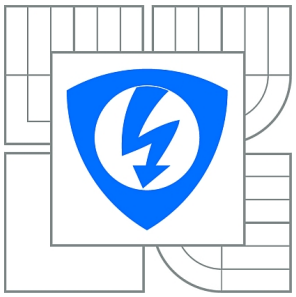
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

FORMÁLNÍ ANALÝZA KRYPTOGRAFICKÝCH PROTOKOLŮ

FORMAL ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

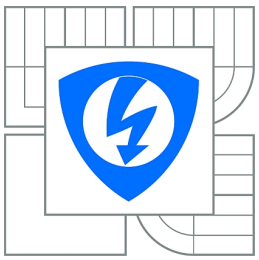
AUTOR PRÁCE — Bc. PETER PETROVSKÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. VLASTIMIL ČLUPEK

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Peter Petrovský	***ID:***	136574
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Formální analýza kryptografických protokolů

POKYNY PRO VYPRACOVÁNÍ:

Popište metody používané k ohodnocení formální bezpečnosti u kryptografických protokolů. Zaměřte se na princip činnosti těchto metod a na jejich matematický základ. Proveďte analýzu existujících nástrojů využívaných k automatickému a poloautomatickému vyhodnocení bezpečnosti kryptografických protokolů. Popište způsob práce s těmito nástroji a porovnejte jejich možnosti. Srovnejte efektivnost těchto nástrojů s ručními technikami sloužícími k ohodnocení formální bezpečnosti kryptografických protokolů. Pomocí vybraných testovacích nástrojů proveďte formální analýzu bezpečnosti zvolených kryptografických protokolů.

DOPORUČENÁ LITERATURA:

[1] PATEL, Reema, et al. Comparative analysis of formal model checking tools for security protocol verification. In: Recent Trends in Network Security and Applications. Springer Berlin Heidelberg, 2010. p. 152-163.
[2] MEADOWS, Catherine. Open issues in formal methods for cryptographic protocol analysis. In: DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings. IEEE, 2000. p. 237-250.
[3] MENEZES, Alfred J.; VAN OORSCHOT, Paul C.; VANSTONE, Scott A. Handbook of applied cryptography. CRC press, 2010.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: Ing. Vlastimil Člupek
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

ABSTRAKT

Táto diplomová práca sa zaoberá kryptografiou. Popisuje sa jej základné rozdelenie a problémy teórie čísel, ktoré musí riešiť. Taktiež sa zaoberá metódami, ktoré sa používajú k ohodnoteniu formálnej bezpečnosti kryptografických protokolov z matematického hľadiska. Analyzujú sa nástroje využívané k automatickému a poloautomatickému vyhodnoteniu bezpečnosti kryptografických protokolov. Popisuje sa spôsob práce s týmito nástrojmi a nakoniec sa otestuje bezpečnosť protokolov Kerberos, EKE a protokolov jednosmernej autentizácie využívajúcich symetrickú kryptografiu, funkciu HMAC a hashovaciu funkciu postupne v nástrojoch AVISPA, ProVerif a Scyther. Na záver je porovnanie výsledkov.

KĽÚČOVÉ SLOVÁ

Kyptografia, Dolev-Yao, BAN logika, AVISPA, ProVerif, Scyther, Kerberos, EKE, jednosmerná autentizácia, analýza

ABSTRACT

This diploma thesis deals with cryptography. It describes its basic allocation and problems of number theory that needs to be addressed. It also deals with methods used to review the formal security of cryptographic protocols from a mathematical point of view. It analyse the tools used to automatic and semi-automatic evaluation of the safety of cryptographic protocols. It describes the process of working with these tools and finally test the security of protocols Kerberos, EKE and Unilateral authentication protocols using symmetric cryptography, HMAC function and hash function. These tests are in tools AVISPA, ProVerif and Scyther. At the end is comparison of results.

KEYWORDS

Cryptography, Dolev-Yao, BAN logic, AVISPA, ProVerif, Scyther, Kerberos, EKE, Unilateral authentication, analysis

PETROVSKÝ, Peter *Formální analýza kryptografických protokolů*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 78 s. Vedúci práce bol Ing. Vlastimil Člupek

PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Formální analýza kryptografických protokolů“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúceho autorského zákona č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád by som poďakoval vedúcemu semestrálnej práce pánovi Ing. Vlastimilovi Člupkovi za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Kryptografia sa po stáročia vyvíjala k väčšej zložitosti zároveň s ľudskou civilizáciou a mnohokrát ovplyvnila beh dejín. V dnešnej dobe sa kladú na bezpečnosť čoraz väčšie nároky. Súvisí to s rozvojom technológií a s čoraz sofistikovanejšími útokmi. Častokrát sa nachádzajú rôzne chyby už pri vývoji protokolov, ktoré môžu dizajnéri prehliadnuť. Preto sú potrebné metódy, ktoré urýchlia vývoj a analýzu týchto protokolov. Navyše ak sú tieto metódy použité pre certifikáciu protokolov, potom musia byť matematicky precízne, aby boli dostupné presné údaje o rozsahu a význame výsledkov analýzy. Túto úlohu je možné splniť pomocou formálnych metód. Využitím týchto metód sa v posledných desaťročiach vytvorilo množstvo automatizovaných a polo-automatizovaných nástrojov, ktoré dokážu vyhodnotiť formálnu bezpečnosť rôznych kryptografických protokolov.

V tejto diplomovej práci sa zameriam na kryptografiu, kde sa zmienim o jej histórii a rôznych použitých historických metódach. Popíšem jej základné rozdelenie a zameriam sa na problémy teórie čísel, ktoré musí riešiť, aby bola zabezpečená čo najväčšia bezpečnosť. Ďalej popíšem najpoužívanejšie metódy, ktoré sa používajú pri vyhodnotení formálnej bezpečnosti protokolov. Nakoniec prevediem analýzu protokolov Kerberos, EKE a protokolov jednosmernej autentizácie využívajúcich symetrickú kryptografiu, HMAC funkciu a hashovaciu funkciu. Tieto protokoly budú postupne testované v nástrojoch AVISPA, ProVerif a Scyther. Popíšem spôsob práce s týmito nástrojmi a vykonám analýzu bezpečnosti vybraných protokolov.

1 ÚVOD DO KRYPTOGRAFIE

Kryptografia je veda využívajúca matematiku k prevedeniu textu do podoby, ktorá je čitateľná len zo špeciálnou znalosťou. Umožňuje ukladať citlivé informácie alebo prenášať ich cez nezabezpečené siete (ako je internet), takže nemôžu byť prečítané nikým iným, okrem toho, kto pozná potrebné tajomstvo (kľúč). Stala sa modernou vedou s praktickým dopadom na ochranu súkromia, na bezpečnosť elektronických systémov a na bezpečnosť elektronickej komunikácie. Významnou aplikáciou kryptografie je oblasť informačnej bezpečnosti, oblasť ochrany autorských práv a autorizácie dokumentov.

Prvé zmienky o kryptografii môžeme nájsť pred niekoľkými tisíckami rokov. Celé obdobie môžeme rozdeliť do dvoch častí. Tou prvou je klasická kryptografia, ktorá približne trvala až do prvej polovice 20. storočia. Táto epocha sa vyznačovala tým, že k šifrovaniu stačilo len pero a papier. Behom prvej polovice 20. storočia ale začali vznikať sofistikované prístroje, ktoré umožňovali zložitejší postup pri šifrovaní. Tým približne začala druhá časť, ktorú nazývame moderná kryptografia.

1.1 Klasická kryptografia

K prvým použitým metódam patrí steganografia. Ide o ukrývanie správy ako takej. Sem patria rôzne neviditeľné atramenty, vyrývanie správy do drevenej tabuľky, ktorá sa zaleje voskom a pod. Ďalšou metódou sú substitučné šifry, ktoré spočívajú v nahradení každého znaku správy iným znakom podľa nejakého pravidla. K významným šifrám patrila Caesarova šifra, ktorá spočívala v posune písmen. Je pomenovaná po Juliovi Caesarovi, ktorý ju pravdepodobne používal ako prvý. Každé písmeno tajnej správy je posunuté v abecede o pevný počet pozíc. K ďalším metódam patrila tabuľka zámen, ktorá je založená na zámene znaku za iný bez akejkoľvek vnútornej súvislosti či na základe znalosti kľúča.

Významné šifry boli aj aditivné šifry, a to Vigenerova šifra a Vernamova šifra. Vigenerova šifra používa heslo, ktorého znaky určujú posunutie otvoreného textu tak, že otvorený text sa rozdelí na bloky znakov rovnako dlhé ako je heslo a každý znak sa sčíta s odpovedajúcim znakom hesla. Vernamova šifra podobne ako Vigenerova spočíva v sčítaní písmen otvoreného textu a hesla, avšak heslo je blok náhodne zvolených dát s rovnakou veľkosťou ako je otvorený text. Pokiaľ sú splnené podmienky úplne náhodného kľúča, je táto metóda absolútne bezpečná.

Ďalšou metódou sú transpozičné šifry, ktoré spočívajú v zmene poradia znakov podľa určitého pravidla. Napríklad tak, že otvorený text je zapísaný do tabuľky po riadkoch a šifrovaný text vznikne čítaním stĺpcov tejto tabuľky. Veľmi významnou

metódou bola počas 2. svetovej vojny Nemcami používaná Enigma. Ide o mechanický stroj, ktorý robil pomerne zložité operácie so vstupným textom, ale zároveň sa dal pomerne jednoducho ovládať. Už počas vojny sa ju podarilo prelomiť.

1.2 Moderná kryptografia

Moderná kryptografia je spojená s rozvojom elektronickej formy komunikácie, kde zachytenie prenášanej správy je relatívne jednoduché a technicky realizovateľné. Elektronická forma komunikácie, ktorá prešla od drôtovej komunikácie (telefón, telegraf) k bezdrôtovej (rádiovej) komunikácii, vyžadovala zdokonalenie šifrovacích algoritmov. Prenosový kanál je ľahké monitorovať, odpočúvať a teda prenášanú správu nie je prakticky možné chrániť bez použitia šifrovania. Modernú kryptografiu delíme na symetrickú a na asymetrickú kryptografiu.

Keďže základom všetkých kryptografických protokolov sú rôzne matematické funkcie, moderná kryptografia musí riešiť niekoľko problémov teórie čísel, aby bola zabezpečená čo najväčšia bezpečnosť. Medzi nich patrí hlavne faktorizácia čísel (RSA), problém diskrétneho logaritmu (protokol Diffie-Hellman) a problém eliptického diskrétneho logaritmu (protokol ECDH).

Problém faktorizácie čísel

Podstatou tohto problému je rozloženie čísla na súčin menších čísel. Tento problém využíva napríklad algoritmus RSA. U RSA ide o určenie kladného celého čísla n, ktoré je produktom dvoch odlišných nepárnych prvočísel p a q, presnejšie ich súčinom $n = pq$.

Princíp algoritmu RSA je nasledujúci: najskôr sa vypočíta hodnota Eulerovej funkcie $\varphi(n) = (p-1)(q-1)$. Zvolí sa celé číslo e menšie než $\varphi(n)$, ktoré je s $\varphi(n)$ nesúdeliteľné. Ďalej sa nájde číslo d tak, aby platilo $de \equiv 1 (\mathrm{mod}\ \varphi(n))$, kde symbol $\equiv$ značí kongruenciu zbytkových tried. Ak e je prvočíslo, tak $d = (1 + r * \varphi(n))/e$, kde $r = [(e-1)\varphi(n)^{(e-2)}]$. Verejným kľúčom je potom dvojica (n, e), pričom n sa označuje ako modul a e ako šifrovací či verejný exponent. Tento kľúč sa odošle opačnej strane. Súkromným kľúčom je dvojica (n, d), kde d sa označuje ako dešifrovací či súkromný exponent. Tento kľúč ostáva na strane odosielateľa.

Zašifrovanie správy z, kde platí $z < n$, sa potom prevedie na šifrovanú správu c pomocou vzorca $c = z^e \bmod n$. Dešifrovanie prebieha tak, že po prijatí šifrovanej správy c získa opačná strana pôvodnú nezašifrovanú správu pomocou vzorca $z = c^d \bmod n$. Podrobnejšie je tento algoritmus popísaný v [1].

Problém diskrétneho logaritmu

Nech a je primitívna odmocnina celého čísla $n > 1$. Ak najväčší spoločný deliteľ $NSD(b, n) = 1$, potom najmenšie kladné celé číslo i pre ktoré $b \equiv a^i (\text{mod } n)$, kde $1 \leq i \leq \phi(n)$, sa nazýva diskrétny logaritmus čísla b o základe a (mod n). $\phi(n)$ je rád celého čísla a mod n.

Predpokladajme vzťah $b \equiv a^i (\text{mod } p)$. Ak je dané i, a a prvočíslo p, nie je obtiažne vypočítať $b \equiv a^i (\text{mod } p)$. Ak je však dané prvočíslo p, generátor a cyklickej grupy $\mathbb{Z}_p^*$ a $b \in \mathbb{Z}_p^*$, je výpočtovo obtiažne nájsť i také, že $b \equiv a^i (\text{mod } p)$, $i \in < 1, p-1 >$, teda vyriešiť tzv. problém diskrétneho logaritmu. Viac je možné nájsť v [2].

Na probléme diskrétneho logaritmu je založená celá rada kryptografických algoritmov, z nich najznámejší je Diffie-Hellman. Diffie-Hellman umožňuje cez nezabezpečený kanál bezpečne vytvoriť kľúč pre symetrické šifry, protokol je ale náchylný na útok Muž uprostred (Man in the middle). Výsledkom je vytvorenie symetrického šifrovacieho kľúča, ktorý môže byť následne použitý pre šifrovanie zbytku komunikácie. Podstatou je generovanie zdieľaného zabezpečeného čísla ZZ, ktoré je generované následovne: $ZZ = g^{(x_b * x_a)} \text{ mod } p$, pričom jednotlivé strany vykonajú výpočet: $ZZ = (y_b^{x_a} \text{ mod } p = y_a^{x_b} \text{ mod } p)$, kde y_a je verejný kľúč strany a vypočítaný podľa $y_a = g^{x_a} \text{ mod } p$; y_b je verejný kľúč strany b vypočítaný podľa $y_b = g^{x_b} \text{ mod } p$; x_a je súkromný kľúč strany a; x_b je súkromný kľúč strany b; $g = h^{(p-1)/q} \text{ mod } p$, kde h je hocijaké celé číslo z $1 < h < p-1$ také, aby platilo $h^{(p-1)/q} \text{ mod } p > 1$; p a q sú veľké prvočísla. Podrobnejšie je Diffie-Hellman popísaný v [3].

Problém eliptického diskrétneho logaritmu

Jedná sa o analógiu už existujúcich systémov s verejným kľúčom, kde je modulárna aritmetika nahradená aritmetikou budovanou na základe operácií s bodmi na eliptickej krivke. Hierarchicky sa volia dva typy algebraických štruktúr: konečné teleso a eliptická krivka reprezentujúca skupinu bodov, nad ktorou je vlastný asymetrický algoritmus definovaný. Bezpečnosť spočíva v obtiažnosti riešenia úlohy diskrétneho logaritmu pre eliptické krivky.

Majme body $Q, R \in E_p(a, b)$ a rovnicu $Q = kR$, kde $k < p$. Zo znalosti R a k ľahko vypočítame Q, ale vypočítať k len zo znalosti Q a R je výpočtovo obtiažne. Kladné číslo k sa nazýva diskrétny logaritmus bodu Q vzhľadom k základu R nad eliptickou krivkou E. Podrobný popis je možné nájsť v [4].

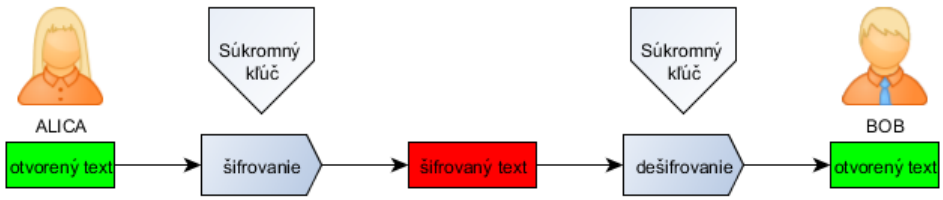
Na tomto probléme je založený protokol ECDH (Eliptic Curve Diffie-Hellman). Základom je vytvorenie zdieľaného kľúča. Komunikujúce strany sa musia najprv dohodnúť na parametroch (p, a, b, G, n, h), p je prvočíslo, ktorým definujeme teleso; konštanty a, b sú z rovnice eliptickej krivky; bod G je na eliptickej krivke; jeho rád

n a kofaktor h, ktorý udáva podiel počtu prvkov skupiny bodov na eliptickej krivke a rádu bodu G.

Každá strana si musí zvoliť svoje kľúče, ktoré sa skladajú zo súkromného kľúča d (náhodne vybrané celé kladné číslo z intervalu $< 1, n-1 >$) a verejného kľúča Q, kde $Q = dG$. Jedna strana bude mať dvojicu d_A, Q_A a druhá d_B, Q_B. Obe strany si medzi sebou vymenia verejné kľúče Q_A a Q_B. Po výmene môže prvá strana nájsť bod $Z = d_A Q_B$ a druhá $Z' = d_B Q_A$. Tieto body Z, Z' sú totožné, pretože $Z = d_A Q_B = d_A(d_B G) = d_A d_B G = d_B(d_A G) = d_B Q_A = Z'$. [4]

1.2.1 Symetrická

Na šifrovanie a dešifrovanie dát sa používa rovnaký tajný kľúč (obr. 1.1) a ich bezpečnosť je založená na utajení tohto kľúča. Podstatnou výhodou je ich nízka výpočtová náročnosť. Na druhej strane veľkou nevýhodou je nutnosť zdieľania tajného kľúča, takže sa odosielateľ a príjemca tajnej správy musia dopredu dohodnúť na tajnom kľúči. Medzi v praxi používané šifry patrí AES (Advanced Encryption Standard), Triple DES (Data Encryption Standard), RC4 (Ron's Code, prelomený, ale je možné sa s ním stretnúť u protokolu WEP), a ďalšie. Symetrické šifry môžeme rozdeliť na blokové symetrické šifry a prúdové symetrické šifry.



Obr. 1.1: Šifrovanie a dešifrovanie dát pomocou jedného kľúča.

Blokové symetrické šifry realizujú proces šifrovania resp. dešifrovania po blokoch. Veľa algoritmov symetrických blokových šifier používa štruktúru, kde sa otvorený text o dĺžke n-bitov transformuje na zašifrovaný text o rovnakej dĺžke n-bitov. Pretože blok o dĺžke n-bitov môže vytvoriť 2^n rôznych blokov otvoreného textu, bloková šifra realizuje transformáciu týchto 2^n blokov otvoreného textu na 2^n blokov zašifrovaného textu o dĺžke n-bitov.

Medzi výhody blokových šifier patrí vysoká úroveň difúzie (zašifrované bloky vykazujú odlišné frekvenčné a štatistické charakteristiky od pôvodných nezašifrovaných blokov) a odolnosť voči narušeniu (do bloku nie je možné pridať žiadny znak, pretože by sa zmenila dĺžka bloku, pri dešifrovaní by potom bola odhalená modifikácia). Medzi nevýhody blokových šifier patrí oneskorenie (dešifrovanie prebieha až

po prijatí celého bloku) a šírenie chýb (zlé prijatie jedného znaku sa premietne do celého bloku). K predstaviteľom patrí už prelomená šifra DES alebo v praxi používaná šifra AES.

Prúdové symetrické šifry spracovávajú otvorený text bit po bite, v niektorých prípadoch byte po byte. Jednotlivé bity sú kombinované (typicky pomocou funkcie XOR) s pseudonáhodným prúdom bitov tzv. keystreamom, ktorý je vytvorený na základe tajného kľúča a šifrovacieho algoritmu. Prúdové šifry sa podľa spôsobu tvorenia keystreamu ďalej delia na synchronné a asynchronné. K predstaviteľom patrí napríklad RC4, Rabbit, Salsa20, Trivium, Grain a iné.

U synchronných šifier je keystream generovaný nezávisle na otvorenom a zašifrovanom texte. Generovanie keystreamu závisí len na aktuálnom stave šifrovacieho algoritmu a na tajnom kľúči. Pri používaní synchronných prúdových šifier musia byť komunikujúce strany synchronizované, v tomto prípade to znamená, že musia zdieľať nie len tajný kľúč, ale aj stav algoritmu prúdovej šifry. Pokiaľ dôjde pri prenose ku zmene jedného šifrovacieho znaku za iný, tak pri dešifrovaní bude ovplyvnený len tento jeden znak.

Asynchronné prúdové šifry využívajú ku generovaniu keystreamu tajný kľúč a určitý fixný počet predchádzajúcich znakov šifrovaného textu. Podobne ako u synchronnej šifry by mali byť komunikujúce strany synchronizované. Na rozdiel od synchronných sú schopné sa po určitom počte znakov samy zosynchronizovať. Pokiaľ dôjde pri prenose k zmene jedného znaku za iný, tak pri dešifrovaní bude ovplyvnený len fixný počet následujúcich znakov. Dešifrovanie zbytku kryptogramu by malo prebehnúť bezo zmeny.

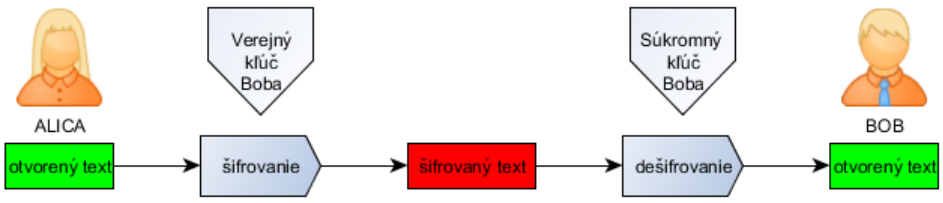
Medzi výhody prúdových šifier patrí vysoká rýchlosť šifrovania a dešifrovania (šifrovanie a dešifrovanie znaku je nezávislé na ostatných znakoch správy) a malé šírenie chýb (každý znak je šifrovaný samostatne). Medzi nevýhody patrí nízka úroveň difúzie (zašifrovaný text môže vykazovať rovnaké frekvenčné a štatistické charakteristiky ako pôvodný otvorený text, čo uľahčuje kryptoanalýzu) a malá odolnosť voči úmyselným falzifikáciám (v prípade prelomenia šifry môže byť prenášaný text modifikovaný bez toho, aby príjemca modifikáciu rozpoznal).

1.2.2 Asymetrická

Z pohľadu bezpečnosti má bežná internetová komunikácia dva podstatné nedostatky. Odosielateľ a príjemca dát nemajú istotu, že si dáta po ceste neprečíta tretia osoba. Príjemcovi zároveň chýba záruka, že prijaté dáta naozaj pochádzajú od deklarovaného odosielateľa. Ako riešenie oboch menovaných problémov sa ukazuje asymetrická kryptografia, garantujúca pravosť identity odosielateľa a diskrétny prenos dát.

K zástupcom patrí napríklad RSA (Rivest-Shamir-Adleman) alebo DSA (Digital Signature Algorithm).

Na rozdiel od symetrickej kryptografie sa nepracuje s jedným kľúčom, ale s kľúčovým párom (obr. 1.2). V tomto páre je prvý kľúč súkromný a druhý verejný. Kľúče medzi sebou majú určitú matematickú súvislosť, avšak súkromný kľúč prakticky nie je možné odvodiť z verejného kľúča. Podstata asymetrickej kryptografie spočíva v tom, že k zašifrovaniu dát sa používa jeden kľúč a k ich dešifrovaniu druhý kľúč. Kým súkromný kľúč jeho držiteľ uchováva v tajnosti, verejný kľúč môžeme voľne distribuovať otvorenými komunikačnými kanálmi. Tým odpadá nutnosť si dopredu bezpečnou cestou vymeniť spoločný šifrovací kľúč.



Obr. 1.2: Príklad asymetrického šifrovania.

Využitie asymetrickej kryptografie je v zásade dvojité a vychádza z poradia použitia šifrovacích kľúčov. V prvom prípade je cieľom zaistenie dôvernosti prenášaných dát. Ľubovoľný odosielateľ môže zašifrovať dáta pomocou verejného kľúča príjemcu. Vzhľadom k výpočtovej náročnosti asymetrickej kryptografie sa v praxi pre šifrovanie dát používa symetrická kryptografia, asymetrická kryptografia sa použije len pre bezpečné doručenie symetrického šifrovacieho kľúča.

V druhom prípade, kde je postup opačný, ide o určenie totožnosti odosielateľa a zaručenie integrity dát. Zo správy sa pomocou hashovacej funkcie vytvorí takzvaný otlačok, ktorý odosielateľ zašifruje svojím súkromným kľúčom. Pretože týmto spôsobom môže dáta zašifrovať jedine držiteľ súkromného kľúča, má príjemca istotu ohľadne osoby odosielateľa dát. Integrita dát je overená porovnaním zašifrovaného otlačku dát s otlačkom, ktorý si z prijatých dát vygeneruje príjemca. Na tomto princípe je založený elektronický podpis. [5]

2 FORMÁLNA BEZPEČNOSŤ KRYPTOGRAFICKÝCH PROTOKOLOV

Počítačová sieť môže obsahovať votrelcov, ktorí môžu čítať, upravovať a mazať stopy a môžu ovládať jeden alebo viac sieťových princípov. Vzhľadom k tomu, protokoly sú často predmetom neintuitívnych útokov, ktoré nie sú ľahko viditeľné ani starostlivým inšpektorom. Najmä keď sa predpoklady o prostredí, v ktorom protokol pracuje, menia.

Z týchto dôvodov sa zistilo, že formálne metódy môžu byť užitočné pri analyzovaní bezpečnosti kryptografických protokolov. Umožňujú urobiť oboje, dôkladnú analýzu odlišných ciest, ktoré môže útočník použiť a presne určiť predpoklady prostredia, ktoré boli vykonané. Podľa [6] a [7] patria k najpoužívanejším metódam formálnej analýzy Dolev-Yao model a BAN (Burrows, Abadi, Needham) logika.

2.1 Dolev-Yao model

Dolev a Yao vytvorili model [8], ktorý umožňuje študovať problém zabezpečenia protokolov, s veľmi malými predpokladmi správania sabotéra. V systéme s verejným kľúčom má každý užívateľ X funkciu šifrovania E_x a dešifrovania D_x. Oboje sú mapované z $\{0,1\}^*$ (množina všetkých konečných binárnych sekvencií) do $\{0,1\}^*$. Zabezpečený verejný adresár obsahuje všetky (X, E_x) páry, kým dešifrovacia funkcia je známa len užívateľovi X.

2.1.1 Kaskádové protokoly

V tejto časti sa uvažuje jednoduchá trieda protokolov, v ktorých jediné operácie využívané užívateľom na generovanie správy sú šifrovanie a dešifrovanie. Cieľom je analýza bezpečnosti týchto protokolov proti sabotérovi. Aby sa to dosiahlo, je potrebné špecifikovať syntax protokolu, t. j. aké operácie užívateľ aplikuje v každom kroku generovania správy, a pravidlá, ktoré môže sabotér použiť na získanie správy.

Notácia

Nech Σ je konečná sada rôznych symbolov. Použijeme Σ^* na množinu všetkých konečných postupností zložených zo symbolov Σ, sada Σ^* tiež obsahuje prázdny reťazec λ. Je definované $\Sigma^+ = \Sigma^*$ - $\{\lambda\}$, t. j. sada neprázdnych slov cez Σ. Zreťazenie slov α a β je označené $\alpha\beta$. Nech $\gamma = \alpha\beta$ je slovo, kde α je nazývaná predponou γ a β je príponou γ.

Základnou vlastnosťou operátorov s verejným kľúčom sú $E_x D_x = D_x E_x = 1$, funkcia identity. Ako výsledok, akýkoľvek reťazec operátorov formy $\sigma E_x D_x \sigma'$ bude ekvivalent $\sigma\sigma'$ v zmysle, že $(\sigma E_x D_x \sigma')\mathrm{P} = (\sigma\sigma')\mathrm{P}$ pre všetky $\mathrm{P} \in \{0,1\}^*$. Pre ľubovoľný reťazec γ operátorov, nech $\gamma|_x$ označuje kompletne redukovaný reťazec z γ vymazaním všetkých E_x a D_x párov iteratívne, kým nie je potrebná žiadna ďalšia redukcia. $\bar{\gamma}$ je redukovaná forma γ. Pre hocijaký reťazec γ, nech $lt(\gamma)$ je množina symbolov v γ.

Model

Dvojstranný kaskádový protokol T je špecifikovaný sériou konečných reťazcov

$$\begin{aligned} \tilde{\alpha_i} &\in \{z_1, z_2, z_3\}^*, \qquad 1 \le i \le t \\ \tilde{\beta_i} &\in \{z_1, z_2, z_4\}^*, \qquad 1 \le i \le t', \end{aligned}$$

kde $t' = t$ alebo $t-1$. Pre každý pár odlišných užívateľov X a Y, nech $\alpha_i(X,Y)$, $\beta_i(X,Y)$ označuje reťazec α_i, β_i so symbolmi z_1, z_2, z_3, z_4, respektíve zamenené za E_X, E_Y, D_X, D_Y.

Jasnejšie $\alpha_i(X,Y) \in \{E_X, E_Y, D_X\}^*$ a $\beta_i(X,Y) \in \{E_X, E_Y, D_Y\}^*$. Keď užívateľ X chce preniesť zabezpečenú správu M užívateľovi Y, vymieňajú si správy podľa T. Protokol je jednotný v tom, že $\alpha_i(A,B)$ a $\beta_i(A,B)$ pre akýchkoľvek užívateľov A, B môžu byť získané z $\alpha_i(X,Y)$ a $\beta_i(X,Y)$.

Nech T je dvojstranný kaskádový protokol špecifikovaný $\{\tilde{\alpha_i}, \tilde{\beta_j} | 1 \le i \le t, 1 \le j \le t'\}$, a nech (X,Y) sú odlišný užívatelia. Je definované:

$$\begin{aligned} N_1(X,Y) &= \alpha_1(X,Y), \\ N_{2j}(X,Y) &= \beta_j(X,Y) N_{2j-1}(X,Y), \qquad 1 \le j \le t', \\ N_{2i+1}(X,Y) &= \alpha_{i+1}(X,Y) N_{2i}(X,Y), \qquad 1 \le i \le t-1. \end{aligned}$$

Keď X chce odoslať správu M na Y, vymieňaná správa je potom $N_i(X,Y)M$, kde $i = 1, 2, \ldots, t+t'$.

Nech T je dvojstranný kaskádový protokol špecifikovaný $\{\tilde{\alpha_i}, \tilde{\beta_j}\}$. Je definované:

$$\begin{aligned} \Sigma_1(Z) &= E \cup \{D_Z\}, \\ \Sigma_2 &= \{\alpha_i(A,B) | \text{ pre všetky } A \ne B \text{ a } i \ge 2\}, \\ \Sigma_3 &= \{\beta_i(A,B) | \text{ pre všetky } A \ne B \text{ a } i \ge 1\}. \end{aligned}$$

T je ohrozené, ak nejaké $\gamma \in (\Sigma_1(Z) \cup \Sigma_2 \cup \Sigma_3)^*$ také, že existuje $\overline{\gamma N_i(X,Y)} = \lambda$ pre nejaké $N_i(X,Y)$. Jednoducho povedané, definícia bezpečnosti pre T je nezávislá na výbere X, Y, Z.

Ak sa X snaží odoslať správu M na Y (použitím protokolu T), samotné správy prenášané medzi nimi sú potom $N_i(X,Y)M$ $(i = 1, 2, \ldots)$ a môžu padnúť do rúk

sabotéra Z. Použitím akejkoľvek $N_i(X,Y)M$, sabotér Z má šancu transformovať ju opakovane použitím niektorého z troch nasledujúcich typov operátorov:

a) akékoľvek $\sigma \in \Sigma_1(Z)$;
b) akékoľvek $\sigma \in \Sigma_3 : Z$ môže začať prenos správy s užívateľom B, tvrdiac, že je užívateľom A a odoslať akýkoľvek reťazec P na B v $(2i-1)$ správe; Z potom dostane späť $\beta_i(A,B)P$, účinne uvedeným operátorom $\beta_i(A,B)$ na vybranom P;
c) akékoľvek $\sigma \in \Sigma_3$: nech $\sigma = \alpha_i(A,B)$; je šanca, že A môže chcieť niekedy v budúcnosti preniesť správu na B; Z môže zachytiť $(i-1)$ odpoveď z B na A, zabrániť jej dosiahnutiu A a zameniť ju s hocijakým reťazcom P a prijať z A reťazec $\alpha_i(A,B)P$.

Ako výsledok, Z má možnosť získať reťazec $\gamma N_i(X,Y)M$ pre akékoľvek $\gamma \in (\Sigma_1(Z) \cup \Sigma_2 \cup \Sigma_3)^*$. To znamená, že Z môže odvodiť M, ak $\gamma N_i(X,Y) = \lambda$ pre nejaké $\gamma \in (\Sigma_1(Z) \cup \Sigma_2 \cup \Sigma_3)^*$.

Charakterizácia bezpečnostných protokolov

Nech $\pi \in \{E,D\}^*$ je reťazec a A je užívateľovo meno. π ma vyvažovaciu vlastnosť vzhľadom k A, ak $D_A \in lt(\pi)$ obsahuje $E_A \in lt(\pi)$. Zabezpečovacia vlastnosť je neoddeliteľná súčasť zabezpečenia kaskádových protokolov.

Nech X, Y sú dve odlišné užívateľské mená. Dvojstranný kaskádový protokol $T = \{\tilde{\alpha}_i, \tilde{\beta}_j\}$ je vyvážený kaskádový protokol ak

1) pre každé $i \geq 2$, $\alpha_i(X,Y)$ má vyváženú vlastnosť vzhľadom k X,
2) pre každé $i \geq 1$, $\beta_i(X,Y)$ má vyváženú vlastnosť vzhľadom k Y.

Dvojstavový kaskádový protokol $T = \{\tilde{\alpha}_i, \tilde{\beta}_j\}$ zabezpečený iba ak $lt(\alpha_1(X,Y)) \cap \{E_X, E_Y\} \neq \varnothing$ a T je vyvážené. Každý dvojnásobne overený protokol je nezabezpečený. Dôkazy všetkých tvrdení je možné nájsť v [8].

2.1.2 Name-Stamp protokoly

V tejto časti bude popísaný model, kde protokoly pripájajú pred šifrovaním meno do správy.

Neformálny popis

Je predpoklad, že mená všetkých užívateľov majú rovnakú dĺžku m bitov. Pre hocijaký reťazec $\gamma \in \{0,1\}^*$, kde $\gamma =$ head(γ)tail(γ) (head(γ) je hlavička o dĺžke n bitov a tail(γ) je suffix o dĺžke s bitov), môže užívateľ Y použiť niektorú z nasledujúcich operácií:

a) šifrovanie E_X;
b) dešifrovanie D_Y;

c) pripojenie i_X; s $i_X\gamma = \gamma X$;
d) porovnanie názvov d_X; s $d_X\gamma =$ head(γ) ak head$(\gamma) = X$;
e) vymazanie d, s $d\gamma =$ head(γ).

Podľa name-stamp protokolu, ľubovoľný užívateľom prenášaný text sa získa použitím operácií a)-e) na posledný prijatý text. Na uistenie dokončenia komunikácie sa požaduje, aby akýkoľvek text prenášaný medzi dvoma bežnými užívateľmi X, Y mal formu $\gamma \in \{E_A, D_A, i_A, d_A, d | \text{všetkých užívateľov } A\}^* M$.

Sabotér má možnosť zachytiť všetky správy medzi X a Y, modifikovať ich operáciami a)-e) a použiť ich voľne v každom rozhovore, ktorý sa začal buď ním, alebo inými.

Formálny popis

Dvojstranný name-stamp protokol T je špecifikovaný sadou reťazcov

$$\tilde{\alpha}_i \in (F - \{z_2\})^*, \quad \tilde{\beta}_j \in (F - \{z_1\})^*,$$

kde $F = \{z_1, z_2, \ldots, z_9\}$, $1 \le i \le t$ a $1 \le j \le t'$ ($t' = t$ alebo $t-1$). Nech $\alpha_i(X, Y)$ a $\beta_j(X, Y)$ označujú reťazce $\tilde{\alpha}_i$ a $\tilde{\beta}_j$, kde $z_1, z_2, \ldots, z_9$ sú nahradené D_X, D_Y, E_X, E_Y, i_X, i_Y, d_X, d_Y, d.

Nech X, Y, Z sú traja odlišní užívatelia. Dvoj-stavový name-stamp protokol T je nezabezpečený ak existuje reťazec $\gamma \in V^*_{Z,T}\{\overline{N_i(X,Y)}\}$ taký, že $\overline{\gamma} = \lambda$; kde $\overline{N_i(X,Y)}$ je sekvencia správ prenášaná medzi X a Y. Množina $V_{Z,T}$ je definovaná

$$\begin{aligned} V_{Z,T} = \{&\alpha_j(A,B) | \text{ všetky } A \neq B, \text{ všetky } j \ge 2\} \\ &\cup \{\beta_j(A,B) | \text{ všetky } A \neq B, \text{ všetky } j\} \\ &\cup \{E_A, i_A, d_A, d | \text{ všetky } A\} \cup \{D_Z\}. \end{aligned}$$

Algoritmus na kontrolu bezpečnosti protokolu

Pre daný dvojstranný name-stamp protokol T, špecifikovaný $\{\alpha_i, \beta_j\}$, sa použije n na označenie vstupnej dĺžky $\Sigma_i|\alpha_i| + \Sigma_j|\beta_j|$. V princípe sabotér Z môže začať konverzáciu s hocijakým užívateľom v sieti. Pre zjednodušenie sa určí, že Z komunikuje len s X a Y. Ďalej je definované:

$$\begin{aligned} S = \{&\alpha_i(A,B) | A, B \in \{X, Y, Z\}, A \neq B, i \ge 2\} \\ &\cup \{\beta_i(A,B) | A, B \in \{X, Y, Z\}, A \neq B\} \\ &\cup \{E_A, i_A, d_A, d | A = X, Y, Z\} \cup \{D_Z\}. \end{aligned}$$

Potom protokol T je nezabezpečený len ak reťazec $\gamma \in S^*\{N_i(X,Y)\}$ existuje také, aby $\bar{\gamma} = \lambda$. Nech $\eta \in \{E_A, D_A, i_A, d | A = X, Y\}^*$ je nezredukovateľný reťazec. Ďalej je

definovaná množina $C(\eta)$, ktorá je množinou všetkých nezredukovateľných reťazcov $\delta \in \{E_A, D_A, i_A, d_A, d|$ všetkých $A\}^*$ splňujúcich $\overline{\delta\eta} = \lambda$. Ak η obsahuje nejaké d, potom $C(\eta) = \varnothing$. Inak nech $\eta = b_1, b_2, \ldots, b_i$, potom $C(\eta)$ obsahuje všetky reťazce $b_t^c b_{t-1}^c \ \ldots \ b_1^c$, kde $(E_A)^c = D_A, (D_A)^c = E_A, (i_A)^c = d_A$ alebo d.

Protokol T je ďalej nezabezpečený len ak reťazec $\gamma \in S^*$ existuje taký, aby $\bar{\gamma} \in C(\rho_{ij})$ pre nejaké $1 \leq j \leq s$. Nastáva tu daný problém: Daná je množina reťazcov $S = \{h_1, h_2, \ldots, h_p\}$ a reťazec ρ, kde

$$h_i \in \{E_A, D_A, i_A, d_A, d|A = X, Y, Z\}^*$$
$$\text{a } \rho \in \{E_A, D_A, i_A|A = X, Y\}^*,$$

niekto môže v čase $O(q^7)$ rozhodnúť, či reťazec $\gamma \in S^*$ existuje taký, aby $\bar{\gamma} \in C(\rho)$. (q je definované ako $\sum_{i=1}^{n} |h_i| + |\rho|$).

Rozšírená slovná úloha

Nech $\Sigma = \{a_1, a_2, \ldots, a_r\}$ je abeceda, t.j. množina odlišných symbolov. $u \to v$ je nazývané transformačné pravidlo, kde $u \in \Sigma^+$ a $v \in \Sigma^*$. Nech $\Gamma = \{u_1 \to v_1, u_2 \to v_2, \ldots, u_q \to v_q\}$ je množina transformačných pravidiel. Pre dva reťazce γ, $\delta \in \Sigma^*$ sa napíše $\gamma \Rightarrow_\Gamma \delta$ ak γ môže byť transformovaná do δ opakovaným použitím pravidla Γ, t.j. výmenou podreťazca u_i za v_i. Pre reťazec podmnožín G_i z Σ, $\eta = G_1, G_2, \ldots, G_q$, nech $L(\eta) = \{\gamma|\gamma = g_1, g_2, \ldots, g_q\}$, kde $g_i \in G_i$. Použije sa notácia $\gamma \Rightarrow_\Gamma L(\eta)$ ak $\gamma \Rightarrow_\Gamma \rho$ pre nejaké $\rho \in L(\eta)$. Rozšírená slovná úloha pre Σ, Γ môže byť uvedené nasledovne: Daná je množina vstupných reťazcov $\delta_1, \delta_2, \ldots, \delta_p (\delta_j \in \Sigma^*)$ a reťazec podmnožín $\eta = G_1, G_2, \ldots, G_q (G_i \subseteq \Sigma; G_i \neq \varnothing)$. Je potrebné zistiť, či konkatenancia $\Delta = \delta_{i1}, \delta_{i2}, \ldots, \delta_{is}$ existuje taká, aby $\Delta \Rightarrow_\Gamma L(\eta)$. Rozšírená slovná úloha pre Σ, Γ môže byť vyriešená v čase $O(n^7)$, kde n je vstupná dĺžka.

2.1.3 Netrpezlivý sabotér

Na prelomenie protokolu, ktorý je nezabezpečený, je potrebné, aby bol sabotér príjemcom konverzácie. V tejto časti sa popíše charakterizácia protokolov, ktoré môžu byť ohrozené netrpezlivým sabotérom, t.j. takým, ktorý len iniciuje konverzácie. Existuje algoritmus, ktorý môže v čase $O(n^8)$ rozhodnúť, či daný dvoj-stavový namestamp protokol T je zabezpečený voči netrpezlivému sabotérovi. Pre kaskádové protokoly by mala byť definícia bezpečnosti modifikovaná nasledovne: T je nezabezpečené (proti netrpezlivému sabotérovi) ak nejaké $\gamma \in (\Sigma_1(Z) \cup \Sigma_3)^*$ existuje také, aby $\overline{\gamma N_i(X, Y)} = \lambda$ pre nejaké $N_i(X, Y)$; inak je T zabezpečené.

Nech X, Y sú odlišné užívateľské mená. Dvoj-stavový kaskádový protokol $T = \{\tilde{\alpha}_i, \tilde{\beta}_j\}$ je zabezpečený proti netrpezlivému sabotérovi len ak pre každé $k \geq 1$:

1. $lt(\overline{N_k(X, Y)}) \cap \{E_X, E_Y\} \neq \varnothing$

2. $\beta_k(X, Y)$ má vlastnosť vyvažovania vzhľadom k Y.

Ďalej nech A je nejaké užívateľské meno a nech η je reťazec. Podreťazec π z η je nazývaný A-podreťazec ak jedno z nasledujúcich je pravdivé pre nejaké $X, Y \neq A$:

1. $\eta = \eta_1 D_X \pi D_Y \eta_2$;
2. $\eta = \eta_1 D_X \pi$;
3. $\eta = \pi D_Y \eta_2$.

Reťazec η je silne A-balancovaný ak každé A podreťazec π má vyvažovaciu vlastnosť vzhľadom k A (viď kaskádové protokoly). Potom nech η je silne A-balancovaný reťazec. Ak $\eta = \eta_1 D_B \eta_2$ s $B \neq A$, potom η_1 a η_2 sú oboje silne A-balancované. Ďalej ak $\eta = \eta_1 \eta_2$ a $E_A \notin lt(\eta_2)$, potom η_1 je silne A-balancovaná. Nech γ, δ sú silne A-balancované reťazce dané v redukovanej forme. Ak $(lt(\gamma) \cup lt(\delta)) \cap \{E_A, D_A\} \neq \varnothing$, potom $\overline{\gamma\delta} \neq \lambda$.

Dôkazy všetkých tvrdení daných v podkapitole o Dolev-Yao je možné nájsť v [8].

2.2 BAN logika

Autentizačné protokoly sú základom bezpečnosti v mnohých distribuovaných systémoch, a preto je potrebné zabezpečiť, aby tieto protokoly fungovali správne. Bohužiaľ, ich dizajn je veľmi náchylný k chybám. Množstvo protokolov, ktoré je možné nájsť v literatúre, obsahuje redundancie alebo bezpečnostné chyby. Pomocou jednoduchej logiky bolo umožnené opísať vieru dôveryhodných strán zapojených do autentizačných protokolov a vývoj týchto presvedčení v dôsledku komunikácie.

2.2.1 Základné pojmy

Burrows, Abadi a Needham [9] vytvorili logiku známu ako BAN logika (BAN sú iniciály ich mien). Správy sú identifikované príkazmi v logike. Typicky, symboly A, B a S označujú špecifické princípy; K_{ab}, K_{as} a K_{bs} označujú špecifické zdieľané kľúče; K_a, K_b a K_s označujú špecifické verejné kľúče, a K_a^{-1}, K_b^{-1} a K_s^{-1} označujú zodpovedajúce tajné kľúče; N_a, N_b a N_s označujú špecifické tvrdenia. Symboly P, Q a R sa pohybujú nad splnomocniteľmi; X, Y sa pohybujú nad tvrdeniami; K sa pohybuje nad šifrovacími kľúčmi.

Jediná výroková spojitosť je konjunkcia označená čiarkou. S konjunkciou sa zaobchádza ako so sadou a berú sa za samozrejmosť vlastnosti ako je asociatívnosť a komutatívnosť. Okrem konjunkcie sa používajú nasledovné konštrukcie:

- *P verí X*: alebo P má právo veriť X. Konkrétne principal (splnomocniteľ) P môže pôsobiť akoby X bolo pravdivé. Táto konštrukcia má zásadný význam pre logiku.

- P *vidí* X: Niekto odoslal správu obsahujúcu X do P, ktoré môže čítať a opakovať X (prípadne po spravení nejakého popisu).
- P *povedalo* X: Principal P v určitej dobe poslal správu obsahujúcu tvrdenie X. Nie je známe, či bola správa odoslaná dávno alebo pri súčasnom behu protokolu, ale je známe, že P potom veril X.
- P *kontroluje* X: P má právomoc nad X. Principal P je autorita na X a mal by byť dôveryhodný v tejto veci. Napríklad dôveryhodný server často správne generuje šifrovacie kľúče. To môže byť vyjadrené za predpokladu, ak principal verí, že server má právomoc nad tvrdeniami o kvalite kľúčov.
- *neopakovateľné*(X): Vzorec X je neopakovateľný, to znamená, že X nebolo odoslané v správe kedykoľvek pred aktuálnym behom protokolu. Toto zvyčajne platí pre kódové slovo. To znamená, že výrazy boli vynájdené pre účel byť neopakovateľné. Kódové slovo zvyčajne zahŕňa časovú pečiatku alebo číslo, ktoré sa používa iba raz.
- $P \overset{\mathrm{K}}{\leftrightarrow} Q$: P a Q môžu na komunikáciu používať zdieľaný kľúč K. Kľúč K je dobrý v tom, že nikdy nebude objavený akýmkoľvek splnomocniteľom okrem P alebo Q, alebo splnomocniteľom dôveryhodným pre P alebo Q.
- $\overset{\mathrm{K}}{\rightarrow} P$: P má K ako verejný kľúč. Zodpovedajúci tajný kľúč (označený K^{-1}) nebude nikdy objavený akýmkoľvek splnomocniteľom okrem P, alebo splnomocniteľom dôveryhodným pre P.
- $P \overset{\mathrm{X}}{\rightleftharpoons} Q$: Formula X je tajnosť známa len P a Q, prípadne splnomocniteľom dôveryhodným pre nich. Len P a Q môžu použiť X na preukázanie svojej identity voči sebe. Príkladom tajnosti je heslo.
- $\{X\}_K$: Značí to, že X je zašifrované kľúčom K.
- $\langle X \rangle_Y$: Predstavuje to X v kombinácii s Y. Predpokladá sa, že Y je tajné, a že jeho prítomnosť dokazuje totožnosť toho, kto prednesie $\langle X \rangle_Y$.

2.2.2 Logické predpoklady

Pri autentizácii je potrebné sa zaoberať rozdielom medzi dvoma epochami: *minulosť* a *súčasnosť*. Súčasná epocha začína na začiatku konkrétneho behu uvažovaného protokolu. Všetky správy odoslané do tejto doby sú považované za minulé a autentizačné protokoly by mali byť opatrné, aby sa zabránilo akceptovaniu takýchto správ ako súčasných. Všetky dôvery dané v súčasnosti sú stabilné po celú dobu behu protokolu. Avšak dôvery dané v minulosti nie sú nevyhnutne prevedené do súčasnosti.

Predpokladá sa, že šifrovanie garantuje, že každý zašifrovaný úsek nemôže byť zmenený alebo poskladaný dohromady z menších šifrovaných oddielov. Každá zašifrovaná správa obsahuje dostatočnú redundanciu, aby dovolila splnomocniteľovi,

ktorý dešifruje, overiť, že používa správny kľúč. Okrem toho, správy obsahujú dostatočné informácie pre splnomocniteľa odhaliť (a ignorovať) svoje vlastné správy.

Medzi hlavné logické predpoklady použité v dôkazoch patrí:

1) Pravidlá *významu správy* sa týkajú interpretácie správ. Dva z troch sa týkajú interpretácii šifrovaných správ a tretie sa týka interpretácii správ s tajomstvom. Všetky vysvetľujú, ako odvodiť dôveru o pôvode správ. Pre zdieľané kľúče sa postupuje podľa

$$\frac{P \text{ verí } Q \overset{\text{K}}{\leftrightarrow} P,\ P \text{ vidí } \{X\}_K}{P \text{ verí } Q \text{ povedalo } X}$$

To znamená, že ak principal P verí, že šifrovací kľúč K je zdieľaný s principalom Q a vie, že tvrdenie X bolo šifrované týmto kľúčom, potom P verí, že tvrdenie X bolo v určitej dobe poslané principalom Q. Pre toto pravidlo je zjavné, že je potrebné zabezpečiť, aby P neposlalo správu samo sebe; stačí pripomenúť, že $\{X\}_K$ značí vzorec v tvare $\{X\}_K$ z R, a vyžaduje, aby $R \neq P$. Podobne sa pre verejné kľúče postupuje podľa

$$\frac{P \text{ verí } \overset{\text{K}}{\leftrightarrow} Q,\ P \text{ vidí } \{X\}_{K^{-1}}}{P \text{ verí } Q \text{ povedalo } X}$$

Pre zdieľané tajomstvá postupujeme podľa

$$\frac{P \text{ verí } Q \overset{\text{Y}}{\rightleftharpoons} P,\ P \text{ vidí } \langle X\rangle_Y}{P \text{ verí } Q \text{ povedalo } X}$$

To znamená, že ak je tajomstvo Y zdieľané medzi P a Q a $\langle X\rangle_Y$ je známe pre P, potom P verí, že tvrdenie X bolo v určitej dobe poslané principalom Q. Tento postup je zjavný, pretože pravidlá pre viditeľnosť garantujú, že $\langle X\rangle_Y$ nebolo v rámci P poslané samo sebe.

2) Pravidlo *overenie kódového slova* vyjadruje kontrolu, že správa je neopakovateľná, a teda, že odosielateľ jej stále verí:

$$\frac{P \text{ verí neopakovateľné}(X),\ P \text{ verí } Q \text{ povedalo } X}{P \text{ verí } Q \text{ verí } X}$$

Ak P verí, že X je neopakovateľné a mohlo byť vyslovené len nedávno (v súčasnosti), a tvrdenie Y bolo v určitej dobe poslané principalom Q (buď v minulosti, alebo v prítomnosti), potom P verí, že zároveň Q verí tvrdeniu X. Pre jednoduchosť X musí byť obyčajný text; to znamená, že by nemalo obsahovať žiadne časti vzorca $\{Y\}_K$.

3) Pravidlo *príslušnosti* ustanovuje, že ak P verí, že Q má právomoc nad X, potom P verí dôvere Q v X. V konečnom dôsledku teda P dôveruje tvrdeniu X:

$$\frac{P \text{ verí } Q \text{ kontroluje } X,\ P \text{ verí } Q \text{ verí } X}{P \text{ verí } X}$$

4) Ak splnomocniteľ vidí vzorec, potom vidí aj jeho zložky, ak pozná potrebné kľúče:

$$\frac{P \text{ vidí } (X,Y)}{P \text{ vidí } X}, \quad \frac{P \text{ vidí } \langle X \rangle_Y}{P \text{ vidí } X}, \quad \frac{P \text{ verí } Q \overset{\text{K}}{\leftrightarrow} P,\ P \text{ vidí } \{X\}_K}{P \text{ vidí } X},$$
$$\frac{P \text{ verí } \overset{\text{K}}{\rightarrow} P,\ P \text{ vidí } \{X\}_K}{P \text{ vidí } X}, \quad \frac{P \text{ verí } \overset{\text{K}}{\rightarrow} Q,\ P \text{ vidí } \{X\}_{K^{-1}}}{P \text{ vidí } X}.$$

Štvrté pravidlo je odôvodnené implicitným predpokladom, že ak P verí, že K je jeho verejný kľúč, potom P pozná zodpovedajúci tajný kľúč K^{-1}. Ak P vidí X a P vidí Y z toho nevyplýva automaticky, že P vidí X a Y zároveň. Znamená to, že X a Y boli uverejnené v rovnakom čase.

5) Ak je jedna časť vzorca neopakovateľná, potom celý vzorec musí byť tiež neopakovateľný:

$$\frac{P \text{ verí neopakovateľné}(X)}{P \text{ verí neopakovateľné}(X,Y)}.$$

2.2.3 Kvantifikátory v delegáciach

Delegačné vyhlásenia typicky zmieňujú jednu alebo viac premenných. Splnomocniteľ A môže nechať server S generovať ľubovoľný zdieľaný kľúč pre A a B. To sa dá vyjadriť ako

$$A \text{ verí } S \text{ kontroluje } A \overset{\text{K}}{\leftrightarrow} B$$

Tu je kľúč K univerzálne kvantifikovaný a túto kvantifikáciu je možné explicitne napísať ako

$$A \text{ verí } \forall K(S \text{ kontroluje } A \overset{\text{K}}{\leftrightarrow} B)$$

Pre zložitejšie delegačné vyhlásenia je zvyčajne potrebné písať kvantifikátory explicitne, aby sa predišlo nejasnostiam. Napríklad je možné overiť, že dva vzorce znamenajú rôzne významy.

$$A \text{ verí } \forall K(S \text{ kontroluje } B \text{ kontroluje } A \overset{\text{K}}{\leftrightarrow} B)$$
$$A \text{ verí } S \text{ kontroluje } \forall K(B \text{ kontroluje } A \overset{\text{K}}{\leftrightarrow} B)$$

Použitá je schopnosť vytvoriť inštanciu premenných v príslušných príkazoch, čo sa prejavilo pravidlom

$$\frac{P \text{ verí } \forall V_1 \ldots \forall V_n.(Q \text{ kontroluje } X)}{P \text{ verí } Q' \text{ kontroluje } X'}$$

kde Q' kontroluje X' je výsledok konkretizácie premenných $V_1 \ldots V_n$ súčasne v Q kontroluje X. Formálna manipulácia kvantifikátorov je teda veľmi jednoduchá.

2.2.4 Idealizované protokoly

Autentizačné protokoly sú popísané uvedením ich správy, kde každá správa je zvyčajne písaná vo forme

$$P \to Q: \text{ správa.}$$

To znamená, že splnomocniteľ P odošle správu splnomocniteľovi Q. Správa sa predkladá v neformálnom zápise navrhnutom ako dátový reťazec. Táto prezentácia je často nejednoznačná a nie je vhodným základom pre formálnu analýzu. Preto sa každý krok protokolu transformuje do idealizovanej formy. Napríklad krok protokolu

$$A \to B: \ \{A, K_{ab}\}_{K_{bs}}$$

môže povedať B, ktoré pozná kľúč K_{bs}, že K_{ab} je kľúčom ku komunikácii s A. Tento krok by mal byť potom idealizovaný ako

$$A \to B: \ \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}.$$

Aj keď to nie je úplne triviálne, odvodenie praktického kódovania z idealizovaného protokolu je oveľa menej časovo náročné ako pochopenie významu určitého neformálneho kódovania. Avšak, na štúdovanie protokolov z existujúcej literatúry je potrebné najprv vytvoriť idealizované formy pre každý protokol.

2.2.5 Analýza protokolu

Pre analýzu idealizovaných protokolov sa píše vzorec pred prvou správou a po každej správe. Hlavné pravidlá pre odvodenie právnych poznámok sú:

- ak X má pred správou $P \to Q: \ Y$ potom obaja X a Q vidia Y dané následne
- ak Y môže byť odvodené z X pomocou logických postupov, potom Y platí vždy, keď platí X.

Vysvetlivka protokolu je akoby sled pripomienok v rámci viery splnomocniteľov, ktoré sú viditeľné v priebehu overovania. Krok za krokom, je možné sledovať vývoj od počiatočnej dôvery až po konečnú - od pôvodných predpokladov k záverom.

2.2.6 Ciele autentizácie, formalizované

Počiatočné predpoklady musia byť vždy vykonané pre zaručenie úspechu každého protokolu. Vo väčšine prípadov, predpoklady sú štandardné a zrejmé pre typ uvažovaného protokolu. Preto je možné považovať autentizáciu medzi A a B za dokončenú, ak existuje K také, že

$$A \text{ verí } A \overset{\mathrm{K}}{\longleftrightarrow} B, \qquad B \text{ verí } A \overset{\mathrm{K}}{\longleftrightarrow} B$$

Niektoré protokoly dosiahnu viac než to, ako je uvedené v nasledujúcom príklade

$$A \text{ verí } B \text{ verí } A \overset{\mathrm{K}}{\leftrightarrow} B, \qquad B \text{ verí } A \text{ verí } A \overset{\mathrm{K}}{\leftrightarrow} B$$

Iné protokoly dosiahnu iba slabšie konečné stavy. Napríklad také, kde A verí B a B verí X, pre niektoré X, ktoré odrážajú iba dôveru A v B, že nedávno odoslalo správy. Niektoré protokoly s verejným kľúčom nemajú viesť k výmene zdieľaného kľúča, ale namiesto toho prenášať inú časť dát v prípade, že požadované ciele sú všeobecne zrejmé z kontextu.

V tejto podkapitole bol popísaný mechanizmus BAN logiky z [9], kde je možné nájsť aj bližšie informácie.

3 NÁSTROJE K VYHODNOTENIU BEZPEČNOSTI

Návrh kryptografických protokolov je veľmi náročný problém. Závažné útoky môžu byť vykonávané nielen útokom na kryptografiu a jej prelomením, ale aj napadnutím samotnej komunikácie. Tieto útoky využívajú nedostatky v dizajne protokolu, podľa ktorých môžu byť protokoly prelomené chytrou manipuláciou a opätovným preposlaním správy v rozpore s ich predpokladaným dizajnom. Zahŕňa to útoky ako: *muž uprostred*, kde je útočník zapojený do dvoch paralelne vykonávajúcich sedení a prechádza správami medzi nimi; *opakovaný útok*, kde sa správy nahrané z predchádzajúcich sedení hrajú na tie budúce; *odrazové útoky*, kde odoslané informácie sú odoslané späť k autorovi; a *útoky na typy chýb(zmätenosť)*, kde sú správy rôznych typov nahradené do protokolu (napr. výmena mena za kľúč).

Počas posledných dvoch desaťročí urobila bezpečnostná komunita značné pokroky pri vývoji formálnych metód pre analýzu kryptografických protokolov, čím sa zabráni vyššie uvedeným útokom. Tieto metódy a nástroje môžu byť rozdelené podľa niekoľkých hľadísk [10]:

1. *Kontrola modelu verzus dokazovanie viet*: kontrola modelu zistí, že model M, typicky formalizovaný ako Kripkeho štruktúra, má vlastnosť ϕ. V dokazovaní viet sa v prvom rade znižuje overenie na dokázanie vety alebo logika vyššieho rádu. Model M formalizuje priamo sémantiku protokolu ako súbor stôp.
2. *Obmedzené verzus neobmedzené*: Protokoly môžu byť často napadnuté chytrou manipuláciou a prehrávaním správ. Tieto útoky môžu byť dosť zložité a môžu vyžadovať viac paralelne vykonávaných relácií. Preto pri automatickom overení môže byť buď obmedzený model, aby sa problém stal rozhodnuteľný, alebo neobmedzený model, aby sa dosiahol pokus o výrobu konečnej charakterizácie nekonečnej množiny dosiahnuteľných stavov (alebo stôp).
3. *Symbolické verzus kryptografické*: Štandardný model Dolev-Yao sa používa vo väčšine metód. Tento model poskytuje silnú idealizáciu aktuálnych kryptografických operácií. Táto idealizácia sa nazýva symbolická a zjednodušuje konštrukciu dôkazu. Na rozdiel od toho, u kryptografického postupu sú dôkazy konštruované redukciou tak, ako je to v teórii zložitosti.

Existuje mnoho automatických a poloautomatických nástrojov pre analýzu bezpečnostných protokolov. Napr. AVISPA, CryptoVerif, ProVerif, Casper/FDR, Scyther, Spi2Java a iné. V tejto kapitole popíšem tieto nástroje bližšie.

3.1 Casper/FDR

Tento nástroj sa skladá z nástroja Casper, ktorý prekladá popis protokolu do procesnej algebry CSP (Communication Sequential Processes), a z CSP modelového kontroléru FDR (Failures Divergences Refinement). Poskytuje časovo založenú analýzu pre analyzovanie veľkej zbierky existujúcich protokolov s použitím Dolev-Yao modelu. U CSP a FDR je metóda analýzy bezpečnostných protokolov nasledujúca: každý agent zúčastňujúci sa protokolu je modelovaný ako CSP proces; najvšeobecnejší útočník, ktorý môže pracovať s protokolom je tiež modelovaný ako CSP proces; výsledný systém je testovaný proti špecifikácii zastupujúcej požadované bezpečnostné vlastnosti, FDR potom prehľadáva stavový priestor, aby zistil, či môže dôjsť k nejakej nezabezpečenej stope (sekvencii správ); ak FDR zistí, že špecifikácia nie je splnená, potom vráti systémovú stopu, kde nespĺňa danú špecifikáciu; táto stopa zodpovedá útoku na protokol.

Avšak úloha produkovať CSP systémový popis je veľmi časovo náročná, a možno len pre ľudí s praxou v CSP, preto sa pre zjednodušenie tohto procesu používa Casper. Užívateľ špecifikuje protokol pomocou abstraktnejšej notácie, podobnej zápisu v akademickej literatúre, a Casper ju skompiluje do CSP kódu, vhodného na kontrolu pomocou FDR. V súčasnosti sa používa FDR2, ktorý je jeho druhou verziou.[11]

3.2 CryptoVerif

CryptoVerif je automatický overovateľ protokolov, ktorý môže ukladať výstupy dôkazov daných protokolov do príkazového riadka alebo do súborov (v bežnom alebo v Latex formáte). Jeho automatické dokazovanie má dva prístupy: nepriamy (používa Dolev-Yao model) a priamy (návrh automatického nástroja pre overenie protokolov vo výpočtovom modeli). Tento nástroj dokazuje vlastnosť utajenia a zhody, ďalej poskytuje všeobecnú metódu určenia vlastností kryptografických primitív, pracuje s neobmedzeným počtom relácií s aktívnym protivníkom a dáva medzu pravdepodobnosti útoku. V podstate sa opiera o kolekciu herných transformácií s cieľom transformovať pôvodný protokol do hry, v ktorej je zrejmá požadovaná vlastnosť bezpečnosti. Dôkazom je nasledujúca sekvencia hier: prvá hra je ideálna taká, kde jedna ide z jednej hry do ďalšej podľa syntaktických transformácií alebo použitím definície bezpečnosti kryptografických primitív a posledná hra je ideálne taká, kde ochrana objektu je zrejmá z herného tvaru. Hry sú formalizované výpočtovou metódou podobnou Pi. Tento výpočet je čisto pravdepodobnostný a beží v polynomiálnom čase.[12]

3.3 ProVerif

ProVerif je automatický overovateľ kryptografických protokolov vo formálnom modeli (Dolev-Yao model). Môže zvládnuť mnoho rôznych kryptografických primitív, vrátane kryptografie zdieľaného a verejného kľúča (šifrovanie a podpisovanie), hashovacej funkcie a Diffie-Hellmanovej dohode kľúčov, ktoré sú uvedené ako pravidlami prepisu, tak aj rovnicami. Analyzuje neohraničený počet spustení použitím cez aproximácie a reprezentujúcich protokolov podľa Horn ustanovení. Akceptuje dva druhy vstupných súborov: Horn ustanovenia a podmnožinu výpočtovej metódy Pi. ProVerif popisuje súbor procesov, kde každý definovaný proces môže byť spustený niekoľkokrát. Tento nástroj používa abstrakciu generácie neopakovateľných univerzálnych čísel, čo dovoľuje nekonečné možnosti pri overovaní rôznych typov protokolov. Overovateľ môže dokázať nasledujúce vlastnosti: utajenie (protivník nemôže získať tajomstvo), autentizáciu a všeobecnejšie korešpondenčné vlastnosti, silu tajomstva (útočník nevidí rozdiel, keď sa hodnota tajomstva zmení) a ekvivalenciu medzi procesmi, ktorá sa líši iba podmienkami. Podľa daného popisu protokolu sa môže stať jedna zo štyroch vecí: Po prvé, nástroj môže oznámiť, že vlastnosť je falošná a poskytne stopy útoku. Po druhé, vlastnosť môže byť preukázaná ako správna. Po tretie, nástroj zahlási, že vlasnosť nie je možné preukázať, napr. keď je nájdený falošný útok. Po štvrté, je možné, že nástroj nebude ukončený.[13]

3.4 Scyther

Scyther je automatický overovateľ protokolov, ktorý poskytuje grafické užívateľské rozhranie (GUI) doplnené príkazovým riadkom a skriptovacím rozhraním Python. GUI je zameraný na užívateľov, ktorý sa zaujímajú o overenie alebo pochopenie protokolu. Príkazový riadok a skriptovacie rozhrania uľahčujú používanie tohto nástroja na overenie veľmi rozsiahlych protokolových testov. Scyther kombinuje rad nových funkcií s najmodernejšou výkonnosťou. Po prvé, vymedzuje a zároveň umožňuje preukázanie správnosti protokolov pre neobmedzené množstvo relácií. Na rozdiel od iných neobmedzených verifikačných nástrojov poskytuje užitočné výsledky aj v prípade, že žiadny útok nie je nájdený. Po druhé, pomáha pri analýze protokolu tým, že poskytuje triedy správania protokolu (alebo triedy útokov), na rozdiel od jednej stopy útoku poskytovanej inými nástrojmi. Po tretie, Scyther uľahčuje tzv. viac-protokolová analýza. V takejto analýze sa analyzuje paralelné zloženie dvoch protokolov. Takáto analýza bola neuskutočniteľná pre protokolové nástroje kvôli explózii stavového priestoru. S výkonom poskytnutým nástrojom Scyther sa analýza viacerých protokolov stala uskutočniteľnou.

Pri danom popise protokolu v *spdl* jazyku možno Scyther použiť tromi rôznymi spôsobmi: s cieľom overiť, či sú nároky zabezpečenia v popise protokolu držané alebo nie; s cieľom automaticky generovať vhodné bezpečnostné tvrdenia pre protokol a overiť ich; s cieľom analyzovať protokol prevedením úplnej charakterizácie.[14]

3.5 Spi2Java

Spi2Java je vývoj rámca riadený modelom pre vykonávanie bezpečnostných protokolov v jave. V týchto rámcoch je paradigma v kombinácii s formálnymi metódami za účelom vytvorenia protokolu s vysokou dôverou bezpečnosti. Tento rámec je založený na symbolických formálnych modeloch v štýle modelu Dolev-Yao. Analýzou týchto modelov je možné zistiť mnoho druhov logických chýb a môžu overiť splnenie zamýšľaných bezpečnostných vlastností. Po tom, čo bola dosiahnutá správnosť modelu, môžu byť modely poloautomaticky zdokonalené do Java interoperabilných implementácií so zárukou, že niektoré bezpečnostné vlastnosti Dolev-Yao sú zachované v konečnej realizácii.

Pôvodný rámec bol založený na špecifikácii textového jazyka spi-kalkulus. Neskôr bol do Spi2Java pridaný grafický zápis špecifikácie pre modely protokolu a grafické užívateľské rozhranie na báze Eclipse (Spi2JavaGUI) pre vizuálne modelovanie a bezproblémový vývoj. Ďalšie úsilie v smere zjednodušenia používania Spi2Java pre Java skúsených vývojárov bolo vytvorenie JavaSPI, čo je rámec podobný Spi2Java, kde sú spi-kalkulus modely vyjadrené pomocou programovacieho jazyka Java.[15]

3.6 Tamarin Prover

Tamarin Prover je automatický nástroj pre overenie bezpečnostných protokolov. Realizuje obmedzený riešiaci algoritmus, ktorý podporuje falzifikáciu aj overenie bezpečnostných protokolov vzhľadom na neobmedzený počet relácií. Model základného bezpečnostného protokolu používa množinový prepis na špecifikáciu protokolu a schopnosti protivníka, strážený fragment logiky prvého rádu na špecifikáciu bezpečnostných vlastností a porovnávacie teórie na modelovanie algebraických vlastností kryptografických operátorov. Je modelovaný ako označený prechodový systém, ktorého stav sa skladá zo znalosti protivníka, správ na sieti, informácií o čerstvo vytvorených hodnotách a stavu protokolu. Protivník a protokol na seba vzájomne pôsobia aktualizovanými sieťovými správami a neopakovateľnosťou informácie. Schopnosti protivníka a protokolu sú špecifikované spoločne ako súbor (označených) množinových prepisovacích pravidiel. Bezpečnostné vlastnosti sú modelované ako trasovacie vlastnosti prenosového systému. Tamarin Prover podporuje režim dávkovej

analýzy a interaktívnu konštrukciu bezpečnostných dôkazov pomocou GUI. Používa Maude systém (http://maude.cs.uiuc.edu/) ako zjednocujúci backend a GraphViz softvér (http://www.graphviz.org/) na vizualizáciu obmedzených systémov.[16]

3.7 AVISPA

AVISPA (Automated Validation of Internet Security Protocols and Applications) je automatizovaný overovač internetových bezpečnostných protokolov a aplikácií. Je to vlastne push-button nástroj pre automatickú validáciu internetových protokolov a aplikácií citlivých na bezpečnosť. Poskytuje modulárny a expresívny formálny jazyk pre špecifikáciu protokolov a ich bezpečnostných vlastností, a integruje rôzne konce, ktoré implementujú rôzne najmodernejšie techniky automatizovanej analýzy.

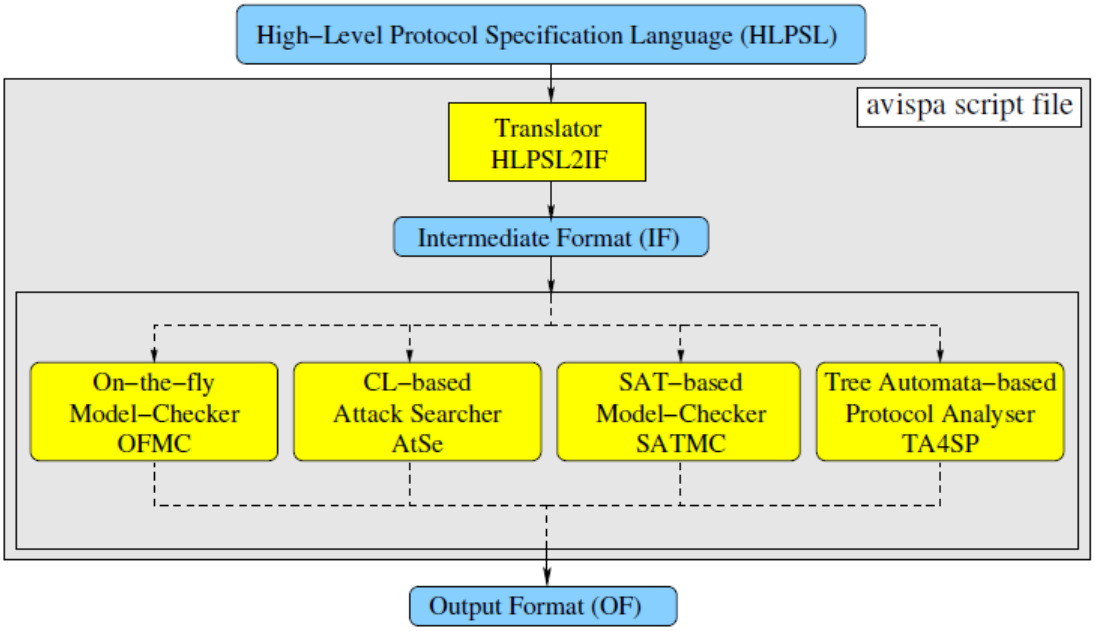
Protokoly skúmané pomocou AVISPA musia byť špecifikované v HLPSL (High Level Protocols Specification Language) a zapísané do súboru s príponou *hlpsl*. Tento jazyk je založený na úlohách: *základné úlohy* pre zástupcov z každej špecifickej úlohy a *zložené úlohy* pre zastupovanie scenárov základných úloh. Každá úloha je nezávislá od ostatných, získava nejaké počiatočné informácie o parametroch (napr. agentov, šifrovacie kľúče, atď.) a komunikuje s ostatnými úlohami po kanáloch. Pre zjednodušenie HLPSL špecifikácií bol navrhnutý SPAN (Security Protocol ANimator). Zo špecifikácie HLPSL pomáha SPAN interaktívne budovať MSC (Message Sequence Charts) o vykonávaní protokolu. Vzhľadom k tomu, že SPAN implementuje aktívneho votrelca, môže byť tiež použitý pre interaktívne nájdenie a budovanie útokov cez protokoly.

AVISPA (obr. 3.1) využíva Dolev-Yao model a skladá sa zo štyroch nasledujúcich nástrojov: CL-AtSe, OFMC, SAT-MC a TA4SP, ktoré sú popísané nižšie.[17]

3.7.1 CL-AtSe

CL-AtSe (Constraint Logic based Attack Searcher) poskytuje preklad z akejkoľvek špecifikácie bezpečnostného protokolu (písanej ako prenosový stav v IF (Intermediate Format)) do súboru obmedzení, ktoré môžu byť účinne použité k nájdeniu útokov na protokoly.

Každý krok protokolu je modelovaný obmedzením protivníkových znalostí. K obmedzeniam patria napr. podmienky ako je rovnosť, nerovnosť a (ne) členstvo prvku v zozname. Pre interpretovanie prechodového stavu IF je každá úloha čiastočne vykonaná tak, aby vypísala presný a zároveň minimálny zoznam modelovaných obmedzení. Stavy a znalosti účastníkov sú eliminované použitím globálnych premenných. Akýkoľvek krok protokolu sa uskutočňuje pridaním nových obmedzení do systému



Obr. 3.1: Architektúra nástroja AVISPA[17].

a znížením alebo vylúčením iných obmedzení. Nakoniec je stav systému v každom kroku testovaný proti poskytnutiu sady bezpečnostných vlastností.

Algoritmus pre analýzu používaný v CL-AtSe je určený len pre obmedzené množstvo cyklov, teda ohraničené množstvo krokov v akejkoľvek stope. To znamená, že ak špecifikácia protokolu je bez cyklu, potom sa analyzuje celá špecifikácia, inak musí užívateľ poskytnúť maximálny počet iterácií cyklu.[17]

3.7.2 OFMC

OFMC (On the Fly Model Checker) vytvára nekonečný strom definovaný problémom analýzy protokolu, teda za chodu (on-the-fly), preto názov spätne končiaci. Využíva rad symbolických techník reprezentujúcich stavový priestor. OFMC sa môže použiť nielen pre efektívne falšovanie protokolov (tj. rýchla detekcia útokov), ale aj pre overovanie (tj. dokazovanie správnosti protokolu) obmedzené počtom sedení - bez ohraničujúcich správ, ktoré môže útočník generovať. Najvýznamnejšou novinkou OFMC v tejto verzii je, že užívateľ môže špecifikovať algebraickú teóriu na podmienkach správy a analýzy protokolu, ktoré sa vykonávajú.[17]

3.7.3 SAT-MC

SAT-MC (boolean SATisfiability based Model Checker) vytvára výrokovú formulu kódujúcu obmedzené odvíjanie prechodového vzťahu určené pomocou IF, počia-

točný stav a množinu stavov predstavujúcich narušenie bezpečnostných vlastností. (Kompilácie SAT z IF špecifikácií vyplývajú z kombinácie zníženia bezpečnostných problémov a kódovacích techník SAT pre plánovanie.) Výroková formula sa potom privádza do najmodernejšieho SAT riešiteľa a každý nájdený model je preložený späť do útoku.

SAT-MC môže byť použité nielen k objavovaniu útokov na protokoly, ale tiež pre overenie (tj. preukazujúce, že protokol spĺňa jeho požiadavky na zabezpečenie) uzavretého počtu relácií, čo je problém, ktorý bol preukázaný, že patrí do rovnakej zložitosti ako SAT, tj. NP (Nondeterministic Polynomial) - úplné.[17]

3.7.4 TA4SP

Nástroj TA4SP (Tree Automata tool based on Automatic Approximations for the Analysis of Security Protocols) počíta vzhľadom k počiatočnému stavu buď cez aproximácie (over-approximation), alebo v rámci aproximácií (under-approximation) znalostí útočníka. Robí to pomocou prepisovania na stromové jazyky v súvislosti s neobmedzeným počtom sedení. TA4SP používa knižnicu stromových automatov Timbuk 2.0 (vyvinutá Th. Genet IRISA, v Rennes vo Francúzsku a je dostupná na http://www.irisa.fr/celtique/genet/timbuk/) na vykonanie výpočtu útočníkových znalostí.

Tento nástroj môže v rámci aproximácie kontextu ukázať, bez akýchkoľvek voliteľných abstrakcií, že protokol je chybný kvôli danej bezpečnostnej vlastnosti. Vzhľadom k tomu je empirická stratégia pre overenie protokolu s TA4SP nasledovná:

1. užívateľ počíta cez aproximáciu a kontroluje bezpečnostné vlastnosti
2. v prípade, že prvý krok neumožňuje zabezpečiť utajenie, potom užívateľ postupne počíta v rámci aproximácie, kým nezíska útok v primeranom čase.

Avšak táto empirická stratégia nemusí vždy viesť k očakávanému výsledku. V skutočnosti nepresvedčivý výsledok (použitím cez aproximáciu) neznamená, že existuje skutočný útok. TA4SP nespracováva súbory a podmienky, iba overí bezpečnostné vlastnosti so zadaným modelom.[17]

3.8 Porovnanie nástrojov pre formálnu analýzu

Všetky vyššie zmienené nástroje majú spoločné to, že využívajú Dolev-Yao model pri vyhodnotení bezpečnosti. Všetky okrem Cl-AtSe, OFMC a SAT-MC pracujú s neobmedzeným počtom relácií. Až na nástroj Spi2Java sú všetky ostatné nástroje automatické. Je možné ich spustiť vo Windowse, v Linuxe aj v Mac OS. Výstupy väčšiny z nich sú možné do príkazového riadku alebo GUI. V tabuľke 3.1 je možné

nájsť porovnanie základných vlastností všetkých nástrojov. Namiesto jediného nástroja AVISPA tam sú rozpísané jednotlivé zložky. K jednotlivým nástrojom sú vypísané jazyky a špecifikácie, ktoré používajú, možné výstupy, spôsoby transformácie jednotlivých špecifikácií protokolov a použitá analýza v rámci nástroja. Každý z týchto nástrojov má svoje silné a slabé stránky, ako je možné vidieť v tabuľke 3.2. Samozrejme výhod a nevýhod je v rámci nástrojov viac, vypísal som len tie najdôležitejšie. Opäť sú tu v rámci AVISPA popísané jednotlivé zložky. V prípade AVISPA ako celku je výhodou jednoduché modelovanie protokolu v HLPSL jazyku a k nevýhodám patrí to, že linky medzi agentmi a kľúčmi sú ťažko kódovateľné.

Tab. 3.1: Základné porovnanie nástrojov.

Nástroj	**Špecifikácia**	**Výstup**	**Transformácia**	**Analýza**
Cl-AtSe	HLPSL/IF	SPAN/ prík. riadok	úlohy	obmedzenia znalostí
OFMC	HLPSL	SPAN/ prík. riadok	úlohy	za chodu
SAT-MC	HLPSL/IF/ SAT	SPAN/ prík. riadok	úlohy	útoku a overenia
TA4SP	HLPSL	SPAN/ prík. riadok	úlohy	cez a v rámci aproximácií
Casper/FDR	CSP/ Casper	FDR	procesy	časová
CryptoVerif	Pi metóda	prík. riadok	hra	nepriama a priama
ProVerif	Horn/Pi	prík. riadok	procesy	cez aproximácie
Scyther	SPDL	GUI/Python/ prík. riadok	triedy správania protokolu	viac-protokolová
Spi2Java	Java/ Spi kalkulus	Spi2JavaGUI/ JavaSPI	formálne modely	modelov
Tamarin Prover	Maude	GUI/ GraphViz	množinová dosiahnuteľnosť	spätnej

V ďalšom texte mojej práce sa zameriam na nástroje AVISPA, ProVerif a Scyther, ktoré použijem pre neskoršiu analýzu vybraných protokolov. AVISPA ma oslovila hlavne kvôli jej širokej možnosti použitia a možnosti vyhodnotiť analýzu pomocou jej štyroch častí, vďaka ktorým bude výsledná analýza vykonaná dôkladnejšie. Dôležitá pri rozhodovaní bola aj jednoduchosť modelovania protokolov. Využijem hlavne nástroj SPAN, vďaka ktorému je možné vykresliť komunikáciu medzi agentmi a vyobraziť možný útok na protokoly.

ProVerif ma zaujal hlavne kvôli jeho rýchlosti nájdenia útoku a jednoduchosti modelovania do procesov. V prípade Scytheru je zaujímavé, že nevyužíva aproximácie a dokáže podrobne a pekne graficky vyobraziť útok.

Tab. 3.2: Výhody a nevýhody nástrojov.

Nástroj	**Výhody**	**Nevýhody**
Cl-AtSe	zjednodušuje pôvodné protokoly	pomalší pre komplexnejšie protokoly
OFMC	najefektívnejší v AVISPA	nie je možné overiť skupinové bezpečnostné protokoly
SAT-MC	vužíva SAT riešiteľa	exponenciálne správanie
TA4SP	využíva bežné stromové jazyky	nie je schopný preukázať stopy útoku
Casper/FDR	popis v jednoduchom abstraktnom jazyku	exponenciálne správanie, pomalá analýza
CryptoVerif	vyhodnocuje pravdepodobnosť úspechu útoku	v niektorých prípadoch vyžaduje manuálne vedenie
ProVerif	rýchlosť nájdenia útoku	veľmi komplexný, dlhé modelovanie
Scyther	nevyužíva aproximácie	pri niektorých protokoloch nie je možné vykonať úplnú kontrolu
Spi2Java	výslovne určuje vykonané kontroly	poloautomatický nástroj
Tamarin Prover	obmedzuje množinu stôp pomocou axiómov	ťažké na zápis, vyžadované pomocné lemmy

4 TESTOVANÉ PROTOKOLY

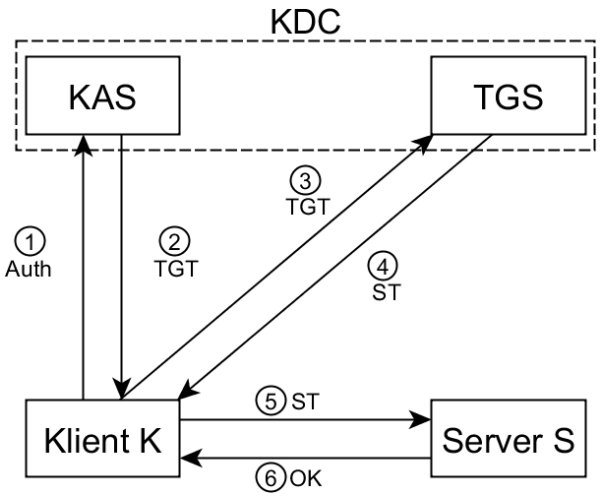
V rámci mojej diplomovej práce sa budem ďalej zaoberať testovaním vybraných protokolov. Na začiatok som si vybral protokol Kerberos, ktorý, kvôli jeho rozsiahlosti, je ideálnym protokolom na preskúšanie v rôznych nástrojoch. Ďalej sa budem zaoberať ešte nikde neoverenými protokolmi jednosmernej autentizácie, ktoré môžu byť implementované na hardwarovo a výpočtovo obmedzených zariadeniach. V tejto kapitole popíšem ich základné princípy a ich testovaním sa budem zaoberať v ďalšej kapitole.

4.1 Kerberos

Protokol Kerberos [18] bol navrhnutý tak, aby sa oprávnený užívateľ prihlásil na terminál raz denne (typicky) a potom transparentne pristupoval ku všetkým sieťovým zdrojom počas zvyšku dňa. Napríklad zakaždým, keď užívateľ chce načítať súbor zo vzdialeného servera, Kerberos sa postará o požadovanú autentizáciu za scénou bez zásahu užívateľa. Aktuálna verzia tohto protokolu je Kerberos 5 [19].

4.1.1 Princípy

Kerberos je realizovaný ako množstvo softvérových agentov alebo princípov, kde každý z nich manipuluje odlišný aspekt overenia, napríklad, keď užívateľ na svojom termináli požaduje sieťové služby také ako vzdialená tlačiareň. Najskôr klientsky proces akceptuje heslo používateľa a transparentne spracováva overovací aspekt každého zo svojich požiadaviek na účet. Potom je služba sprostredkovaná pomocou procesu servera. Kerberos sa opiera o ďalších dvoch administratívnych agentov spolu známych ako KDC (Key Distribution Center): KAS (Kerberos Authentication Server), ktorý autentizuje užívateľa a zaisťuje zodpovedajúcemu klientovi používať sieť na deň, a TGS (Ticket Granting Server), ktorý overuje klienta do každého požadovaného servera na základe týchto poverení. Princíp je zobrazený na obrázku 4.1.
Po tom, čo sa užívateľ prihlási, KAS autentizuje reprezentujúci klientsky proces a poskytne poverenie používať systém v ten deň. Toto poverenie od KAS je TGT (Ticket Granting Ticket). Vždy, keď chce užívateľ používať sieťové služby, klient v jeho mene sa bude usilovať o autentizáciu do serveru S. Toto sa uskutoční v dvoch krokoch: v prvom sa užívateľ pokúsi o prístup na S, klient K prepošle TGT získaný z KAS na TGS, ktorý na oplátku poskytne poverenie pre S. Toto poverenie je ST (Service Ticket) a prepošle sa na server S, vďaka čomu sa povolí prístup k službe. Pri každom ďalšom prístupe k tejto konkrétnej službe klient odošle ST na server S bez účasti TGS.



Obr. 4.1: Princíp Kerberos autentizácie.

4.1.2 Základné overenie výmeny

Správy vymieňané počas typickej autentizácie vo vnútri oblasti (intra-realm) relácie medzi klientom K a serverom S sú:

1. $K \rightarrow KAS \ : \ K, TGS, t0, n_1$
2. $KAS \rightarrow K \ : \ K, \{AK, K, TGS, t_{KAS}\}_{k_{TGS}}, \{AK, n_1, t_{KAS}, TGS\}_{k_K}$
3. $K \rightarrow TGS \ : \ K, \{AK, K, t_{KAS}\}_{k_{TGS}}, \{K, t_K\}_{AK}, S, n_2$
4. $TGS \rightarrow K \ : \ K, \{SK, K, t_{TGS}\}_{k_S}, \{SK, n_2, t_{TGS}, S\}_{AK}$
5. $K \rightarrow S \ : \ \{SK, K, t_{TGS}\}_{k_S}, \{K, t'_K\}_{SK}$
6. $S \rightarrow K \ : \ \{t'_K\}_{SK}$

Ďalej bude popísaná každá z troch spätných ciest medzi klientom K a KAS, TGS a serverom S. [20]

Výmena autentizačných služieb($K \Leftrightarrow KAS$)

Táto výmena sa vykonáva hneď, keď sa používateľ prvýkrát prihlási ku kerberizovanej sieti. Klientsky proces K generuje unikátne číslo n_1 a odošle ho na KAS spolu so svojim vlastným menom K, čo nepriamo identifikuje užívateľa, a názov TGS. Po rozpoznaní K, KAS odpovie správou obsahujúcou dve šifrované zložky: TGT - $\{AK, K, TGS, t_{KAS}\}_{k_{TGS}}$, ktoré sa uloží do medzipamäte K a bude použité na získanie tiketov pre služby na zvyšok dňa, a $\{AK, n_1, t_{KAS}, TGS\}_{k_K}$ s ktorým KAS informuje K o parametroch tiketu. TGT je určený pre TGS a je šifrované s dlhodobým kľúčom k_{TGS}, ktoré KAS zdieľa s TGS. Okrem mena K klienta a mena serveru TGS obsahuje čerstvo generovaný autentizačný kľúč AK, ktorý K zdieľa s TGS a časovú značku t_{KAS}. Dlhodobo tajný kľúč k_K medzi K a KAS je použitý

na zašifrovanie druhej zložky (KAS ho odvodí z hesla užívateľa). Časová značka t_{KAS} ubezpečí TGS a K, že tento tiket bol nedávno vydaný, pretože všetky princípy Kerberosu majú voľne synchronizovaný čas. Unikátne číslo n_1 v druhej zložke sa viaže k odpovedi na pôvodnú žiadosť K.

Výmena udelenia tiketu($K \Leftrightarrow TGS$)

Táto výmena sa vykoná prvýkrát, keď sa užívateľ pokúsi o prístup k službe na serveri S. V odchádzajúcej správe, K prenáša z medzipamäte uložené TGT a meno serveru S spolu s čerstvo generovaným unikátnym číslom n_2 (opäť naviazanie tejto žiadosti a následnej odpovede), a overenie $\{K, t_K\}_{AK}$, kde t_K je časová značka. Overenie dokáže TGS, že K skutočne pozná autentizačný kľúč AK.

Po autentizácii K a overení, že mu je dovolené používať S, TGS posiela reakciu s rovnakou štruktúrou ako druhá správa vyššie s výnimkou, že tiket služby ST - $\{SK, K, t_{TGS}\}_{k_S}$ je šifrované pomocou dlhodobého kľúča k_S zdieľaného medzi TGS a S, a obsahuje čerstvo generovaný kľúč služby SK, zdieľaný medzi K a S, meno K a časovú značku t_{TGS}. Druhá zložka je zašifrovaná pomocou kľúča AK. K uloží ST do vyrovnávacej pamäte.

Klient/server výmena ($K \Leftrightarrow S$)

Táto výmena prebieha zakaždým, keď klient K začne novú reláciu so serverom S. K s tiketom služby ST kontaktuje S. Odpoveď zo servera S je nepovinná, môže byť zahrnutá v rámci následnej aplikačnej výmeny. Ak je prítomná, poskytuje záruku klientovi K, že S je živý, napríklad vrátením časovej značky t'_K, ktorá je zašifrovaná pomocou kľúča služby SK v obsiahnutej žiadosti. [20]

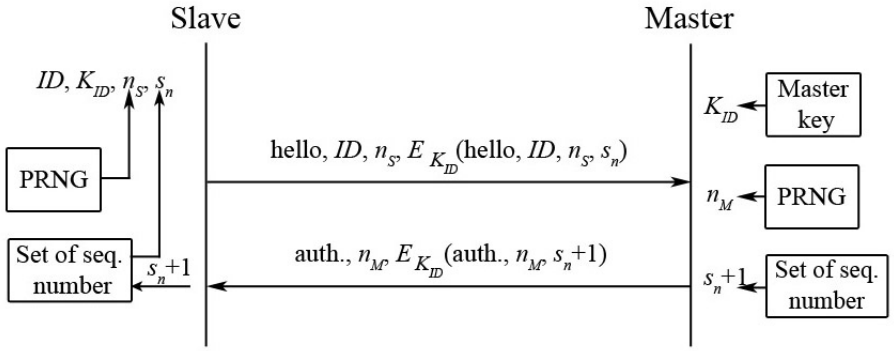
4.2 Protokoly jednosmernej autentizácie

Člupek a Zeman v článku [21], ktorý bude prezentovaný na konferencii TSP 2015 (http://tsp.vutbr.cz/), navrhli riešenie jednosmernej autentizácie na nízko nákladových zariadeniach, kde je prvý účastník autentizovaný druhým účastníkom alebo druhý účastník autentizovaný prvým účastníkom. Tieto nízko nákladové zariadenia sú výpočtovo a zdrojovo obmedzené zariadenia. Patria k nim napríklad mikrokontroléry (napr. MSP430F2254) a smart karty (napr. ML5-80K-65). Z dôvodu ich obmedzení je možné použiť len ľahkú alebo ultra-ľahkú kryptografiu. Ľahká kryptografia zahŕňa symetrickú kryptografiu, HMAC funkciu a hashovaciu funkciu. Typické je použitie pseudonáhodného číslicového generátoru (PRNG). Ultra-ľahká kryptografia zahŕňa jednoduché logické bitové operácie ako je OR, XOR, AND, shift, atď. V ďalšom texte budú popísané ich návrhy, kde hrajú hlavnú úlohu Slave a Master.

4.2.1 Jednosmerná autentizácia použitím symetrickej šifry

Vďaka tomu, že niektoré symetrické šifry majú malú spotrebu zdrojov, je možné ich implementovať na nízko nákladové zariadenia. Na tento účel je možné použiť napríklad XTEA (eXtended Tiny Encryption Algorithm) alebo odľahčený variant AES s dĺžkou kľúča 128 bitov.

Na obr. 4.2 je ukázaný proces exekúcie jednosmernej autentizácie použitím symetrickej šifry.



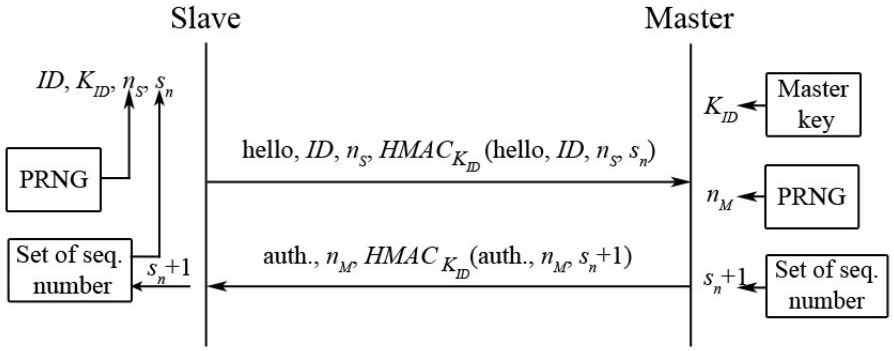
Obr. 4.2: Jednosmerná autentizácia použitím symetrickej šifry [21].

Jeho princíp je nasledujúci: Na začiatku vlastní Slave identitu ID, zdieľaný kľúč K_{ID} a tajné sekvenčné číslo s_n, ktoré sú získané od Mastra zabezpečeným kanálom. Master si kľúč K_{ID} užívateľa Slave generuje po prijatí jeho ID. Proces autentizácie je otvorený Slaveom. Vygeneruje unikátne číslo n_S a odošle sekvenciu $hello, ID, n_S, E_{K_{ID}}(hello, ID, n_S, s_n)$ Mastrovi. Master po dešifrovaní porovná $hello, ID$ a n_S s prijatými nešifrovanými hodnotami. Ak sa rovnajú, je garantovaná ich autentičnosť. Potom porovná aj prijaté dešifrované sekvenčné číslo s_n so svojím číslom s_n, ktoré má uložené vo svojej databáze k odpovedajúcemu ID. Ak sa rovnajú, navýši sekvenčné číslo o 1 a vygeneruje unikátne číslo n_M. Tie potom odošle Slaveovi v sekvencii $auth., n_M, E_{K_{ID}}(auth., n_M, s_{n+1})$. Slave navýši s_n o 1 a po dešifrovaní porovná všetky hodnoty. Ak sa rovnajú, potom je garantovaná ich autentičnosť. Pre šifrovanú komunikáciu môžu Slave a Master vypočítať zdieľaný šifrovací kľúč $K_{session} = E_{K_{ID}}(K_{ID}, n_M)$, ktorý môže byť zredukovaný na požadovanú dĺžku.

4.2.2 Jednosmerná autentizácia použitím funkcie HMAC

HMAC (keyed-Hash Message Authentication Code) funkcia je typom MAC (Message Authentication Code) funkcie počítanej kryptografickú hash funkciu v kombinácii s tajným šifrovacím kľúčom. Na tento účel je možné použiť napr. SHA2 (Secure Hash Algorithm) alebo SHA3 hashovacie funkcie, presnejšie ich odľahčené verzie.

Obrázok 4.3 ukazuje proces exekúcie jednosmernej autentizácie použitím HMAC funkcie.



Obr. 4.3: Jednosmerná autentizácia použitím funkcie HMAC [21].

Princíp autentizácie je podobný vyššie popísanému protokolu s použitím symetrickej šifry. Symetrická šifrovacia funkcia je ale nahradená HMAC funkciou. Všeobecná HMAC funkcia má tvar:

$$HMAC_K(m) = h\Big((K \oplus opad) \parallel h\Big((K \oplus ipad) \parallel m\Big)\Big),$$

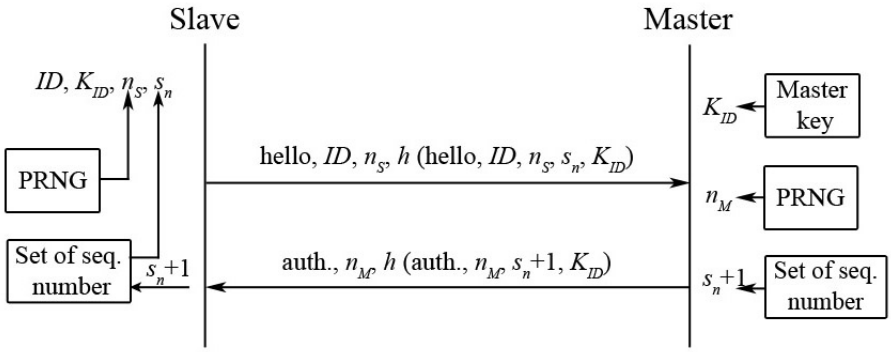
kde h je hashovacia funkcia, K je tajný kľúč zarovnaný nulami na veľkosť základného bloku, m je správa, $\parallel$ označuje zreťazenie, $\oplus$ označuje exkluzívnu disjunkciu (XOR), $opad = 0x5c5c5c \ldots 5c5c$ vonkajšie zarovnanie a $ipad = 0x363636 \ldots 3636$ vnútorné zarovnanie. Opad a ipad sú dve dlhé konštanty o veľkosti základného bloku.

Proces autentizácie je opäť otvorený Slaveom. Opäť si vygeneruje n_S a odošle sekvenciu $hello, ID, n_S, HMAC_{K_{ID}}(hello, ID, n_S, s_n)$. Po prijatí, Master si vypočíta $HMAC_{K_{ID}}(hello, ID, n_S, s_n)$ a porovná ho s prijatým $HMAC$. Ak sa rovnajú, tak je garantovaná autentičnosť prijatých dát. Master navýši s_n o 1 a generuje n_M. Tieto hodnoty pošle Slaveovi v sekvencii $auth., n_M, HMAC_{K_{ID}}(auth., n_M, s_{n+1})$. Slave navýši hodnotu s_n o 1 a vypočíta vlastný $HMAC$, ktorý porovná s prijatým. Ak sa rovnajú tak je zaručená autentičnosť prijatých dát. Pre ďalšiu komunikáciu môžu Slave a Master vypočítať $K_{session} = HMAC_{K_{ID}}(K_{ID}, n_M)$, pričom dĺžka kľúča je daná výstupnou veľkosťou otlačku hashovacej funkcie.

4.2.3 Jednosmerná autentizácia použitím hashovacej funkcie

U HMAC funkcie je hashovacia funkcia pri každom výpočte použitá dvakrát. To znamená, že Master a Slave počítajú hashovaciu funkciu štyrikrát na každej strane. V tejto časti sa použije výpočet dvoch hashovacích funkcií na každej strane.

Obrázok 4.4 ukazuje proces exekúcie jednosmernej autentizácie použitím hashovacej funkcie.



Obr. 4.4: Jednosmerná autentizácia použitím hashovacej funkcie [21].

Princíp je opäť podobný vyššie popísaným protokolom. Slave získa od Mastera zabezpečeným kanálom ID, K_{ID} a s_n. Vygeneruje unikátne číslo n_S a odošle sekvenciu $hello, ID, n_S, h(hello, ID, n_S, s_n, K_{ID})$ Mastrovi. Master po prijatí vypočíta vlastný hash a porovná ho s prijatým. Ak sa rovnajú, tak je zaručená autentičnosť prijatých dát. Master navýši s_n o 1 a vygeneruje unikátne číslo n_M. Tieto hodnoty potom odošle Slaveovi v sekvencii $auth., n_M, h(auth., n_M, s_{n+1}, K_{ID})$. Slave navýši svoje s_n o 1 a vypočíta vlastný hash a porovná ho s prijatým. Ak sa rovnajú, tak je zaručená autentičnosť prijatých dát. Pre šifrovanú komunikáciu môžu Slave a Master vypočítať zdieľaný šifrovaný kľúč $K_{session} = h(K_{ID}, n_M)$. Dĺžka kľúča je daná veľkosťou výstupného otlačku použitej hashovacej funkcie.

4.3 Encrypted key exchange

Protokol Encrypted key exchange (EKE) umožňuje dvom stranám zdieľať spoločné heslo na výmenu dôverných a overených informácií cez nezabezpečenú sieť pomocou asymetrickej a symetrickej kryptografie. EKE môže byť použité s rôznymi asymetrickými kryptosystémami a distribuovanými systémami s použitím verejného kľúča. Pri popise tohto protokolu som vychádzal z [22].

4.3.1 Základná výmena správ

Správy vymieňané medzi dvoma stranami (Alica a Bob) sú nasledujúce:

1. $Alica \rightarrow Bob : \{EA\}_P$
2. $Bob \rightarrow Alica : \{\{R\}_E A\}_P$
3. $Alica \rightarrow Bob : \{challengeA\}_R$

4. $Bob \rightarrow Alica : \{challengeA, challengeB\}_R$
5. $Alica \rightarrow Bob : \{challengeB\}_R$

kde najskôr Alica vygeneruje náhodný verejný kľúč EA a zašifruje ho zdieľaným heslom P a odošle ho na Boba. Túto správu Bob dešifruje a získa kľúč EA. Bob vygeneruje náhodný tajný kľúč R a zašifruje ho najskôr verejným kľúčom EA a potom to celé zdieľaným heslom P. Po prijatí Alica dešifruje správu a získa tajný kľúč R. Alica vygeneruje unikátnu výzvu $challengeA$ a zašifruje ju kľúčom R. Bob po dešifrovaní prijatej správy získa danú výzvu a vygeneruje vlastnú $challengeB$. Obe výzvy zašifruje kľúčom R a odošle Alici. Alica dešifruje správu a porovná prijatú výzvu so svojou vygenerovanou. Ak sa rovnajú, tak odošle dešifrovanú výzvu $challengeB$ zašifrovanú kľúčom R na Boba. Bob po dešifrovaní správy porovná prijatú výzvu s vygenerovanou a ak sa rovnajú, tak login je úspešný a pre ďalšiu komunikáciu sa použije symetrický kľúč R.

Tento protokol v jeho pôvodnej forme nie je zabezpečený. Je náchylný na man-in-the-middle (muž uprostred) útok. EKE protokol je zaradený do tejto diplomovej práce kvôli ukážke vyhodnotenia vo vybraných nástrojoch a vyobrazení tohto útoku týmito nástrojmi.

5 FORMÁLNA ANALÝZA PROTOKOLOV

K formálnej analýze kryptografických protokolov som si vybral nástroje AVISPA (presnejšie jeho grafické rozhranie SPAN), ProVerif a Scyther. Pomocou nich vykonám analýzu protokolu Kerberos, vyššie popísaných protokolov jednosmernej autentizácie a protokolu EKE. Popíšem spôsob práce s týmito nástrojmi a potom vykonám analýzu jednotlivých protokolov.

Analýzu som vykonal vo virtuálnom stroji nástroja HyperV od Microsoftu, kde bol nainštalovaný operačný systém Debian GNU/Linux 7.7.0 i386. V rámci tohto nástroja mu bolo priradených 512 MB RAM, ktorú využíval dynamicky až do veľkosti 2048 MB, ďalej 1 procesor a 10 GB harddisk.

5.1 AVISPA(SPAN)

Zo stránok www.avispa-project.org je možné stiahnuť grafické rozhranie SPAN. Ja som nainštaloval aktuálnu verziu 1.6. K samotnému chodu aplikácie je nutné doinštalovať knihovňu Tcl/Tk 8.5, kde Tcl je programovací jazyk a Tk je toolkit grafického užívateľského rozhrania. Inštalácia nie je nijak komplikovaná, je len potrebné rozbaliť stiahnutý súbor do požadovaného priečinku a pred spustením nastaviť dve premenné prostredia, ktorým nastavíme cestu k rozbalenému SPANu:

```
export SPAN=/usr/avispa/span
export AVISPA_PACKAGE=/usr/avispa/span
```

Spustenie nástroja je možné príkazom `./span`, kde môže byť pridaný rovno názov súboru, ktorý chceme otvoriť. Tieto súbory sú uložené s príponou `.hlpsl`.

Modelovanie v HLPSL

Pri popise práce v tomto nástroji som vychádzal z tutoriálu [23]. Základom modelovania v tomto nástroji je namodelovanie rolí (role), relácie (session), prostredia (environment) a cieľov(goals), ktoré chceme dosiahnuť. Definovanie rolí je v podstate všade rovnaké a obsahuje hlavičku, kde sa definujú agenti, kľúče, funkcie a prenosový kanál (dy = Dolev-Yao). Pomocou `played_by` sa určí agent, ktorý bude hrať danú úlohu. Určia sa jeho lokálne premenné písané veľkým začiatočným písmenom, definujú sa konštanty s malým začiatočným písmenom a určí sa inicializačný stav pomocou `init`. Nakoniec sa definuje samotný prenos (`transition`), kde sa namodelujú vymieňané odoslané a prijaté správy. V tomto prípade je dôležité, že každá hodnota označená „'“ značí novú hodnotu (T' = fresh T).

U úlohy relácie (session) sa všetky role zlepujú dokopy a prebiehajú paralelne. V definícii je potrebné zachovať poradie podľa definovania v jednotlivých hlavičkách

rolí. V úlohe prostredie (environment) sa určia znalosti útočníka, kde sa určí, ktoré z vlastnosti definovaných rolí pozná a dajú sa do relácie. Každá úloha sa ukončuje príkazom `end role`. V cieľoch (goals) sa určia ciele, ktoré chceme dosiahnuť a overiť (bezpečnosť kľúča, tajných hodnôt a pod.). Nakoniec sa modelovanie ukončí príkazom `environment()`, ktorým sa konkretizuje najvyššia úloha.

Nástroj OFMC nevyužíva ciele (goals) na vyhodnotenie bezpečnosti, namiesto toho využíva príkazy `witness` (svedok), `secret` (tajomstvo) a `request` (žiadosť). Príkazy `witness` a `request` sú použité na overenie, že sa principal správne domnieva o prítomnosti jeho určeného seberovného (peer) v aktuálnej relácii a že dosiahol určitého stavu a súhlasí s určitou hodnotou, ktorá je typicky fresh. Objavujú sa v pároch s identickým tretím parametrom => `/\ request(A,B,na,NA)` | `/\ witness(B,A,na,NA)`. Ako parametre vystupujú konštanty (napr. `sn`), ktoré sú typu `protocol_id`. Tento typ sa používa na vzájomné pridruženie `witness` a `(w)request`. Príkaz `request` je možné použiť s predponou `w-`, ktorý označuje slabú (weak) autentizáciu, kedy nie je použitá ochrana proti opakovanému útoku. Príkaz `secret` sa používa, ak chceme vyjadriť, že určitá hodnota (reprezentovaná termínom T) produkovaná alebo vybraná úlohou A je zdieľané tajomstvo medzi A a množinou agentov => `/\ secret(T,A)` | `/\ secret(T,B)`.

Vyhodnotenie bezpečnosti

Po vybraní nástroja, v ktorom chceme testovať, sa nám vyobrazí, či je daný protokol zabezpečený (safe) alebo nie (unsafe) a ukáže sa útok. Na konci sa zobrazia štatistiky jednotlivých testov, kde je možné vidieť aj čas vyhodnocovania (obr.5.1). Po vyhodnotení je možné vyobraziť grafickú simuláciu protokolu (obr.A.1), simuláciu útočníka a simuláciu útoku (v prípade, že je unsafe (obr.A.2), kde je možné sledovať presné parametre.

5.1.1 Kerberos

Pri modelovaní tohto protokolu som vychádzal z vyššie popísaného spôsobu výmeny správ počas typickej autentizácie vo vnútri oblasti (intra-realm) medzi klientom a serverom. Základom tohto modelovania je zadefinovanie úloh (role) KAS, TGS, klient a server. V hlavičke každého z týchto úloh sú agenti, ktorý majú s danou úlohou súvis (vymieňajú si s ňou správy). Ďalej tam je definovaný kanál pomocou `SND` a `RCV`, ktoré označujú kanál prijímania respektíve odosielania (zatiaľ je podporované len dy - Dolev-Yao) a nakoniec symetrické kľúče, ktoré používajú. V lokálnych premenných sa zadefinujú používané unikátne čísla (nonce), ďalšie použité kľúče a časové značky. Potom sa pomocou `init` nastaví `State` na požadovanú hodnotu. Samotný prenos (transition) je pre každú úlohu iný. Ako príklad je pre úlohu KAS:

```
1. State   = 0 /\ RCV(K.TGS.T0'.N1')  =|>
   State' := 1 /\ AK' := new()
               /\ T1start' := new()
               /\ T1exp' := new()
               /\ SND(K.{K.TGS.AK'.T1start'.T1exp'}_Ktgs.
                      {TGS.AK'.T1start'.T1exp'.N1'}_Kk)
               /\ witness(KAS,K,ak1,AK')
               /\ witness(KAS,TGS,ak2,AK')
               /\ secret(AK',sec_kas_AK,{KAS,K,TGS})
```

Kde agent najskôr príjme správu od klienta, ktorý sa identifikuje pomocou K, názov TGS od ktorého ma získať povolenie a nové hodnoty časovača T0 a unikátneho čísla N1. Znakom `=|>` sa prepne do ďalšieho stavu, kde najskôr vygeneruje nový kľúč AK a nastaví nové hodnoty časovača, ktoré určujú dĺžku platnosti kľúča. KAS späť odošle identifikáciu klienta, ktorý mu poslal požiadavok, TGT v tvare `{K.TGS.AK'.T1start.T1exp}_Ktgs` a informáciu o parametroch tiketu v tvare `{TGS .AK'.T1start'.T1exp'.N1'}_Kk`. Witness (svedok) dosvedčuje, že tajný kľúč AK je zabezpečený najskôr medzi KAS a K a potom aj medzi KAS a TGS, ktoré potvrdzujú konštanty `ak1` a `ak2`. Nakoniec sa overí bezpečnosť tajného kľúča AK zdieľaného medzi K, KAS a TGS príkazom `secret`, kde sa overuje pomocou konštanty `sec_kas_AK`.

V úlohe relácia (session) sa špecifikujú jednotlivé kanály a pomocou príkazu `composition` sa usporiadajú do paralelných relácií:

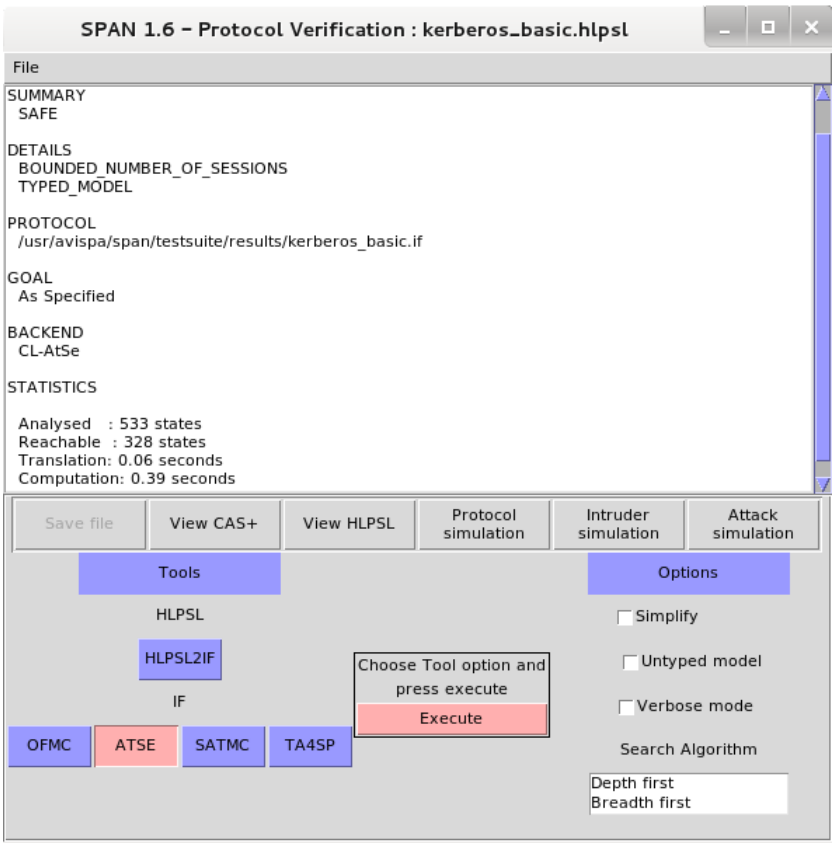
```
composition
              klient(K,KAS,TGS,S,SendK,ReceiveK,Kk)
         /\   kAS(KAS,K,TGS,SendKAS,ReceiveKAS,Kk,Ktgs)
         /\   tGS(TGS,KAS,S,K,SendTGS,ReceiveTGS,Ktgs,Ks)
         /\   server(S,TGS,K,SendS,ReceiveS,Ks)
```

V úlohe prostredie (environment) sa príkazom `intruder_knowledge` určia počiatočné znalosti útočníka, ktoré sú v tomto prípade znalosť všetkých agentov a jeho vlastný kľúč `kikas`, ktorý sa bude snažiť použiť na komunikáciu s KAS. Tieto jeho vlastnosti sa zakomponujú do relácie, kde sa najskôr určí normálna relácia bez útočníka a relácia s útočníkom, kde v úvodnej relácii v rámci klienta nahradí samotného klienta K a kľúč K_k za jeho kľúč *kikas*. Nakoniec sa určia ciele (goals) najskôr príkazom `secrecy_of` a jednotlivé strany, ktoré zdieľajú kľúče AK a SK, aby sa zistila bezpečnosť týchto kľúčov zo všetkých komunikujúcich strán. Potom príkazom `weak_authentication_on` a daný parameter, v tomto prípade parametre `ak1`, `ak2`, `sk1`, `sk2` a ďalšie, ktorým sa žiada overenie slabej autentizácie (nie je tam

žiadna ochrana pred replay útokom) týchto parametrov. Samozrejme, modelovanie sa ukončí príkazom `environment()`. Celý zdrojový kód je možné nájsť v priloženom CD.

Vyhodnotenie bezpečnosti

Po otestovaní vo všetkých nástrojoch v rámci AVISPA vyšlo, že namodelovaný protokol Kerberos basic je zabezpečený aj všetky požadované tajné hodnoty (AK,SK,...). Výsledok z nástroja Cl-AtSe je možné vidieť na obrázku 5.1.



Obr. 5.1: Vyhodnotenie bezpečnosti protokolu Kerberos basic v Cl-AtSe.

5.1.2 Protokoly jednosmernej autentizácie

Princíp modelovania navrhnutých protokolov jednosmernej autentizácie je v podstate podobný. Do modelovania sa nezahŕňa počiatočný prenos ID, K_{ID} a sekvenčného čísla s_n od Mastra. Modelovaná je bezpečnosť protokolu až po prijatí týchto hodnôt. U každého protokolu je potrebné zadefinovať jednotlivé role slave a master, u ktorých hrajú hlavnú úlohu agenti Slave a Master. Ďalej je potrebné zadefinovať kanály odosielania (`SND`) a prijímania (`RCV`), ktoré využívajú Dolev-Yao model.

V rámci každej úlohy (role) sa zadefinujú správy `Hello`, `Auth` a postupne vygenerujú jedinečné čísla `n_M` a `n_S`. Problém u nástroja AVISPA je ten, že nepodporuje modelovanie aritmetických operácií. Tento problém sa dá obísť definovaním funkcie `Succ`, ktorá navýši hodnotu `S_n` o 1. Definovaná je v lokálnych premenných. Odlišnosti v definíciách jednotlivých protokolov budú ďalej popísané jednotlivo.

Použitie symetrickej šifry

V tomto prípade je potrebné do hlavičky zadefinovať symetrický kľúč K_{ID}. Do lokálnych premenných sa ďalej definujú sekvenčné čísla `S_n` a `S_n1` (označuje sekvenčné číslo s_n navýšené o 1) a `ID`. Ako konštanty vystupujú `sn`, `sn1`, `sec_S_K_ID` a `sec_M_K_ID`. Inicializačný stav sa nastaví na hodnotu 0. Samotný prenos správ je napríklad pre úlohu *slave* nasledujúci:

```
1. State   = 0   /\ RCV( start ) =|>
   State' := 1   /\ N_s' := new()
                 /\ SND(Hello.ID.N_s'.{Hello.ID.N_s'.S_n'}_K_ID)
                 /\ witness(Slave,Master,sn,S_n')
                 /\ secret(K_ID',sec_S_K_ID,{Master,Slave})
2. State   = 1   /\ RCV(Auth.N_M.Sym1')
                 /\ S_n1' = Succ(S_n')
                 /\ Sym1' = {Auth.N_M.S_n1'}_K_ID =|>
   State' := 2   /\ wrequest(Slave,Master,sn1,S_n1')
```

kde `RCV( start )` označuje začiatok komunikácie. Potom sa prepne do ďalšieho stavu a vytvorí sa jedinečné číslo `N_s'`, to sa spolu s `Hello`, `ID` a ich zašifrovanou hodnotou pridanou o `S_n'` odošle na Mastra. Witness dosvedčuje, že sekvenčné číslo `S_n` je zabezpečené a secret overí bezpečnosť kľúča `K_ID`. To všetko sa deje v rámci prvej výmeny medzi Slaveom a Mastrom.

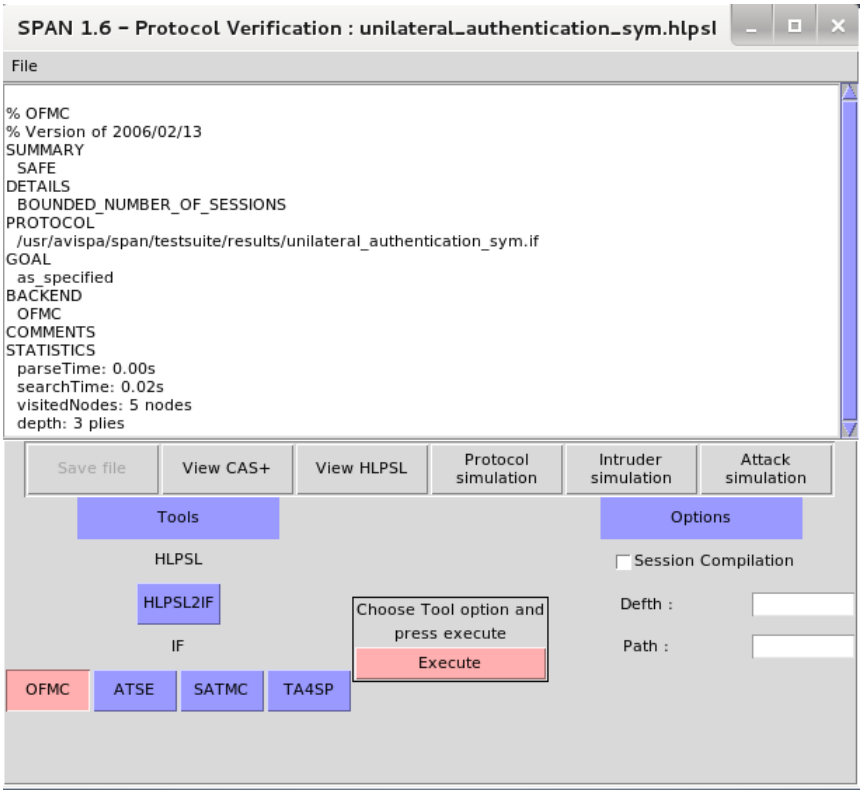
V druhej výmene sa najskôr prijíme od Mastra správa `Auth` spolu s jedinečným číslom `N_M` a zašifrovanou správou `Sym1`. V ďalšom kroku Slave navýši hodnotu sekvenčného čísla `S_n` o 1 a potom porovná prijatú zašifrovanú správu s vlastnou vypočítanou správou `{Auth.N_M.S_n1'}_K_ID`. Ak sa rovnajú, tak je zaručená autentičnosť správ. V poslednom kroku sa požaduje overenie pomocou `wrequest`, či je sekvenčné číslo `S_n1` zabezpečené. Úloha sa nakoniec ukončí príkazom `end role`.

V úlohe relácia (session) sa jednotlivé kanály usporiadajú do paralelných relácií:

```
composition
            slave(Slave,Master,SendSlave,ReceiveSlave,K_ID)
         /\ master(Slave,Master,SendMaster,ReceiveMaster,K_ID)
```

V úlohe prostredie (environment) sa zadefinujú znalosti protivníka: `slave, master` a `k_I` (kľúč útočníka). Tie sa usporiadajú paralelne do relácií. Nakoniec sa v cieloch (goal) vypíše požadované overenie zabezpečenia kľúča `K_ID` a sekvenčných čísel `sn` a `sn1`. Celý kód je možné nájsť v priloženom CD.

Vyhodnotenie daného návrhu protokolu vyšlo vo všetkých nástrojoch v rámci AVISPA ako zabezpečené (safe), iba u TA4SP ako bezvýchodiskové (inconclusive), pretože tento nástroj nepodporuje výskyt premenných v počiatočnom stave. Výsledok z nástroja OFMC je možné vidieť na obrázku 5.2.



Obr. 5.2: Vyhodnotenie bezpečnosti protokolu jednosmernej autentizácie použitím symetrickej kryptografie v OFMC.

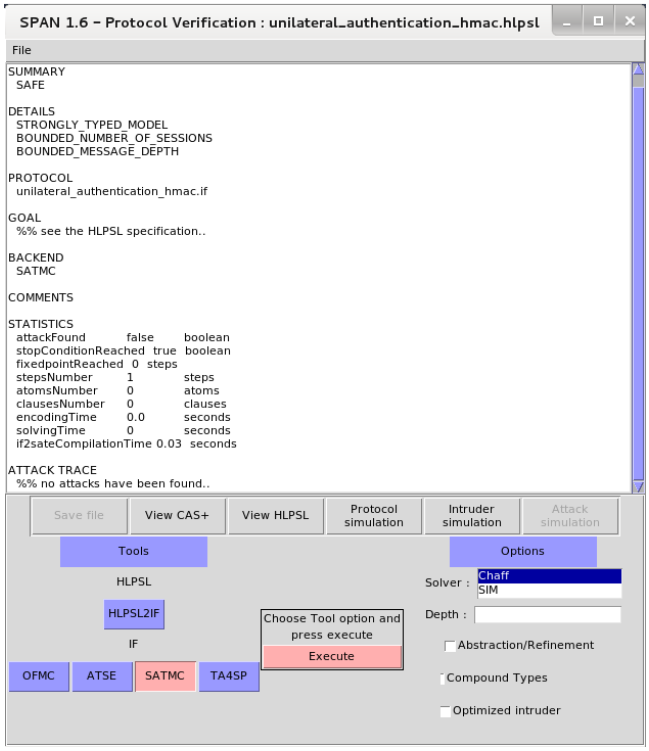
Použitie funkcie HMAC

V tomto prípade je potrebné zadefinovať do hlavičky $HMAC$ funkciu. Do lokálnych premenných sa zadefinujú rovnaké hodnoty ako v predošlom prípade. Navyše sa pridá hodnota kľúča doplneného do veľkosti základného bloku a xor-ovaného s hodnotou opad a potom ipad, tie sú označené ako `K_IDXOR1` a `K_IDXOR2`. Definícia konštánt zahrňuje parametre potrebné pre overenie zabezpečenia doplnených kľúčov a funkcie HMAC z oboch strán a overenie zabezpečenia sekvenčných čísel S_n

a S_{n+1}. Inicializačný stav sa nastaví na hodnotu 0 a samotný prenos medzi Slaveom a Masterom zo strany Mastra je:

```
1. State = 0    /\ RCV(Hello.ID.N_s'.HMAC1')
                /\ HMAC1' = Hash(K_IDXOR1.Hash(K_IDXOR2.Hello.ID.N_s'.
                /\ S_n')) =|>
   State' := 1  /\ S_n1' := Succ(S_n')
                /\ N_M' := new()
                /\ HMAC2' := Hash(K_IDXOR1'.Hash(K_IDXOR2'.Auth.N_M'.
                /\ S_n1'))
                /\ SND(Auth.N_M'.HMAC2')
```

kde Master najskôr príjme od Slavea hodnoty `Hello, ID, N_s` a hodnotu funkcie `HMAC1`. Túto hodnotu v ďalšom kroku porovná so svojou vypočítanou hodnotou. Ak sa rovná, prepne sa do ďalšieho stavu. Tu navýši `S_n` o hodnotu 1 a potom vytvorí nové číslo `N_M`. Kvôli zjednodušeniu sa vypočíta funkcia `HMAC2` osobitne a potom sa spolu so správou `Auth` a číslom `N_M` odošle Slaveovi. V tejto ukážke sú vynechané požiadavky na overenie pomocou `witness, wrequest` a `secret`. Celý kód je možné nájsť na priloženom CD.



Obr. 5.3: Vyhodnotenie bezpečnosti protokolu jednosmernej autentizácie použitím funkcie HMAC v SAT-MC.

Podobne ako v predošlom prípade sa v úlohe relácia usporiadajú jednotlivé kanály do paralelných relácií. Oproti predošlému prípadu sa kľúč `K_ID` zamení za `HMAC` funkciu. V úlohe prostredie sa zadefinujú znalosti protivníka, a to `master,slave` a `hmac`. Nakoniec sa zadefinujú ciele, ktoré chceme dosiahnuť (napr. overenie zabezpečenia funkcie `HMAC` z oboch strán).

Vyhodnotenie daného návrhu vyšlo vo všetkých nástrojoch ako zabezpečené, okrem TA4SP, kde vyšlo bezvýchodiskové. Výsledok z nástroja SAT-MC je možné vidieť na obrázku 5.3.

Použitie hashovacej funkcie

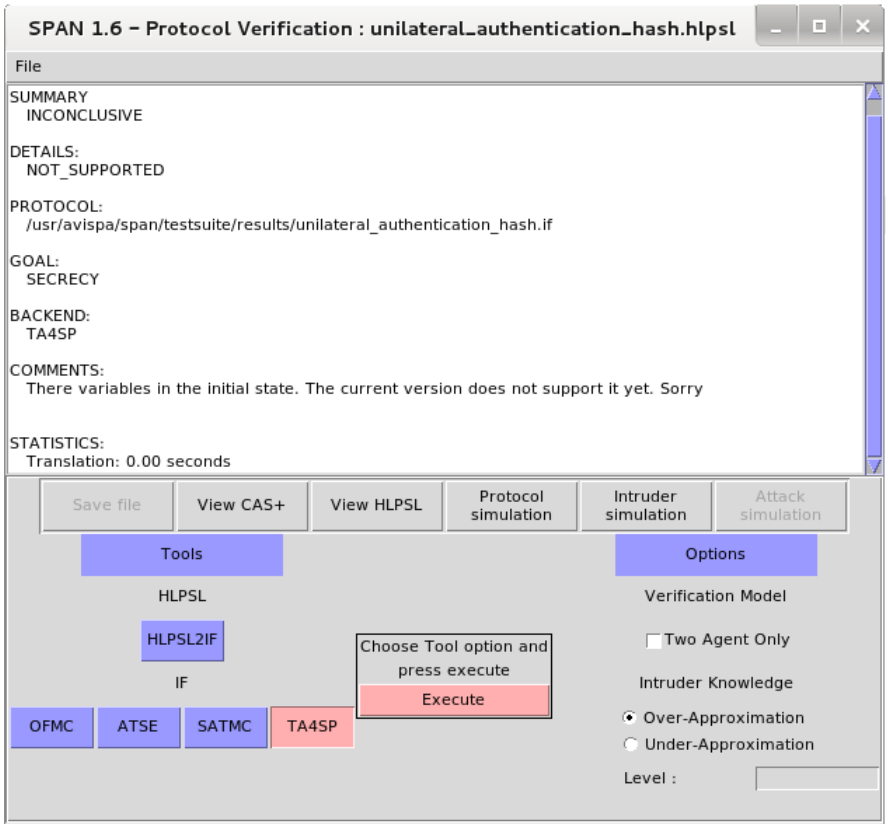
Tu je potrebné pridať do hlavičky hashovaciu funkciu *Hash*. Táto definícia neurčuje presne použitú hashovaciu funkciu (napr. md5 alebo sha-1), je možné použiť akúkoľvek. Modelovanie je veľmi podobné predošlým príkladom. Do lokálnych sa okrem vyššie zmienených základných parametrov pridá symetrický kľúč `K_ID`. V rámci konštánt sa zadefinujú parametre pre overenie zabezpečenia hashovacej funkcie a kľúča `K_ID` z oboch strán a zabezpečenia sekvenčných čísel. Inicializačný stav sa nastaví opäť na 0. Samotný prenos medzi Slaveom a Masterom je veľmi podobný tomu z modelovania použitia funkcie `HMAC`. Ako príklad je uvedená úloha Slave:

```
1. State   = 0  /\ RCV( start ) =|>
   State' := 1  /\ N_s' := new_()
                /\ SND(Hello.ID.N_s'.Hash(Hello.ID.N_s'.S_n'.K_ID))
2. State   = 1  /\ RCV(Auth.N_M.Hash2')
                /\ S_n1' = Succ(S_n')
                /\ Hash2' = Hash(Auth.N_M'.S_n1'.K_ID) =|>
   State' := 2  /\ wrequest(Slave,Master,sn1,S_n1')
```

kde Slave začne komunikáciu. Prepne sa do ďalšieho stavu a vytvorí jedinečné číslo `N_s`, to sa spolu s `Hello, ID` a zahashovanou hodnotou parametrov odošle na Mastra. V ďalšom kroku príjme správu obsahujúcu hodnoty `Auth, N_M` a zahashovanú funkciu od Mastra. Slave si porovná prijatú hodnotu so svojou vypočítanou a ak sa rovnajú, prepne sa do ďalšieho kroku. Tu sa požaduje overenie bezpečnosti sekvenčného čísla s_{n+1}.

V úlohe relácia sa kanály usporiadajú do paralelných relácií. Oproti predošlým príkladom sa na koniec určí funkcia `Hash`. V úlohe prostredie sa zadefinujú znalosti protivníka `slave, master` a `hash`. Nakoniec sa v cieľoch vypíše požadované overenie zabezpečenia hashovacej funkcie a kľúča `K_ID` z oboch strán a sekvenčných čísel. Celý kód je možné nájsť v priloženom CD.

Vyhodnotenie vyšlo vo všetkých nástrojoch ako zabezpečené, iba u TA4SP ako bezvýchodiskové. Výsledok z nástroja TA4SP je možné vidieť na obrázku 5.4.



Obr. 5.4: Vyhodnotenie bezpečnosti protokolu jednosmernej autentizácie použitím funkcie Hash v TA4SP.

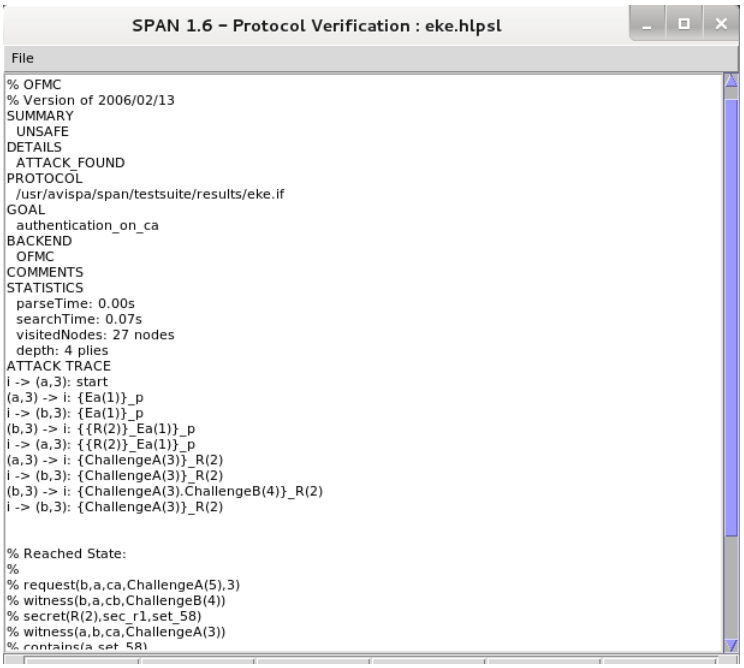
5.1.3 Encrypted key exchange

Modelovanie tohto protokolu je v podstate jednoduché. Je potrebné namodelovať dve úlohy, a to Alica a Bob. V hlavičkách oboch úloh sa zadefinujú agenti `A` a `B`, zdieľané heslo `P` a kanál prijímania (`RCV`) a odosielania (`SND`). V lokálnych premenných sa zadefinuje stav, verejný kľúč `Ea`, výzvy `ChallengeA` a `ChallengeB` a symetrický kľúč `R`. Ako konštanty sú uvedené `sec_r1` a `sec_r2`. Samotný prenos správ je napríklad pre Boba nasledujúci:

```
1. Stav   = 0  /\ RCV ({Ea'}_P)  =|>
   Stav' := 1  /\ R' := new()
               /\ SND({{R'}_Ea'}_P)
               /\ secret(R',sec_r2,{A,B})
2. Stav   = 1  /\ RCV({ChallengeA'}_R)  =|>
   Stav' := 2  /\ ChallengeB' := new()
               /\ SND({ChallengeA'.ChallengeB'}_R)
               /\ witness(B,A,cb,ChallengeB')
```

```
3. Stav   = 2  /\ RCV({ChallengeB'}_R)  =|>
   Stav' := 3  /\ request(B,A,ca,ChallengeA')
```

kde celý prenos je popísaný už vyššie pri definícii protokolu. Navyše sú tu `request`, ktoré požaduje najskôr overenie bezpečnosti výzvy `ChallengeA` a potom aj `ChallengeB`, tieto sú dosvedčované s `witness` a nakoniec `secret`, ktoré overuje bezpečnosť kľúča `R`. Toto patrí aj k cieľom (goal) overovania. V úlohe relácia sa dajú jednotlivé úlohy do paralelných relácií a v úlohe prostredie sa zadefinujú znalosti útočníka, ktoré sú `a,b`. Zdrojový kód je možné vidieť na priloženom CD. Vyhodnotenie tohto protokolu vyšlo vo všetkých častiach nástroja AVISPA ako nezabezpečené (unsafe). Ukážku vyhodnotenia v nástroji OFMC je možné vidieť na obrázku 5.5 a vyobrazenie útoku na obrázku A.2.



Obr. 5.5: Vyhodnotenie bezpečnosti protokolu EKE v OFMC.

5.2 ProVerif

Nástroj ProVerif je možné stiahnuť zo stránok www.prosecco.gforge.inria.fr. Ja som nainštaloval verziu 1.89. K jeho chodu je potrebné doinštalovať Objective Caml - OCaml verziu 4.02.1 (univerzálny programovací jazyk), ktorý poskytuje byte-kódový kompilátor (ocamlc) a native-kódový kompilátor (ocamlopt). Native-kódový kompilátor nie je striktne požadovaný, ale je odporúčaný. Po stiahnutí súboru a rozbalení do požadovaného priečinku sa ProVerif nainštaluje jednoducho príkazom

`./build` v danom priečinku. Samotné spúšťanie tohto nástroja sa vykoná príkazom `./proverif [options] <filename>`. Na písanie kódu je odporúčané použiť textový editor emacs, ale je možné použiť aj hociktorý iný, len je potrebné uložiť daný súbor s príponou `.pv`.

Modelovanie

Pri popise práce v tomto nástroji som vychádzal z manuálu [24]. Modelovanie v tomto nástroji sa skladá z troch základných častí: *deklarácia* (popis), kde sa zadefinuje správanie primitív, *process macros*, kde sa zadefinujú podprocesy a *hlavný proces*, kde je zakódovaný protokol s použitím makier (môže sa ukončiť 0).

Deklarácia sa začína definovaním kanála príkazom `free c:channel`, v rámci ktorého budú jednotlivé procesy spolu komunikovať. Ďalej sa zadefinujú použité typy (napr. `type server`), funkcie (napr. `fun encryption()`), premenné (napr. `free K:key`) a konštanty (napr. `const S:server`). Ak je pri definovaní premenných použitý len príkaz `free`, potom sú tieto premenné známe útočníkovi. Ak je požadované, aby neboli známe, je potrebné doplniť príkaz `free` o doplnok `[private]`. Potom sa zadefinujú `event` (udalosti), ktoré slúžia na zachytávanie vzťahov medzi javmi. Tieto vzťahy môžu byť vyjadrené v tvare: ak udalosť *e* bola vykonaná, potom udalosť *e'* bola vykonaná predtým. Okrem toho, tieto udalosti môžu obsahovať argumenty, ktoré umožňujú študovať vzťahy medzi argumentmi udalostí. Označujú sa na dôležité etapy protokolu, ale nijak neovplyvňujú jeho správanie. Pomocou príkazu `query` (dotaz) sa sledujú vybrané parametre, ktorými sa nakoniec vyhodnotí bezpečnosť daného protokolu. Tieto `query` môžu byť zamerané buď čisto na útočníka (`attacker`) alebo aj na udalosti (`event`).

U process macros sa definujú jednotlivé makrá podprocesov príkazom `let <process> (parametre)`, pričom parametre nie sú povinné. V týchto podprocesoch sa definujú jednotlivé vymieňané správy medzi procesmi. Jednotlivé správy sa určujú príkazmi `in` a `out`, pričom sa určí najskôr kanál, po ktorom komunikujú a potom prijaté (in) alebo odoslané (out) parametre (napr. `out(c,(K,n1))`).

V hlavnom procese sa definuje čo a ako sa má spustiť. V rámci toho sa využíva znak |, ktorý značí, že procesy budú spúšťané paralelne. Napríklad:

```
process( (!processA) | (!processB) )
```

Vyhodnotenie bezpečnosti

Pri vyhodnotení sa najskôr vyobrazí celý proces, a potom vyhodnotené jednotlivé požadované parametre z `query`. Ak je daný parameter zabezpečený, ukáže sa vyhodnotenie, že je true (pravdivé). Ak by bol nezabezpečený, ukáže sa, že je false

(nepravdivé) a vyobrazí sa, kde presne útočník získa daný parameter, ukáže sa celý útok. Ukážku je možné vidieť na obrázku A.3.

5.2.1 Kerberos

Modelovanie tohto protokolu začína určením kanála, po ktorom budú jednotlivé procesy spolu komunikovať (`free c:channel`). Zadefinujú sa používané typy a potom šifrovacia funkcia, ktoré definuje zároveň aj dešifrovaciu funkciu.

```
fun enc(bitstring, key): bitstring.
reduc forall x: bitstring, y: key; dec(enc(x,y),y) = x
```

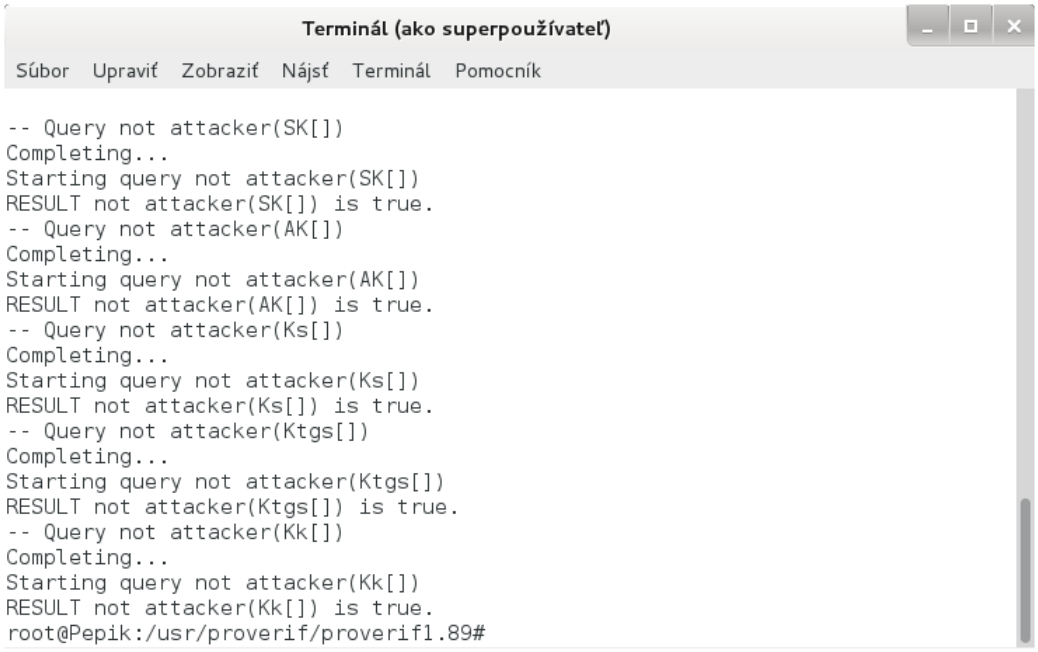
kde `dec(enc(x,y),y) = x` hovorí, že dešifrovaním šifrovaných hodnôt `x` a `y` podľa `y` dostaneme hodnotu `x`. V ďalších krokoch sa určia súkromné (`free Ks: key [private]`) a verejné (`free uptk: timestamp`) premenné a použité konštanty. Pomocou `query attacker()` sa určia premenné, ktoré sa budú overovať (napr. `Kk, AK`). Pred samotným definovaním makier procesov sa zadefinujú udalosti, pomocou ktorých sa zachytávajú vzťahy medzi javmi (napr. `event BeginKKAS(klient)`). V rámci tohto protokolu je potrebné zadefinovať 4 makrá podprocesov a to: `processKlient`, `processKas`, `processTgs` a `processServer`. Definícia napríklad pre `processServer` je nasledujúca:

```
let processServer=
          in(c,(m4:bitstring,m6:bitstring);
          let(host:klient,SK1:key,tk:timestamp)= dec(m4, Ks) in
          event BeginSK(host);
          let m7=enc((S,uptk),SK) in
          out(c,m7);
          event EndSK(S,host,SK1,uptk).
```

kde najskôr server príjme zašifrovanú správu skladajúcu sa z dvoch častí od klienta. Prvá časť `m4` je ST (Service ticket), ktorý server dešifruje svojím kľúčom `Ks`. Z neho zistí názov klienta, ktorému bude povolené používať služby a symetrický kľúč `SK` spolu s dobou platnosti tiketu (`tk`). Druhá časť slúži klientovi na overenie, že server S je živý, tým že S vráti časovú značku klientovi K. Príkazmy `event BeginSK(host)` a `event EndSK(S,host,SK1,uptk)` sa vytvorili udalosti, ktoré slúžia na overenie vzťahov medzi klientom K a serverom S. V ďalšej správe vytvorí server odpoveď na klienta s obnovenou hodnotou časovej značky a názvom serveru zašifrovanými kľúčom `SK`. Túto správu odošle kanálom `c` späť na klienta. Viac je možné vidieť v priloženom CD. Samotný proces je definovaný ako:

```
process
       (
       (!processKlient)|(!processKas)|(!processTgs)|(!processServer)
       )
```

Vyhodnotenie bezpečnosti tohto protokolu je možné vidieť na obrázku 5.6, ktorý je výpisom z terminálu. Samotné vyhodnotenie začína výpisom celého procesu. Ak sa nenájde žiadna bezpečnostná chyba (ako v tomto prípade), tak sa vyobrazí, že sledovaný parameter (premenná) je true (pravdivý).



Obr. 5.6: Vyhodnotenie bezpečnosti protokolu Kerberos v nástroji ProVerif.

5.2.2 Protokoly jednosmernej autentizácie

Podobne ako pri modelovaní týchto protokolov v nástroji AVISPA, ani tu sa neberie do úvahy počiatočný prenos ID, K_{ID} a sekvenčného čísla s_n od Mastra. Ako prvé je potrebné zadefinovať kanál, po ktorom budú jednotlivé procesy komunikovať. Ako ďalšie sa zadefinujú potrebné typy a vytvorí sa funkcia na výpočet súčtu dvoch hodnôt, keďže ProVerif nepodporuje aritmetické operácie (`fun sum(integer,integer) :integer`). Ďalej k spoločnej definícií patrí aj definovanie premenných a konštánt. Hlavný proces je definovaný ako `process((!processSlave)|(!processMaster))`.

Použitie symetrickej šifry

V tomto prípade sa zadefinuje tá istá šifrovacia funkcia aká bola použitá u protokolu Kerberos. Do premenných sa pridá tajný symetrický kľúč `K_ID` a sekvenčné čísla `s_n` a `s_n1`. Do konštánt sa zadefinujú správy `Hello,Auth` a `ID` spolu s prirodzeným číslom `one`, ktoré predstavuje číslo 1. Vykoná sa kontrola zabezpečenia kľúča K_{ID} a sekvenčných čísel s_n a s_{n+1}. Definujú sa jednotlivé udalosti, z ktorých sa pre ukážku zadefinuje overenie na základe návezných udalostí:

```
query y:integer, z:integer; event(EndSM(y)) ==> event(EndMS(y,z).
query x:ident; event(BeginMS(x)) ==> event(BeginSM(x)).
```

kde napríklad v druhom prípade sa overuje, že udalosť od Mastra k Slaveovi naväzuje na udalosť, kde sa overí, že Master získa identifikátor (ID) po tom, čo ho Slave odošle Masterovi. Nasleduje samotná definícia makier procesov. Pre proces Slave je nasledujúca:

```
let processSlave=
     event BeginSM(ID);
     new n_s:nonce;
     let m1=enc(Hello,ID,n_s,s_n),K_ID) in
     out(c,(Hello,ID,n_s,m1));
     in(c,(Auth1:message,n_M1:nonce,m2:bitstring));
     let(A:message,nm:nonce,sn1:integer)=dec(m2,K_ID) in
     let snm11=sum(s_n,one) in
     if A=Auth1 && nm=n_M1 && sn1=snm11 then
     event EndSM(sn1) else 0.
```

kde sa najskôr začne udalosť `BeginSM(ID)` a vygeneruje jedinečné číslo `n_s`. Zašifruje sa daná správa do reťazca bitov `m1` a odošle sa spolu s `Hello,ID` a `n_s` Masterovi. Potom od Mastra príjme správu `Auth1`, jedinečné číslo `n_M1` a reťazec bitov `m2`, z ktorého po dešifrovaní získa jednotlivé hodnoty. Slave navýši hodnotu sekvenčného čísla `s_n` o 1. Tieto odšifrované hodnoty porovná s nezašifrovanými prijatými hodnotami a ak sa rovnajú, potom sa ukončí udalosť `EndSM(sn1)`. V opačnom prípade sa proces ukončí, čo je dané príkazom `0`. Viac je možné vidieť v priloženom CD. Vyhodnotenie v nástroji ProVerif vyšli všetky pozorované parametre ako zabezpečené.

Použitie funkcie HMAC

Definícia tohto protokolu je o trochu zložitejšia v tom, že je potrebné zadefinovať každú použitú hashovaciu funkciu osobitne (napr. `fun h1(bitstring,bitstring) :bitstring`). Je to z toho dôvodu, že funkcie je potrebné zadefinovať vždy pre rôzne

použité typy. Podobne ako u nástroja AVISPA, aj tu sa berie do úvahy už doplnený kľúč K_{ID} do veľkosti základného bloku a xor-ovaného s hodnotu opad a ipad (`K_IDXOR1`). Definujú sa použité premenné a konštanty a zadá sa, čo sa bude overovať. V tomto prípade to bude okrem sekvenčných čísel aj overenie bezpečnosti rozšíreného a doplneného kľúča `K_IDXOR1` a `K_IDXOR2`. Bližšie je možné vidieť v priloženom CD. Nasleduje definícia makier procesov, kde pre Mastra je:

```
let processMaster=
    in(c,(Hello1:message,ID1:ident,n_s1:nonce,m2:bitstring));
    let hm3=h2(Hello1,ID1,n_s1,s_n,K_IDXOR2) in
    let hm4=h1(K_IDXOR1,hm3) in
    event BeginMS(ID1);
    if hm4=m2 then
    let s_n1=sum(s_n,one);
    new n_m:nonce;
    let m3=h4(Auth,n_m,s_n1,K_IDXOR2) in
    let m4=h3(K_IDXOR1,m3) in
    out(c,(Auth,n_m,m4));
    event EndMS(s_n1,K_IDXOR1,K_IDXOR2) else 0.
```

kde Master príjme správu od Slavea, ktorá obsahuje okrem iného aj reťazec bitov `m2`. Master vypočíta vlastnú hodnotu HMAC funkcie z prijatých hodnôt a porovná s prijatou hodnotou HMAC funkcie. Ak sa rovnajú, navýši sa hodnota sekvenčného čísla `s_n` o 1 a vytvorí sa nové jedinečné číslo `n_m`. Master vypočíta novú hodnotu HMAC funkcie `m4`, ktorú odošle spolu s ďalšími parametrami Slaveovi. V opačnom prípade sa proces ukončí. Vyhodnotenie daného protokolu a jeho sledovaných častí vyšlo ako zabezpečené.

Použitie hashovacej funkcie

Modelovanie tohto protokolu je podobné tomu predošlému. Opäť sa zadefinujú jednotlivé hashovacie funkcie a kontroluje sa zabezpečenie kľúča `K_ID` a sekvenčných čísel. Ďalej sa zadefinujú udalosti a makrá jednotlivých procesov. Pre proces Slave je nasledujúce:

```
let processSlave=
    event BeginSM(ID);
    new n_s:nonce;
    let m1=hs(Hello,ID,n_s,s_n,K_ID) in
    out(c,(Hello,ID,n_s,m1));
    in(c,(Auth1:message,n_m1:nonce,m2:bitstring));
```

```
let sn2=sum(s_n,one) in
let h2=hm(Auth1,n_M1,sn2,K_ID) in
if h2=m2 then
event EndSM(sn2,K_ID) else 0.
```

kde sa začne udalosť a vytvorí nové jedinečné číslo `n_s`. Vytvorí sa hash `m1`, kde sa zahashujú jednotlivé parametre spolu s kľúčom `K_ID` a odošle sa spolu s danými nešifrovanými hodnotami Masterovi. Potom sa príjme správa od Mastra, kde sa okrem iného nachádza aj hash `m2`. Slave navýši hodnotu sekvenčného čísla `s_n` o 1 a vypočíta vlastný hash z prijatých hodnôt, kľúča `K_ID` a navýšeného sekvenčného čísla a porovná ho s prijatým hashom. Ak sa rovnajú ukončí sa udalosť `EndSM()`. V opačnom prípade sa proces ukončí. Viac je možné vidieť v priloženom CD. ProVerif vyhodnotil daný protokol a jeho sledované časti ako zabezpečené.

5.2.3 Encrypted key exchange

V tomto prípade je potrebné najskôr zadefinovať kanál, po ktorom budú dané podprocesy komunikovať (`free c:channel`). Ďalej je potrebné zadefinovať použité typy. Použité boli `text`, `symmetric_key` a `public_key`. Keďže jednotlivé zložky správy pri symetrickom šifrovaní sú odlišné, je potrebné pre každú správu zadefinovať vlastnú funkciu. Bližšie je možné vidieť v priloženom CD. Ďalej je potrebné pridať funkciu asymetrickej kryptografie:

```
fun aenc(symmetric_key, public_key):bitstring.
reduc forall m: symmetric_key, k: public_key; adec(aenc(m,k),k) = m.
```

Ako premenné boli použité kľúče `Ea`, `R` a heslo `P`. Cieľom overovania je zabezpečenie kľúča `R` a hesla `P`. Nasleduje definícia jednotlivých makier procesov (`processAlica` a `processBob`). Pre jednoduchosť sú ukázané niektoré vymieňané správy v smere od Alice k Bobovi:

```
in(c,b1:bitstring);
let(b11:bitstring)= dec(b1,P) in
let(r1:symmetric_key)= adec(b11,Ea) in
new challengeA: text;
out(c,enc3(challengeA,r1));
in(c,b2:bitstring);
let(ca1:text,cb1:text)= dec(b2,R) in
if ca1=challengeA then
let a2=enc3(cb1,R) in
```

kde nie je zobrazená prvá ani posledná vymieňaná správa. Celý kód je možné vidieť na priloženom CD. V prípade druhej správy Alica príjme bitový tok `b1`, ktorý sa dešifruje najskôr pomocou symetrickej kryptografie (pomocou zdieľaného hesla P), tým sa získa bitový reťazec `b11`. Ten potom Alica dešifruje pomocou asymetrickej kryptografie (pomocou svojho súkromného kľúča, ktorý je matematicky zviazaný s verejným kľúčom `EA`, ktorý Alica zaslala v prvej správe Bobovi). Tým sa získa hodnota kľúča R (`r1`), ktorý bude použitý pri ďalšom šifrovaní. Vygeneruje sa nová výzva `challengeA` a zašifrovaná kľúčom `r1` sa odošle na Boba. Potom sa príjme bitový tok `b2`. Po jeho dešifrovaní sa získa hodnota `ca1` a `cb1`. Hodnota `ca1` sa porovná s vygenerovanou. Ak sa rovnajú, Alica odošle zašifrovanú hodnotu výzvy `cb1` kľúčom `R` na Boba. Hlavný proces má tvar `(!processAlica)|(!processBob)`. Vyhodnotenie zabezpečenia tohto protokolu je možné vidieť na obrázku A.3. Tento protokol je nezabezpečený a vyobrazil sa útok na neho.

5.3 Scyther

Nástroj Scyther je možné stiahnuť zo stránok www.cs.ox.ac.uk. Ja som nainštaloval verziu Compromise-0.9.2, ktorá je pre podporu rôznych modelov protivníka. Pre chod tohto nástroja je potrebné doinštalovať *graphviz*, ktorý slúži na vykreslenie grafov, *python*, pretože je v ňom napísaný GUI a *wxpython* knižnice, ktoré sa používajú na vykreslenie miniaplikácie. Po stiahnutí tohto nástroja do požadovaného priečinka a jeho rozbalení je možné ho spustiť viacerými spôsobmi. Pre prácu v GUI je potrebné v termináli (nie pod root) zadať v priečinku príkaz `./scyther-gui.py`. Ďalej je možné ho spustiť aj bez GUI len v termináli, ale ja sa zameriam len na prácu v GUI. Jazyk použitý na modelovanie je založený na C/Java syntaxi a je case-sensitive (citlivý na veľkosť písmen).

Modelovanie

Pri popise práce v tomto nástroji som vychádzal z manuálu [25]. Protokoly v tomto nástroji sú definované ako množina úloh a tieto úlohy ako sekvencia udalostí. Scyther manipuluje s termínmi: *atomické*, kde ide o hocijaký identifikátor, typicky reťazec alfanumerických znakov; *konštanty* a *premenné*, kde príkaz `fresh` označuje náhodné čísla a príkaz `secret` tajné parametre. Scyther má preddefinované užívateľské typy Agent, Function, Nonce a Ticket. Príkazom `usertype` je možné dodefinovať vlastné typy. Pred definovaním samotného protokolu je možné zadefinovať pomocné protokoly, ktoré sú špecifické počiatočným znakom @. Je tu možné pridať aj ďalšie súbory príkazom `include`. Samotná definícia protokolu vyžaduje minimálny vstup:

```
protocol Example (I,R) {
                role I {}
                role R {}
                }.
```

V rámci každej úlohy (role) sa najskôr zadefinujú premenné, kde príkazom `fresh` definujeme nové, náhodné hodnoty a príkazom `var`, prijaté hodnoty. Ďalej sa zadefinujú jednotlivé vymieňané správy medzi úlohami príkazmi `send` a `recv` s poradovými číslami, ktoré medzi sebou korešpondujú (`send_1` musí mať `recv_1`). Nakoniec sa príkazom `claim` (požiadavka) určí to, čo bude overované. Tento príkaz sa používa s ďalšími upresňujúcimi tvrdeniami. Pri použití tvrdenia `Secret` sa vyžaduje termín parametrov (`claim(R,Secret,S)`, kde S je parameter neznámy pre útočníka a R je úloha. Pri použití tvrdenia `Reachable` (dosiahnuteľný) a po jeho overení Scyther skontroluje, či môže byť toto tvrdenie celkovo dosiahnuteľné. Je pravdivé, ak existuje stopa, v ktorej došlo k tomuto tvrdeniu. Je to užitočné pre kontrolu, či sa nenachádza nejaká chyba v špecifikácii protokolu. Viac tvrdení pre `claim` (Alive, Weakagree, Commit, Running,...) je možné nájsť v jeho manuále.

Scyther umožňuje používať príkaz `macro`, ktorým sa vytvárajú skratky pre konkrétny termín a tým pádom sa zjednodušujú špecifikácie protokolov. Pre potlačenie varovania, že `send` nemá korešpondujúci `recv`, sa používa znak ! (`send_!1`). Na porovnávanie sa používajú nové funkcie `match(X,Y)` (zhoda) a `not match(X,Y)`, ktoré porovnávajú, či sú dané prvky zhodné alebo nie.

Vyhodnotenie bezpečnosti

Po vyhodnotení sa vyobrazí okno s určením bezpečnosti podľa tvrdení v `claim`. Ak je daná vlastnosť zabezpečená, vyobrazí sa OK a prípadne Verified (overené). V prípade, že je nezabezpečená, vyobrazí sa Fail a vykreslí sa graf útoku. OK a Fail sú vyobrazené inými odtieňmi zelenej a červenej, čím je tmavší odtieň, tým je daná vlastnosť zabezpečenejšia resp. nezabezpečenejšia.

5.3.1 Kerberos

Modelovanie tohto protokolu začína definíciou užívateľských typov a zabezpečených funkcií, ktoré sa použijú pri definícii pomocných protokolov. Tieto pomocné protokoly je nutné použiť kvôli neschopnosti nástroja Scyther rozoznať problémy v symetrii, kde `k(I,R)!=k(R,I)`. Príklad použitia je nasledujúci:

```
protocol @swap-key-kkas(I,R)
        {    role I {
                        var T:Ticket;
```

```
                recv_!1(R,I, {T}kkas(I,R));
                send_!2(I,R, {T}kkas(R,I)); }
        role R { } }
```

Potom nasleduje definícia samotného protokolu. Je potrebné zadefinovať 4 úlohy, a to K, KAS, TGS a S. V každej z jednotlivých úloh sa zadefinujú nové použité hodnoty (príkazom `fresh`) alebo prijaté hodnoty (príkazom `var`). Všade sa zadefinujú použité kľúče (`AK`), jedinečné čísla (`n1`) a časové značky (`ttgsstart`). Nasleduje výmena správ. Napríklad pre úlohu TGS je nasledujúca:

```
recv_3(K,TGS, { AK,K,tkasstart,tkasexp }ktgs(TGS,KAS),
              { K,tkstart,tkexp }AK, K,S,n2);
send_4(TGS,K, K, { SK,K,ttgsstart,ttgsexp }ks(S,TGS),
              { SK,n2,ttgsstart,ttgsexp,S }AK);
```

kde sa najskôr príjme správa od klienta, v ktorej je TGT, názov serveru `S` spolu s jedinečným číslom `n2` a overenie, ktoré dokáže TGS, že skutočne pozná autentizačný kľúč `AK` (`{ K,tkstart,tkexp }AK`). Potom TGS odošle späť na klienta odpoveď s názvom klienta `K`, ST šifrovaný kľúčom `ks(S,TGS)` a upresnenie parametrov tiketu zašifrované kľúčom `AK`. Nakoniec sa príkazom `claim` určí, že sa bude overovať dosiahnuteľnosť úlohy TGS (tým sa overí, či K a KAS sú dôveryhodné) a bezpečnosť kľúča SK. Ide o príkazy `claim(TGS,Reachable)` a `claim(TGS,Secret,SK)`. Viac je možné vidieť v priloženom CD. Vyhodnotenie bezpečnosti tohto protokolu je možné vidieť na obrázku 5.7.

Scyther results : verify

Claim				Status		Comments	Patterns
kerberosBasic	K	kerberosBasic,K1	Reachable	Ok	Verified	At least 1 trace pattern.	1 trace pattern
		kerberosBasic,K2	Secret AK	Ok		No attacks within bounds.	
		kerberosBasic,K3	Secret SK	Ok		No attacks within bounds.	
	KAS	kerberosBasic,KAS1	Secret AK	Ok		No attacks within bounds.	
	TGS	kerberosBasic,TGS1	Reachable	Ok	Verified	At least 1 trace pattern.	1 trace pattern
		kerberosBasic,TGS2	Secret SK	Ok		No attacks within bounds.	
	S	kerberosBasic,S1	Reachable	Ok	Verified	At least 1 trace pattern.	1 trace pattern
		kerberosBasic,S2	Secret SK	Ok		No attacks within bounds.	

Done.

Obr. 5.7: Vyhodnotenie bezpečnosti protokolu Kerberos v nástroji Scyther.

5.3.2 Protokoly jednosmernej autentizácie

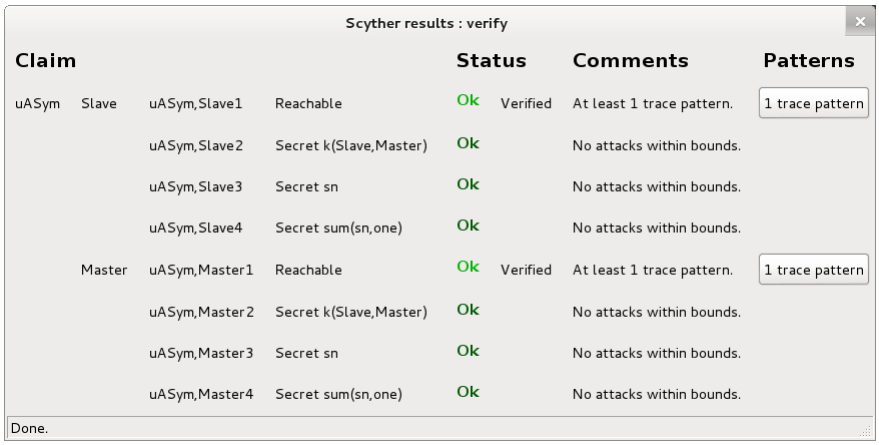
Tak ako u predošlých nástrojov sa neberie pri modelovaní týchto protokolov do úvahy počiatočný prenos ID, K_{ID} a sekvenčného čísla s_n od Mastra. Medzi spoločné črty definície každého z týchto protokolov patrí definícia použitých typov a konštánt. Je potrebné definovať aj funkciu `sum`, ktorá slúži ako aritmetická operácia sčítavanie, keďže ani tento nástroj nepodporuje aritmetické operácie. U každého je potrebné zadefinovať dve role (úlohy), a to Slave a Master.

Použitie symetrickej šifry

Pri použití symetrickej šifry je potrebné podobne ako u protokolu Kerberos zadefinovať pomocný protokol. V rámci samotných úloh sa zadefinujú všetky nové (`fresh`) a prijaté (`var`) parametre. Samotná definícia protokolu napríklad z pozície Slavea je:

```
macro sn1 = sum(sn,one);
send_1(Slave,Master, Hello,ID,ns,{ Hello,ID,ns,sn }k(Slave,Master));
recv_2(Master,Slave, X,Y,{ Auth,nM,sn1 }k(Slave,Master));
match(X,Auth);
match(Y,nM);
```

kde sa najskôr navýši hodnota sekvenčného čísla `sn` o 1. Potom sa odošle zo Slavea správa Mastrovi s hodnotami `Hello,ID,ns` a ich zašifrovanou hodnotou spoločne s `sn`. Následne sa od Mastra príjmu hodnoty `X,Y` a zašifrovaná správa. Po dešifrovaní sa príkazom `match` porovnajú tieto hodnoty s prijatými `X` a `Y`. Ak sa rovnajú je zaručená autentičnosť. Príkazom `claim` sa vyžaduje overenie, či je Slave dosiahnu-



Obr. 5.8: Vyhodnotenie bezpečnosti protokolu jednosmernej autentizácie použitím symetrickej kryptografie v nástroji Scyther.

teľný (užitočné pre kontrolu, či sa nenachádza nejaká chyba v špecifikácii protokolu), či je bezpečný kľúč zdieľaný medzi nimi (K_{ID}) a či sú bezpečné sekvenčné čísla `sn` a `sn1`. Viac je možné vidieť v priloženom CD. Výsledok po prevedení testu je možné vidieť na obrázku 5.8.

Použitie funkcie HMAC

U funkcie HMAC sa podobne ako v predošlom prípade najskôr zadefinujú použité typy a konštanty, kde je potrebné dodefinovať hashovaciu a HMAC funkciu. Podrobnejšie je možné vidieť v priloženom CD. V prípade modelovanie protokolu pre úlohu Master sa najskôr definujú nové hodnoty, prijaté hodnoty a dve makrá, ktoré slúžia na zjednodušenie protokolu a potom vlastný prenos:

```
macro sn1 = sum(sn,one);
macro HMAC1 = H(KIDXOR1,H(Hello,ID,ns,sn,KIDXOR2));
recv_1(Slave,Master, Hello,ID,ns,X);
match(X,HMAC1);
send_2(Master,Slave, Auth,nM,H(KIDXOR1,H(Auth,nM,sn1,KIDXOR2)));
```

kde Master najskôr príjme správu od Slavea s danými parametrami a HMAC funkciou vypočítanú hodnotu `X`. Potom sa táto hodnota porovná s vlastnou vypočítanou hodnotou funkcie HMAC (`HMAC1`). Ak sa rovnajú je zaručená autentičnosť a odošle sa na Slavea odpoveď s danými parametrami a ich vypočítanou hodnotou funkcie HMAC. V tomto prípade sa príkazom `claim` vyžaduje overenie bezpečnosti jednotlivých rozšírených a doplnených kľúčov a sekvenčných čísel. Overí sa aj dosiahnuteľnosť danej úlohy. Vyhodnotenie bezpečnosti tohto protokolu je podobné ako v predošlom prípade. Je bezpečný vo všetkých zisťovaných bodoch.

Použitie hashovacej funkcie

Pri modelovaní tohto protokolu sa opäť definujú použité typy spolu s konštantami, kde sa pridá definícia hashovacej funkcie. V rámci jednotlivých úloh sa definujú nové a prijaté hodnoty. Pre príklad, v prípade úlohy Slave sa najskôr definujú dve makrá kvôli zjednodušeniu a potom je samotný prenos:

```
macro sn2 = sum(sn,one);
macro H2 = H(Auth,nM,sn2,KID);
send_1(Slave,Master, Hello,ID,ns,H(Hello,ID,ns,sn,KID);
recv_2(Master,Slave, Auth,nM,Y);
match(Y,H2);
```

kde Slave odošle Mastrovi správu s danými parametrami a hash týchto hodnôt spolu so sekvenčným číslom `sn` a tajným kľúčom `KID`. Potom Slave príjme správu, kde sa

nachádzajú dané parametre spolu so zahashovanou hodnotou `Y`. Slave porovná prijatý hash so svojím vypočítaným hashom, ak sa rovnajú je zaručená autentičnosť. Nakoniec sa vyžiada overenie bezpečnosti daného protokolu tým, že sa overí dosiahnuteľnosť danej úlohy, bezpečnosť tajného kľúča `KID` a sekvenčných čísel. Viac je možné vidieť v priloženom CD. Vyhodnotenie je veľmi podobné predošlým. Vo všetkých sledovaných bodoch je tento protokol bezpečný.

5.3.3 Encrypted key exchange

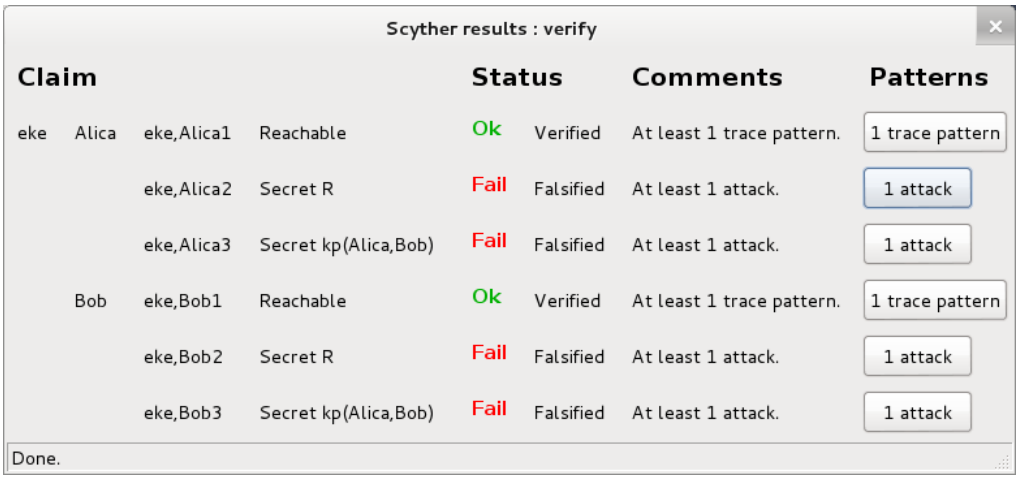
Modelovanie tohto protokolu v nástroji Scyther sa začína definovaním použitých typov, a to `SymmetricKey`, `AsymmetricKey` a `Text`. Kvôli vyššie popísanému problému nerovnosti u symetrickej kryptografie (viď Kerberos) je potrebné zadefinovať pomocný protokol:

```
protocol @swap-key-kp(I,R) {
                role I { var T: Ticket;
                    recv_!1(R,I, { T }kp(I,R));
                    send_!2(I,R, { T }kp(R,I)); }
                role R { } }
```

Ďalej už nasleduje samotná definícia protokolu EKE. Pre úlohu Alica sú vymieňané správy nasledujúce:

```
send_1(Alica,Bob, { EA }kp(Alica,Bob));
recv_2(Bob,Alica, { {R}EA }kp(Alica,Bob));
send_3(Alica,Bob, { challengeA }R);
recv_4(Bob,Alica, { challengeA,challengeB }R);
send_5(Alica,Bob, { challengeB }R);
```

kde Alica najskôr vygeneruje verejný kľúč `EA`. Tento kľúč zašifruje heslom `kp(Alica, Bob)` a odošle na Boba. V nasledujúcej správe získa Alica dešifrovaním prijatej správy najskôr podľa hesla `kp(Alica,Bob)` a potom pomocou jej súkromného kľúča, ktorý je matematicky zviazaný s verejným kľúčom `EA` (použitý k zašifrovaniu kľúča `R`), symetrický kľúč `R`, ktorý bude použitý ku šifrovaniu ďalšej komunikácie. Alica vygeneruje výzvu `challengeA` a zašifrovanú kľúčom `R` ju odošle na Boba. Potom Alica dešifrovaním ďalšej prijatej správy kľúčom `R` získa výzvu `challengeB` od Boba. Nakoniec túto výzvu zašifruje kľúčom `R` a odošle na Boba. Overenie u oboch úloh je rovnaké. Overuje sa, či sú dosiahnuteľné (kontrola chyby v kóde) a bezpečnosť kľúča `R` a zdieľaného hesla `kp(Alica,Bob)`. Viac je možné vidieť na priloženom CD. Vyhodnotením daného protokolu vyšlo, že je nezabezpečený (obr. 5.9). Z tohto obrázka je aj vidieť, že nie je v definícii protokolu žiadna chyba (status `Reachable` je OK). Príklad útoku na úlohu Alica a kľúč `R` je možné vidieť na obrázku A.4.



Obr. 5.9: Vyhodnotenie bezpečnosti protokolu EKE v nástroji Scyther.

5.4 Porovnanie

Prácou v týchto nástrojoch som prišiel k záveru, že nástroje k automatizovanému a poloautomatizovanému vyhodnoteniu bezpečnosti kryptografických protokolov sú veľmi efektívne, najmä pri porovnaní s ručnými metódami, kde by boli potrebné náročné a zdĺhavé ručné výpočty. Každý z týchto nástrojov má svoje silné a slabé stránky. Ich spoločnou veľkou výhodou je, že dokážu veľmi podrobne popísať útok. Nástroje AVISPA a Scyther dokážu tieto útoky vyobraziť aj vo forme grafu, pričom nástroj Scyther vyhodnocoval najlepšie (keďže má najlepšie prepracované grafické rozhranie). Tieto grafy sú navyše samo popisujúce, čiže je jednoduché pochopiť daný výstup. Nástroj ProVerif napriek tomu, že nemá grafické rozhranie, dokáže útok popísať veľmi podrobne. Hneď ako nájde útok, zastaví sa v tom bode a pokračuje postupnou ukážkou celej komunikácie medzi útočníkom a obeťou.

V prípade porovnania rýchlosti vyhodnotenia bezpečnosti protokolov sú na tom najlepšie nástroj ProVerif a súčasti nástroja AVISPA OFMC a Cl-AtSe, ktoré zvládli vyhodnotiť aj náročnejší protokol Kerberos pod 1 sekundu. Najpomalším bol SAT-MC, ktorý bezpečnosť overil až za 6532 sekúnd. Porovnanie vyhodnotenia bezpečnosti protokolu Kerberos je možné vidieť v tabuľke 5.1. Platí, že čím náročnejší protokol tým dlhší čas vyhodnotenia bezpečnosti najmä v nástrojoch SAT-MC a Scyther. Samozrejme, závisí aj od výkonu daného počítača. Jednoduchšie protokoly popísané vyššie zvládli všetky nástroje pod 1 sekundu. Veľkým mínusom pre Scyther je, že ak sa nachádza niekde v modelovanom protokole nejaká logická chyba (problém so symetrickou kryptografiou), tak je schopný vyhodnocovať aj viac ako deň a výsledkom je, že daný protokol je nezabezpečený.

Tab. 5.1: Porovnanie rýchlosti vyhodnotenia bezpečnosti protokolu Kerberos.

Nástroj	**Čas vyhodnotenia** [s]
Cl-Atse	<1
OFMC	<1
SAT-MC	6532
TA-4SP	20
ProVerif	<1
Scyther	10

Vo všetkých nástrojoch je potrebné zadať presne to, čo chceme overovať. Navyše sa nevyhodnotí nič, preto je potrebné dôkladne si premyslieť čo overovať. Nástroj AVISPA vyhodnotí len to, čo je zadané v `goals` a požadované pomocou `request`, `witness` a `secret`. ProVerif len to, čo je zadané v `query` a Scyther len to, čo je zadané v `claim`. Pri vyhodnotení bezpečnosti sa používa aj porovnávanie medzi prijatými a vlastnými hodnotami. Jedine nástroj ProVerif používa priamo $if \ldots then \ldots else$. V ďalších dvoch nástrojoch nie je toto porovnanie úplne jednoznačné, AVISPA využíva len znak porovnania = po prijatí správy a Scyther len príkazy `match` a `not match`. Pomocou týchto nástrojov je možné overiť akýkoľvek parameter. Napríklad u Scythera dosiahnuteľnosť (Reachable) danej roli, ktorá je dobrá pre kontrolu, či sa nenachádza v špecifikácii nejaká chyba. ProVerif dokáže overiť aj nadväzujúce udalosti.

Problémom týchto nástrojov je, že nevedia narábať s aritmetickými počtami. Je potrebné zadať pomocné funkcie, ktoré dajú z dvoch zadaných hodnôt jednu výslednú (u AVISPA je to funkcia `Succ()`, u ProVerif a Scyther funkcia `sum(x,y)`). U nástroja Scyther nastáva problém so symetrickou kryptografiou, kde nie je schopný rozoznať problémy v symetrii pri `k(I,R)!=k(R,I)`. Rieši sa to použitím pomocných protokolov.

6 ZÁVER

V mojej práci som sa zameral na kryptografiu. Popisujem jej rozdelenie na klasickú a modernú, v rámci ktorej som popísal najznámejšie problémy teórie čísel také ako faktorizácia čísel, problém diskrétneho logaritmu a problém eliptického diskrétneho logaritmu. Ďalej som sa zaoberal modelmi Dolev-Yao a BAN logika, ktoré sa používajú k vyhodnoteniu formálnej bezpečnosti protokolov. Zistil som, že v drvivej väčšine sa používa Dolev-Yao model, ktorý je sofistikovanejší. Využíva sa v mnohých nástrojoch využívaných k formálnej analýze bezpečnostných protokolov. K najpoužívanejším patria automatické nástroje AVISPA (ktorá sa ďalej skladá z nástrojov OFMC, CL-AtSe, SAT-MC a TA4SP), ProVerif, CryptoVerif, Scyther a Casper/FDR, ktorý sa skladá z nástroja Casper a z CSP modelového kontroléru FDR. Popísal som vybrané protokoly Kerberos, EKE a protokoly jednosmernej autentizácie, ktoré som neskôr otestoval v nástrojoch AVISPA, ProVerif a Scyther. Všetky majú svoje výhody a nevýhody. Najlepšie sa mi pracovalo s nástrojom ProVerif, kde modelovanie bolo v celku intuitívne a dokázal som namodelovať presne to, čo som chcel. U nástroja Scyther oceňujem hlavne jeho dôkladné grafické vyobrazenie útoku a u nástroja AVISPA možnosť testovať pomocou jeho štyroch častí, čím je výsledná analýza dôkladnejšia. Menšou nevýhodou u AVISPA je, že jeho časť TA4SP vyhodnotí často bezpečnosť protokolu ako Inconclusive (bezvýchodiskový) kvôli rôznym problémom (napr. nepodporuje výskyt premenných v inicializačnom stave). U nástroja Scyther je nedostatkom neúplný užívateľský manuál, čím sa práca v ňom stáva náročnejšou. Zo všetkého najdôležitejšie je pochopiť daný protokol ako pracuje, akým spôsobom si vymieňa správy a čo presne má v daných správach byť. Z toho sa vytvorí idealizovaný model, s ktorým je možné ďalej pracovať v nástrojoch.

LITERATÚRA

[1] JONSSON, J., KALISKI B. *RFC 3447 - Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. Technická správa, Internet Engineering Task Force, 2003, [cit. 30. 10. 2014]. Dostupné z URL: <https://tools.ietf.org/html/rfc3447#section-3>.

[2] MENEZES, Alfred J., VAN OORSCHOT, Paul C., VANSTONE, Scott A. *Handbook of applied crypthography*. CRC press, 2010.

[3] RESCORLA, E. *RFC 2631 - Diffie-Hellman Key Agreement Method*. Technická správa, Internet Engineering Task Force, 1999, [cit. 31. 10. 2014]. Dostupné z URL: <https://www.ietf.org/rfc/rfc2631.txt>.

[4] OCHODKOVÁ, Eliška. *Matematické základy kryptografických algoritmu*. Plzeň, Ostrava, 2011, 96 s.

[5] LEVICKÝ, Dušan. *Kryptografia v informačnej bezpečnosti*. Košice: elfa nakladateľstvo, 2005, 266 s. ISBN 80-8086-022-x.

[6] MEADOWS, Catherine. *Open issues in formal methods for cryptographic protocol analysis*. In: DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings. IEEE, 2000. p. 237-250.

[7] MEADOWS, Catherine A. *Formal verification of cryptographic protocols: A survey*. IN: Advances in Cryptology - ASIACRYPT'94, 1995. p. 133-150.

[8] DOLEV, D. a Yao, A. *On the security of public key protocols*. IEEE Transactions on information theory, 29(2):198-208, March 1983.

[9] BURROWS, M., ABADI, M., NEEDHAM, R. *A logic of authentication.* . ACM Transactions in Computer Systems, 8(1):18-36, February 1990.

[10] MATSUO, S., MIYAZAKI, K., OTSUKA, A., BASIN, D. *How to evaluate the security of real-life cryptographic protocols? The cases of ISO/IEC 29128 and CRYPTEC*. In: FC'10 Proceedings of the 14th international conference on Financial cryptography and data security, 2010, p. 182-194.

[11] LOWE, G., BROADFOOT, P., DILLOWAY, CH., HUI, M. L. *Casper: A compiler for the analysis of security protocols-User manual and tutorial*. Oxford University Computing Laboratory, September 28, 2009, 85 s., [cit. 2. 12. 2014]. Dostupné z URL: <http://www.cs.ox.ac.uk/gavin.lowe/Security/Casper/manual.pdf>.

[12] SEEBA, I. *CryptoVerif - the tool of crypto analysis*. The University of Tartu, Department of Computer Science, 2010, 12 s., [cit. 2. 12. 2014]. Dostupné z URL: <http://courses.cs.ut.ee/2010/security-seminar-spring/uploads/Main/seeba-final.pdf>.

[13] CREMERS, J.F.C., LAFOURCADE, P., NADEAU, P. *Comparing state spaces in automatic security protocol analysis*. IN: Formal to practical security, Lecture notes in computer science, Volume 5458, 2009. p. 70-94.

[14] CREMERS, C. *The Scyther tool: Verification, falsification, and analysis of security protocols*. In: CAV'08, volume 5123/2008, 2008. p. 414-418.

[15] SISTO, R., PIRONTI, A., POZZA, D. *Spi2Java User Manual - Version 3.1.*. Politecnico di Torino, 33 s., [cit. 2. 12. 2014]. Dostupné z URL: <http://spi2java.polito.it/>.

[16] MEIER, S., SCHMIDT, B. *Tamarin Prover*. ETH Zurich, [cit. 8. 12. 2014]. Dostupné z URL: <http://www.infsec.ethz.ch/research/software/tamarin>.

[17] The team AVISPA. *AVISPA v1.1 User Manual.* . Information Society Technologies Programme (1998-2002), June 30, 2006, 88 s., [cit. 19. 11. 2014]. Dostupné z URL: <http://www.avispa-project.org/>.

[18] NEUMAN, C., TS'O, T. *Kerberos: An Authentication Service for Computer Networks*. IEEE Communications, 32(9):33-38, 1994.

[19] NEUMAN, C., YU, T., HARTMAN, S., RAEBURN, K. *RFC4120 - The Kerberos Network Authentication Service (V5)*. Technická správa, Internet Engineering Task Force, July 2005, [cit. 20. 11. 2014]. Dostupné z URL: <http://www.ietf.org/rfc/rfc4120>.

[20] BUTLER, F., CERVESATO, I., JAGGARD, A. D., SCEDROV, A., WALSTAD, CH. *Formal analysis of Kerberos 5*. School of Computer Science at Research Showcase @ CMU,November 2006, 52 s.

[21] CLUPEK, V., ZEMAN, V. *Unilateral Authentication on Low-cost Devices*. In 38th International Conference on Telecommunications and Signal Processing TSP. 2015.

[22] BELLOVIN, S., MERRIT, M. *Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks*. Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, May 1992, 13 s.

[23] PASTO, D. and Siemens AVISPA team. *HLPSL Tutorial - Beginner's Guide to Modeling and Checking Security Protocols*. Siemens AVISPA team, April 2008, 49 s., [cit. 10. 3. 2015]. Dostupné z URL: <`http://www.avispa-project.org/`>.

[24] BLANCHET, B., SMYTH, B., CHEVAL, V. *ProVerif 1.89: Automatic Cryptographic Protokol Verifier, User Manual and Tutorial*. INRIA Paris-Rocquencourt, Paris, France, December 2014, 113 s., [cit. 20. 3. 2015]. Dostupné z URL: <`http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`>.

[25] CREMERS, C. *Scyther - User Manual*. DRAFT, Februar 2014, 52 s., [cit. 25. 3. 2015]. Dostupné z URL: <`http://www.cs.ox.ac.uk/people/cas.cremers/scyther/index.html`>.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

AES (Advanced Encryption Standard)

AVISPA (Automated Validation of Internet Security Protocols and Applications)

BAN (Burrows, Abadi, Needham)

CL-AtSe (Constraint Logic based Attack Searcher)

CSP (Communication Sequential Processes)

DES (Data Encryption Standard)

DSA (Digital Signature Algorithm)

ECDH (Eliptic Curve Diffie-Hellman)

EKE (Encrypted key exchange)

FDR (Failures Divergences Refinement)

GUI (Graphical User Interface)

HLPSL (High Level Protocols Specification Language)

HMAC (keyed-Hash Message Authentication Code)

KAS (Kerberos Authentication Server)

KDC (Key Distribution Center)

MAC (Message Authentication Code)

MSC (Message Sequence Charts)

NP (Nondeterministic Polynomial)

OFMC (On the Fly Model Checker)

RC4 (Ron‘s Code 4)

RSA (Rivest-Shamir-Adleman)

SAT-MC (boolean SATisfiability based Model Checker)

SHA (Securu Hash Algorithm)

SPAN (Security Protocol ANimator)

ST (Service Ticket)

TA4SP (Tree Automata tool based on Automatic Approximations for the Analysis of Security Protocols)

TGS (Ticket Granting Server)

TGT (Ticket Granting Ticket)

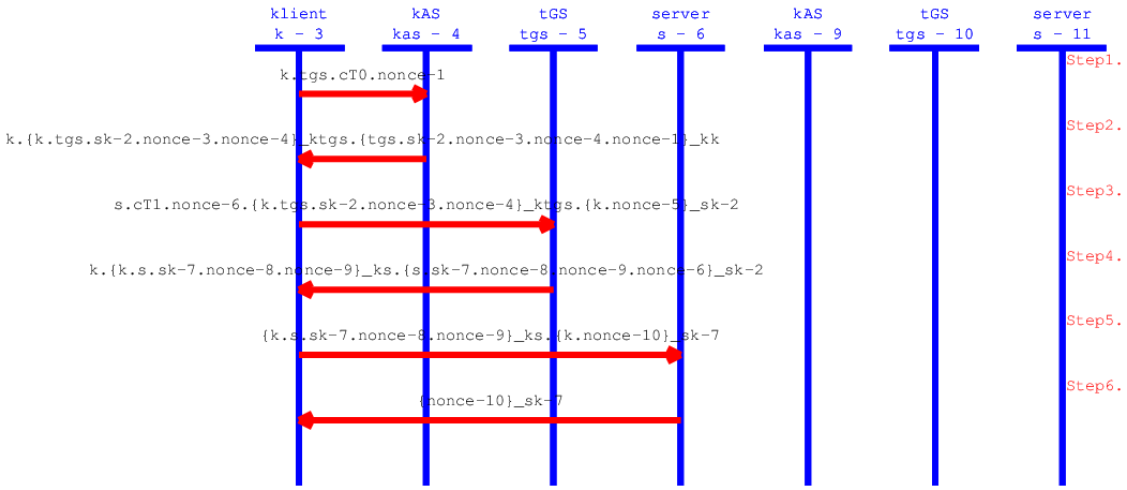
WEP (Wired Equivalent Privacy)

XTEA (Extended Tiny Encryption Algorithm)

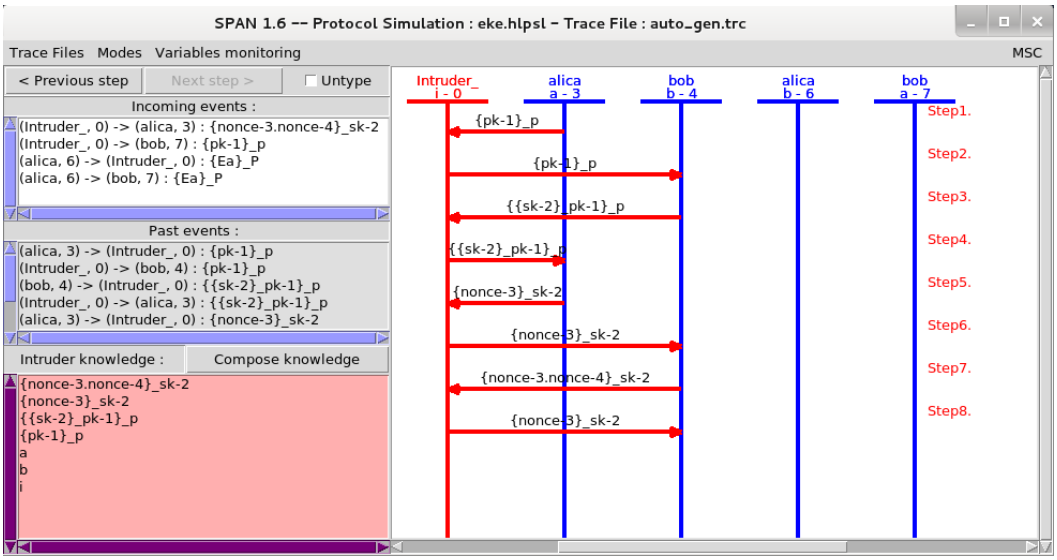
ZOZNAM PRÍLOH

A UKÁŽKY VÝPISU Z POUŽITÝCH PROGRAMOV

A.1 AVISPA

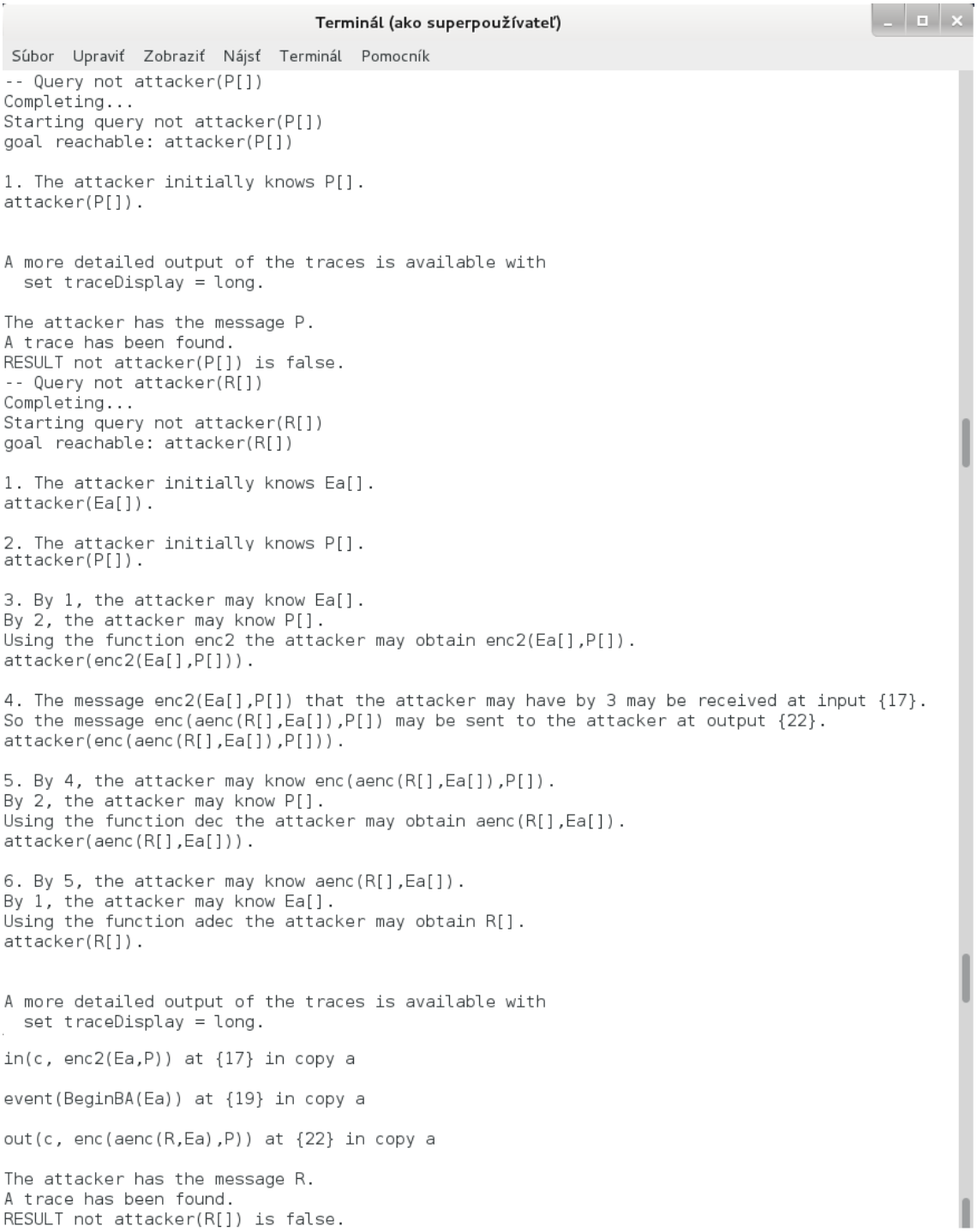


Obr. A.1: Grafická simulácia protokolu Kerberos.



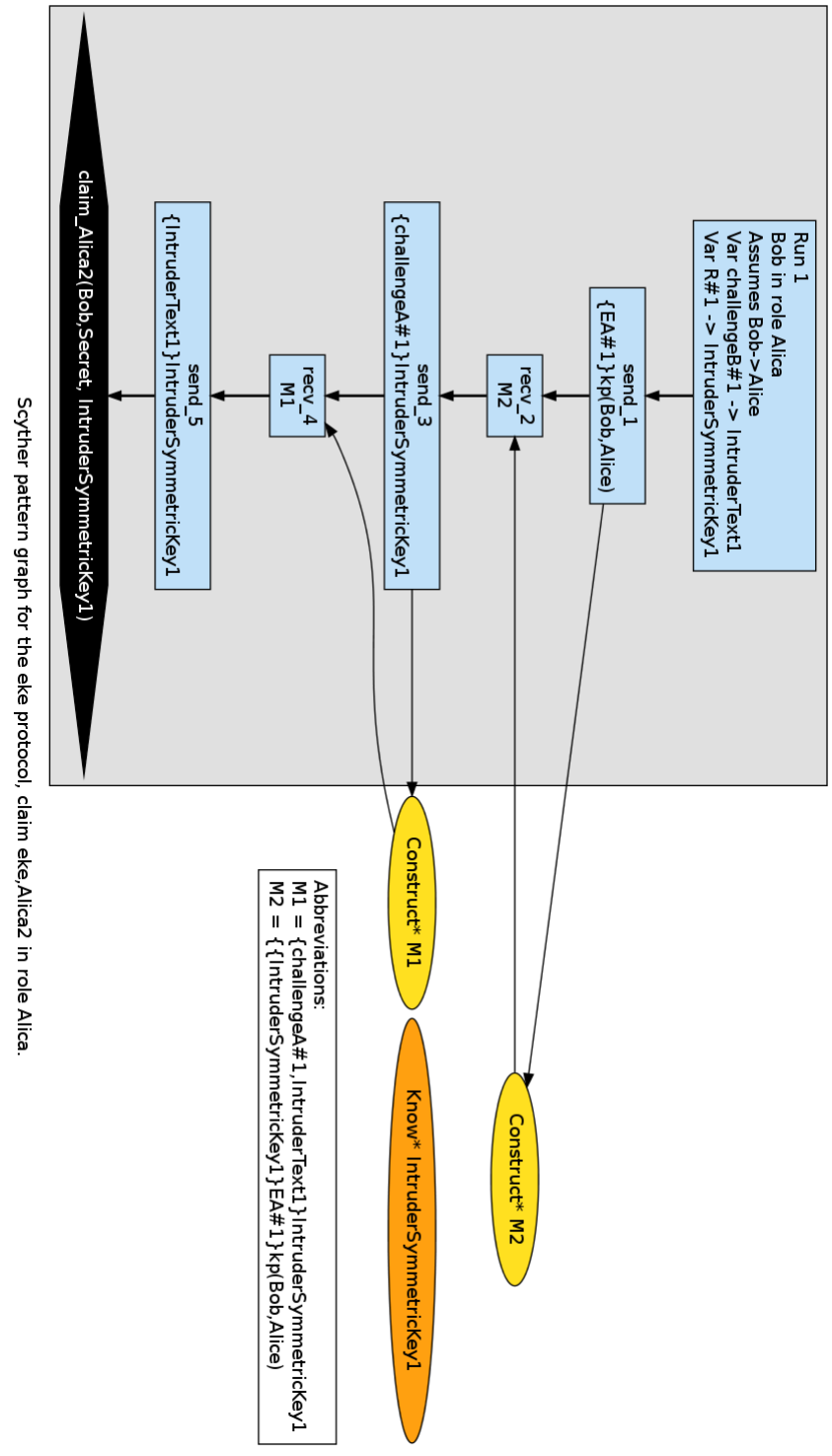
Obr. A.2: Vyobrazenie útoku na protokol EKE v nástroji AVISPA.

A.2 ProVerif



Obr. A.3: Vyobrazenie útoku na protokol EKE v nástroji ProVerif.

A.3 Scyther



Obr. A.4: Vyobrazenie útoku na protokol EKE v nástroji Scyther.