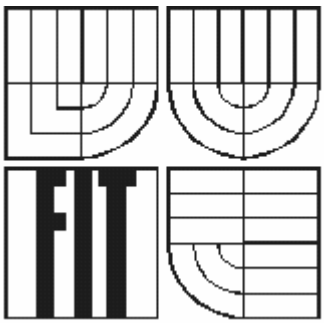
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Knihovna pro podporu návrhu bezpečnostních protokolů

Ročníkový projekt

Karel Tomášek **2006**

© Karel Tomášek

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením ing. Pavla Očenáška a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal. Dále tímto dávám Fakultě informačních technologií Vysokého učení technického v Brně svolení k použití a úpravám textu i jeho částí pro vědecké a výukové účely při zachování autorských práv.

Podpis autora .....

2. května 2006

Tímto bych chtěl poděkovat svému vedoucímu práce za cenné rady a připomínky při řešení tohoto projektu.

Abstrakt: Bezpečnostní protokoly se používají pro zabezpečenou komunikaci přes nezabezpečenou síť. Tento dokument má za účel letmé seznámení s formálním popisem bezpečnostních protokolů a jednoduchým popisem několika protokolů. Také má seznámit s knihovnou funkcí, která umožňuje simulaci protokolu.

Klíčová slova: bezpečnostní protokol, komunikace, autentizace, symetrické a asymetrické šifrování.

Abstrakct: Security protocols are used for security communication over insecure network. This document gives a brief introduction into description and use of security protocols and proposes a library of functions that could be used for simulation and verification of security protocols.

Keywords: security protocol, communication, authentication, symmetric and asymmetric cryptography.

Obsah

1 Úvod

V začátcích vytváření počítačových sítí se na bezpečnost přenášených dat moc nehledělo. Se vzrůstajícím počtem uživatelů sítě se objevil problém, jak bezpečně přenášet po sítí data. Z tohoto důvodu se začaly vyvíjet bezpečnostní protokoly. Bezpečnostní protokoly, které jsou používány v distribuovaných systémech pro autentizací a související úkoly, jsou při návrhu náchylné k různým chybám. Proto je nutné navrhované protokoly formálně verifikovat před jejich implementací. Z tohoto důvodu také vznikla knihovna funkcí pro podporu návrhu bezpečnostních protokolů.

2 Základní pojmy

AUTENTIZACE

Autentizace je proces, při kterém účastník nějakého distribuovaného systému prokazuje svoji identitu. Pokud je v komunikačním protokolu zařazen kvalitní autentizační mechanismus, je pro útočníka velmi obtížné podvrhnout zprávu s falešným původem.

AUTENTIZAČNÍ PROTOKOL

Soubor pravidel, podle kterých účastníci komunikace zasílají zprávy tak, aby prokázali svoji identitu.

DŮVĚRNOST

Vzájemně komunikující subjekty mohou požadovat, aby jejich komunikace nemohla být odposlouchávána žádnou jinou neoprávněnou stranou. Nejčastějším prostředkem pro dosažení důvěrnosti je kryptografie.

KRYPTOGRAFIE

Jedná se o takové zabezpečení komunikace, kdy vnější útočník nemůže bez znalosti klíčů získat originální zprávy. Kryptografie využívá dvou symetrických procesů: šifrování a dešifrování. Šifrování lze rozdělit podle typu použitých algoritmů a klíčů na symetrické a asymetrické.

Asymetrické šifrování je založeno na použití dvou párových klíčů. Každý subjekt má svůj veřejný a privátní klíč. V tomto případě je možné nejen zprávy šifrovat, ale také podepisovat: odesílatel data nejdříve dešifruje svým soukromým klíčem a výsledek přidá ke zprávě. Všichni příjemci, kteří mají veřejný klíč odesílatele mohou podpis „zašifrovat“. Pokud získají doručenou zprávu, mají jistotu o jejím původci, protože nikdo kromě majitele veřejného klíče nemohl vytvořit odpovídající šifru.

Asymetrická kryptografie má oproti klasické, symetrické velkou výhodu v tom, že není nutné domlouvat tajný sdílený klíč před zahájením utajené komunikace. Na druhou stranu však hrozí podvržení veřejného klíče (lze volně šířit veřejné klíče s nepravou identitou). Tomu zamezují certifikační autority, které potvrzují pravost veřejných klíčů komunikujících subjektů.

INTEGRITA

Jedná se o nejčastěji požadovanou vlastnost při přenosu dat. Přenášená data nesmí být změněna žádnou neautorizovanou stranou, nesmí být ani uměle zadržována, či opakovaně vysílána po neoprávněném odposlechu.

NONCE

Anglický termín „nonce“ je používán přímo ve formální specifikaci bezpečnostních protokolů. Jedná se o náhodně vygenerovaná čísla. Někdy bývají rozlišována jako odhadnutelná (sekvenčně generovaná čísla, časové razítko) nebo neodhadnutelná (např. náhodné 40-bitové číslo).

RELAČNÍ KLÍČ

Tímto pojmem se rozumí klíč jedné komunikační relace), který byl ustaven na začátku komunikace za účelem jejího zabezpečení.

PROTOKOL

Protokol je množina pravidel a konvencí, která definuje komunikaci mezi dvěma a více agenty. Tito agenti, známí také jako "principals", mohou být koncoví uživatelé, procesy nebo výpočetní systémy. V kryptografických protokolech je alespoň jedna část zašifrována.

3 Bezpečnostní protokoly

3.1 Protokoly pro bezpečnou komunikaci

3.1.1 Úvod

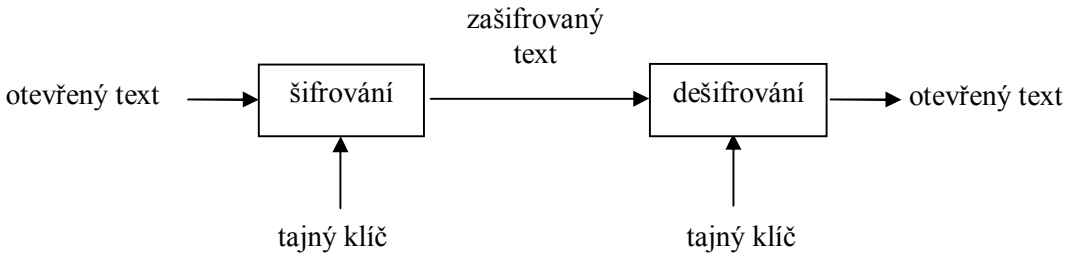
V dnešní době je potřeba při komunikaci po nezabezpečené síti přenášet i citlivá data. V počátcích vytváření počítačových sítí se na bezpečnost přenosu informací moc nehledělo. Zpočátku byla taková síť většinou akademická nebo vojenská, kde se předpokládala jakási kolegialita. Data přenášená touto sítí byla snadno dostupná a tak nějaké zabezpečení přenosu bylo nepotřebné. K autentizaci v této síti stačilo zadat heslo, které se ovšem sítí posílalo nezabezpečené. Postupem času se z této sítě vyvinul Internet. Objevila se potřeba posílat nezabezpečenou sítí i citlivá data. Bezpečnostní protokoly byly teprve ve vývoji a jejich formální modely byly pro praxi nepoužitelné. K implementaci a nasazení těchto protokolů došlo až později.

Dnes jsou již počítačové sítě součástí běžného života společnosti. Požadavky na zabezpečení však stále rostou. Neustále se vyvíjejí nové bezpečné protokoly. Aby byly protokoly opravdu bezpečné, tak je součástí jejich návrhu i formální verifikace. Je to vlastně analýza protokolu po formální stránce, kdy se zjišťuje, zda je možné protokol prolomit / napadnout.

3.1.2 Symetrické bezpečnostní protokoly

Symetrické bezpečnostní protokoly používají pro zabezpečení přenášené informace symetrické šifrování. Mezi nejpoužívanější algoritmy symetrického šifrování patří algoritmus DES a jeho klony, algoritmus IDEA, AES, RC2 a RC5.

Symetrický algoritmus



Obr. 1: Postup šifrování textu symetrickým algoritmem

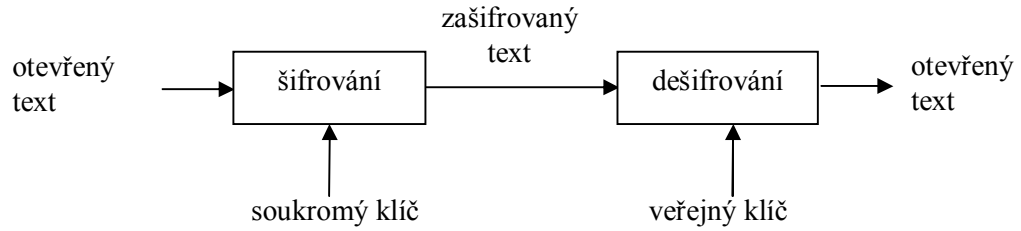
Pro zabezpečenou komunikaci symetrickým algoritmem máme ovšem jeden malý problém. Pro každého uživatele sítě, se kterým chceme bezpečně komunikovat, je potřeba klíč k šifrování. Pro komunikaci např. 100 účastníků ale potřebujeme 10 000 různých klíčů a to je neúnosné. Proto vznikla tzv. symetrická správa klíčů. V tomto případě má každý uživatel jenom jeden symetrický klíč a to klíč pro komunikaci se serverem, který podle jeho typu vygeneruje nebo předá klíč relace, se kterým se bude komunikace šifrovat.

Centralizovaná správa klíčů má dva módy. Key Translation Center (KTC) je server, který klíč jen předává. Oproti tomu Key Distribution Center (KDC) klíč na žádost sám generuje.

3.1.3 Asymetrické bezpečnostní protokoly

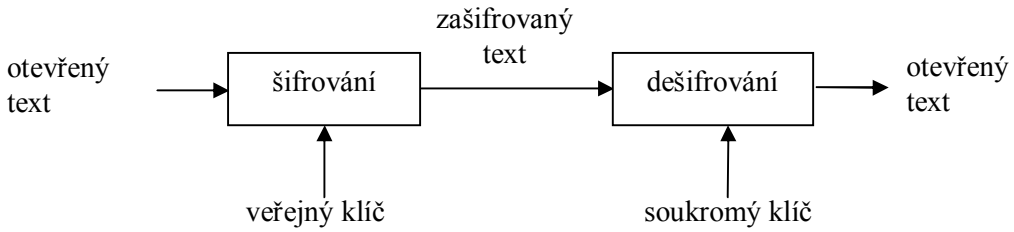
Asymetrické bezpečnostní protokoly používají pro bezpečný přenos asymetrickou kryptografii. Mezi asymetrické algoritmy patří RSA, DSS, nebo Knapsack. Pro šifrování a dešifrování potřebujeme dva klíče, tzv. soukromý a veřejný klíč. Soukromý klíč reprezentuje identitu vlastníka, veřejný klíč je volně k dispozici všem účastníkům komunikace.

Asymetrický algoritmus



Obr. 2: Elektronický podpis

Tento postup zaručí autentizaci odesilatele, ale ne bezpečnost. Zprávu může kdokoli dešifrovat. Tomuto algoritmu se říká elektronický podpis.



Obr. 3: Asymetrické šifrování textu

Tento postup šifrování umožňuje bezpečně přenést informace, ale není možné zjistit, kdo zašifrovanou zprávu poslal.

Pokud požadujeme od algoritmu bezpečný přenos i zachování autenticity, potom je potřeba oba algoritmy spojit v jeden.

4 Formální popis protokolu

Aby bylo možné o protokolu říct, že je bezpečný, je třeba, aby protokol obstál při pokusu o jeho prolomení. Obecná zabezpečená komunikace probíhá tak, že se zasílají zprávy šifrované dočasnými klíči, které by neměly být známy vnějším pozorovatelům.

4.1 Zpráva

Základem komunikace je zpráva. Při formálním zápisu zpráv bereme v úvahu její následující charakteristiky:

- označení subjektů komunikace *A, B*, … ;
- nonces *Na, Nb*, … ;
- klíče *Ka, Kb, Kab*, … ;
- složená zpráva $\{ X, X' \}$,
- šifrovaná zpráva klíčem K **Crypt** $K\,X$, někdy také označené $\{ X \}K$.

Při použití šifrování metodou veřejného klíče předpokládáme, že K^{-1} je inverzní ke klíči K. Pokud se klíče rovnají, potom se jedná o symetrickou kryptografii. Dále předpokládáme, že zašifrovanou zprávu nemůžeme přečíst bez znalosti příslušných klíčů. [1]

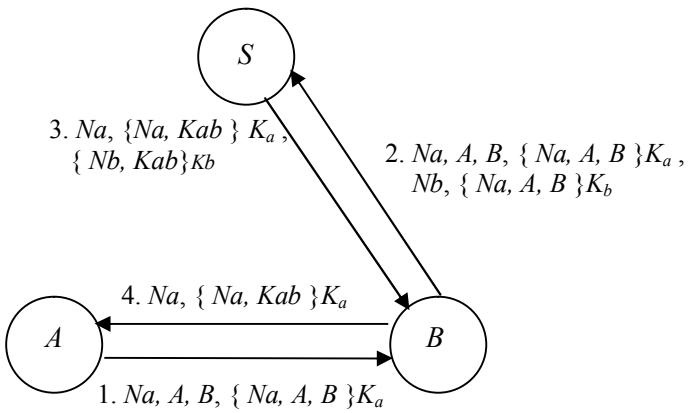
4.2 Příklady protokolů

4.2.1 Otway-Rees protokol

Tento protokol [3] byl navrhnut jako autentizační protokol se sdílenými klíči, který obsahuje dva účastníky a autentizační server. Jedná se tedy o symetrický autentizační protokol. Protokol je atraktivní z důvodu, že obsahuje malé množství zpráv.

Formální zápis:
1. $A \rightarrow B$: *Na, A, B,* $\{ Na, A, B \}_{Ka}$
2. $B \rightarrow S$: *Na, A, B,* $\{ Na, A, B \}_{Ka}$, *Nb,* $\{ Na, A, B \}_{Kb}$
3. $S \rightarrow B$: *Na,* $\{Na, Kab \}_{Ka}$, $\{ Nb, Kab \}_{Kb}$
4. $B \rightarrow A$: *Na,* $\{ Na, Kab \}_{Ka}$

Ka je klíč subjektu *A*, *Kb* je klíč subjektu *B*. $\{ X \}_K$ značí zprávu *X* zašifrovanou klíčem *K*.



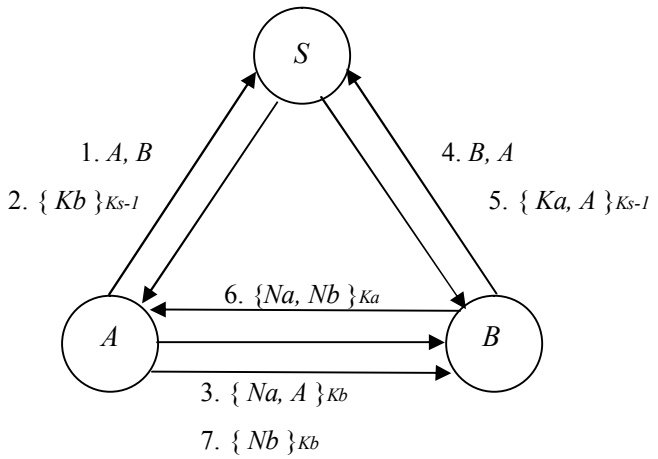
Obr. 4: Neformální popis Otway-Rees protokolu

Popis algoritmu:
1. *A* naváže komunikaci s *B* a vygeneruje nonce *Na* pro identifikaci spojení. Tuto zprávu *A* zašifruje svým klíčem.
2. *B*, protože nemůže dešifrovat část zašifrovanou klíčem *Ka*, přidá na konec zprávu, která obsahuje identifikaci spojení zašifrovanou klíčem *Kb*.
3. Server *S* si zprávy dešifruje, protože zná klíče *Ka* a *Kb*, klíčem *Ka* zašifruje nonce *Na* a klíč relace *Kab*. Totéž udělá pro klíč *Kb*. Tuto zprávu pošle *B*.
4. *B* zprávu přijme, vezme si tu část, kterou může dešifrovat a zbytek posílá *A*. Tak se klíč relace *Kab* dostane k *A*.
   Díky nonce *Na* a *Nb* všichni účastníci ví, že komunikace není zastaralá.

4.2.2 Needham-Schroeder protokol

Mnoho existujících autentizačních protokolů je odvozeno od Needham-Schroeder protokolu [1], který vznikl v roce 1978. Původní protokol byl navržen na autentizaci symetrickými klíči, avšak modifikoval se i na asymetrický autentizační protokol. Je to jednoduchý protokol, na kterém se testuje knihovna.

Formální zápis:
1. $A \rightarrow S$: A, B
2. $S \rightarrow A$: $\{ Kb \}_{Ks-1}$
3. $A \rightarrow B$: $\{ Na, A \}_{Kb}$
4. $B \rightarrow S$: B, A
5. $S \rightarrow B$: $\{ Ka, A \}_{Ks-1}$
6. $B \rightarrow A$: $\{ Na, Nb \}_{Ka}$
7. $A \rightarrow B$: $\{ Nb \}_{Kb}$



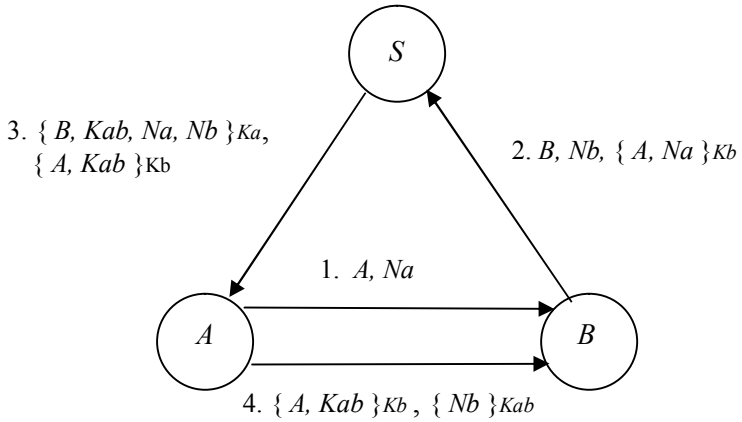
Obr. 5: Neformální popis Needham-Schroeder protokolu

Popis algoritmu
1. *A* posílá S informaci, kdo bude komunikovat
2. *S* zašifruje klíč *Kb* svým veřejným klíčem a posílá tuto zprávu *A*
3. *A* pošle *B* zprávu zašifrovanou jeho klíčem. Obsahem zprávy je nonce *Na*, která funguje jako identifikátor spojení a identifikaci sebe sama.
4. *B* si tuto zprávu přečte a ptá se server *S* na *A*.
5. Server posílá klíč K_A s informací, že klíč patří uživateli *A*
6. *B* posílá *A* zprávu složenou z původního nonce *Na* a přidá si své nonce *Nb*.
7. *A* tuto zprávu dešifruje a zjišťuje, zda nonce *Na* je stejné jako to vygenerované ním a tím si ověří, že zpráva nebyla diskreditována. Pokud tento test proběhne správně, potom *A* posílá nonce *Nb* subjektu *B* zašifrovanou klíčem *Kb*. *B* tuto zprávu přijme, dešifruje a porovná nonce

Nb s tím, co sám vygeneroval. Pokud se shodují, potom každý se subjektů komunikace ví, že klíče pro komunikaci jsou v pořádku.

4.2.3 Yahalom protokol

Trojúhelníková konstrukce umožňuje účastníkům zjistit, zda je druhá strana aktivní. [1]
Formální zápis:
1. $A \rightarrow B : A, Na$
2. $B \rightarrow S : B, Nb, \{ A, Na \}_{Kb}$
3. $S \rightarrow A : \{ B, Kab, Na, Nb \}_{Ka}, \{ A, Kab \}_{Kb}$
4. $A \rightarrow B : \{ A, Kab \}_{Kb}, \{ Nb \}_{Kab}$



Obr. 6: Neformální popis protokolu Yahalom

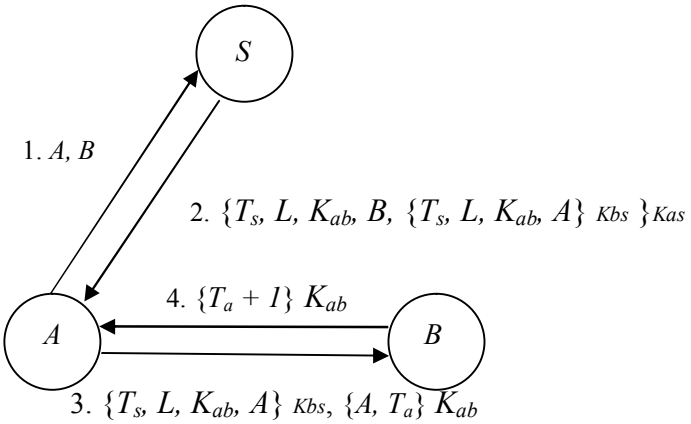
Popis algoritmu

1. *A* informuje *B*, že s ním chce komunikovat. K identifikaci komunikace použije nonce *Na*.
2. *B* posílá zprávu *S*, aby vygeneroval klíč relace *Kab*. Zpráva se skládá z identifikace *B*, z nonce *Nb*, která identifikuje aktuální komunikaci a ze zašifrované zprávy skládající se z identifikace iniciátora komunikace a jeho nonce.
3. Server si vytvoří dvě zprávy. Pro každého účastníka jednu, kterou zašifruje jeho klíčem. Tyto zprávy potom spojí do jediné a pošle *A*
4. *A* si z přijaté zprávy vybere svoji část a zkontroluje si nonce *Na*. Při dešifrování zprávy také získá klíč relace *Kab*. Zbývající část zprávy, kterou obdržel *A* od *S* pošle *B*. K této zprávě ještě přidá zprávu zašifrovanou klíčem relace *Kab* , která se skládá z nonce *Nb*. Tím si *B* ověří, zda je zpráva aktuální.

4.2.4 Kerberos protokol

Kerberos protokol [3] byl vyvinut jako část projektu Athena na MIT a nyní je užíván několika jinými organizacemi. Je založen na protokolu Needham-Schroeder, ale používá časové známky jako nonce, kvůli odstranění problémů z poslední sekce a redukci počtu potřebných zpráv. Tento formální popis je jen jemně zjednodušený oproti skutečně používanému algoritmu.

Formální zápis:
1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{Kbs} \}_{Kas}$
3. $A \rightarrow B : \{T_s, L, K_{ab}, A\}_{Kbs}, \{A, T_a\} K_{ab}$
4. $B \rightarrow A : \{T_a + 1\} K_{ab}$



Obr. 7: Neformální popis protokolu Kerberos

Tady T_s a T_a jsou časové známky a L je doba života.

1. Pošle se zpráva serveru *S* o tom, že subjekty *A* a *B* chtějí komunikovat.
2. *S* pošle nazpět zprávu složenou ze dvou zpráv, první část je určena *A*, druhou část *A* nedokáže dešifrovat a posílá ji v kroku 3 *B*. Tu část, kterou *A* dokáže dešifrovat, obsahuje časovou známku vydanou serverem *S* s její dobou života. Touto zprávou *A* také získá klíč relace.
3. *A* přidá ke zprávě, kterou dostala od *S* svou časovou známku, svou identifikaci a tuto část zašifruje klíčem relace *Kab*.
4. *B* zprávu dešifruje a zjistí si klíč relace *Kab*. Tímto klíčem dešifruje druhou část zprávy a zkontroluje časové známky k ověření platnosti komunikace. Časovou známku *Ta,* zvýšenou o jedna a zašifrovanou klíčem relace *Kab* , posílá zpět *A*, aby si *A* ověřil živost subjektu *B*.

Všechny popsané protokoly je možné nasimulovat v programu, který je součástí ročníkového projektu a slouží k demonstraci navržené knihovny.

5 Popis knihovny pro podporu návrhu bezpečnostních protokolů

5.1 Složení knihovny

Knihovna je složena z hlavičkového souboru *protocol.h* a ze souboru *protocol.cpp*. Knihovna umožňuje vytvořit třídu *cprotocol*, se kterou je možné pracovat jako s modelem protokolu.

5.2 Třídy knihovny a jejich metody

Knihovna obsahuje tři různé třídy. Pokud se podíváme, jak vlastně protokol pracuje, tak je patrné, co všechno bude obsahovat. Základní jednotkou komunikace je tady zpráva. A proto se vlastně protokoly navrhují, aby bezpečně přenesly požadovaná data. Aby knihovna správně fungovala, je potřeba, aby byl na vstupu soubor s textovou reprezentací protokolu.

Definované typy

```
typedef unsigned int uint;
```

5.3 Potřebné struktury

```
struct sname{        - struktura pro popis jednotlivých položek
      string name;  - jméno v textové podobě
      string bin;   - jméno v binární podobě
};

struct skey{         - klíč má oproti klasickému jménu ještě
                     - položku flag, do které se ukládá informace
                     - o typu klíče
      string name;
      string bin;
      uint flag;
};
```

5.4 Třida cmessage

```
class cmessage
{
private:

      sname mess;
      skey key;
      vector<cmessage> message;

public:
      void add_message(cmessage message);
      cmessage();
      cmessage(sname mess);
      uint get_size();
      cmessage get_message(uint i);
      sname get_mess();
      skey get_key();
      void clear();
      ~cmessage();
      void Crypt_message(skey key);
      cmessage Decrypt_message(skey key);
};
```

Jméno každé položky má dvě části. Je to binární část a jeho textová reprezentace. Tento popis reprezentuje struktura *sname*. Klíč má proti tomu ještě položku *flag*, která určuje, k jaké kryptografii klíč patří.

Zprávy se vytvářejí dvojího druhu. První druh je jednoduchá zpráva. Tuto zprávu poznáme tak, že zpráva má obsazenou položku *mess*, která je typu *smess*. Druhý typ můžeme nazvat složenou zprávou. Tato zpráva má ve vektoru *message* uloženy jednoduché nebo složené zprávy. Tento typ také bývá často zašifrován, takže u tohoto typu bývá vyplněna položka *key* typu *skey*. Touto konstrukcí je zajištěno i několikanásobné složení zprávy.

Třída *cmessage* má několik metod, které umožňují přístup k položkám třídy. Konstruktor třídy umožňuje vytvářet dva druhy zpráv. V prvním případě je to konstruktor bez parametrů a nastaví všechny položky třídy na výchozí hodnoty. Tyto hodnoty jsou vesměs prázdné řetězce. V druhém případě volání metody obsahuje parametr *sname mess*, která uloží tuto zprávu do položky *mess*. Tímto způsobem je možné vytvořit jednoduchou zprávu.
Destruktor třídy smaže vnitřní vektor zpráv.
Metoda *add_message(cmessage message)* přidává do vnitřního vektoru zpráv *message* další zprávu.
Metoda *get_message(uint)* vrací zprávu z n-té položky vektoru ve formě struktury smessage.
Metoda *get_mess()* vrací hodnotu jednoduché zprávy uloženou v položce *mess*.
Metoda *get_size()* vrací velikost vektoru zpráv.
Metoda *clear()* maže vnitřní vektor zpráv.
Metoda *get_message(uint i)* vrací zprávu uloženou ve vektoru zpráv *message* na pozici *i*.
Metoda *get_key()* vrátí hodnotu klíče zprávy.
Metoda *clear()* maže vnitřní vektor zpráv.
Metoda *Crypt_message(skey key)* zašifruje danou zprávu klíčem *key*. Tato metoda přidá ke složené zprávě klíč *key*.
Metoda *Decrypt_message(skey key)* dešifruje zprávu tak, že zjistí zda klíč předaný parametrem *key* je stejný jako klíč uvnitř zprávy. Potom tato metoda vrátí složenou zprávu bez klíče.

5.5 Třída csubject

```
class csubject
{
      private:
            vector<cmessage> snd;
            vector<cmessage> recieve;
            sname name;

      public:
            csubject(string name, string bin);
            ~csubject();
            void add_recv(cmessage message);
            void add_send(cmessage message);
            cmessage pop_send();
            static void send(cmessage message, csubject* where);
            static void recv(cmessage message, csubject* where);
            cmessage get_message(uint pozice, int typ);
            int get_size(int typ);
            sname get_name();
      };
```

Třída *csubject* popisuje jednotlivé subjekty komunikace. Obsahuje jméno subjektu, vektory posílaných a přijímaných zpráv.

Veřejné metody

Metody *add_recv(cmessage message)* a *add_recv(cmessage message)* ukládají zprávy do daných vektorů.
Metoda *pop_send()* vyjme zprávu, která se má odeslat.
Metody *send(cmessage message, csubject* where)* a *recv(cmessage message, csubject* where)* simulují posílání zpráv od odesilatele k příjemci.
Metoda *get_message(uint pozice, int typ)* vrací zprávu, kterou má subjekt uloženou na n-té pozici ve vektoru *recieve* nebo *send*. Parametr *typ* určuje, z jakého vektoru se zpráva bude brát. Pro hodnotu 1 je to vektor *recieve*, pro hodnotu 2 potom *snd*.
Metoda *get_size(int typ)* vrací počet zpráv ve vektoru *recieve* nebo *snd*, zase záleží na parametru *typ*.
Metoda *get_name ()* vrací jméno subjektu.

5.6 Třída cprotocol

```
struct suk{     - struktura používaná na uložení ukazatelů do vektoru
      uint from;
      uint where;
};

class cprotocol
{
      public:
            void init(std::ifstream * instr, std::ifstream * bin);
            ~cprotocol();
            void run(uint pozice);
            void run();
            void spy(uint pozice, uint uk_spy);
            uint find(string name, uint typ);
            uint get_size();
            uint num_of_mess(uint pozice,int typ);
            cmessage get_message(uint subj, uint mess, int typ);
            sname get_name_subj(uint subj);
            uint get_is_size();
            suk get_instr(uint pozice);
            uint get_spy_size();
            string extract_message(cmessage message);
            void add_spy(sname name);

      private:
            vector<suk> instr_set;
            vector<csubject> subjects;
            vector<csubject> spys;
            string get_word(std::ifstream * instr);
            cmessage give_message(std::ifstream * instr,
                                  std::ifstream * bin, sname &name);
      };
```

Veřejné metody

Metoda *init(std::ifstream * instr, std::ifstream * bin)* inicializuje objekt této třídy. Ze vstupních souborů načte pravidla protokolu, který se má testovat. Dva soubory jsou tam z důvodu, že by se mohlo chtít kontrolovat i binární vyjádření jednotlivých položek. Struktura souboru bude popsána níže.

Metoda *run* má dvě interpretace. První se používá při krokovém průchodu algoritmem protokolu. Proto má parametr *pozice*, aby se vědělo, které pravidlo se má použít. Druhá potom provede všechny kroky algoritmu najednou.
Metoda *spy(uint pozice, uint uk_spy)* je metoda, která v této implementaci vypisuje jednotlivá pravidla. V pozdějším využití této knihovny se metoda překryje jinou implementací. Teď má jen ukázat, že se správně odchytávají přenášené zprávy s informací, kdo kam zprávu poslal. Parametr *uk_spy* je pořadí ve vektoru *spys*.
Metoda *find(string name, uint typ)* vrací index ve vektoru, který je určen parametrem typ. Pro hodnotu 1 se prohledává vektor *subjects*, pro hodnotu 2 potom vektor *spys*. Při nalezení vrací jeho index ve vektoru. Při nenalezení potom 0.
Metoda *get_size()* vrací počet subjektů komunikace.
Metoda *num_of_mess(uint pozice, int typ)* vrací počet zpráv ve vektorech *recv* a *snd* subjektu na pozici dané parametrem *pozice*. Parametr typ zase určuje, který vektor se bude brát. Pro hodnotu 1 se pracuje s vektorem *recieve*, pro hodnotu 2 potom se *send*.
Metoda *get_message(uint subj, uint mess, int typ)* vrací zprávu od subjektu uloženého na pozici dané parametrem *subj*, z vektoru zpráv daného parametrem *typ*, která je uložena u subjektu na pozici dané parametrem *mess*.
Metoda *get_name_subj(uint subj)* vrací jméno subjektu uloženého na pozici udávané parametrem *subj*.
Metoda *get_is_size()* vrací velikost instrukční sady.
Metoda *get_spy_size()* vrací počet špiónů v protokolu.
Metoda *void add_spy(sname name)* umožňuje přidat dalšího agenta.

Soukromé metody

Metoda *get_word(std::ifstream * instr)* vrací načtenou hodnotu slova ze souboru daného parametrem ** instr*.
Metoda *give_message(std::ifstream * instr, std::ifstream * bin, sname &name)* pomáhá při rozbalení zprávy. Tato funkce je rekurzivní a zprávu předanou v objektu typu *cmessage* převede na řetězec.

Popis konstrukce třídy *cprotocol*

Třída se skládá ze tří položek.
Položka *inst_set* je vektor ukazatelů na aktéry komunikace. Vektor se skládá z položek typu *suk*. Tato struktura je složena ze dvou položek a to položky *from* a položky *where*, které jsou typ *unsigned int*. V metodě *run* se potom tento vektor prochází a simuluje posílání zpráv. Z tohoto vektoru si také bere metoda *spy* aktéry komunikace. Kterou položku vektoru má vzít zjistí z vektoru svých přijatých zpráv, kde se ukládá veškerá komunikace. Každé položce tohoto vektoru potom odpovídá právě jedna položka ve vektor *instr_set*.
Položka *subjects* je vektor subjektů komunikace. Tento vektor je vektor s typem *csubject*. V tomto vektoru jsou uloženi všichni aktéři komunikace podle pravidel protokolu.
Položka *spys* je vektor špiónů, kteří odposlouchávají komunikaci. Pro správnou funkci zjišťování aktérů komunikace je potřeba, aby byl před zahájením komunikace alespoň jeden špión ve vektoru *spys*.

6 Popis testovacího programu

Program s názvem `test.exe` je spustitelný z příkazové řádky. Tento binární soubor byl vytvořen ve Visual C++ v prostředí Windows a z toho také vyplývá, kde je možné soubor spouštět.

6.1 Popis vstupního souboru

Vstupní soubor musí být v textovém formátu a tvaru popsaném o řádek níže. Formální popis pravidla je uveden na prvním řádku, odpovídající vstup do programu je popsán o řádek níže.

Přepis pravidel:
$A \rightarrow B : A, B, \{N_A, N_B\}_{Kab}$
A,B,A,B,{NA,NB}Kab;

6.2 Práce s programem pro testování

Na CD jsou uloženy zdrojové kódy jak knihovny, tak testovacího programu. Implementaci jsem prováděl v programovacím jazyce C++ v prostředí Visual C++ pod operačním systémem Windows.

Pro spuštění programu je potřeba jako parametr přidat soubory pro načtení protokolu. První musí být soubor s textovou reprezentací protokolu. Druhý soubor s binární reprezentací pro předvedení principu není důležitý. Protože knihovna pracuje se jmény, které se skládají z textové a binární části, není potřeba pracovat s binární reprezentací dat.

6.3 Porovnání výsledků programu

Program, který využívá knihovnu `protocol.cpp`, byl testován na protokolu Needham-Schroeder.

Pravidla protokolu
1. $A \rightarrow S$: *A, B*
2. $S \rightarrow A$: $\{ Kb \}_{Ks-1}$
3. $A \rightarrow B$: $\{ Na, A \}_{Kb}$
4. $B \rightarrow S$: *B, A*
5. $S \rightarrow B$: $\{ Ka, A \}_{Ks-1}$
6. $B \rightarrow A$: $\{ Na, Nb \}_{Ka}$
7. $A \rightarrow B$: $\{ Nb \}_{Kb}$

Program vypisuje velikost instrukční sady a jednotlivé kroky protokolu. U každého kroku je uvedeno pravidlo, které se provede a zprávy k odeslání jednotlivých subjektů komunikace a také zprávy, které daný subjekt přijal.

Jak je vidět, máme 7 pravidel protokolu a 3 účastníky komunikace. Vypíšeme si také jednotlivé subjekty a podíváme se, co mají v položkách *recv* a *snd*.

U všech subjektů bude vektor *recv* prázdný, protože se ještě nic neodeslalo. Takže ve vektoru *snd* budou mít subjekty tyto informace.

A: *A, B;* $\{ Na, A \}_{Kb}$; $\{ Nb \}_{Kb}$
B: *B, A;* $\{ Na, Nb \}_{Ka}$
S: $\{ Kb \}_{Ks-1}$; $\{ Ka, A \}_{Ks-1}$

Pro názornost v tomto případě středník odděluje jednotlivé zprávy ve vektoru.

Následuje výstup programu:

```
**********************************************************************************

  Demonstracni program knihovny pro podporu navrhu bezpecnostnich protokolu

**********************************************************************************

Velikost instrukcni sady : 7

Pocet ucastniku komunikace : 3
Vypis obsahu polozek recieve a send u jednotlivych polozek v krocich algoritmu

Krok 0
Polozky send ucastniku komunikace:
Subjekt A-> A, B,
Subjekt A-> {Na, A, }Kb
Subjekt A-> {Nb, }Kb
Subjekt S-> {Kb, }KS-1
Subjekt S-> {Ka, A, }Ks-1
Subjekt B-> B, A,
Subjekt B-> {Na, Nb, }Ka

Polozky recieve ucastniku komunikace:
```

Obr. 8: Výpis programu na obrazovku

Po zadání libovolného písmena posuneme algoritmus dále.

Teď se má provést první krok protokolu.
Pravidlo, které se k tomu použije je $A \rightarrow S$: *A, B*. To znamená, že subjekt *A* pošle zprávu serveru *S*, takže by se měla smazat zpráva z vektoru *snd* subjektu *A* a přidat do vektoru *recv* serveru *S*. Položky jednotlivých subjektů komunikace by měly vypadat takto:

Vektor *recv*	Vektor *snd*
A: $\{ Na, A \}_{Kb}; \{ Nb \}_{Kb}$	
B: *B, A;* $\{ Na, Nb \}_{Ka}$	
S: $\{ Kb \}_{Ks-1};$ $\{ Ka, A \}_{Ks-1}$	*A, B*

Následuje výstup programu:

```
Krok 1
Pravidlo: A -> S: A, B,

Polozky send ucastniku komunikace:
Subjekt A-> {Na, A, }Kb
Subjekt A-> {Nb, }Kb
Subjekt S-> {Kb, }KS-1
Subjekt S-> {Ka, A, }Ks-1
Subjekt B-> B, A,
Subjekt B-> {Na, Nb, }Ka

Polozky recieve ucastniku komunikace:
Subjekt S-> A, B,

***   Press char to continue   ***
```

Obr. 9: Výpis programu na obrazovku

Další krok protokolu má pravidlo $S \rightarrow A$: $\{ Kb \}_{Ks-1}$ značí, že se bude posílat od subjektu S k subjektu A. Obsah jednotlivých vektorů by měl vypadat následovně:

Vektor *recv*	Vektor *snd*
A: $\{ Na, A \}_{Kb}$; $\{ Nb \}_{Kb}$	$\{ Kb \}_{Ks-1}$
B: B, A; $\{ Na, Nb \}_{Ka}$	
S: $\{ Ka, A \}_{Ks-1}$	A, B

Následuje výstup programu:

```
Krok 2
Pravidlo: S -> A: <Kb, >KS-1

Polozky send ucastniku komunikace:
Subjekt A-> <Na, A, >Kb
Subjekt A-> <Nb, >Kb
Subjekt S-> <Ka, A, >Ks-1
Subjekt B-> B, A,
Subjekt B-> <Na, Nb, >Ka

Polozky recieve ucastniku komunikace:
Subjekt A-> <Kb, >KS-1
Subjekt S-> A, B,

***   Press char to continue   ***
```

Obr. 10: Výpis programu na obrazovku

Další kroky jsou podobné. Budu udávat jen, co se očekává a výpis programu.

Krok 3.

Pravidlo: $A \rightarrow B$: $\{ Na, A \}_{Kb}$

Vektor *recv*	Vektor *snd*
A: $\{ Nb \}_{Kb}$	$\{ Kb \}_{Ks-1}$
B: $B, A;$ $\{ Na, Nb \}_{Ka}$	$\{ Na, A \}_{Kb}$
S: $\{ Ka, A \}_{Ks-1}$	$A, B;$

Následuje výstup programu:

```
Krok 3
Pravidlo: A -> B: <Na, A, >Kb

Polozky send ucastniku komunikace:
Subjekt A-> <Nb, >Kb
Subjekt S-> <Ka, A, >Ks-1
Subjekt B-> B, A,
Subjekt B-> <Na, Nb, >Ka

Polozky recieve ucastniku komunikace:
Subjekt A-> <Kb, >KS-1
Subjekt S-> A, B,
Subjekt B-> <Na, A, >Kb

***   Press char to continue   ***
```

Obr. 11: Výpis programu na obrazovku

Krok 4.
Pravidlo $B \rightarrow S$: *B, A*

Vektor *recv*	Vektor *snd*
A: { *Nb* }$_{Kb}$	{ *Kb* }$_{Ks-1}$
B: { *Na, Nb* }$_{Ka}$	{ *Na, A* }$_{Kb}$
S: { *Ka, A* } $_{Ks-1}$	*A, B;* *B, A*

Následuje výstup programu:

```
Krok 4
Pravidlo: B -> S: B, A,

Polozky send ucastniku komunikace:
Subjekt A-> <Nb, >Kb
Subjekt S-> <Ka, A, >Ks-1
Subjekt B-> <Na, Nb, >Ka

Polozky recieve ucastniku komunikace:
Subjekt A-> <Kb, >KS-1
Subjekt S-> A, B,
Subjekt S-> B, A,
Subjekt B-> <Na, A, >Kb

***   Press char to continue   ***
```

Obr. 12: Výpis programu na obrazovku

Krok 5.
Pravidlo $S \rightarrow B$: { *Ka, A* } $_{Ks-1}$

Vektor *recv*	Vektor *snd*
A: { *Nb* }$_{Kb}$	{ *Kb* }$_{Ks-1}$
B: { *Na, Nb* }$_{Ka}$	{ *Na, A* }$_{Kb}$; *Ka, A* } $_{Ks-1}$
S:	*A, B; B, A*

Následuje výstup programu:

```
Krok 5
Pravidlo: S -> B: <Ka, A, >Ks-1

Polozky send ucastniku komunikace:
Subjekt A-> <Nb, >Kb
Subjekt B-> <Na, Nb, >Ka

Polozky recieve ucastniku komunikace:
Subjekt A-> <Kb, >KS-1
Subjekt S-> A, B,
Subjekt S-> B, A,
Subjekt B-> <Na, A, >Kb
Subjekt B-> <Ka, A, >Ks-1

***   Press char to continue   ***
```

Obr. 13: Výpis programu na obrazovku

Krok 6.
Pravidlo $B \rightarrow A$: $\{ Na, Nb \}_{Ka}$

Vektor *recv*	Vektor *snd*
A: $\{ Nb \}_{Kb}$	$\{ Kb \}_{Ks-1}$; $\{ Na, Nb \}_{Ka}$
B:	$\{ Na, A \}_{Kb}$; $Ka, A \}_{Ks-1}$
S:	*A, B; B, A*

Následuje výstup programu:

```
Krok 6
Pravidlo: B -> A: <Na, Nb, >Ka

Polozky send ucastniku komunikace:
Subjekt A-> <Nb, >Kb

Polozky recieve ucastniku komunikace:
Subjekt A-> <Kb, >KS-1
Subjekt A-> <Na, Nb, >Ka
Subjekt S-> A, B,
Subjekt S-> B, A,
Subjekt B-> <Na, A, >Kb
Subjekt B-> <Ka, A, >Ks-1

***   Press char to continue   ***
```

Obr. 14: Výpis programu na obrazovku

Krok 7.
Pravidlo $A \rightarrow B$: $\{ Nb \}_{Kb}$

Vektor *recv*	Vektor *snd*
A:	$\{ Kb \}_{Ks-1}$; $\{ Na, Nb \}_{Ka}$
B:	$\{ Na, A \}_{Kb}$; $Ka, A \}_{Ks-1}$; $\{ Nb \}_{Kb}$
S:	*A, B; B, A*

Následuje výstup programu:

```
Krok 7
Pravidlo: A -> B: <Nb, >Kb

Polozky send ucastniku komunikace:

Polozky recieve ucastniku komunikace:
Subjekt A-> <Kb, >KS-1
Subjekt A-> <Na, Nb, >Ka
Subjekt S-> A, B,
Subjekt S-> B, A,
Subjekt B-> <Na, A, >Kb
Subjekt B-> <Ka, A, >Ks-1
Subjekt B-> <Nb, >Kb

***   Press char to continue   ***
```

Obr. 15: Výpis programu na obrazovku

Jak je vidět z tohoto výpisu, tak logický postup protokolu, který jsme si sami odvodili, souhlasí s výstupem programu. Program tedy odsimuloval provádění protokolu správně. V souborech *needham.txt*, *yahalom.txt, otway.txt* a *kerberos.txt* jsou nachystané protokoly na testování tímto programem.

7 Závěr

Tento ročníkový projekt měl za úkol návrh knihovny, která by simulovala daný algoritmus. Simulace je potřeba k analýze útoků na protokol. Tato práce v první části stručně popisuje některé bezpečnostní protokoly a jejich formální popis. Jsou to všechno protokoly, na kterých se dále stavělo. Příkladem může být i Needham-Schroeder protokol, který je předchůdcem protokolu Kerberos. V druhé části je potom popsána navržená knihovna a testovací program. Tento program je pouze pro demonstraci funkce navržené knihovny a nemá jiné využití.

Knihovna se bude moci využít pro návrh bezpečnostních protokolů, hlavně v části jeho formální verifikace. S její pomocí lze analyzovat chování algoritmu. Při zadávání projektu se plánovalo, že při dalším využití knihovny se některé metody překryjí, aby lépe vyhovovaly další práci. Nyní je metoda *spy* navržena tak, aby vypisovala prováděná pravidla. V budoucnu by tato metoda pomáhala při pasivním útoku na zprávu. Knihovna je navržena tak, aby komunikace subjektů co nejvíce odpovídala skutečnosti. Na výpisech testovacího programu je možné vidět jednotlivé zasílání zpráv. Toho lze využít při útocích na zprávy a zjišťovat, jak se protokol bude chovat při podvržení zprávy útočníkem.

Literatura

[1] Pavel Očenášek: Verifikace bezpečnostních protokolů. Brno 2003. 55 s. Diplomová práce na Fakultě informačních technologií Vysokého učení technického. Vedoucí diplomové práce Daniel Cvrček, 55 stran

[2] Martín Abadi, Roger Needham: Prudent Engineering Practice for Cryptographic Protocols, Preliminary version of this paper has appeared in the Preceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy. listopad 1995, 51 stran

[3] Michael Burrows., Martín Abadi, Roger Needham: A Logic of Authentication, SCR Research Report 39, únor 1989, 32 stran

[4] Stefanos Gritzalis, Diomidis Spinellis, Panagiotis Georgiadis: Security Protocols over open network and distributed systems: Formal methods for their Analysis, Design, and Verification, Computer Communications, 22(8): 697-707, květen 1999, 21 stran