Masarykova univerzita
Fakulta informatiky

Integrace RemSig do klientských aplikací

Bakalářská práce

Ondřej Přikryl

Brno, jaro 2016

Místo tohoto listu vložte kopie oficiálního podepsaného zadání práce a prohlášení autora školního díla.

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Ondřej Přikryl

Vedoucí práce: RNDr. Michal Procházka, Ph.D.

Shrnutí

«abstract»

Klíčová slova

remsig, pkcs11, csp, oauth2.0, ...

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Prvky informačních technologií se stávají každodenní součástí života a většina lidí by si bez nich již nedokázala život představit. Informační technologie uživatelům v mnoha směrech usnadňují život. Slouží jak k práci, tak i ke komunikaci se známými, zábavě a mnoha dalším činnostem. S rozšířením se však zvyšuje i potenciální riziko, které zneužití těchto prvků přináší. V době, kdy internet poskytuje základní komunikaci mezi více než miliardou lidí a je čím dál tím více používán jako nástroj k obchodování, se bezpečnost stává nesmírně důležitou otázkou, která by neměla zůstat nepovšimnuta. Částečnou odpovědí na tuto otázku bylo zavedení kryptografických systémů. I ty však nenabízí řešení ve všech případech a lidé by si měli stále dávat velký pozor.

Kryptografie nebyla původní součástí návrhu internetu, ale v posledních dekádách se díky rozmachu digitálních technologií stala důležitým prvkem internetové infrastruktury. Přestože je skryta před zraky laických uživatelů, jde o důležitý vědní obor, který v současnosti prožívá obrovský posun kupředu. Tento vývoj je způsoben zejména tím, že si uživatelé začínají uvědomovat potřebu chránit své data a soukromí i ve virtuálním světě. Moderní kryptografie má mnoho podob a odvětví, jednou z nich je digitální podpis (též nazýván elektronický podpis), který je nezbytnou součástí systému RemSig.

Systém RemSig je bezpečné vzdálené úložiště certifikátů a privátních klíčů, které uživatelům poskytuje možnost podepisovat dokumenty bez nutnosti neustálého fyzického přístupu k zařízení, na němž je certifikát uložen. Důležitou součástí systému RemSig je uživatelská přívětivost. V současné době je systém možné používat pouze z webového rozhraní INET [1] a IS MU [2] , a proto je nutná implementace knihoven, které integrují funkce RemSigu do klientských aplikací. Náplní mé bakalářské práce je tyto knihovny implementovat a popsat základní prvky CSP a PKCS#11.

TODO popis struktury práce

Výsledkem mé bakalářské práce jsou knihovny propojující RemSig a klientské aplikace . Důsledkem toho již není zapotřebí používat

1. `https://inet.muni.cz`
2. `https://is.muni.cz`

webové rozhraní INET a IS MU, což zaměstnancům MU výrazně usnadní práci při podepisování digitálních dokumentů. Nezávislost systému na webovém rozhraní univerzity také umožňuje jeho rozšíření mimo univerzitní kruhy.

2 Digitální podpis

Digitální podpis slouží k nahrazení obyčejného podpisu v digitálním světě. Je založen na asymetrické kryptografii a k jeho použití je zapotřebí dvojice klíčů - soukromého a veřejného. Digitální podpis zaručuje následující vlastnosti:

- Autenticitu - každý podpis je jednoznačně spojen s uživatelskou entitou, kterou může být jak běžný uživatel, tak i organizace nebo státní útvar (digitální značka).
- Integritu dat - zaručuje, že data nebyla po cestě pozměněna. To je velice důležité k ujištění, že data, která byla odeslána, druhá strana také přijala.
- Nepopiratelnost - autor nemůže popřít, že data nepodepsal. To je zajištěno tím, že k vytvoření podpisu je zapotřebí soukromý klíč, který má vlastník podpisu v držení a není veřejně znám.

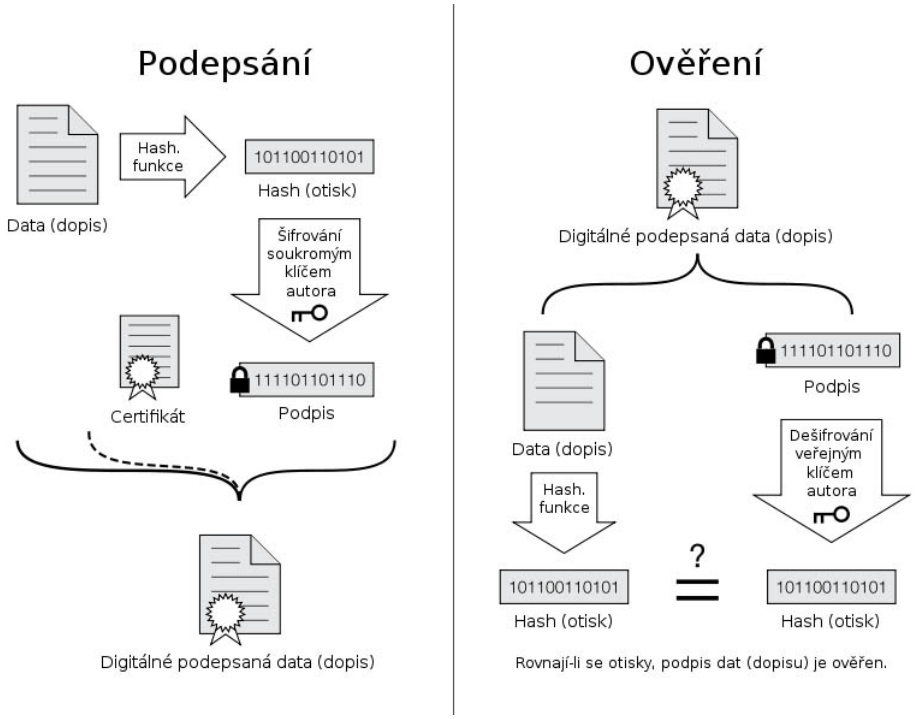
2.1 Princip použití

Uživatel si vytvoří dvojici klíčů, veřejný a soukromý. Soukromý klíč si ponechá a veřejný nechá potvrdit certifikační autoritou a následně jej zvěřejní. Podepsaný veřejný klíč nazýváme ceritifát. Podpis dokumentu poté probíhá následovně:

- uživatel spočítá haš (ang. hash) dokumentu - například pomocí rychlé hašovací funkce SHA256
- haš zašifruje svým soukromým klíčem - například pomocí algoritmu RSA
- data poté odešle společně s certifikátem a zašifrovaným hašem

Ověřování podpisu probíhá analogicky:

- osoba provádějící ověření spočítá haš dokumentu
- za pomoci veřejného klíče osoby která dokument podepsala dešifruje přijatý digitální podpis a ověří, že získaný haš je shodný s vypočteným hašem



Obrázek 2.1: Principy digitálního podpisu.

Pokud si haše odpovídají, pak je jisté, že přijatý dokument je shodný s odeslaným a je tedy zaručena integrita. Aby byla zajištěna nepopiratelnost a autenticita je nutné ověřit, že daný klíč opravdu patří osobě, která poslala podepsaný dokument. Procesu, který toto zajišťuje se říká *přenos důvěry*.

2.2 Přenos důvěry

Aby bylo možné pomocí digitálního podpisu dosáhnout autenticity a nepopiratelnosti, je veřejný klíč dané osoby podepsán důvěryhodnou certifikační autoritou. Tímto způsobem je vyjádřeno, že certifikační autorita věří, že majitel veřejného klíče je skutečně ten za koho se vydává. Uživatel musí certifikační autoritě prokázat svoji totožnost předtím, než získá certifikát. Další možností k dosažení auntenticity je tzv. *síť důvěry*. Jedná se o decentralizovanou alternativu k centralizovanému modelu certifikačních autorit.

3 RemSig

Systém RemSig je bezpečné vzdálené úložiště certifikátů a privátních klíčů, které uživatelům poskytuje možnost podepisovat dokumenty bez nutnosti neustálého fyzického přístupu k zařízení, na němž je certifikát uložen. Díky tomu odpadá uživatelům systému RemSig povinnost nosit s sebou hardwarové zařízení, což zvyšuje uživatelskou přívětivost i funkcionalitu. Uživatelé totiž mohou podepisovat dokumenty, i když u sebe právě nemají token, nebo když jsou na jiném počítači, který není nakonfigurovaný pro práci s digitálními podpisy nebo není důvěryhodný.

3.1 Aktuální využití

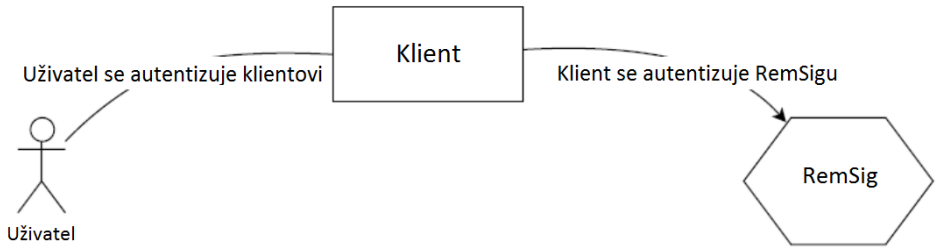
RemSig umožňuje podepisování dokumentů přímo z webového rozhraní systému INET a IS MU a je v souladu se současnou českou legislativou. RemSig je napojen na služby certifikační autority PostSignum [1], což je další výhodou pro zaměstnance MU, kteří tak nemusí řešit vydání certifikátu třetí stranou, ale vše si pohodlně nastaví v rozhraní INET. Aktuálně je používána verze implementovaná v jazyce PHP, ale k dispozici je již nová verze implementovaná v jazyce Java, která již brzy nahradí starou implementaci.

3.2 Autorizace

Pro přístup k API systému je zapotřebí, aby se uživatel autentizoval klientovi, který zprostředkovává komunikaci mezi uživatelem a veřejně přístupnou API RemSigu. Po autentizaci uživatele, Remsig obdrží od klienta unikátní identifikátor, podle kterého určí, ke kterým operacím je klient autorizován. Jestliže se identifikátor nenachází v seznamu pro řízení přístupu (ACL - Access Control List) [2] , klient není oprávněn k žádné operaci.

---

1. Kvalifikovaná certifikační autorita České Republiky, viz. `www.postsignum.cz/`
2. viz. `https://cs.wikipedia.org/wiki/Access_control_list`



Obrázek 3.1: RemSig - Autentizace pomocí klienta.

3.2.1 Autentizace certifikátem

V nové verzi je podporována autentizace certifikátem. Poté, co se uživatel autentizuje klientovi, klient odešle RemSigu certifikát, pomocí kterého RemSig určí, ke kterým operacím je klient oprávněn. Tento přístup však není vyhovovující v případech, kdy chce klient komunikovat přímo s RemSigem. K tomu slouží protokol OpenID Connect.

3.2.2 OpenID Connect

OpenID Connect je již třetí verzí protokolu OpenID. Jedná se o protokol sloužící k ověřování identit, který staví na protokolu OAuth 2.0. OpenID Connect umožňuje klientovi ověřit identitu koncového uživatele a získat základní informace o profilu. Podrobnému popisu protokolu OAuth 2.0 je věnována vlastní kapitola.

3.3 Role

Po úspěšné autentizaci získá klient přístup k metodám odpovídajícím jeho roli. RemSig rozlišuje dva typy rolí:

- Signer - podepisující
- Manager - manažer

Uživatelé systému IS MU mají roli podeposijícího. Ten má přístup k následujícím operacím:

- *sign* - podepíše data podle defaultních kryptografických mechanismů
- *signPKCS7* - vytvoří PKCS#7 [3] podpis dat
- *signPdf* - slouží k podepisování PDF dokumentů

Zatímco manažer má přístup k následujícím operacím:

- *importCertificate* - importuje certifikát do systému RemSig
- *importPKCS12* - importuje strukturu dle standardu PKCS#12 [4]
- *exportPKCS12* - exportuje soukromý klíč a certifikát ve struktuře definované standardem PKCS#12
- *changePassword* - změní heslo, kterým je chráněn soukromý klíč
- *changeCertificateStatus* - změní status certifikátu
- *listCertificates* - vypíše všechny certifikáty uložené v systému RemSig pro danného uživatele
- *checkPassword* - zjistí, zda-li poskytnuté heslo skutečně slouží pro odemknutí privátního klíče

3.4 Funkcionalita

Volání metod probíhá pomocí veřejně dostupné API. Při požadavku se klient musí autentizovat pomocí přístupového tokenu nebo certifikátu. Parametry a data jsou posílány formou XML dokumentu pomocí metody HTTP POST. Data odeslána RemSigu na podpis musí být kódována metodou BASE64[5], přijatá data jsou také kódována pomocí BASE64. Kvůli bezpečnosti je použit protokol HTTPS.

---

3. viz. `http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-7-cryptographic-message-syntax-standar.htm`
4. viz. `http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs12-personal-information-exchange-syntax-standard.htm`
5. viz. `https://cs.wikipedia.org/wiki/Base64`

```
Input XML:
<?xml version="1.0"?>
<remsig>
        <person>
                <uco>1234</uco>
        </person>
</remsig>
```

Obrázek 3.2: Ukázka XML požadavku metody *listCertificates*.

```
OutputXML:
<?xml version="1.0"?>
<remsig>
        <certificate id="1">
                ...    - certificate informations
        </certificate>
        <certificate id="2">
                ...
        </certificate>
        <operationId>1111</operationId>
</remsig>
```

Obrázek 3.3: Ukázka XML odpovědi metody *listCertificates*.

3.4.1 Správa úložiště

Aby uživatel mohl používat systém RemSig, musí v něm mít uložen certifikát a soukromý klíč. Remsig umožňuje generování dvojice soukromého a veřejného klíče přímo v systému. Po vygenerování je soukromý klíč zašifrován pomocí přiloženého PINu, který si uživatel sám zvolí. Dvojice je poté uložena do úložiště a klientovi je nazpět zaslán veřejný klíč, který je potřeba podepsat certifikační autoritou. Podepsaný certifikát je nutno opět odeslat systému. Uživatel však nemusí použít systém pro generování nové dvojice, ale může importovat již existující dvojici ve struktuře PKCS#12 do systému.

- Import certifikátu

  Metoda *importCertificate* slouží k importování certifikátu do úložiště. Certifikát musí být ve formátu PEM. Tato metoda je používána v případě, kdy byla dvojice klíčů vygenerována systémem RemSig a veřejný klíč byl předán na podepsání certifikační autoritou.

- Import struktury PKCS#12

  Metoda *importPKCS12* slouží k nahrání dvojice vytvořené mimo systém RemSig. Tato dvojice je uložena ve struktuře PKCS#12, která je zabezpečena heslem. Poté, co uživatel zadá heslo, je ze struktury exportována dvojice a uložena do úložiště RemSig. Uživatel musí poskytnout nové heslo, kterým bude zašifrován soukromý klíč.

- Export struktury PKCS#12

  Metoda *exportPKCS12* slouží k exportování dvojice ze systému RemSig. Jedinou možností jak exportovat certifikát a soukromý klíč je vytvořením PKCS#12 struktury, do které je po zadání uživatelova PINu k soukromému klíči uložen certifikát a dešifrovaný soukromý klíč. Celá struktura je posléze zamknuta heslem.

3.4.2 Podepisování dat

Jestliže má uživatel k dispozici certifikát a soukromý klíč, může začít používat metody vzdáleného podpisu. RemSig má k dispozici 3 metody na podepisování dat.

- Defaultní podpis

  Metoda *sign* slouží k podpisu dat. K podpisu je použit defaultní kryptografický mechanismus.

- PKCS#7 podpis

  Digitální podpisy se od sebe mohou lišit v použitých algoritmech, ve formátu výstupu a v tom, zda-li výsledný podpis má být přiložen k datům či má být obdržen samostatně.

Metoda *signPKCS7* umožňuje nastavit tyto hodnoty a vytvořit podpis dle standardu PKCS#7. Systém RemSig používá k nastavení těchto hodnot tzv. *prifily*.

Profily
Jestliže má organizace specifické požadavky na PKCS#7 podpis, může si vytvořit profil, ve kterém si nastaví požadované hodnoty. Při vytváření nového profilu musí být nastaveny následující atributy:

- algorithm - algoritmus, který bude použit k podpisu
- no-detach - určuje, zda-li bude podpis přiložen k datům
- encoding - určuje formát výstupu

```
<ceskaposta_01>
<algorithm>SHA1withRSA</algorithm>
<nodetach>TRUE</nodetach>
<encoding>PEM</encoding>
</ceskaposta_01>
```

Obrázek 3.4: Profil české pošty.

Na obrázku je uveden profil české pošty s označením jedna. Metoda s tímto profilem použije k vytvoření haše dat algoritmus SHA1 a výsledný haš zašifruje pomocí algoritmu RSA, podpis bude přiložen společně s daty a výstup bude ve formátu PEM.

- Podpis PDF dokumentu

  Pro podepisování PDF dokumentů slouží metoda *signPdf*. Uživatel si zde může zvolit, zda bude vložen vodoznak do dokumentu či ne. K dispozici je již přednastavený vodoznak pro dokumenty MU. Uživatel si však může vytvořit vlastní nastavení, ve kterém je možné uvést pozici vodoznaku, stránky na kterých bude vodoznak umístěn, obrázek pozadí vodoznaku a text, který bude obsahovat.

4 OAuth 2.0

OAuth 2.0 (RFC 6749) je autorizační protokol (resp. framework), který umožňuje aplikacím třetích stran získat přístup ke službám HTTP, a to buď jménem registrovaného uživatele anebo jménem aplikace třetí strany. Protokol specifikuje pouze autentizaci klienta a výdej přístupového tokenu, přístup k jednotlivým zdrojům je již zcela nezávislý na protokolu. Je tedy zapotřebí veřejně dostupné API. Protokol OAuth 2.0 nahrazuje starý protokol OAuth 1.0 a není s ním zpětně kompatibilní.

4.1 Motivace

V tradičním modelu autentizace klient-server, žádá-li uživatel o přístup k zabezpečenému obsahu, je zapotřebí jeho autentizace pomocí přihlašovacích údajů. Aby tedy aplikace třetí strany mohla přistoupit k témuž obsahu, je zapotřebí s ní sdílet přihlašovací údaje oprávněné osoby. To však s sebou přináší několik problémů:

- Aplikace třetích stran si uchovávají přihlašovací údaje k budoucímu použití. Údaje bývají většinou uchovávány v otevřené podobě. Uchovávání pouze otisku je nedostatečné.
- Uživatel ve většině případů nemůže omezit přístup aplikace k chráněnému obsahu, a to jak časovou platností, tak i omezením obsahu, ke kterému získá aplikace přístup.
- Obvykle lze odebrat přístup aplikaci k zabezpečenému obsahu pouze změnou hesla. To se však projeví i u dalších aplikací, které jsou závislé na stejných přihlašovacích údajích.
- Kompromitace aplikace třetí strany znamená ohrožení přihlašovacích údajů uživatele a všech souborů na nich závislých.

OAuth tyto problémy řeší zavedením autorizační vrstvy a oddělením rolí klienta (aplikace) a vlastníka chráněného zdroje. V protokolu aplikace zažádá o přístup ke chráněnému obsahu a jsou jí přiděleny přihlašovací údaje (přístupový token) odlišné od přihlašovacích údajů uživatele.

4.2 Role

OAuth 2.0 definuje 4 typy rolí:

- Vlastník zdrojů:
  Entita schopná přidělit přístup ke chráněnému obsahu. Jedná-li se o osobu, nazýváme ji koncový uživatel.
- Server zdrojů:
  Poskytovatel a hostitel chráněného zdroje, schopný obsluhovat požadavky ke chráněnému zdroji pomocí přístupového tokenu.
- Klient:
  Aplikace, která vyžaduje přístup ke chráněnému obsahu.
- Server zdrojů:
  Server, který vydává klientovi přístupový token po úspěšné autentizaci oprávněné osoby.

Protokol nijak nespecifikuje interakci mezi autorizačním serverem a serverem zdrojů. Může se jednat o nezávislé entity, ale i o jeden server.

4.3 Přístupový token

Pokaždé když klient potřebuje přistoupit k chráněnému obsahu, potřebuje přístupový token (anglicky *access token*). Přístupový token je řetězec znaků, který opravňuje klienta k přístupu k chráněnému obsahu. V řetězci jsou skrytě uloženy informace o délce platnosti tokenu, o rozsahu přístupu k chráněnému obsahu a informace o vlastníkovi obsahu. Délka platnosti se může lišit implementací, ale typicky bývá platnost tokenu nastavena na 1 hodinu (3600 sekund).

4.4 Obnovující token

Obnovující token (anglicky *refresh token*) je získán společně s přístupovým tokenem a slouží k obnovení přístupového tokenu po vypršení jeho platnosti. Možnost získání přístupového tokenu pomocí obnovujícího tokenu bývá ve většině případech omezena počtem.

4.5 Průběh protokolu

OAuth definuje několik základních způsobů autorizace vedoucích k získání přístupového tokenu. Patří mezi ně:

- Autorizačním kódem:
  Jedná se o doporučený průběh, ke kterému je zapotřebí uživatelský agent (prohlížeč). Klient s použitím prohlížeče přesměruje uživatele na přihlašovací stránku. Po úspěšné autentizaci zašle autorizační server klientovi autorizační kód. Klient pro získání přístupového tokenu zašle autorizační kód autorizačnímu serveru.

- Implicitně:
  Tento typ je zjednodušením průběhu autorizace autorizačním kódem. Slouží pro aplikace běžící v prohlížeči, které používají skriptovací jazyky jako je například JavaScript. Narozdíl od předchozího typu, klient získá přístupový token přímo po přihlášení. Tento typ je rychlejší, jelikož redukuje počet výzev a odpovědí, které je potřeba udělat pro získání tokenu.

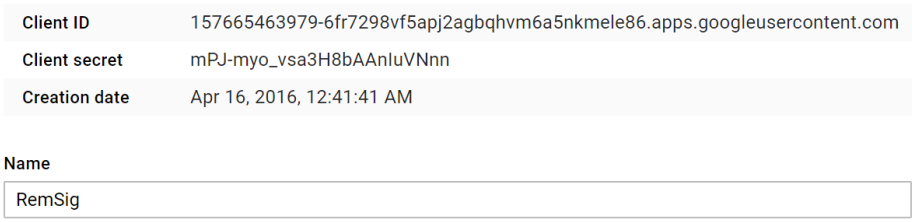
- Zasláním přihlašovacích údajů:
  V tomto typu uživatel poskytne přihlašovací údaje klientovi. Klient zašle přihlašovací údaje autorizačnímu serveru, který po úspěšné autentizaci uživatele zašle zpět přístupový token. Přihlašovací údaje jsou použity jen pro získání přístupového a obnovujícího tokenu, není tedy potřeba jejich uložení pro budoucí použití. Tento typ je používán pouze při vysoké důvěře mezi uživatelem a aplikací.

4.6 Registrace klienta

Předtím než začne klient používat protokol, je potřeba jeho registrace na autorizačním serveru. Pro zvýšení bezpečnosti by klient měl poskytnout:

- jeho typ - webová aplikace, nativní aplikace, zařízení
- adresu, na kterou bude token přesměrován
- jakoukoliv jinou informaci, kterou autorizační server požaduje (název, popis, smluvní podmínky, atd...).

Poté co se klient úspěšně zaregistruje, obdrží od autorizačního serveru unikátní identifikátor a heslo. Tyto údaje jsou nutné pro autentizaci klienta serveru během průběhu protokolu. Je-li při průběhu vyžadována adresa přesměrování, tak musí souhlasit s adresou uvedenou klientem při registraci.



Obrázek 4.1: Google OAuth - registrace klienta.

5 PKCS#11

S nástupem kryptografie přišla nutnost zavést jednotné protokoly a postupy pro zajištění kompatibility koncových aplikací. Ačkoliv se vývojáři shodli na základních kryptografických postupech, kompatibilita mezi jednotlivými implementacemi nebyla nikdy zaručena. Jednotný standard byl tedy nutností. Tuto potřebu naplnila organizace RSA Laboritories zavedením jednotného standardu Public-Key Cryptography Standards, zkráceně PKCS. Jedná se o celou rodinu standardů mezi které patří například i PKCS#11, který je využit v implementaci knihoven RemSig.

PKCS#11 je platformově nezávislé aplikační programové rozhraní (API) pro kryptografické tokeny, nazývané Cryptoki podle Cryptographic Token Interface. PKCS#11 je nyní ve verzi 2.20 a to již od roku 2004. S ohledem na stáří protokolu probíhá od roku 2009 testování nové verze protokolu s označením 2.30, který však ještě nebyl standardizován.

5.1 Cíle návrhu

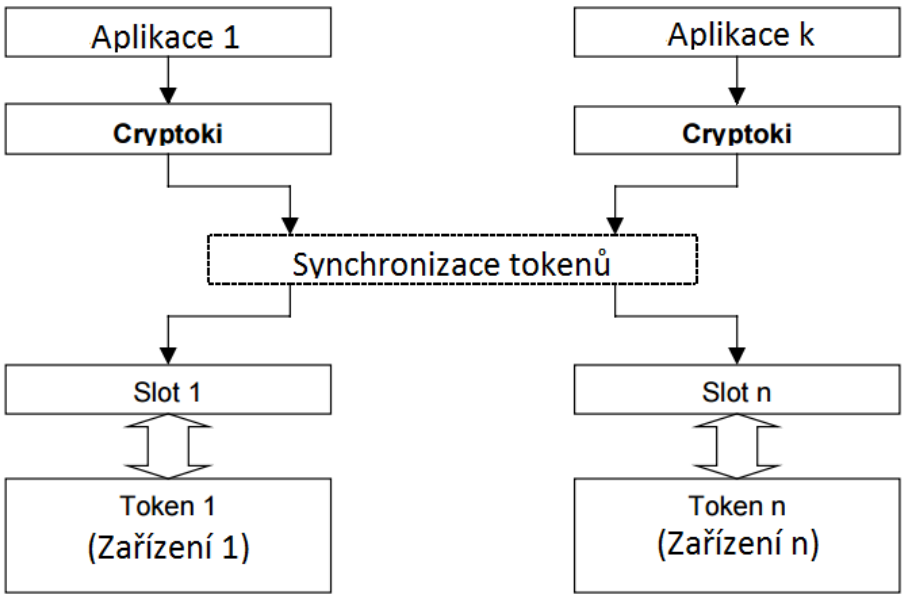
Cryptoki byl navržen s důrazem na jednotnost přístupu k rozdílným zařízením. Cílem rohraní je poskytnout vyšší aplikační vrstvě shodný přístup ke kryptografickému zařízení bez ohledu na konkrétní typ zařízení. Může se tak jednat o Smart Card [1] , PCMCIA [2] kartu nebo síťovou službu, aplikace využívající Cryptoki s nimi může pracovat bez ohledu na jejich hardwarovou odlišnost.

Druhotným cílem bylo sdílení prostředků. S rozvojem víceúlohových operačních systémů se stalo žádoucím být schopen sdílet jedno zařízení mezi více aplikacemi. Stejně tak jedna aplikace by měla být schopna operovat s více kryptografickými zařízeními.

1. viz. `https://cs.wikipedia.org/wiki/Smart_Card`
2. viz. `https://cs.wikipedia.org/wiki/PCMCIA`

5.2 Model návrhu

Model použití Cryptoki začíná jednou nebo více aplikacemi, které požadují přístup k jednomu nebo více kryptografických zařízením. Tyto zařízení provedou operace požadované aplikací a vrátí výsledek.



Obrázek 5.1: PKCS11 - Obecný model.

Cryptoki je tedy rozhraní mezi vyšší vrstvou uživatelských aplikací a nižší hardwarovou vrstvou, která vykonává kryptografické operace. Jak bylo popsáno výše, tato hardwarová vrstva může být reprezentována kartami, tokeny nebo dokonce vzdálenými síťovými službami.

Díky tomu, že Cryptoki maskuje skutečnou hardwarovou podobu připojeného zařízení a zobrazuje je jako totožné jednotky, volající aplikace nemusí znát přesné rozhraní ke kterému je zařízení připojeno či jaký ovladač pro ně použít.

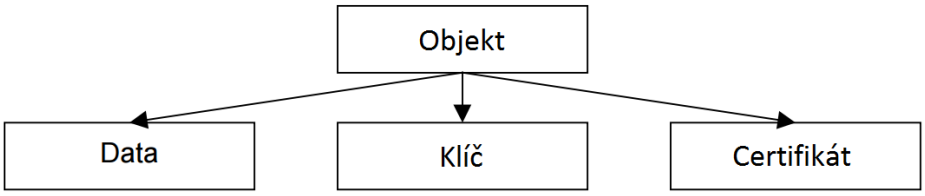
Je nutné podotknout, že Cryptoki je pouze standard, nikoli knihovna. Tímto standardem je definované pouze rozhraní poskytované aplika-

cím, nikoliv podporované vlastnosti. Tyto se mohou lišit podle dané implementace a navíc standard nepředpisuje jaké vlastnosti mají být implementovány.

5.2.1 Objekty

Cryptoki vidí token jako zařízení, na kterém jsou uchovávány objekty a může vykonávat kryptografické operace. Cryptoki definuje 3 základní typy objektů:

- data - jsou definovány a spravovány aplikacemi
- certifikáty - uchovává certifikáty veřejného klíče
- klíče - uchovává kryptografické klíče



Obrázek 5.2: PKCS11 - Objekty.

Objekty jsou zároveň klasifikovány dle jejich životnosti a viditelnosti. Podle životnosti jsou objekty rozděleny na *session* a *token* objekty. *Token objekty* jsou objekty viditelné všem aplikacím připojeným k tokenu a jejich životnost není ovlivněna žádnou relací (tzn. setrvávají na tokenu dokud nejsou odstraněny). Zatímco *session objekty* jsou viditelné pouze aplikaci, která je vytvořila a zanikají s ukončením relace. Objekty se dále mohou dělit na soukromé a veřejné. Pro přístup k veřejnému objektu se aplikace nemusí autentizovat tokenu, zatímco u soukromých je požadováno přihlášení pomocí PINu nebo jiné autentizační metody (např. biometrická zařízení).

5.2.2 Uživatelé

Cryptoki definuje dva typy uživatelů:

- Security Officer (dále již jen SO)
- normální uživatel

Oba vykonávají v systému různé úlohy, ale standard nedefinuje jejich vzájemný vztah. Je tedy v pořádku, když je stejná osoba normálnímm uživatelem i SO.

Účelem SO je inicializovat token a nastavit normálnímu uživateli PIN nebo jinou formu autentizace. SO, na rozdíl od normálního uživatele, dokáže přistoupit pouze k veřejným objektům a nikoli k soukromým.

Normální uživatel je jediný kdo může přistupovat k soukromým objektům, ale až poté co se úspěšně autentizuje. Autentizace normálního uživatele může proběhnout až poté, co SO nastaví uživateli jeho PIN.

5.2.3 Relace

Pro práci s funkcemi a objekty tokenu je potřeba vytvořit alespoň jednu relaci (ang. *session*). Jedná se o logické spojení, ve kterém aplikace komunikuje s tokenem. Relace se dělí na dva základní druhy:

- R/O relace (pouze pro čtení)
- R/W relace (pro čtení i zápis).

R/W relace umožňují aplikaci číst, vytvářet, modifikovat a mazat objekty uložené na tokenu, zatímco R/O relace má přístup pouze ke čtení těchto objektů. Toto omezení se však vztahuje pouze na *token objekty*, nikoliv na objekty vytvořené relací, ke kterým mají oba dva typy plný přístup. Po vytvoření relace má aplikace přístup pouze k veřejným objektům. Aby relace poskytovala přístup k soukromým objektům tokenu, je nutné aby se uživatel autentizoval tokenu.

Cryptoki umožňuje aplikacím vytvářet k jednomu tokenu více relací, ale zároveň také umožňuje tokenu omezit počet R/O a R/W relací, které může aplikace využít. Při zániku relace jsou odstraněny všechny *session objekty* této relace a to i v případě, že jsou tyto objekty

používány jinou relací.

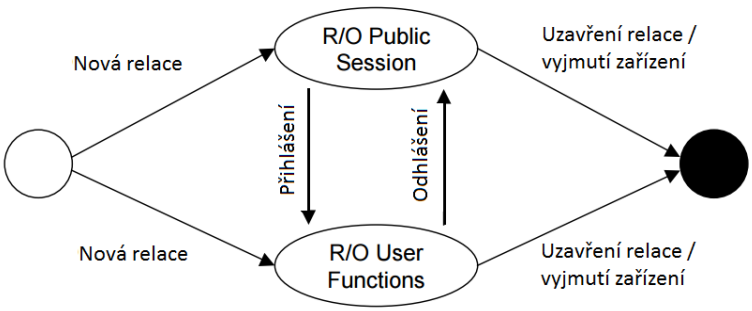
Stavy
Otevřená relace může být v jednom z pěti možných stavů. Tyto stavy definují, které objekty budou dostupné a jaké operace na nich budou povoleny. Pro R/O relace platí:

- R/O Public Session

  Aplikace si otevřela R/O relaci a doposud není autentizována tokenu. Aplikace má právo ke čtení veřejných *token objektů* a právo ke čtení/zápisu veřejných *session objektů*.

- R/O User Functions

  Aplikace si otevřela R/O relaci a uživatel se autentizoval tokenu. Aplikace má přístup ke čtení všech *token objektů* a právo ke čtení/zápisu všech *session objeků*.



Obrázek 5.3: PKCS11 - Objekty.

- R/W Public Session

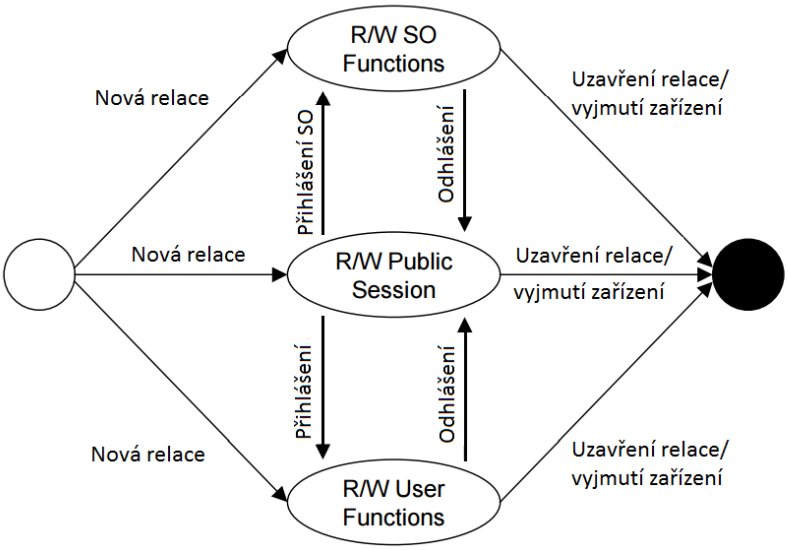
  Aplikace si otevřela R/W relaci a doposud není autentizována tokenu. Aplikace má právo ke čtení/zápisu všech veřejných objektů.

- R/W User Functions

  Aplikace si otevřela R/W relaci a uživatel se autentizoval tokenu. Aplikace má přístup ke čtení/zápisu všech objektů.

- R/W SO Functions

  Aplikace si otevřela R/W relaci a SO se autentizoval tokenu. Aplikace má přístup pouze ke čtení/zápisu všech veřejných *token objeků*. SO může nastavit PIN uživateli.

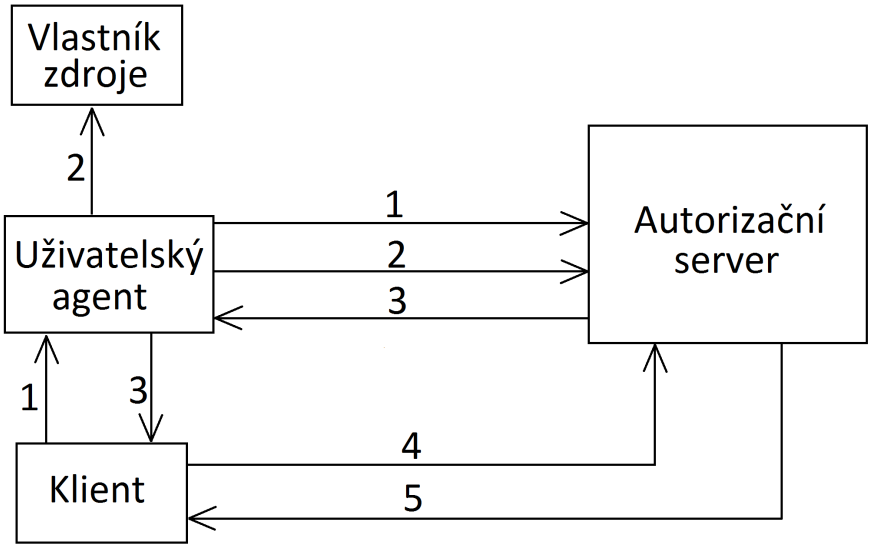


Obrázek 5.4: PKCS11 - Objekty.

6 Praktická část

6.1 OAuth 2.0

V praktické části jsem implementoval funkce standardního způsobu autorizace autorizačním kódem. Parametry jsou zasílané metodou POST.



Obrázek 6.1: OAuth 2.0 - Autorizace autorizačním kódem.

1. Klient pomocí prohlížeče přesměruje vlastníka zdroje na autorizační server. Klient v požadavku uvede svůj identifikátor, požadovaný rozsah přístupu, identifikační řetězec [1] a adresu, na kterou má být zaslán autorizační kód.

2. Vlastník zdroje se autentizuje serveru a potvrdí klientovi přístup o daném rozsahu.

1. Identifikační řetězec (tzv. state) slouží k identifikaci sezení při větším počtu různých OAuth požadavků.

3. Autorizační server přesměruje prohlížeč na adresu obdrženou v prvním kroku a předá jí autorizační kód a identifikační řetězec obdržený v prvním kroku.

4. Klient zažádá autorizační server o přístupový token. V požadavku uvede autorizační kód přijatý v předchozím kroku, identifikátor klienta, příslušné heslo a adresu použitou k získání autorizačního kódu v prním kroku.

5. Autorizační server zkontroluje identifikační údaje klienta a ověří platnost autorizačního kódu. Jestliže jsou všechny údaje platné, autorizační server zašle přístupový a obnovující token klientovi.

6.2 Úpravy systému RemSig

RemSig v nynější podobě rozeznává dvě základní role. Role *Signer* je určena pouze pro podepisování dat, zatímco role *Manager* pouze pro správu systému. Implementace knihoven však využívá funkcionalitu obou rolí. Z tohoto důvodu bude vytvořena nová role, která bude obsahovat následující funkce:

- *listMyCertificates* - vypíše certifikáty přihlášeného uživatele
- *checkPassword* - zkontroluje heslo k privátnímu klíči
- *sign* - vytvoří digitální podpis

Nová role bude určena přímo pro knihovny integrující RemSig do klientských aplikací. Jelikož je jako autentizační metoda těchto knihoven použit protokol OAuth a pro server je možné získat z přístupového tokenu edentitu uživatele, všem výše uvedeným metodám je odebrán parameter *"uco"*, který sloužil k identifikaci uživatele.

6.3 PKCS#11 modul

V druhé části praktické práce jsem implementovat PKCS#11 modul, který slouží k propojení systému RemSig a aplikací, které modul PKCS#11 podporují. Mezi tyto aplikace patří například Mozilla Firefox,

Safari, Mozilla Thunderbird, Evolution a mnoho dalších [2]. Standard PKCS#11 nepředpisuje, aby všechny jeho funkce byly implementovány, avšak definuje, které funkce by měli být implementovány, aby modul měl nějakou funkčnost. S ohledem na systém RemSig byl vytvořen profil PKCS#11, který umožňuje práci s digitálním podpisem. Virtuální zařízení - token, je zobrazováno jako zařízení pouze pro čtení a tím pádem všechny funkce, které vytvářejí, upravují nebo mažou objekty nejsou podporovány. Každý aktivní token je již inicializován a uživatel má nastavené přihlašovací heslo. Aktivní token zároveň obsahuje právě jeden certifikát a jeden privátní klíč, k danému certifikátu.

6.3.1 Obecné funkce

C_Initialize - inicializuje celý modul a vytvoří kontext klientské strany, který slouží k uchovávání informací o modulu. Tato funkce dále nastavuje logování a získává certifikáty z úložiště RemSig. Ke každému certifikátu obdrženému metodou *listMyCertificates* je vytvořen virtuální token, který je posléze připojen do virtuálního slotu.

C_Finalize - tato funkce je volána při ukončení práce s modulem. Slouží k uvolnění kontextu a k uzavření souboru logování.

C_GetFunctionList - obsahuje seznam všech implementovaných funkcí modulu.

C_GetInfo - poskytuje informace o modulu. Je zde uvedena použitá verze cryptoki a název, výrobce a verze modulu.

6.3.2 Správa modulu

C_GetSlotList - slouží k získání seznamu dostupných slotů. Aplikace může určit, zda-li má být navrácen kompletní seznam nebo pouze seznam slotů, ve kterých je přítomen token.

C_GetSlotInfo - představuje základní informace o každém statickém slotu.

---

2. Seznam podporovaných aplikací: `https://github.com/OpenSC/OpenSC/wiki/Using-smart-cards-with-applications`.

C_GetTokenInfo - slouží k získání informací o tokenu. Funkce nastaví tokenu hodnoty popis, seriové číslo a vydavatel podle certifikátu, který obsahuje. Dále každému tokenu nastaví následující vlastnosti:

- *CKF_WRITE_PROTECTED* - token je pouze pro čtení.
- *CFK_LOGIN_REQUIRED* - uživatel musí být přihlášen pro přístup k některým objektům a funkcím.
- *CKF_USER_PIN_INITIALIZED* - PIN uživatele je již nastaven, zabrání volání funkce *C_InitPin*.
- *CKF_TOKEN_INITIALIZED* - token je již inicializován, zabrání volání funkce *C_InitToken*.

C_GetMechanismList - funkce vrací seznam podporovaných kryptografických mechanismů tokenu. V seznamu se nyní nachází pouze jeden prvek a to *CKM_SHA256_RSA_PKCS*, který odpovídá základnímu mechanismu metody *sign* používané systémem RemSig.

C_GetMechanismInfo - slouží k získání podrobných informací o mechanismu.

6.3.3 Správa relací

C_OpenSession - vytvoří novou relaci mezi aplikací a tokenem. Jestliže má token již otevřenou existující relaci s aplikací, synchronizuje jejich stavy.

C_CloseSession - uzavře relaci a uvolní paměť.

C_CloseAllSessions - ukončí všechny relace a uvolní paměť, je volána před ukončením práce s modulem.

C_GetSessionInfo - poskytuje informace o relaci, její stav, ID slotu, ke kterému je vytvořena a další.

C_Login - autentizuje uživatele tokenu. Při úspěšném přihlášení, funkce

nastaví uživateli příslušnou roli, čímž uživatel získá přístup k privátním objektům a funkcím. Všechny relace, které souvisí se stejným tokenem jsou následně aktualizovány do přihlášeného stavu. PIN je uložen do kontextu tokenu.

C_Logout - odhlásí uživatele a uvolní paměť. Všechny relace jsou aktualizovány do nepřihlášeného stavu.

6.3.4 Práce s objekty a digitální podpis

C_FindObjectsInit - inicializuje vyhledávání objektů, poskytnutá šablona je uložena do kontextu relace, která použila funkci.

C_FindObjects - zahájí samotné vyhledávání objektů. Každý token vlastní jeden certifikát a k němu jeden privátní klíč. Jestliže je objekt nalezen, funkce vrací identifikátor objektu.

C_FindObjectsFinal - ukončí operaci vyhledávání a uvolní šablonu z relačního kontextu.

C_GetAttributeValue - funkce slouží k získání atributu objektu, opět je poskytnuta šablona, která určuje, které atributy jsou požadovány.

C_SignInit - autentizuje uživatele tokenu. Při úspěšném přihlášení, funkce nastaví uživateli příslušnou roli, čímž uživatel získá přístup k privátním objektům a funkcím. Všechny relace, které souvisí se stejným tokenem jsou následně aktualizovány do přihlášeného stavu.

C_Sign - odhlásí uživatele a uvolní paměť.

6.3.5 Práce s modulem

Jestliže chce aplikace pracovat s modulem, pak musí pro jeho plnou funkčnost splnit sérii příkazů. Typickým příkladem práce aplikace s modulem je následující posloupnost:

1. Uživatel musí získat přístupový token pomocí OAuth aplikace. Jestliže token není k dispozici, modul se nepodaří inicializovat.

2. Aplikace inicializuje modul, který získá certifikáty ze systéme RemSig.

3. Poté si aplikace vylistuje dostupné tokeny a vybere si token, se kterým chce pracovat. Aplikace si může získat informace o tokenu a mechanismech, které podporuje.

4. Pro práci s tokenem si aplikace vytvoří jednu nebo více relací pomocí funkce *C_OpenSession*.

5. V tomto stavu je možné vyhledat veřejné objekty tokenu. Pro přístup k privátním objektům je zapotřebí se přihlásit pomocí funkce *C_Login*. PIN k tokenu je odeslán systému RemSig pro ověření správnosti. Po úspěšném přihlášení jsou aktualizovány všechny relace do přihlášeného stavu.

6. Aplikace inicializuje vyhledávání objektů pomocí funkce *C_FindObjectsInit*, přičemž funkci předá šablonu, která specifikuje hledaný objekt.

7. Následuje samotné vyhledávání objektů. Funkce *C_FindObjects* získá identifikátory objektů, které se shodují s parametry šablony.

8. Vyhledávání je ukončeno zavoláním funkce *C_FindObjectsFinal*, paměť obsahující šablonu je uvolněna.

9. Aplikace vykoná kryptografické operace, v našem případě digitální podpis dat. Pro použití jeného tokenu není potřeba opět inicializovat modul, ale lze začít od kroku číslo 3.

10. Při ukončení práce s modulem jsou zavolány funkce *C_CloseSessions* nebo *C_CloseAllSessions*, které ukončí relace mezi aplikací a tokenem a následně *C_Finalize*, který uvolní kontext modulu.