VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJE A INTERAKTIVNÍ PROSTŘEDÍ PRO SIMULACI KOMUNIKACE

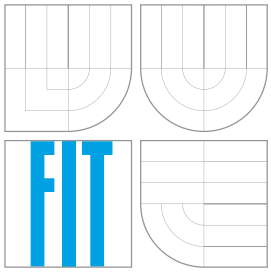
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER MIKUŠ

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJE A INTERAKTIVNÍ PROSTŘEDÍ PRO SIMULACI KOMUNIKACE

TOOLS FOR ENVIRONMENT FOR THE SIMULATION OF COMMUNICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETER MIKUŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL OČENÁŠEK, Ph.D.

BRNO 2010

Zadání diplomové práce/4899/2009/xmikus06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Mikuš Peter, Bc.**
Obor: Informační systémy
Téma: **Nástroje a interaktivní prostředí pro simulaci komunikace**
Tools for Environment for the Simulation of Communication
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s principy a použitím komunikačních protokolů (KP), zaměřte se na bezpečnostní protokoly (BP).
2. Seznamte se s nástroji a prostředí pro interaktivní simulaci komunikačních protokolů.
3. Proveďte srovnání jednotlivých prostředí, uveďte jejich principy, specifické vlastnosti, nedostatky a cíle použití. Zaměřte se také na licenci a dostupnost nástrojů a jejich detailní parametry.
4. V dostupných nástrojích a prostředích demonstračně implementujte vybrané KP/BP. Porovnejte způsob a výsledky implementace.
5. K dostupným nástrojům napiště uživatelskou příručku, která bude popisovat principy instalace a nejdůležitější charakteristiky použití. Pro daný nástroj vytvořte sadu příkladů. Navrhněte a implementujte demonstrační úlohy pro využití ve výuce v síťových předmětech na FIT VUT.
6. Diskutujte získané znalosti a možnosti dalšího pokračování projektu.

Literatura:

- Law A., Kelton D.: Simulation Modelling and Analysis, McGraw-Hill, 1991, ISBN 0-07-100803-9
- Ross, S.: Simulation, Academic Press, 2002, ISBN 0-12-598053-1
- Očenášek Pavel: Verifikace bezpečnostních protokolů : diplomová práce, Brno, CZ, FIT VUT, 2003, p. 54
- Další literatura dle pokynů vedoucího práce.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 - 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Očenášek Pavel, Ing.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Komunikace mezi zařízeními se neobejde bez jasně definovaných pravidel. Tato pravidla se společně označují jako komunikační protokol. V této práci se zabývám komunikačními protokoly, konkrétně bezpečnostními protokoly. Jejich návrh vyžaduje speciální nástroje, které umožňují simulaci a odhalení bezpečnostních chyb v jejich návrhu. Zaměřuji se tedy na nástroje umožňující interaktivní simulaci bezpečnostních protokolů. Detailně popisuji jednotlivé nástroje a srovnávám jejich kladné a záporné vlastnosti. V dostupných nástrojích implementuji bezpečnostní protokoly. Výsledem práce jsou demostrační úlohy využitelné v síťových předmětech na FIT VUT.

Abstract

Communication between devices should be based on predefined rules. These rules are called communication protocols. In this master thesis I am concerned with communication protocols, specially security protocols. Their design demands specialized tools, that will provide interactive simulations and security testing. I have described each of these tools in detail and mentioned about their properties, their pros and cons. In available tools I have implemented security protocols. The result is set of demonstration tasks that are usable in network courses at FIT VUT.

Klíčová slova

komunikační protokol, bezpečnostní protokol, simulace, nástroje, útoky, demonstrační úlohy, Scyther.

Keywords

communication protocol, security protocol, simulation, tools, attacks, demonstration tasks, Scyther.

Citace

Peter Mikuš. Nástroje a interaktivní prostředí pro simulaci komunikace, diplomová práce, Brno, FIT VUT v Brně, 2010

Nástroje a interaktivní prostředí pro simulaci komunikace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Pavla Očenáška, Ph.D.

.....
Peter Mikuš
30. dubna 2010

Poděkování

Ďakujem svojmu vedúcemu Pavlovi Očenáškovi za odbornú pomoc a usmerňovanie pri písaní diplomovej práce.

Obsah

Kapitola 1

Úvod

Komunikácia medzi elektronickými zariadeniami najrôznejších druhov od počítačov cez mobilné telefóny až po platobné terminály je jednou zo základných požiadaviek dnešnej doby. Prenos informácií medzi zariadeniami sa však nezaobíde bez jasne definovaných pravidiel. Tieto pravidlá sa spoločne označujú ako komunikačný protokol. Informácie prenášané pomocou týchto protokolov cez sieť možu mať vysokú hodnotu a sú tak terčom útokov. Špeciálnu oblasť preto predstavujú bezpečnostné protokoly nasadzované predovšetkým na miestach, kde vyžadujeme zabezpečenú, šifrovanú komunikáciu. Jedná sa o protokoly prenášajúce citlivé informácie napríklad platobné príkazy či autentizačné údaje. Návrh, testovanie a správne pochopenie komunikačných a bezpečnostných protokolov si vyžaduje špecifické nástroje a prostredia umožňujúce ich simuláciu. Niektoré nástroje dokážu odhaliť prípadné chyby alebo bezpečnostné zraniteľnosti v návrhu.

Cieľom tejto práce je zoznámiť sa s princípmi, použitím a interaktívnou simuláciou komunikačných protokolov so zameraním na bezpečnostné protokoly. Na základe získaných poznatkov a pomocou dostupných nástrojov odsimulovať vybrané bezpečnostné protokoly a k týmto nástrojom napísať uživateľskú príručku. Motiváciou je pomocou sady úloh a príkladov, zoznámiť čitateľa s danou problematikou a zároveň vytvoriť zoznam nástrojov umožňujúcich interaktívnu simuláciu a demonštrovať ich možnosti. Získané poznatky môžu byť využité v sieťových predmetoch na FIT VUT.

V nasledujúcej kapitole sa budem venovať teoretickému základu z oblasti protokolov a ich vlastností, hlavne bezpečnosti. Stručne popíšem základné výrazové prostriedky pre popis protokolov. V tretej kapitole budú uvedené jednotlivé nástroje, ktoré umožňujú ich interaktívnu simuláciu a k týmto nástrojom uvediem hlavné vlastnosti, dostupnosť, detailné parametre, typ licencie a cieľ použitia. Na vybraných protokoloch odsimulujem a porovnám spôsoby implementácie v kapitole štyri a navrhnem sadu demonstračných úloh a príkladov v kapitole piatej. V záverečnej kapitole zhodnotím svoju prácu a získané poznatky.

Kapitola 2

Komunikačné protokoly

V tejto kapitole uvediem teoretický úvod do problematiky komunikačných protokolov obecne a následne sa zameriam na bezpečnostné protokoly. Spomeniem základné vlastnosti bezpečnostných protokolov ako aj výrazové prostriedky pre ich popis.

Protokoly predstavujú formálnu stránku riadenia komunikácie pretože poskytujú formálny nástroj na uskutočňovanie vecnej náplne riadenia komunikácie a pomáhajú tak realizovať príslušné komunikačné funkcie. Komunikačné protokoly tvoria súhrn pravidiel formátu a procedúr určujúcich výmenu údajov medzi dvomi komunikujúcimi prvkami [19]. Protokoly sú implementované hardwarom, softwarom prípadne kombináciou obidvoch. Na najnižšej úrovni protokol definuje správanie hardwarového spojenia.

Komunikácia v sieti prebieha na jednotlivých vrstvách (úrovniach). Každá vrstva vykonáva špecifickú funkciu a komunikuje s vrstvou na rovnakej úrovni. Poskytuje svoje služby vrstve vyššej a naopak využíva služby vrstvy nižšej. Typickým príkladom vrstevnej komunikácie je model *ISO/OSI*. Tento model je snahou o štandardizáciu komunikácie v počítačových sieťach.

Medzi najznámejšie komunikačné protokoly patria napríklad protokoly HTTP, SMTP, IP, TCP a ďalšie.

2.1 Bezpečnostné protokoly

Samostatnú oblasť komunikačných protokolov predstavujú bezpečnostné protokoly. Tie nachádzajú uplatnenie najmä v situáciách, kde potrebujeme prenášať citlivé informácie ako napríklad tajné kľúče, autentizačné údaje prípadne transakcie na bankovom účte. Tieto protokoly zaisťujú pomocou kryptografických mechanismov (šifrovania) základné bezpečnostné funkcie ako sú *autentizácia*, *autorizácia*, *integrita*, *dôvernosť* a niekedy *nepopierateľnosť*. Bezpečnostné funkcie možno uplatniť na každej z vrstiev referenčného ISO/OSI modelu [8].

Fyzická vrstva Problémy fyzickej vrstvy majú technický alebo technologický charakter. Cielený útok na túto vrstvu, može byť napríklad: prerušenie vodičov, úmyselné rušenie (interferencia, zkreslenie, ...), odposluch (bezdrôtové prenosy), zachytenie vyžarovania.

Linková vrstva Protokoly linkovej vrstvy spravidla zaisťujú ochranu proti technickým zlyhaniam (kontrolné súčty, sekvenčné čísla rámcov, ...) a zaisťujú ochranu proti úmyselným útokom. Takým protokolom je špecifikácia PPP (Point-to-Point Protocol), ktorá pozostáva zo sady protokolov a tie zapúzdrujú rámce prenosových technológií Ethernet, ATM, FrameRelay, HDLC za účelom zabezpečenia autenticity a dôvernosti spojenia. Patria sem napríklad protokoly LCP, PAP, CHAP, EAP [10].

Sieťová vrstva Z hľadiska informačnej bezpečnosti je táto vrstva veľmi významná, pretože môže byť na nej uplatnená rada bezpečnostných služieb. Poskytuje autentizáciu, dôveryhodnosť a integritu. Bezpečnostné protokoly sieťovej vrstvy sú buď riešenia proprietárne (SKIP) alebo štandardy (IPsec) [10].

Transportná vrstva Bezpečný transport dát zaisťujú protokoly SSL (Secure Socket Layer) a TLS (Transport Layer Security Protocol). Implementačná vrstva SSL/TLS je medzivrstvou transportnej a aplikačnej vrstvy TCP/IP architektúry. Zaisťuje bezpečnosť rôznym aplikačným protokolom ako sú napríklad HTTP, FTP, IMAP. Bezpečnostné služby, ktoré protokoly SSL/TLS pokrývajú sú dôvernosť, integrita, autentifikácia a nepopierateľnosť [10].

Aplikačná vrstva Najvyššia vrstva architektúry TCP/IP ponúka najviac možností pre implementáciu bezpečnostných služieb v komunikačních procesoch. Niektoré z implementácií môžu byť zdieľané rôznymi sieťovými službami. Z tejto skupiny je veľmi perspektívny protokol SET (Secure Electronic Transaction), ktorý je určený pre internetové platobné transakcie [10].

Z hľadiska šifrovania komunikácie (správ), možno bezpečnostné protokoly rozdeliť na protokoly používajúce asymetrické a symetrické šifrovanie.

Asymetrické šifrovanie Pri tomto type šifrovania sú použité dva typy kľúčov. Súkromný a verejný. Súkromný kľúč je tajomstvom vlastníka a umožňuje dešifrovanie správy. Verejný kľúč šifruje správu a je voľne k dispozícii. Medzi algoritmy patrí RSA, DSA, Diffie-Hellman. Príkladom využitia sú protokoly IKE, TMN alebo SET. Asymetrický model možno použiť pre implementáciu dôvernosti, autentizácie a nepopierateľnosti (digitálny podpis) [9].

Symetrické šifrovanie Používa jeden typ šifrovacieho kľúča – tajný kľúč (zdielaný kľúč). Slúži zároveň na šifrovanie a dešifrovanie. Ktokoľvek, kto vlastní tento kľúč má prístup k správe. Medzi algoritmy patrí DES, 3DES, IDEA, RC2, RC5, AES. Príkladom je *Yahalom* protokol. Symetrické šifrovanie zaisťuje dôvernosť a autentizáciu [9].

2.2 Funkcie bezpečnostných protokolov

Ako bolo spomenuté, medzi hlavné funkcie (služby), ktoré poskytujú bezpečnostné protokoly patria autentizácia, autorizácia, integrita, dôvernosť a nepopierateľnosť.

2.2.1 Autentizácia

Autentizácia je proces, ktorý rieši overenie identity jendej alebo obidvoch komunikujúcich strán. Pokiaľ je v komunikačnom protokole zaradený kvalitný autentizačný mechanizmus, je pre útočníka veľmi obtiažne podvrhnúť zprávu s falošným pôvodom [15].

Silná autentizácia Systém umožňuje silnú autentizáciu ak je splnená nasledujúca podmienka: pokiaľ subjekt A obdrží správu, v ktorej je ako odosielateľ uvedený subjekt B, potom B odoslal práve túto správu subjektu A [15].

Slabá autentizácia Systém umožňuje slabú autentizáciu ak je splnená nasledujúca podmienka: pokiaľ subjekt A obdrží správu a ako odosielateľ je uvedený subjekt B, potom buď B odoslal práve túto správu subjektu A alebo B uvedenú skutočnosť poprie [15].

Identitu možno overiť pomocou [16]:

- toho, čo používateľ pozná, napríklad meno, heslo alebo PIN
- toho, čo používateľ vlastní, napríklad privátny kľúč, smart card
- toho, čím používateľ je, napríklad biometrické údaje
- toho, čo používateľ dokáže, napríklad CAPTCHA

2.2.2 Autorizácia

Autorizácia alebo tiež služba riadenia prístupu, zaisťuje udelenie, prípadne odoprenie prístupu na základe overenia identity. Najčastejšie je spájaná práve so službou autentifikácie. Autentifikačné údaje môžu byť ďalej porovnané napríklad s tabuľkou prístupových práv (ACL) a výsledkom je úroveň oprávnenia pristupovať k požadovanej informácií či objektu. Tieto služby bývajú málokedy súčasťou sieťových protokolov a často sú implementované až v operačnom systéme alebo aplikácií [8].

2.2.3 Integrita

Integrita je vlastnosť, kedy prenášané dáta nesmú byť zmenené žiadnou neautorizovanou stranou, nesmú byť umelo zadržiavané, či opakovane vysielané po neoprávnenom odposluchu [15]. Úlohou tejto služby je teda najčastejšie pomocou kryptografických algoritmov zabezpečiť túto vlastnosť. Z kryptografických algoritmov sa používajú napríklad MD5, SHA-1, SHA-2.

Službu možno rozdeliť na [8]:

- Integrita spojenia – poskytuje ochranu pred neautorizovanou modifikáciou bez ohľadu na nadviazané spojenia.
- Integrita prenosu zpráv – zaisťuje ochranu pred neautorizovanou modifikáciou v rámci nadviazaného spojenia.
- Selektívna integrita spojenia a selektívna integrita zpráv – majú za úlohu zaistiť integritu iba niektorých častí prenášanej správy.

2.2.4 Dôvernosť

Táto skupina služieb poskytuje ochranu prenášaných dát pred neautorizovaným odhalením. Službu možno rozdeliť na [8]:

- Dôvernosť prenosu správ – poskytuje ochranu pred neautorizovaným odhalením bez ohľadu na nadviazené spojenie.
- Dôvernosť spojenia – zaisťuje ochranu pred neautorizovaným odhalením v rámci nadviazaného spojenia. Táto služba vyžaduje existujúce spojenie.
- Dôvernosť toku dát (Traffic Flow Confidentiality) – má za úlohu zabrániť útočníkovi, aby zo znalostí toku dát (adresy prenášaných správ, dĺžka prenášaných správ, časové intervaly medzi prenášanými správami, ...) dokázal odvodiť dôverné informácie o prenášaných dátach.
- Selektívna dôvernosť – má za úlohu zaistiť dôvernosť iba niektorých častí prenášanej správy.

2.2.5 Nepopierateľnosť

Služba nepopierateľnosť (odosielateľa, doručenia), má za úlohu jednoznačne preukázať príjemcovi resp. odosielateľovi na jednej strane, odoslanie resp. prijatie na strane druhej.

2.3 Popis protokolov

2.3.1 Formálny popis

Správa Je základným prvkom komunikácie. Správy, ktoré sú vymieňané medzi subjektami pomocou protokolu, sú zložené z menších, atomických správ. Existujú štyri typy atomických správ [4]:

- Kľúče – sú požívané k šifrovaniu správ. Hlavnou vlastnosťou je, že každý kľúč k má kľúč inverzný k^{-1}. Pri symetrickom šifrovaní, je dešifrovací kľuč rovnaký ako šifrovací, teda $k = k^{-1}$. Zároveň platí $(k^{-1})^{-1} = k$.
  Kľúče sa označujú: $K_a, K_b, K_{ab}, \ldots$
- Nonce – si možno predstaviť ako náhodne generované čísla. Základom je, aby nikto nebol schopný vopred odhadnúť túto hodnotu. Zaručuje tzv. čerstvosť správy (freshness). Každá správa obsahujúca nonce je generovaná po tom, čo bola vygenerovananá hodnota nonce. Nonce sa označuje: $N_a, N_b, \ldots$
- Dáta – neovplyvňujú spôsob fungovania protokolu ale sú predmetom komunikácie.
- Mená účastníkov – odkazujú na účastníkov komunikácie. Účastníci sa najčastejšie označujú ako: $A, B, S, \ldots$ (*Alice*, *Bob*, *Server*, ...)

Správy zložené z atomických správ sa označujú ako zložené správy a zapisujú sa $\{M, N\}$. Zašifrovaná správa M, kľúčom k sa zapisuje $\{M\}_k$. Odoslanie správy možno zapísať:

$$A \rightarrow B : \{A, N_a\}_k$$

2.3.2 Jazyky

SPL (Security Protocol Language) je asynchrónny, procesne orientovaný jazyk založený na π-*Calculus*. Každý agent (proces) je definovaný ako výraz (*term*) v algebre. Tá umožňuje prezentovať sekvenčné aj paralelné vstupné a výstupné akcie [3]. Akcie môžu obsahovať správy.

Napríklad odoslanie správy od A k B, ktorá bude obsahovat *nonce*, sa zapíše

$$out\ new\{y\}\{y, A\}_{Pub(B)}$$

kde y predstavuje *nonce*. Prijatie očakávanej správy sa zapíše

$$in\ pat\{x, Z\}\{x, Z\}_{Pub(B)}$$

kde Z je vzor očakávanej správy (pattern).

NetPDL (Network Protocol Description Language) je aplikačne nezávislý jazyk pre popis paketov, ktorý umožňuje vytvárať univerzálnu databázu s popismi protokolov. Jednou z hlavných výhod je jednoduchosť popisu. NetPDL sa nezameriava na popis správania protokolu. Naproti tomu sa zameriava na efektívny popis paketov, ich hlavičiek a zapúzdrenia. NetPDL je založený na XML a preto je možné ho spracovať pomocou programov a knižníc podporujúcich XML. Zároveň je možné jednoducho aktualizovať zmeny v špecifikácií protokolov [20].

Príklad hlavičky paketu IPv4 zapísaného v NetPDL:

```
<protocol name='IPv4'>
  <format>
    <fields>
      <field type='bit' name='ver' longname='Version' size='1'
      mask='0xF0'/>
      <field type='bit' name='hlen' longname='Header length' size='1'
      mask='0x0F'/>
      <field type='fixed' name='tos' longname='Type of service' size='1'/>
      ...
    </fields>
  </format>
</protocol>
```

HLPSL Jazyk je vyvíjaný v rámci frameworku projektu *AVISPA*. Je modulárny a umožňuje špecifikáciu riadiaceho toku, datových štruktúr a rôznych modelov útokov. Jeho sémantika je postavená na Lamport's TLA (Temporal Logic of Actions). Definuje popis rolí pomocou konečného automatu, kde prechod znamená prijatú alebo odoslanú správu. Explicitne definuje role, generovanie nonce a výmenu správ [2].

Príklad zápisu role v jazyku HLPSL:

```
role alice(A,B:agent, G:text, Snd,Rcv:channel(dy)) played_by A def=
  local State:nat, Na,Nsecret:text, X,K:message
```

```
  init  State:=1
transition
  1.   State=1 /\ Rcv(start) =|>
       State':=2 /\ Na':=new() /\ Snd(exp(G,Na'))
  2.   State=2 /\ Rcv(X')  =|>
       State':=3 /\ K':=exp(X',Na) /\ Nsecret':= new() /\ Snd({Nsecret'}_K')
end role
```

Role sú spojené v *session*, kde zdieľajú vedomosti:

```
role session (A,B:agent, G:text) def=
  local SND_A,RCV_A,SND_B,RCV_B:channel(dy) def=
composition
  alice(A,B,G,SND_A,RCV_A) /\ bob(B,A,G,SND_B,RCV_B)
end role
```

Prostredie slúži k spusteniu a verifikácií protokolu. Nastavuje úvodné vedomosti útočníka a *session*:

```
role environment() def=
  const a,b:agent, g:text
composition
  session(a,b,g,Snd,Rcv)
end role
```

CAS+ Jazyk bol navrhnutý pre jednoduchú špecifikáciu a verifikáciu bezpečnostných protokolov. Hlavným cieľom bolo vytvoriť jazyk, ktorý by bol jednoduchý a poskytoval možnosti jazyka HSPSL [21]. Špecifikácia protokolu pozostáva zo šiestich častí: deklarácia identifikátorov, postupnosť správ, znalosti agenta, *session* premenné, znalosti útočníka a cieľ verifikácie.

Príklad zápisu protokolu:

```
protocol TV;  %  symmetric  key
identifiers
  A,B : user;
  Ins : number;
  K : symmetric_key;
messages
  1. B -> A : B,{Ins}K
  2. A -> B : A,B,{Ins}K
knowledge
  B : A,K;
  A : K;
session_instances
  [B:tv,A:scard,K:onekey];
intruder_knowledge
  scard;
goal
  A authenticates B on Ins;
```

2.4 Formálna verifikácia

Formálna verifikácia je proces overenia, či daný model (algoritmus) spĺňa špecifikáciu.

Kontrola modelom Je založená na konštrukcii pravdepodobnostných množín útokov, ktoré vychádzajú z algebraických vlastností protokolov [15]. Môže dôjsť k expanzii stavov, pretože je potrebné prehľadať stavový priestor. Príkladom sú rôzne špecifikačné jazyky a expertné systémy.

Autentizačná logika Predstavuje modálnu logiku podobnú tým, ktoré boli vytvorené pre analýzu a vývoj vedomostí a predpokladov. Je založená na axiomizácií vedomostí a predpokladov [15]. Príkladom je BAN, GNY prípadne KPL logika.

Induktívne techniky Tieto metódy nahrádzajú náročné prehľadávanie teorémov o tomto prehľadávaní. Tieto techniky sú komplementom ku kontrole modelom. Sú založené na formalizácií problémov, pri ktorých sa používajú hypotézy a autentizačné vlastnosti. Techniky formálne modelujú aktuálne operácie v protokole a overujú teorémy týchto operácií [15]. Príkladom je *theorem proving* (Isabelle) alebo *spi calculus*.

2.5 Metódy útokov

K základným typom útokov, ktoré majú za cieľ odhaliť obsah prenášaných informácií alebo prerušiť komunikáciu patrí odpočúvanie, podstrčenie falošnej identity, modifikácia prenášaných správ a prerušenie komunikácie.

Odpočúvanie Komunikácia medzi dvoma subjektami *Alice* a *Bob* je odpočúvaná útočníkom. Útočník vystupuje ako pasívny účastník.

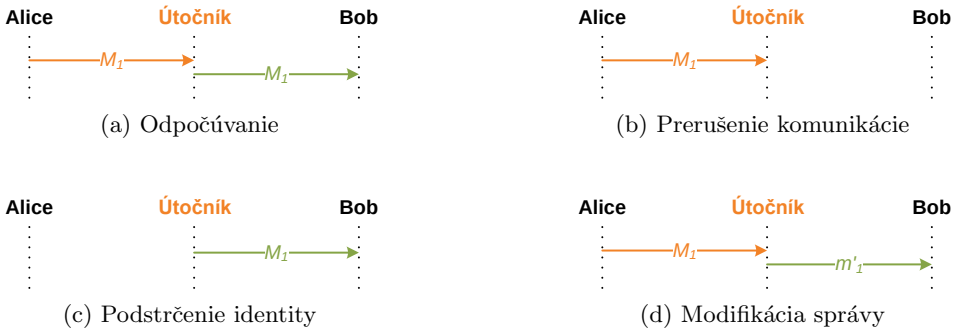
Prerušenie komunikácie Komunikácia medzi subjektom *Alice* a *Bob* je prerušená útočníkom.

Podstrčenie identity Útočník predstiera falošnú identitu voči niektorému zo subjektov komunikácie.

Modifikácia správy Útočník modifikuje správy posielané medzi dvoma subjektami.

2.5.1 Útok zo stredu

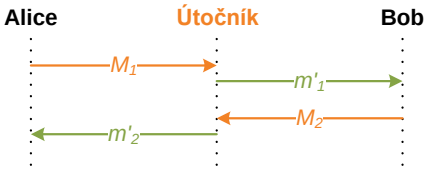
Tento útok spočíva v tom, že útočník vstúpi do komunikácie medzi subjekty *Alice* a *Bob*. Útočník odpočúva, analyzuje a prípadne pozmeňuje správy. Zároveň útočník predstiera voči *Alice* identitu *Boba* a voči *Bobovi* predstiera identitu *Alice*. Obe strany komunikujú bežným spôsobom a nevedia o prítomnosti útočníka. Útok je znázornený na obr. 2.2. Prevenciou je použitie silnej autentifikácie, PKI (public key infrastructure) a súkromných kľúčov. Jedná sa o aktívny typ útoku [14, 12].



(a) Odpočúvanie (b) Prerušenie komunikácie

(c) Podstrčenie identity (d) Modifikácia správy

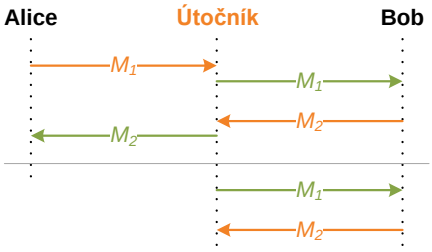
Obrázek 2.1: Metódy útokov v sieti (časový diagram)



Obrázek 2.2: Útok zo stredu (časový diagram)

2.5.2 Útok prehrávaním

V tomto útoku, podobne ako pri útoku zo stredu, vystupuje útočník v roli prostredníka. Útočník odpočúva komunikáciu a následne ju opakuje voči niektorému zo subjektov, pričom tento daný subjekt nedokáže overiť jeho skutočnú identitu. Ochranou voči tomuto typu útoku je použitie časového razítka [15]. Podobne ako útok zo stredu aj tento útok je aktívny. Útok je znázornený na obr. 2.3.



Obrázek 2.3: Útok prehrávaním (časový diagram)

Kapitola 3

Nástroje

V nasledujúcej kapitole predstavujem nástroje umožňujúce návrh, analýzu a simuláciu bezpečnostných protokolov. Ku každému nástroju uvádzam detailný popis vrátane zdroja, typu licencie a podporovaných operačných systémov.

3.1 AVISPA

Adresa: `http://avispa-project.org/`

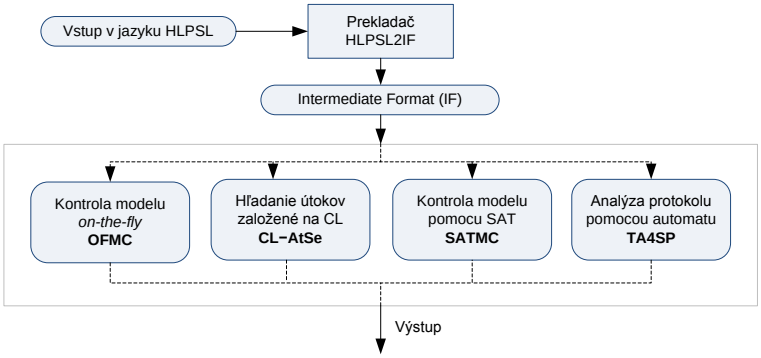
Aktuálna verzia: 1.1

Autor: Alessandro Armando, Yannick Chevalier, David Basin, Jorge Cuellar, ...

Licencia: Neuvedená

Podporované OS: Linux, Mac OS (Software) / všetky OS (Webové rozhranie)

AVISPA je projekt založený európskou komisiou ako súčasť programu *Information Society Technologies Programme* a vznikol 1.1.2003. V rámci projektu AVISPA vznikol rovnomenný nástroj na tvorbu a analýzu bezpečnostných protokolov [18].

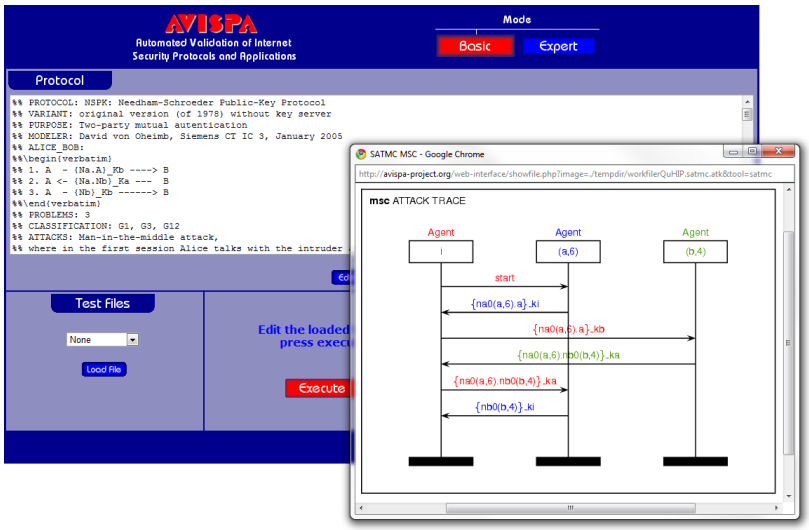


Obrázek 3.1: AVISPA architektúra

Nástroj sa skladá z niekoľkých komponent. Jeho architektúra je znázornená na obr.3.1. Na vstupe očakáva popis protokolu v jazyku HLPSL (vrátane vlastností, ktoré chceme verifiko-

vať). Tento vstup následne prevedie pomocou prekladača *HLPSL2IF Translator* do ekvivalentnej špecifikácie vo formálnom jazyku IF (Intermediate Format). Takto transformovanú špecifikáciu odovzdá na vstup jednotlivým komponentám. Každá komponenta implementuje rôzne techniky, ktoré prehľadavajú odpovedajúci stavový priestor a hľadajú stavy, ktoré predstavujú útoky na špecifikované vlastnosti (model checking). Výstupný formát komponent je presne definovaný a udáva či bol problém nájdený, prípadne či nastala chyba [24].

Nástroj existuje v dvoch verziách. Software stiahnuteľný z domovskej stránky určený pre Linux a Mac OS alebo ako webové rozhranie nachádzajúce sa na domovskej adrese.



Obrázek 3.2: Needham–Schroeder Public Key protokol v programe AVISPA

Výhody:

- Rôzne komponenty určené k analýze protokolu
- Multiplatformný nástroj (webové rozhranie)
- Vizualizácia útoku na protokol

Nevýhody:

- Nie je možné vizualizovať priebeh protokolu

3.2 GRASP

Adresa: Nezistená

Aktuálna verzia: Nezistená

Autor: Air Force Academy (VISE)

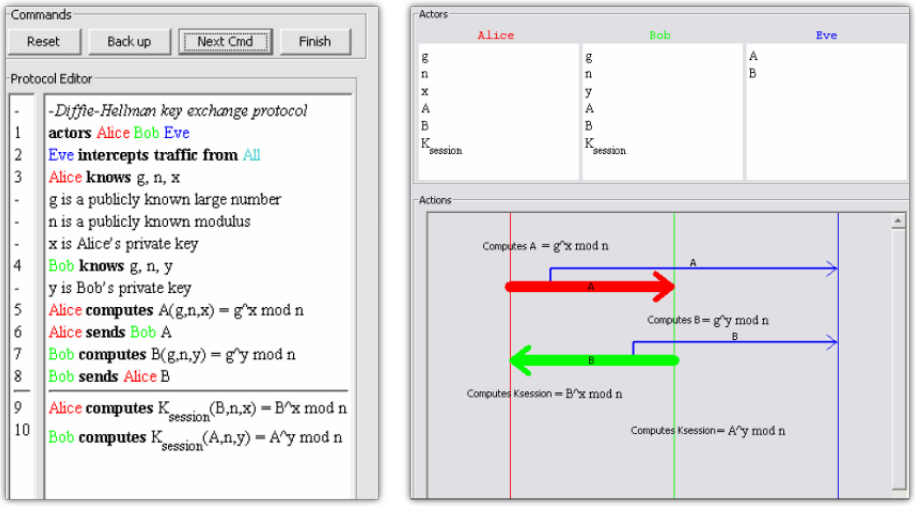
Licencia: Nezistená

Podporované OS: Windows, Linux, Mac OS (Java)

GRASP (GRaphical Aid for Security Protocols) je nástroj určený k interaktívnej vizualizácií bezpečnostných protokolov. Cieľom pri návrhu nástroja bolo graficky ilustrovať bezpečnostné protokoly a umožniť interaktívne meniť parametre (on-the-fly) s cieľom pozorovať zmeny. Zároveň musel byť dostatočne jednoduchý a názorný. Primárne mal byť nasadzovaný ako učebná pomôcka [23].

GRASP je napísaný v jazyku Java (Swing) čo z neho robí multi-platformný nástroj. Pozostáva z troch hlavných častí: editačné okno (zadávanie vstupu), oblasť okna s výsledkami, kde sa zobrazujú účastníci a ich znalosti a časový diagram znázorňujúci interaktivitu medzi účastníkmi (posielanie správ). Má vlastný jazyk pre popis protokolov s veľmi jednoduchou syntaxou. V každom kroku simulácie má používateľ možnosť meniť parametre prostredia, počet účastníkov, správy a pozorovať zmeny.

Program je vyvíjaný v rámci Air Force Academy (VISE) a je voľne dostupný [23].



Obrázek 3.3: Diffie–Hellman protokol v programe GRASP [23]

Výhody:

- Interaktívny nástroj s podporou *on-the-fly* zásahov do simulácie
- Multi-platformný nástroj programovaný v Jave

Nevýhody:

- Nedostupnosť nástroja na internete (v čase písania práce nebol dostupný)

3.3 Scyther

Adresa: `http://people.inf.ethz.ch/cremersc/scyther/index.html`

Aktuálna verzia: 1.0-beta7

Autor: Cas Cremers

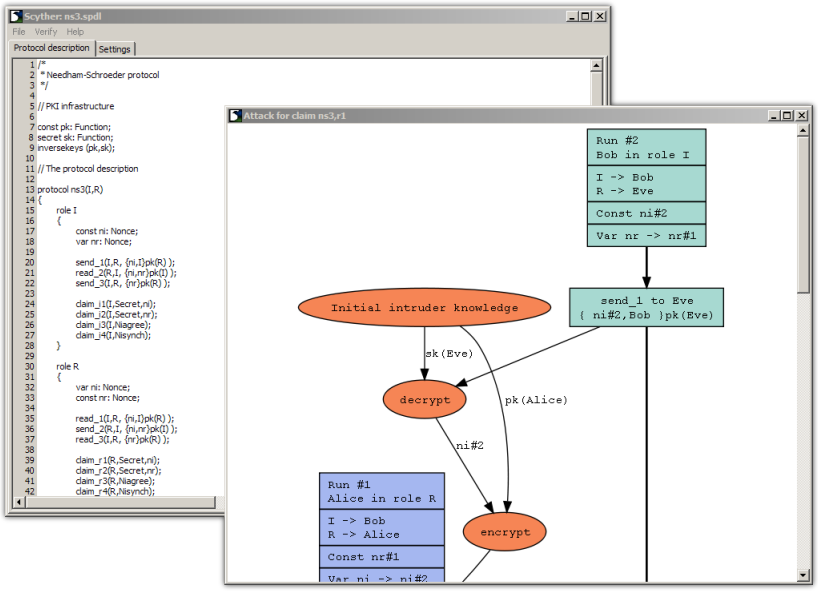
Licencia: Nešpecifikovaná (Voľne stiahnuteľné)

Podporované OS: Windows, Linux, Mac OS

Scyther je voľne stiahnuteľný nástroj, určený predovšetkým na verifikáciu a analýzu bezpečnostných protokolov. Je postavený na algoritme, ktorý poskytuje zhustenú reprezentáciu (nekonečnej) množiny stôp. To pomáha nástroju pri analýze tried možných útokov a chovania protokolu. Nástroj umožní dokazovať správnosť bližšie nešpecifikovaného počtu spojení. Dokáže pracovať paralelne s niekoľkými (pod)protokolmi [6].

Program je napísaný v jazyku Python a má konzolové ovládanie ako aj grafické rozhranie. Ako vstup očakáva popis protokolu v jazyku SPDL. Program dokáže zistiť, či sú dodržané bezpečnostné požiadavky v protokole prípadne tieto bezpečnostné požiadavky generovať a následne overiť. Dokáže analyzovať protokol na základe interakcie účastníkov.

Program nedokáže pracovať interaktívne a nedokáže za behu meniť podmienky. Je nutné upraviť zdrojový súbor (napr. vo vstavanom editore) a následne spustiť analýzu. Vzhľadom na jeho rýchlosť, názornosť a jednoduchosť sa hodí ako učebná pomôcka.



Obrázek 3.4: Needham–Schroeder protokol v programe Scyther

Výhody:

- Rýchlosť kompilácie
- Podpora veľkého počtu komunikačných spojení (neobmedzený počet [6])

- Pracuje paralelne s niekoľkými (pod)protokolmi
- Jednoduchosť použitia a názornosť pri zobrazení činnosti protokolu

Nevýhody:

- Nutnosť kompilácie vstupu pred samotnou analýzou
- Nemožnosť meniť parametre protokolu za behu

3.4 SPAN

Adresa: `http://www.irisa.fr/celtique/genet/span/`

Aktuálna verzia: 1.6

Autor: Yann Glouche, Thomas Genet, Olivier Heen, Erwan Houssay, Ronan Saillard

Licencia: LGPL

Podporované OS: Windows, Linux, Mac OS

SPAN nástroj je voľne dostupný pod licenciou LGPL. Je postavený na verifikačnom nástroji AVISPA a pridáva hlavne podporu animácie [7]. Na vstupe očakáva protokol popísaný v jazyku HLPSL alebo CAS+. Zo vstupu v jazyku CAS+ si automaticky vytvorí odpovdajúci HLPSL vstup. Tento vstup ďalej spracuje a výsledkom je postupnost správ posielaných v protokole.

Nástroj následne dokáže prehľadne zobrazovať správy v podobe sekvenčného diagramu. Dokáže krokovať priebeh komunikácie. Používateľ si može nastaviť sledované premenné (napr. *nonce*, A, B, ...) a tie počas animácie sledovať. Podporuje vetvenie komunikácie a v každom kroku ponúkne ďalšie možnosti, ktorým sa komunikácia bude uberať. Výslednú postupnosť správ (interný formát programu) možno uložiť vo formáte postscript alebo PDF a v budúcnosti kedykoľvek načítať.

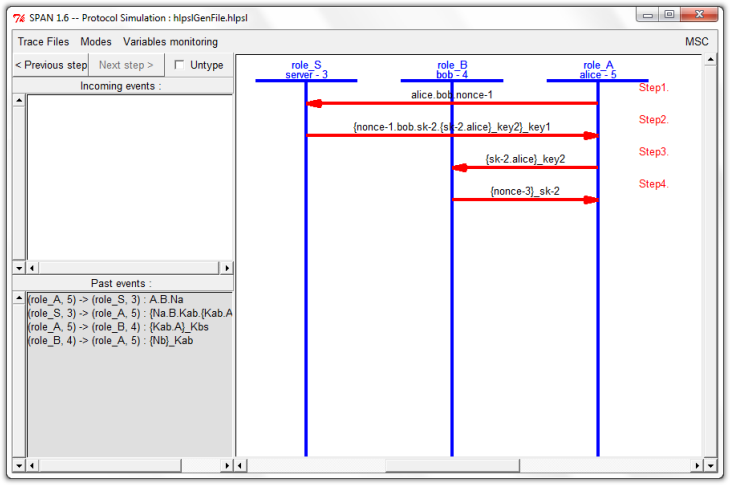
Program dokáže pracovať interaktívne a krokovať priebeh simulácie. Je vhodný ako učebná pomôcka.

Výhody:

- Ineraktivita počas simulácie
- Rýchla kompilácia
- Názornosť zobrazenia protokolu

Nevýhody:

- Nutnosť kompilovať vstup do interného formátu programu
- Mierne neprehladné GUI



Obrázek 3.5: Needham–Schroeder Shared Key protokol v programe SPAN

3.5 SPEAR 2

Adresa: `http://www.cs.uct.ac.za/Research/DNA/SPEAR2/index.html`

Aktuálna verzia: 1.0.0.25

Autor: Elton Saul

Licencia: Neuvedená (Voľne stiahnuteľné)

Podporované OS: Windows

SPEAR 2 je nástroj na návrh a analýzu bezpečnostných protokolov. Vychádza z pôvodného SPEAR projektu a hlavným cieľom je poskytnúť univerzálny framework, ktorý by umožnil efektívne navrhovať bezpečnostné protokoly. Kombinuje formálnu špecifikáciu, analýzu výkonnosti a bezpečnosti a automatické generovanie kódu. Využíva upravenú verziu GNY logiky (autentizačná logika) doplnenú o digitálne podpisy a certifikáty [22].

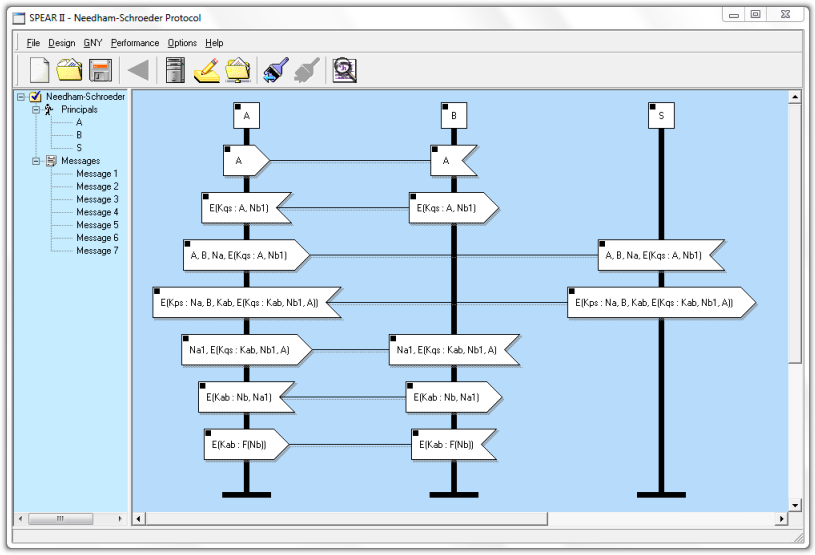
Nástroj poskytuje prehľadné používateľské rozhranie pre pohodlné vytvorenie nového alebo existujúceho protokolu. Podporuje ASN.1 zápis (popis správ). Podporuje načítanie a ukladanie do interného formátu (špecifický pre SPEAR2). Umožňuje export protokolu do formátu LaTeX, PROLOG alebo do textového súboru. Pre grafickú prezentáciu využíva sekvenčný diagram. Program neumožňuje interaktívnu simuláciu. Je vhodný predovšetkým pre fázu návrhu a verifikácie bezpečnostných protokolov.

Výhody:

- Kvalitné GUI a jednoduché vytváranie protokolov

Nevýhody:

- Nepodporuje žiadny z dostupných jazykov pre popis protokolov
- Neposkytuje možnosti vizualizácie útokov



Obrázek 3.6: Needham–Schroeder protokol v programe SPEAR 2

3.6 NS2

Adresa: `http://www.isi.edu/nsnam/ns/`

Aktuálna verzia: 2.34

Autor: M.Sc. Frida Eng, Ph.D. Fredrik Gunnarso

Licencia: GNU GPL

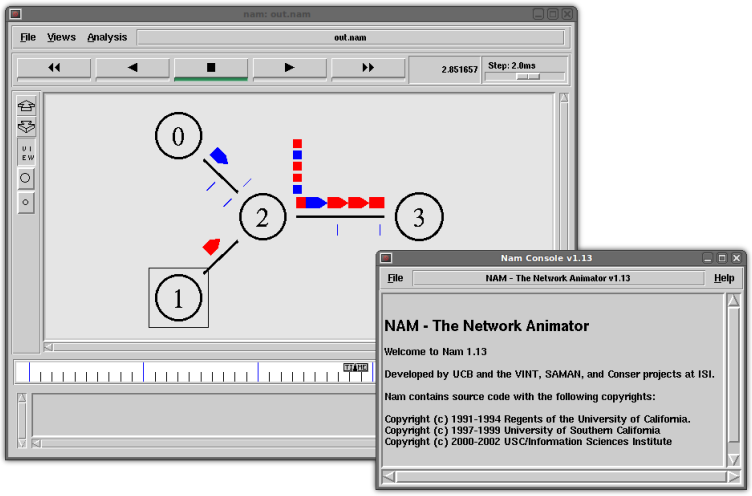
Podporované OS: Windows (Cygwin), UNIX, Linux, Mac OS

Vývoj NS2 vychádza z projektu *REAL network simulator*. Je určený na simuláciu smerovacích a multicastových protokolov. Rovnako podporuje simuláciu známych protokolov ako sú TCP, UDP, RTP, SRM. Program je napísaný v jazyku C++, Tcl, OTcl (Object Tcl). Je voľne šíriteľný pod licenciou GNU GPL, objektovo-orientovaný a poskytuje prostriedky pre špecifikáciu vstupných dát aj pre analýzu a prezentáciu výsledkov [18].
Program očakáva vstup v jazyku Tcl. V skripte je nutné popísať sieťovú topológiu a komunikačný protokol (posielanie a prijímanie správ).

Program je modulárny a podporuje tvorbu (programovanie) nových modulov. Pri inštalácii je potrebné kompilovať zdrojové kódy a vyžaduje množstvo závislostí. Jeho používateľské rozhranie (založené na OTcl) je zastaralé, obsahuje však všetky potrebné nástroje pre tvorbu simulovaného prostredia. Simulácia umožňuje krokovanie a rozhranie obsahuje prehľadnú časovú os.

Výhody:

- Voľne dosupný pod licenciou GNU GPL a možnosť prispieť k vývoju prostredníctvom modulov



Obrázek 3.7: Simulácia sieťovej komunikácie v NS2

- Podporuje široké možnosti tvorby simulovaného prostredia

Nevýhody:

- Zastaralé GUI
- Neobsahuje nástroje pre verifikáciu bezpečnostných nástrojov ako ostatné uvedené nástroje.

Kapitola 4

Implementácia bezpečnostných protokolov

V tejto kapitole odsimulujem vhodne zvolené bezpečnostné protokoly pomocou nástrojov uvedených v kapitole predchádzajúcej. Cieľom bude demonštrovať spôsoby implementácie.

Ako príklad som vybral protokoly *Needham-Schroeder Public Key* a *Yahalom*. Hlavným kritériom pri výbere bola zložitosť protokolu, tj. je vhodné aby zložitosť ich návrhu pokryla možnosti nástrojov. V protokole by mali vystupovať viacerí účastníci. Rovnako je vhodné aby posielané správy obsahovali identity subjektov, hodnoty nonce a kľúče (symetrické, verejné, súkromné). Ďalším významným kritériom je existencia zraniteľností. Jednotlivé nástroje by mali byť schopné odhaliť a znázorniť potenciálny útok.

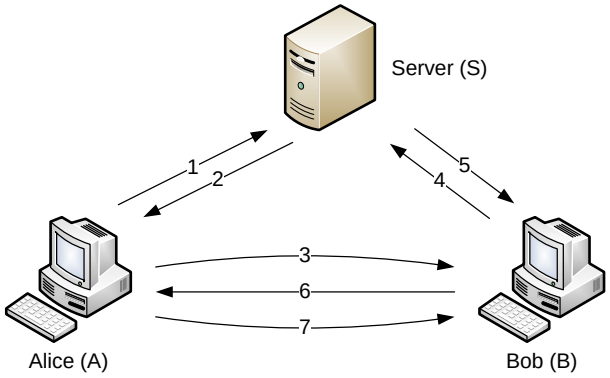
V druhej časti kapitoly porovnám nástroje z pohľadu implementácie vrátane možností, ktoré ponúkajú pri zadávaní vstupu.

4.1 Needham-Schroeder Public Key protokol

Jedná sa o verziu Needham-Schroeder protokolu založenú na asymetrickom šifrovaní. Poskytuje vzájomnú autentifikáciu subjektov použitím dôveryhodného kľúča serveru a verejných kľúčov. Server distribuuje kľúče na požiadanie. Protokol bol navrhutý ku komunikácií cez nezabezpečenú sieť [13]. Autormi je dvojica Roger Needham a Michael Schroeder.

Formálny popis protokolu

$$
\begin{array}{lll}
1. & A \rightarrow S & : A, B \\
2. & S \rightarrow A & : \{K_{pb}, B\}_{K_{ss}} \\
3. & A \rightarrow B & : \{N_a, A\}_{K_{pb}} \\
4. & B \rightarrow S & : B, A \\
5. & S \rightarrow B & : \{K_{pa}, A\}_{K_{ss}} \\
6. & B \rightarrow A & : \{N_a, N_b\}_{K_{pa}} \\
7. & A \rightarrow B & : \{N_b\}_{K_{pb}}
\end{array}
\tag{4.1}
$$



Obrázek 4.1: Needham-Schroeder Public Key protokol

Známy útok Protokol je náchylný na útok typu *man-in-the-middle*. Útočník I sa môže vydávať za účastníka A tak, že ho donúti iniciovať s ním nové spojenie. Správy preposiela smerom k B (predstiera identitu A). V dôkaze (4.2) sa predpokladá komunikácia len medzi A, B, I. Rovnako sa predpokladá znalosť kľúčov (K_a, K_b, K_i) u jednotlivých účastníkov [13, 11].

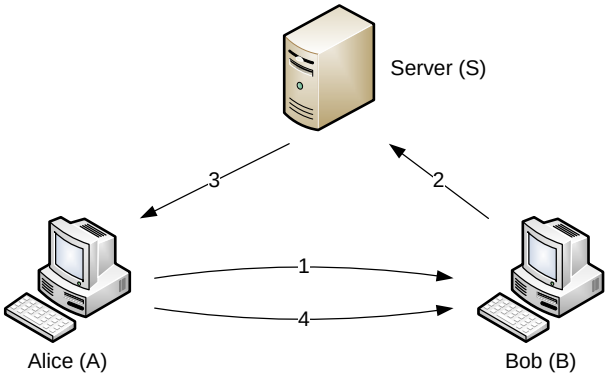
$$\begin{array}{lcl} 1.3. & A \rightarrow I & : \{N_a, A\}_{K_{pi}} \\ 2.3. & I(A) \rightarrow B & : \{N_a, A\}_{K_{pb}} \\ 2.6. & B \rightarrow I(A) & : \{N_a, N_b\}_{K_{pa}} \\ 1.6. & I \rightarrow A & : \{N_a, N_b\}_{K_{pa}} \\ 1.7. & A \rightarrow I & : \{N_b\}_{K_{pi}} \\ 2.7. & I(A) \rightarrow B & : \{N_b\}_{K_{pb}} \end{array} \tag{4.2}$$

Modifikácia *Lowe* túto chybu odstraňuje pridaním identity B do správy [11]:

$$2.6 \qquad B \rightarrow A \qquad : \{N_a, N_b, B\}_{K_a} \tag{4.3}$$

4.2 Yahalom protokol

Poskytuje vzájomnú autentifikáciu účastníkov a distribúciu tajného kľúča. Bol navrhnutý pre použitie cez nezabezpečenú sieť (napr. Internet). Využíva symetrické šifrovanie. O distribúciu tajného kľúča sa stará dôveryhodný arbiter (server) [17].



Obrázek 4.2: Yahalom protokol

Formálny popis protokolu

$$
\begin{array}{lll}
1. & A \rightarrow B & : A, N_a \\
2. & B \rightarrow S & : B, \{A, N_a, N_b\}_{K_{bs}} \\
3. & S \rightarrow A & : \{B, K_{ab}, N_a, N_b\}_{K_{as}}, \{A, K_{ab}\}_{K_{bs}} \\
4. & A \rightarrow B & : \{A, K_{ab}\}_{K_{bs}}, \{N_b\}_{K_{ab}}
\end{array}
\tag{4.4}
$$

Známy útok K Yahalom protokolu nebol publikovaný žiadny útok [17].

4.3 Spôsob implementácie

Z kapitoly 3 sú dostupné a zároveň využiteľné pre simuláciu bezpečnostných protokolov nástroje AVISPA, SPAN, Scyther, SPEAR 2. Každý z nich sa vyznačuje vlastným implementačným jazykom. Prehľad podporovaných jazykov je v tabuľke 4.1.

	AVISPA	**SPAN**	**Scyther**	**SPEAR 2**
Jazyk	HLPSL	HLPSL, CAS+	SPDL	SPR

Tabulka 4.1: Prehľad jazykov podporovaných v nástrojoch

Rozhodol som sa preto ku každému nástroju pristupovať jednotlivo a ako základ implementovať zvolené protokoly.

4.3.1 Implementácia v AVISPA

Nástroj podporuje implementáciu jedine v jazyku HLPSL. Jazyk berie ako základ popis rolí. Každá z rolí popisuje aké informácie účastník pozná na začiatku, začiatočný stav a prechody (zmena stavu) [1]. V podstate sa jedná o prepis stavového automatu do textovej podoby.

Prijatie alebo odoslanie správy vyvolá prechod do iného stavu. Reláciu prechodu zapíšeme ako `=|>` a zjednotenie ako `/\`.

Začneme s deklaráciou samotnej role, ktorá má tvar

```
role alice (A, B: agent,
            Ka, Ks: public_key,
            KeyRing: (agent.public_key) set, % Množina kľúčov
            Snd, Rcv: channel(dy)) % Komunikačný kanál
```

kde parameter `dy` znamená model útočníka. Pokračujeme jej správaním a lokálnymi premennými. Premenné musia začínať veľkým písmenom a konštanty malým.

```
played_by A def=
  local State : nat, % Prirodzené číslo
        Na, Nb: text, % nonce
        Kb: public_key
  init State := 0 % Počiatočný stav
  transition
   ...
end role
```

Prechod teda znamená sériu počiatočných podmienok a akcií spustených po jeho vykonaní. Prechody možeme rozlíšiť identifikátorom (v nasladujúcom príklade `ask.`)

```
   ask.    State  = 0 /\ Rcv(start) /\ not(in(B.Kb', KeyRing))
       =|> State':= 1 /\ Snd(A.B)
```

Takto definovanú rolu spojíme spolu s ostatnými do kompozície (session) protokolu a ďalej do kompozície prostredia. Prostredie definuje aj rolu útočníka.

```
composition
     /\_{in(A.B.Ka.Kb,Instances)} (alice(A,B,Ka,Ks,KeySet(A),Snd,Rcv)
       /\ bob(A,B,Kb,Ks,KeySet(B),Snd,Rcv))
```

Na záver definujeme množinu cieľových bezpečnostných požiadavkov (tajnosť, autentifikácia, ...) `goal` a najvyššiu kompozíciu `environment()`.

Komentáre až do konca riadku začínajú `%`.

4.3.2 Implementácia v SPAN

U predchádzajúceho nástroja som popísal implementáciu v HLPSL, preto sa teraz budem zaoberať jazykom CAS+. Oproti HLPSL sa vyznačuje jednoduchosťou zápisu. Zdrojový kód popisujúci protokol je prehľadný a skladá sa z niekoľkých častí. V úvode deklarujeme názov protokolu (prvý riadok).

```
protocol NeedhamSchroederPublicKey;
```

Nasleduje deklarácia identifikátorov a premenných (jednotlivé entity a kľúče). Deklarovať možno `public_key` (verejný kľúč), `symetric_key` (symetrický kľúč), `user` (účastník), `function` (jednocestná funkcia napr. hash) a `number` (obecné dáta, nonce, záznam, text).

```
identifiers
  A,B,S      : user;
  Na,Nb      : number;
  KPa,KPb,KPs : public_key;
```

V ďalšej časti zapíšeme postupnosť správ. Táto časť tvorí jadro popisovaného protokolu. Základný tvar je

$$(i.\ S_i \rightarrow_i R_i : M_i)_{1 \leq i \leq n}$$

kde $i \leq n$ je číslo kroku [21]. Privátny kľúč končí apostrofom napr. `Kab'`. Zápis je podobný formálnemu zápisu protokolov.

```
messages
  1. A -> S : A,B
  2. S -> A : {KPb, B}KPs'
  3. A -> B : {Na, A}KPb
  4. B -> S : B,A
  5. S -> B : {KPa, A}KPs'
  6. B -> A : {Na, Nb}KPa
  7. A -> B : {Nb}KPb
```

Svoj význam má aj šípka znázorňujúca zaslanie správy. Bežný kanál (Dolev-Yao) zapíšeme ascii znakom `->`, kanál chránený proti zápisu a čítaniu `=>` a kanál chránený proti zápisu `~>` [21].

Časť popisujúca znalosti účastníkov, tj. všetky dáta, ktoré sú z pohľadu účastníkov známe ešte pre začiatkom priebehu protokolu.

```
knowledge
  A : A,B,S,KPa,KPb,KPs;
  B : A,B,S,KPa,KPb,KPs;
  S : A,B,S,KPa,KPb,KPs;
```

V časti *session instances* sa priraďuje hodnota k premenným. V praxi to znamená, že je možné popísať rôzne systémy v ktorých beží protokol. Rovnako je možné v tejto časti vytvoriť *session* s útočníkom I. K útočníkovi sa vzťahuje aj deklarácia znalostí podobne ako pri účastníkoch.

```
session_instances
  [A:alice,B:bob,S:server,KPa:pk1,KPb:pk2,KPs:pk3];
```

V poslednej časti uvedieme ciele (vlastnosti), ktoré chceme dokázať. Existujú tri typy: utajenosť, autentifikácia a slabá autentifikácia.

```
goal
  A authenticates B on Na;
  B authenticates A on Nb;
```

Komentáre do konca riadku začínajú `%`.

4.3.3 Implementácia v Scyther

Nástroj využíva vlastný jazyk SPDL (Security Protocol Descritpion Language). Na prípone súboru nezaleží. Súbory môžeme do seba zanorovať, čo sa hodí pri modelovaní zložitejších protokolov

```
include  'filename';
```

Scyther je postavený na manipulácii s výrazmi (*term*). Atomický výraz je pritom ľubovolný identifikátor, typicky reťazec alfanumerických znakov [5]. K dispozícii sú základné preddefinované typy `Function` (kľúče, hash), `Agent` (účastníci), `Nonce` alebo `Ticket` a udalosti `Secret` (tajomstvo).

Na úvod teda deklarujeme kľúče

```
const pk: Function;
secret sk: Function;
inversekeys(pk,sk); // Asymetrický pár kľúčov
```

kde `Const` značí globálnu konštantu. Nasleduje popis správania protokolu, ktorý má (v prípade Needham Schroeder-PK) troch účastníkov

```
protocol NeedhamSchroederPK(I,R,S) {}
```

V rámci protokolu definujeme správanie jednotlivých účastníkov. Hodnoty, ktoré účastník pozná, zapisujeme na úvod. Komunikácia prebieha typicky posielaním a prijímaním správ. Správy sú číslované aby ich interpret dokázal správne zoradiť do sekvenčného diagramu. Najdôležitejšiu časť predstavujú bezpečnostné požiadavky (určujú kontrolované vlastnosti).

```
role I {
  const Ni: Nonce; // Globálna premenná
  var Nr: Nonce; // Lokálna premenná

  send_1(I,S,(I,R)); // Správa od I k S, obsahujúca (I,R)
  read_2(S,I, {pk(R), R}sk(S));
  ...
  claim_I1(I,Secret,Ni);
  claim_I2(I,Secret,Nr);
  claim_I3(I,Nisynch);
}
```

V záverečných nastaveniach protokolu uvedieme dôveryhodných účastníkov, nedôveryhodných útočníkov (cieľ útočníka) a ohrozené dáta.

```
const Alice,Bob,Server,Compromised: Agent;
untrusted Compromised;
const nc: Nonce;
compromised sk(Compromised);
```

Jazyk je *case-sensitive*. Komentáre možno zapisovať viacerými spôsobmi podobne ako v jazyku C. Jednoriadkový komentár `#` alebo `//`, prípadne viacriadkový komentár `/*  */`. Biele znaky (nový riadok, medzera) sú ignorované.

4.3.4 Implementácia v SPEAR 2

Nástroj sa od ostatných odlišuje hlavne možnosťou modelovať protokoly priamo pomocou používateľského rozhrania. Táto vlastnosť urýchľuje a zjednodušuje celý proces. Samotný vstupný súbor obsahujúci popisovaný protokol má niekoľko desiatok kilobajtov, preto sa zameriam hlavne na implementáciu pomocou GUI. Ako príklad uvediem časti kódu zdrojového súboru.

Na úvod modelujeme účastníkov (*Design* → *AddCanvasObject* → *Principal*). Na úrovni zdrojového kódu majú účastníci tvar

```
PRINCIPAL {
    NAME = 'A'
}
```

Prejdeme k modelovaniu vlastných správ (*Design* → *AddCanvasObject* → *Message*). V tomto kroku už máme k dispozícii objekty (účastníkov) a teda pri vytváraní obsahu správy vyberáme z ponuky existujúcich objektov (*Add* → *Existing* → *Component* → *A*). Ak narazíme na neexistujúci objekt je nutné si ho najprv vytvoriť (napr. symetrické/asymetrické kľúče). Situácia je rovnaká na úrovni zdrojového kódu. Objekty správy vytvoríme ako

```
PAYLOAD (mcComponent, 15042384) {
    ID = 'A'
    ASN.1 = ''
}
```

kde prvý parameter je typ objektu (napr. `mcComponent`, `mcTimeStamp`, `mcPrivKey`, `mcPubKey`, `mcSharedSecret`, `mcPrivEncryption`) a druhý parameter je identifikátor, na ktorý sa neskôr budeme odkazovať. Ako je možné vidieť program podporuje ASN.1 zápis.

Postupne môžeme prejsť k modelovaniu vlastných správ. Zápis správy obsahuje identifikáciu odosielateľa, príjemcu, akciu pred/po odoslaní/prijatí a identifikátory samotných objektov. Príklad správy

```
MESSAGE {
    SENDER = 'A'
    RECEIVER = 'S'
    ACTIVE_SETTINGS = 'subprotocol'
    ACTION {
        SENT_PREACTION = ''
        SENT_POSTACTION = ''
        RECEIVED_PREACTION = ''
        RECEIVED_POSTACTION = ''
    }
    ROOT_PAYLOAD {
        ID = 'Message 1'
        DESCRIPTOR = ''
    }
```

```
    COMPONENTS {
        TOP_LEVEL: 15042384 15238424
    }
}
```

Po vytvorení správ (správanie protokolu) musíme definovať predpoklady a ciele protokolu. Program pracuje s GNY logikou a definícia zahrňuje *beliefs* a *possesions* u jednotlivých účastníkov ako na strane predpokladov tak na strane cieľov. Zápis v zdrojovom súbore je značne rozsiahly a pracný. Osvedčilo sa mi modelovanie na úrovni GUI. Definovanie GNY pravidiel prebieha rovnakým postupom ako vytváranie správ. Predpoklady existujú zvlášť pre každého účastníka

- *Fresh Components* – Aktuálnosť objektu
- *Recognizable Components* – Znalosti objektu
- *Suitable Public Keys* – Znalosť kľúčov
- *Trustworthy Principals* – Dôveryhodný účastníci
- *Jurisdiction* – Dôvera v objekt, ktorý je dôveryhodný pre iného účastníka
- *Suitable Secret* – Tajomstvo medzi objektami

4.4 Výsledky implementácie

Každý z nástrojov dovoľuje popísať požadovaný protokol pomocou jazyka. Výnimkou je nástroj SPEAR 2, ktorý umožňuje modelovanie aj na úrovni GUI. Jazyky sú rôzne ale majú spoločné rysy. Sémantika je podobná, no odlišujú sa hlavne v syntaxi. Základom v jazyku HLPSL a SPDL sú role a ich popis vrátane správania. Pri jazyku CAS+ naopak deklarujeme entity a správanie oddelene. Implementácia v SPR zahrňuje hierarchiu objektov (entity, obsah správy, šifrovanie, správa) a GNY pravidlá.

Kedže nástroje sú primárne určené k simulácii bezpečnostných protokolov, ich jazyky podporuju všetky vlastnosti ako sú symetrické/asymetrické šifrovanie (kľúče), *nonce*, časové známky, ... Môžeme pracovať s rôznymi rolami (útočník, server, účastníci).

	AVISPA	**SPAN**	**Scyther**	**SPEAR 2**
Jazyk	HLPSL	HLPSL, CAS+	SPDL	SPR
GUI režim	○	○	○	•
Textový režim	•	•	•	○
Vlastnosti	Definícia rolí Stavový automat	Popis správania Popis entít	Definícia rolí	Popis objektov GNY logika ASN.1
○ - nepodporuje, • - podporuje				

Tabulka 4.2: Výsledky porovnania implementácie

Kapitola 5

Charakteristiky nástrojov

Od spôsobov implementácie prejdem v tejto kapitole ku charakteristikám používania jednotlivých nástrojov. To zahrňuje najmä spôsoby inštalácie a ovládania.

5.1 AVISPA

5.1.1 Inštalácia

Program existuje v dvoch verziách. Jedna verzia je dostupná pre operačný systém Linux a Mac OS, druhá je dostupná prostredníctvom internetového prehliadača. Z pohľadu inštalácie je jednoduchšie použiť online verziu. Nevýhodou je závislosť na internetovom pripojení a dostupnosti služby.

5.1.2 Ovládanie

Používateľské rozhranie online verzie programu môžeme rozdeliť na niekoľko častí: editačné okno, panel nástrojov, práca so súborom a voľba režimu.

Editačné okno Tento panel zastáva viacero funkcií. V základnom režime zobrazuje načítaný protokol. Pre editáciu sa musíme prepnúť do editačného režimu (*edit*). Po dokončení uložiť (*save*), čím sa opäť vrátime do základného režimu. Uloženie je nevyhnutné pre aplikovanie zmien. Poslednou funkciou je stavový a chybový výpis. Ten sa objaví po použití niektorého zo vstavaných nástrojov.

Panel nástrojov Ponuka vstavaných nástrojov pre verifikáciu protokolu (obr. 3.1).

- HLPSL2IF - Prekladač z jazyka HLPSL do ekvivalentného formálneho jazyka *IF* (nízkoúrovňový popis). Prevod je nevyhnutný pre ďaľšie spracovanie.
- OFMC - *On-the-fly Model-Checker* verifikuje protokol skúmaním prechodov ohraničenej *session* (model), popísaných v jazyku IF [24].
- CL-AtSe - *Constraint-Logic-based Attack Searcher* aplikuje obmedzujúce podmienky stanoveného počtu *session* k verifikácií protokolu [24].
- SATMC - *SAT Model-Checker* redukuje vstupný problém do sekvencie pre *SAT solvers* [24].

- TA4SP - neohraničená verifikácia protokolu aproximovaním útočníkových znalostí [24].

Práca so súbormi Načítanie predpripravených alebo vlastných zdrojových súborov.

Režim Voľba medzi základným a expertným režimom. Rozdiel spočíva v tom, že v expertnom režime je dostupný panel nástrojov, zatiaľ čo v základnom sa po spustení simulácie zavolajú automaticky všetky vstavané nástroje počínajúc prekladom do *IF* (obr. 3.1).

Po odsimulovaní dostávame v editačnom okne suhrnný výstup od jednotlivých nástrojov s výsledokom verifikácie. Zároveň si môžeme u každého nástroja zobraziť podrobný výpis ako aj graficku podobu útoku (časový diagram).

5.2 SPAN

5.2.1 Inštalácia

Nástroj je distribuovaný v štyroch verziách. Binárne verzie pre operačné systémy Windows, Linux a Mac OS a v podobe zdrojových kódov. Pre svoj behu vyžaduje Tcl/Tk knižnice.

Windows Inštalačný súbor spolu s inštaláciou Tcl/Tk knižnice (vyžadovaná je minimálna verzia 8.3) je stiahnuteľný z domovskej stránky programu. Po inštalácii je potrebné pridať premennú prostredia (path) smerujúcu na adresár s Tcl/Tk

```
Path   %Path%;C:\PROGRA~1\Tcl\bin
```

a vytvoriť novú premennú k adresáru s nástrojom SPAN

```
SPAN   C:\SPAN
```

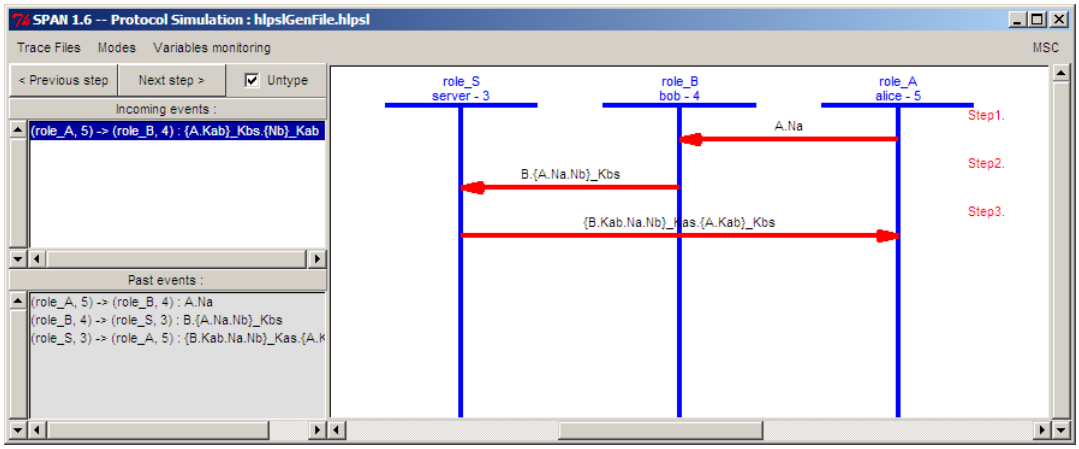
Linux Po stiahnutí programu z domovskej stránky, stačí rozbaliť obsah do ľubovolného adresára. Podobne ako na platforme Windows, požadovaná je prítomnosť Tcl/Tk (verzia 8.5). Následne stačí nastaviť premenné prostredia (path)

```
export   SPAN=/home/user/span
export   AVISPA_PACKAGE=/home/uset/span
```

5.2.2 Ovládanie

Ako už bolo spomenuté, nástroj vychádza z projektu AVISPA. Jeho základ, vrátane podporných nástrojov (obr. 3.1), je rovnaký. Poskytuje naviac interaktívnu vizualizáciu simulácie protokolu, útoku a útočníka.

V okne vizualizácie (obr. 5.1) sa na ľavej strane objavujú prichádzajúce udalosti. Po príchode sa simulácia zastaví a môžeme určiť, ktorým smerom bude priebeh pokračovať. Simuláciu možno krokovať dopredu aj dozadu (*Previous step*, *Next step*). Nastaviť sa dá sledovanie konkrétnych premenných (účastníkov, kľúčov, ...). Možeme nastaviť spôsob zobrazenia správ, čím sprehladníme sekvenčný diagram. Výsledný diagram môžeme uložiť a opätovne načítať.



Obrázek 5.1: Priebeh protokolu – Yahalom protokol

Simulácia útoku a simulácia útočníka ponúka rovnaké možnosti a rozloženie ovládacích prvkov. Rozdiel je v tom, že môžeme explicitne nastaviť a nechať zobraziť útočníkové vedomosti.

Oproti nástroju AVISPA dokáže SPAN načítať protokol v jazyku CAS+ a následne konvertovať do jazyka HLPSL. Počas môjho testovania sa u niektorých protokolov vyskytli problémy pri konverzii. Vo väčšine prípadov prebehla konverzia v poriadku.

5.3 SPEAR 2

5.3.1 Inštalácia

Nástroj je dostupný pre platformu Windows. Inštalačné súbory sú k dispozícii na domovskej stránke projektu. K analýze GNY, program vyžaduje prítomnosť SWI–Prolog (inštalačný súbor je rovnako dostupný na stránke projektu). V nastaveniach programu sa musí vyplniť cesta k prologu.

5.3.2 Ovládanie

Program disponuje širokými možnosťami ovládania. Najväčšiu časť okna tvorí plocha (*canvas*) s časovým diagramom protokolu. Jedná sa o abstraktný pohľad na modelovaný protokol. Na ľavej strane je stručný stromový prehľad vytvorených objektov.

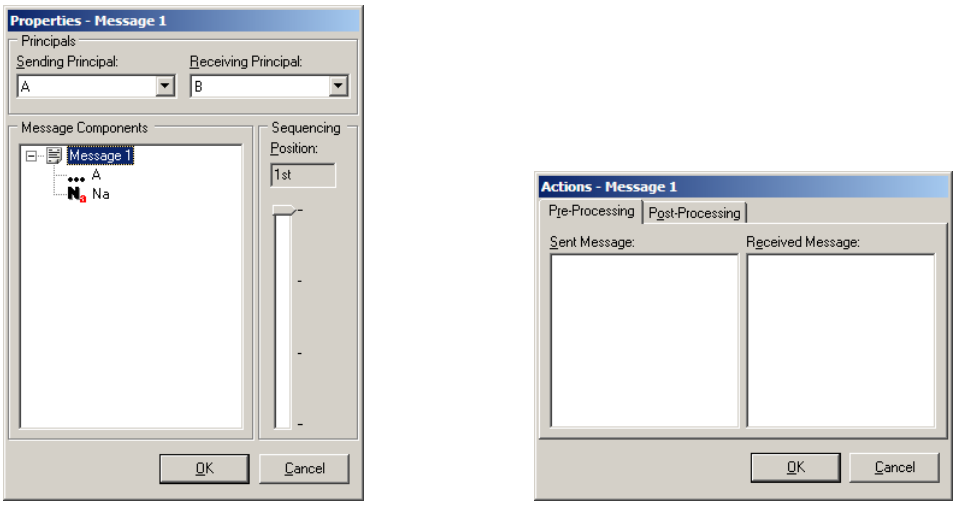
Priamo na ploche môžeme cez kontextové menu (*Add Canvas Object*) vytvoriť objekty (účastník, správa, podprotokol). V dialogu pre vytvorenie nového objektu nastavíme dodatočné vlastnosti ako napr. meno, príjemca, odosielateľ, obsah správy, poradie (obr. 5.2a). Pri vytváraní obsahu správy máme na výber vytvorenie nového objektu alebo použitie existujúcich. Objekty rozdeľujeme na dve skupiny

- *Terminal* - nonce, časové razítko, zdieľané tajomstvo, symetrické/asymetrické kľúče.
- *Non-terminal* - funkcia, hash, symetrické/asymetrické šifrovanie, skupina.

pričom terminálny objekt nemôže obsahovať ďalšie objekty.

Vyvolaním kontextovej ponuky môžeme správy zmazať (*Delete*), duplikovať (*Duplicate*), nastaviť akciu pred/po prijatí/odoslaní (obr. 5.2b) alebo zmeniť vlastnosti (*Properties*).

Účastníkovi môžeme pridať správu (*Add Message*) alebo zmeniť meno (*Edit name*).



(a) Vlastnosti správy

(b) Nastavenie akcie

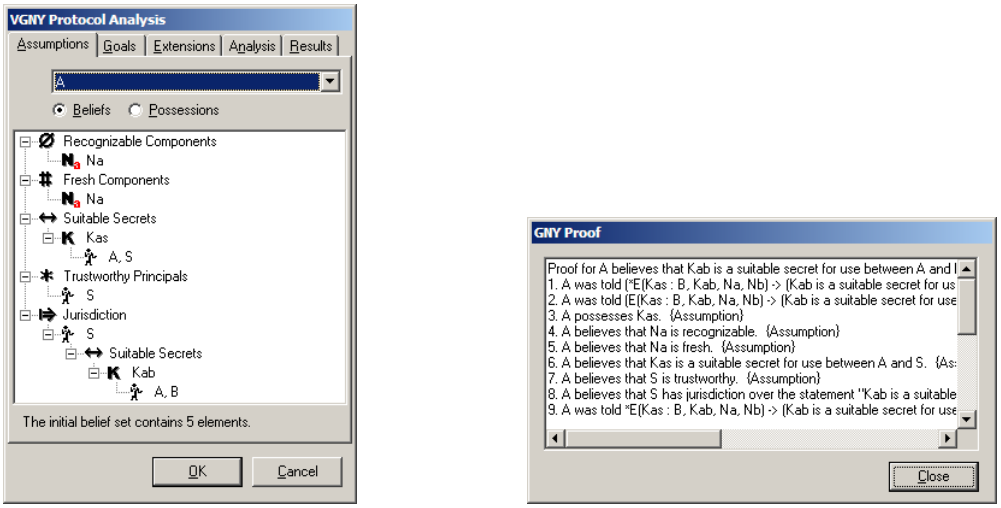
Obrázek 5.2: Nastavenie správy v SPEAR 2

Objekty môžeme navzájom presúvať myšou a meniť tak ich poradie. Program dokáže zvýrazniť konkrétne objekty cez funkciu *Design→Component Tracker*, čo pomôže pri modelovaní zložitých protokolov. Program dovoľuje vytvoriť podprotokol v ľubovolnom mieste.

Najdôležitejšia časť progamu je GNY analýza protokolu. V jednom dialogovom okne (*GNY→Analyze Protocol*), sú všetky potrebné nastavenia. Rovnako ako v prípade vytvárania správy aj tu sú objekty hierarchicky usporiadané.

- *Assumptions* - predpoklady u jednotlivých účastníkov (*beliefs and possessions*), tj. znalosti pred začiatkom komunikácie.
- *Goals* - cieľové požiadavky (*beliefs and possessions*) u každého účastníka.
- *Extensions* - pripojenie rozšírenia vybraným formuliam.
- *Analysis* - nastavenie ciest k prologu a pracovnému adresáru. Spustenie samotnej analýzy.
- *Result* - výsledky analýzy (obr. 5.3b). Prehľad a detaily valídnych a nesprávnych predpokladov a požiadavkov (*beliefs and possessions*).

Výsledné pravidlá GNY logiky ako aj formálny popis vytvoreného protokolu, môžeme zobraziť a exportovať do textového formátu alebo súboru LaTeX. Program dokáže vypočítať synchrónny a optimálny beh protokolu (*Calculate Synchronous rounds*, *Calculate Optimal rounds*).



(a) Predpoklady (b) Výsledky analázy

Obrázek 5.3: GNY analýza v SPEAR 2

5.4 Scyther

5.4.1 Inštalácia

Program (dostupný na stránkach projektu) je napísaný v jazyku Python, čo z neho robí multi-platformný nástroj. Vyžaduje inštaláciu Python, wxPython a GraphViz (odkazy sú na stránkach projektu Scyther). Knižnica GraphViz vykresluje grafy.

5.4.2 Ovládanie

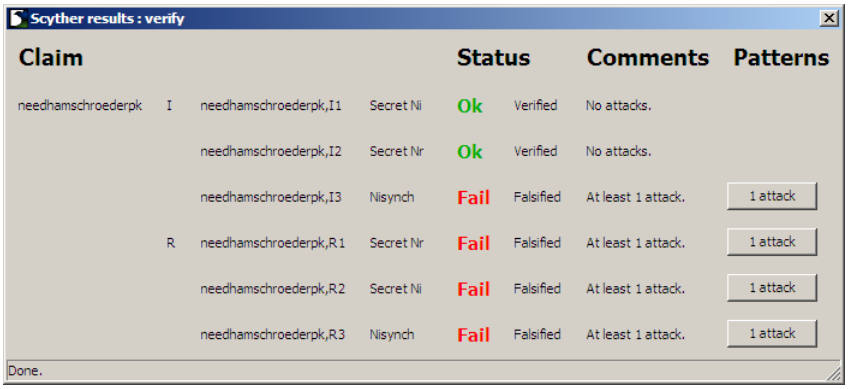
Na rozdiel od predchádzajúceho nástroja SPEAR 2, ponúka rozhranie Scyther menej možností. Hlavné okno má dve záložky: popis protokolu (*Protocol description*) a nastavenia (*Settings*).

Popis protokolu V tejto záložke môžeme editovať protokol. Protokol je popísaný v jazyku SPDL. Dôraz je kladený na bezpečnostné požiadavky (*claims*).

Nastavenia Hlavné nastavenia programu. Ovplyvňujú verifikáciu obmedzením počtu behov, maximálnym počtom vzorov a dodatočnými parametrami pre jadro programu. Možnosť zmeniť veľkost písma sa hodí najmä pri rozsiahlych protokoloch.

Ak máme výsledný protokol popísaný a nastavené hlavné paremetre, môžeme spustiť verifikáciu v menu *Verify→Verify protocol* (`F1`). Zobrazí sa nám okno (obr. 5.4), kde každý riadok predstavuje jednu požiadavku (*claim*). Na riadku je vypísaný v poradí: názov protokolu, rola (identifikácia účastníka), identifikátor požiadavky, typ požiadavky a parameter, stav informujúci o nájdenom útoku, komentár a tlačítko pre grafické zobrazenie útoku.

Často sa stáva, že program nenájde útok a končí s výsledkom *No attack within bounds*. Pri verifikačnom procese Scyther prehľadáva strom všetkých možností avšak je obmedzený

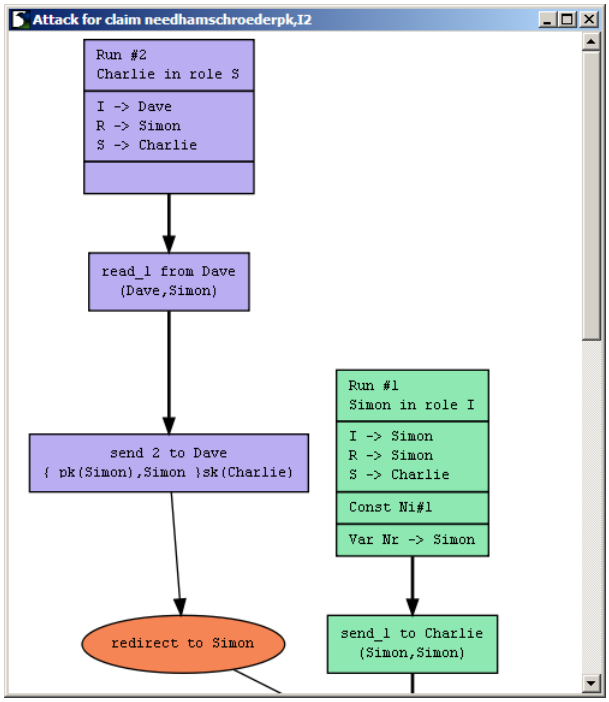


Obrázek 5.4: Výsledky verifikácie – Needham Schroeder protokol

počtom behov (nastavenia programu). Je dôležité zvoliť vhodnú hodnotu. Príliš veľká hodnota znamená viac času potrebného na verifikáciu. Malá hodnota nemusí nájsť útok.

Okrem verifikácie si môžeme nechať graficky zobraziť role cez menu *Verify→Characterize roles* (`F2`). Nové okno má rovnaký obsah ako v prípade verifikácie.

Ak sa rozhodneme zobraziť útok alebo niektorú z rolí, otvorí sa nové okno (obr. 5.5). Hlavnými prvkami sú obdĺžniky spojené šípkami. Šípky reprezentujú poradie. Obdĺžnik znázorňuje nový beh, udalosť behu a udalosť spôsobenú bezpečnostnou podmienkou. Vertikálna os predstavuje beh protokolu (výskyt role).



Obrázek 5.5: Znázornenie útoku – Needham Schroeder protokol

Kapitola 6

Demonstračné úlohy

Pre ďaľšie použitie som ako vhodný nástroj vybral Scyther. Nástroj je voľne dostupný. Vďaka svojej jednoduchosti a schopnosti vizualizovať protokoly (útoky) je vhodnou učebnou pomôckou. Program dokáže pracovať so symetrickými aj asymetrickými verziami protokolov a následne ich verifikovať. Prípadné útoky nad protokolom zobrazí graficky diagramom.

Obsahom kapitoly bude pre tento nástroj ako aj nástroje SPAN a SPEAR2 napísať sadu príkladov a demonstračných úloh. Cvičenia budú vychádzať z implementácie protokolov uvedených v kapitole 4 (*Needham-Schroeder Public Key*, *Yahalom*) doplnené o protokoly *Otway Rees* a *Denning-Sacco Shared Key*. Zároveň si kladú za cieľ zoznámiť čitateľa s problematikou bezpečnostných protokolov, ich simulácie a verifikácie konkrétnym nástrojom. Tieto úlohy nájdu uplatnenie pri výuke sieťovo orientovaných predmetov na fakulte FIT VUT.

6.1 Úloha č.1

Cieľ úlohy

- Zoznámiť sa s nástrojom Scyther.
- Vyskúšať si implementáciu symetrických a asymetrických bezpečnostných protokolov.
- Vyskúšať si verifikáciu bezpečnostných protokolov v nástroji Scyther.

Študijné materiály

- V tejto úlohe bude cieľom implementovať protokoly *Needham-Schroeder Public Key*, jeho modifikovanú verziu *Lowe*, *Yahalom*, *Otway Rees*, *Denning-Sacco Shared Key* a jeho modifikovanú verziu *Lowe*.
- Študijné materiály
  - Needham-Schroeder Public Key
    `http://www.lsv.ens-cachan.fr/Software/spore/nspk.html`
    Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995.

- Lowe's fixed version of Needham-Schroder Public Key
  `http://www.lsv.ens-cachan.fr/Software/spore/nspkLowe.html`
  Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995.
- Yahalom
  `http://www.lsv.ens-cachan.fr/Software/spore/yahalom.html`
  Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, february 1989.
- Otway Rees
  `http://www.lsv.ens-cachan.fr/Software/spore/otwayRees.html`
  D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.
- Denning-Sacco Shared Key
  `http://www.lsv.ens-cachan.fr/Software/spore/denningSacco.html`
  D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.
- Lowe modifikácia Denning-Sacco Shared Key
  `http://www.lsv.ens-cachan.fr/Software/spore/denningSaccoLowe.html`
  Gavin Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

Príprava prostredia

- Stiahnite inštalačný súbor z domovskej stránky projektu Scyther:
  `http://people.inf.ethz.ch/cremersc/scyther/install-windows.html`
- Rozbalte stiahnutý archív do ľubovolného adresára.
- Stiahnite a nainštalujte knižnicu GraphViz. Aktuálna verzia je dostupná na adrese `http://www.graphviz.org/Download_windows.php`.
- Nástroj Scyther je napísaný v jazyku Python. Nainštalujte potrebné knižnice Python a wxPython zo stránok
  `http://www.python.org/download/`
  `http://www.wxpython.org/download.php`.
- Spustite `scyther-gui.py` z rozbaleného adresára.
- Zavrite okno s informáciami o nástroji.
- Hlavné okno nástroja má dve záložky (Protocol description, Settings). Nechajte aktívnu záložku *Protocol description*, v ktorej budeme popisovať protokol.

6.1.1 Implementácia Needham-Schroeder Public Key

V tomto príklade si vyskúšate implementáciu a verifikáciu *Needham-Schroeder Public Key*.

1. V úvode deklarujte globálne premenné. Medzi globálne premenné, tj. premenné viditeľné všetkým, v našom protokole patrí *public key*. V nástroji Scyther sú kľúče deklarované ako funkcie

```
1 const pk: Function;
2 secret sk: Function;
```

kde kľúčové slovo `const` znamená, že premenná bude automaticky pridaná k znalostiam útočníka. Kľúčové slovo `secret` naopak zaručí, že premenná nebude patriť medzi počiatočné útočníkové znalosti.

Aký je rozdiel medzi symetrickým a asymetrickým šifrovaním?

2. Verejný a súkromný kľúč sú navzájom inverzné. Túto skutočnosť zapíšte ako

```
3 inversekeys(pk,sk);
```

Ktoré kľúče budú použité pri šifrovaní správ medzi entitami `A` (*Alice*) a `B` (*Bob*)?

3. Definujte protokol pomocou kľúčového slova `protocol`

```
4 protocol NSPK(A,B,S) {
```

kde `A,B,S` sú identifikátory role. Zloženú zátvorku ešte neuzatvárajte, pretože budeme popisovať jeho správanie.

4. V rámci protokolu definujte rolu `A` (*Alice*) pomocou kľúčového slova `role` následovaného identifikátorom

```
5   role A {
```

5. Každá rola má svoje lokálne premenné. Tieto premenné určujú jej znalosti. V našom protokole má rola `A` dve premenné *nonce*

```
6       const Na: Nonce;
7       var Nb: Nonce;
```

kde do premennej `Nb` bude uložený *nonce* prijatý od role `B`.

Aký účel plní hodnota *nonce*? Je možné modelovať protokol bez tejto hodnoty?

6. Rola má svoje správanie. To zahrňuje posielanie a prijímanie správ. Zapisujeme len správy týkajúce sa konkrétnej roli. Zapísať musíme odoslanie aj prijatie. Obecný zápis má tvar

$$[\texttt{`read'}|\texttt{`send'}]\_\langle label\rangle\ \texttt{(}\langle od\rangle, \langle komu\rangle, \texttt{(}\langle term\rangle[, \langle term\rangle]\texttt{))};$$

kde $\langle label\rangle$ je pomenovanie správy v rámci celkovej komunikácie (odoslanie a prijatie konkrétnej správy musí mať rovnaké označenie).

Zapíšte správanie (správy) role `A`

```
8       send_1(A,S, (A,B));
9       read_2(S,A, {pk(B),B}sk(S));
10      send_3(A,B, {Na,A}pk(B));
11      read_6(B,A, {Na,Nb}pk(A));
12      send_7(A,B, {Nb}pk(B));
```

7. Definujte bezpečnostné požiadavky, ktoré má nástroj kontrolovať. V našom prípade sa jedná o tajnosť *secret* hodnoty *nonce*

```
13      claim_A1(A,Secret,Na);
14      claim_A2(A,Secret,Nb);
```

   V tomto momente máme definovanú rolu `A`, nezabudnite uzavrieť zátvorku

```
15    }
```

8. Analogicky na základe krokov 4. až 7. definujte rolu `B` (Bob)

```
16    role B {
17      const Nb: Nonce;
18      var Na: Nonce;
19
20      read_3(A,B, {Na,A}pk(B));
21      send_4(B,S, (B,A));
22      read_5(S,B, {pk(A),A}sk(S));
23      send_6(B,A, {Na,Nb}pk(A));
24      read_7(A,B, {Nb}pk(B));
25
26      claim_B1(B,Secret,Nb);
27      claim_B2(B,Secret,Na);
28    }
```

9. Definujte rolu servera `S`. Akú úlohu v tomto protokole plní server?

```
29    role S {
30      read_1(A,S, (A,B));
31      send_2(S,A, {pk(B),B}sk(S));
32      read_4(B,S, (B,A));
33      send_5(S,B, {pk(A),A}sk(S));
34    }
```

   V tomto okamihu máme definované správanie protokolu, nezabudnite uzavrieť zátvorku

```
35  }
```

10. Deklarujte účastníkov ako konštanty typu `Agent`

```
36  const Alice,Bob,Server,Compromised: Agent;
```

11. Na záver deklarujte nedôveryhodného účastníka (cieľ útočníka) a jeho prezradené informácie

```
37  untrusted Compromised;
38  const nc: Nonce;
39  compromised sk(Compromised);
40  compromised pk(Compromised);
41  compromised pk(Server);
```

12. Súbor uložte na disk (napr. *NSPK.spdl*). Spustite verifikáciu cez menu *Verify→Verify protocol* (`F1`). Nechajte si zobraziť útok.

6.1.2 Implementácia Needham-Schroeder Public Key - Lowe

V predchádzajúcom príklade ste implementovali a odsimulovali nezabezpečenú verziu *Needham-Schroeder PK* protokolu. Teraz si vyskúšate implementovať jeho modifikovanú verziu *Lowe* a porovnáte výsledok.V čom spočíva rozdiel medzi pôvodnou a modifikovanou verziou?

1. Načítajte uloženú verziu *NSPK.spdl* cez menu *File→Open*
2. Modifikovaná verzia pridáva do správy medzi *Bob* a *Alice* v 6. kroku identitu `B`. Modifikujte zdrojový súbor tak, aby odpovedal verzii *Lowe*

```
11      read_6(B,A, {Na,Nb,B}pk(A));

23      send_6(B,A, {Na,Nb,B}pk(A));
```

3. Spustite verifikáciu cez menu *Verify→Verify protocol* (`F1`). Porovnajte a diskutujte zmeny.

6.1.3 Implementácia Yahalom

V tomto príklade si vyskúšate implementáciu protokolu *Yahalom*, ktorý využíva symetrické šifrovanie.

1. Deklarujte symetrický kľúč ako funkciu. Tento kľúč bude tajný a nebude patriť medzi počiatočné útočníkové znalosti.

```
1 secret k: Function;
```

2. Deklarujte nový datový používateľský typ `SessionKey`. Tento typ bude určený pre *session* kľúč.

```
2 usertype SessionKey;
```

   Aký kľúč bude použitý pri šifrovaní správ medzi entitami `A` (*Alice*) a `B` (*Bob*)?

3. Definujte protokol *Yahalom* pomocou kľúčového slova `protocol`. Protokol bude obsahovať role *Alice*, *Bob*, *Server*

```
3 protocol Yahalom(A,B,S) {
```

4. V rámci protokolu definujte rolu `A` (*Alice*) kľúčovým slovom `role`

```
4   role A {
```

5. Deklarujte jej lokálne premenné: *nonce* (`Nonce`), dosaditeľný term (`ticket`) a *session* kľúč (`SessionKey`)

```
5       const Na: Nonce;
6       var Nb: Nonce;
7       var T: Ticket;
8       var Kab: SessionKey;
```

6. Popíšte správanie role `A` pomocou správ

```
9       send_1(A,B, A,Na);
10      read_3(S,A, {B,Kab,Na,Nb}k(A,S),T);
11      send_4(A,B, T,{Nb}Kab);
```

7. Deklarujte bezpečnostné požiadavky *claims* na tajnosť *session* kľúča

```
12      claim_A1(A,Secret,Kab);
```

   v tomto momente máme úplne definovanú rolu `A`. Uzavrite zloženú zátvorku

```
13    }
```

8. Opakujete kroky 4. až 7. a definujte rolu `B` (*Bob*)

```
14    role B {
15      const Nb: Nonce;
16      var Na: Nonce;
17      var T: Ticket;
18      var Kab: SessionKey;
19
20      read_1(A,B, A,Na);
21      send_2(B,S, B,{A,Na,Nb}k(A,S));
22      read_4(A,B, {A,Kab}k(A,S),{Nb}Kab);
23
24      claim_B1(B,Secret,Kab);
25    }
```

9. Podobne ako role `A` a `B` definujte poslednú rolu `S` (*Server*) Akú úlohu plní server v tomto protokole?

```
26    role S {
27      const Kab: SessionKey;
28      var Na,Nb: Nonce;
29
30      read_2(B,S, B,{A,Na,Nb}k(A,S));
31      send_3(S,A, {B,Kab,Na,Nb}k(A,S), {A,Kab}k(A,S));
32
33      claim_S1(S,Secret,Kab);
34    }
```

   v tomto momente máme definované správanie protokolu. Uzavrite zloženú zátvorku

```
35  }
```

10. Deklarujte účastníkov ako typ `Agent`. Deklarujte nedôveryhodného účastníka (útočníkov cieľ) a jeho uniknuté informácie

```
36  const Alice,Bob,Server,Compromised: Agent;
37  untrusted Compromised;
38  compromised k(Compromised,Server);
```

11. Spustite verifikáciu cez menu *Verify→Verify protocol* (`F1`) a overte, či existuje dostupný útok.

6.1.4 Implementácia Otway Rees

V tomto príklade si vyskúšate implementáciu protokolu *Otway Rees*, ktorý využíva symetrické šifrovanie.

1. Deklarujte symetrický kľúč ako funkciu. Tento kľúč nebude patriť medzi počiatočné útočníkové znalosti.

```
1 secret k: Function;
```

2. Deklarujte nový datový používateľský typ `SessionKey`. Tento typ bude určený pre *session* kľúč. Zároveň definujte typ reťazec `String`

```
2 usertype String,SessionKey;
```

3. Definujte protokol *Otway Rees* pomocou kľúčového slova `protocol`. Protokol bude obsahovať role *Alice*, *Bob*, *Server*

```
3 protocol OtwayRees(A,B,S) {
```

4. V rámci protokolu definujte rolu `A` (*Alice*) kľúčovým slovom `role`

```
4   role A {
```

5. Deklarujte jej lokálne premenné *nonce*, reťazec `String` a *session* kľúč `SessionKey`

```
5       const Na: Nonce;
6       const M: String;
7       var Kab: SessionKey;
```

6. Popíšte správanie role `A` pomocou správ

```
8       send_1(A,B, M,A,B,{Na,M,A,B}k(A,S));
9       read_4(B,A, M,{Na,Kab}k(A,S));
```

7. Deklarujte bezpečnostné požiadavky `claim` na tajnosť *session* kľúča a synchronizáciu *nonce*

```
10      claim_A1(A,Secret,Kab);
11      claim_A2(A,Nisynch);
```

v tomto momente máme úplne definovanú rolu `A`. Uzavrite zloženú zátvorku

```
12  }
```

8. Opakujete kroky 4. až 7. a definujte rolu `B` (*Bob*)

```
13    role B {
14      var M: String;
15      const Nb: Nonce;
16      var Kab: SessionKey;
17      var T1,T2: Ticket;
18
19      read_1(A,B, M,A,B,T1);
20      send_2(B,S, M,A,B,T1,{Nb,M,A,B}k(B,S));
21      read_3(S,B, M,T2,{Nb,Kab}k(B,S));
22      send_4(B,A, M,T2);
23
24      claim_R1(B,Secret,Kab);
25      claim_R2(B,Nisynch);
26    }
```

9. Podobne ako role `A` a `B` definujte poslednú rolu `S` (*Server*). Akú úlohu plní v tomto protokole server?

```
27    role S {
28      var Na,Nb: Nonce;
29      var M: String;
30      const Kab: SessionKey;
31
32      read_2(B,S, M,A,B,{Na,M,A,B}k(A,S),{Nb,M,A,B}k(B,S));
33      send_3(S,B, M,{Na,Kab}k(A,S),{Nb,Kab}k(B,S));
34    }
```

   v tomto momente máme definované správanie protokolu. Uzavrite zloženú zátvorku

```
35  }
```

10. Deklarujte účastníkov ako typ `Agent`. Deklarujte nedôveryhodného účastníka (útočníkov cieľ) a jeho uniknuté informácie

```
36  const Alice,Bob,Server,Compromised: Agent;
37  untrusted Compromised;
38  compromised k(Compromised,Server);
```

11. Spustite verifikáciu cez menu *Verify→Verify protocol* (`F1`) a overte, či existuje dostupný útok.

6.1.5 Implementácia Denning-Sacco Shared Key

V tomto príklade si vyskúšate implementáciu protokolu *Denning-Sacco Shared Key*, ktorý využíva symetrické šifrovanie.

1. Deklarujte symetrický kľúč ako funkciu. Tento kľúč nebude patriť medzi počiatočné útočníkové znalosti

```
1  secret k: Function;
```

2. Deklarujte nový datový typ `SessionKey`. Tento typ bude určený pre *session* kľúč. Zároveň deklarujte typ časové razítko `TimeStamp`, ktorý tento protokol používa

```
usertype SessionKey;
usertype TimeStamp;
```

   Aký kľúč bude použitý pri šifrovaní správ medzi entitami `A` (*Alice*) a `B` (*Bob*)?

3. Definujte protokol *DenningSacco* pomocou kľúčového slova `protocol`. Protokol bude obsahovať role *Alice*, *Bob*, *Server*

```
protocol DenningSacco(A,B,S) {
```

4. V rámci protokolu definujte rolu `A` (*Alice*) kľúčovým slovom `role`

```
  role A {
```

5. Deklarujte jej lokálne premenné: časové razítko (`TimeStamp`), dosaditeľný term (`Ticket`) a *session* kľúč (`SessionKey`)

```
    var W: Ticket;
    var Kab: SessionKey;
    var T: TimeStamp;
```

   Aký význam má použitie časového razítka?

6. Popíšte správanie role `A` pomocou správ

```
    send_1(A,S, A,B);
    read_2(S,A, {B,Kab,T,W}k(A,S));
    send_3(A,B, W);
```

7. Deklarujte bezpečnostné požiadavky `claim` na tajnosť *session* kľúča a synchronizáciu *nonce*. Sledovanie synchronizácie umožní odhaliť *replay* útok

```
    claim_A1(A,Nisynch);
    claim_A2(A,Secret,Kab);
```

   v tomto momente máme úplne definovanú rolu `A`. Uzavrite zloženú zátvorku

```
  }
```

8. Podľa krokov 4. až 7. definujte rolu `B` (*Bob*)

```
  role B {
    var Kab: SessionKey;
    var T: TimeStamp;

    read_3(A,B, {Kab,A,T}k(B,S));

    claim_B1(B,Nisynch);
    claim_B2(B,Secret,Kab);
  }
```

9. Podobne ako role `A` a `B` definujte poslednú rolu `S` (*Server*). Akú úlohu plní server v tomto protokole?

```
24    role S {
25      var W: Ticket;
26      const Kab: SessionKey;
27      const T: TimeStamp;
28
29      read_1(A,S, A,B);
30      send_2(S,A, {B,Kab,T,{Kab,A,T}k(B,S)}k(A,S));
31    }
```

   v tomto momente máme definované správanie protokolu. Uzavrite zloženú zátvorku

```
32  }
```

10. Deklarujte účastníkov ako typ `Agent`. Deklarujte nedôveryhodného účastníka (útočníkov cieľ) a uniknuté informácie

```
33  const Alice,Bob,Server,Compromised: Agent;
34  compromised k(Compromissed,Server);
```

11. Súbor uložte na disk (napr. *DSSK.spdl*). Spustite verifikáciu cez menu *Verify→Verify protocol* (`F1`) a overte, či existuje dostupný útok.

6.1.6 Implementácia Denning-Sacco Shared Key - Lowe

V predchádzajúcom príklade ste implementovali a odsimulovali nezabezpečenú verziu *Denning-Sacco SK* protokolu. Teraz si vyskúšate implementovať jeho modifikovanú verziu *Lowe* a porovnáte výsledok. V čom spočíva rozdiel oproti pôvodnej verzií?

1. Načítajte uloženú verziu *DSSK.spdl* cez menu *File→Open*

2. Modifikovaná verzia upravuje protokoly tak, aby využívali *nonce*. Zároveň pridáva funkciu dekrementácie. Modifikujte zdrojový súbor tak, aby odpovedal verzii *Lowe*

   Modifikácia vyžaduje deklaráciu nového datového typu `PseudoFunction` na globálnej úrovni. Jej správanie nieje nutné explicitne vyjadriť

```
4  usertype PseudoFunction;
5  const dec: PseudoFunction;
```

   Ďalej upravte správanie rolí tak, aby odpovedalo modifikácií *Lowe*. Nezabudnite deklarovať lokálnu premennú *nonce*

```
11      var Nb: Nonce;

23      const Nb: Nonce;

15      read_4(B,A, {Nb}Kab);
16      send_5(A,B, {{Nb}dec}Kab);
```

```
26      send_4(B,A, {Nb}Kab);
27      read_5(A,B, {{Nb}dec}Kab);
```

3. Spustite verifikáciu cez menu *Verify→Verify protocol* (`F1`). Porovnajte a diskutujte zmeny.

6.2 Úloha č.2

Cieľ úlohy

- Zoznámiť sa s nástrojom SPAN.
- Vyskúšať si implementáciu symetrických a asymetrických bezpečnostných protokolov.
- Vyskúšať si verifikáciu bezpečnostných protokolov v nástroji SPAN.

Študijné materiály

- V tejto úlohe bude cieľom implementovať protokoly *Needham-Schroeder Public Key*, *Yahalom* a protokol *Denning-Sacco Shared Key*.
- Študijné materiály
  - Needham-Schroeder Public Key
    `http://www.lsv.ens-cachan.fr/Software/spore/nspk.html`
    Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995.
  - Yahalom
    `http://www.lsv.ens-cachan.fr/Software/spore/yahalom.html`
    Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, february 1989.
  - Denning-Sacco Shared Key
    `http://www.lsv.ens-cachan.fr/Software/spore/denningSacco.html`
    D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.

Príprava prostredia

- Stiahnite a nainštalujte knižnicu Tcl/Tk. Aktuálna verzia je dostupná na adrese `http://sourceforge.net/projects/tcl/files/Tcl/8.3.2/tcl832.exe/download`.
- Stiahnite inštalačný súbor z domovskej stránky projektu SPAN a spustite inštaláciu: `http://www.irisa.fr/celtique/genet/span/`
- Spustite `span.exe` z adresára aplikácie.

6.2.1 Implementácia Needham-Schroeder Public Key

V tomto príklade si vyskúšate implementáciu a verifikáciu *Needham-Schroeder Public Key*.

1. Po spustení nástroja zadávajte kód protokolu v hlavnej editovacej časti okna.
2. Na úvod deklarujte protokol kľúčovým slovom `protocol` nasledovaným jeho menom

```
protocol NeedhamSchroederPublicKey;
```

3. Nasleduje deklarácia identifikátorov (premenných). Obecný zápis premennej má tvar

$$nazov : typ;$$

Deklarujte účastníkov (`user`), *nonce* (`number`) a verejné kľúče (`public_key`) ako premenné

```
identifiers
 A,B,S       : user;
 Na,Nb       : number;
 KPa,KPb,KPs : public_key;
```

4. V ďalšej časti definujte správanie protokolu pomocou správ. Správy majú obecný tvar

$$N.[od] \rightarrow [komu] : obsah, [\{sifrovany\_obsah\}kluc]$$

kde $N = 1..k$.

```
messages
 1. A -> S : A,B
 2. S -> A : {KPb, B}KPs'
 3. A -> B : {Na, A}KPb
 4. B -> S : B,A
 5. S -> B : {KPa, A}KPs'
 6. B -> A : {Na, Nb}KPa
 7. A -> B : {Nb}KPb
```

kde ' značí súkromný kľúč.

5. Po dokončení popisu správ správania deklarujte znalosti jednotlivých účastníkov

```
knowledge
 A : A,B,S,KPa,KPb,KPs;
 B : A,B,S,KPa,KPb,KPs;
 S : A,B,S,KPa,KPb,KPs;
```

6. Kedže nástroj umožňuje pracovať s viacerými výskytmi protokolu súčasne, je potrebné priradiť k jednotlivým premenným konkrétne hodnoty. V časti `session_instances` priraďte identifikátorom z kroku 3 konkrétne hodnoty

```
session_instances
 [A:alice,B:bob,S:server,KPa:pk1,KPb:pk2,KPs:pk3];
```

7. V časti `intruder_knowledge` definujte počiatočné znalosti útočníka

```
intruder_knowledge
 alice,bob,pk1,pk3;
```

8. Na záver v časti `goal` definujte bezpečnostné požiadavky protokolu. V našom prípade sa jedná o autentifikáciu účastníkov pomocou *nonce*

```
goal
 A authenticates B on Na;
 B authenticates A on Nb;
```

9. Uložte protokol v jazyku *CAS+*. Kedže nástroj pracuje s jazykom *HLPSL*, automaticky pri uložení (načítaní) ponúkne konverziu do tohto formátu. Potvrďte voľbu `'Yes'`.

10. V dolnej časti okna vyberte najprv nástroj *OFMC* potom *ATSE*, spustite verifikáciu a zistite, či existuje k danému protokolu útok. Prípadný útok si nechajte vizualizovať.

6.2.2 Implementácia Yahalom

V tomto príklade si vyskúšate implementáciu protokolu *Yahalom*, ktorý využíva symetrické šifrovanie.

1. Po spustení nástroja zadávajte kód protokolu v hlavnej editovacej časti okna.

2. Na úvod deklarujte protokol kľúčovým slovom `protocol` nasledovaným jeho menom

```
protocol Yahalom;
```

3. Nasleduje deklarácia identifikátorov (premenných). Pripomeňme, že obecný zápis premennej má tvar

$$nazov : typ;$$

Deklarujte účastníkov (`user`), *nonce* (`number`) a verejné kľúče (`public_key`) ako premenné

```
identifiers
 A,B,S       : user;
 Na,Nb       : number;
 KPa,KPb,KPs : public_key;
```

4. V ďalšej časti definujte správanie protokolu pomocou správ

```
messages
 1. A -> B : A,Na
 2. B -> S : B,{A,Na,Nb}Kbs
 3. S -> A : {B,Kab,Na,Nb}Kas,{A,Kab}Kbs
 4. A -> B : {A,Kab}Kbs,{Nb}Kab
```

5. Po dokončení popisu správ správania deklarujte počiatočné znalosti jednotlivých účastníkov

```
knowledge
 A : A,B,S,Kas;
 B : B,S,Kbs;
 S : S,A,B,Kas,Kbs;
```

6. Nástroj umožňuje pracovať s viacerými výskytmi protokolu súčasne, je potrebné priradiť k jednotlivým premenným konkrétne hodnoty. V časti `session_instances` preto priraďte identifikátorom z kroku 3 konkrétne hodnoty

```
session_instances
 [A:alice,B:bob,S:server,Kas:key1,Kbs:key2];
```

7. Na záver v časti `goal` definujte bezpečnostné požiadavky protokolu. V našom prípade sa jedná o tajnosť *session* kľúča

```
goal
 secrecy_of Kab [];
```

8. Uložte protokol v jazyku *CAS+*. Kedže nástroj pracuje s jazykom *HLPSL*, automaticky pri uložení (načítaní) ponúkne konverziu do tohto formátu. Potvrďte voľbu `'Yes'`.

9. V dolnej časti okna vyberte najprv nástroj *OFMC* potom *ATSE*, spustite verifikáciu a zistite, či existuje k danému protokolu útok.

6.2.3 Implementácia Denning-Sacco Shared Key

V tomto príklade si vyskúšate implementáciu a verifikáciu *Denning-Sacco Shared Key*. Na akom šifrovaní je založený protokol?

1. Na úvod deklarujte protokol kľúčovým slovom `protocol` nasledovaným jeho menom.

```
protocol DenningSaccoSharedKey;
```

2. Nasleduje deklarácia identifikátorov (premenných). Ako bolo spomenuté, obecný zápis premennej má tvar

$$nazov : typ;$$

Deklarujte účastníkov (`user`), *nonce* (`number`) a symetrické kľúče (`symmetric_key`) ako premenné

```
identifiers
 A,B,S       : user;
 T           : number;
 Kas,Kbs,Kab : symmetric_key;
```

3. V ďalšej časti definujte správanie protokolu pomocou správ. Aké kľúče budú použité pre šifrovanie komunikácie?

```
messages
 1. A -> S : A,B
 2. S -> A : {B,Kab,T,{Kab,A,T}Kbs}Kas
 3. A -> B : {Kab,A,T}Kbs
```

4. Po dokončení popisu správania deklarujte počiatočné znalosti jednotlivých účastníkov.

```
10 knowledge
11  A : A,B,S;
12  B : A,B,S;
13  S : A,B,S;
```

5. Kedže nástroj umožňuje pracovať s viacerými výskytmi protokolu súčasne, je potrebné priradiť k jednotlivým premenným konkrétne hodnoty. Priraďte preto v časti `session_instances` identifikátorom z kroku 3 konkrétne hodnoty

```
14 knowledge
15 session_instances
16  [A:alice,B:bob,S:server,Kas:key1,Kbs:key2];
```

6. Na záver v časti `goal` definujte bezpečnostné požiadavky protokolu. V našom prípade sa jedná o tajnosť *session* kľúča

```
17 goal
18  secrecy_of Kab [];
```

7. Uložte protokol v jazyku *CAS+*. Kedže nástroj pracuje s jazykom *HLPSL*, automaticky pri uložení (načítaní) ponúkne konverziu do tohto formátu. Potvrďte voľbu `'Yes'`.

8. V dolnej časti okna vyberte najprv nástroj *OFMC* potom *ATSE*, spustite verifikáciu a zistite, či existuje k danému protokolu útok. Prípadný útok si nechajte vizualizovať.

6.3 Úloha č.3

Cieľ úlohy

- Zoznámiť sa s nástrojom SPEAR 2.
- Vyskúšať si implementáciu symetrických a asymetrických bezpečnostných protokolov.
- Vyskúšať si verifikáciu bezpečnostných protokolov v nástroji SPEAR 2.

Študijné materiály

- V tejto úlohe bude cieľom implementovať protokoly *Needham-Schroeder Public Key* a *Yahalom*.
- Študijné materiály
  - Needham-Schroeder Public Key
    `http://www.lsv.ens-cachan.fr/Software/spore/nspk.html`
    Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995.
  - Yahalom
    `http://www.lsv.ens-cachan.fr/Software/spore/yahalom.html`
    Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, february 1989.

Príprava prostredia

- Stiahnite a nainštalujte SWI-Prolog z adresy
  `http://www.swi-prolog.org/download/stable/bin/w32pl583.exe`
- Stiahnite inštalačný súbor z domovskej stránky projektu a spustite inštaláciu
  `http://www.cs.uct.ac.za/Research/DNA/SPEAR2/index.html`
- Spustite `Spear2.exe` z adresára aplikácie.

6.3.1 Implementácia Needham-Schroeder Public Key

V tomto príklade si vyskúšate implementáciu a verifikáciu *Needham-Schroeder Public Key*.

1. Po spustení aplikácie si môžte všimnúť, že okno je rozdelené na dve časti. V ľavej časti je stromový prehľad objektov. V pravej časti (*canvas*) sú tieto objekty usporiadane do časového diagramu. V hlavnom menu aplikácie sa nachádzajú všetky potrebné nástroje.

2. Na úvod pridajte účastníkov `A` (*Alice*), `B` (*Bob*) a `S` (*Server*) cez menu

   $$Design \rightarrow AddCanvasObject \rightarrow Principal \qquad (Ctrl + P)$$

3. V ďaľšom kroku pridajte prvú správu cez menu

   $$Design \rightarrow AddCanvasObject \rightarrow Message \qquad (Ctrl + M)$$

4. V dialógovom okne pre novú správu nastavte príjemcu a odosielateľa. Dialógové okno nezatvárajte, budeme pridávať objekty do správy.

5. Prvá správa *Needham-Schroeder* protokolu obsahuje iba dve entity `A` a `B`. Pravým tlačítkom nad panelom *Message Components* pridajte existujúcich účastníkov

   $$Add \rightarrow Existing \rightarrow Component \rightarrow A$$
   $$Add \rightarrow Existing \rightarrow Component \rightarrow B$$

6. Podľa krokov 2. až 5. pridajte ďaľšie správy podľa špecifikácie protokolu. Objekty správy, ktoré ešte neboli vytvorené, vytvorte priamo v dialógovom okne pre novú správu

   $$Add \rightarrow New \rightarrow [typ\_objektu]$$

   Obdobným spôsobom funguje pridanie šifrovaného obsahu, kedy najprv vytvoríme objekt *Public/Private Encryption*

   $$Add \rightarrow New \rightarrow Public/Private\ Encryption$$

   a následne pridáme v rámci tohto objektu obsah (podobne ako pri správe). Na záver upravíme vlastnosti objektu *Encryption* (konkrétne *Edit Key*), kde vpíšeme identifikátor objektu (napr. `a`) komu kľúč patrí.

   Poradie správ možno dodatočne meniť vo vlastnostiach správy, prípadne interaktívne myšou v hlavnom okne v časovom diagrame.

7. Po dokončení skontrolujte v časovom diagrame či modelovaný protokol odpovedá špecifikácií.

8. Teraz potrebujete definovať bezpečnostné predpoklady (*beliefs*, *proofs*) *GNY logiky* cez menu

   $$GNY \rightarrow Initial\ Assumptions \qquad (Ctrl + Alt + A)$$

   Predpoklady existujú zvlášť pre každého účastníka. Podobne ako pri vytváraní správy, pravým tlačítkom myši v záložke *Assumptions* pridajte postupne objekty

   - *Fresh Components* – Aktuálnosť objektu
   - *Recognizable Components* – Znalosti objektu
   - *Suitable Public Keys* – Znalosť kľúčov
   - *Trustworthy Principals* – Dôveryhodný účastníci
   - *Jurisdiction* – Dôvera v objekt, ktorý je dôveryhodný pre iného účastníka
   - *Suitable Secret* – Tajomstvo medzi objektami

9. Na záver rovnako ako v kroku 7, definujte bezpečnostné ciele protokolu

   $$GNY \rightarrow Intended\ Goals \qquad (Ctrl + Alt + G)$$

   Predpoklady sa rovnako pridávajú zvlášť pre každého účastníka. Pridajte postupne objekty.

10. Spustite analýzu protokolu cez menu

    $$GNY \rightarrow Analyze\ Protocol \mid Analyze \qquad (Ctrl + Alt + Z)$$

    Po dokončení analýzy sa prepnite do záložky *Result*. Diskutujte zistené poznatky.

6.3.2 Implementácia Yahalom

V tomto príklade si vyskúšate implementáciu a verifikáciu *Yahalom*.

1. Po spustení aplikácie si môžte všimnúť, že okno je rozdelené na dve časti. V ľavej časti je stromový prehľad objektov. V pravej časti (*canvas*) sú tieto objekty usporiadane do časového diagramu. V hlavnom menu aplikácie sa nachádzajú všetky potrebné nástroje.

2. Na úvod pridajte účastníkov `A` (*Alice*), `B` (*Bob*) a `S` (*Server*) cez menu

   $$Design \rightarrow AddCanvasObject \rightarrow Principal \qquad (Ctrl + P)$$

3. V ďaľšom kroku pridajte prvú správu cez menu

   $$Design \rightarrow AddCanvasObject \rightarrow Message \qquad (Ctrl + M)$$

4. V dialógovom okne pre novú správu nastavte príjemcu a odosielateľa. Dialógové okno nezatvárajte, budeme pridávať objekty do správy.

5. Prvá správa *Yahalom* protokolu obsahuje entitu `A` a objekt `nonce`. Pravým tlačítkom nad panelom *Message Components* pridajte existujúcich účastníkov

$$Add \rightarrow Existing \rightarrow Component \rightarrow A$$
$$Add \rightarrow New \rightarrow Nonce$$

6. Podľa krokov 2. až 5. pridajte ďalšie správy podľa špecifikácie protokolu. Objekty správy, ktoré ešte neboli vytvorené, vytvorte priamo v dialógovom okne pre novú správu

$$Add \rightarrow New \rightarrow [typ\_objektu]$$

   Obdobným spôsobom funguje pridanie šifrovaného obsahu, kedy najprv vytvoríme objekt *Public/Private Encryption*

$$Add \rightarrow New \rightarrow Public/Private\ Encryption$$

   a následne pridáme v rámci tohto objektu obsah (podobne ako pri správe). Na záver upravíme vlastnosti objektu *Encryption* (konkrétne *Edit Key*), kde vpíšeme identifikátor objektu (napr. `a`) komu kľúč patrí.

   Poradie správ možno dodatočne meniť vo vlastnostiach správy, prípadne interaktívne myšou v hlavnom okne v časovom diagrame.

7. Po dokončení skontrolujte v časovom diagrame či modelovaný protokol odpovedá špecifikácií.

8. Teraz potrebujete definovať bezpečnostné predpoklady (*beliefs*, *proofs*) *GNY logiky* cez menu

$$GNY \rightarrow Initial\ Assumptions \qquad (Ctrl + Alt + A)$$

   Predpoklady existujú zvlášť pre každého účastníka. Podobne ako pri vytváraní správy, pravým tlačítkom myši v záložke *Assumptions* pridajte postupne objekty

   - *Fresh Components* – Aktuálnosť objektu
   - *Recognizable Components* – Znalosti objektu
   - *Suitable Public Keys* – Znalosť kľúčov
   - *Trustworthy Principals* – Dôveryhodný účastníci
   - *Jurisdiction* – Dôvera v objekt, ktorý je dôveryhodný pre iného účastníka
   - *Suitable Secret* – Tajomstvo medzi objektami

9. Na záver rovnako ako v kroku 7, definujte bezpečnostné ciele protokolu

$$GNY \rightarrow Intended\ Goals \qquad (Ctrl + Alt + G)$$

   Predpoklady sa rovnako pridávajú zvlášť pre každého účastníka. Pridajte postupne objekty.

10. Spustite analýzu protokolu cez menu

$$GNY \rightarrow Analyze\ Protocol \mid Analyze \qquad (Ctrl + Alt + Z)$$

    Po dokončení analýzy sa prepnite do záložky *Result*. Diskutujte zistené poznatky.

6.4 Zhrnutie

Demonstračné úlohy pozostávajú z implementácie protokolov *Needham-Schroeder Public Key*, *Needham-Schroeder Public Key - Lowe*, *Yahalom*, *Denning-Sacco Shared Key*, *Denning-Sacco Shared Key - Lowe* a *Otway Rees*. Na nich si možno vyskúšať modelovanie symetrického aj asymetrického šifrovania. Zároveň niektoré protokoly obsahujú zraniteľnosti. Medzi úlohy som sa rozhodol okrem nástroja Scyther zaradiť aj ďalšie. Konkrétne sa jedná o nástroje SPAN a SPEAR 2.

Kapitola 7

Záver

Cieľom tejto práce bolo zoznámiť sa s princípmi a použitím komunikačných protokolov, konkrétne bezpečnostných protokolov. Zároveň sa zoznámiť s dostupnými nástrojmi a prostrediami pre interaktívnu simuláciu komunikácie, tieto nástroje vzájomne porovnať a napísať ich vlastnosti.

V druhej kapitole som sa venoval teoretickému úvodu do problematiky bezpečnostných protokolov, zameral sa na ich vlastnosti, funkcie a možné typy útokov. Rovnako som sa venoval výrazovým prostriedkom pre ich popis. V tretej kapitole som predstavil vybrané nástroje. Hlavným kritériom pre výber bola schopnosť interaktívne simulovať bezpečnostné protokoly. Pri každom nástroji som popísal jeho vlastnosti a parametre a snažil sa zhrnúť jeho výhody resp. nevýhody do niekoľkých bodov. Následne som jednotlivé nástroje porovnal z pohľadu implementácie a napísal používateľskú príručku. Na záver som navrhol sadu demonstračných úloh a príkladov.

Každý z nástrojov uvedený v tejto práci s výnimkou *NS2* je schopný simulovať a verifikovať bezpečnostné protokoly. Okrem GRASP sú ostatné nástroje voľne dostupné. Vyznačujú sa vlastnými jazykmi pre popis protokolov a prostredím pre simuláciu. Špecifický je aj prístup k modelovaniu protokolov. Nástroje AVISPA a SPAN zakladajú na modelovaní stavového automatu, Scyther vyžaduje definíciu rolí a SPEAR2 popisuje objekty a využíva GNY logiku pre počiatočné a cieľové požiadavky. V jednotlivých nástrojoch som implementoval zvolené bezpečnostné protokoly a navrhol sadu demonstračných úloh. Tieto úlohy si kladú za cieľ zoznámiť čitateľa s problematikou modelovania a verifikácie bezpečnostných protokolov. Zároveň demonštrujú možnosti a špecifické spôsoby implementácie. Výsledky práce ako aj vytvorené úlohy, nájdu uplatnenie v sieťovo orientovaných predmetoch na FIT VUT. Za vhodný nástroj pre tento účel považujem Scyther, ktorý sa vyznačuje jednoduchosťou modelovania a zároveň spĺňa všetky požiadavky na simuláciu a verifikáciu bezpečnostných protokolov.

Ďalšie pokračovanie práce by spočívalo v aktualizovaní a doplnení zoznamu nástrojov. Rovnako je možné rozšíriť demonstračné úlohy o ďalšie protokoly a úlohy.

Literatura

[1] team AVISPA. HLPSL Tutorial – A Beginner's Guide to Modelling and Analysing Internet Security Protocols. 2006.

[2] Boichut, Y.; Genet, T.; Glouche, Y.; aj.. Using Animation to Improve Formal Specifications of Security Protocols. In *Joint conference SAR-SSI*, 2007.

[3] Bouroulet, R.; Klaudel, H.; Pelz, E.. A Semantics of Security Protocol Language (SPL) using a Class of Composable High-Level Petri Nets. *Application of Concurrency to System Design, International Conference on*, ročník 0, 2004: str. 99.

[4] Clarke, E. M.; Jha, S.; Marrero, W.. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, ročník 9, č. 4, 2000: s. 443–487, ISSN 1049-331X.

[5] Cremers, C.. Scyther 1.0 – User Manual. Květen 2007.

[6] Cremers, C.. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, Lecture Notes in Computer Science, Springer, 2008, s. 414–418.

[7] Glouche, Y.; Genet, T.; Heen, O.; aj.. A Security Protocol Animator Tool for AVISPA. In *ARTIST-2 workshop on security of embedded systems, Pisa (Italy)*, Květen 2006.

[8] Hanáček, P.. Bezpečnostní funkce v počítačových sítích. *Zpravodaj ÚVT MU*, ročník 10, č. 2, 1999: s. 5–9, [Online; cit. 2009-11-30].
URL `http://www.ics.muni.cz/zpravodaj/articles/171.html`

[9] Hanáček, P.; Staudek, J.. *Bezpečnost informačních systémů :metodická příručka zabezpečování produktů a systémů budovaných na bázi informačních technologií.* Úřad pro státní informační systém, Praha, 2000, ISBN 80-85867-35-4, 127 s.

[10] Kunderová, L.. Bezpečnost komunikačních procesů. [Online; cit. 2009-11-30].
URL `https://akela.mendelu.cz/~lidak/bis/11prot.htm`

[11] Lowe, G.. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, ročník 56, 1995: s. 131–133.

[12] Monahan, B.. Introducing ASPECT — a tool for checking protocol security. Září 2002.

[13] Needham, R.; Schroeder, M.. Using Encryption for Authentification in Large Networks of Computers. *CACM*, ročník 21, č. 12, Prosinec 1978.

[14] Oppliger, R.; Hauser, R.; Basin, D.. SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle. *Computer Communications*, ročník 29, č. 12, 2006: s. 2238 – 2246, ISSN 0140-3664.

[15] Očenášek, P.. *Verifikace bezpečnostních protokolů*. Diplomová práce, FIT VUT v Brně, Květen 2003.

[16] Očenášek, P.. IBS: Bezpečnosť a počítačové sítě. FIT VUT v Brně, Únor 2008, [cit. 2009-11-30].

[17] Paulson, L. C.. Relations Between Secrets: Two Formal Analyses of the Yahalom Protocol. *J. Computer Security*, 2001.

[18] Ptáček, M.. *Specifikační jazyky a nástroje pro analýzu a verifikaci bezpečnostních protokolů*. Diplomová práce, FIT VUT v Brně, Květen 2007.

[19] Pužmanová, R.. *Moderní komunikační sítě od A do Z / 2. aktualiz. vyd.* Computer Press, Brno, 2006, ISBN 80-251-1278-0, 430 s.

[20] Risso, F.; Baldi, M.. NetPDL: An extensible XML-based language for packet header description. *Computer Networks*, ročník 50, č. 5, 2006: s. 688–706, ISSN 1389-1286.

[21] Saillard, R.; Genet, T.. CAS+. Září 2007.

[22] Saul, E.; Hutchison, A.. SPEAR II - The Security Protocol Engineering and Analysis Resource. Září 1999.

[23] Schweitzer, D.; Baird, L.; Collins, M.; aj.. GRASP: A Visualization Tool for Teaching Security Protocols. In *10th Colloquium for Information Systems Security Education University of Maryland*, Červen 2006.

[24] Viganò, L.. Automated Security Protocol Analysis With the AVISPA Tool. *Electronic Notes in Theoretical Computer Science (Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS XXI)*, ročník 155, 2006: s. 61–86.

Dodatek A

Obsah CD

- \doc
  - **\thesis** - Text diplomovej práce vo formáte PDF
  - **\practice** - Demonstračné úlohy vo formáte PDF
- \src
  - **\text** - Zdrojový text diplomovej práce vo formáte LaTeX
  - **\practice** - Zdrojový text demonstračných úloh vo formáte LaTeX a Microsoft word 2007+
  - **\protocols** - Zdrojové súbory vybraných bezpečnostných protokolov
- \tools
  - **\scyther** - Aktuálna verzia nástroja Scyther vrátane podporných knižníc