FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Zápočtová úloha z předmětu KIV/BIT

RSA (autentizace a šifrování)

25. dubna 2016

Lukáš Černý

1 Úvod

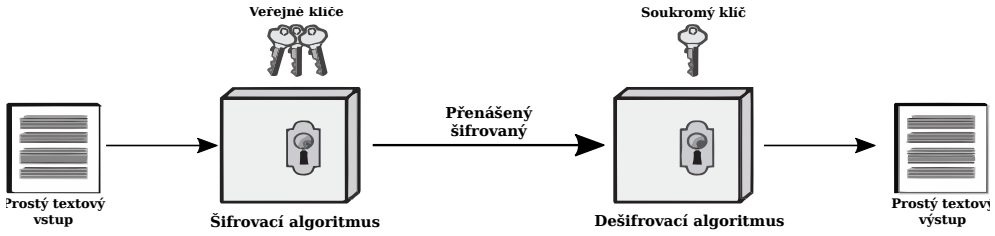
Zadání mé semestrální práce je zaměřeno na implementaci asynchronní šifry RSA. Šifru využívm pro autentizaci počítače (klienta) se serverem. Program slouží jako simulace autentizace, kde jsou vytvořeny dva klienti a jeden server. Pomocí vygenerovaného páru klíčů a následné distribuci veřejného klíče je možné provést autentizaci.

2 Teorie

Algoritmus RSA byl publikovaný v roce 1978. Je vhodný pro šifrování i podepisování dokumentů. Při zvolení dostatečně dlouhého klíče se jedná o bezpečnou šifru, je tedy velice rozšířená. Jeden z příkladů použití je například autentizace přes protokol SSH. RSA patří mezi asynchronní šifry. Rozšíl mezi synchronní a asynchronní je ten, že používá veřejný a soukromý klíč pro šifrování a dešifrování, zatímco u synchronní je pouze jeden klíč, kterým se šifruje i dešifruje.

2.1 Proces šifrování a dešifrování

Při procesu šifrování a dešifrování se používá pár klíčů (soukromý a veřejný). Jak je z názvu patrné, tak veřejný klíč je k dispozici veřejně a pomocí něho je možné zprávu zašifrovat. Soukromý klíč vlastní pouze strana, která si zašifrovanou zprávu může přečíst (viz obr. 1). Nevýhoda tohoto řešení je, že pokud budeme chtít jednu zprávu odeslat více lidem, budeme muset tolikrát, kolik je lidí.



Obrázek 1: Model asymetrického šifrování

Veřejný klíč a soukromí klíč se vytvoří následujícími vzorci:

$$n = p * q$$
$$x = (p-1) * (q-1)$$
$$e \Rightarrow e < x \wedge (e \ \ gcd \ \ x) < 1$$
$$d = e^{-1} (mod\, x)$$

Popis vzorců: Číslo **n** získáme vynásobením dvou náhodně vygenerovaných velkých prvočísel **p** a **q**. Odečtení jedničky od **p** a **q** a následný součinem získáme **x**. Číslo **x** je nutné mít pro vytvoření čísla **e**, které musí být menší než **x** a zároveň jsou spolu nesoudělné. Číslo **d** poté získáme tak, aby platilo **d * e * mod x = 1**. Veřejný klíč se skládá z čísel **e** a **n** a soukromí klíč je číslo **d**.

Šifrování:

$$C = P^e mod\, n$$

Dešifrování:

$$P = C^d mod\, n$$

(C = šifrovaný text, P = nešifrovaný text)

3 Možnosti řešení

Proces autentizace s využitím šifrovacího algoritmu RSA se využívá například při přihlášení na linuxový server (počítač) přes sít. Vytvořit simulaci takového rešení jde řešit dvěma způsoby.

Autentizace přes sít

Tento způsob více odpovídá reálnému využití. Uživatel, který se bude chtít autentizovat vůči jinému uživateli, si musí vygenerovat pár klíčů a veřejný klíč distribuovat ostatním uživatelům. Právě distribuce klíčů může být v tomto návrhu překážkou. V procesu autentizace se již předpokládá, že potřebný veřejný klíč je k dispozici. Řešení by mohlo být vytvoření souboru se známými hosty, který by se při spuštění aplikace načetl. Následná výměna šifrované zprávy by již proběhla po síti.

Autentizace v jedné aplikaci

Simulace v rámci jedné aplikace je jednodušší na vytvoření. Jednodušší je to právě v tom, že se vše děje v paměti počítače a žádná data nejdou přes sít (lépe se ladí). V předchozím způsobu byla těžší distribuce klíčů, v tomto případě lze distribuovat klíče jednoduše, protože vidíme všechny uživatele a jednoduše si můžeme vybrat, komu veřejný klíč přiřadíme. Následný proces autentizace je stejný jako přes sít.

4 Navržené řešení

Simulaci autentizace jsem vytvořil jako jednu aplikaci, kde jsou celkem tři počítače - dva klienti a jeden server. Každý počítač má umožněno generování páru klíčů a obsahuje seznam veřejných klíčů. Ovšem pouze klienti se mohou připojovat. Tímto způsobem jsem chtěl naznačit, že pouze klienti se mohou autentizovat vůči stroji.

5 Implementace programu

Program je postaven na buildovacím nástroji Maven od Apache. Programovací jazyk jsem zvolil Javu ve verzi 8 s grafickou knihovnou JavaFX. Logika programu je rozdělena na třívrstvou architekturu. JavaFX umožňuje popisovat grafiku pomocí *fxml* souborů, takže každé okno má vlastní soubor a k tomu příslušný *controller*.

Pro uložení klíčů jsem využil třídu BigInteger z balíku java.Math. BigInteger umožňuje uložit hodnotu libovolné velikosti, jelikož se nejedná o primitivní datový typ, ale o obalovou třídu využívající dynamické alokace paměti. Třída obsahuje metody usnadňující práci s instancí objektu.

Generování páru klíčů

```
SecureRandom r = new SecureRandom();
p = BigInteger.probablePrime(512, r);
q = BigInteger.probablePrime(512, r);
n = p.multiply(q);

x = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

e = BigInteger.probablePrime(256, r);
while (x.gcd(e). intValue() > 1)
        e = BigInteger.probablePrime(256, r);

d = e.modInverse(x);
```

Jak je vidět výše na ukázce kódu, využil jsem statickou metodu *probablePrime*. Metoda vrací prvočíslo, podle požadavků zadané parametry. Prvním parametrem je velikost generovaného čísla a druhý parametr je instance třídy *SecureRandom*. Třída *SecureRandom* se využívá pro bezpečné generování čísel. Pro číslo **p** a **q** jsem zvolil velikost čísla o velikosti 512 bitů. Tato velikost je dostatečně veliká, aby čísla byla neodhadnutelná. Další ulehčení, která nabízí BigInteger je instanční metoda *modInverse*, která umožňuje provést výpočet $e^{-1}\,(mod\,x)$.

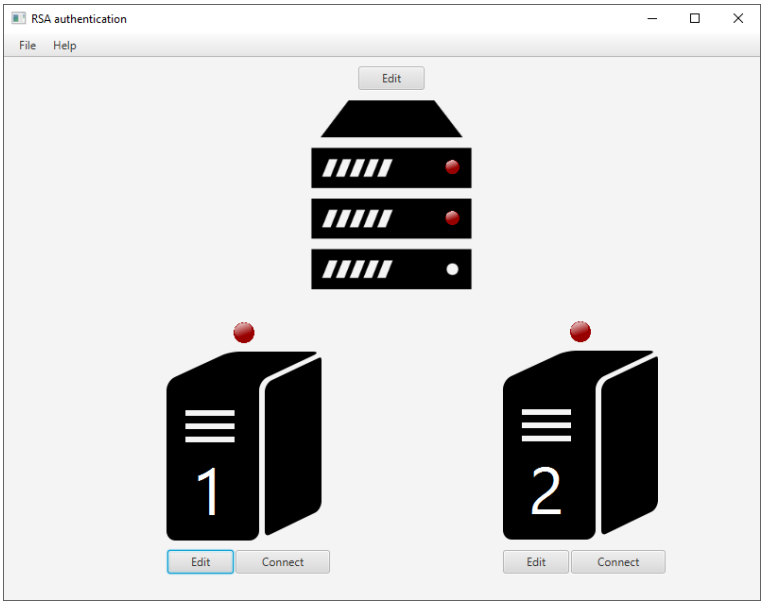
Struktura programu

- **Main** - spouští aplikaci
- **AppController** - inicializuje počítače a provádí autentizaci
- **KeyMachine** - generuje pár klíčů
- **RSA** - provádí šifrování a dešifrování
- **Key** - rozhraní pro klíče
- **PrivateKey** - vytváří veřejný klíč
- **PublicKey** - vytváří soukromý klíč
- **Client** - uchovává informace o počítači (klientu)
- **LocateString** - převádí klíč na lokalizovaný řetězec
- **RootController** - kontroler pro hlavní obrazovku
- **EditClientController** - kontroler pro úpravu počítače
- **AddPublicKeyController** - kontroler pro přidání veřejného klíče
- **AboutController** - kontroler pro okno o aplikaci
- **FXMLTemplatex** - načítá jednotlivé šablony grafiky

6 Obsluha programu

Projekt je ke stažení na GitHubu [1]. Před překladem projektu se ujistěte, že máte na počítači nainstalovaný buildovací nástroj maven, Javu 8 a JavaFX (Java od spolčenosti Oracle obsahuje i JavaFX). Poté spustěte script s názvem *run.sh* a projekt se přeloží a spustí.

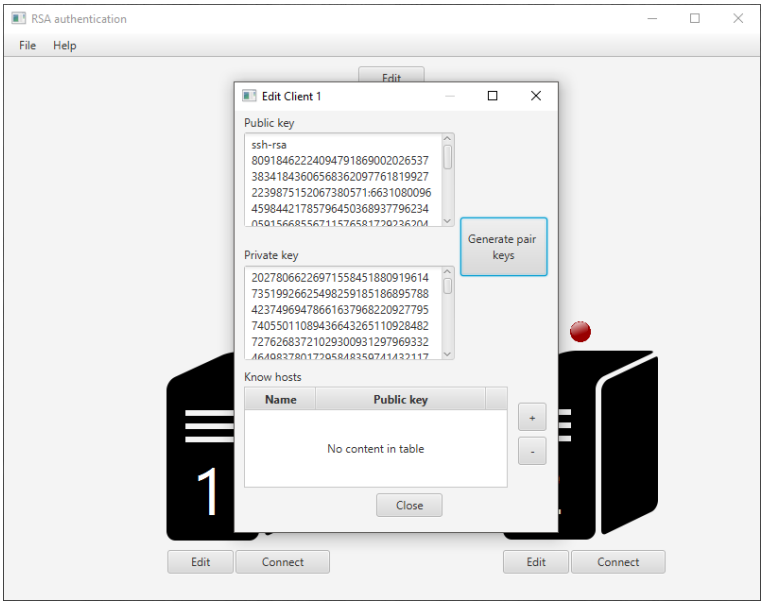
Po spuštění se objeví hlavní okno aplikaci (viz obr. 2). Hlavní okno obsahuje dva počítačové klienty a jeden server. Klienti mají možnost připojit se na server.



Obrázek 2: Hlavní okno aplikace.

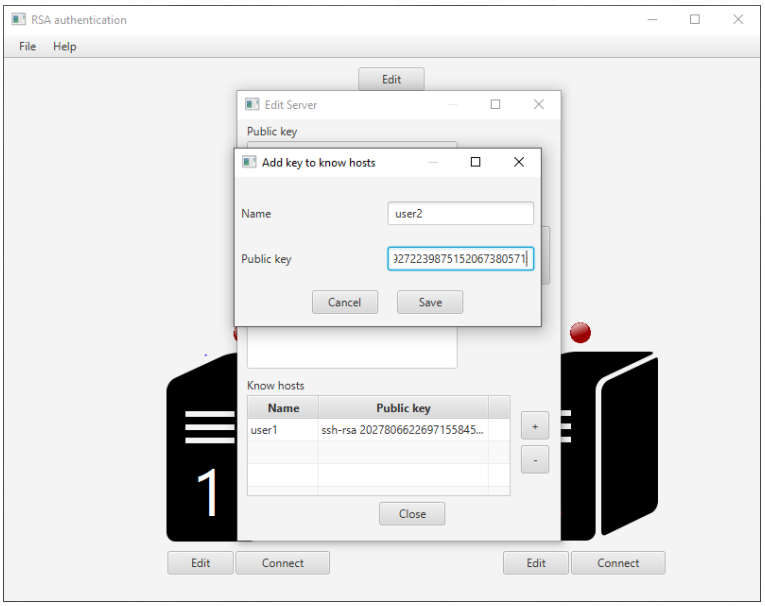
[1] https://github.com/vrenclouff/rsa_auth

Pro připojení klienta na server je potřeba mít sadu klíčů. Po kliknutí na tlačítko *Edit* u klienta se objeví okno (viz obr. 3), kde je možnost vygenerovat pár klíčů. Veřejný klíč nyní celý označte a zkopírujeme si ho do schránky.



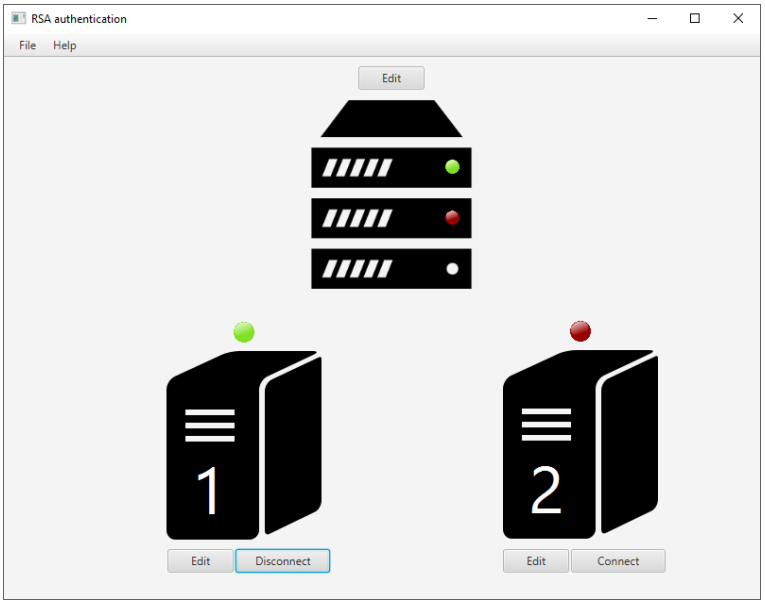
Obrázek 3: Možnosti úpravy uživatele.

Zkopírovaný klíč nyní dostaneme mezi známé hosty na serveru. Otevřete úpravu serveru a objeví se stejné okno jako při editaci klienta. V dolní části okna se nacházejí tlačítka + a -, pomocí nichž provedeme přidání klienta mezi známé hosty (viz obr. 4).



Obrázek 4: Přidání uživatele mezi známé uživatele na serveru.

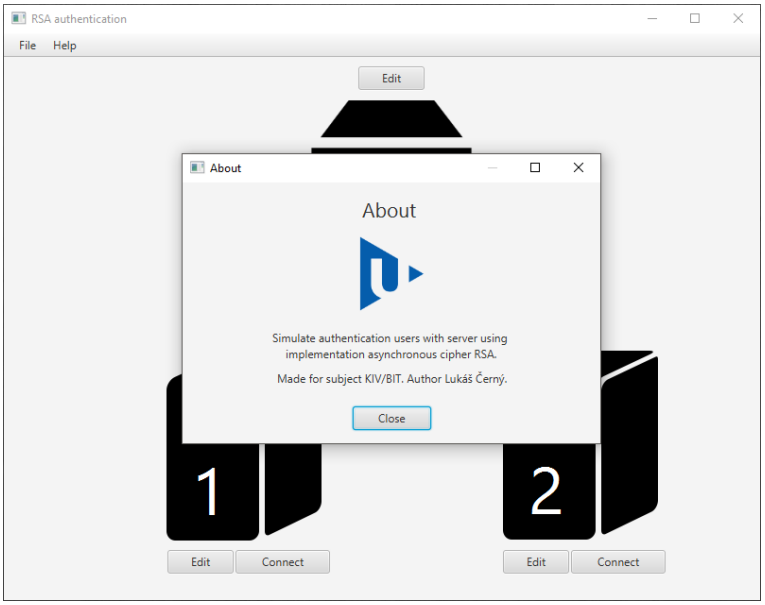
Po stisku tlačítka *Připojit* se pokusíme o přihlášení na server. Pokud autentizace proběhla v pořádku, označí se zelenou barvou stav připojení (viz obr. 5).



Obrázek 5: Připojení uživatele k serveru.

Další možnosti se skvývají v horní nabídce aplikace. Nacházejí se zde dvě volby:

- **Soubor**
  - Restartovat - vymaže nastavení
  - Ukončít - ukončí aplikaci
- **Nápověda**
  - O Aplikaci - zobrazí okno s informacemi o aplikaci (viz obr. 6)



Obrázek 6: Informace o aplikaci

7 Závěr

Aplikace je plně funkční a splňuje všechny zadané pořadavky. Z pohledu výkonu je nejvíce náročné generování páru klíčů, jelikož se generují prvočísla velké velikosti. Jedná se ale o proces, který uživatel nedělá příliš často a neovlivňuje se tím běh aplikace. Samotný běh aplikace je plynulý. Další rozšířitelnost je možná, do menu lze přídat další položky a dopsat jejich funkčnost. Podle mého názoru již není další funkčnost nutná.