuje podporu pro digitální podpisy, které jsou vždy umístěny v samotném dokumentu. Aplikacím, které PDF soubor např. zobrazují, toto umožňuje provádět jisté typy modifikací, které samotný digitální podpis nijak nezneplatní.

Pokud je PDF dokument podepsán, odpovídající veřejný klíč je vložen do PDF dokumentu. Hodnota podpisu v sobě může v sobě zahrnovat doplňkové informace. Jako například informace o zobrazení podpisu, časový údaj o vzniku podpisu a další. Struktura PDF dokumentu s podpisem je patrná z obrázku níže.



Obr. 2.2.1. Umístění podpisu v PDF dokumentu



Obr. 2.2.2. Rozsah bytů

Proces podpisu dokumentu je následující:

1. Nejdříve je samotný dokument načten do proudu bytů.
2. Následně je dokument zapsán na disk s vhodnou mezerou, do které se později uloží hodnota podpisu (na obrázku výše znázorněno jako *Signature value*). Zápis na disk je určen čtveřicí indexů/offsetů *ByteRange*.
3. Spočte se otisk dokumentu (např. pomocí algoritmu SHA-256), respektive jeho dvou částí zapsaných na disku, jak je znázorněno na obrázku výše.

4. Hodnota otisku je podepsána/zašifrována soukromým klíčem. Šifrovaná hodnota slouží k vytvoření PKCS#7⁶ objektu.
5. Tento objekt slouží jako podpis a je umístěn do mezery vytvořené v bodě 1. Zbylé bity jsou přepsány hodnotou nula.

Ověření podpisu je de facto opačnou sekvencí kroků:

1. Spočte se otisk dokumentu stejným algoritmem, jaký byl použit pro podpis. Do výpočtu otisku nevstupuje část, ve které je uložena hodnota podpisu.
2. Spočtený otisk je dešifrován odpovídajícím veřejným klíčem.
3. Dešifrovaný otisk je porovnán s otiskem uloženým v části PKCS#7 objektu.
4. Pokud jsou obě hodnoty otisku identické, podpis lze považovat za platný.

Podrobný a přesný popis vytváření a ověření PDF dokumentů je však ještě složitější než jak je popsáno výše. Formát PDF v sobě např. zahrnuje podporu násobných podpisů a zobrazení vybrané podepsané verze dokumentu. Dalším příkladem zmíněné větší složitosti může být parametr podpisu, kterým uživatel při podpisu může určit, zda v dokumentu smí být prováděny změny, které by neměly vliv na ověření. Např. vyplňování polí formuláře nebo vytváření anotací. Podrobnější popis je nad rámec této práce a lze jej získat na URL <http://www.adobe.com>, odkud pochází i dva obrázky uvedené výše.

2.3 S/MIME

S/MIME je protokol end-to-end zabezpečení e-mailových zpráv, které provádí odesílatel. Zabezpečená zpráva pak putuje sítí k adresátovi, aniž by během jejího transportu mohl její zabezpečení kdokoliv porušit. Zpráva je tak zabezpečena proti útokům na celé cestě mezi odesílatelem a adresátem.

Dnes je aktuální verzí protokolu verze 3.1. Ta v sobě zahrnuje normy RFC-3850⁷ a RFC-3851⁸. Od pořízení zprávy v lokálním prostředí odesílatele až po její odeslání protokolem prochází zpráva několika kroky [3]:

⁶ <http://www.rsa.com/rsalabs/node.asp?id=2129>

1. Vytvoření zprávy.
2. Prove se konverze (kanonizace) zprávy do jednoznačné reprezentace. Ta je nutná nejen pro případ digitálního podpisu, ale i pro ostatní způsoby zabezpečení. Např. v případě čistého textu se jedná o nahrazení konců řádků dvojicí znaků CR a LF.
3. Kódování do 7bit se provádí, zejména pokud bude zpráva přenášena elektronickou poštou. Např. se využívá kódování Base64⁹.
4. Zpráva je doplněna o MIME¹⁰ hlavičky, které například specifikují typ dat a způsob jejich kódování. Výsledkem MIME zpráva skládající se z hlaviček, prázdného řádku a těla zprávy. Kromě MIME hlaviček mohou být i doplněny i klasické RFC-(2)822 hlavičky (např. hlavička *Subject*, *To*, *From* nebo *Cc*).
5. Výsledná MIME zpráva vstupuje do CMS¹¹ stroje vždy jako CMS typ *id-data*.
6. Jelikož výsledek CMS stroje je binární a zpráva bude pravděpodobně přenášena elektronickou poštou, musí zpět do bodu kanonizace a konverze do 7bit.
7. Výsledek případné kanonizace a kódování se zpět obalí MIME hlavičkami, které sdělují, o jaká data se jedná (např. *Content-type: application/pkcs7-mime*). Nyní další postup závisí na tom, jestli se má zpráva dále zabezpečovat (např. zatím je jen podepsaná a ještě by se měla vložit do elektronické obálky). Pokud se bude dále zabezpečovat, znovu vytvořená MIME zpráva se odešle do CMS stroje. Pokud ne,

⁷ <http://tools.ietf.org/html/rfc3850>

⁸ <http://tools.ietf.org/html/rfc3851>

⁹ <http://www.ietf.org/rfc/rfc2045.txt>

¹⁰ <http://www.ietf.org/rfc/rfc2045.txt>

¹¹ <http://www.ietf.org/rfc/rfc3852.txt>

8. Pošle se dále na doplnění o RFC-(2)822 hlavičky.
9. Posledním krokem je pak odeslání zprávy např. protokolem SMTP¹², tj. vložení zprávy do SMTP obálky. Jestli je protokol nějak zabezpečen např. SSL, již neřešíme.

Příklad podepsané zprávy je zobrazen níže.

¹² <http://www.ietf.org/rfc/rfc5321.txt>

```

Message-ID: <3110987.4.1335736867673.JavaMail.simkovicp@simkovicp-PC>
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature";
micalg=shal;
    boundary="-----_Part_3_29431935.1335736867673"
From: petr.simkovic@gmail.com
To: petr.simkovic@gmail.com
Subject: Predmet

-----_Part_3_29431935.1335736867673
From: petr.simkovic@gmail.com
To: petr.simkovic@gmail.com
Subject: Predmet
Content-Type: multipart/mixed;
    boundary="-----_Part_0_25024808.1335736867573"

-----_Part_0_25024808.1335736867573
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: 7bit

Text zpravy
-----_Part_0_25024808.1335736867573--

-----_Part_3_29431935.1335736867673
Content-Type: application/pkcs7-signature; name=smime.p7s; smime-type=signed-
data
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIAGCSqGSIB3DQEHAQCAMIACAQExCzAJBgUrDgMCGGUAMIAAGCSqGSIB3DQEHAQAoIAWggLNMIC
NqADAgECAgEMMA0GCSqGSIb3DQEBBQUAMIGOMQswCQYDVQQGEwJDWjEOMAwGA1UECBMFQ1pFQ0gx
DTALBgNVBACtBFpMSU4xDDAKBgNVBAoTA1VUQjEMMAoGA1UECXMdRkFJMRswGQYDVQQDExJQZXRy
IFNpbWtvdmljIC0gQ0ExJzAlBgkqhkiG9w0BCQEWGHBldHIuc21ta292aWNAbG9naWNhLmNvbTAE
Fw0xMjAzMTkxNjQxMTVaFw0xMzAzMTkxNjQxMTVaMIGQMqswCQYDVQQGEwJDWjEOMAwGA1UECBMF
Q1pFQ0gxDTALBgNVBACtBFpMSU4xDDAKBgNVBAoTA1VUQjEMMAoGA1UECXMdRkFJMR4wHAYDVQQD
ExVQZXRyIFNpbWtvdmljIC0gU01JTUUXJjAkBgkqhkiG9w0BCQEWf3BldHIuc21ta292aWNAAZ21h
aWwuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDB3ExQMd89XfY8mn6kHT11m08RoL1t
wcA1EGFAyKbe0+VcR/J3m8Ys1I1FbHzd+HWqwhT2voTF4HCADX+Q/Q5D6q9Q6rSRF5QbI1e0/RT
WDMPJNYdHmSbH8vVbykfQZfxTS4P4MqiKJY/u4BsGdfzSsCFU0m/fQPe4WrnNTQzwIDAQBozcw
NTALBgNVHQ8EBAMCBAwEwYDVR01BAwwCgYIKwYBBQUHAWQwEQYJYIZIAYb4QgEBBAQDAgUGMA0G
CSqGSIB3DQEBBQUAA4GBAJIBENkQyhNWy5t/glCpixuOjvRwwXEF9GyIf6USjVTiqArMHS+VZda
zZa7ENWzLFRG7U1pymbiTJCvCXwbZmHLNwiBYHycb8FPU79nAlMkMBquYMenKKIjm4xhMf4gKJ
wG9D0edN0LQ/RyFOxKuRsVcCO70HtpXeCUoH5YC/AAaxggJ8MIICeAIBATCBLDCBjJELMAkGA1UE
BhMCQ1oxdDjAMBgNVBAGTBUNARUNIMQ0wCwYDVQQHEwRaTElOMQwwCgYDVQQKEwNVVEIxDDAKBgNV
BAStA0ZBSTEbMBkGA1UEAxMSUGV0ciBtaW1rb3ZpYyAtIENBMSwJQYJKoZIhvcNAQkBFhhwZXRy
LnNpbWtvdmljJGxvZ21jYS5jb20CAQwwCQYFKw4DAhoFAKCCAT0wGAYJKoZIhvcNAQkDMQsGCSqG
SIb3DQEHAQAcBgkqhkiG9w0BCQUxXcNMTIwNDI5MjIwMTA3WjAjBgkqhkiG9w0BCQQxXfGQUVMP6
5CysR8Rv2cfFalUyXB3b9hkwNAYJKoZIhvcNAQkPMSwJTAkBgkqhkiG9w0DBzAObggqhkig9w0D
AgICAIAwBwYFKw4DAgcwGACGcyqGSIB3DQEJEAILMYGXoIGUMIGOMQswCQYDVQQGEwJDWjEOMAwG
A1UECAwFQ1pFQ0gxDTALBgNVBACtBFpMSU4xDDAKBgNVBAoMA1VUQjEMMAoGA1UECwwDRkFJMRsw
GQYDVQQDDBJQZXRyIFNpbWtvdmljIC0gQ0ExJzAlBgkqhkiG9w0BCQEWGHBldHIuc21ta292aWNAb
G9naWNhLmNvbQIBDDANBgkqhkiG9w0BAQEFAASBgIrcCjYyGPSf6Pj7FnImg4A/ub4wecqar6LN
lx4yJn7BSScxQPCywwKFKfUfOmL4q9J0XcPftG8KyR6xc9GDFE8bHtJKAOUa7/dnXvCLjI5eLt3Z
962yWPS/eazsUbjX6mneQksFVPagkUYmYeV508uluWOCgqd+cvpdD/Em16XqAAAAA
-----_Part_3_29431935.1335736867673--

```

S/MIME spolu protokolem CMS tři následující způsoby zabezpečení zpráv:

1. Digitální podpis (CMS typ *SignedData*),
2. Elektronickou obálku (CMS typ *EnvelopedData*),
3. Komprimovanou zprávu (CMS typ *CompressedData* od verze S/MIME 1.3).

Dále nás bude zajímat jen typ *SignedData*. S/MIME klade důraz na využívání podepisovaných atributů v CMS zprávě *SignedData*. Pro přehled následuje tabulka podepisovaných atributů.

Podepisovaný atribut	Standard	Význam
<i>contentType</i>	CMS	Specifikuje typ obsahu.
<i>messageDigest</i>	CMS	Obsahuje kontrolní součet z podepisované zprávy.
<i>signingTime</i>	CMS	Čas podpisu zprávy (není garantován třetí stranou).
<i>sMIME Capabilities</i>	S/MIME	Obsahuje podepisovací a symetrické šifrovací algoritmy.
<i>sMIMEEncryptionKeyPreference</i>	S/MIME	Umožňuje odesílateli specifikovat certifikát s jeho preferovaným šifrovací klíčem (pro odpověď).
<i>signingCertificate</i>	ESS	Identifikace certifikátu pro verifikaci.

2.3-1 Podepisované atributy v S/MIME zprávě

Podepsání a ověření takovýchto zpráv se provádí obecným způsobem podepisování. Tj. vytvořením otisku a jeho následným zašifrováním soukromým klíčem. U S/MIME standardu bych ještě rád zmínil, že existuje možnost zašifrování zprávy pomocí veřejného klíče. Číst takovou zprávu může tedy jen držitel soukromého klíče. Nevýhodou tohoto standardu je fakt, že dosud neexistuje způsob vytvoření garance správnosti hodnoty *signingTime*.

3 DIGITÁLNÍ ČASOVÉ RAZÍTKO

Certifikát veřejného klíče je datová struktura, která spojuje identifikaci osoby s jeho veřejným klíčem. Tato vazba je stvrzena digitálním podpisem nezávislé třetí strany – certifikační autoritou. Časové razítko je svým způsobem obdobná datová struktura, která však svazuje dokument s určitým časem. Jednoduše řečeno obsahuje čas, otisk dokumentu, vydavatele razítka a pořadové číslo. To vše je rovněž stvrzeno nezávislou třetí stranou – autoritou pro vydávání časových razítek TSA (Time stamping authority).

Časové razítko tak slouží jako důkaz, že dokument existoval v daném čase. Samotný dokument je v razítku reprezentován pomocí tzv. *message imprint*, což je dvojice otisku dokumentu a algoritmu pro výpočet otisku. Na rozdíl od digitálního podpisu zde chybí totožnost žadatele. Důvod je prostý. TSA jeho totožnost nezkoumá a z hlediska platné legislativy, tak činit ani nesmí. Samotné razítko tedy není důkazem, že dokument v daném čase měla v držení určitá osoba. Co je však stejné, je to, že jakákoliv pozdější změna dokumentu znamená změnu jeho otisku a tudíž i zneplatnění razítka

Fyzicky razítko může existovat samostatně vně dokumentu, ale nejčastější varianta je použití rozšířeného digitálního podpisu. Samotný podpis v sobě žádný časový údaj nezahrnuje a tato skutečnost může způsobovat zejména po vypršení certifikátu jistý problém. Autor dokument popře, že dokument podepsal. Následně prohlásí, že útočník našel k jeho veřejnému klíči odpovídající soukromý klíč, kterým dokument tento útočník podepsal. Hledání soukromého klíče může trvat i několik desítek let. Přesto například v bankovním sektoru tato absence časového údaje, může způsobovat vážné komplikace. Pro zvolení varianty rozšířeného digitálního podpisu můžou hrát roli následující skutečnosti:

- Mnohdy je podstatnější, kdy byl dokument podepsán, než vytvořen (analogie s vytvářením a platností zákonů).
- Bez existence razítka nelze dokázat, zda byl dokument podepsán po vypršení certifikátu (technicky to není problém).
- Pokud dokument platí delší dobu, než je platnost certifikátu, je po jeho vypršení platnost dokumentu sporná. [3]

Pro vydávání časových razítek existuje samostatný protokol TSP definovaný v RFC-3161¹³. Ten popisuje žádost o časové razítko (sekvence dat nazvaná *TimeStampReq*) a odpověď na tuto žádost (sekvence nazvaná *TimeStampResp*). Bližší popis těchto sekvencí lze naléznout ve zmíněném RFC.

Jak bylo zmíněno výše, časové razítko je datová struktura digitálně podepsaná TSA, která razítko vydala. Proto se jeho ověřování provede pomocí certifikátu TSA. Ten však vydala a podepsala certifikační autorita. A tak se opět dostáváme k ověření certifikátu, které je popsáno sekci 1.3 Ověření certifikátu. Rozdíl spočívá jen v tom, že kontrolovaný čas není právě teď aktuální, ale použije se čas z razítka. Tím máme jistotu, že certifikát, kterým TSA razítko podepsala, byl v době podpisu platný. Pokud tomu tak je, zbývá ověřit digitální podpis razítka. To se provede spočítáním otisku původního dokumentu a jeho porovnáním s otiskem uvedeným v razítku (*message imprint*).

Tímto se dostáváme k tomu hlavnímu. A to je ověření pravosti podepsaných dat po vypršení certifikátu. Řešení je už zřejmé – používání rozšířeného digitálního podpisu o časové razítko. Pokud jej použijeme, jsme schopni ověřit platnost dokumentu i po vypršení platnosti certifikátu, kterým byl dokument podepsán. Takové ověření se provádí ve dvou krocích:

1. Ověření platnosti časového razítka (viz odstavec výše).
2. Ověření platnosti certifikátu použitého pro podpis dokumentu (viz sekce 1.3 Ověření certifikátu, rozdíl je opět v použití času z razítka).

Pokud obě ověření dopadnou kladně, lze takovýto podpis považovat za platný.

3.1 Specifika formátů

Zde bych zmínil, že podepisování PDF dokumentu s použitím s časového razítka spočívá jen v tom, že při podpisu se odpověď *TimeStampResp* od TSA zapouzdří do části, do které se ukládá digitální podpis PDF dokumentu, tj. část *SignatureValue*.

¹³ <http://www.ietf.org/rfc/rfc3161.txt>

U XML formátu se rozšířeným digitálním podpisem zabývá ETSI standard XML Advanced XML Electronic Signatures¹⁴ (XAdES). Ten definuje další elementy, které se přidávají do samotného XML podpisu, tj. do elementu *Signature*. Ty hlavní jsou:

- *SignedProperties* – element obsahující údaje získané z odpovědi od TSA,
- *SigningTime* – čas podpisu,
- *SigningCertificate* – certifikát, kterým TSA podepsala odpověď. Může obsahovat i jeho kořenový certifikát.

```
<ds:Signature Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103" xmlns:ds="http://www.w3.org/2000/
- <ds:SignedInfo>
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
  + <ds:Reference Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-ref0" URI="">
  + <ds:Reference Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-ref1" URI="#2">
  + <ds:Reference URI="#xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-signedprops" Type="http://
</ds:SignedInfo>
<ds:SignatureValue Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-sigvalue">
  AnD3uXKMgAGtKkruEEs1EfnAfjHHIEN0nWI9ka5L8RbiWI/DUjmadSHI5nsFjiyOxyigbR1a7Brt
  CI1qkDppf8WdKMs42+yuHYANgJG+FAwQsw+m1Y/zGiVFrhxYeAwOxKyA0wEHkCh3/zNeezCwuiJp tl
+ <ds:KeyInfo>
- <ds:Object>
  - <xades:QualifyingProperties Target="#xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103" xmlns:xades1
  xmlns:xades="http://uri.etsi.org/01903/v1.3.2#">
    - <xades:SignedProperties Id="xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103-signedprops">
      - <xades:SignedSignatureProperties>
        <xades:SigningTime>2012-03-19T14:53:49.982+01:00</xades:SigningTime>
        - <xades:SigningCertificate>
          + <xades:Cert>
          + <xades:Cert>
        </xades:SigningCertificate>
```

Obr. 3.1.1. Ukázka XML podpisu s razítkem

Z obrázku výše si můžeme všimnout, že XML podpis v sobě zahrnuje i otisk elementu, ve kterém je uloženo razítko. Pro připomenutí opakuji, že pro všechny elementy, které jsou odkazovány elementy *Reference*, se spočítají otisky. Ty se sduží v elementu *SignedInfo*, pro který se vypočte výsledný otisk, který se podepíše a jeho hodnota se uloží do elementu *SignatureValue*. Tímto je tedy zaručeno, že jakoukoliv změnu hodnot v části pro razítko lze zjistit ověřením podpisu XML dokumentu a zároveň lze měnit např. odsazení elementů v podpisu.

¹⁴ <http://www.w3.org/TR/XAdES/>

4 LEGISLATIVA

Dne 15. dubna 2010 nabyla účinnosti novela zákona o elektronickém podpisu (č. 101/2010 Sb.). Tento předpis v reakci na komitologické rozhodnutí 2009/767/ES přidává Ministerstvu vnitra povinnost vést a zveřejňovat seznam důvěryhodných certifikačních služeb a stanoví orgánům veřejné moci povinnost uznávat kvalifikované certifikáty vydané v ostatních členských státech EU.

4.1 Kvalifikovaný certifikát

Kvalifikovaný certifikát splňuje všechny aktuální požadavky dané legislativou, zejména zákonem o elektronickém podpisu (zákon č. 227/2000 Sb., o elektronickém podpisu a o změně některých dalších zákonů v aktuálním znění). Kvalifikovaný certifikát je standardizován také v rámci Evropské unie (Směrnice Evropského parlamentu a Rady 1999/93/ES).

Představuje nedílnou součást bezpečné komunikace občanů se státní správou a samosprávou, stejně jako komerčních aplikací. Tento produkt využijí všichni občané, firmy a úřady. Vysokou přidanou hodnotu přináší i specifickým skupinám, jako jsou lékaři, advokáti anebo firmám komunikujícím s Českou správou sociálního zabezpečení, s finančními úřady, se zdravotními pojišťovnami, s celním úřadem aj. Pro firmy i ostatní uživatele znamená používání kvalifikovaného certifikátu výraznou časovou úsporu a volnost (např. odpadá nutnost komunikovat s úřady pouze v úředních hodinách).

Z technologického pohledu je možné využít kvalifikovaný certifikát k vytváření elektronického podpisu, ověřování elektronických podpisů a zajištění nepopiratelnosti (vazba mezi dokumentem a osobou vytvářející elektronický podpis). Použití kvalifikovaného certifikátu pro další úkony, jako je například šifrování přenášených zpráv, je naopak legislativou omezeno.[8]

4.2 Kvalifikovaný systémový certifikát

Kvalifikované systémové certifikáty jsou certifikáty vydanými akreditovaným poskytovatelem certifikačních služeb ve smyslu zákona č. 227/2000 Sb., o elektronickém podpisu. Kvalifikované systémové certifikáty tvoří samostatnou kategorii certifikátů odlišnou od kvalifikovaných certifikátů (bez přívlastku systémový). Tyto certifikáty jsou

určeny k bezpečnému vytváření a ověřování elektronických značek. Označujícím subjektem (tvůrcem elektronické značky) může být nejen fyzická osoba, ale i právnická osoba nebo orgán státní správy či samosprávy. Elektronickou značku je možno vytvářet automatizovaně (např. v elektronických podatelkách pro potvrzování doručení podání).

Slouží zejména pro automatizované systémy, které využívají technologii založenou na principech elektronického podpisu bez součinnosti konkrétní fyzické osoby (např. elektronická fakturace, hromadné zaslání e-mailů, doručenek e-podatelný apod.).

Použití kvalifikovaného systémového certifikátu je v souladu s platnou legislativou vázáno na specializované hardwarové zařízení. Lze používat pro:

- vytváření elektronické značky
- ověřování elektronických značek
- bezpečné ověřování elektronických značek
- zajištění nepopiratelnosti (vazba mezi dokumentem a subjektem vytvářející elektronickou značku) [10]

4.3 Komerční certifikát

Komerční certifikát je vhodný pro obchodní použití mimo oblast komunikace s orgány veřejné moci, na které se vztahuje povinnost využívat certifikáty kvalifikované. Nejčastěji se používá v komunikaci mezi komerčními subjekty.

Využití má tak např. v podobě zaměstnaneckých certifikátů, jejichž účelem je zajištění interní bezpečné komunikace mezi zaměstnanci, mezi jednotlivými pobočkami a dislokovanými pracovišti, případně pro realizaci vzdáleného přístupu zaměstnanců k firemním datovým zdrojům. Jeho použití je možné také v uzavřených systémech, pokud je mezi účastníky bezpečné komunikace uzavřena smlouva, řešící podmínky této komunikace.

V současné době mají komerční certifikáty použití zejména pro šifrování a autentizaci. Jedná se především o neanonymní přístup na webové servery a předávání šifrovaných dat, jak e-mailovou poštou, tak prostřednictvím webových formulářů.

Komerční certifikáty mají širší oblast použití než kvalifikované certifikáty, které lze ze zákona použít jen pro první zmiňovaný účel. Na rozdíl od kvalifikovaných certifikátů

nejsou však komerční certifikáty automaticky uznávány. Obě komunikující strany se musí dohodnout (např. smluvně), že budou důvěřovat komerční certifikační autoritě. Komerční certifikáty mohou být vydávány osobám i technologickým komponentám (aplikace, zařízení, servery).

4.4 Elektronický podpis

V dřívější době, kdy existovala pouze papírová podoba dat, si lidé vystačili s obyčejným podpisem. Pokud člověk podepsal dokument, vyjadřoval tím vůli, že je s daným dokumentem obeznámen a s jeho zněním souhlasí.

Dne 29. 6. 2000 byl ve Sbírce zákonů, částce 68 zveřejněn zákon 227/2000 Sb.¹⁵, o elektronickém podpisu a o změně některých dalších zákonů (zákon o elektronickém podpisu). Smyslem zákona o elektronickém podpisu je umožnit použití digitálního podpisu v rámci elektronické komunikace jako ekvivalent podpisu vlastnoručního při běžné listinné formě komunikace. Zákon byl vytvořen na základě směrnice Evropské unie 1999/93/EC ze dne 13. 12. 1999 a upraven později zákony č. 226/2002, 517/2002 a 440/2004 Sb.

Jeho definice je následující. „Elektronickým podpisem se rozumí údaje v elektronické podobě, které jsou připojené k datové zprávě nebo jsou s ní logicky spojené a které slouží jako metoda k jednoznačnému ověření identity podepsané osoby ve vztahu k datové zprávě“. Pro většinu účelů je však toto nedostačující. Není např. zaručeno, že podepsaný dokument nebyl později změněn nebo že podepsaná osoba s dokumentem opravdu souhlasí.

4.5 Zaručený elektronický podpis

Kvůli obecné definici elektronického podpisu česká legislativa přichází s dalším pojmem, a to zaručený elektronický podpis. Ten musí splňovat následující požadavky:

- je jednoznačně spojen s podepisující osobou,
- umožňuje identifikaci podepisující osoby ve vztahu k datové zprávě,

¹⁵ <http://www.mvcr.cz/soubor/zakon-c-227-2000-sb-o-elektronickem-podpisu.aspx>

- byl vytvořen a připojen k datové zprávě pomocí prostředků, které podepisující osoba může udržet pod svou výhradní kontrolou,
- je k datové zprávě, ke které se vztahuje, připojen takovým způsobem, že je možno zjistit jakoukoliv následnou změnu dat.

4.6 Elektronická značka

Dalším důležitým pojmem je elektronická značka. Ta je definována jako: údaje v elektronické podobě, které jsou připojené k datové zprávě nebo jsou s ní logicky spojené a které splňují následující požadavky:

- jsou jednoznačně spojené s označující osobou a umožňují její identifikaci prostřednictvím kvalifikovaného systémového certifikátu,
- byly vytvořeny a připojeny k datové zprávě pomocí prostředků pro vytváření elektronických značek, které označující osoba může udržet pod svou výhradní kontrolou,
- jsou k datové zprávě, ke které se vztahují, připojeny takovým způsobem, že je možné zjistit jakoukoli následnou změnu dat.

4.7 Legislativa v praxi aneb co z toho vyplývá

Zaručený elektronický podpis dle zákona zaručuje, že podepisující osoba se před vlastním podepsáním s dokumentem seznámila. U elektronické značky se počítá s tím, že označující osoba dokument označila automatizovaně bez přímého ověření obsahu datové zprávy. Dále ze zákona vyplývá povinnost osob, aby s prostředky pro vytváření elektronických značek či podpisů nakládaly tak, aby nemohlo dojít k jejich neoprávněnému použití. Osoba je rovněž povinna podávat přesné a pravdivé informace poskytovateli certifikačních služeb ve vztahu ke kvalifikovanému certifikátu. Pro ověření zaručeného elektronického podpisu je vydáván certifikát veřejného klíče.

V dalším textu budu používat termín elektronický podpis ve smyslu zaručeného elektronického podpisu, pokud nebude řečeno jinak. Připojením elektronického podpisu k dokumentu jsou tedy z právního hlediska zajištěny následující vlastnosti:

- autenticita – autorem dokumentu je podepisující osoba,

- integrita – změna dokumentu po podepsání je zjistitelná,
- nepopiratelnost – autor dokumentu nemůže popřít, že dokument nepodepsal.

Termín elektronický podpis bývá často zaměňován za pojem digitální podpis. Jde o elektronický podpis, který je reprezentován pouze číslicemi a který používá technik asymetrické kryptografie. Ta využívá dva klíče: soukromý a veřejný. Soukromý klíč je určen pro podepisování dokumentů a osoba jej má ve výhradním vlastnictví. Veřejný klíč je určen pro ověření podpisu.

Z hlediska kryptografie lze digitální podpis chápat jako systém kryptografických funkcí zajišťující následující vlastnosti:

- Identifikace – podpis musí být vždy úzce svázán k podepisující entitě. Samotnou identifikaci zajišťuje připojený certifikát s uvedeným jedinečným jménem entity. V ČR se k tomuto používá identifikátor MPSV, který je součástí vydávaných certifikátů. Např. u certifikační autority PostSignum se používá rozšíření *SubjectAlternativeName*¹⁶.
- Autenticita – jde o proces, jehož výsledkem je ověření, že daná entita je skutečně tou, za kterou se vydává. Základem je ověření podpisu pomocí veřejného klíče, přičemž musíme věřit, že:
 - soukromý klíč výhradně vlastní podepisující entita,
 - veřejný klíč patří podepisující entitě,
 - celý mechanismus podpisu je bezpečný.
- Integrita – představuje proces ověření, že daná zpráva nebyla změněna,
- Nepopiratelnost – znamená, že podepisující entita nemůže popřít, že zprávu podepsala. Opírá se o předchozí předpoklad autenticity.

¹⁶ http://www.postsignum.cz/files/ostatni_dokumenty/QCA_ikmpsv.pdf

5 SOUČASNÉ EJB MODULY PRO PKI

Pro zajištění služeb PKI infrastruktury můžeme použít řadu nástrojů a softwaru. Mezi ty nejúspěšnější patří OpenCA Project - (<http://openca.org/>). Jde o úsilí komunity vývojářů vytvořit open source certifikační autoritu, která implementuje nejvíce používanější protokoly. Projekt používá mnoho open source projektů, např. OpenLDAP, OpenSSL nebo Apache Project. Vývoj v projektu je rozdělen do dílčích projektů:

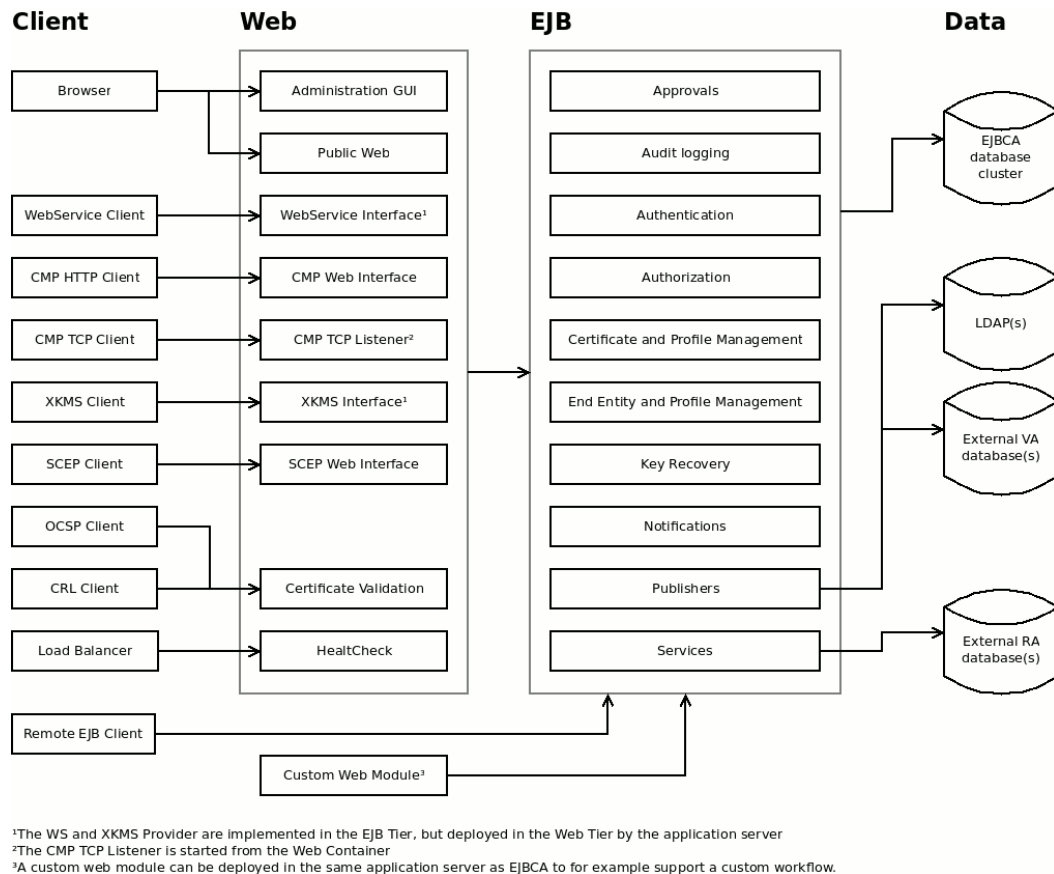
- OpenCA PKI – plně hodnotná PKI infrastruktura,
- LibPKI – knihovna pro vývoj aplikací používající PKI,
- OpenCA OCSPD – malý Online Certificate Status Protocol démon.

Projekt avšak používá programovací jazyk C/C++ a nenabízí žádné rozhraní pro digitální podpis.

Pokud bychom se zaměřili na programovací jazyk Java, konkrétně na EJB moduly zajišťující služby PKI, najdeme zde jeden projekt – EJBCA. Jde o volně šiřitelný produkt udržovaný a sponzorovaný švédskou společností PrimeKey Solutions AB šířený pod licencí LGPL (GNU Lesser General Public License). EJBCA podporuje řadu standardů z oblasti PKI (např. X.509, OCSP, CMP) a řadu asymetrických kryptografických algoritmů (např. RSA nebo DSA).

Všechny jeho webové moduly jsou zahrnuty do balíku Web Archives (WAR) a umístěny uvnitř Enterprise Archivu (EAR) spolu s EJB moduly zajišťujícími business logiku (např. mapování objektů do tabulek relační databáze). Na obrázku níže je znázorněna jeho architektura.

Na obrázku níže zobrazujícího architekturu modulu je patrné, že EJB modul používá databázi. V ní mohou být uloženy uživatelská data, ale i samotné certifikáty. Za zmínku stojí, že je podporována široká škála databází. Úspěšně byla otestována práce např. s databází PostgreSQL, MySQL, Oracle, Sybase, MS-SQL, DB2, Derby nebo Informix. Certifikáty mohou být uloženy i v LDAP. Dále modul může využívat externí databázi pro registrační či validační autoritu.

Obr. 4.7.1. Architektura EJBCA¹⁷

Samotný EJB modul zajišťuje jádro celé aplikace. Sem patří např. autentizace a autorizace pro webové rozhraní (aplikace např. umožňuje funkcionalitu vypršení platnosti hesla) nebo vytvoření a schválení žádosti o nový certifikát. Implementována je i podpora pro OSCP a CRL. Vytváření profilů a entit pro typy žádostí o nový certifikát. Nad tímto modulem pak stojí několik rozhraní popsaných níže:

- webové rozhraní – pro administraci aplikace nebo uživatelskou práci s aplikací,
- webové služby – určené zejména pro administraci,
- implementace/podpora protokolů – např. OSCP, CRL, SCEP,
- remote EJB rozhraní pro další EJB moduly.

¹⁷ <http://www.ejbca.org/architecture.html>

Z popisu je tedy patrné, že ani tato aplikace neposkytuje služby pro ověřování či vytváření digitálního podpisu. Na závěr bych uvedl a shrnul klady a zápory aplikace.

Klady:

- možnost použití v clusterovaném prostředí,
- volně šiřitelný,
- podpora řady standardů a protokolů,
- propracované administrátorské webové rozhraní,
- podpora notifikací např. pro expirované certifikáty,
- podpora mnoha databází,
- podpora více aplikačních serverů (JBoss, Glassfish, WebLogic).

Zápory:

- nutná instalace databáze nebo LDAP,
- nutná znalost aplikačních serverů pro běh aplikace,
- instalace a konfigurace není tak snadná, jak ji autoři popisují,
- neposkytuje služby pro ověření či vytvoření digitálního podpisu.

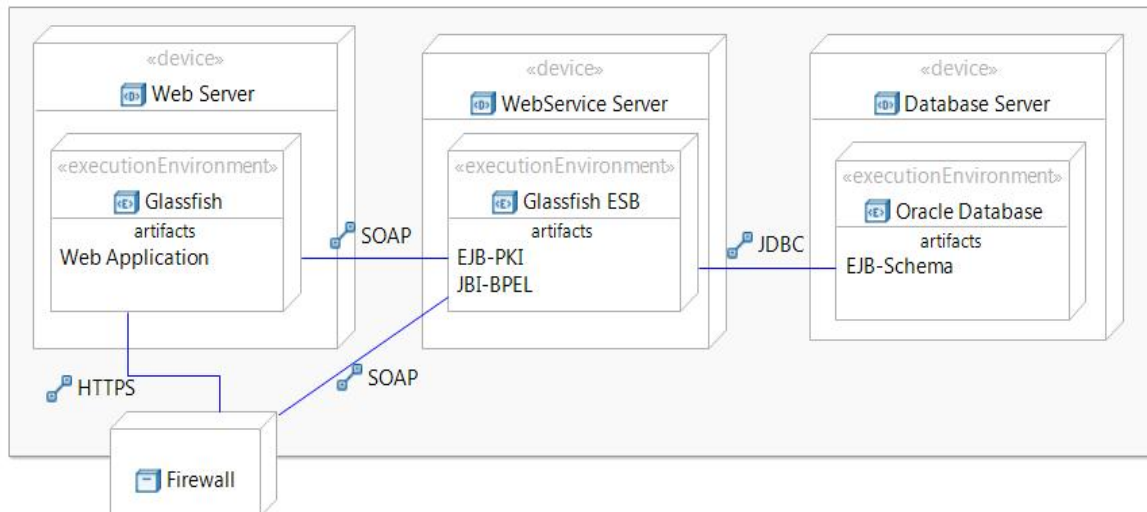
II. PRAKTICKÁ ČÁST

6 POHLED NA CELÝ SYSTÉM

I když zadání této práce přímo nevyžaduje vytvoření uceleného systému, rozhodl jsem se tak učinit. Důvody mého rozhodnutí jsou následující:

- Lepší pochopení souvislostí mezi jednotlivými částmi, které jsou implementovány.
- Vytvoření uceleného systému poskytuje lepší představu o uplatnění výsledků této práce v praxi.

Zároveň bych zde poznamenal, že veškerá implementace byla provedena pomocí vývojového prostředí Netbeans¹⁸ v programovacím jazyku Java. Výjimku tvoří jen databázové skripty. Volba tohoto prostředí byla učiněna zejména kvůli jeho dobré podpoře práce s WSDL a BPEL procesy. Na obrázku níže se už nachází zobrazení celého systému pomocí tzv. deployment diagramu. Jeho podoba nemusí nutně vypadat tak, jak je znázorněno (např. všechny části lze umístit do jednoho serveru). Budu však z tohoto vycházet.



Obr. 4.7.1. Deployment diagram

Z diagramu je patrné, že celý systém se skládá z několika provázaných částí. Začlenění systému „do světa“, jinými slovy jeho připojení k Internetu, je realizováno pomocí

¹⁸ <http://www.netbeans.org>

firewallu. Skrz něj prochází veškerá komunikace mezi systémem a vnějším světem. Jeho účel je jediný – zabezpečení. Firewall například blokuje požadavky přicházející na porty, které části systému používají pro svou interní komunikaci a které nechceme, aby byly pro vnější svět přístupné.

Jak již bylo řečeno, systém se skládá z několika následujících částí:

- PKI EJB modulu,
- Implementace business procesů pomocí jazyka BPEL,
- Databáze,
- Webové aplikace.

PKI modul slouží vytváření a ověřování digitálních podpisů, posílání a ověřování zabezpečených e-mailů. S ostatními částmi komunikuje prostřednictvím zabezpečených webových služeb. Jde o webovou aplikaci a BPEL business procesy, které tvoří přístupné rozhraní pomocí zabezpečených webových služeb. Business procesy pak pro svou práci využívají databázi. Vše je pro názornost zobrazeno v tabulce níže. V ní se vnější komunikaci pak rozumí komunikační protokol, který by daná část používala pro zpřístupnění vnějšímu světu.

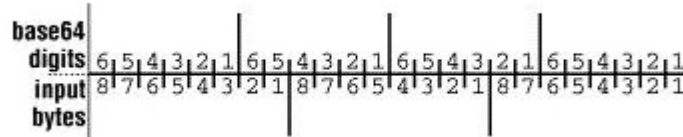
	Vnější komunikace	Zabezpečení	Přístupné světu
PKI EJB modul	SOAP	Basic autentizace	Ne
Business procesy	SOAP	Basic autentizace	Ano
Databáze	JDBC	Ne	Ne
Webová aplikace	HTTP, HTTPS	Ne	Ano

4.7-1 Části systému

6.1 Base64

Protože webové služby implementované v této práci budou využívat parametrů typu Base64 a tento pojem zde už několikrát zazněl, pro úplnost uvádím, o co vlastně jde. Base64 je formát pro reprezentování binárních dat pomocí ASCII znaků. Jeho podstata spočívá v tom, že každé tři vstupní bajty ($3 \times 8 = 24$ bitů) jsou převedeny na čtyři base64

číslice ($4 \times 6 = 24$ bitů). Přičemž každá taková base 64 číslice představuje index do tabulky, jejíž hodnoty jsou ASCII znaky. Počet možných číslic a tedy i znaků je 64 (0 až 63), tj. počet možných hodnot uložených v šesti bitech). Obrázek níže ukazuje převod bajtů do base64 číslic. Base64 je plně popsáno v RFC1521 v sekci 5.2¹⁹.



Obr. 6.1.1. Převod bajtů do base64 [5]

Vlastnosti:

- Výsledný řetězec se skládá z tisknutelných znaků ASCII.
- Rozlišuje se velikost písmen.
- Nemá kontrolní mechanismus.
- Kódování probíhá binárně (neřeší se znakové kódování apod.).
- Base64 je binárně bezpečné.
- Délka výsledného řetězce se obvykle navýší o 33 %.

¹⁹ <http://www.ietf.org/rfc/rfc1521.txt>

7 PKI EJB MODUL

EJB je řízená serverová komponenta určená pro modulární tvorbu rozsáhlých aplikací běžící v tzv. EJB kontejneru aplikačního serveru. [4] Ve specifikaci EJB²⁰ je detailněji popsáno, jakým způsobem má aplikační server spolu s EJB kontejnerem komunikovat a jak zajišťovat jeho funkcionalitu. V implementovaném systému je jako aplikační server použit GlassfishESB podporující EJB kontejnery a implementující specifikaci JBI spolu s BPEL procesy – ESB.

Implementovaný PKI EJB modul poskytuje základní práci využívající kryptografických služeb. Mezi ně patří:

- Vytváření (rozšířeného) digitálního podpisu pro XML a PDF dokumenty,
- Ověření (rozšířeného) digitálního podpisu pro XML a PDF dokumenty,
- Zasílání a ověřování e-mailů – S/MIME.

Přídavek rozšířený je v závorce uveden záměrně. Modul umí rovněž vytvořit a ověřit digitální podpis bez časového razítka. V případě XML formátu dokonce i ve více implementacích, vše je uvedeno v dalším textu. Tutu funkcionalitu pak svému okolí poskytuje pomocí zabezpečených webových služeb.

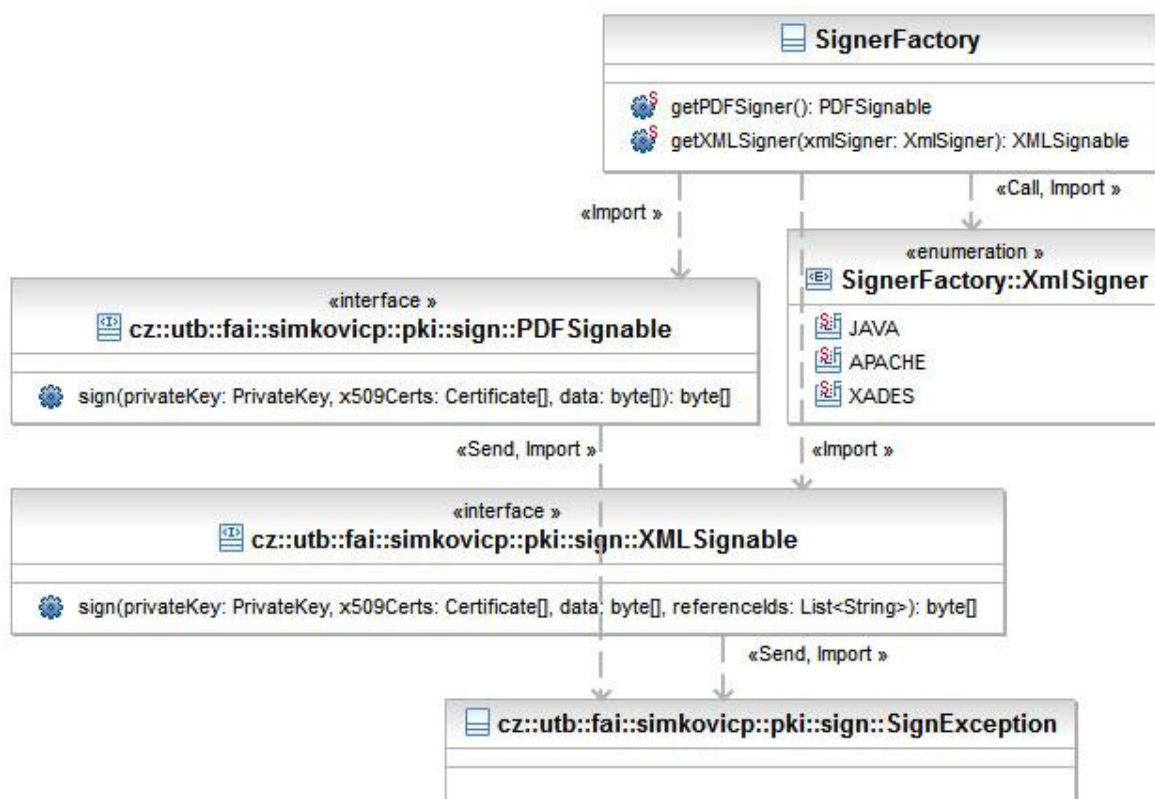
Než se dostanu k popisu samotné funkcionality modulu, rád bych se ještě na tomto místě zmínil o možnosti konfigurace. Téměř žádná aplikace se dnes ke svému běhu bez ní neobejde. Proto byla implementována samostatná knihovna, která je použita nejen v tomto modulu, ale např. i ve webové aplikaci. Její implementaci, stejně jako všechny implementace částí systému, včetně zdokumentovaného API PKI EJB modulu, můžeme nalézt v zabaleném archivu, který je přiložen k této práci. Knihovna implementuje načítání konfiguračních parametrů z tzv. *properties* souboru, jehož jméno je možné kvůli vícenásobnému použití definovat. Všechny konfigurační soubory se nachází v adresáři aplikačního serveru *domains/domain1/lib/classes*. Konfigurace je tak plně pod kontrolou správce systému a nezávisí na modulu či aplikaci, která by tento konfigurační soubor obsahovala. PKI EJB modul používá konfigurační soubor se jménem

²⁰ <http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html>

pki-utb.properties. Konfigurační soubor použitý v této práci se rovněž nachází v příloženém archivu.

7.1 Vytváření rozšířeného digitálního podpisu

O vytváření digitálních podpisů, časových razítek a jejich ověřování jsem se již zmínil v teoretické části práce. Nyní popíši, jak toto může být implementováno pomocí programovacího jazyka Java. A protože si myslím, že vše je lépe pochopitelné, když můžeme vidět daný problém znázorněný nejprve jako celek (a nejlépe v grafické podobě), uvádím zde zjednodušený tzv. class diagram týkající se této funkcionality.



Obr. 7.1.1. Class diagram podepisování

Hlavní třídou pro vytvoření podpisu je tovární třída `SignerFactory` poskytující implementace dvou rozhraní:

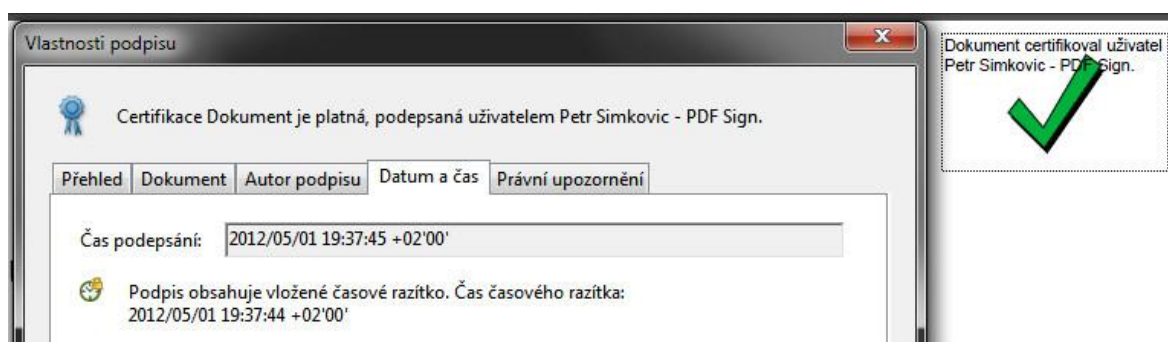
- `PDFSignable` – určeno pro podpis PDF dokumentů,
- `XMLSignable` – určeno pro podpis XML dokumentů.

Pro vývojáře, který by tento EJB modul používal jako API, neexistuje jiný způsob získání instancí těchto implementací rozhraní a musí tedy pracovat jen s těmito rozhraními

a tovární třídou. Změna jejich implementací nijak tedy nemůže ovlivnit API postavené nad tímto modulem. Obě rozhraní poskytují metody pro vytvoření digitálního podpisu s časovým razítkem. Pokud při podpisu vznikne chyba, vyhodí se výjimka `SignException`, v opačném případě se vrátí podepsaný dokument jako pole bytů.

7.1.1 PDF

Pro podpis PDF dokumentů je použita volně šiřitelná knihovna `iTextPdf`²¹. Podepisuje se celý dokument. Zda se má do podpisu vložit i časové razítko, je určeno konfigurací pomocí definování URL pro TSA a její případnou autentizaci, pokud ji autorita vyžaduje. Do PDF souboru, respektive do horního pravého rohu první stránky je vkládán rámeček zobrazující indikaci podpisu pomocí třídy `PdfSignatureAppearance`. Ta také poskytuje metody, jak do dokumentu vložit informaci, proč byl dokument podepsán, nebo kontakt na osobu, která může o podpisu podat nějaké informace. Vše se provádí ve spojitosti s danou konfigurací modulu. Výsledkem je podepsaný PDF dokument s časovým razítkem jako například na obrázku níže.



Obr. 7.1.2. Ukázka podepsaného PDF s časovým razítkem

7.1.2 XML

Pro podepisování XML dokumentů byly implementovány tři způsoby podpisů. Každý způsob se vyznačuje jinou použitou knihovnou. O tom, jaká implementace má být zvolena rozhoduje předávaný parametr již zmíněné tovární třídě. Tento parametr je vlastně enumerace `XmlSigner`, jak je vidět z class diagramu, a může nabývat následujících hodnot:

²¹ <http://itextpdf.sourceforge.net/>

- JAVA – podpis je vytvářen standardními prostředky jazyka JAVA. Především se používají třídy z package `javax.xml.crypto.dsig`. Neposkytuje však přidání časového razítka.
- APACHE – Zde se používá knihovna projektu Apache Santurio²², která poskytuje širší škálu možností vytvoření podpisu, než u předchozího způsobu. Avšak i tato knihovna dosud nenabízí možnost použití časového razítka.
- XADES – Poslední způsob je použití knihovny `xades4j`²³. Jak už název vypovídá, zde už podporu pro časové razítko můžeme nalézt.

Rozdíl mezi implementovaným PDF a XML podpisem tkví v tom, že u XML můžeme volitelně určit seznam ID, které představují podepisované části dokumentu. Pokud seznam není uveden nebo je prázdný, podepíše se celý dokument. To je důvod, proč pro podepisování vznikla dvě rozhraní. Jak může výsledný XML podpis vypadat lze vidět na obrázku uvedeného v sekci 2.1.1 nebo 3.1.

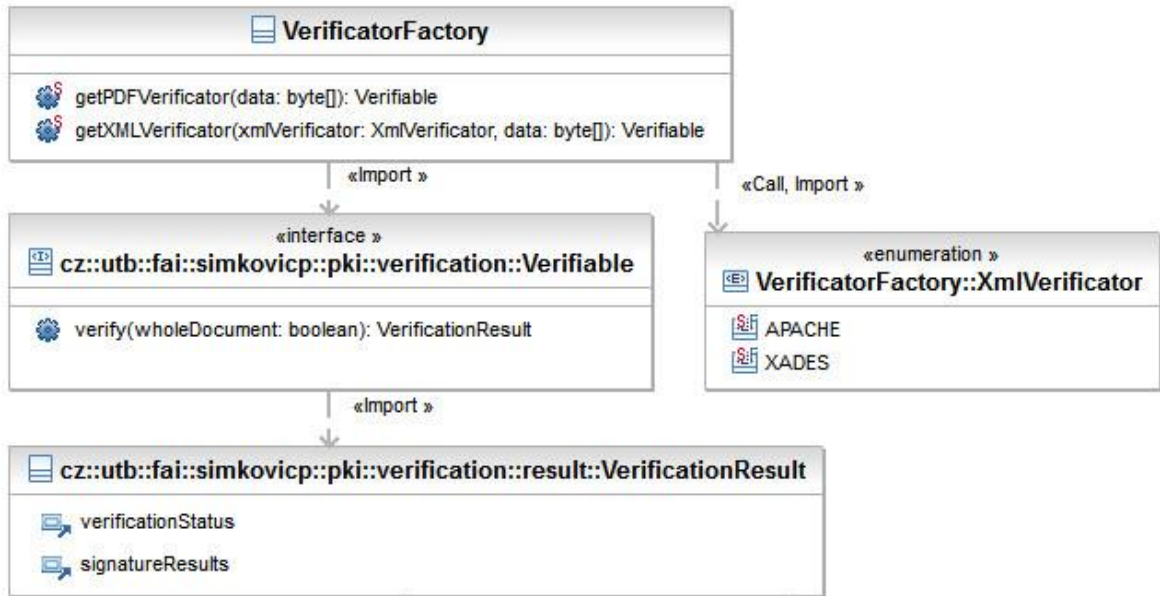
7.2 Ověřování rozšířeného digitálního podpisu

Stejně jako v předchozí části nejprve uvedu class diagram pro ověření PDF a XML dokumentů, viz. obrázek níže. Jak je zřejmé, situace je podobná. Funkcionalitu ověřování zajišťuje tovární třída `VerifierFactory`, která vrací instanci třídy implementující rozhraní `Verifiable`. To nabízí metodu pro ověření podpisu a jako výsledek, tedy návratová hodnota této metody, je třída `VerificationResult`. Ověření podpisu pro PDF i XML dokumenty je velice podobné a proto nebudu verifikaci rozvádět pro oba formáty zvlášť. V obou případech dochází ke kontrole digitálního podpisu a časového razítka, pokud je v podpisu obsaženo. Pro XML dokumenty jsou implementovány dva způsoby. Tentokrát je parametr předávaný tovární třídě typu `Xmlverificator` a opět jde o enumeraci s výčtem hodnot:

²² <http://santuario.apache.org/>

²³ <http://code.google.com/p/xades4j/>

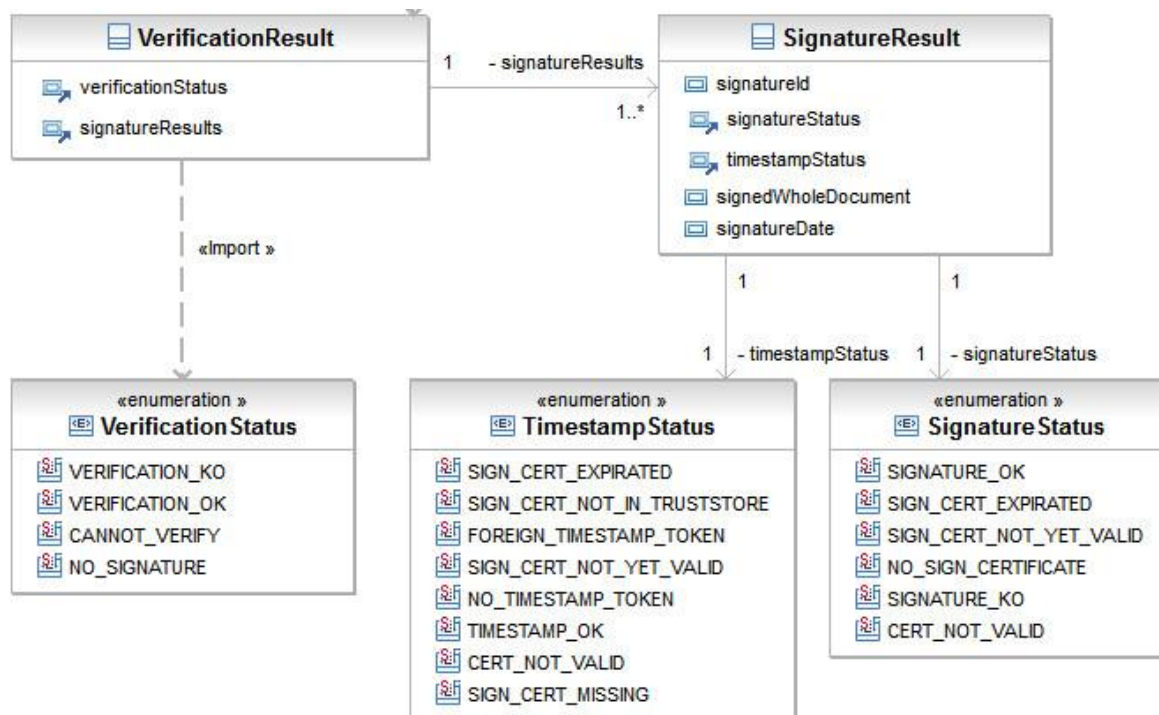
- APACHE – verifikace pomocí knihovny projektu Apache Santurio. Nepodporuje časová razítka.
- XADES – verifikace pomocí knihovny xades4j s podporou časových razítek.



Obr. 7.2.1. Class diagram ověřování

Za zmínku ovšem stojí popis třídy představující výsledek ověření podpisu `VerificationResult`, jejíž class diagram je zobrazen na obrázku na následující stránce. Tato třída obsahuje dvě proměnné:

- `verificationStatus` – hodnota enumerace `VerificationStatus` vyjadřující celkový výsledek ověření dokumentu. Může nabývat hodnot:
 - `VERIFICATION_OK` – dokument je platný,
 - `VERIFICATION_KO` – dokument není platný,
 - `NO_SIGNATURE` – dokument neobsahuje digitální podpis,
 - `CANNOT_VERIFY` – při verifikaci došlo k chybě.
- `signatureResults` – seznam výsledků ověření všech podpisů v dokumentu. Výsledek je instance třídy `SignatureResult`, která je popsána níže.



Obr. 7.2.2. Class diagram výsledku ověření

Jak již bylo zmíněno, v dokumentu se ověřují všechny nalezené digitální podpisy. Výsledkem ověření jednoho podpisu je třída `SignatureResult`, která obsahuje následující proměnné:

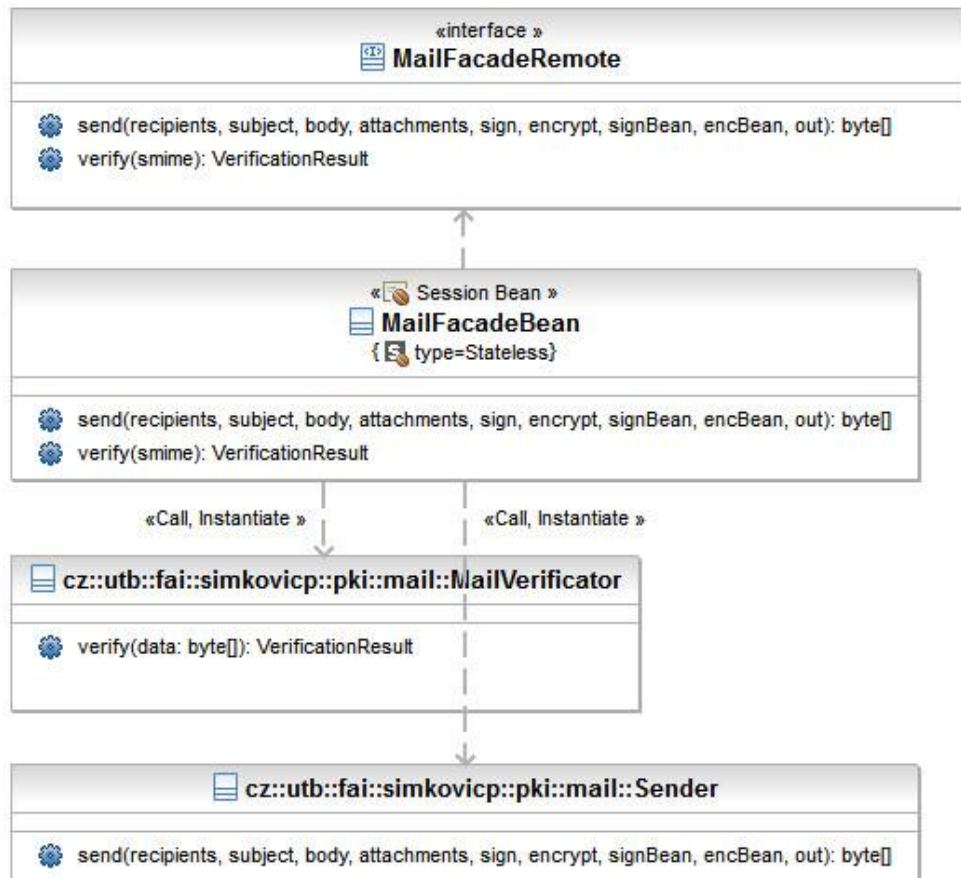
- `signatureId` – identifikace podpisu v rámci dokumentu,
- `signatureDate` – datum a čas z časového razítka,
- `signedWholeDocument` – `boolean` pro proměnná vyjadřující, zda jde o podpis celého dokumentu,
- `signatureStatus` – hodnota enumerace `SignatureStatus` vyjadřující výsledek ověření daného podpisu. Může nabývat hodnot:
 - `SIGNATURE_OK` – podpis je v pořádku,
 - `SIGNATURE_KO` – podpis není v pořádku,
 - `CERT_NOT_VALID` – certifikát není obecně platný (např. mu nedůvěřujeme),
 - `SIGN_CERT_EXPIRATED` – certifikát v době podpisu byl již vyexpirovaný,

- SIGN_CERT_NOT_YET_VALID – certifikát v době podpisu ještě nebyl platný,
- NO_SIGN_CERTIFICATE – podpisový certifikát nebyl nalezen.
- timestampStatus - hodnota enumerace `TimestampStatus` vyjadřující výsledek ověření časového razítka. Může nabývat hodnot:
 - TIMESTAMP_OK – razítko je platné,
 - NO_TIMESTAMP_TOKEN – podpis neobsahuje časové razítko,
 - FOREIGN_TIMESTAMP_TOKEN – razítko není určené pro tento dokument (význam u PDF),
 - SIGN_CERT_MISSING – podpisový certifikát nebyl nalezen,
 - SIGN_CERT_NOT_IN_TRUSTSTORE – kořenový certifikát není v úložišti důvěryhodných autorit,
 - SIGN_CERT_EXPIRATED - certifikát v době podpisu byl již vyexpirovaný,
 - SIGN_CERT_NOT_YET_VALID - certifikát v době podpisu ještě nebyl platný,
 - CERT_NOT_VALID – certifikát není platný.

7.3 Zaslání a ověřování e-mailů

Součástí PKI EJB modulu je i možnost zaslání a ověřování e-mailů. Při implementaci jsem ovšem narazil na nekompatibilitu knihoven pracující s Bouncy Castle²⁴ knihovnamí nabízející API pro kryptografické operace. To jsem vyřešil vytvořením samostatného EJB modulu nabízející zaslání a ověření e-mailu pomocí Stateless Session Bean implementující `javax.ejb.Remote` rozhraní, jak je znázorněno na obrázku níže. Díky tomuto rozhraní je možné tento bean injektovat do PKI EJB modulu.

²⁴ <http://www.bouncycastle.org/java.html>



Obr. 7.3.1. Class diagram modulu pro zasílání mailů

Protože zvolené vývojové prostředí Netbeans projekty implicitně sestavuje pomocí nástroje Ant²⁵, jsem sestavování doplnil o cíl *remote-jar*.

```

<target name="remote-jar">
  <mkdir dir="dist" />
  <jar compress="false" destfile="dist/pki-mail-remote.jar">
    <fileset dir="build/jar">
      <include name="**/*FacadeRemote*" />
      <include name="**/*Attachment*" />
      <include name="**/*EncryptionBean*" />
      <include name="**/*MailException*" />
      <include name="**/*Recipient*" />
    </fileset>
  </jar>
</target>
  
```

To zajistí vytvoření „remote knihovny“, která obsahuje potřebné třídy pro použití tohoto EJB modulu v PKI EJB modulu.

²⁵ <http://ant.apache.org/>

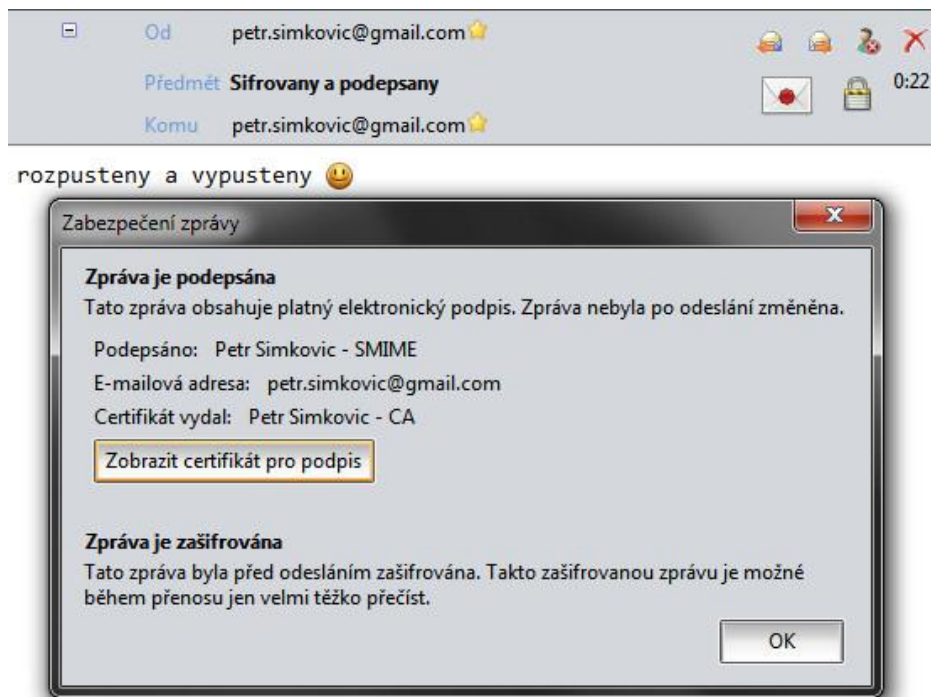
Posílání mailů je implementováno pomocí standardních tříd balíčku `javax.mail`. E-mail je možné zabezpečit dvěma způsoby:

- Podepsáním – provádí se soukromým klíčem,
- Zašifrováním – pomocí veřejného klíče.

Pro výsledek ověření e-mailu je použita stejná třída jako pro ověřování v PKI EJB, tj. `VerificationResult`. Toto zajišťuje podobný mechanismu, který je zmíněn u sestavování modulu s tím rozdílem, že při sestavování PKI EJB modulu se vytváří „common PKI knihovna“, která tuto třídu obsahuje a může tak být použita i zde. Vzhledem k tomu, že u e-mailů má cenu jen ověření integrity, používá se pouze proměnná `verificationStatus` s možnými hodnotami `VERIFICATION_OK` a `VERIFICATION_KO`. Na obrázcích níže je vidět zacházení mailových klientů s S/MIME zprávami. První obrázek představuje zobrazení podepsané a zašifrované zprávy v Thunderbird bez nainportování soukromého klíče. Druhý obrázek zobrazuje tu samou zprávu s nainportovaným soukromým klíčem a příslušným kořenovým certifikátem.



Obr. 7.3.2. Zobrazení S/MIME zprávy bez soukromého klíče



Obr. 7.3.3. Zobrazení S/MIME zprávy se soukromým klíčem

7.4 Práce s certifikáty

Již několikrát zaznělo, že podepisování a ověřování digitálních podpisů se provádí pomocí klíčů obsažených v certifikátech. Proto PKI EJB modul musí s těmito certifikáty umět pracovat. Vše zajišťuje třída `x509Manager`, která je implementována pomocí návrhového vzoru Singleton. Díky tomuto používá modul pouze jedinou instanci této třídy. Důvod použití je prostý. Samotné certifikáty jsou jen načítány, nijak se nemění, a proto je vhodné zajistit, aby práce s nimi byla co nejrychlejší. Tedy načíst je pouze jednou.

Jednotlivé certifikáty se pak postupně načítají při vykonávání `init` metody, která je volána z konstruktoru této třídy. Jaké certifikáty se mají načíst je uvedeno v konfiguračním souboru, který je opět přiložen k práci v zabaleném archivu. Jméno property indikující, že jde o PKCS12 certifikát, začíná prefixem `pki.p12` a končí identifikátorem certifikátu (volí jej libovolně správce systému/aplikace), který tímto zajistí rozlišení mezi jednotlivými certifikáty. Každý certifikát je pak reprezentován instancí třídy `PKCS12Device`, která si v sobě uchovává následující proměnné pro:

- privátní klíč
- veřejný klíč

- certifikační řetězec – představuje certifikační cestu uvedenou v kapitole o ověřování certifikátů.

Proměnné pak nabízí prostřednictvím svých metod. Následuje ukázka definice certifikátů v konfiguračním souboru. V ní si lze povšimnout, že existuje možnost indexování (obecně Java podporuje vícenásobné uložení certifikátu v jednom souboru), dále že se definuje jméno (pro rozeznání certifikátu), cesta k certifikátu, heslo pro otevření souboru, heslo pro získání privátního klíče a alias (odpovídá položce *common name* z předmětu certifikátu).

```
pki.p12.name.smime=SMIME
pki.p12.path.smime=/opt/certs/Petr_Simkovic_-_SMIME.p12
pki.p12.passwd.smime=...
pki.p12.alias.smime=Petr Simkovic - SMIME
pki.p12.apasswd.smime=...

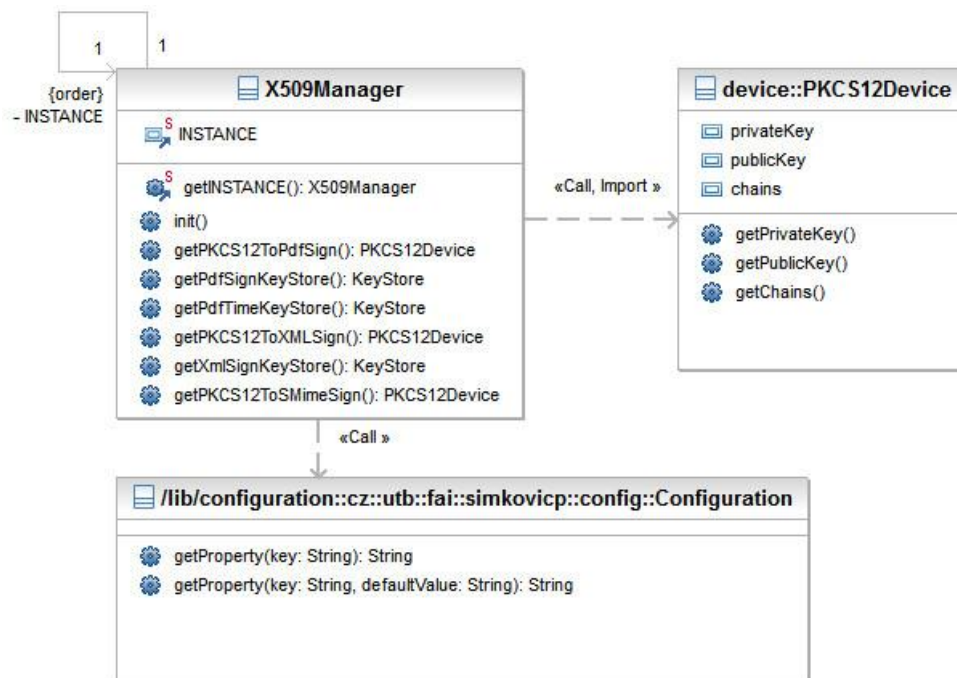
pki.p12.name.ps_sign=PDF_SIGN
pki.p12.path.ps_sign=/opt/certs/Petr_Simkovic_-_PDF_Sign.p12
pki.p12.passwd.ps_sign=...
pki.p12.alias.ps_sign.1=Petr Simkovic - PDF Sign 1
pki.p12.apasswd.ps_sign.1=...
pki.p12.alias.ps_sign.2=Petr Simkovic - PDF Sign 2
pki.p12.apasswd.ps_sign.2=...
```

Během inicializace třídy `X509Manager` se ovšem nenačítají jen certifikáty, ale i JKS úložiště certifikátů pro ověření digitálního podpisu a časového razítka. Ty obsahují seznam důvěryhodných autorit. Zde je konfigurace výrazně jednodušší a je zřejmá z konfiguračního souboru. Potřebné je znát jen cestu k úložišti a případné heslo k jeho otevření. Inicializovaná úložiště pak instance poskytuje pomocí svých metod. API práce s certifikáty je nastíněno na obrázku níže.

Poslední věcí, kterou je potřebné znát, představuje propojení mezi načtenými certifikáty a jejich použitím v modulu. To se provádí pomocí šesti řádků v konfiguraci:

```
pki.xml.signer.name=PDF_SIGN
pki.xml.signer.alias=Peťr Simkovic - PDF Sign 1
pki.pdf.signer.name=XML_SIGN
pki.pdf.signer.alias=Peťr Simkovic - XML Sign
pki.smime.signer.name=SMIME
pki.smime.signer.alias=Peťr Simkovic - SMIME
```

Z ukázky je patrné, že se propojí zvolené jméno certifikátu a alias (*common name*). Soukromé klíče těchto certifikátů se pak použijí pro vytvoření XML, PDF a S/MIME podpisu. V případě šifrování mailu jde o veřejný klíč.



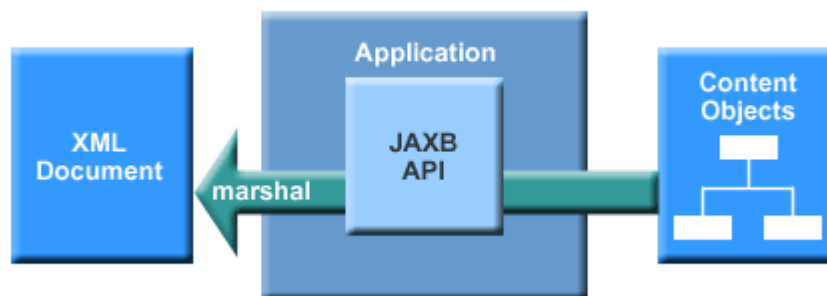
Obr.7.4.1. Class diagram práce s certifikáty

7.5 Webové služby

Služby jsou realizovány pomocí JAX-WS²⁶. Jde o technologii pro realizaci XML webových služeb a jejich klientů v Java EE. Lze tak tedy vytvořit jak klienta, tak i poskytovatele webové služby. Pro svou komunikaci používá klasické SOAP zprávy, které jsou posílány pomocí protokolu HTTP(S). Samotné webové služby jsou definovány jako třídy a jejich metody představují operace dané služby. Vše je realizováno pomocí anotací, které slouží k automatickému převodu do popisu WSDL. Implementovaný klient takovéto služby je nezávislý na platformě, tzn. pro službu implementovanou pomocí JAX-WS lze realizovat klienta např. implementovaného v C++. Funguje to i opačně. Pro službu implementovanou např. v C++, lze vytvořit klienta pomocí JAX-WS. Prostředí Netbeans k tomuto nabízí grafické rozhraní, které umožňuje klienta vytvořit pouhým vybráním souboru s WSDL definicí a naopak z třídy definující službu vygenerovat odpovídající WSDL definici a XSD schéma. JAX-WS ke své práci využívá další technologii – JAXB.

²⁶ http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/

Jde o technologii, která umožňuje mapovat XML soubory do java tříd. Díky ní lze podle XSD schématu vygenerovat odpovídající třídu. To se hodí právě u SOAP zpráv. Na vstupu operace služby vlastně přichází XML, které se pomocí operace *unmarshalling* namapuje do instance dané třídy. Vzniklá instance pak představuje přijatou zprávu. Při posílání zprávy se instance opačně musí převést do XML, to zajišťuje duální operace *marshal*, znázorněno na obrázku níže.



Obr. 7.5.1. Operace marshal v JAXB²⁷

7.5.1 Implementace

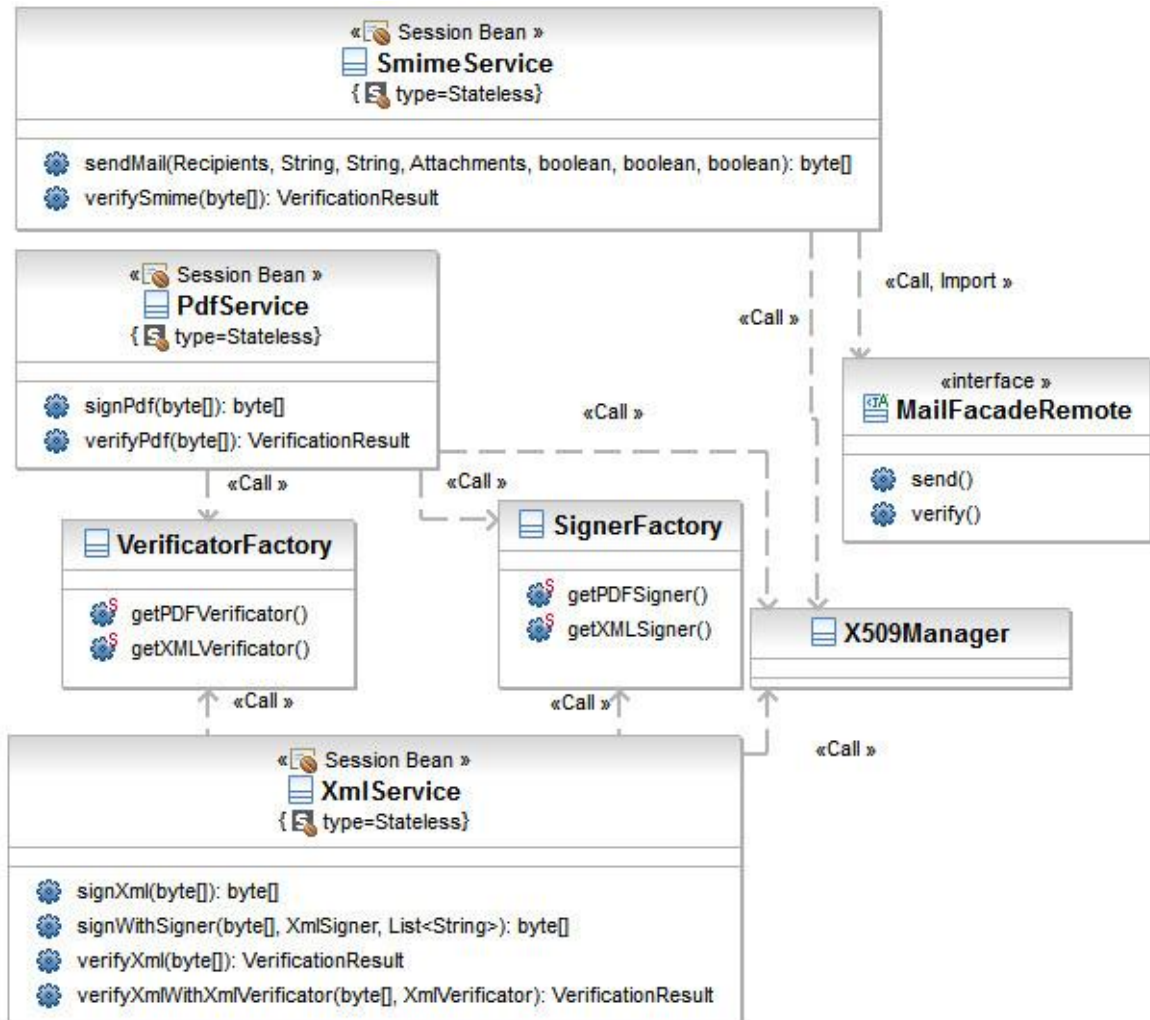
Služby zprostředkovávají funkcionalitu PKI EJB modulu jeho okolí. K tomu využívají metody a třídy popsané v předchozích částech. Jak služby zapadají do modulu je znázorněno na níže uvedeném obrázku níže. Kvůli omezení velikosti není diagram úplný, chybí na něm např. názvy balíčků a názvy parametrů metod. Navzdory tomu je možné si všimnout, že modul nabízí tři služby, které jsou členěny dle formátu dokumentů na práci s:

- PDF dokumenty,
- XML dokumenty,
- S/MIME.

Popisovat detailněji všechny služby nemá smysl, k tomu slouží WSDL defice. Obecně fungují všechny na stejném principu. Popis provedu na dvou příkladech, první bude vytvoření PDF podpisu, druhý jeho ověření. Malá odlišnost je jen u služby pro operace s S/MIME. Protože tyto operace jsou implementovány v jiném EJB modulu, který pro tyto

²⁷ <http://www.oracle.com/technetwork/articles/javase/jaxb-141136.html>

účely implementuje *Stateless Session Bean* implementující *Remote* rozhraní, musí být do třídy `SmimeService` injektován pomocí `@EJB` anotace proxy objekt zastupující instanci tohoto *Remote* rozhraní.



Obr. 7.5.2. Class diagram webových služeb PKI EJB modulu

Implementace operace služby pro vytvoření PDF podpisu vypadá následovně:

```

@WebMethod(operationName = "signPdf")
@WebResult(name="signedPdf")
public byte[] signPdf(@WebParam(name = "pdf") byte[] pdf)
    throws SignException {
    LOGGER.debug("signPdf() - " + pdf);
    LOGGER.debug("signPdf() - principal:" +
        wsContext.getUserPrincipal());
    PDFSignable signable = SignerFactory.getPDFSigner();
    PKCS12Device device = X509Manager.
        getInstance().getPKCS12ToPdfSign();
    return signable.
        sign(device.getPrivateKey(), device.getChains(), pdf);
}
  
```

Na prvním řádku pomocí JAX-WS anotace nadefinujeme, že operace služby se bude jmenovat *signPdf*. Druhý řádek říká, že výsledek operace bude nést jméno *signedPdf* (představuje to vlastně název XML elementu v SOAP odpovědi). Na dalším řádku deklarujeme java metodu s jejími parametry a návratovou hodnotou, v případě chyby se vyhodí výjimka *SignException* (v SOAP se vrátí tradiční *Fault*). Definován je rovněž název vstupního elementu. Jeho typ je pole bajtů. Protože v XML by se těžko tento typ přímo zapsal do nějakého elementu, využívá SOAP kódování Base64, které nám jej převede do řetězce tisknutelných znaků. Konverze se děje automaticky díky JAX-WS. Následující řádky provedou logování vstupu a jméno uživatele, pokud je služba zabezpečena autentizací. Následně se získá instance rozhraní pro podpis PDF dokumentu, získá se potřebný certifikát pro podepsání, které se následně provede. Výsledkem je pole bajtů reprezentující podepsaný PDF dokument. Logování výjimky zde je zbytečné, protože to se už provádí v třídě zajišťující podpis.

Ověření je o něco jednodušší. Není třeba zde získat žádný podpisový certifikát.

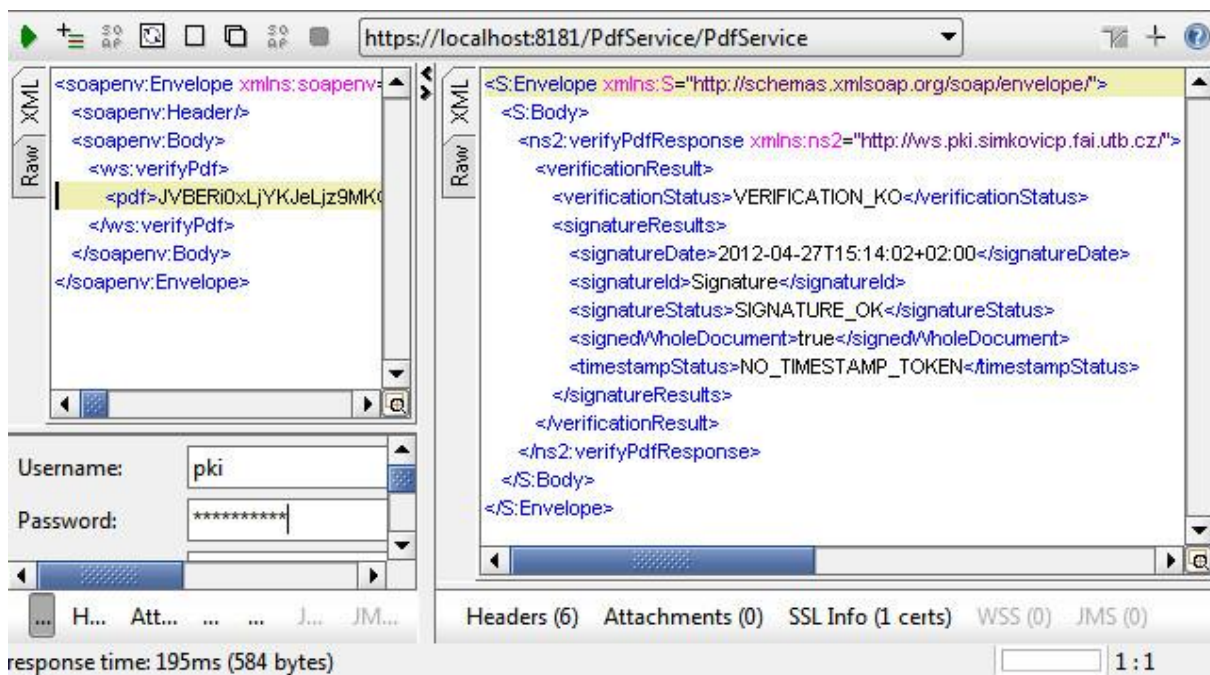
```
@WebMethod(operationName = "verifyPdf")
@WebResult(name="verificationResult")
public VerificationResult verifyPdf(
    @WebParam(name = "pdf") byte[] pdf)
    throws VerificationException {
    LOGGER.debug("verifyPdf() - " + pdf);
    LOGGER.debug("verifyPdf() - principal:" +
        wsContext.getUserPrincipal());
    Verifiable verifiable =
        VerificatorFactory.getPDFVerificator(pdf);
    VerificationResult result = verifiable.verify(true);
    LOGGER.debug("verify - result: " + result);
    if (result.getVerificationException() != null) {
        throw result.getVerificationException();
    }
    return result;
}
```

Úryvek kódu je podobný tomu předcházejícímu. I zde se definuje jméno operace, výsledku a předávaných hodnot včetně jejich typů. Po zalogování vstupu se získá instance třídy pro ověření pomocí tovární metody, ověří se PDF dokument a vrátí se výsledek ověření. Na obrázku, který se nachází na další straně, je ukázáno, jak zavolání služby může vypadat včetně výsledku mapování objektu do XML pomocí JAX-WS.

Prostředí Netbeans nabízí automatické vygenerování WSDL definice a XSD schématu každé služby, čehož bude využito níže pro tvorbu BPEL procesů. Služby jsou dostupné pomocí URL na portu, který je definovaný na straně pro aplikačního serveru pro HTTP(S) komunikaci. Pokud za URL doplníme řetězec „?wsdl“, dostaneme pak WSDL definici

služby, ve které lze nalézt odkaz na XSD schema. Přehled služeb s jejich URL je následující:

- `http[s]://host:port/XmlService/XmlService,`
- `http[s]://host:port/PdfService/PdfService,`
- `http[s]://host:port/SmimeService/SmimeService.`



Obr. 7.5.3. Ukázka ověření PDF pomocí webové služby PKI EJB

7.5.2 Zabezpečení

Kvůli větší bezpečnosti je u volání služeb PKI EJB nutná autentizace. Patrné to je i na obrázku výše. Zabezpečení služeb je definováno v tzv. deployment descriptoru `sun-ejb-jar.xml`²⁸. Jeho obsah je následující:

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>XmlService</ejb-name>
      <webservice-endpoint>
        <port-component-name>XmlService</port-component-name>
        <login-config>
```

²⁸ <http://docs.oracle.com/cd/E19316-01/820-4337/6nfqe0rl5/index.html>

```

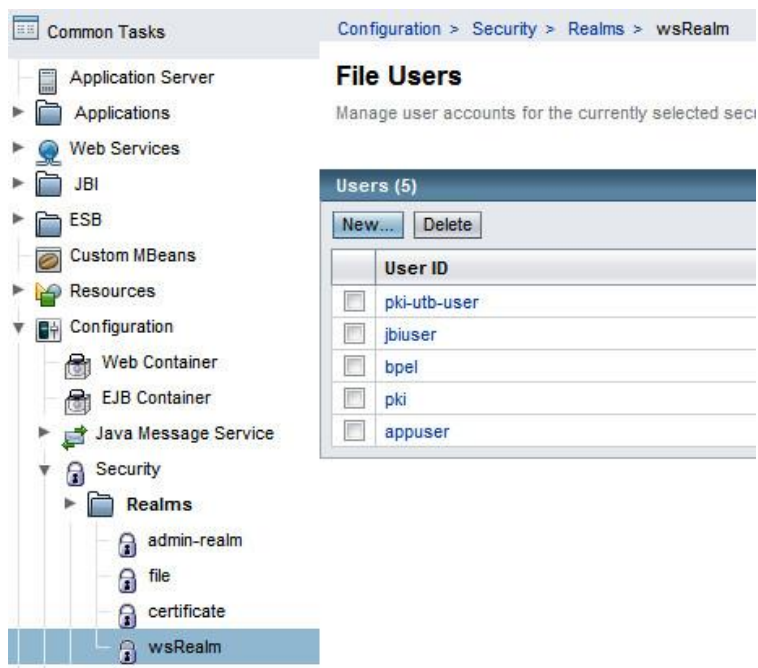
        <auth-method>BASIC</auth-method>
        <realm>wsRealm</realm>
    </login-config>
</webservice-endpoint>
</ejb>
...

```

Z úryvku lze rozpoznat, že autentizace se provádí pro každou službu zvlášť uvedením

- jména EJB - název třídy, jde vlastně o implementaci *Stateless Session Bean*.,
- názvu portu - pojem z WSDL pro rozlišení, kdyby existovalo více EJB tříd se stejným jménem; port zde představuje množinu všech operací dané služby,
- typ autentizace – použita BASIC autentizace (uvedení jména a hesla) a tzv. security realmu. Ve stručnosti jde o kolekci uživatelů přiřazených např. do skupin. Podle typu realmu závisí, jak a kde jsou uloženy údaje pro autentizaci (např. databáze, LDAP, soubor, certifikáty).

Autentizace služeb PKI EJB modulu používá FileRealm, který zajišťuje uložení jmen a hesel uživatelů v souboru aplikačního serveru, kdy hesla nejsou uložena v otevřeném tvaru. Název realmu je `wsRealm`. Jeho vytvoření a správa uživatelů se nejlépe provádí v administrátorské konzoli aplikačního serveru Glassfish, jak je zobrazeno na následujícím obrázku.



Obr. 7.5.4. Autentizační FileRealm PKI EJB modulu

8 DATABÁZE

Při rozhodování, jaký databázový stroj zvolit, hrály roli následující požadavky. Databáze musí běžet v prostředí Windows i Linux, protože vývoj celého systému byl střídavě prováděn pod těmito operačními systémy. V neposlední řadě musí být databáze použitelná i v komerčním prostředí a musí disponovat dobrou dokumentací a nabízet dobré služby podpory. Samozřejmostí je použití transakčního zpracování dat a dobrá implementace jazyka SQL.

Při výběru databázového stroje byla nakonec zvolena databáze *Oracle Database 10g Express Edition*, která všechny výše uvedené požadavky dokonale splňuje. Jde o zdarma šířenou verzi komerční verze databáze od společnosti Oracle. Ale i tak ji lze použít pro běžný provoz malých a středních aplikací. Limitována je objemem uložených dat, jehož výše činí 4GB.

Pro práci databáze s implementovaným systémem byl vytvořen samostatný tzv. *tablespace*, který představuje logickou skupinu souborů pro ukládání dat včetně např. indexů a slouží mimo jiné pro oddělení dat různých aplikací. Dále bylo pro rozhraní BPEL procesů vytvořeno schéma pro uživatele *utbmid* spolu s tabulkami a uloženými procedurami, které pro svou činnost bude potřebovat. Všechny tyto DML příkazy jsou členěny v samostatných skriptech a je možné je nalézt v příloženém archivu této práce.

Jako příklad a ukázkou bych zmínil vytvoření procedury *LOG_XML* z package *LOG_PKG* pro logování BPEL procesů pracujících s XML soubory.

Procedura má na vstupu celkem pět parametrů:

- *PI_LOG_XML_ID* – jednoznačná identifikace záznamu,
- *PI_PROCESS_ID* – jednoznačná identifikace BPEL procesu,
- *PI_TIME_STAMP* – čas práce s XML souborem,
- *PI_PHASE* – fáze zpracování (např. přijato, podepsáno),
- *PI_XML_CONTENT* – obsah zpracovávaného XML souboru.

Za povšimnutí stojí, že procedura mění obvyklý způsob transakčního zpracování. Toho je docíleno uvedením řádku *pragma autonomous_transaction*. Autonomní transakce znamená, že procedura může provádět commit a rollback nezávisle na své rodičovské

transakci, ze které je volána. Díky tomuto je každý logovací záznam v tabulce uložen, i když během zpracování XML souboru dojde k nečekávané chybě a všechny případné databázové operace během tohoto zpracování jsou rollbackovány. Výsledek logování lze vidět na obrázku níže.

LOG_XML_ID	PROCESS_ID	TIME_STAMP	PHASE	XML_CONTENT
149	127.0.1.1:-32ffc912:136f2f49465:-7ea8	27.04.2012 16:29:32,900855000	sign in	PHhtbD48L3htbD4=
150	127.0.1.1:-32ffc912:136f2f49465:-7ea8	27.04.2012 16:29:33,471370000	sign decoded	<xml> </xml>
151	127.0.1.1:-32ffc912:136f2f49465:-7ea8	27.04.2012 16:29:35,413024000	sign signedBase64	PD94bWwgdmVyc2lvbj0IMS4...
152	127.0.1.1:-32ffc912:136f2f49465:-7ea4	27.04.2012 16:29:54,825921000	sign in	PHhtbD48L6823643htbD4=

Obr. 7.5.1. Příklad logovacích záznamů v databázi

Z obrázku je nyní zřejmý každý význam vstupního parametru procedury, stejně tak to, že se jedná o podpis XML dokumentu. Níže pro úplnost uvádím kód pro vytvoření této procedury.

```

PROCEDURE LOG_XML
(
  PI_LOG_XML_ID IN INTEGER default null,
  PI_PROCESS_ID IN varchar2,
  PI_TIME_STAMP IN TIMESTAMP default systimestamp,
  PI_PHASE IN varchar2,
  PI_XML_CONTENT IN CLOB default null
)
IS
pragma autonomous_transaction;
LOG_XML_ID INTEGER;
TIME_STAMP timestamp;
begin
LOG_XML_ID := PI_LOG_XML_ID;
if LOG_XML_ID is null then
  select LOG_XML_ID_SEQ.nextval into LOG_XML_ID from dual;
end if;
time_stamp := pi_time_stamp;
if time_stamp is null then
  time_stamp := systimestamp;
end if;
insert into LOG_XML (LOG_XML_ID, PROCESS_ID, TIME_STAMP, PHASE,
XML_CONTENT)
  values (LOG_XML_ID, PI_PROCESS_ID, TIME_STAMP, PI_PHASE,
PI_XML_CONTENT);
  commit;
end;
```

9 BPEL PROCESY

Business proces v pojetí kontextu JBI znamená uspořádaný sled služeb, který je vytvořen podle určitých pravidel a lze jej podle aktuálních potřeb změnit. K modelování procesů se používá BPEL jazyk. Pro chod procesů je pak nutné běhové prostředí, které popisuje specifikace JBI²⁹, která vznikla za účelem vytvoření standardizované integrační platformy pro jazyk Java. Ve vytvořeném systému toto běhové prostředí v sobě zahrnuje použitý aplikační server GlassfishESB a práci s BPEL procesy plně graficky i textově podporuje vývojové prostředí Netbeans.

Při vývoji jsou procesy umístěny v BPEL modulu, který spolu EJB moduly vytváří tzv. CASA projekt. Název je odvozen od anglického termínu Composite Application. Každý BPEL proces se skládá z několika kroků: příjmu vstupní zprávy (*receive*), zavolání jiné služby (*invoke*), vrácení odpovědi (*reply*), přiřazování hodnot (*assign*) a několika dalších. Tyto kroky lze libovolně skládat do jedné skupiny a spouštět je v tzv. aktivitách, např. sekvence (skupiny jsou vykonávány sekvenčně po sobě), tok (umožňuje skupině běžet paralelně vedle další skupiny). Samozřejmostí je použití např. větvení nebo práci s výjimkami. Každý proces si pak definuje a používá svoje vlastní proměnné [2].

Volání jiné služby se provádí pomocí tzv. *partnerLinku*, který představuje také vstupní prvek pro spuštění procesu. Ten nemusí nutně začínat vstupní zprávou přicházející zvnějšku, ale může to být speciální WSDL soubor odpovídající *cron* výrazu (jakého známe např. z Unixových operačních systémů). Pak je proces spuštěn automaticky v definovaných periodách. Naopak volání jiné služby může znamenat např. zavolání databázové procedury, přenos souboru pomocí protokolu FTP, volání REST služeb, vykonání operace nad LDAP a mnohé jiné. Vše je možné parametrizovat na straně aplikačního serveru, takže např. stejný proces běžící na více aplikačních serverech může používat jinou databázi či volat služby na jiných adresách. Hodnoty parametrů se definují na straně aplikačního serveru.

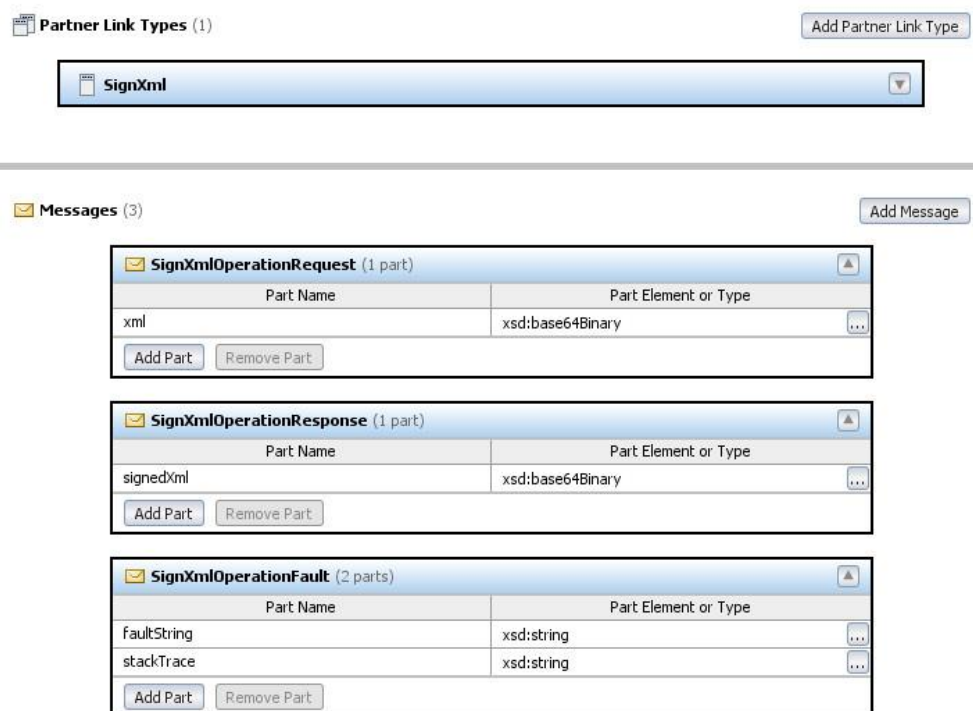
V této práci bylo implementováno celkem šest business procesů v rámci jednoho BPEL modulu. Každý zprostředkovává jednu konkrétní funkcionalitu. To je zajištěno naimportováním WSDL definic služeb PKI EJB modulu. Jde o tyto procesy:

1. podepsání XML,
2. ověření XML,
3. podepsání PDF,
4. ověření PDF,
5. odeslání e-mailu,
6. ověření e-mailu.

V následujícím textu bude na příkladu procesu pro podepsání XML dokumentu ukázáno, jak implementace procesu může vypadat.

9.1 Příjem zprávy z vnějšího okolí

Pro tuto činnost je vytvořená samostatná definice WSDL pomocí grafického rozhraní pro práci s WSDL v prostředí Netbeans, viz. obrázek níže.



Obr. 9.1.1. WSDL pro příjem XML

²⁹ <http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html>

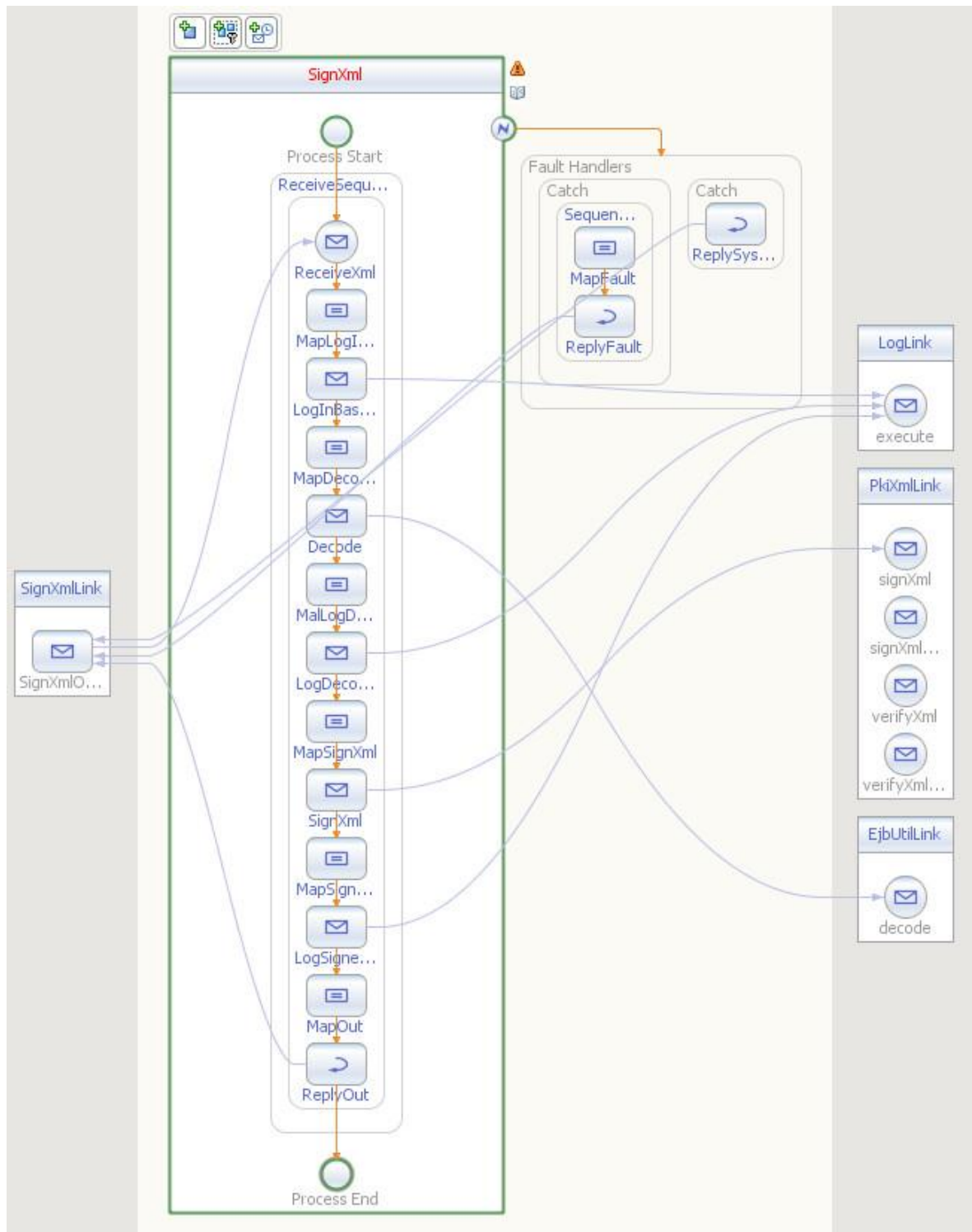
Z obrázku je zřejmé, že jde o jednu operaci s názvem *SignXmlOperation*. Vstup představuje element s názvem *xml*, jehož hodnota představuje dokument určený k podepsání v kódování base64. Výstupem je pak podepsaný dokument v témže kódování. Díky tomuto faktu je možné podepsat jakýkoliv XML dokument, tedy sestavený podle libovolného XDS schématu. Pokud při zpracování vznikne chyba, je vrácen standardní SOAP fault s popisem této chyby. Výhodou je, že pro definování parametrů je možné použít datových typů naimportovaných XSD schématů. Využito je toho např. při definici WSDL pro posílání e-mailů. Stejně jako služby PKI EJB modulu jsou zabezpečeny i služby procesů. Toho je docíleno tentokrát použitím definice *policy* pro BASIC autentizaci. I zde se využívá již zmíněný *FileRealm wsRealm*.

```
<service name="SignXmlService">
  <port name="SignXmlPort" binding="tns:SignXmlBinding">
    <soap:address
location="http://localhost:${HttpDefaultPort}/SignXmlService/SignXmlPort"
/>
    <wsp:PolicyReference
URI="#HttpBasicAuthBindingBindingPolicy"/>
  </port>
</service>
<wsp:Policy wsu:Id="HttpBasicAuthBindingBindingPolicy">
  <mysp:MustSupportBasicAuthentication on="true">
    <mysp:BasicAuthenticationDetail>
      <mysp:Realm realmName="wsRealm"/>
    </mysp:BasicAuthenticationDetail>
  </mysp:MustSupportBasicAuthentication>
</wsp:Policy>
```

Všechny procesy tímto způsobem nabízí rozhraní postavené na webových službách. Při faktu, že jde o servisně orientovanou architekturu, to vlastně není ani žádné překvapení. O zajištění komunikace se stará komponenta *sun-http-binding*, která má své vlastní nastavení. Např. pro port této komunikace. Díky tomu lze jednoduchým způsobem na firewallu povolit jen tento port a naopak pro vnější svět zakázat port, který je využíván pro komunikaci PKI EJB modulu.

9.2 Zpracování požadavku

Zpracování představuje sled kroků, které začíná příjmem zprávy pomocí služby definované v předcházející části. Tato WSDL definice slouží tedy k vytvoření vstupního *parterLinku*. Nejlépe je to patrné na obrázku níže, který představuje celý proces zpracování.



Obr. 9.2.1. Schéma celého BPEL procesu

Na obrázku jsou v pravé části patrné i další *partnerLinky*. První, jmenující se LogLink, slouží k logování celého průběhu zpracování pomocí uložené procedury popsané v předcházející kapitole. Volání je zajištěno speciální definicí WSDL, ve které je uveden

i název JDBC resource pro zajištění k přístupu ke konkrétní databázi. Resource se definuje na straně aplikačního severu. Ta se během zpracování volá hned třikrát. K zalogování vstupního XML dokumentu v base64 kódování, zalogování v čitelném tvaru (tj. v dekodovaném tvaru) a nakonec zalogování výsledného podepsaného dokumentu. Dekódování se provádí pomocí služby dalšího EJB modulu, která je zde reprezentována *partnerLinkem* s názvem *EjbUtilLink*. Výsledek logování je zachycen na obrázku nacházející se v kapitole 8. Samotné podepsání zajišťuje poslední třetí *partnerLink* *PkiXmlLink*, který je vytvořen pomocí importovaného WSDL PKI EJB modulu pro práci s XML dokumenty. Protože jsou služby tohoto modulu zabezpečeny BASIC autentizací, je nutné do naimportovaných WSDL definic přidat následující *policy*.

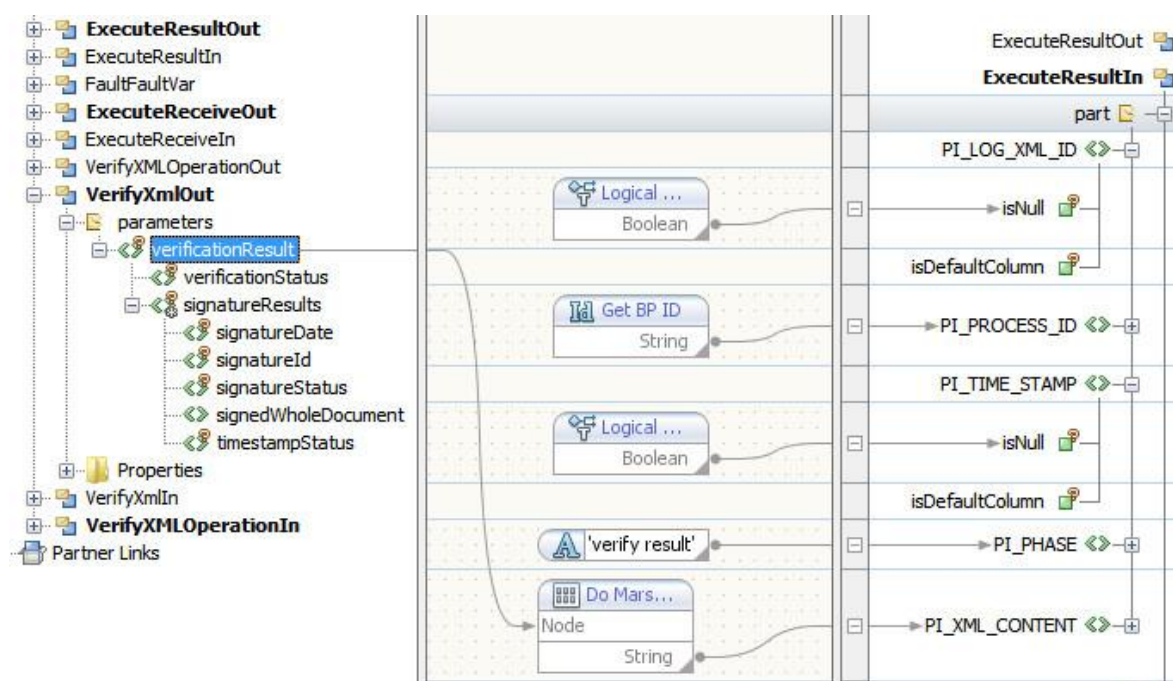
```
<wsp:Policy wsu:Id="HttpBasicAuthBindingBindingPolicy">
  <mysp:MustSupportBasicAuthentication on="true">
    <mysp:BasicAuthenticationDetail>
      <mysp:WssTokenCompare/>
    </mysp:BasicAuthenticationDetail>
  </mysp:MustSupportBasicAuthentication>
  <mysp:UsernameToken
mysp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/I
ncludeToken/AlwaysToRecipient">
  <wsp:Policy>
    <sp:WssUsernameToken10>${pkusername}</sp:WssUsernameToken10>
    <sp:WssPassword>${pkpassword}</sp:WssPassword>
  </wsp:Policy>
  </mysp:UsernameToken>
</wsp:Policy>
```

Jméno a heslo lze zadat přímo, ale z hlediska bezpečnosti a možnosti použití na různých prostředích, je i zde zvolena možnost zadání skutečné hodnoty na straně použitého aplikačního serveru.

Kromě využívání *partnerLinků* a jejich volání lze ještě na obrázku vidět kroky představující přiřazení hodnot, např. právě pro tyto volání. Přiřazení je možné provádět textově, ale i graficky, jak je znázorněno na obrázku níže. Obrázek představuje přiřazení odpovědi PKI EJB modulu na ověření XML dokumentu pro její zalogování do databáze. Proces ověření je použit záměrně, aby bylo skutečně patrné, jak technologie JAX-WS spolu JAXB vývojáři pomáhá. Na class diagramu v sekci 7.5.1 je vidět, už to bylo i zmíněno, že výsledek ověření reprezentuje instance objektu *VerificationResult*. Zde je dobrý příklad toho, jak pomocí těchto dvou technologií lze jakýkoliv objekt použít ve webových službách a vytvořit podle něj XSD schéma. Lze tak zde rovněž vidět, jak pomocí operace *marshall*

převést objekt na řetězec znaků a ten uložit do databáze. Příklad řetězce může pak vypadat např. takto:

```
<?xml version="1.0" encoding="UTF-"?>
<verificationResult>
<verificationStatus>VERIFICATION_OK</verificationStatus>
<signatureResults>
<signatureDate>2012-0319T14:53:49.982+01:00</signatureDate>
<signatureId>xmldsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103</signatureId>
<signatureStatus>SIGNATURE_OK</signatureStatus>
<signedWholeDocument>>true</signedWholeDocument>
<timestampStatus>TIMESTAMP_OK</timestampStatus>
</signatureResults>
</verificationResult>
```



Obr. 9.2.2. Přřazení hodnot v BPEL procesu

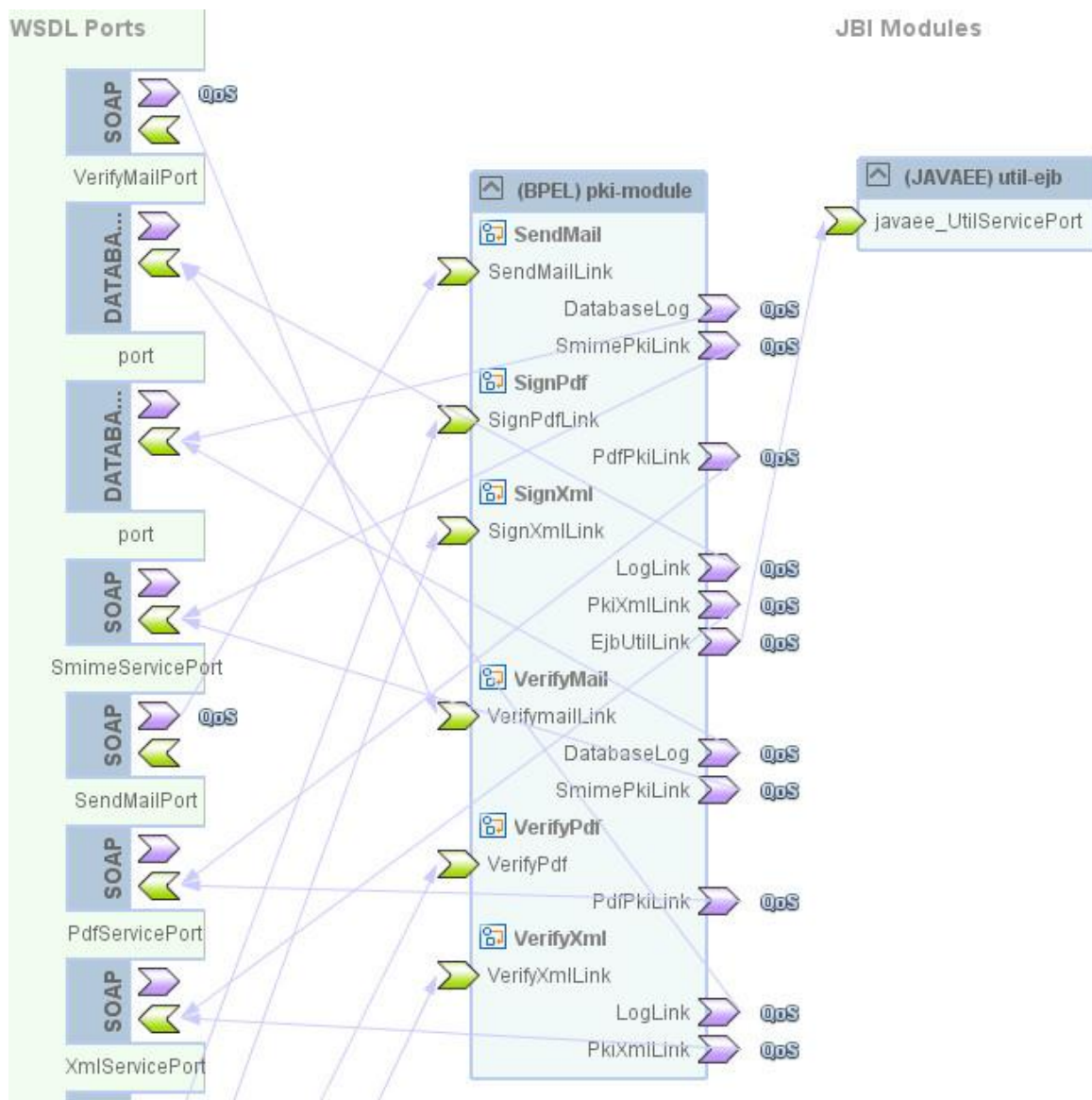
Dále je z obrázku patrné přřazení jedinečného identifikátoru BPEL procesu, názvu fáze zpracování a sdělení, že jedinečný identifikátor a čas záznamu se má vytvořit až za běhu procedury.

9.3 Odpověď a ošetření chyb během zpracování

Pokud vše dopadne dobře, přřadíme kroku *reply* výsledek zpracování a vrátíme jej v odpovědi na volání služby, která stojí na začátku zpracování procesu. V případě chyby se přřadí chybový text a ten se vrátí jako SOAP fault. Na obrázku, který představuje celé schéma procesu, to je zachyceno v pravé horní části. Za zmínku stojí, že se zachycují dvě

výjimky. První je instancí třídy `SignException` nebo `VerificationException`, druhá je pak obecná tzv. `SystemFault` vzniklá např. při chybě během logování.

Na obrázku níže jen pro zajímavost uvádím část schématu celého implementovaného CASA projektu zahrnující všechny BPEL procesy. Zde se např. definují jména parametrů pro zadaná URL služeb PKI EJB modulu, hodnoty by se pak opět definovaly na straně aplikačního serveru.



Obr. 9.3.1. Schéma CASA projektu

10 WEBOVÁ APLIKACE

Služeb PKI EJB modulu nevyužívají jen BPEL procesy, ale i zde implementovaná ukázková webová aplikace. Zdrojové soubory aplikace jsou přiloženy k této práci. Aplikace je implementována jako Enterprise Application skládající se ze tří následujících modulů:

- společného modulu pro konfiguraci všech modulů,
- EJB modulu pro komunikaci s PKI EJB modulem,
- webového modulu.

Výsledným produktem této implementace bude tedy Enterprise Archive.

10.1 Společný modul

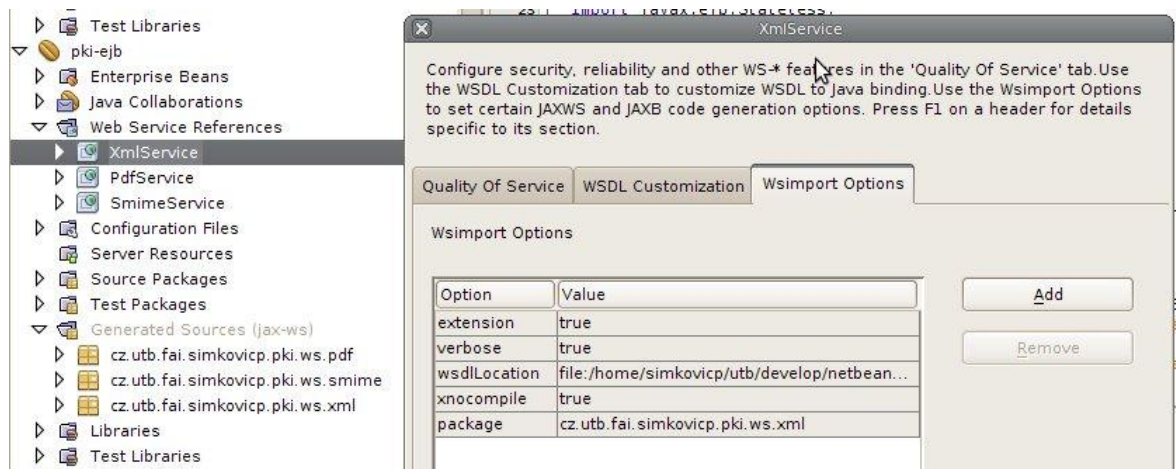
Modul využívá knihovny pro konfiguraci, která již byla zmíněna při podpisu PKI EJB modulu. Jméno konfiguračního souboru bylo zvoleno jako `demo-app.properties`, jeho umístění se opět nachází v adresáři aplikačního serveru a je opět přiložen v zabaleném archivu k práci. Pro lepší práci s konfigurací byla vytvořena třída `AppConfigConstants`, která definuje názvy jednotlivých *properties*. Důvod je jednoduchý, může se stát, že jednu *property* bude používat více modulů. Změna jména této *property* bude pak znamenat jen zásah na jednom místě, a to v deklaraci dané konstanty. Pokud bychom konstanty nepoužívali, musela by se najít všechna místa, kde se tato *property* používá, a změny na nich provést. Patrné to je z ukázky kódu:

```
// pouziti konstanty pro jmeno property
endpoint = configuration.getProperty(
    AppConfigConstants.PKI_WS_XML_ENDPOINT);
// pouziti textoveho retezce
endpoint = configuration.getProperty(„pki.ws.xml.endpoint“);
```

10.2 EJB modul pro komunikaci s PKI EJB modulem

Činnost tohoto modulu je zřejmá už z názvu této části. Jeho hlavní funkcí je zprostředkovávat komunikaci mezi PKI EJB modulem, implementovaným a popsáným v jedné z předcházejících kapitol, a webovým modulem, který bude popsán v další části. Tento modul tedy představuje tzv. *Client Tier* implementované enterprise aplikace.

Na rozdíl od PKI modulu, kde jsou implementovány webové služby, zde jsou implementováni klienti využívající jejich operací. Jejich automatické vytvoření zajišťuje vývojové prostředí Netbeans. To je schopno klienta vytvořit např. z URL, na kterém se nachází definice WSDL, nebo z lokálního WSDL souboru. K jejich vytváření se opět používá JAX-WS. Výsledkem jsou vygenerované objekty, které slouží k volání služby a vytvoření instancí pro předání parametrů operace. Aby se objekty nacházely v balíčku, jehož název chceme zvolit, musíme upravit parametry pro import WSDL definice. To se provádí např. způsobem, jaký je znázorněn na obrázku níže.



Obr. 7.5.2-1. Parametry JAX-WS klienta

Volání webových služeb je implementováno v *Stateless Session Bean* s názvem *WebServiceFacadeBean*. Ta implementuje *Local* rozhraní *WebServiceFacadeLocal* deklarující metody, pro volání služeb PKI EJB modulu z webového modulu. Do této bean jsou injektovány pomocí *@EJB* anotací proxy objekty služeb. Pro službu pro práci s XML dokumenty to může vypadat např. takto:

```
@WebServiceRef(wsdlLocation = "META-INF/wsdl/XmlService.wsdl")
private XmlService_Service xmlService;
```

Zavolání operace pro podpis pak může vypadat následovně.

```
@Override
public byte[] signXml(byte[] xml) throws PKIWebServiceException {
    try {
        XmlService servicePort = xmlService.getXmlServicePort();
        BindingProvider bp = (BindingProvider) servicePort;
        addBindingProperties(bp,
            AppConfigConstants.PKI_WS_XML_ENDPOINT,
            AppConfigConstants.PKI_WS_XML_TIMEOUT);
        byte[] out = servicePort.signXml(xml);
        return out;
    } catch (Exception ex) {
        String msg = "Cannot init web service client for xml
```

```
signing.";  
        LOGGER.error(msg, ex);  
        throw new PKIWebServiceException(ex);  
    }  
}
```

Nejdříve se z proxy objektu získá tzv. port, což představuje objekt obsahující metody odpovídající operacím služby. Díky vytváření klienta pomocí JAX-WS jej můžeme přetypovat na objekt `javax.xml.ws.BindingProvider` a definovat URL služby s spolu `timeoutem`, což bude vysvětleno níže, není to však povinné. Uvádím to však z důvodu, že v metodě `addBindingProperties` se zadává jméno a heslo pro autentizaci, protože služby jsou zabezpečené BASIC autentizací. Další činnost je zřejmé, zavolá se operace pro podepsání XML dokumentu a její výsledek se vrátí jako návratová hodnoty metody. V případě neúspěchu se chyba zalogue a vyhodí se výjimka. Volitelné definování URL služby a `timeoutu`, který definuje čas, po který jsme ochotní čekat na výsledek, je znázorněno níže (pouze názorně, skutečný kód je možné vidět ve zdrojových souborech).

```
// target endpoint address  
bProvider.getRequestContext().put(  
    BindingProvider.ENDPOINT_ADDRESS_PROPERTY, „http://...“);  
// timeout in milliseconds  
bProvider.getRequestContext().put(  
    „com.sun.xml.ws.request.timeout“, 600);  
// define username  
bProvider.getRequestContext().put(  
    BindingProvider.USERNAME_PROPERTY, „pki“);  
// define password  
bProvider.getRequestContext().put(  
    BindingProvider.PASSWORD_PROPERTY, „****“);
```

V implementovaném EJB modulu se *properties* načítají z konfiguračního souboru. Tento způsob implementace umožňuje, že výsledná aplikace může být umístěna na různých serverech a používat služby rozdílných PKI EJB modulů.

10.3 Webový modul

Modul poskytuje rozhraní pro uživatele používající webový prohlížeč. Při jeho implementaci byl použit framework ICEFaces³⁰. Jde o open-source Rich Internet Application development framework založený na standardu JavaServer Faces2. K vývoji

³⁰ <http://www.icesoft.org/>

nabízí Ajaxové komponenty, které se pro demonstraci činnosti PKI EJB hodí a samotný vývoj aplikace značně ulehčí.

Pro jednotný vzhled všech stránek byl použit šablonovací framework Facelets³¹. I v tomto případě jde opět o open source framework, který slouží k tvorbě prezentační vrstvy ve webových aplikacích postavených na JavaServer Faces. Od verze JSF 2 plně nahrazuje technologii JavaServer Pages. Každá definice stránky tak jen obsahuje seznam komponent, které se zobrazují v hlavní části.

Aplikace je lokalizovaná do anglického a českého jazyka. Jaký jazyk je použit, závisí na nastavení jazykových preferencí webového prohlížeče. Každá stránka je pak realizována pomocí jedné JSP stránky a jedné *managed backing bean*. Propojení je definováno v deskriptoru `faces-config.xml`. Demonstrace je provedena na následující hlavní funkcionalitě:

- vytvoření a ověření XML podpisu,
- vytvoření a ověření PDF podpisu,
- poslání a ověření e-mailu.

10.3.1 Vytvoření a ověření XML podpisu

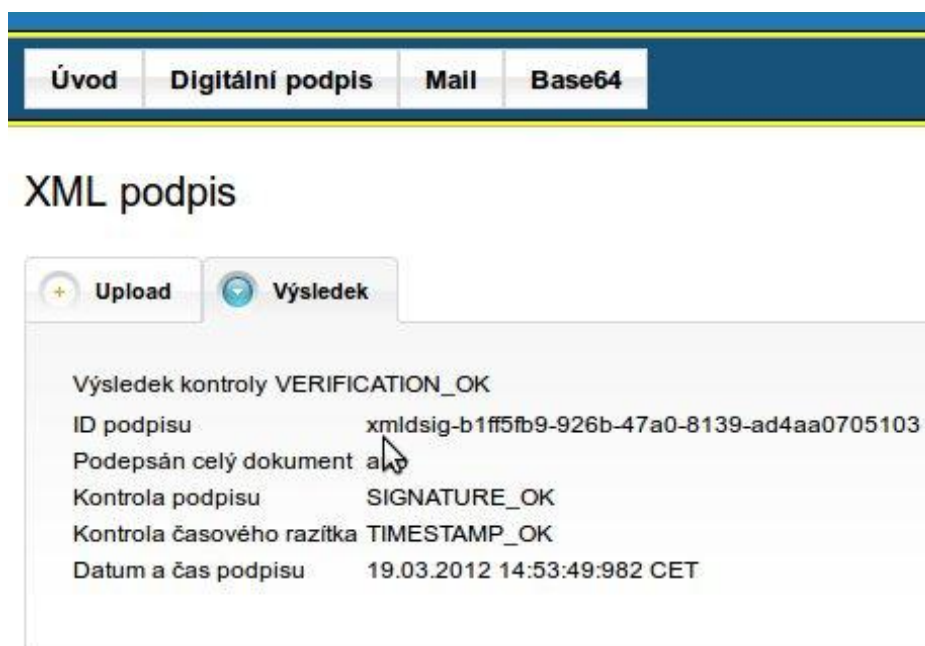
Formulář umožňuje podepsání a ověření digitálního XML podpisu implementovaného v PKI EJB modulu. Pro výběr operace a jejich parametrů, zmíněných v kapitole o PKI EJB modulu, slouží výběrová tlačítka. Pro vložení souboru slouží komponenta, která je patrná na obrázku níže. Po nahrání souboru se XML soubor odešle na PKI, kde se podepíše nebo ověří podpis, a výsledek se zobrazí na dalším panelu. V případě podepsání se nabídne odkaz pro stáhnutí podepsaného dokumentu a náhled souboru. Vše je znázorněno na obrázcích níže.

³¹ <http://facelets.java.net/>



The screenshot shows a web application interface with a blue header containing navigation tabs: 'Úvod', 'Digitální podpis', 'Mall', and 'Base64'. Below the header, the title 'XML podpis' is displayed. The main content area features two tabs: 'Upload' (active) and 'Výsledek'. Under the 'Upload' tab, there is a section titled 'Vyber soubor:' with a file input field, a 'Browse...' button, and an 'Upload' button. Below this, there are two rows of radio button options: the first row has 'Nic', 'Podepsat' (selected), and 'Ověřit'; the second row has 'APACHE', 'JAVA', and 'XADES' (selected).

Obr. 10.3.1. Webové rozhraní pro XML podpis



The screenshot shows the same web application interface as in the previous image, but with the 'Výsledek' tab selected. The main content area displays the verification result for 'VERIFICATION_OK'. The result is presented as a table with the following data:

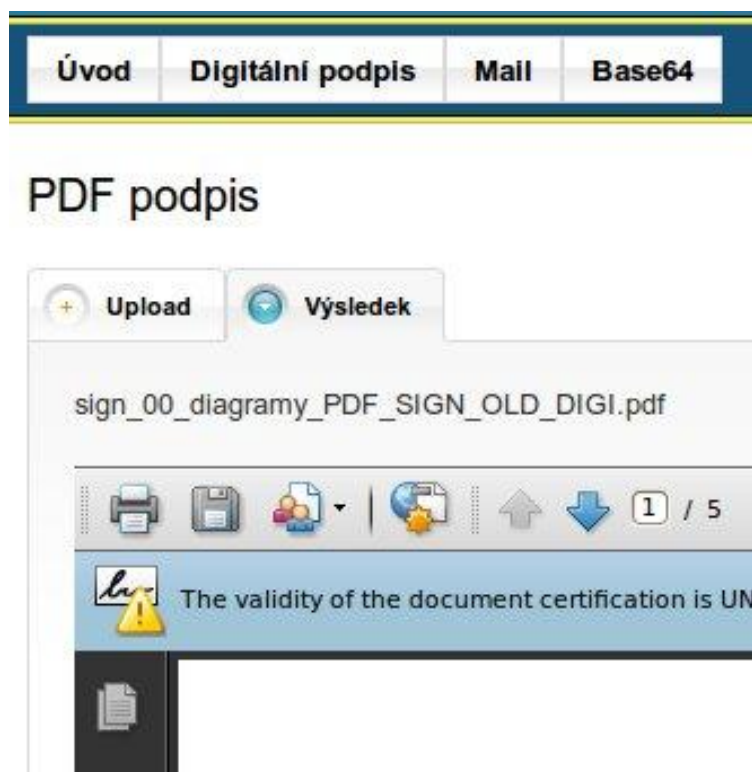
Výsledek kontroly VERIFICATION_OK	
ID podpisu	xmlsig-b1ff5fb9-926b-47a0-8139-ad4aa0705103
Podepsán celý dokument	ano
Kontrola podpisu	SIGNATURE_OK
Kontrola časového razítka	TIMESTAMP_OK
Datum a čas podpisu	19.03.2012 14:53:49:982 CET

Obr. 10.3.2. Výsledek ověření XML podpisu pomocí webového rozhraní

10.3.2 Vytvoření a ověření PDF podpisu

Formulář pro podepsání PDF dokumentu a jeho ověření nabízí podobnou funkcionalitu jako v případě XML. Chybí zde jen volby pro operace. Nebudu zde proto uvádět podrobnější popis. Raději se zmíním o tom, že implementace *managed bean* obou formulářů je provedena pomocí abstraktní třídy zajišťující společné chování obou formulářů. Jednotlivá XML a PDF *managed bean* pak tuto funkcionalitu z této třídy dědí

a překrývají metodu pro podepsání a ověření. V podobném duchu s pomocí Facelets je zajištěn společný vzhled těchto formulářů, viz. zdrojové kódy, kdy je vytvořen společný *template* těchto dvou stránek. Zmiňovaný odkaz pro stažení a náhled podepsaného dokumentu je patrný na obrázku níže.



Obr. 10.3.3. Výsledek podepsání dokumentu pomocí webového rozhraní

10.3.3 Poslání a ověření e-mailu

Pro funkci vytvoření, odeslání a ověření zabezpečené e-mailové zprávy byl rovněž vytvořen samostatný formulář. Pro odeslání e-mailu je možné zadat příjemce v tradičních třech kategoriích, tj. příjemce, příjemce v kopii a skryté kopii. Pokud jich má být více, je nutné jednotlivé příjemce oddělit znakem středník. Dále se zadává předmět a text zprávy. Ke zprávě je možné přiložit přílohy pomocí nahrání souborů. Pro volbu zabezpečení jsou k dispozici tři možnosti výběru:

- podepsat – zpráva bude podepsána,
- šifrovat – zpráva bude poslána v zašifrované podobě,
- zobrazit obsah – po odeslání se automaticky zobrazí zpráva ve tvaru, jakým putuje k příjemci a který je vytvořen dle standardů.

Na další záložce je možné zprávu vložit ve tvaru, který je zmíněn výše, a ověřit ji. Výsledek se poté zobrazí na další záložce. Vše je opět patrné z následujících obrázků.

E-mail

Odesláni S/MIME Ověření

To: petr.simkovic@gmail.com Přílohy: Browse... Upload

Kopie:

Skrytá kopie:

Předmět: Demonstrace

Text:

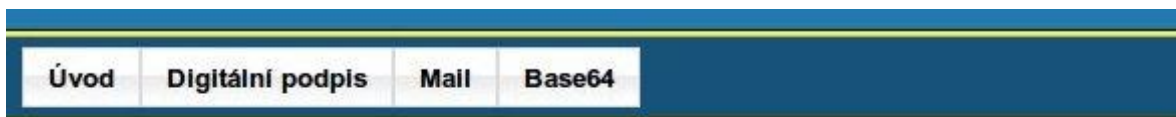
Text

Podepsat Šifrovat Zobrazit obsah Nový Poslat

• diagramy.pdf Odstranit

• 00_diagramy_PDF_SIGN_OLD_chain.pdf Odstranit

Obr. 10.3.4. Webové rozhraní pro odeslání e-mailu



E-mail

Odesláni S/MIME Ověření

Ověřit

From: petr.simkovic2@gmail.com
 To: petr.simkovic@logica.com
 Message-ID: <24530783.5.1336045930633.JavaMail.simkovicp@penguin>
 Subject: Demonstrace
 MIME-Version: 1.0
 Content-Type: application/pkcs7-mime; name="smime.p7m"; smime-type=enveloped-data
 Content-Transfer-Encoding: base64
 Content-Disposition: attachment; filename="smime.p7m"
 Content-Description: S/MIME Encrypted Message

MIAGCSqGSIb3DQEHA6CAMIACAQAxggEwMIIBLAIBADCBI DCBjjELMAkGA1UEBhMCQ1oxDjAMBgNV
 BAgtBU NaR UNIMQ0wCwYDVQQHEwRaTEIOMQwwCgYDVQQKEwNVVEIxDDAKBgNVBA sTA0ZBSTEbMBkG
 A1UEAxMSUGV0ciBTaW1rb3ZpYyAtIENBMScwJQYJKoZIhvcNAQkBFhhwZXRyLnNpbWtvdmljQGxv
 Z2ljYS5jb20CAQwwDQYJKoZIhvcNAQEBBQAEGYCPBpZWVzSrpoSs7mSAMTXZOVik6gKEUi36kIiX
 9AFXFG/cwJdUWm6e6UWO3bmr bFS1I3rYTDfsvBg4qYTNKPIKSSKsudFacwqhF4HtwHO9zaUQOx1i
 EvDcuAFpUVGdXIMieBRJInt0J+VIKinsSYzEhOpUaWTrtis8GaapRZaF1DCABakahkiG9w0BBwEw

Obr. 10.3.5. Formát e-mailu

11 LOGOVÁNÍ

Pro logování aplikace a všech EJB modulů je použita log4j knihovna. Ta byla zvolena především proto, že log4j je thread-safe, optimalizován pro rychlost a nabízí rollovatelný logovací soubor. Všechny logy jsou umístěny v adresáři aplikačního serveru Glassfish `./domain/domain1/log`. Co všechno a do jakého souboru se loguje je závislé na tom, v jaké package se právě vykonávající metoda třídy nachází a jak je definován konfigurační soubor. Pro logování bylo definováno několik appenderů. Všechny jsou typu `RollingFileAppender`. Příklad definice appenderu je níže.

```
log4j.appender.ALL_IN_ONE=org.apache.log4j.RollingFileAppender
log4j.appender.ALL_IN_ONE.File=../logs/all.log
log4j.appender.ALL_IN_ONE.Append=true
log4j.appender.ALL_IN_ONE.MaxFileSize=10MB
log4j.appender.ALL_IN_ONE.MaxBackupIndex=8
log4j.appender.ALL_IN_ONE.Threshold=DEBUG
```

Z příkladu lze například vyčíst, že jméno appenderu je `ALL_IN_ONE`. Dále jméno logovacího souboru, maximální velikost jednoho souboru a maximální počet souborů.

Appender	Package	Jméno logu	Obsah logu
ALL_IN_ONE	všechny	all.log	Všechny třídy včetně použitých knihoven
UTB	cz.utb.fai.simkovicp	utb.log	Všechny třídy celého systému, tj. Napsané v rámci této práce
PKI	cz.utb.fai.simkovicp.pki	pki.log	Třídy PKI EJB modulu
DEMO	cz.utb.fai.simkovicp.demoapp cz.utb.fai.simkovicp.ejb.pki cz.utb.fai.simkovicp.ejb.common	demo-app.log	Třídy webové aplikace a EJB modulů, které využívá (tj. modul pro komunikaci s PKI EJB modulem a common-ejb modul)
JBI	cz.utb.fai.simkovicp.ejb.jbi	jbi.log	Třídy EJB modulů použitých vJBI

10.3-1 Přehled appenderů a logování aplikace

ZÁVĚR

Tato diplomová práce se obecně zabývá problematikou digitálního podpisu a časového razítka. Zmiňován je způsob jejich vytváření a ověřování. Konkrétněji se pak zmiňuje, jak je tato problematika aplikována u konkrétních formátů dokumentů – XML a PDF formáty a jak se dotýká zabezpečení elektronické komunikace pomocí e-mailů.

V teoretické části je tato problematika více rozebrána a popsána. Zejména jde o popis technik, standardů a různých prvků, které je potřeba znát pro implementaci PKI EJB modulu, který představuje jeden z praktických výstupů této práce. V teoretické části lze najít odpovědi a řešení bodů, které jsou uvedeny v pokynech pro vypracování a nejsou implementačního charakteru. Pro úplnost je uvedeno i zasazení této problematiky do legislativy České republiky a Evropské unie.

Praktická část je téměř celá realizována v programovacím jazyku Java a často využívá pokročilých technologií a znalostí Java EE. Kvůli rozsahu práce je vždy zmíněn alespoň krátký úvod do dané technologie. Zbytek praktické práce využívá databáze, kdy výstupy tvoří databázové skripty zajišťující vytvoření prostředí pro uložení dat.

Jádro celé implementace tvoří PKI EJB modul, který zajišťuje kryptografické operace jako je podepsání XML nebo PDF dokumentu a opatření časovým razítkem, dále ověření těchto podpisů, ale také posílání a ověřování zabezpečených e-mailových zpráv. Nad tímto modulem pak byla implementována dvě rozhraní – webové pro klasické webové prohlížeče a rozhraní používající webové služby definované v rámci business procesů. Jednotlivé implementace byly začleněny do jednoho systému, což by čtenáři mělo ulehčit představu o funkci každé části a jak jich lze využívat.

Veškeré součásti praktické části, jako jsou zdrojové kódy implementací v Javě, jejich dokumentace, veškeré databázové skripty, příklady souborů pro logování, použité certifikáty a jiné, jsou uloženy na přiloženém CD i archívu, který je odevzdán spolu s touto prací.

ZÁVĚR V ANGLIČTINĚ

This thesis deals with the issues of the digital signature and digital time stamp. The way of the creating and verification is mentioned too. The detailed focus is represented by PDF/XML document and the security of the e-mail communication.

In the theory part, these issues are described in detail. The point is the description of the techniques, standards and other necessary terms to implement the PKI EJB module. In this part, it is possible to find the answers and solutions of the theoretic instructions. There is a introduction into the legislation of Czech republic and European Union here too.

The most important components of the practical part are implemented and written in Java. These components use the advanced technologies and knowledge of Java EE. Other component is represented by database to storing data.

The main implementation part is PKI EJB module. It allows to perform the cryptographic operations, for example to create the digital signature of XML or PDF document with digital time stamp, to verify these signatures, to send or verify the secure mail messages. This module is used for implementation two interfaces – web interface and interface using the web services defined in business processes. The particular implementations were integrated in the system for the better understanding of its function.

All parts of the practise part (source codes, documentation, database scripts, certificates, atc.) are stored on attached CD and archive which is handed with this thesis.

SEZNAM POUŽITÉ LITERATURY

Monografie:

- [1] VASILIEV, Juri. *Beginning Database-Driven Application Development in JAVA EE Using GlassFish*. 1. vyd. Apress, 2008, 409 s. ISBN 978-1-4302-0963-8.
- [2] SALTER, David a JENNINGS, Frank. *Building SAO-Based Composite Applications Using Netbeans IDE 6*. 1. vyd. Birgmingham: Packet publishing Ltd., 2008, 287 s. ISBN 978-1-847192-62-2.
- [3] DOSTÁLEK, Libor a VOHNOUTOVÁ, Marta. *Velký průvodce infrastrukturou PKI a technologií elektronického podpisu*. 1. vyd. Computer Press, 2008, 536 s. ISBN 80-251-0828-7.
- [4] KEITH, Mike a SCHINCARIOL, Merrick. *Pro EJB 3*. Apress, 2006, 452 s. ISBN 978-1-59059-645-6.
- [5] KNUDSEN, Jonathan B. *Java Cryptography*. 1. vyd. O'Reilly, 1998, 362 s. ISBN 1-56592-402-9.

Internetové zdroje:

- [6] Security Features in Java SE. *Java SE Technical Documentation: The Java Tutorials* [online]. 1995 [cit. 2012-03-28]. Dostupné z: <http://docs.oracle.com/javase/tutorial/security/index.html>.
- [7] Java XML Digital Signatures. *Oracle technology network* [online]. 2006 [cit. 2012-03-28]. Dostupné z: <http://www.oracle.com/technetwork/articles/javase/dig-signatures-141823.html>.
- [8] První certifikační autorita. *Kvalifikovaný certifikát* [online]. [cit. 2012-03-28]. Dostupné z: <http://www.ica.cz/Kvalifikovany-certifikat>.
- [9] První certifikační autorita. *Komerční certifikát* [online]. [cit. 2012-03-28]. Dostupné z: <http://www.ica.cz/Komercni-certifikat>.
- [10] První certifikační autorita. *Kvalifikovaný systémový certifikát* [online]. [cit. 2012-03-28]. Dostupné z: <http://www.ica.cz/Kvalifikovany-systemovy-certifikat>.

[11] W3C. *XML-Signature Syntax and Processing* [online].

[cit. 2012-03-28]. Dostupné z: <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PKI	Public Key Infrastructure – infrastruktura pro správu a šíření veřejných klíčů
CA	certificate authority - certifikační autorita vydávající např. digitální certifikáty
RSA	Rivest-Shamir-Adleman algorithm - šifrovací algoritmus
EDI	Electronic Data Exchange – typ struktury dat
RFC	Request for Comments – označení pro standardy
XML	Extensible Markup Language – rozšiřitelný značkovací jazyk
PDF	Portable Document Format – typ formátu dokumentu
MIME	Multipurpose Internet Mail Extensions – standard pro elektronickou poštu
S/MIME	Secure/MIME – standard pro zabezpečení elektronické pošty
URI	Uniform Resource Identifier - jednoznačná identifikace zdroje
IT	Information Technology - informační technologie
CMS	Cyprographic Message Syntax – standard pro zabezpečení zpráv
ETSI	Evropský telekomunikační institut pro standardy
TSA	Timestamp Authority - Autorita poskytující digitální časové razítko
WSDL	Web Service Description Language - jazyk pro popis webových služeb
JBIC	Java Business Integration – java standard pro výměnu dat pomocí SOAP
API	Application Programming Interface – popis rozhraní, např. jeho metod
URL	Uniform Resource Locator – jednoznačný lokátor zdroje
SOA	Service-oriented Architecture – architektura využívající webové služby
SOAP	Simple Object Access Protocol – protokol popisuje strukturovanou výměnu dat
HTTP	Hypertext Transfer Protocol – protokol pro výměnu hypertextových dokumentů
HTTPS	Hypertext Transfer Protocol Secure - nadstavba HTTP pro zabezpečení
JDBC	Java Database Connectivity – API pro přístup a práci k databázi pro jazyk Java
BPEL	Business Process Execution Language – jazyk pro psaní business procesů

XSD	XML Schema Definition – schéma pro popis XML struktury
JAX-WS	Java API for XML Web Services – Java API pro webové služby
JAXB	Java Architecture for XML Binding – technologie mapování objektů do XML

SEZNAM OBRÁZKŮ

Obr. 1.1.1. Vydavatel certifikátu.....	13
Obr. 1.3.1. Certifikační cesta	15
Obr. 1.3.1. Proces digitálního podpisu.....	17
Obr. 1.3.2. Proces ověření digitálního podpisu.....	18
Obr. 2.1.1. XML podpis.....	22
Obr. 2.2.1. Umístění podpisu v PDF dokumentu.....	23
Obr. 2.2.2. Rozsah bytů	23
Obr. 3.1.1. Ukázka XML podpisu s razítkem	31
Obr. 4.7.1. Architektura EJBCA	38
Obr. 4.7.1. Deployment diagram.....	41
Obr. 6.1.1. Převod bajtů do base64 [5]	43
Obr. 7.1.1. Class diagram podepisování	45
Obr. 7.1.2. Ukázka podepsaného PDF s časovým razítkem	46
Obr. 7.2.1. Class diagram ověřování.....	48
Obr. 7.2.2. Class diagram výsledku ověření	49
Obr. 7.3.1. Class diagram modulu pro zasílání mailů.....	51
Obr. 7.3.2. Zobrazení S/MIME zprávy bez soukromého klíče	52
Obr. 7.3.3. Zobrazení S/MIME zprávy se soukromým klíčem	53
Obr.7.4.1. Class diagram práce s certifikáty	55
Obr. 7.5.1. Operace marshal v JAXB	56
Obr. 7.5.2. Class diagram webových služeb PKI EJB modulu.....	57
Obr. 7.5.3. Ukázka ověření PDF pomocí webové služby PKI EJB	59
Obr. 7.5.4. Autentizační FileRealm PKI EJB modulu	60
Obr. 7.5.1. Příklad logovacích záznamů v databázi.....	62
Obr. 9.1.1. WSDL pro příjem XML	64
Obr. 9.2.1. Schéma celého BPEL procesu	66
Obr. 9.2.2. Přiřazení hodnot v BPEL procesu.....	68
Obr. 9.3.1. Schéma CASA projektu.....	69
Obr. 10.3.1. Webové rozhraní pro XML podpis.....	74
Obr. 10.3.2. Výsledek ověření XML podpisu pomocí webového rozhraní	74
Obr. 10.3.3. Výsledek podepsání dokumentu pomocí webového rozhraní	75

Obr. 10.3.4. Webové rozhraní pro odeslání e-mailu.....	76
Obr. 10.3.5. Formát e-mailu.....	76

SEZNAM TABULEK

2.3-1 Podepisované atributy v S/MIME zprávě.....	28
4.7-1 Části systému.....	42
10.3-1 Přehled appenderů a logování aplikace.....	77