
KIV/BIT – Semestrální práce

# Implementace elektronického podpisu s využitím RSA a SHA-1

11. dubna 2017

*Autor:*
František Pártl
A15B0305P
fpartl@students.zcu.cz

Celková doba vypracování činí přibližně 32 hodin.

# Obsah

# Kapitola 1

# Zadání

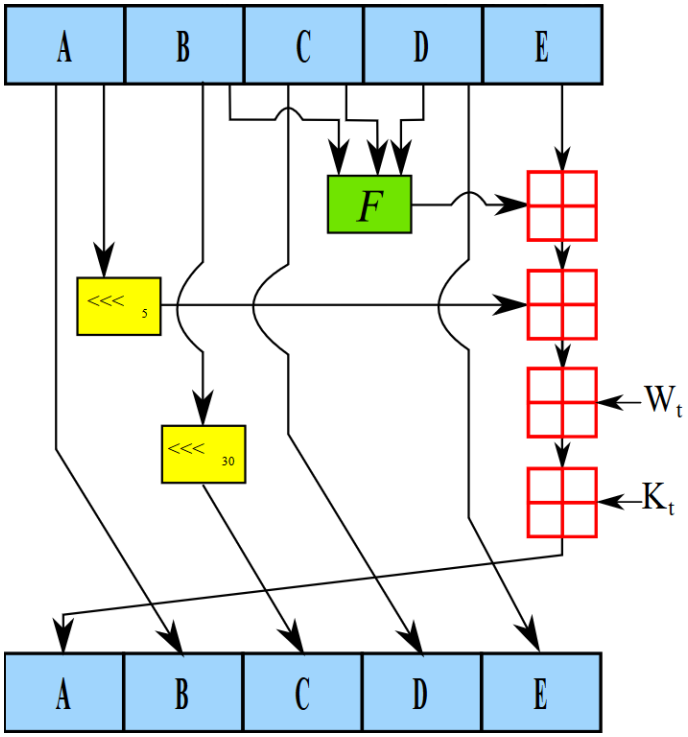
Cílem semestrální práce je implementovat elektronický podpis s užitím asymetrické šifrovací funkce RSA a kryptografické hash funkce SHA-1.

# Kapitola 2

# Analýza problému

## 2.1 SHA-1

SHA-1 je kryptografická hashovací funkce, která ze vstupních dat s maximální délkou $2^{64} - 1$ bitů vytváří 160 bitů dlouhý otisk. Schéma popisující její funkci viz 6.1. Tvorba otisku probíhá ve 4 rundách, v každé pak 20 kroků $\rightarrow$ 80 iterací. Výpočet funkce F pracuje podle aktuální rundy, viz obrázek 2.2.




Obrázek 2.1: Hashovací funkce SHA-1
(zdroj: wikipedia.org)

$$F_1(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$F_2(B, C, D) = B \oplus C \oplus D$$
$$F_3(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$$
$$F_4(B, C, D) = B \oplus C \oplus D$$

Obrázek 2.2: Funkce $F$ během jednotlivých rund

## 2.2 RSA

RSA je asymetrická šifra (s veřejným a soukromým klíčem), která je vhodná jak pro šifrování, tak pro podepisování elektronických dokumentů. Bezpečnost RSA je založena na předpokladu, že rozložení velkého čísla na součin prvočísel je velmi obtížná úloha. Čísla $p$ a $q$ je tedy prakticky nemožné zjistit v přijatelném čase.

### 2.2.1 Výpočet klíčů

1. Zvolí se dvě různá velká prvočísla $p$ a $q$.
2. Vypočte se veřejný modul $n = pq$.
3. Získá se hodnota Eulerovy funkce $\varphi(x) = (p-1)(q-1)$.
4. Zvolí se veřejný exponent $e$ tak, aby $e < \varphi(x)$ a bylo s $\varphi(x)$ nesoudělné.
5. Nalezení soukromého exponentu $d$ tak, aby $de \equiv 1 \ (mod\ \varphi(x))$ — pomocí rozšířeného Euclidova algoritmu.
6. Soukromý exponent $d$ a modul $n$ tvoří privátní klíč. Veřejný klíč pak obsahuje čísla $e$ a modul $n$.

### 2.2.2 Šifrování a dešifrování

Po vygenerování klíčů se otevřený text $P$ šifruje následujícím způsobem:

1. Zpráva se rozdělí na bloky $P_i$ kratší než $n$.
2. $C_i = P_i^e \ mod\ n$

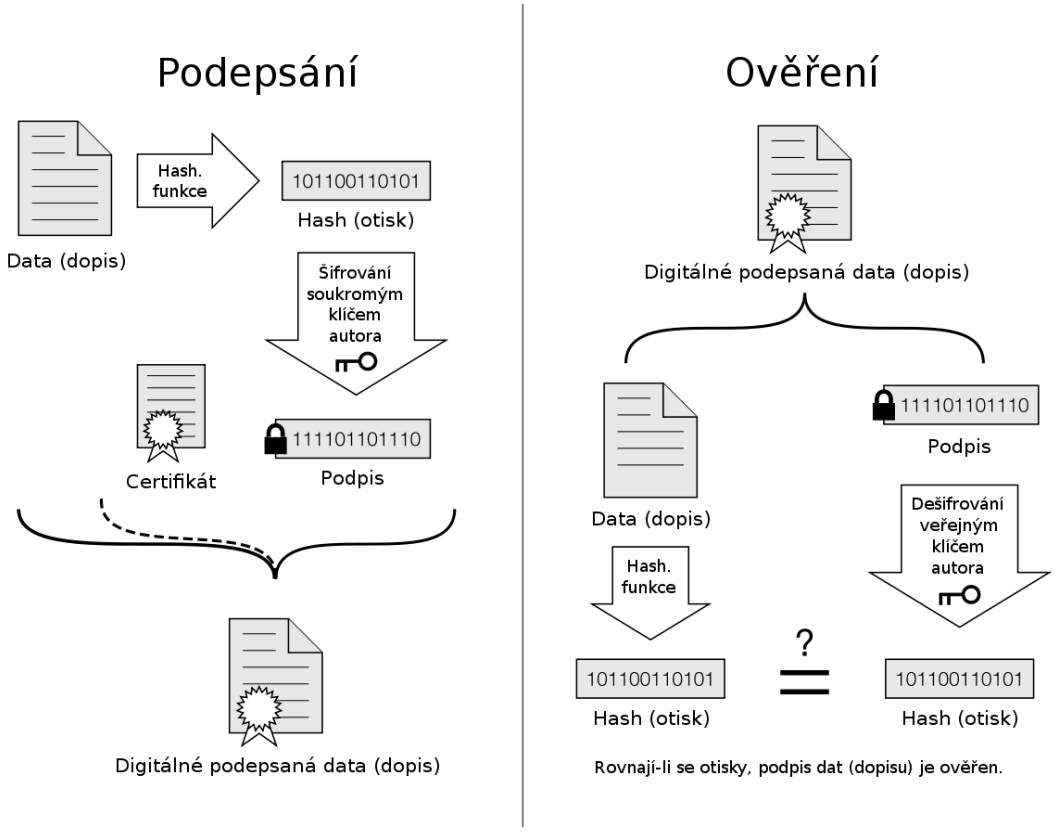
Dešifrování zprávy pak:

1. $P_i = C_i^d \ mod\ n$

## 2.3 Elektronický podpis

Elektronický podpis je aplikací asymetrické kryptografie (tj. šifrování s veřejným a soukromým klíčem). Nejprve se vypočte hash dokumentu (v tomto případě SHA-1). Výsledný hash je poté zašifrován autorovým privátním klíčem, čímž vznikne elektronický podpis.

Při ověření podpisu je nejprve znovu vypočten hash přijatého dokumentu. Poté je pomocí veřejného klíče autora dešifrován obsah elektronického podpisu a výsledek je porovnán s výpočtem hashe zprávy. Pokud jsou oba otisky shodné, je podpis z matematického hlediska platný. V tuto chvíli však není možné považovat platný elektronický podpis za důvěryhodný, protože není jisté, kdo je majitelem veřejného klíče, pomocí kterého došlo k matematickému ověření podpisu. Úplné schéma elektronického podpisu viz obr. 2.3.

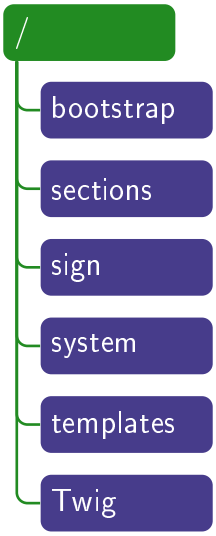



Obrázek 2.3: Grafické znázornění tvorby a ověření elektronického podpisu (zdroj: wikipedia.org)

# Kapitola 3

# Návrh řešení

Elektronický podpis byl implementován do webové aplikace, která simuluje vygenerování RSA klíčů, podepsání zprávy, její přenos a následnou kontrolu. Souborová struktura aplikace je ukázána na obrázku 3.1.




Obrázek 3.1: Souborová struktura webové aplikace

Složky ***boostrap*** a ***Twig*** obsahují nástroje třetích stran, které umožňují snazší grafické navržení aplikace. Složka ***templates*** pak obsahuje možné statické šablony, do kterých pak nástroj *Twig* umístí dynamický obsah webové stránky.

Adresář ***sections*** obsahuje kontrolery jednotlivých sekcí webu. Složka ***system*** obsahuje konfigurační soubor, podle kterého je řízen přístup do sekcí. Jsou zde také uvedené všechny řídící konstanty a třída pro tvorbu hlášení uživatelům.

Implementace SHA-1 a RSA je uložena v adresáři ***sign***. Pro všechny výpočty je použita knihovna GMP, která dokáže provádět aritmetické operace s čísly neomezené délky.

# Kapitola 4

# Popis řešení

## 4.1 Generování klíčů pro RSA

Generování provádí třída `RSAGenerator`. Konstruktor této třídy neočekává žádný parametr, protože je třída navržená tak, aby nejprve zkontrolovala existenci vygenerovaných hodnot v `$_SESSION` a podle toho následně vyčkávala na zadání délky čísel $p$, $q$ a $e$ skrze veřejné gettry. Pokud jsou hodnoty vyplněné, volání metody `generate()` provede samotné vygenerování klíčů, které jsou pro další použití uloženy do `$_SESSION`.

### 4.1.1 Generování prvočísel

Program nejprve náhodně vygeneruje $m$ 1 a 0. Takto vygenerované binární číslo se poté předá funkci `gmp_nextprime`, která vrátí nejmenší další prvočíslo. Funkce `gmp_nextprime` k číslu pouze přičítá jedničku a pomocí Miller-Rabinova testu vyhodnocuje, jestli je získané číslo prvočíslem. Pokud ne, číslou pouze opět inkrementuje a cyklus se opakuje.

## 4.2 Generování SHA-1

Výpočet otisku SHA-1 pro elektronický podpis zprostředkovává třída `SHA1Generator`, která ve svém konstruktoru neočekává žádný parametr, a samotné generování se provádí metodou:

```
instanceObjektu->generate(text);
```

Konkrétní výpočet algoritmu SHA-1 byl sestaven podle pseudokódu na webových stránkách *https://en.wikipedia.org/wiki/SHA-1*.

Generování bylo testováno porovnáním s výsledkem funkce `sha1` z balíku PHP, viz obr. 6.1.

**Varování!** Pro správnou funkčnost třídy je nutné, aby byl interpret jazyka PHP nastaven na 32 bitové integery. Při 64 bitových integerech třída generuje otisky dlouhé 80 znaků.

## 4.3 Šifrování textu

Šifrování textu pomocí RSA zabezpečuje třída `RSACryptor`. Ta ve svém konstruktoru přijme patřičný klíč a veřejný modul $n$. Pomocí metody `encrypt` je pak možné šifrovat zadaný text. Tato metoda vrací zašifrovaný text jako číslo. Šifrování textu ukazuje obrázek 4.1.

### 4.3.1 Převod řetězce znaků na číslo

Jednotlivé znaky vstupního řetězce se pomocí UTF-8 kódování převádějí na jejich binární reprezentaci. Tyto osmice bitů jsou poté konkatenovány do řetězce binárních čísel, který je nakonec, pomocí knihovny GMP, interpretován jako číslo.

## 4.4 Dešifrování textu

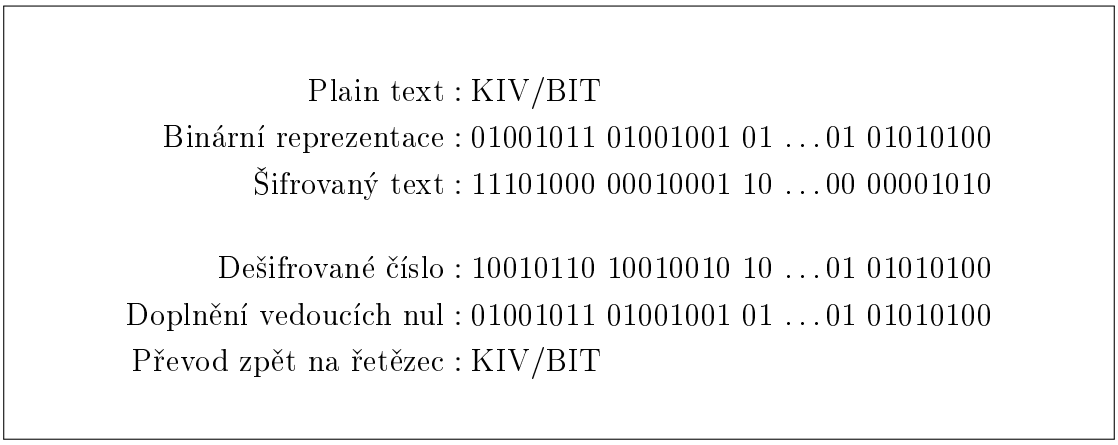
O dešifrování textu se stará třída `RSADecryptor`. Konstruktor této třídy očekává daný exponent a veřejný modul $n$. Medoda `decrypt` vyžaduje šifrovaný text ve formě čísla, se kterým provede patřičný výpočet a výsledek převede zpět na řetězec znaků. Dešifrování je ukázáno na obrázku 4.1.

### 4.4.1 Převod čísla na řetězec znaků

Nejprve se dané dekadické číslo převede na jeho binární podobu. Číslo se zleva doplní nulami tak, aby počet jeho cifer byl dělitelný 8, protože vedoucí nuly byly při převodu binárního řetězce na číslo ořezány. Získaný binární řetězec je pak po jednotlivých osmicích převáděn zpět na UTF-8 znaky.

## 4.5 Zasílání podepsaných zpráv

Uchování přenášené zprávy zabezpečuje třída `Message`. Ta kvůli věrné simulaci elektronického podpisu zahrnuje veškeré náležitosti podepsané zprávy. Dále obsahuje atributy, ve kterých je zapsána zpráva před upravením a po upravení, aby bylo možné ji po Oskarově zásahu zobrazit v sekci „Bob“.




Obrázek 4.1: Šifrování a dešifrování zadaného textu.

# Kapitola 5

# Uživatelská dokumentace

## 5.1 Instalace webového systému

Pro instalaci stačí nakopírovat zdrojové soubory do adresáře přístupného pomocí webového serveru. Na tomto webovém serveru musí být spuštěna podpora PHP 5 >= 5.6.0, povolená webová session a nainstalována knihovna pro práci s čísly libovolné délky — GMP. Pokud tato knihovna na serveru chybí, můžete ji doinstalovat pomocí terminálu příkazem:

```
apt-get install php5-gmp
```

Pro správné fungování aplikace je také nutné nastavit v aplikaci její novou adresu, aby aplikace věděla, kam odkazovat při událostech. To provedete nastavením konstatny `SELF_HTTP_ADDR` v konfiguračním souboru `/system/config.php`.

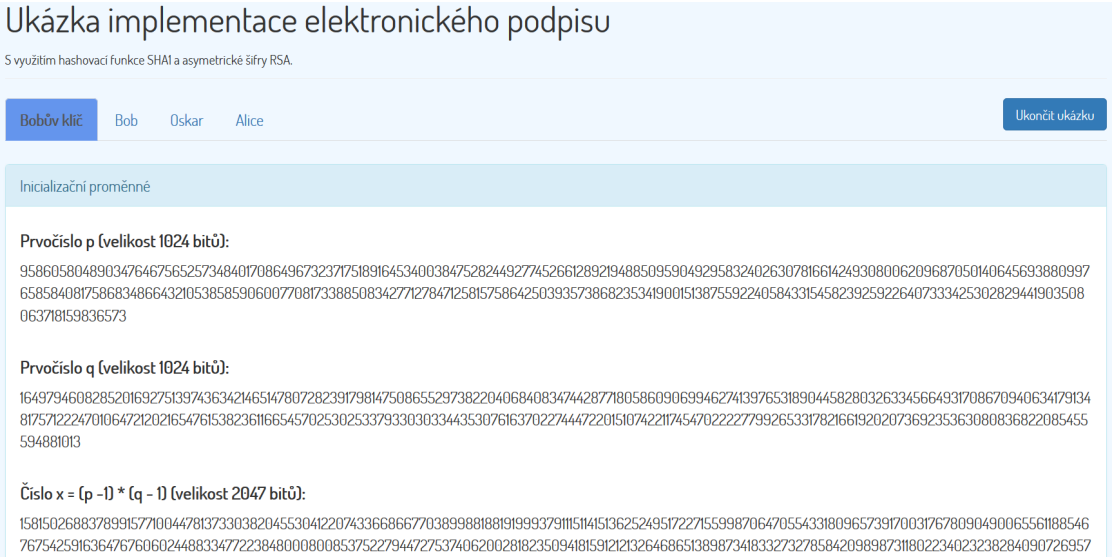
```
define('SELF_HTTP_ADDR', 'URL adresa aplikace');
```

## 5.2 Ovládání aplikace

1. Při spuštění je uživatel požádán o zadání délek inicializačních prvočísel $p$ a $q$ (systém je testován s délkou prvočísel 2048 bitů, u generování delších čísel je nutná trpělivost). Dále systém vyžaduje zadat délku veřejného klíče.
2. Vygenerované klíče je možné zhlédnout v sekci „Bobův klíč“.
3. Ukázka pokračuje v sekci „Bob“, kde je možné poslat Alici textovou zprávu.
4. Po podepsání a odeslání zprávy pokračuje ukázka v sekci „Oskar“. Ta simuluje útočníka, který zprávu odchytil a může změnit její obsah.
5. Poslední sekcí je „Alice“, která přijala zprávu od Boba a podle přiložené elektronického podpisu může ověřit, že zprávu skutečně psal. Po přečtení

zprávy je zpráva smazána a uživatel je odkázán do sekce „Bob“, kde může dojít k odeslání další zprávy.

6. V pravém horním rohu obrazovky je tlačítko „Ukončit ukázku“. Klepnutím na toto tlačítko se uvolní vynegerované klíče a uživatel je přesměrován na jejich opětovné vygenerování (bod 1).




Obrázek 5.1: Sekce „Bobův klíč“ ve webové aplikaci

# Kapitola 6

# Závěr

Vytvořená aplikace je použitím třídy `RSAGenerator` schopná generovat náhodné klíče pro RSA algoritmus podle zadané délky čísel $p$, $q$ a $e$ (veřejný exponent). Pomocí vygenerovaných klíčů jsou třídy `RSACryptor` a `RSADecryptor` schopné šifrovat a dešifrovat zadaný text. Tyto třídy získávají klíče skrze parametry konstruktoru, takže lze aplikaci využít jak pro šifrování, tak pro elektronický podpis.

Byla rovněž implementována třída `SHA1Generator`, s jejíž pomocí je možné ze zadaného textu počítat otisky typu *SHA-1*. Výsledek výpočtu byl kontrolován s funkcí `sha1`, která je dostupná v balíku *PHP 4 >= 4.3.0* (viz obr. 6.1).

**Vstupní řetězec** : $KIV/BIT$
*SHA1Generator* : $6a79c6779857547af3da2bef02633b9f781a1eba$
*PHP sha1* : $6a79c6779857547af3da2bef02633b9f781a1eba$

Obrázek 6.1: Výsledky `SHA1Generator.generate()` a `sha1` z balíku PHP