VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

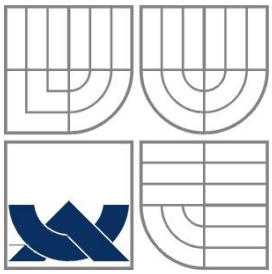
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# ANALÝZA VYBRANÝCH BEZPEČNOSTNÍCH PROTOKOLŮ

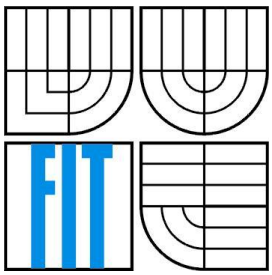
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MAREK MALECKÝ
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# ANALÝZA VYBRANÝCH BEZPEČNOSTNÍCH PROTOKOLŮ

ANALYSIS OF SELECTED SECURITY PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MAREK MALECKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL OČENÁŠEK, Ph.D.
SUPERVISOR

BRNO 2010

# Zadání

1. Nastudujte vybrané bezpečnostní komunikační protokoly a proveďte jejich vzájemné srovnání.
2. Proveďte hloubkovou studii nástrojů a prostředí pro analýzu, simulaci a implementaci komunikačních protokolů.
3. Vzájemně jednotlivé nástroje porovnejte. Diskutujte vhodnost jednotlivých nástrojů pro různé druhy komunikačních protokolů. Zaměřte se na bezpečnostní protokoly.
4. Ve zvoleném prostředí (po konzultaci s vedoucím práce) analyzujte vybrané bezpečnostní protokoly tak, aby je bylo možné simulovat a zkoumat a jejich chování a vlastnosti. Analýzu proveďte tak, aby ji bylo moţné využít v dalších projektech.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti rozšíření projektu.

# Abstrakt

Předmětem této diplomové práce je studium dostupných bezpečnostních protokolů a nástrojů sloužících k jejich verifikaci. První část práce se krátce věnuje popisu pojmů souvisejících s oblastí bezpečnostních protokolů a verifikačních logik. Druhá část již přímo uvádí jednotlivé protokoly spolu s nalezenými útoky a chybami v návrhu. V další kapitole jsou detailněji popsány nejdůležitější nástroje pro automatickou analýzu bezpečnostních protokolů. Hlavní část práce se zabývá verifikací vybraných bezpečnostních protokolů ve zvoleném nástroji Scyther. Na závěr jsou uvedeny příklady víceprotokolových útoků spolu s přehledovou tabulkou.



# Abstract

The subject of this thesis is to study available security protocols and tools for their verification. The first part is devoted to briefly describe the concepts related to the area of security protocols and verification logics. The second part directly lists various protocols, along with attacks and errors found in design. Next chapter describes the most important tools for automatic analysis of security protocols in more detail. The main part deals with verification of security protocols selected in the chosen tool called Scyther. In conclusion, examples of multiprotocol attacks along with a summary table are displayed.



# Klíčová slova

Bezpečnostní protokol, útok na bezpečnostní protokol, víceprotokolové útoky, oboustranná autentizace, verifikační logika, Yahalom, Isabelle, AVISPA, Scyther.

# Keywords

Security protocol, attack on security protocol, multiprotocol attacks, mutual authentication, verification logic, Yahalom, Isabelle, AVISPA, Scyther.


# Citace

Marek Malecký: Analýza vybraných bezpečnostních protokolů, diplomová práce, Brno, FIT VUT v Brně, 2010.

# Analýza vybraných bezpečnostních protokolů

## Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně pod vedením Ing. Pavla Očenáška. Při práci jsem vycházel z literatury a pramenů uvedených v seznamu použitých zdrojů a z konzultací s vedoucím diplomové práce.

……

24.05.2010

Marek Malecký

## Poděkování

Děkuji především vedoucímu své diplomové práce Ing. Pavlu Očenáškovi, Ph.D. za užitečné rady, připomínky a čas, který mi věnoval. Děkuji také Janě Virágové za připomenutí pravidel českého pravopisu.

# Obsah

# 1 Úvod

Bezpečnost hraje v současném světě velmi důležitou roli a o světě počítačů to platí dvojnásob. Informace se stala jedním z nejvíce ceněných artiklů. Ten, kdo ví více, má v konkurenčním prostředí nezanedbatelnou výhodu. S tím, jak se mění počet připojených uživatelů a stanic k internetu, se také mění bezpečnostní požadavky a standardy nutné pro jejich ochranu. To, co bylo bezpečné před deseti lety, se najednou jeví jako nedostatečné. Kryptografické algoritmy jsou prolomeny během několika dnů s pomocí velkého počtu propojených počítačů, komunikace už neprobíhá pouze po zabezpečených linkách, znalosti útočníků jsou díky sdíleným informacím daleko větší atd.

Tento problém se nevyhnul ani oblasti bezpečnostních protokolů. S příchodem nových analytických a verifikačních metod se ukázalo, že některé bezpečnostní protokoly jsou chybně navrženy, což umožňuje nepoctivým uživatelům úspěšně napadnout probíhající komunikaci a tím získat přístup k ceněným informacím. Proto je nutné, aby nově navrhované či nyní používané bezpečnostní protokoly byly důsledně analyzovány a případné chyby odstraněny. A zde se otevírá prostor pro automatické verifikační nástroje, se kterými lze dané bezpečnostní protokoly analyzovat v relativně krátkém časovém úseku.

Tato práce se zabývá bezpečnostními protokoly a nástroji, které slouží pro jejich verifikaci. Po krátkém úvodu do problematiky jsou jednotlivé části věnovány představení verifikačních logik a vybraných bezpečnostních protokolů. Bezpečnostním protokolům je věnována celkem podstatná část teoretického úvodu v této práci. To proto, aby bylo pro čtenáře snadnější pochopit principy a slabiny uvedených protokolů, ale také aby byl vytvořen základ pro budoucí analýzu. Pátá kapitola je věnována detailnímu popisu vybraných dostupných nástrojů a jejich srovnání. Některé z uvedených nástrojů již byly použity v nějaké z dřívějších bakalářských a diplomových prací[1], nebudou proto zahrnuty do seznamu kandidátů pro praktickou část.

V šesté kapitole budou analyzovány vybrané bezpečnostní protokoly pomocí zvoleného nástroje. Cílem bude ověřit známé útoky na tyto bezpečnostní protokoly nebo nalézt nové, ještě nepublikované. U každého protokolu bude uveden detailnější popis vlastností protokolu (pokud již nebyl uveden v teoretickém úvodu), následován bezpečnostními požadavky, které by měl protokol zajistit. Verifikace těchto požadavků je stěžejní částí diplomové práce. Případné útoky jsou prezentovány jak písemným popisem, tak grafickým zobrazením.

V poslední části diplomové práce je popsána zvláštní třída útoků na bezpečnostní protokoly, takzvané multiprotokolové útoky. Struktura popisu multiprotokolových útoků se bude řídit stejnými pravidly jako v kapitole 6. Na úplný závěr je uvedena přehledová tabulka analyzovaných protokolů, nalezených útoků a výsledků verifikace.

[1] Například práce M. Práčka [53] nebo D. Ondráčka [5].

# 2 Pojmy z oblasti bezpečnostních protokolů

## 2.1 Bezpečnostní protokol

Souhrnný název pro sadu pravidel, která by měla sloužit k zabezpečení přenosu citlivých dat přes nezabezpečenou síť. Tato pravidla aplikují šifrovací metody a bezpečnostní funkce s cílem dosáhnout určitých bezpečnostních požadavků [2].

## 2.2 Kategorizace útoků na bezpečnostní protokoly

Následující kapitola čerpá z [1].

### 2.2.1 Aktivní

V těchto útocích se útočník snaží pomocí modifikace či falzifikace dat oklamat bezpečnostní systém a tím získat přístup k cenným informacím. Je velice těžké kompletně zamezit možnosti aktivního útoku na daný systém pouze preventivními opatřeními. Cílem je tedy spíše detekce těchto útoků a následné rychlé zotavení ze škod jimi způsobených.
Aktivní útoky můžeme rozdělit na čtyři základní typy:

a) **Podvržení identity**: útočník maskuje svoji reálnou identitu a předstírá, že je někdo jiný. Tento útok většinou ještě využívá jiný typ aktivního útoku, např. útoku přehráním.

b) **Útok přehráním:** útočník zopakuje určitá autentizační data, která získal předtím, například odposlechnutím komunikace, a tím získá přístup do systému.

c) **Modifikace zprávy:** část původní pravé zprávy je změněna, pozdržena nebo je přehozeno její pořadí za účelem získání neautorizovaného přístupu.

d) **Odmítnutí služby:** tento útok se liší tím, že jeho cílem není získání přístupu, ale poškození chodu nějaké služby. Útočník zahltí danou službu svými požadavky, což způsobí její nedostupnost pro ostatní uživatele či dokonce její kolaps na delší dobu.

### 2.2.2 Pasivní

Pasivní útoky se zabývají, jak již název sám napovídá, pasivním získáváním dat, například monitorováním a odposlouchávání komunikace. Útočník nevyvíjí nikterak zvláštní aktivitu při napadání systému, spíše se snaží shromažďovat citlivé informace. Tyto útoky jsou náročné na odhalení, proto jsou účinnější preventivní opatření než detekce těchto útoků.
Mezi pasivní útoky patří **odposlouchávání.** To lze ještě rozdělit do dvou podkategorií:

a) **Získání informací z obsahu zprávy** – např. telefonní hovory, email, přenosná média atd.

b) **Analýza provozu** – útočník může získat informace kupříkladu o infrastruktuře, i když není schopen porozumět obsahu zpráv.

## 2.3 Klíč

Klíč je informace, která se používá k šifrování či dešifrování zpráv. Je více druhů klíčů, *symetrický (sdílený) klíč* používaný v symetrické kryptografii, *veřejný* a *soukromý klíč* používaný v asymetrické kryptografii [4].

## 2.4 Kryptografie

Kryptografie je disciplína, která se zabývá tím, jakými metodami lze převést normálně čitelnou zprávu do podoby zprávy utajované, to znamená čitelné jen pomocí určité speciální znalosti. Tyto metody můžeme dohromady nazývat jako šifrování, což je kryptografický algoritmus, který převede zprávu z čitelné podoby (prostý text) na nečitelnou šifru (šifrovaný text) [1].

### 2.4.1 Symetrická šifra

Pro šifrování i dešifrování se používá stejný (symetrický) klíč známý všem účastníkům komunikace.

### 2.4.2 Asymetrická šifra

Tato šifra používá dva *rozdílné* klíče k šifrování a dešifrování zasílaných zpráv. Díky tomuto odpadá nutnost výměny klíčů, která je podmínkou v symetrickém šifrování. K šifrování může kdokoliv používat *veřejný klíč*, zatímco k dešifrování použije každý svůj *soukromý klíč* [4].

## 2.5 Nonce

Tento výraz vznikl jako zkratka pro *number used once* (číslo použité jen jednou). Toto pseudonáhodné číslo má zaručit odolnost vůči útokům přehráním u autentizačních protokolů tím, že se při každém novém běhu protokolu přidá do zprávy [3]. Obnovuje tzv. „čerstvost“ zpráv (nelze použít starší zprávy). Nově vygenerované nonce může sloužit jako sekvenční číslo při identifikaci.

# 3 Verifikační logiky

## 3.1 BAN logika

**Autoři: M. Burrows, M. Abadi, R. Needham**

**Rok: 1989**

**Publikace: BURROWS, M. and ABADI, M. and NEEDHAM, R. A Logic of Authentication. Technical Report 39, *Digital Systems Research Center*, February 1989, p. 1-19.**

Tato logika pojmenovaná po svých tvůrcích představuje metody, jak lze analyzovat existující bezpečnostní protokoly, a tím ukázat jejich nedostatky a slabiny. K zápisu se používá logických formulí, které tak formálně popisují přímo zobrazený krok v analyzovaném bezpečnostním protokolu [7].

Krátce by se tato logika dala charakterizovat jako „kdo věří co a komu“. BAN logika sice položila základ pro formální analýzu bezpečnostních protokolů, nicméně její formální jazyk je celkem omezený.

### 3.1.1 Hlavní logické formule

Kapitoly 3.1.1 a 3.1.2 vycházejí z textu uvedeného v [6].

$A \mid\equiv X$: *A* důvěřuje *X*. *A* věří, že *X* je pravé.

$A \triangleleft X$: *A* vidí *X*. Někdo zaslal *A* zprávu obsahující *X*. *A* ji může přečíst nebo zopakovat.

$A \mid\sim X$: *A* jednou řekl *X*. *A* dříve zaslal zprávu obsahující *X*. *A* důvěřovalo *X*, když zasílal zprávu.

$A \Rightarrow X$: *A* má práva na *X*. *A* je považováno za důvěryhodnou autoritu kontrolující *X*.

$\#(X)$: formule *X* je čerstvá. Zaručuje, že *X* nebylo nikdy předtím použito v předchozích běžích protokolu. To platí pro nonce.

$A \overset{K}{\leftrightarrow} B$: *K* je dobrý klíč ke komunikaci. *A* a *B* používají sdílený klíč *K*, o kterém věří, že je používán pouze jimi nebo třetí důvěryhodnou stranou jednoho z nich.

$\overset{K}{\mapsto} A$: *A* má veřejný klíč *K*. Odpovídající soukromý klíč $K^{-1}$ je znám jen *A* nebo důvěryhodné třetí straně.

$A \overset{X}{\Leftrightarrow} B$: *A* a *B* sdílejí tajemství *X*. Tajemství znají jen *A*, *B* nebo třetí důvěryhodná strana. Typickým příkladem je heslo.

$(X)_K$: formule *X* je šifrována klíčem *K*.

### 3.1.2 Hlavní logická pravidla

#### 3.1.2.1 Pravidlo významu zprávy (Message meaning rule)

Toto pravidlo říká, že identita odesilatele může být zjištěna ze šifrovacího klíče, který byl použit k zašifrování zprávy. Pravidlo platí při použití sdíleného klíče a počáteční předpoklad je $P \neq R$, tedy tvrzení, že $P$ neposlal zprávu $\{X\}_k$ sobě.

$$\frac{P \mid\equiv P \leftrightarrow Q,\ P \lhd \{X\}k}{P \mid\equiv Q \mid\sim X}$$

#### 3.1.2.2 Pravidlo ověření nonce (Nonce verification rule)

V tomto pravidlu se tvrdí, že pokud $P$ věří nonce $X$, pak zároveň musí věřit, že $Q$ také věří $X$. Počáteční předpoklad je, že nonce $X$ není šifrované.

$$\frac{P \mid\equiv \#\{X\},\ P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

#### 3.1.2.3 Pravidlo kompetence (Jurisdiction rule)

Pravidlo formalizuje pojem, co to znamená pro účastníka mít právo na tvrzení $X$.

$$\frac{P \mid\equiv \#\{X\}, P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

Přehled formulí a pravidel lze najít v [6].

### 3.1.3 Analýza protokolu

Analýzu lze rozdělit do následujících kroků:

1. Sepsání předpokladů počátečního stavu.
2. Převedení originálního formálního zápisu protokolu do tzv. *idealizované formy*[2] - idealizace.
3. Výrazy originálního protokolu jsou převedeny do logických formulí vyjadřujících zamýšlený význam zpráv protokolu – anotace.
4. Aplikují se pravidla na předpoklady a tvrzení s cílem zjistit přesvědčení jednotlivých stran protokolu.

Příklady analýzy viz [7] a [10].

[2] Idealizovaná forma protokolu odstraňuje takové části, které nemají vliv na „přesvědčení“ příjemce.

## 3.2 GNY logika

**Autoři: L. Gong, R. Needham, R. Yahalom**

**Rok: 1990**

**Publikace: GONG, L., NEEDHAM, R., YAHALOM, R. Reasoning about Belief in Cryptographic Protocols. In *IEEE Symposium on Security and Privacy*, 1990, p. 234-248.**

Tato logika vychází ze svého předchůdce BAN logiky. Její autoři značně rozšiřují možnosti původní logiky přidáním více pravidel a formulí. Byly přidány logické formule pro zápis jednosměrných hashovacích funkcí, veřejných a soukromých klíčů. Také jsou vyřešeny některé nedostatky a chyby v BAN logice, které nedovolovaly její použití pro určité techniky používané v bezpečnostních protokolech.

Obecně se dá říci, že GNY logika je více detailnější, pracuje s jemnějšími odchylkami v tvrzeních, což jí na jednu stranu dává větší možnost rozlišovat a vyjádřit se, ale také to přináší větší výpočetní a časovou náročnost analytických kroků.

### 3.2.1 Nové pojmy zavedené GNY logikou

#### 3.2.1.1 Vlastnictví (Possesion)

Značení: **P ∋ X**

*P vlastní X.*

Tato formule zavádí termín *vlastnictví* zprávy nebo klíče. Detailněji nám tedy rozlišuje, zda účastník je opravdu vlastníkem určitého atributu, než jak tomu bylo v BAN logice, kde se implicitně toto předpokládalo.

#### 3.2.1.2 Rozpoznatelnost (Recognizability)

Značení: **P |≡ $\phi$(X)**

*P věří, že X je rozpoznatelné.*

*P* rozpozná zprávu *X* podle toho, jestli vyhovuje jeho nastaveným očekáváním ještě před tím, než tuto zprávu přijme.

#### 3.2.1.3 Poctivost (Honesty)

Značení: **P |≡ Q |=> Q |≡ ***

*P věří, že Q je poctivé a kompetentní.*

V BAN logice se předpokládá, že strany účastnící se vykonávání bezpečnostního protokolu jsou důvěryhodné, tzn. pokud účastník právě poslal zprávu, předpokládá se, že v ní věří. Aby byl tento předpoklad vyjádřen, GNY zavádí pojem *poctivost.*

### 3.2.2 Příklady formulí a pravidel

**Pravidlo vlastnictví:**

$$\frac{P \ni (X, Y)}{P \ni X}$$

**Pravidlo rozpoznatelnosti:**

$$\frac{P \mid\equiv \phi(X), P \in K, P \in (X)k}{P \mid\equiv \phi(\{X\}k)}$$

**Pravidlo významu zprávy:**

$$\frac{P \mid\equiv Q \mid\sim X, P \mid\equiv \#(X)}{P \mid\equiv Q \mid\sim X}$$

Přehled formulí a pravidel lze najít v [8].

## 3.3 SVO logika

**Autoři: P. F. Syverson, Paul C. van Oorschot**

**Rok: 1996**

**Publikace: SYVERSON, P. and OORSCHOT, P. C. van. A Unified Cryptographic Protocol Logic. Technical Report 5540-227, *Naval Research Laboratory*, 1996, p. 1-29.**

Autoři této logiky si dali za cíl sjednotit původní BAN logiku spolu s jejími následnými rozšířeními v podobě GNY, AT[11] a VO[12]. Neudělali to ale stylem shromáždění všech symbolů a pravidel do jednoho celku. Vytvořili novou sémantiku, která vystihuje většinu požadovaných rysů z původních logik, ale má zároveň relativně jednoduchý zápis a použití.

### 3.3.1 Jazyk SVO

Nechť $\mathcal{MT}$ je nejmenší jazyk nad množinou zpráv splňující:

- $X$ je zpráva, pokud platí $X \in \mathcal{T}$,
- $F(X_1, ..., X_n)$ je zpráva, pokud $X_1, ..., X_n$ jsou zprávy a $F$ je funkce,
- φ je zpráva, jestliže φ je formule.

Nechť $\mathcal{FT}$ je nejmenší jazyk nad množinou formulí splňující:

- Pokud $X$ je zpráva a $P$ účastník, pak:
  - *P vídí X*
  - *P obdržel X*

- *P říká X*
- *P řekl X*
- *fresh(X)* jsou formule

- Pokud *X* a *Y* jsou zprávy a *K* je klíč, pak:
  - *SV*[3] *(X, K, Y)* je formule (aplikací klíče *K* na podpis *X* se ověří podpis *Y*)
- Pokud *P* a *Q* jsou účastnící a *K* je klíč, pak:
  - $P \leftrightarrow Q$ (pouze *P* a *Q* sdílejí klíč *K*, plus případný autentizační server)
  - $PK\psi$ (*P*, *K*)
  - $PK\sigma$ (*P*, *K*)
  - $PK\delta$ (*P*, *K*) jsou formule
- Pokud *P* je účastník a φ je formule
  - *P* věří φ
  - *P* kontroluje φ (*P* je důvěryhodná autorita ohledně φ)
- ¬φ a $\varphi \wedge \psi$ jsou formule, jestliže φ a ψ jsou formule

### 3.3.2 Axiomy SVO logiky (příklady)

SVO logika je založena na výrokové modální logice a používá dvě odvozovací pravidla:

1. **Modus ponens:** z φ a $\varphi \supset$ [4] ψ odvodíme ψ
2. **Pravidlo vynucování**: z $\vdash \varphi$ odvoď $\vdash P$ věří φ

Poznámka ke značení: $\{X\}_K$ značí zašifrování *X* pomocí sdíleného nebo veřejného klíče *K*, $K^{-1}$ je komplement klíče *K*, $[X]_K$ značí zprávu *X* digitálně podepsanou klíčem *K*.

#### 3.3.2.1 Pravidlo důvěry

Pro každého účastníka *P*, formule φ a ψ platí:

Axiom 1. *P* věří $\varphi \wedge$ *P* věří $(\varphi \supset \psi) \supset$ *P* věří ψ

Axiom 2. *P* věří $\varphi \supset$ *P* věří (*P* věří φ)

První axiom říká, *P* věří všemu, co logicky vyplývá z jeho důvěry, v druhém *P* může říci to, čemu věří.

---

[3] SV – Signature verification, ověření podpisu.

[4] SVO logika používá tuto značku (horseshoe) pro podmínku jako náhradu za → , která je používána jako symbol pro zaslání zprávy v notaci bezpečnostních protokolů.

#### 3.3.2.2 Pravidlo přijímání

Axiom 1. $P$ přijal $(X_1,...,X_n) \supset P$ přijal $X_i$

Axiom 2. ($P$ přijal $\{X\}_K \wedge P$ vidí $K^{-1}) \supset P$ přijal $X$

Axiom 3. $P$ přijal $[X]_K \supset P$ přijal $X$

Pokud $P$ přijal konkatenaci zpráv $X$, pak také přijal určitou zprávu z této konkatenace. Dále pokud $P$ přijal šifrované $X$ a zároveň zná klíč $K$, pak můžeme říci, že přijal nezašifrovanou zprávu $X$. Nakonec, jestliže přijal digitálně podepsané $X$, pak také přijal $X$.

#### 3.3.2.3 Pravidlo říkání

Axiom 1. $P$ řekl $(X_1,...,X_n) \supset (P$ řekl $X_i \wedge P$ vidí $X_i)$

Axiom 2. $P$ říká $(X_1,...,X_n) \supset (P$ řekl $(X_1,...,X_n) \wedge P$ říká $X_i)$

Tyto axiomy můžeme shrnout do tvrzení, že účastník $P$ vždycky vidí to, co řekl.

#### 3.3.2.4 Pravidlo kompetence

Axiom 1. ($P$ kontroluje $\varphi \wedge P$ říká $\varphi) \supset \varphi$

$P$ jako autorita má v otázce ohledně $\varphi$ vždycky hlavní slovo.

Všechna pravidla SVO logiky zde nemá cenu uvádět. Případné zájemce bych odkázal na referenční literaturu [13].

## 3.4 ASVO logika (Automatic Considered SVO)

**Autoři: Taekyoung Kwon a Seongan Lim**

**Rok: 2003**

**Publikace: KWON, T. and LIM, S. Automation-Considered Logic of Authentication and Key Distribution. Lecture Notes. In *Computer Science*, Technical Report 442-457, Springer-Verlag, 2003, p. 1-15.**

ASVO [14] je založena na SVO logice, proto ji zde jen stručně uvedeme. Používá podobný sémantický model pro zápis komunikace bezpečnostních protokolů, který ale některými vylepšeními oproti původní logice dovoluje minimalizovat požadavky na idealizaci protokolů. Oproti SVO logice byly z idealizačního procesu odstraněny kroky interpretace a porozumění zprávám protokolu, které jsou nyní vyjádřeny přímo samotnou logikou. Pro tento účel byl vytvořen konkrétnější jazyk a přidány axiomy pro lepší vyjadřování ohledně víry účastníků.

# 4 Bezpečnostní protokoly

Cílem této kapitoly není zdlouhavý seznam všech bezpečnostních protokolů, se kterými se dneska můžeme setkat. Především jde o jakési představení několika protokolů, jejichž bezpečnostní stránka bude testována v praktické části. Nejsou zde uvedené veškeré protokoly analyzované v nástroji Scyther a naopak. Záměrem také není zacházet do přílišných detailů při popisu. Spíše se jedná o kratší popis funkce a vlastností každého protokolu spolu s uvedením známých útoků. Pro toho, kdo by měl větší zájem seznámit se s některým z protokolů, je u každého uvedena jeho primární publikace.

## 4.1 Andrew Secure RPC

**Autor: Mahadev Satyanarazanan**

**Publikace: SATYANARAZANAN, M. Integrating Security in a Large Distributed System. *ACM Transactions on Computer Systems*, 1989, p. 1-34.**

**Kryptografie: Symetrická**

**Syntaktický zápis komunikace:**

1. *A → B: A, {Na}Kab*
2. *B → A: {Na+1, Nb}Kab*
3. *A → B: {Nb+1}Kab*
4. *B → A: {K'ab, N'b}Kab*

*A*, *B* : účastníci komunikace (*Alice* a *Bob*)
*Na*, *Nb*, *N'b* : nonce
*Kab*, *K'ab*: klíče

**Popis:**

Tento bezpečnostní protokol funguje za účelem distribuce sdíleného klíče, zprávy jsou šifrovány klíčem *Kab* do té doby, než si účastníci nevymění nový relační klíč *K'ab*. Ten je zaslán *Alici* zpátky ve čtvrtém kroku spolu s nově vygenerovaným nonce *N'b*, které slouží jako sekvenční číslo pro zabezpečení další komunikace.

**Známé útoky: Útok přehráním**

Jelikož ve čtvrtém kroku *B → A*: {*K'ab, N'b*}*Kab* je zasílána *Alici* zpráva neobsahující žádný prvek zaručující čerstvost, může útočník použít **útok přehráním.** Útočník tak může v další relaci podvrhnout *Bobovi* zkompromitovaný klíč *K'ab*, který získal ze starší odposlechnuté komunikace.

1. *A → B: A, {Na}Kab*
2. *B → A: {Na+1, Nb}Kab*
3. *A → B: {Nb+1}Kab*
4. *B → A: {K'ab, N'b}Kab*
5. *A → B : A, { Ma}Kab*
6. *B → A : { Ma+1, Mb}Kab*
7. *A → B :{ Mb+1}Kab*
8. *B → U(A) :{K''ab, M'b}Kab*
9. *U(B) → A :{K'ab, N'b}Kab*

## 4.2 Modifikovaný Andrew Secure RPC

**Autor: Michael Burrows, Martin Abadi, Roger Needham**

**Publikace: BURROWS, M. and ABADI, M. and NEEDHAM, R. A Logic of Authentication. Technical Report 39, *Digital Systems Research Center*, February 1989, p. 1-19**

**Kryptografie: Symetrická**

**Syntaktický zápis komunikace:**

1. *A → B: A, {Na}Kab*
2. *B → A: {Na+1, Nb}Kab*
3. *A → B: {Nb+1}Kab*
4. *B → A: {K'ab, N'b, Na}Kab*

*A, B* : účastníci komunikace (*Alice* a *Bob*)

*Na, Nb, N'b* : nonce

*Kab, K'ab*: klíče

**Popis:**

Tato vylepšená verze původního Andrews Secure RPC protokolu opravuje chybu, která umožňovala získat útočníkovi relační klíč. Jak je vidět, do kroku 4 bylo přidáno nonce *Na* znemožňující nyní použití útoku přehráním.

**Známé útoky:**

Žádné.

## 4.3 Denning – Sacco shared key

**Autor: Dorothy E. Denning a Giovanni Maria Sacco**

**Publikace: DENNING, D. E. and SACCO, G. M. Timestamps in Key Distribution Protocols. *Communication of the ACM*, 1981, p. 533-535.**

**Kryptografie: Symetrická, s použitím autentizačního serveru a časových razítek**

**Syntaktický zápis komunikace:**

1. $A \rightarrow S: A, B$
2. $S \rightarrow A: \{B, Kab, T, \{Kab, A, T\}Kbs\}Kas$
3. $A \rightarrow B: \{Kab, A, T\}Kbs$

*A, B, S*: Účastníci komunikace (*Alice*, *Bob*, *Server*)

*Kab*, *Kbs*: Klíče

*T*: Časové razítko

**Popis:**

Denning – Sacco vznikl na bázi Needham – Schroeder protokolu [51], ale opravuje jeho chyby ohledně čerstvosti zpráv. Tím, že je do zpráv zavedeno časové razítko, se znemožňuje použití útoku přehráním. Protokol distribuuje symetrický sdílený klíč s pomocí důvěryhodného serveru a vzájemné autentizace.

**Známé útoky: Násobný útok**[5]

Útok publikovaný G. Lowem [15] využívá neschopnosti protokolu vyrovnat se s vícenásobným přístupem. Do kroku 3 se komunikace odehrává normálně:

1. *A → S: A, B*
2. *S → A: {B, Kab, T, {Kab, A, T}Kbs}Kas*
3. *A → B: {Kab, A, T}Kbs*

nicméně ve 4. kroku je *Bobovi* přehrána zpráva z kroku 3. Ten si myslí, že se *Alice* snaží navázat druhé spojení a přijme tuto zprávu:

4. *U(A) → B: {Kab, A, T}Kbs*

*U(A)* : Útočník vydávající se za *Alici*

## 4.4 Modifikovaný Denning – Sacco shared key

**Autor: Gavin Lowe**

**Publikace: LOWE, G. A Family of Attacks upon Authentication Protocols. Technical Report 5, *Department of Mathematics and Computer Science*, *University of Leicester*, 1997, p. 1-11.**

**Kryptografie: Symetrická, s použitím autentizačního serveru a časových razítek**

**Syntaktický zápis komunikace:**

1. *A → S: A, B*
2. *S → A: {B, Kab, T, {Kab, A, T}Kbs}Kas*
3. *A → B: {Kab, A, T}Kbs*
4. *B → A: {Nb}Kab*
5. *A → B: {Nb+1}Kab*

**Popis:**

Lowe upravil původní protokol [16] na základě jím zjištěné chyby. Aby nebylo možné podvrhnout zprávu v kroku 4 jejím zopakováním, je do protokolu implementováno nonce *Nb*. To ověří schopnost *Alice* přečíst šifrovanou zprávu a tím i její totožnost.

**Známé útoky:**

Žádné.

[5] Multiplicity attack – autor G. Lowe. Útočník je schopen opakováním zprávy přesvědčit protokol, že stejný účastník žádá o navázání více relací a tím zfalšovat svoji identitu.

## 4.5 Kao Chow Authentication verze 1

**Autor: I Long Kao a Randy Chow**

**Publikace: KAO, I. L. and CHOW, R. An Efficient and Secure Authentication Protocol Using Uncertified Keys. *Operating Systems Review*, 1995, vol. 29, no. 3, p. 14-21.**

**Kryptografie: Symetrická, s použitím autentizačního serveru**

**Syntaktický zápis komunikace:**

1. *A → S: A, B, Na*
2. *S → B: {A, B, Na, Kab}Kas, {A, B, Na, Kab}Kbs*
3. *B → A: {A, B, Na, Kab}Kas, {Na}Kab, Nb*
4. *A → B: {Nb}Kab*

*A*, *B*, *S*: účastníci komunikace (*Alice*, *Bob*, *Server*)

*Na*, *Nb*: nonce

*Kab*, *Kas*, *Kbs*: klíče

**Popis:**

Protokol používá symetrické kryptografie a důvěryhodného serveru k výměně nového relačního klíče mezi účastníky. Klíč *Kas* je na začátku známý pouze *Alici* a *Serveru*, respektive klíč *Kbs Bobovi* a *Serveru*. Druhá strana je tedy po přijetí zprávy schopna rozluštit pouze svojí část, ze které získá relační klíč *Kab*. Ten je pak předán iniciátorovi akce spolu s nově vygenerovaným *Nb*, sloužícím k potvrzení čerstvosti autentizace. Protokol musí garantovat utajitelnost relačního klíče *Kab* v každém kroku, klíč musí být známý pouze účastníkům a musí pocházet ze současné relace od *Serveru*.

**Známé útoky: Denning – Sacco útok**

Pokud uvažujeme, že útočník byl schopen odposlechnout starší komunikaci a odhalit relační klíč *Kab*, může pak použít tento zkompromitovaný klíč k přesvědčení *Boba*, že stále jedná s *Alicí* [16]:

1. *A → U(B): {Kab, A}Kbs*

Útočník získal klíč *Kab*:

2. *U(A) → B: {Kab, A}Kbs*
3. *B → U(A): {Nb}Kab*
4. *U(A) → B: {Nb+1}Kab*

*U(A), U(B):* Útočník vydávající se za *Alici*, *Boba*

## 4.6 Kao Chow Authentication verze 2

**Autor: I Long Kao a Randy Chow**

**Publikace: KAO, I. L. and CHOW, R. An Efficient and Secure Authentication Protocol Using Uncertified Keys. *Operating Systems Review*, 1995, vol. 29, no. 3, p. 14-21.**

**Kryptografie: Symetrická, s použitím autentizačního serveru**

**Syntaktický zápis komunikace:**

1. *A → S: A, B, Na*
2. *S → B: {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs*
3. *B → A: B, {A, B, Na, Kab, Kt}Kas, {Na, Kab}Kt, Nb*
4. *A → B: {Nb, Kab}Kt*

*A*, *B*, *S*: účastníci komunikace (*Alice*, *Bob*, *Server*)

*Na*, *Nb*: nonce

*Kab*, *Kas*, *Kbs*, *Kt*: klíče

**Popis:**

Do protokolu byl přidán nový relační klíč *Kt* s cílem zabránit útoku přehráním, který byl možný v první verzi.

**Známé útoky:**

Žádné.

## 4.7 Kao Chow Authentication verze 3

**Autor: I Long Kao a Randy Chow**

**Publikace: KAO, I. L. and CHOW, R. An Efficient and Secure Authentication Protocol Using Uncertified Keys. *Operating Systems Review*, 1995, vol. 29, no. 3, p. 14-21.**

**Kryptografie: Symetrická, s použitím autentizačního serveru**

**Syntaktický zápis komunikace:**

1. *A → S: A, B, Na*
2. *S → B: {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs*
3. *B → A: {A, B, Na, Kab, Kt}Kas, {Na, Kab}Kt, Nb, {A, B, Ta, Kab}Kbs*
4. *A → B: {Nb, Kab}Kt, {A, B, Ta, Kab}Kbs*

*A, B, S*: účastníci komunikace (*Alice*, *Bob*, *Server*)

*Na, Nb*: nonce

*Kab, Kas, Kbs, Kt*: klíče

*Ta*: časové razítko

**Popis:**

Rozšiřuje verzi dva o takzvaný tiket[6] obsahující jak relační klíč *Kab*, tak časové razítko.

**Známé útoky:**

Žádné.

## 4.8 Kerberos verze 5

**Autor: B. Clifford Neuman a Theodore Ts'o**

**Publikace: NEUMAN, B. C. and TS'O, T. Kerberos: An Authentication Service for Computer Networks. Technical Report, ISI/RS-94-399, USC/ISI, 1994, p. 1-8.**

**Kryptografie: Symetrická, s použitím autentizačního serveru, tiketů a časových razítek**

**Syntaktický zápis komunikace:**

1. *K → A: U, G, L1, N1*
2. *A → K: U, {U, K, G, Kkg, T1start, T1konec}Kag, {G, Kkg, T1start, T1konec}Ku*
3. *K → G: S, L2, N2, {U, K, G, Kkg, T1start, T1konec}Kag, {K, T1}Kkg*
4. *G → K: U, {U, K, S, Kks, T2start, T2konec}Kgs, {S, Kks, T2start, T2konec, N2}Kkg*
5. *K → S: {U, K, S, Kks, T2start, T2konec}Kgs, {K, T2}Kks*
6. *S → K: {T2}Kks*

*A, G, K, S, U*: účastníci (*Alice*, *Garant*, *Klient*, *Server*, *Uživatel*)

*N1, N2*: Nonce

T1(start/konec), T2(start/konec): Časové značky

*Kag, Kgs, Kkg, Kks, Ku*: klíče

*L1, L2*: atributy určující životnost spojení

[6] *Tiket* je zpráva, jejíž obsah není jeden z agentů schopen přečíst, protože je šifrována pro něho neznámým klíčem. Většinou se objevuje v protokolech využívajících serveru jako arbitra pro relační klíče či autentizaci.

**Popis:**

*Klient* pošle žádost obsahující *Uživatelovy* přihlašovací údaje *Alici*, na jejímž konci obdrží dočasný klíč pro komunikaci se *Serverem*. *Alice*, která zde vystupuje jako důvěryhodný autentizační server, zašle zpátky tiket s klíčem *Kkg* pro *Garanta* a zprávu zašifrovanou *Ku* obsahující ten samý klíč určený pro použití *Uživatelem*. *Garant* v roli potvrzovacího serveru potvrdí pomocí informací v tiketu oprávněnost *Uživatele* pro komunikaci se *Serverem*. V předposledním a posledním kroku obě strany potvrdí výměnu relačního klíče, čímž byl proces autentizace završen.

**Známé útoky:**

Žádné.

## 4.9 Neumann Stubblebine

**Autor: B. Clifford Neumann a Stuart G. Stubblebine**

**Publikace: NEUMANN, B. C. and STUBBLEBINE, S. G. A Note on the Use of Timestamps as Nonces. *Operating Systems Review*, April 1993, vol. 27, no. 2, p. 10-14.**

**Kryptografie: Symetrická, s použitím autentizačního serveru**

**Syntaktický zápis komunikace:**

1. *A → B: A, Na*
2. *B → S: B, {A, Na, Tb}Kbs, Nb*
3. *S → A: {B, Na, Kab, Tb}Kas, {A, Kab, Tb}Kbs, Nb*
4. *A → B: {A, Kab, Tb}Kbs, {Nb}Kab*
5. *A → B: N'a, {A, Kab, Tb}Kbs*
6. *B → A: N'b, {N'a}Kab*
7. *A → B: {N'b}Kab*

*A*, *B*, *S*: účastníci (*Alice*, *Bob*, *Server*)

*Na*, *Nb*, *N'a*, *N'b*: nonce

*Kas*, *Kab*, *Kbs*: klíče

*Tb*: čas

**Popis:**

V první části protokolu (1-4) probíhá ověřování účastníků komunikace přes důvěryhodný server. *Alice* zahajuje autentizaci vysláním nešifrované zprávy *Bobovi*. Ten se obrátí na *Server* s žádostí o zaslání přihlašovacích údajů *Alici*, jejichž platnost je stanovena hodnotou *Tb*. *Server* ve třetím kroku zašle *Alici* spolu s tiketem ***{A, Kab, Tb}Kbs*** pověření používat relační klíč *Kab* s časem expirace *Tb*. *Alice* kontrolou *Na* ověří, že skutečně komunikuje s *Bobem*, a potvrdí svoji identitu zasláním nonce *Nb* šifrované relačním klíčem.

V dalších krocích (5-7) může *Alice* kdykoli po dobu platnosti určené *Tb* znovu kontaktovat *Boba* jen s použitím tiketu, tedy bez nutnosti autentizace přes *Server*.

**Známé útoky:**

1. *I(A) → B: A, Na*
2. *B → I(S): B, {A, Na, Tb}Kbs, Nb*
3. *vynecháno*
4. *I(A) → B: {A, Na, Tb}Kbs, {Nb}Na*
5. *I(A) → B: {A, Na, Tb}Kbs*
6. *B → I(A): N'b, {N'a}Na*
7. *I(A) → B: {N'b}Na*

V tomto útoku se snaží útočník přesvědčit *Boba*, aby přijal jím navrhované nonce *Na* jako relační klíč. Tato chyba byla odstraněna T. Hwangem a dalšími v modifikovaném Neumann Stubblebine protokolu [36].

**Útok přehráním**

5. *I(A) → B: N'a, {A, Kab, Tb}Kbs*
6. *B → I(A): N'b, {N'a}Kab*

***5b).*** *I(A) → B: N'b, {A, Kab, Tb}Kbs*

***6b)***. *B → I(A): N'b, {N'b}Kab*

7. *I(A) → B: {N'b}Kab*

Útočník přehraje v kroku 5b tiket spolu s nonce *N'b* získaným v kroku 6. Jelikož se jedná o protokol s opakovanou autentizací, *Bob* tento pokus považuje za právoplatný.

**Útok silou**

Popsán Christopherem Weidenbachem [17]. Útočník může získat dostatek šifer, aby byl schopen prolomit klíč *Kbs*.

***2.*** *I(B) → S: B, {A, K0ab, Tb}Kbs, Nb*

***3.*** *S → I(A): {B, Na, K1ab, Tb}Kas, {A, K1ab, Tb}Kbs, Nb*

***2b.*** *I(B) → S: B, {A, K1ab, Tb}Kbs, Nb*

***3b.*** *S → I(A): {B, Na, K2ab, Tb}Kas, {A, K2ab, Tb}Kbs, Nb*

## 4.10 Needham - Schroeder Public key

**Autor: Roger Needham a Michael Schroeder**

**Publikace: NEEDHAM, R. and SCHROEDER, M. Using Encryption for Authentification in Large Networks of Computers. *Communications of the ACM*, December 1978, vol. 21, no. 12, p. 993 -999.**

**Kryptografie: Asymetrická, s použitím serveru pro distribuci klíčů**

**Syntaktický zápis komunikace:**

***1.*** *A → S: A, B*

***2.*** *S → A: {KPb, B}KSs*

***3.*** *A → B: {Na, A}KPb*

***4.*** *B → S: B, A*

***5.*** *S → B: {KPa, A}KSs*

***6.*** *B → A: {Na, Nb}KPa*

***7.*** *A → B: {Nb}KPb*

*A, B, S*: účastníci (*Alice*, *Bob*, *Server*)

*Na, Nb*: nonce

*KPa, KPb, KPs*: veřejné klíče

*KSa, KSb, KSs*: soukromé klíče

**Popis:**

Protokol pracuje na základě spolupráce účastníků se serverem pro distribuci klíčů. Na začátku vyšle *Alice* žádost o *Bobův* veřejný klíč. *Server* odpoví zprávou s klíčem zašifrovanou svým soukromým klíčem. *Alice* následně kontaktuje *Boba*. Ten se také vyptá *Serveru* na veřejný klíč *Alice*. Poté, co dostane odpověď, zašle *Alici* zprávu obsahující nově vygenerované nonce *Nb* a nonce *Na* obdržené od *Alice* v třetím kroku. Po potvrzení by *Alice* a *Bob* měli každý znát identitu toho druhého.

**Známé útoky: Man in the middle**

1. *A → U: {Na, A}KPu*
2. *U(A) → B: {Na, A}KPb*
3. *B → U(A): {Na, Nb}KPa*
4. *U → A: {Na, Nb}KPa*
5. *A → U: {Nb}KPu*
6. *U(A) → B: {Nb}KPb*

Útočník přesvědčí *Alici*, aby s ním navázala spojení. Zároveň ale paralelně komunikuje s *Bobem*, kterého přesvědčí, že je *Alice*. Předpokládá se, že útočník vlastní klíče *KPu* a *KSu*, a každý účastník zná veřejné klíče *KPa, Kpb, KPu***.** *Alice* zašle zprávu útočníkovi, který jí rozšifruje svým soukromým klíčem *Ksu* a přepošle jí *Bobovi*. Ten zašle *Nb*, které sice útočník nepřečte, ale po přeposlání jej *Alice* vrátí zašifrované klíčem *KPu*. Útočníkovi pak stačí jen jej znovu zašifrovat a zaslat *Bobovi*, který je tímto přesvědčen, že komunikuje s *Alicí*.

## 4.11 Woo and Lam Pi

**Autor: T. Y. C. Woo and S. S. Lam**

**Publikace: WOO, T. Y. C. and LAM, S. S. A Lesson on Authentication Protocol Design. *Operating Systems Review*, vol.28, 1994, p. 24-37**.

**Kryptografie: Symetrická, s použitím autentizačního serveru**

**Syntaktický zápis komunikace:**

1. *A → B: A*
2. *B → A: Nb*
3. *A → B: {Nb}Kas*
4. *B → S: {A, {Nb}Kas}Kbs*
5. *S → B: {A, Nb}Kbs*

*A, B, S*: účastnící (*Alice*, *Bob*, *Server*)
*Nb*: nonce
*Kas, Kbs*: klíče

**Popis:**

Woo Lam Pi protokol pracuje jako jednosměrný autentizační protokol. *Alice* se nazývá *iniciátor* protokolu a Bob je *respondér*. V prvním kroku se *Alice* ohlásí *Bobovi*, který jí odpoví zasláním nonce *Nb*. *Alice* pošle nonce zašifrované zpět, ale jelikož *Bob* není schopen přečíst tuto zprávu, zašle ji autentizačnímu *Serveru*. Ten znajíc klíč *Kas* přetransformuje zprávu do nové zprávy šifrované klíčem *Kbs*. Tu už je *Bob* schopen přečíst a autentizace je dokončena.

**Známé útoky:**

1. *U(A) → B: A*
2. *U → B: U*
3. *B → U(A): Nb*
4. *B → U: Nb'*
5. *U(A) → B: X*
6. *U → B: {Nb}Kus*
7. *B → S: {A, X}Kbs*
8. *B → S: {U, {Nb}Kus}Kbs*
9. *S → B: {Nb}kbs*

Tento útok je založen na tom, že se útočník pokusí navázat dvě souběžná spojení s *Bobem*. V jednom se maskuje jako *Alice* a v druhém vystupuje sám za sebe. Získá tím nonce *Nb* a *Nb'*, přičemž zašifruje *Nb* svým vlastním klíčem *Kus*. Když pak autentizační *Server* navrátí tuto zprávu zpátky *Bobovi*, ten je přesvědčen, že je touto zprávou autentizována *Alice* [18].

## 4.12 Porovnání vlastností bezpečnostních protokolů

Tato tabulka čerpá z informací uvedených v online databázi bezpečnostních protokolů SPORE [21].

| Název protokolu | Kryptografie | Nonce | Čas. razítka | Autent. server | Tikety | Známé útoky |
|---|---|---|---|---|---|---|
| Adrew Secure RPC | Sym. | Ano | Ne | Ne | Ne | Útok přehráním |
| Mod. Andrew Secure RPC | Sym | Ano | Ne | Ne | Ne | X |
| Denning – Sacco Shared Key | Sym. | Ne | Ano | Ano | Ne | Násobný útok |
| Mod. Denning – Sacco Shared Key | Sym. | Ano | Ano | Ano | Ne | X |
| Kao Chow Authentication v. 1 | Sym. | Ano | Ne | Ano | Ne | Útok Denning – Saco |
| Kao Chow Authentication v. 2 | Sym. | Ano | Ne | Ano | Ne | X |
| Kao Chow Authentication v. 3 | Sym. | Ano | Ano | Ano | Ano | X |
| Kerberos v. 5 | Sym. | Ano | Ano | Ano | Ano | X |
| Neumann Stubblebine | Sym. | Ano | Ano | Ano | Ano | Útok silou, útok přehráním |
| Needham – Schroeder Public Key | Asym. | Ano | Ne | Ano | Ne | X |
| Woo and Lam Pi | Sym. | Ano | Ne | Ano | Ne | X |

**Tabulka 1. Srovnání uvedených bezpečnostních protokolů**

# 5 Nástroje pro analýzu bezpečnostních protokolů

## 5.1 Isabelle

**Autoři: Larry Paulson, Tobias Nipkow**

**Domovská stránka: http://www.cl.cam.ac.uk/research/hvg/isabelle/**

**Verze/Datum vydání: 2009-1/2009**

**Distribuce: Linux/Windows – cygwin/Mac OS**

**Licence: Volně ke stažení**

**Používaný systém verifikace: Dokazování teorémů pomocí logiky vyššího řádu**

### 5.1.1 Instalace

Uvedeme si zde instalaci pro systém Linux. Hlavní distribuce se skládá ze tří částí:

- Zdrojový kód a dokumentace – Isabelle2009-1.tar.gz
- Poly/ML – polyml-5.3.0.tar.gz
- Proof General – ProogGeneral-3.7.1.1.tar.gz

Dále je možné si stáhnout přídavné části:

- E prover – E-1.0-004.tar.gz
- SPASS – spass-3.0.tar.gz
- Kodkodi – kodkodi-1.2.7.tar.gz
- Haskabelle – Haskabelle2009-1.tar.gz

Po stažení a rozbalení všech potřebných částí je nutné zkontrolovat, jestli používáme správnou verzi Emacs (doporučovaná verze je Emacs 22). Pokud se nedaří spustit Isabelle i s vestavěným ProofGeneral, můžeme program spustit bez ProofGeneral v čistě textovém režimu příkazem – */instalačníadresář/Isabelle/bin/isabelle tty.*

### 5.1.2 Popis

Isabelle je nástroj používaný k zápisu matematických formulí pomocí formálního jazyka a jejich následnému ověřování s využitím logického kalkulu. Specializuje se na interaktivní dokazování matematických theorémů pomocí logiky vyššího řádu [22]. Také se používá ke kontrole správností návrhu hardware a software nebo k dokazování vlastností programovacích jazyků. Je dostupná

rozsáhlá knihovna obsahující teorie pro formální matematiku[7] (základní teorii čísel, matematickou analýzu, matematickou algebru).

Isabelle je soubor několika specializovaných modulů, jako nejdůležitější bych zmínil Isabelle/HOL a Isabelle/Isar.

### 5.1.3 Isabelle/HOL

Následující text čerpá z [22].

Tento nástroj se specializuje na dokazování theorémů pomocí logiky vyššího řádu (HOL – High Order Logic). Typy používané v HOL se podobají typům z funkcionálních programovacích jazyků, je možné převést některé proměnné přímo do kódu SML[8], OCaml[9], Haskell[10].

**Teorie**

Základem práce v Isabelle/HOL jsou teorie. Teorie jsou složené z typů, formulí, proměnných a jejich obecný formát lze zapsat následovně:

**theory T**

**imports $B_1$ … $B_n$**

**begin**

*deklarace, definice a důkazy*

**end**

kde $B_1$ … $B_n$ jsou již existující, nadřazené rodičovské teorie. Výsledný soubor s teorií je uložen jako **T.thy.**

**Typy, termy a formule**

Jsou to základní složky teorií popsané pomocí logiky vyššího řádu.

**Typy –** Základní **typy:** *bool* – nabývá hodnoty pravda, nepravda.

*nat* – vyjadřuje typ přirozených čísel

Typové **konstruktory:** *list* – seznam

*set* – sada

Příkladem je typ seznam, jehož prvky jsou přirozená čísla *(nat)list.*

Typ **funkce***:* značený $\Rightarrow$ , funkce jsou definované jako totální fuknce, takže

$\tau_1 \Rightarrow \tau_2 \Rightarrow \tau_3$ znamená $\tau_1 \Rightarrow (\tau_2 \Rightarrow \tau_3)$.

Typ **proměnné:** značíme **‘a, ‘b** atd.

---

[7] Dostupné na http://isabelle.in.tum.de/library/HOL/ [2010-05-18].

[8] Standard ML – zmodernizovaný funkcionální programovací jazyk ML.

[9] OCaml – rozšíření jazyka Caml o objektově orientované konstrukce.

[10] Haskell – standardizovaný jazyk pro funkcionální programování.

**Termy –** jsou formované aplikováním funkcí na různé argumenty. Př.: Je-li $f$ funkcí $\tau_1 \Rightarrow \tau_2$ a $t$ je term typu $\tau_1$, pak $f$ je term typu $\tau_2$. Termy mohou také obsahovat *λ-abstrakce*, takže funkce zapsaná *λx. x+1* vrací hodnotu *x+1* pro argument *x*.

**Formule –** jsou to termy typu *bool*, nabývají logické hodnoty pravda/nepravda a používají klasické logické operátory negace $\neg$, konjukce $\wedge$, disjunkce $\vee$ a implikace $\longrightarrow$. Pro vyjádření ekvivalence se používá zápis $\tau_1 = \tau_2$.

### 5.1.4 Isabelle/Isar

Text je čerpán z referenčního manuálu [23].

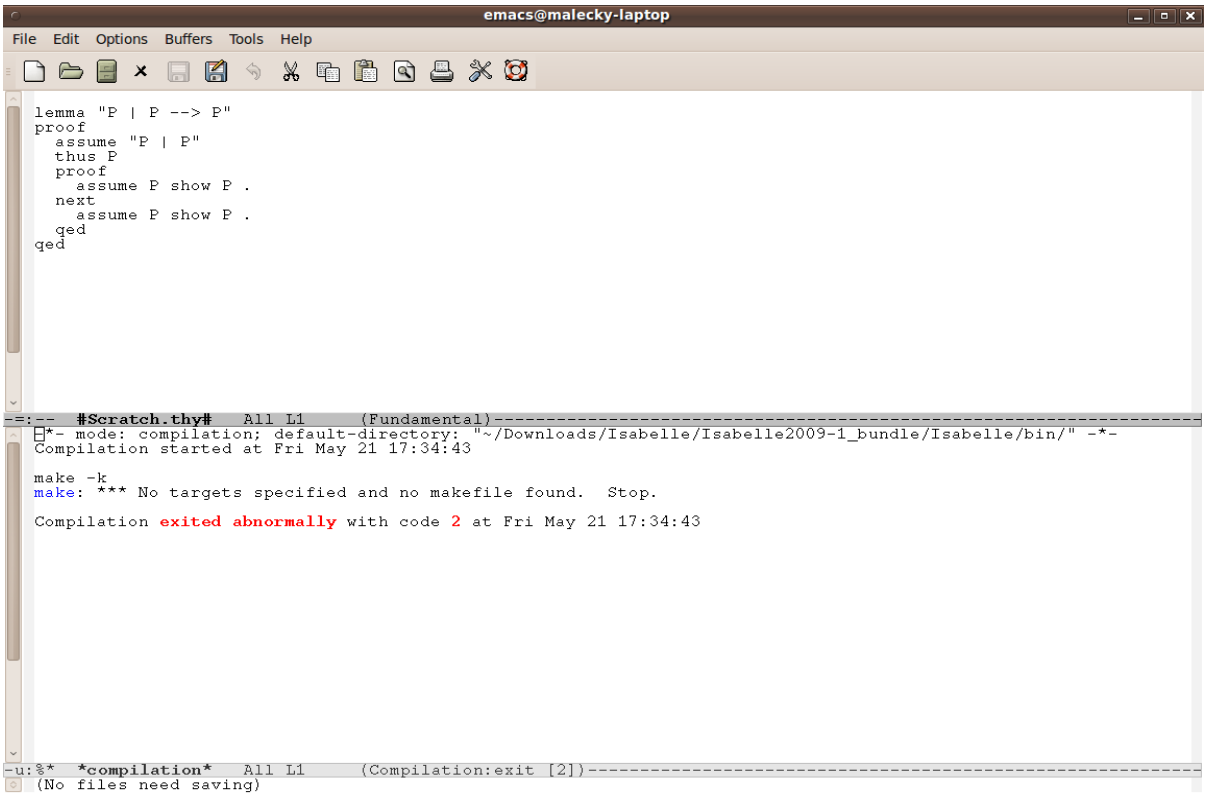
Oproti Isabelle/HOL tento modul používá vlastní interpretované jazykové prostředí, speciálně navržené pro návrh teorií a důkazů. Podporuje vývojové grafy či řízené transakce s neomezeným počtem zpětných kroků. Jazyk Isar přináší konceptuálně rozdílný pohled na automatickou kontrolu důkazů, návrh a popis matematických důkazů se v jazyce Isar stává pro uživatele jednodušším a čitelnějším. Formování důkazů a teorií je tak dostupnější pro širší okruh uživatelů než je tomu tak u klasických ML programovacích jazyků.

Jazyk Isar vychází z tzv. Pure logiky [24] přejímající její výrazy pro popis entit (popis návrhů, faktů a cílů). Všechny úsudky jsou organizovány pod *λ*-kalkulem Pure logiky, není proto nutné zavádět nový kalkulus. Isar je minimalistický, přibližně polovina jazyka je vyjádřena pomocí matematických primitiv.

Základním elementem jazyka jsou **teorémy:**

*theory-stmt* = **theorem** *statement proof* | **definition** . . . | . . .
*proof* = *prfx** **proof** *method stmt** **qed** *method*
*prfx* = **using** *facts*
| **unfolding** *facts*
*stmt* = **{** *stmt** **}**
| **next**
| **note** *name* = *facts*
| **let** *term* = *term*
| **fix** *var+*
| **assume** ⟪*inference*⟫ *name: props*
| **then** *goal*
*goal* = **have** *name: props proof*
| **show** *name: props proof*

Více informací o elementech v uvedeném teorému a dalších prvcích jazyka Isar lze nalézt v referenčním manuálu [23].




**Obrázek 1 - Grafické uživatelské rozhraní ProofGeneral**

### 5.1.5 Porovnání nástroje Isabelle

Nástroj Isabelle a jeho moduly se zdají být schopným prostředkem pro dokazování matematických důkazů a teorií. Díky svému širokému záběru je také možné využít Isabelle pro analýzu bezpečnostních protokolů, jak ukázal Larry Paulson při ověřování správnosti návrhu protokolu Yahalom [31]. Nicméně modelování bezpečnostních protokolů pomocí Isabelle není jednoduchou záležitostí, což omezuje její potenciálni využití v této oblasti.

**Výhody:**

- Velice schopný nástroj k dokazování teorémů
- Široký záběr témat pro konstruování důkazů
- Existence mnoha rozšíření a přídavků
- Dostupná knihovna již vytvořených teorií

**Nevýhody:**

- Není primárně určen k verifikaci bezpečnostních protokolů
- Modelování bezpečnostních protokolů není triviální
- Nutnost znalosti funkcionálního programování a matematické logiky pro správnou kontrukci důkazů
- Nesnadná orientace v grafickém uživatelském rozhraní ProofGeneral

## 5.2 AVISPA

**Autoři: Alessandro Armando, Michael Rusinowitch, David Basin, David Von Oheimb a další**
**Domovská stránka: http://avispa-project.org/**
**Verze/Datum vydání: verze 1.1 / 30.6.2006**
**Distribuce: Linux**
**Licence: Akademická volně ke stažení**
**Používaný systém verifikace: Specifikace protokolu v jazyce HLPSL a analýza jedním z modulů**

### 5.2.1 Instalace

AVISPA je dostupná pouze v distribuci pro Linux. Při instalaci je nutné nastavit proměnnou AVISPA_PACKAGE, aby odkazovala na absolutní cestu končící v AVISPA adresáři.

### 5.2.2 Popis

AVISPA (Automated Validation of Internet Security Protocols and Applications) je nástroj pro automatické ověřování bezpečnostních protokolů a aplikací [27]. AVISPA používá k formalizaci bezpečnostních protokolů a cílů svůj vlastní jazyk nazývaný High Level Protocol Specification Language (HLPSL). Uživatel specifikuje protokol v jazyce HLPSL a následně spustí skripty v nástroji AVISPA, které tuto specifikaci automaticky přeloží do nízkoúrovňového jazyka IF (Intermediate Format). Z tohoto jazyka pak čerpají další moduly.

AVISPA obsahuje čtyři následující moduly:

1. **On-the-fly Model Checker (OFMC) –** tento modul je schopen efektivně detekovat možné útoky na protokol či dokázat, zda je protokol správně navržen. Modul vytvoří na základě požadavků protokolu nekonečný strom stavů, nicméně analýza poté probíhá na konečném stavovém prostoru pro omezený počet relací, aniž by přitom byl omezen počet útočníkem generovaných zpráv.

2. **CL-based Model Checker (CL-ATSE)** – modul transformuje specifikaci protokolu převedenou do IF formy na soubor omezení, podle kterých je pak možné najít případné útoky. Každý krok protokolu je proveden přidáním/odebráním omezení a každý stav systému je testován ohledně bezpečnostních požadavků. Testovací algoritmus používá omezený počet smyček, takže je testován omezený počet kroků protokolu v každém běhu.

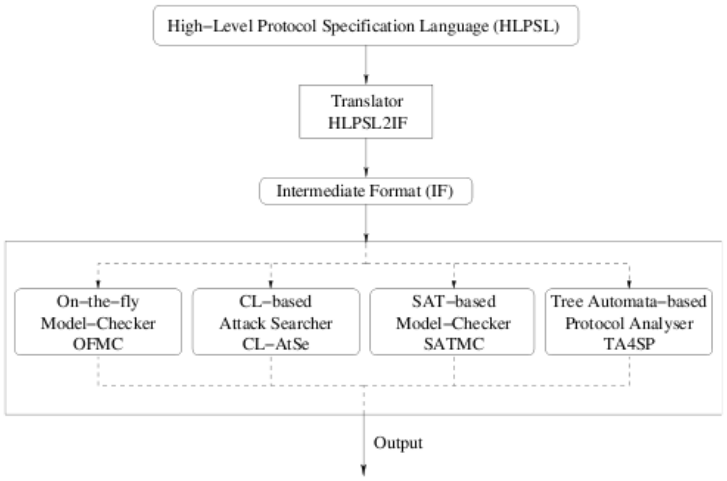
3. **SAT-based Model Checker (SATMC)** – převede specifikaci v IF do symbolické reprezentace výrokovými formulemi. Počáteční stav a množina ostatních stavů pak

představují narušení bezpečnostních vlastností protokolu. Kontrola modelu protokolů na základě redukce do SAT přináší velkou výhodu flexibilního a efektivního nástroje pro verifikaci bezpečnostních protokolů.

4. **Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP)** – tento nástroj aproximuje potenciální vědomosti útočníka prostřednictvím přepisování stromu regulárních jazyků v kontextu neomezeného počtu relací. Tímto lze ukázat, zda je protokol bezpečný pro neomezený počet relací (over-approximation) nebo zda je vadný (under-approximation).

### 5.2.3 Jazyk HLPSL

High Level Protocol Specification Language je expresivní jazyk pro modelování komunikačních a bezpečnostních protokolů[11]. Pomocí tohoto jazyka uživatel komunikuje s nástrojem AVISPA, proto přináší velkou míru abstrakce a obsahuje velkou řadu funkcí, které jsou obvyklé pro bezpečnostní protokoly (model útočníka, šifrovací primitiva atd.). HLPSL je automaticky překládán do nízkoúrovňového jazyka IF pomocí překladače HLPSL2IF.




**Obrázek 2 - Architektura nástroje AVISPA [28]**

HLPSL je založen na temporární logice. Modelování protokolů tedy probíhá způsobem, že se popisují stavy systému a následně se specifikují způsoby, jak se tyto stavy mohou měnit. Změny stavů se popisují *tranzitivními predikáty,* které se vztahují k hodnotám proměnných v současném stavu a hodnotám proměnných v budoucím stavu.

[11] Informace čerpány ze zdroje [28].

Protokoly jsou v jazyce HLPSL rozděleny do *rolí.* **Základní role** popisují akce jednotlivých agentů při běhu protokolu nebo subprotokolu. **Složené role** používají tyto základní role pro modelování celého běhu protokolu nebo jedné relace probíhající mezi více agenty. Pro každou roli jsou definovány *stavové proměnné* popisující současný stav.

#### 5.2.3.1 Proměnné a konstanty

V HLPSL proměnné začínají velkým písmenem a konstanty malým. Každá konstanta či proměnná musí mít svůj jedinečný typ:

- *agent:* tento typ představuje jména účastníků, útočník se podle předpokladu značí **i**.
- *public_key:* tato hodnota představuje agentův veřejný klíč pro asymetrickou kryptografii.
- *symmetric_key:* představuje klíč pro symetrickou kryptografii.
- *text:* tyto hodnoty jsou často užívané jako nonce, pokud se tak stane, je dané hodnotě přiřazen atribut *fresh* naznačující, že následující hodnoty by měly být jedinečné.
- *nat:* typ reprezentující přirozená čísla v kontextech mimo zprávy, většinou se používá pro testy číselné nerovnosti.
- *function:* tento typ představuje funkce na množině zpráv.
- *bool:* typ boolean.

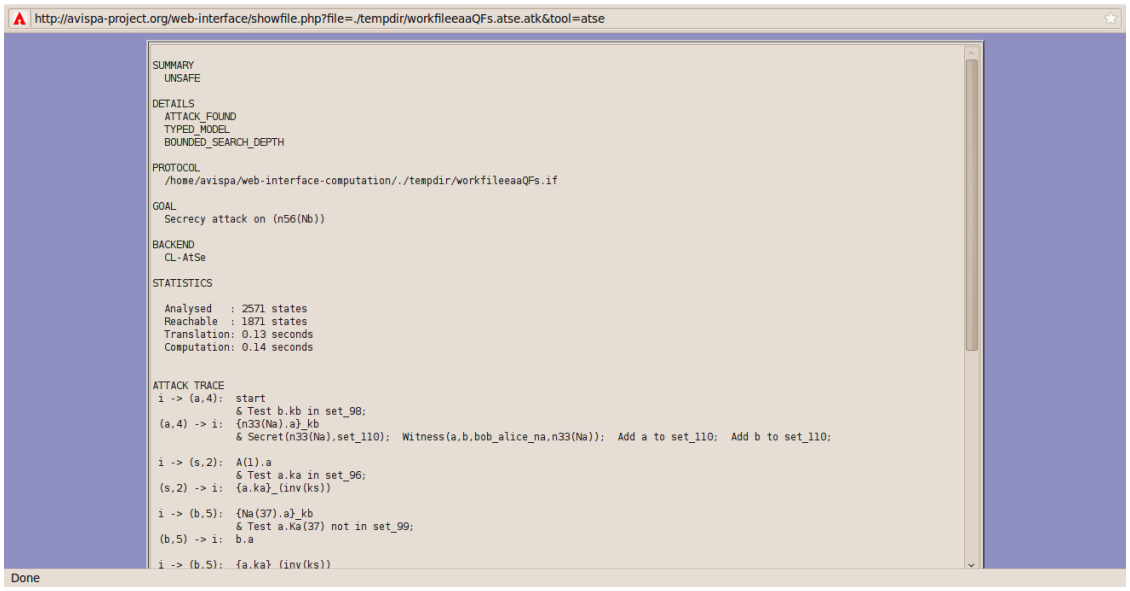
Další informace lze najít v HLPSL manuálu [28].

### 5.2.4 AVISPA online

Na adrese[12] lze nalézt kompletní AVISPA nástroj k okamžitému použití. Uživatelé si tu mohou vyzkoušet analýzu některých ukázkových protokolů, nebo je také možné vložit vlastní protokol v jazyce HLPSL.

[12] *Avispa Automated Validation of Internet Security Protocols and Applications*. Dostupné z WWW: http://avispa-project.org/web-interface/index.php [cit. 2009-12-28].


**Obrázek 3 - Úvodní stránka AVISPA Web Tool**




**Obrázek 4 - Výsledek analýzy v AVISPA Web Tool**

### 5.2.5 SPAN

SPAN je doplněk pro nástroj AVISPA. Je to animátor bezpečnostních protokolů specifikovaných v jazyce HLPSL nebo CAS+ [29]. SPAN interaktivně vytváří schémata sekvenčních zpráv [30], které lze interpretovat jako komunikaci *Alice* a *Boba* v původní HLPSL specifikaci. Také umožňuje monitorovat okamžité hodnoty u účastníků nebo sledovat paralelně více relací v protokolu.

SPAN může běžet v jednom ze tří módů:

1. **Simulace protokolu –** simuluje protokol a vytváří MSC podle specifikace v HLPSL.
2. **Simulace útočníka –** simulace protokolu s aktivním nebo pasivním útočníkem.
3. **Simulace útoku –** automaticky vytváří MSC pro útoky z výstupů z modulů OFMC a CL-ATSE.

### 5.2.6 Porovnání nástroje AVISPA

AVISPA je nástroj, který nám přináší možnost analyzovat bezpečnostní protokoly z více úhlů pohledu a pomocí různých technik. I když nástroj obsahuje více modulů pracujících na rozdílných principech, uživateli stačí specifikovat analyzovaný protokol v jazyce HLPSL, aniž by se musel starat o překlad do jazyka nižší úrovně. Pomocí nástroje je možné analyzovat všechny dostupné bezpečnostní protokoly, potřeba je ale detailní znalost těchto protokolů spolu se zvládnutím jazyka HLPSL. Také výsledky těchto analýz nejsou lehce čitelné pro někoho, kdo není kompletně seznámen s tímto nástrojem.

**Výhody:**

- Schopnost modelovat naprostou většinu bezpečnostních protokolů
- Možnost verifikovat protokol z různých úhlů pomocí rozšiřujících modulů
- Grafické znázornění nalezených útoků pomocí nástroje SPAN
- Možnost rozšíření nástroje přidáním dalšího modulu
- Propracovaný specifikační jazyk HLPSL
- Webová verze AVISPA dostupná odkudkoli

**Nevýhody:**

- Nepřehledné výstupy z analýz
- Nutnost ovládat jazyk HLPSL k bezchybné specifikaci protokolů
- Relativně složité ovládání

## 5.3 Scyther

**Autoři: Cas Cremers**

**Domovská stránka: http://people.inf.ethz.ch/cremersc/scyther/index.html**

**Verze/Datum vydání: 1.0-beta7 / 13.12.2007**

**Distribuce: Windows, Linux, Mac OS**

**Licence: Volně ke stažení**

**Používaná verifikační logika: Stavová prostorová logika**

Následující kapitoly vycházejí především z manuálu pro nástroj Scyther [26].

### 5.3.1 Instalace

Instalace nástroje Scyther je snadná pro každého zkušenějšího uživatele operačního systému. Lze volit mezi distribucemi pro Windows, Linux a Mac OS. Celý nástroj je napsaný v jazyce Python, proto je nutné si nainstalovat potřebné knihovny, jelikož většina operačních systémů neobsahuje Python od začátku. Také je potřeba si doinstalovat program GraphViz, s jehož pomocí jsou vykreslovány grafy v Scytheru. Samotný nástroj se spouští souborem *scyther-gui.py*.

**Požadavky:**

- Python ve verzi alespoň 2.4[13]
- WxPython[14]
- cElementtree[15]
- GraphViz[16]

### 5.3.2 Popis

Scyther je nástroj pro automatickou verifikaci bezpečnostních protokolů, který vyvinul Dr. Cas Cremers, pracující jako odborník na bezpečnost informací pro ETH[17] v Zürichu. Jako vzor sloužil Scytheru nástroj Athena od X. D. Songa, jehož původní myšlenky a nápady rozšiřuje o možnosti analýzy protokolů pracujících na bázi asymetrické kryptografie. Verifikace pracuje s neomezeným množstvím relací a nonce. Je možné přesněji charakterizovat protokoly a tím ověřit celou množinu stavů chování protokolu a také lze ověřovat takzvanou synchronizaci u bezpečnostních protokolů. Spolu s dalšími přídavky nám Scyther dává možnost analyzovat velkou škálu bezpečnostních protokolů uvedenou ve SPORE [21]. Pro verifikaci protokolů Scyther pracuje na předpokladu o takzvané „černé skříňce“. To znamená, že útočník a všichni zúčastnění jsou schopni vidět obsah

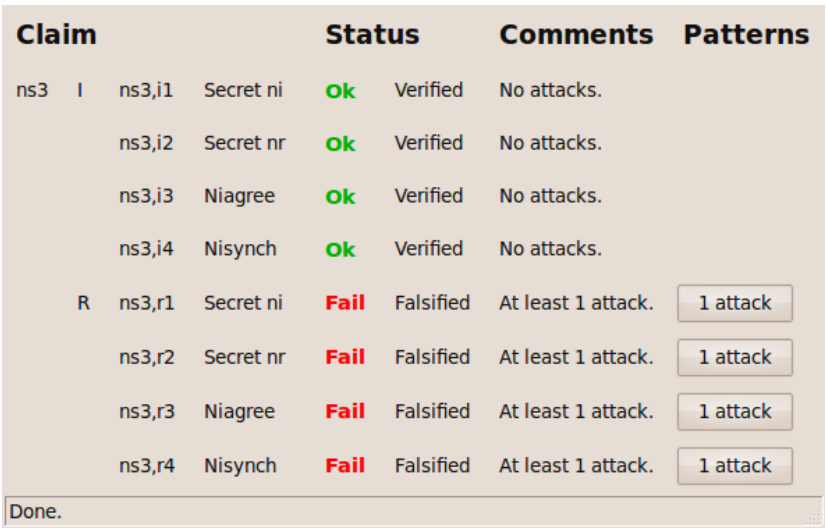
[13] Dostupné z WWW: http://www.python.org/download/ [cit. 2010-04-18].
[14] Dostupné z WWW: http://www.wxpython.org/download.php [cit. 2010-04-18].
[15] Dostupné z WWW: http://effbot.org/zone/celementtree.htm [cit. 2010-04-18].
[16] Dostupné z WWW: http://www.graphviz.org/ [cit. 2010-04-18].
[17] Eidgenössische Technische Hochschule – Švýcarský Federální Technologický Institut.

zašifrované zprávy pouze v případě, vlastní-li příslušný dešifrovací klíč [50]. Dále využívá tzv. Dolev – Yalo bezpečnostního modelu [45], jehož předpoklady jsou útočníkova schopnost kompletně ovládat síťovou infrastrukturu (je schopnen odposlouchávat, modifikovat, zadržovat a přeposílat zprávy posílané v síťovém provozu) a dokonalost šifrovacích algoritmů. Problém je při analýze rozdělen do několika podčástí, které jsou testovány na konečném stavovém prostoru. U některých protokolů lze určit, zda určitý útok nastane ještě před vyčerpáním vymezeného prostoru, u jiných se může stát, že je pro konečný stavový prostor daný problém nerozhodnutelný.

Cas Cremers také neopomněl naprogramovat grafické uživatelské rozhraní pro snazší přehlednost a práci v nástroji Scyther [25]. Stažená verze navíc obsahuje několik ukázkových protokolů uložených s příponou .spdl[18].

### 5.3.3 Grafické uživatelské rozhraní

To je relativně jednoduché. Obsahuje klasické volby pro otevření – uložení zdrojového kódu protokolu, volby pro ověření různých vlastností protokolu, a nesmí chybět nápověda obsahující informace o programu.




**Obrázek 5 - Program Scyther při analýze protokolu**

Práce s programem je vcelku intuitivní. Po vložení zdrojového kódu protokolu a následné verifikaci se nám objeví tabulka s výsledky analýzy. Zde můžeme vidět, zda byly nalezeny nějaké útoky a v jaké fázi. Je tu také možnost zobrazit si nalezený útok v grafické podobě.

[18] Security Protocol Description Language.

### 5.3.4 Programovací jazyk

Veškerá syntaxe tohoto jazyka je čerpána z uživatelského manuálu pro nástroj Scyther [26].
Slouží k zápisu jednotlivých kroků protokolu v nástroji Scyther. Jeho obecné vlastnosti jsou následující:

- K zápisu identifikátorů se standardně používá znaků nebo textových řetězců sestávajících z alfanumerických písmen a znaků ^ nebo -.
- Bílá místa jsou ignorována.
- Komentáře jsou uvozeny znaky // či # pro jednořádkové komentáře; bloky komentovaného textu jsou uvozeny /* a ukončeny */.
- Jazyk je citlivý na velikost použitého písma, lze psát A ≠ a.

#### 5.3.4.1 Termy

Jsou základními částmi programovacího jazyka. Každý identifikátor může být **atomickým termem**. Atomické termy lze také sdružovat do více složitých termů pomocí násobení, šifrování atd.

#### 5.3.4.2 Násobení

Každé dva termy můžou být zkombinovány do násobku termů, například *(x, y)* je zápis pro násobek termů *x* a *y*.

#### 5.3.4.3 Symetrický klíč

Každý term může sloužit jako klíč pro symetrické šifrování. Značení *{Xa}Kab*, zpráva *Xa* je šifrována klíčem *Kab*.

#### 5.3.4.4 Asymetrický klíč

Pro asymetrické klíče se používá dvou funkcí: funkce pro veřejný klíč *const pk: Function* a funkce pro soukromý klíč *secret sk: Function*. Pro asymetrický pár klíčů se používá značení *inversekeys (pk,sk)*, pak lze term zašifrovaný *pk(x)* rozšifrovat pouze párovým klíčem *sk(x)* a opačně.

#### 5.3.4.5 Hashovací funkce

Zapisují se také jako dvě k sobě inverzní funkce. Hashovací funkce *const hash: Function*, inverzní funkce *secret unhash: Function* a párové klíče *inversekeys (hash, unhash)*. Pokud chceme zašifrovat term hashovací funkcí, standardně ji zapíšeme *hash(x)*.

#### 5.3.4.6 Předdefinované typy

*Agent* : Typ užívaný pro agenty.

*Function* : Speciální typ, který se chová jako hashovací funkce, pokud není řečeno jinak. Pokud je dán term *h(x)* a *h* je funkce, pak není možné z ní derivovat *x*. Jako parametr může sloužit více termů.

*Nonce* : Standardní term označující nonce.

*Ticket* : Proměnná typu *Ticket* může být nahrazena jakýmkoliv termem. Tato proměnná je nečitelná pro všechny agenty.

**Bezpečnostní požadavky**

Bezpečnostní požadavky kladené na protokol se definují pomocí *claim*. Každý příkaz *claim* je vhodné opatřit identifikátorem, aby byly jednotlivé požadavky odlišeny. Například pro agenta *A* deklarace *claim_A1(A,Secret, Na)*.

Syntaxe příkazu *claim* je následující: *claim_Identifikátor(Agent, Funkce, Term).*

*Agent* : Identifikátor agenta, pro kterého se daný požadavek bude kontrolovat.

*Funkce* : Využívají se předdefinované typy.

Předdefinované typy: *Secret* – bude kontrolováno, zda uvedený term nebyl zkompromitován při běhu protokolu.

*Empty* – tento požadavek se nekontroluje a používá se spolu s definicí dodatečných parametrů pro verifikaci (backend parameters).

*Reachable* – Scyther kontroluje, zda je požadavek dostupný při běhu protokolu, tj. pokud existuje cesta v protokolu, kterou je možno tohoto požadavku dosáhnout.

Příkaz *claim* může také sloužit pro kontrolu správné synchronizace zpráv a proměnných v protokolu rozšířením základního typu autentizace o tzv. *Non-injective synchronisation* a *Non-injective agreement* [44]. Důležitým faktorem je kontrola těchto požadavků z pohledu lokálního agenta, tzn. agent rozhodne na základě svých poznatků, zda byl protokol vykonán podle očekávání. Proto se v jazyku SPDL tyto požadavky definují k příslušným agentům, např. je lze zapsat jako c*laim_A1(A,Nisynch)* a *claim_A2(A,Niagree)*.

*Non-injective synchronisation* - agent považuje protokol za synchronizovaný, pokud každý běh protokolu pro roli iniciátora A koresponduje s jedinou, unikátní rolí respondéra B. Jednotlivé běhy protokolu jsou od sebe striktně odděleny a jsou prováděny v očekávaném pořadí zasílání a přijímání

zpráv (každému *sent* musí předcházet příslušný *read*). Nemůže se stát, že například agent v běhu 1 přijme zprávu od agenta v běhu 2.

*Non-injective agreement* – je podmnožinou neinjektivní synchronizace. Platí, pokud se iniciátor A dohodl s respondérem B na všech proměnných zaslaných při běhu protokolu. Všechny komunikační události musí nastat před tímto požadavkem. *Non-injective agreement* má slabší váhu než *Non-injective synchronisation*, protože proměnné mohou být přijmuty z jiných běhů protokolu. Nesplnění této podmínky může vést k útokům přehráním zpráv.

### 5.3.5 Pokročilá nastavení v nástroji Scyther

Scyther sice vyniká svou jednoduchostí, ale i tak má uživatel možnost ovlivnit výsledek testování bezpečnostních protokolů pomocí pokročilejších nastavení v záložce Settings. Zde je možné nastavovat položky jako *Maximum number of runs*, *Matching type* a jiné.

Jednotlivé položky lze popsat následovně:

*Maximum number of runs* – udává maximální počet běhů pro jednotlivé role (tzn. instancí protokolu), pro které se bude hledat možný útok. Každý běh je opatřen jedinečným identifikačním číslem a odpovídá jednomu stavu v celkovém stavovém prostoru. Výchozí nastavení pro max. počet běhů je 5.

*Matching type* – *Typed matching* – jeden ze tří modelů pro termy akceptované agenty v testovaném protokolu. Tento model je nejvíce omezující a říká, že agenti jsou schopni kontrolovat typ dat v přijímaných zprávách. Pokud typ dat nesouhlasí, je zpráva odmítnuta (útočník nemůže provést zmatení typů a vydávat např. nonce za relační klíč).

- *Find basic type flaws* – agenti přijímají všechny základní termy, vyjma šifrovaných a párovaných termů. Může dojít k přehození náhodných hodnot (nonce) a šifrovacích klíčů.
- *Find all type flaws* – agenti nedělají rozdíly mezi přijatými termy a náhodnými hodnotami. V tomto modelu mohou být náhodné hodnoty čteny jako šifrované nebo párované termy. To dává útočníkovi možnost vydávat vlastní vygenerované hodnoty za legitimní termy ve zprávách protokolu.

*Maximum number of patterns per claim* – lze nastavit max. počet generovaných vzorů protokolu. Využívá se hlavně při charakterizování

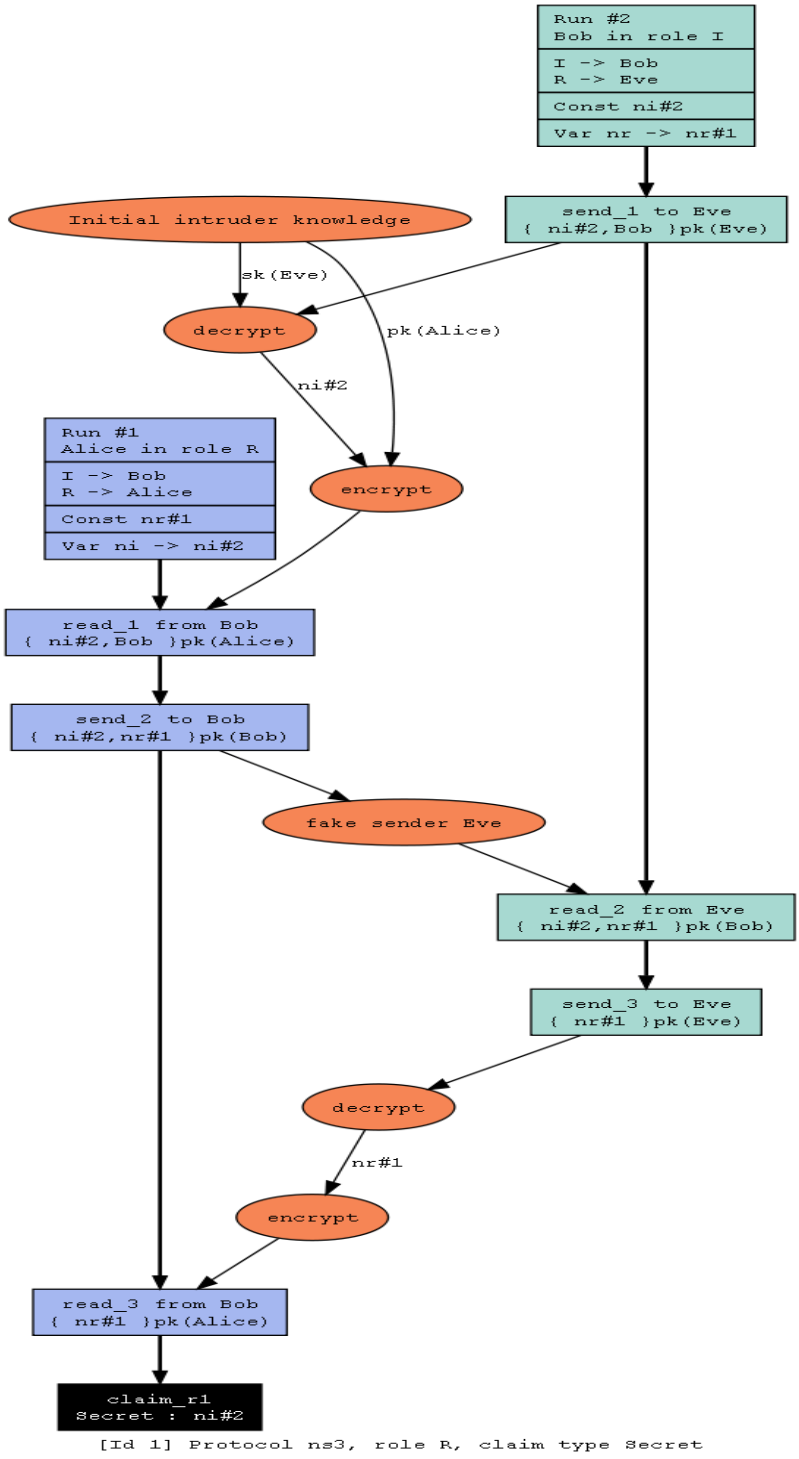
jednotlivých rolí v protokolu (volba verifikace v nástroji Scyther) a pro bezpečnostní požadavky typu *Reachable*.

*Additional backend parameters* – možnost přidání dalších parametrů pro verifikaci, použití s bezpečnostním požadavkem typu *Empty*. Příklady parametrů:
*--max-of-role=N* (nastavení max. počtu běhů),
*--errors=FILE* (přesměrování výstupu do souboru) aj.

*Attack graph font size (in points)* – nastavení velikosti fontů v generovaném grafickém výstupu.

### 5.3.6 Příklad grafického výstupu nástroje Scyther

Pro ukázku byl vybrán jeden z dostupných ukázkových protokolů, konkrétně Needham-Schroeder Symmetric Key protokol.




**Obrázek 6 – Graficky zobrazený útok na protokol**

### 5.3.7 Porovnání nástroje Scyther

Pokud bychom měli porovnávat Scyther s ostatními nástroji, jeho velkou výhodou je rychlost analýzy. Při verifikování jednodušších protokolů rozdíl v rychlosti není samozřejmě tak patrný, nicméně u komplexnějších protokolů už jsou rozdíly v řádu několika sekund. To je nejspíše způsobené jednoduchostí nástroje, Scyther totiž díky svému designu nenabízí mnoho možností jak celkovou analýzu ovlivnit. Hlavním parametrem je počet běhů, kterým lze omezit nekonečný stavový prostor, na kterém bude probíhat analýza.

I přes svoji jednoduchost Scyther dovoluje analyzovat všechny bezpečnostní protokoly uvedené ve SPORE [21] a také některé průmyslové IETF protokoly.

**Výhody:**

- Nenáročná specifikace protokolů v jazyku nástroje Scyther
- Velká efektivita při analýze bezpečnostních protokolů
- Jednoduché ovládání
- Nalezené útoky přehledně vykresleny v grafu

**Nevýhody:**

- Oproti nástrojům AVISPA/Isabelle omezené možnosti modelování protokolů
- Minimální počet možností jak ovlivnit analýzu
- Nevhodný pro modelování IETF protokolů

# 6 Analýza vybraných bezpečnostních protokolů v nástroji Scyther

V následující části práce se budu zabývat praktickou analýzou bezpečnostních protokolů v nástroji Scyther. Jednotlivé podkapitoly s bezpečnostními protokoly jsou členěny podle jednotného klíče, v první části je uveden zápis protokolu pomocí *Alice – Bob* komunikačního modelu, následuje popis příslušného protokolu, specifikace protokolu pomocí *Security Protocols Description Language*. Dále jsou uvedeny bezpečnostní požadavky, které musí protokol splňovat. Stejné požadavky jsou zkoumány v nástroji Scyther. Po části s bezpečnostními požadavky v jazyku SPDL je popsán výsledek verifikace. Pokud je nalezen úspěšný útok na protokol v nástroji Scyther, je graficky zobrazen. Výchozí nastavení Scytheru pro analýzu protokolů (pokud není u jednotlivých protokolů řečeno jinak): Max. number of runs – 5, Matching type – typed matching, Max. number of patterns per claim – 10.

Jak bylo řečeno v předchozí kapitole, Scyther využívá Dolev – Yalo bezpečnostní model [45] při verifikaci bezpečnostních protokolů. Útočník má kompletní kontrolu nad síťovou infrastrukturou, po které se přenášejí zprávy. Také vychází z předpokladu, že použitá kryptografie je bezchybná [50]. Útočník tedy není schopen rozluštit zprávu šifrovanou symetrickým nebo asymetrickým klíčem, pokud nezná příslušný dešifrovací klíč.

Pokud Scyther nalezne současně útoky na *Non-injective synchronisation* a *Non-injective agreement* [44], budeme se zabývat pouze útokem na ne-injektivní synchronizaci. Ta je definována výše v hierarchii autentizačních požadavků, proto pokud byl nalezen útok současně na oba požadavky, má smysl se zabývat tím důležitějším. Také je nutné definovat rozhodnutí, kdy je zkoumaný protokol bezpečný a naopak. Scyther nám bude sloužit jako nástroj k odhalování možných problematických částí v protokolu. Konečný výsledek bude dán prozkoumáním grafického výstupu Scytheru a zhodnocením útočníkem získaných informací. Pokud výsledek analýzy nepovede k narušení základních cílů a požadavků protokolu, bude protokol považován za bezpečný.

Kompletní zdrojový kód a originální grafický výstup z nástroje Scyther je uveden pouze pro první analyzovaný protokol, aby měl čtenář představu, jak je protokol modelován nebo jakého výstupu lze dosáhnout. Pro nadměrnou velikost budou všechny ostatní grafické výstupy modelovaných protokolů umístěny v příloze 3 nebo na přiloženém CD.

## 6.1 Protokol Andrew Secure RPC

**Zápis protokolu**

1. *A → B : A, {Na}Kab*
2. *B → A : {Na+1, Nb}Kab*
3. *A → B : {Nb+1}Kab*
4. *B → A : {K'ab, N'b}Kab*

**Popis**

Důkladný popis prokolu Adrew Secure RPC se nachází v kapitole 4.1 na straně 13. Jen pro připomenutí, jedná se o protokol pracující na bázi symetrické kryptografie a jeho úkolem je bezpečná výměna relačního klíče *K'ab*.

**Specifikace protokolu v nástroji Scyther**

```
usertype RelKlic;              #Uzivatelsky definovany datovy typ, definice relacniho klice
secret k: Function;            #Funkce symetrickeho klice
const noncee: Function;        # Funkce naslednika nonce (Na+1)

protocol AndrewSecure(A,B)     #Nazev protokolu spolu s definici roli v protokolu
{
    role A                     #Iniciator - role Alice
    {
        const Na: Nonce;       #Pokud agent vytvari nonce, je definovano jako konstanta a v teto roli je
                               #neznamo utocnikovi
        var Nb, Nb': Nonce;    #Prijata nonce, pro tuto roli jsou definovana jako pripadne zkompromitovana
                               # -> var
        varKab: RelKlic;       #Prijimany relacni klic, pripadne zkompromitovany

        send_1(A,B, A,{Na}k(A,B) );    #Funkce poslani zpravy, je to synchronni udalost, musi mit definovane
                                       #prislusne read, cislo udava poradi v protokolu
        read_2(B,A, {noncee(Na),Nb}k(A,B) );    #Funkce prijeti zpravy, musi mit prislusny send
        send_3(A,B, {noncee(Nb)}k(A,B) );
        read_4(B,A, {Kab, Nb'}k(A,B) );

        claim_A1(A,Secret, Na);        #Definovany bezpecnostni pozadavek, utajitelnost Na
        claim_A2(A,Secret, Kab);       #Definovany bezpecnostni pozadavek, utajitelnost relacniho klice Kab
        claim_A3(A,Nisynch);           #Definovany bezpecnostni pozadavek, non injektivni synchronizace
        claim_A4(A,Niagree);           #Definovany bezpecnostni pozadavek, non injektivni dohoda
    }
    role B #Responder - role Boba
```

```
    {
        const Nb, Nb': Nonce;
        var Na: Nonce;
        const Kab: RelKlic;

        read_1(A,B, A,{Na}k(A,B) );
        send_2(B,A, {noncee(Na),Nb}k(A,B) );
        read_3(A,B, {noncee(Nb)}k(A,B) );
        send_4(B,A, {Kab, Nb'}k(A,B) );

        claim_B1(B,Secret,Nb);
        claim_B5(B,Secret,Nb');
        claim_B2(B, Secret, Kab);
        claim_B3(B, Nisynch);
        claim_B4(B, Niagree);
    }
}
protocol AndrewCompromised(U)     #Vytvoreni vlastni role utocnika, u nekterych protokolu je nutne
                                  #definovat jeho chovani a znalosti
{
        role U #Role Utocnika
    {
        var A,B: Agent;           #Utocnik je schopen dozvedet se role ostatnich agentu v protokolu
        const Na,Nb,Nb': Nonce;   #utocnik je schopen vytvorit vlastni zkompromitovana nonce
        const Kab: RelKlic;       #utocnik je schopen vytvorit vlastni zkompromitovan relacni klic

        read_!U1(U,U, A,B);       #Utocnik odposlechne identity agentu z predesle komunikace,
                                  #vyjadreno jako ziskani zpravy od sama sebe, je pouzit ! protoze read nema
                                  #ekvivalentni send
        send_U2(U,U, (A,{Na}k(A,B)),{noncee(Na),Nb}k(A,B),{noncee(Nb)}k(A,B),{Kab, Nb'}k(A,B),Kab);
    }
}
const Alice, Bob, Eve: Agent;     #Definovani agentu v protokolu
untrusted Eve;                    #Agent Eva je definovan jako neduveryhodny
```

**Bezpečnostní požadavky**

Tento protokol zajišťuje ve svém běhu aktualizaci starého relačního klíče novým. Starý klíč se používá k šifrování komunikace až do kroku 4, kdy je *Alici* zaslán nový relační klíč spolu s nonce

*Nb'*. Toto nonce pak slouží jako počáteční sekvenční číslo, od kterého je číslována budoucí komunikace šifrovaná novým relačním klíčem *K'ab*.

Protokol by měl zajistit bezpečný přenos nonce *Na, Nb, Nb'* a relačního klíče *K'ab*, což bude ověřeno v nástroji Scyther. Inkrementované nonce posílané v 2. a 3. kroku protokolu jsou vyjádřeny pomocí funkce *noncee(x)*, protože jazyk *spdl* neumí vyjádřit inkrement nonce pomocí matematického zápisu *Nx + 1*. Dále bylo nutné vytvořit druhý protokol pro roli *Utocnik*, která charakterizuje nečestného agenta odposlouchávajícího komunikaci.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

```
#Alice
claim_A1(A,Secret, Na);   #Pozadavek na  utajitelnost Na
claim_A2(A,Secret, Kab); # Pozadavek na  utajitelnost relacniho klice Kab
claim_A3(A,Nisynch);      # Pozadavek  non injektivni synchronizace
claim_A4(A,Niagree);      # Pozadavek non injektivni agreement
```

```
# Bob
claim_B1(B,Secret,Nb);    #Pozadavek na  utajitelnost Nb
claim_B5(B,Secret,Nb');   #Pozadavek na  utajitelnost Nb'
claim_B2(B, Secret, Kab);
claim_B3(B, Nisynch);
claim_B4(B, Niagree);
```

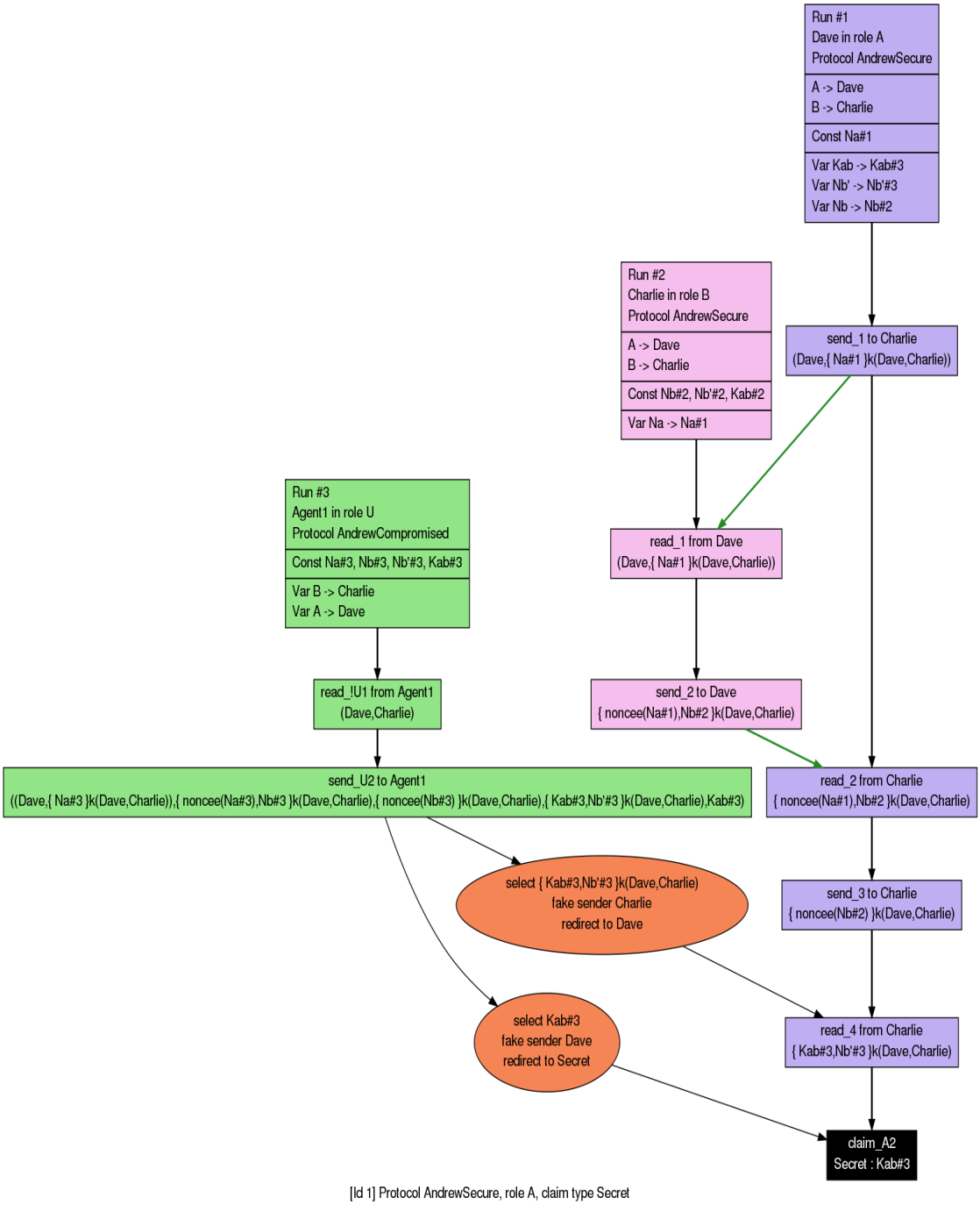
**Výsledek verifikace**

Už při prvním pohledu na schéma komunikace je zřejmý největší nedostatek protokolu. A tím je absence ověření aktuálnosti zprávy v kroku 4. Nonce *Nb'* totiž slouží k zabezpečení budoucí komunikace pomocí sekvenčního čísla, ale bohužel na bezpečnost té současné nemá žádný vliv. Útočník tak může použít starší komunikace k vyjednání používání zkompromitovaného relačního klíče. Na druhou stranu protokol je schopen zajistit utajitelnost všech nonce vyskytujících se v komunikaci.

**Útok přehráním:** *Utocnik* disponuje znalostí identit všech agentů participujících při běhu protokolu. Za jeho počáteční znalost také můžeme považovat odposlechnutou starší komunikaci, v níž se objevuje rel. klíč *Kab#3* a nonce *Nb'#3*. Komunikace mezi čestnými agenty probíhá až do kroku 4 beze změn. V této části *Alice* (v grafu *Dave)* očekává zprávu s novým relačním klíčem. Protože ale zpráva neobsahuje žádný prvek, který by dokazoval její aktuálnost, může *Utocnik* (v grafu jako *Agent1*) podstrčit svoji odposlechnutou zprávu. *Alice* tak věří, že právě to je klíč zaslaný *Bobem* jako odpověď na její žádost, a proto s ním souhlasí.

**Závěr:** Protokol Adrew Secure RPC je po verifikaci nástrojem Scyther považován za **nebezpečný.**

## Grafické zobrazení útoku




**Obrázek 7 - Grafické zobrazení útoku na Kab'**

## 6.2 BAN modifikovaný Andrew Secure RPC protokol

### Zápis protokolu

1. $A \rightarrow B : A, \{Na\}Kab$
2. $B \rightarrow A : \{Na+1, Nb\}Kab$
3. $A \rightarrow B : \{Nb+1\}Kab$
4. $B \rightarrow A : \{K'ab, N'b, Na\}Kab$

### Popis

Jedná se o modifikaci původního protokolu [20] autory Burrowsem, Abadim a Needhamem. Ti po analýze BAN logikou objevili chybu u původního protokolu, kterou opravují přidáním nonce *Na*. Více lze nalézt v části 4.2.

### Specifikace jednotlivých rolí v nástroji Scyther

```
# Iniciator – agent Alice
const Na: Nonce;
var Nb, Nb': Nonce;
var Kab: RelKlic;

send_1(A,B, A,{Na}k(A,B) );
read_2(B,A, {noncee(Na),Nb}k(A,B) );
send_3(A,B, {noncee(Nb)}k(A,B) );
read_4(B,A, {Kab, Nb',Na}k(A,B) );
```

```
# Responder – agent Bob
const Nb, Nb': Nonce;
var Na: Nonce;
const Kab: RelKlic;

read_1(A,B, A,{Na}k(A,B) );
send_2(B,A, {noncee(Na),Nb}k(A,B) );
read_3(A,B, {noncee(Nb)}k(A,B) );
send_4(B,A, {Kab, Nb',Na}k(A,B) );
```

```
# Utocnik
var A,B: Agent;
const Na,Nb,Nb': Nonce;
const Kab: RelKlic;

read_!U1(U,U, A,B);
send_U2(U,U,  (A,{Na}k(A,B)),
{noncee(Na),Nb}k(A,B),
{noncee(Nb)}k(A,B),
{Kab, Nb',Na}k(A,B),Kab);
```

### Bezpečnostní požadavky

Zůstávají stejné podobně jako u původního protokolu Andrew Secure RPC [20]. Musí být zajištěno utajení klíče *K'ab* a nonce *Na, Nb, Nb'*.

### Specifikace bezpečnostních požadavků v nástroji Scyther

```
# Alice
claim_A1(A,Secret, Na);
claim_A2(A,Secret, Kab);
claim_A3(A,Nisynch);
claim_A4(A,Niagree);
```

```
# Bob
claim_B1(B,Secret,Nb);
claim_B2(B,Secret,Nb');
claim_B3(B, Secret, Kab);
claim_B4(B, Nisynch);
claim_B5(B, Niagree);
```

### Výsledek verifikace

Na protokol nebyl nalezen žádný útok pomocí nástroje Scyther.

**Závěr:** BAN modifikovaný Adrew Secure RPC protokol je po verifikaci nástrojem Scyther považován za **bezpečný.**

## 6.3 Challenge/Response Authentication Protocol (MS-CHAPv2)

**Zápis protokolu**

1. $A \rightarrow B : A$
2. $B \rightarrow A : Nb$
3. $A \rightarrow B : Na, hash(Kab,(Nb,Na,A))$
4. $B \rightarrow A : hash(Kab,Na)$

**Popis**

Challenge Authentication Protocol verze 2 [33] lze zařadit do třídy Point-to-Point Tunneling protokolů. Používá se hlavně jako autentizační protokol v bezdrátových sítích (starší operační systémy firmy Microsoft, například Windows 2000). Uživatel, který se chce autentizovat pomocí hesla, pošle tzv. Challenge paket, na který odpoví příslušný autentizační server pomocí Response paketu. Pokud heslo zaslané uživatelem souhlasí s heslem uloženým v databázi, server pošle Success paket, čímž je uživatel autorizován k používání daného připojení.

Autentizace probíhá v následujících krocích [34] [35]:

1. Uživatel (klient) zašle na server výzvu k přihlášení (login)
2. Server odpoví klientovi zasláním 16B náhodného čísla (Authenticator challenge)
3. Klient vytvoří odpověď podle následujících kroků:
    a) Klient si vygeneruje náhodné 16B číslo (Peer authenticator challenge)
    b) Pomocí SHA (Secure Hash Algorithm) vytvoří hash z 16B čísla získaného ze serveru v kroku 2, svého vygenerovaného čísla z kroku 3a) a uživatelského přihlašovacího jména (login)
    c) Klient vygeneruje NT password hash[19] ze svého uživatelského hesla
    d) 16B NT password hash je doplněn 5 nulami a z výsledných 21 bytů jsou vytvořeny tři 7 bytové klíče pomocí DES
    e) Prvních 8 bytů z hashe vygenerovaného v kroku 3b) je zašifrováno pomocí 3 DES klíčů z 3d)
    f) Serveru je zaslána výsledná zpráva sestavená z 16B čísla z 3a) a výsledné 24B šifry z 3e)
4. Server rozluští zprávu pomocí hashe uživat. hesla uloženého v databázi

[19] NT password hash je vytvořen pomocí algoritmu MD4.

5. Pokud heslo souhlasí, server vytvoří následující odpověď jako hash z 16B Peer authenticator challenge a 20B Authenticator Response (proces tvorby lze nalézt v [34] a [35])
6. Klient si také vytvoří 20B Authenticator Response a porovná ho s Auth. Response získaným ze serveru. Pokud hashe souhlasí, jsou oba autentizováni.

### Specifikace jednotlivých rolí v nástroji Scyther

| | |
|---|---|
| ***# Iniciator – agent Alice*** | ***# Responder – agent Bob*** |
| *const Na: Nonce;* | const Nb: Nonce; |
| *var Nb: Nonce;* | var Na: Nonce; |
| *const pass: Password;* | const pass: Password; |
| | |
| *send_1(A,B, A);* | read_1(A,B, A); |
| *read_2(B,A, Nb);* | send_2(B,A, Nb); |
| *send_3(A,B, Na,hash(pass,(Nb,Na,A)) );* | read_3(A,B, Na,hash(pass,(Nb,Na,A)) ); |
| *read_4(B,A, hash(pass,Na) );* | send_4(B,A, hash(pass,Na) ); |

### Bezpečnostní požadavky

Protokol nemůže zajistit bezpečnost nonce *Na* ani *Nb*, protože obě jsou posílané v nezašifrované formě. Nebudeme tedy zkoumat utajitelnost *Na, Nb*. Soustředíme se pouze na zasílané tajemství *pass* (*Kab)* a také budeme samozřejmě zkoumat odolnost protokolu proti přehrávání a prokládání zpráv.

### Specifikace bezpečnostních požadavků v nástroji Scyther

| | |
|---|---|
| ***# Alice*** | ***# Bob*** |
| *claim_A1(A,Secret,pass);* | *claim_B1(B,Secret,pass);* |
| *claim_A2(A,Nisynch);* | *claim_B2(B,Nisynch);* |
| *claim_A3(A,Niagree);* | *claim_B3(B,Niagree);* |

### Výsledek verifikace

V návrhu protokolu MS-CHAPv2 nebyly nástrojem Scyther nalezeny žádné bezpečnostní slabiny. Protokol je schopen zajistit utajitelnost *Kab* i všech nonce.

**Závěr:** Protokol MS-CHAPv2 je po verifikaci nástrojem Scyther považován za **bezpečný.**

Pozn.: Protokol má pár závažných chyb ve způsobu šifrování (např. třetí DES klíč je příliš slabý), což umožňuje použít například **slovníkový útok** [34] [35].

## 6.4 Challenge/Response Authentication Mechanism (CRAM-MD5)

**Zápis protokolu**

1. $A \rightarrow B : A$
2. $B \rightarrow A : Nb,T,B$
3. $A \rightarrow B : A, hash(Kab,T)$

**Popis**

Tento protokol zavádí challenge/response mechanismus pro protokol IMAP. Protože autentizace v IMAP protokolu byla prováděna příkazem LOGIN [38] pouze pomocí hesla a uživatelského jména zasílaného jako nešifrovaný text, byl zaveden autentizační mechanismus pro IMAP [39].

Průběh protokolu CRAM-MD5 lze popsat takto:

1) Uživatel se příhlásí na server (*Bob*) svým uživatelským jménem
2) Odpověď serveru obsahuje řetězec náhodných čísel *Na*, časové razítko *T*, jméno serveru *B*
3) Client (*Alice*) odpoví uživatelským jménem a HMAC-MD-5 16B šifrou [40] z hesla (v protokolu jako rel. klíč *Kab*) a časového razítka *T*
4) Server dešifruje přijatou zprávu a pokud souhlasí uživatelské heslo získané z šifry s heslem uloženým v databázi serveru, je uživatel autentizován

**Specifikace jednotlivých rolí v nástroji Scyther**

```
# Iniciator – agent Alice
var Nb:  Nonce;
const pass: Password;
var T: Timestamp;

send_1(A,B, A);
read_2(B,A, Nb,T,B);
send_3(A,B, A,hash(pass,T) );
```

```
# Responder – agent Bob
const Nb: Nonce;
const pass: Password;
const T:Timestamp;

read_1(A,B, A);
send_2(B,A, Nb,T,B);
read_3(A,B, A,hash(pass,T) );
```

**Bezpečnostní požadavky**

Protokol musí zajistit bezpečný přenos uživatelského hesla *Kab*. Čerstvost zpráv je ověřována pomocí časových razítek, budeme tedy kontrolovat, zda útočník nemůže podvrhnout serveru časové razítko *T*. Standardní jsou požadavky na správnou synchronizaci zpráv a proměnných u *Boba*. Pro *Alici* kontrola synchronizace nemá smysl.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

```
# Alice
claim_A1(A,Secret,pass);
```

```
# Bob
claim_B1(B,Secret,pass);
claim_B2(B,Secret,T);
claim_B3(B,Nisynch);
claim_B4(B,Niagree);
```

**Výsledek verifikace**

V nástroji Scyther nebyla nalezena žádná chyba v testovaných bezpečnostních vlastnostech CRAM-MD5 protokolu.

**Závěr:** Protokol CRAM-MD5 je po verifikaci nástrojem Scyther považován za **bezpečný.**

## 6.5 Denning – Sacco shared key protokol

**Zápis protokolu**

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{B, Kab, T, \{Kab, A, T\}Kbs\}Kas$
3. $A \rightarrow B : \{Kab, A, T\}Kbs$

**Popis**

Tento protokol jsem zvolil pro analýzu, protože namísto většiny protokolů ze SPORE používá časové značky místo nonce pro ověření čerstvosti posílaných zpráv. Relační klíč *Kab* je získáván z důvěryhodného serveru a distribuován mezi další účastníky komunikace.
Detailnější popis spolu se známým útokem lze nalézt v části 4.3.

**Specifikace jednotlivých rolí v nástroji Scyther**

```
# Iniciator – agent Alice
var Kab: RelKlic;
var T: Timestamp;

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

```
# Responder – agent Bob
var Kab: RelKlic;
var T: Timestamp;

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

```
# Role Serveru
const Kab: RelKlic;
const T: Timestamp;

read_1(A,S, A,B);
send_2(S,A, {B,Kab,T,{Kab,A,T}k(B,S)}k(A,S) );
```

**Bezpečnostní požadavky**

Protokol musí zajistit bezpečný přenos relačního klíče *Kab* mezi *Serverem* a *Alicí* v kroku 2, a také mezi *Alicí* a *Bobem* v kroku 3. Časové značky musí zaručovat aktuálnost komunikace a svoji utajitelnost.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

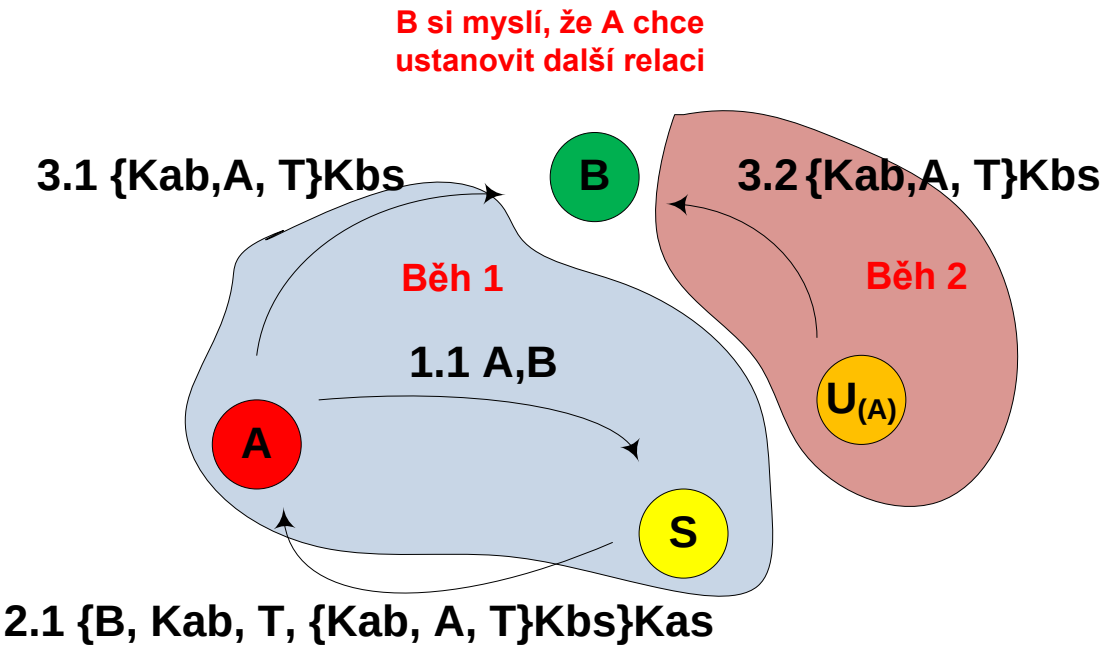
| ***# Alice*** | ***# Bob*** | ***# Server*** |
|---|---|---|
| *claim_A1(A,Secret,Kab);* | *claim_B1(B,Secret,Kab);* | *claim_S1(S,Secret,T);* |
| *claim_A2(A,Secret,T);* | *claim_B2(B,Secret,T);* | |
| *claim_A3(A,Nisynch);* | *claim_B3(B,Nisynch);* | |
| *claim_A4(A,Niagree);* | *claim_B4(B,Niagree);* | |

**Výsledek verifikace**

Na protokol je známý **násobný útok** objevený Gavinem Lowem [15]. Nástroj Scyther naznačuje narušení synchronizace protokolu v kroku 3, kdy *Alice* předává relační klíč *Bobovi*. Pokud běží více instancí protokolu, může *Alice* zaslat zprávu obsahující relační klíč v jiném pořadí. *Bob* pouze pasivně očekává zprávu s relačním klíčem, jejíž jedinou kontrolou čerstvosti je časová značka. Jestliže obdrží více zpráv od stejného uživatele, považuje je za žádosti o navázání dalších sezení. Toho využívá útočník při provádění násobného útoku. Přeposláním zprávy z *{Kab, A, T}Kbs* přesvědčí *Boba* o tom, že se *Alice* snaží navázat další sezení, což vede k útočníkově autentizaci.

**Závěr:** Protokol Denning – Sacco shared key je po verifikaci nástrojem Scyther považován za **nebezpečný.** Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 18.

**Grafické zobrazení útoku**




**Obrázek 8 - Denning – Sacco – násobný útok na rel. klíč**

## 6.6 Lowův modifikovaný Denning – Sacco shared key protokol

**Zápis protokolu**

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{B, Kab, T, \{Kab, A, T\}Kbs\}Kas$
3. $A \rightarrow B : \{Kab, A, T\}Kbs$
4. $B \rightarrow A : \{Nb\}Kab$
5. $A \rightarrow B : \{Nb+1\}Kab$

**Popis**

Lowe modifikoval [15] originální protokol [16] přidáním výměny nonce po předání relačního klíče. Původní část využívající časových razítek zůstala nezměněna.

### Specifikace jednotlivých rolí v nástroji Scyther

```
# Iniciator – agent Alice
var Kab: RelKlic;
var  T: Timestamp;
var Nb: Nonce;

send_1(A,S, A,B);
read_2(S,A, {B,Kab,T,{Kab,A,T}k(B,S)}k(A,S));
send_3(A,B, {Kab,A,T}k(B,S) );
read_4(B,A, {Nb}k(A,B));
send_5(A,B, {noncee(Nb)}k(A,B));
```

```
# Responder – agent Bob
var Kab: RelKlic;
var T: Timestamp;
const  Nb: Nonce;

read_3(A,B, {Kab,A,T}k(B,S));
send_4(B,A, {Nb}k(A,B));
read_5(A,B, {noncee(Nb)}k(A,B));
```

```
# Role Serveru
const Kab: RelKlic;
const T: Timestamp;

read_1(A,S, A,B);
send_2(S,A, {B,Kab,T,{Kab,A,T}k(B,S)}k(A,S));
```

### Bezpečnostní požadavky

Časová razítka a nově přidaná nonce musí zajistit čerstvost zpráv při výměně relačního klíče. Tím se zabrání možnosti provedení útoků přehráním a špatné synchronizaci přijímání zpráv v protokolu. Požadavek utajení relačního klíče při přenosu je samozřejmostí.

### Specifikace bezpečnostních požadavků v nástroji Scyther

```
# Alice
claim_A1(A,Secret,Kab);
claim_A2(A,Secret,Nb);
claim_A3(A,Secret,T);
claim_A4(A,Nisynch);
claim_A5(A,Niagree);
```

```
# Bob
claim_B1(B,Secret,Kab);
claim_B2(B,Secret,Nb);
claim_B3(B,Secret,T);
claim_B4(B,Nisynch);
claim_B5(B,Niagree);
```

```
# Server
claim_S1(S,Secret,T);
```

### Výsledek verifikace

Při verifikaci bylo změněno nastavení velikosti stavového prostoru z 5 na 10, protože Scyther nebyl schopný nalézt jednoznačný závěr pro nastavené maximum 5ti běhů.

Podle výstupu Scytheru je Lowův modifikovaný Denning – Sacco protokol, podobně jako originální verze [16], náchylný k přeposlání zpráv v kroku 3. Útočník ale nemůže použít stejného způsobu přehrání zpráv, a to kvůli přidané výměně nonce.

Nalezený útok na synchronizaci zpráv vychází z obecné deklarace proměnné *Timestamp* v jazyce SPDL. Scyther prověřuje tuto proměnnou jako výsledek uživatelsky deklarované funkce, která může být přeposlána z jiných běhů protokolu. Nepokládá proměnnou *Timestamp* za časové razítko sloužící právě jako prevence proti přeposílání zpráv z jiných nebo starších běhů protokolu.

**Závěr:** Lowův modifikovaný Denning – Sacco protokol je po verifikaci nástrojem Scyther považován za **bezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 19.

## 6.7 EKE – Encrypted Key Exchange protokol

**Zápis protokolu**

1. *A → B : A, {KPa}Kab*
2. *B → A : {{K'ab}KPa}Kab*
3. *A → B : {Na}K'ab*
4. *B → A : {Na,Nb}K'ab*
5. *A → B : {Nb}K'ab*

**Popis**

Encrypted Key Exchange je název pro rodinu protokolů založených na přenosu **náhodně** generovaného hesla, které pak slouží jako dohodnutý relační klíč. Autory tohoto přístupu jsou Steven M. Bellovin a Michael Merrit ve své publikaci [37]. Pro výměnu relačního klíče je používána kombinace symetrické (uživatelské heslo) a asymetrické kryptografie (náhodně generovaný veřejný klíč), která podle autorů zamezuje použití slovníkového útoku na šifrovaný klíč.

*Alice* vygeneruje náhodný veřejný klíč *Kpa*, který zašifruje symetrickým klíčem *Kab* a zašle *Bobovi*. *Bob* po rozšifrování zprávy náhodně vygeneruje relační klíč *K'ab*, který zašifruje veřejným klíčem *KPa* a symetrickým klíčem *Kab*. První dva kroky se nazývají *část výměny relačního klíče* (Key exchange part). Třetím krokem začíná část *výzva – odpověď* (Challenge – Response part). *Alicí* je vygenerováno nonce *Na* (challenge) a šifrované relačním klíčem posláno *Bobovi*. Ten vygeneruje nonce *Nb* a zašle zpátky spolu s *Na*. Pokud přijaté *Na* souhlasí, *Alice* odpoví zasláním *Nb*. Pokud i *Nb* souhlasí, je přihlašování kompletní a oba účastníci pokračují v komunikaci ve stejném sezení, používajíc k šifrovaní symetrický klíč *K'ab*.

**Specifikace jednotlivých rolí v nástroji Scyther**

```
# Iniciator – agent Alice
const Na: Nonce;
var Nb: Nonce;
var K'ab: RelKlic;

send_1(A,B, A,{PK(A)}k(A,B));
read_2(B,A, {{K'ab}PK(A)}k(A,B));
send_3(A,B, {Na}k(A,B));
read_4(B,A, {Na,Nb}k(A,B));
send_5(A,B, {Nb}k(A,B));
```

```
# Responder – agent Bob
const Nb: Nonce;
var Na: Nonce;
const K'ab: RelKlic;

read_1(A,B, A,{PK(A)}k(A,B));
send_2(B,A, {{K'ab}PK(A)}k(A,B));
read_3(A,B, {Na}k(A,B));
send_4(B,A, {Na,Nb}k(A,B));
read_5(A,B, {Nb}k(A,B));
```

**Bezpečnostní požadavky**

V protokolu budeme testovat utajitelnost nonce *Na,Nb* a přenášeného relačního klíče *K'ab*. Dále se bude kontrolovat utajitelnost veřejného klíče *PKa*. Protože tento klíč nemá být podle definice protokolu známý ostatním, považuji za vhodné přidat *PKa* do bezpečnostních požadavků.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

```
# Alice
claim_A1(A,Secret, K'ab);
claim_A2(A,Secret,Na);
claim_A3(A,Secret,PK(A));
claim_A4(A,Niagree);
claim_A5(A,Nisynch);
```

```
# Bob
claim_B1(B,Secret, K'ab);
claim_B2(B,Secret,Nb);
claim_B3(B,Niagree);
claim_B4(B,Nisynch);
```

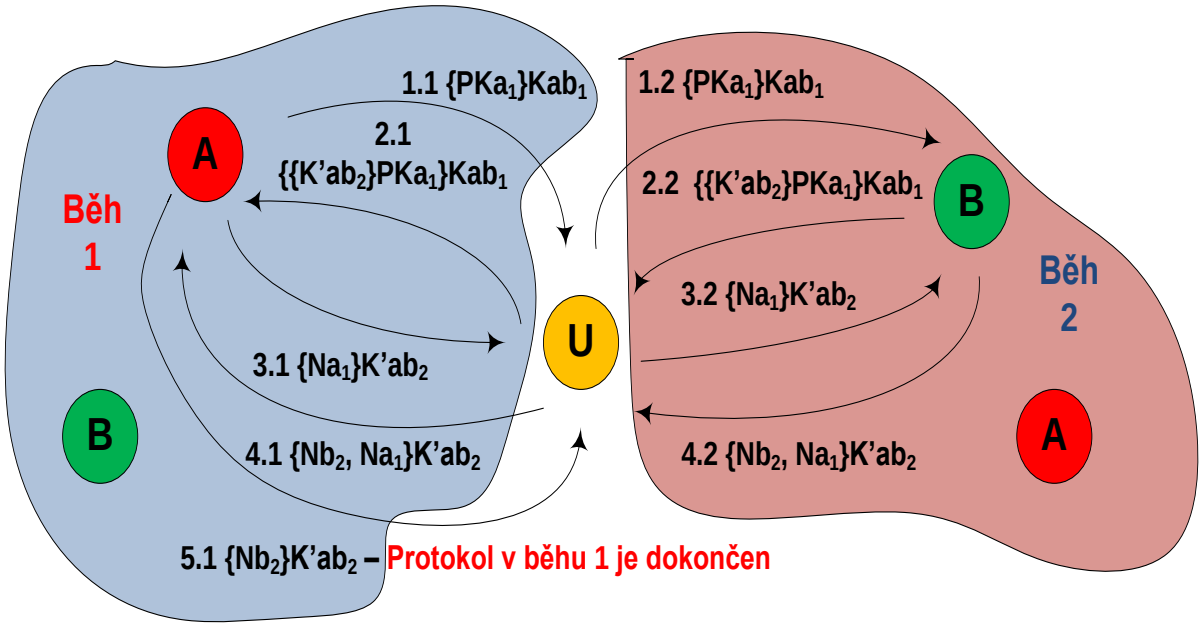
**Výsledek verifikace**

Nalezen útok **Man-in-the-middle** – útočník je schopen přesměrovávat posílání zpráv mezi dvěma paralelními běhy EKE protokolu. Po odposlechnutí první zprávy $\{PKa_1\}Kab_1$ od $Alice_1$ ji útočník přepošle $Bobovi_2$ , čímž začně druhý paralelní běh protokolu. Jako odpověď získá zpátky zprávu s relačním klíčem $K'ab_2$ , kterou předá zpátky $Alici_1$ , a tím dokončí část výměny relačního klíče v běhu 1. $Alice_1$ začíná challenge – response část zasláním $\{Na_1\}K'ab_2$ .

Útočník zprávu přepošle $Bobovi_2$ a jeho odpověď přepošle zpět $Alici_1$ . Následuje potvrzení od $Alice_1$ . Tím je protokol v běhu 1 kompletní a útočníkovi se podařilo zamaskovat jako $Bob_1$ . Chyba je v synchronizaci protokolu, respektive synchronizaci zpráv mezi částí pro výměnu relačního klíče a částí challenge – response.

**Závěr:** Encrypted Key Exchange protokol je po verifikaci nástrojem Scyther považován za **nebezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 20.

**Grafické zobrazení útoku**




**Obrázek 9 - EKE protokol – útok man-in-the-middle**

## 6.8 Needham – Schroeder Public key

**Zápis protokolu**

1. *A → S : A, B*
2. *S → A : {KPb, B}KSs*
3. *A → B : {Na, A}KPb*
4. *B→ S : B, A*
5. *S → B : {KPa, A}KSs*
6. *B→ A : {Na, Nb}KPa*
7. *A → B : {Nb}KPb*

**Popis**

Needham –Schroeder Public key protokol využívá asymetrických klíčů a důvěryhodného serveru. Protokol je detailněji popsán v části 4.10.

**Specifikace jednotlivých rolí v nástroji Scyther**

| *# Iniciator – agent Alice* | *# Responder – agent Bob* | *# Role Serveru* |
|---|---|---|
| *const Na: Nonce;* | *var Na: Nonce;* | |
| *var Nb: Nonce;* | *const Nb: Nonce;* | |
| | | |
| *send_1(A,S, A,B);* | *read_3(A,B, {Na,A}pk(B));* | *read_1(A,S, A,B);* |
| *read_2(S,A, {pk(B), B}sk(S));* | *send_4(B,S, B,A);* | *send_2(S,A, {pk(B), B}sk(S));* |
| *send_3(A,B, {Na,A}pk(B));* | *read_5(S,B, {pk(A),A}sk(S));* | *read_4(B,S, B,A);* |
| *read_6(B,A, {Na,Nb}pk(A));* | *send_6(B,A, {Na,Nb}pk(A));* | *send_5(S,B, {pk(A),A}sk(S));* |
| *send_7(A,B, {Nb}pk(B));* | *read_7(A,B, {Nb}pk(B));* | |

**Bezpečnostní požadavky**

Protokol nepřenáší žádný relační klíč, jeho úkolem je autentizace komunikujících uživatelů. Na konci relace musí být oba účastníci přesvědčeni o identitě toho druhého v závislosti na použitých veřejných klíčích. Informace o přenášených nonce nesmí být dostupné pro nikoho mimo agenty podílející se na vykonávání protokolu.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *claim_A1(A,Secret,Na);* | *claim_B1(B,Secret,Na);* | *No claims* |
| *claim_A2(A,Secret,Nb);* | *claim_B2(B,Secret,Nb);* | |
| *claim_A3(A,Nisynch);* | *claim_B3(B,Nisynch);* | |
| *claim_A4(A,Niagree);* | *claim_B4(B,Niagree);* | |

**Výsledek verifikace**

Scyther nachází známou chybu v nedostatečném ověřování čerstvosti zpráv, vedoucí k úspěšnému útoku přehráním starších zpráv. Po studii grafického výstupu se zdá, že Scyther nalezl totožný útok na Needham – Schroeder Public key protokol, popsaný už Gavinem Lowem v [43].

Útok **man-in-the-middle** – *Alice* v roli iniciátora protokolu zašle na *Server* žádost o veřejný klíč útočníka *U*. *Server* odpoví *Alici* zasláním útočníkova veřejného klíče a identity šifrované soukromým klíčem *KSs*. *Alice* následně zašle útočníkovi zprávu pro potvrzení aktuálnosti jejich komunikace pomocí nonce *Na*. Útočník jednoduše dešifruje tuto zprávu svým soukromým klíčem *KSu* a znovu zašifruje veřejným klíčem *Boba*, čímž získá zprávu *{Na, A}KPb*.

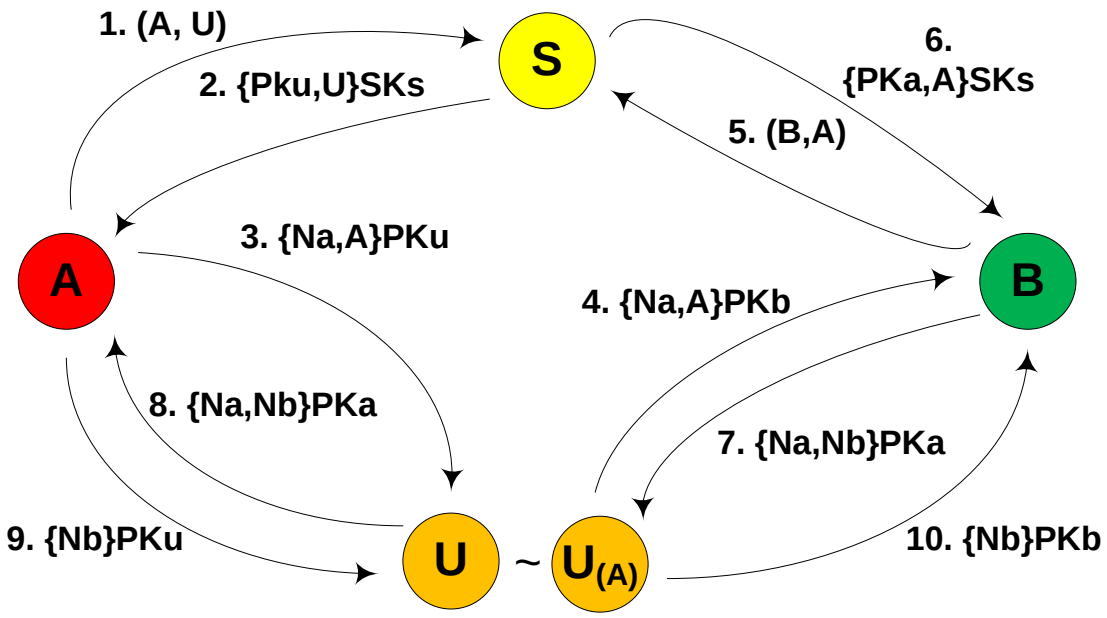
Nyní začíná druhá část útoku, kdy útočník provádí protokol v přestrojení za *Alici* jako iniciátor. Jeho počáteční znalost nyní zahrnuje *{Na, A}KPb*. Útočník zahájí druhou část zasláním *{Na, A}KPb Bobovi*, který si vyžádá od *Serveru* veřejný klíč *Alice*. *Bob* po obdržení odpovědi pokračuje v protokolu zasláním potvrzovací zprávy *{Na, Nb}PKa*. Zde si můžeme všimnout, že se v této zprávě nikde neobjevuje, kým byla vytvořena. V podstatě pak takovéto ověření čerstvosti ztrácí smysl,

protože nonce *Na* může být použito z minulých běhů protokolu, nonce *Nb* může být vytvořeno kýmkoliv a veřejný klíč *PKa* je také známý ostatním agentům.

Zpráva obsahující potvrzovací nonce je útočníkem přeposlána *Alici*. Ta porovná přijaté nonce *Na* a tím si ověří, že jde o odpověď na výzvu z 3. kroku první části protokolu. Jako odpověď zašifruje nonce *Nb* veřejným klíčem *PKu*. Útočník zprávu znovu zašifruje klíčem *PKb* a zašle *Bobovi*. Protokol je dokončen, útočník je autentizován jako *Alice*.

**Závěr:** Protokol Needham – Schroeder Public key je po verifikaci nástrojem Scyther považován za **nebezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 21.

**Grafické zobrazení útoku**




**Obrázek 10 - Needham – Schroeder Public key – útok man-in-the-middle**

## 6.9 Lowova verze protokolu Needham – Schroeder Public key

**Zápis protokolu**

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{KPb, B\}KSs$
3. $A \rightarrow B : \{Na, A\}KPb$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{KPa, A\}KSs$
6. $B \rightarrow A : \{Na, Nb, B\}KPa$
7. $A \rightarrow B : \{Nb\}KPb$

**Popis**

Gavin Lowe pozměnil Needham – Schroeder Public key [43] protokol přidáním identity odesílatele do zprávy zasílané v šestém kroku. Podpis odesílatele zabraňuje útočníkovi v přeposlání zprávy v kroku 6 a díky tomu nelze provést útok nalezený v 6.8.

**Specifikace jednotlivých rolí v nástroji Scyther**

| *# Iniciator – agent Alice* | *# Responder – agent Bob* | *# Role Serveru* |
|---|---|---|
| *const Na: Nonce;* | *var Na: Nonce;* | |
| *var Nb: Nonce;* | *const Nb: Nonce;* | |
| | | |
| *send_1(A,S, A,B);* | *read_3(A,B, {Na,A}pk(B));* | *read_1(A,S, A,B);* |
| *read_2(S,A, {pk(B), B}sk(S));* | *send_4(B,S, B,A);* | *send_2(S,A, {pk(B), B}sk(S));* |
| *send_3(A,B, {Na,A}pk(B));* | *read_5(S,B, {pk(A),A}sk(S));* | *read_4(B,S, B,A);* |
| *read_6(B,A, {Na,Nb,B}pk(A));* | *send_6(B,A, {Na,Nb,B}pk(A));* | *send_5(S,B, {pk(A),A}sk(S));* |
| *send_7(A,B, {Nb}pk(B));* | *read_7(A,B, {Nb}pk(B));* | |

**Bezpečnostní požadavky**

Stejné požadavky jako pro protokol Needham – Schroeder Public key.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *claim_A1(A,Secret,Na);* | *claim_B1(B,Secret,Na);* | *No claims* |
| *claim_A2(A,Secret,Nb);* | *claim_B2(B,Secret,Nb);* | |
| *claim_A3(A,Nisynch);* | *claim_B3(B,Nisynch);* | |
| *claim_A4(A,Niagree);* | *claim_B4(B,Niagree);* | |

**Výsledek verifikace**

Lowův modifikovaný Needham – Schroeder Public key protokol nevyhovuje požadavkům na synchronizaci. Útočník může v kroku 2 a 5 přehrát agentům staré zprávy, a tak nahrazovat roli důvěryhodného serveru. K získání citlivějších informací ale tato činnost nevede.

**Závěr:** Lowův mod. Needham – Schroeder Public key protokol je po verifikaci nástrojem Scyther považován za **bezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 22.

## 6.10 PBK – Purpose Built Keys protokol

**Zápis protokolu**

1. $A \rightarrow B : A, KPa, hash(KPa)$
2. $A \rightarrow B : \{X\}KSa, hash(KPa)$
3. $B \rightarrow A : Nb$
4. $A \rightarrow B : \{Nb\}KSa$

**Popis**

Purpose Built Keys protokol [42] (dále jen PBK) na rozdíl od většiny popsaných protokolů v této práci nevyužívá důvěryhodného serveru nebo obecně třetí strany k uložení či kontrole šifrovacích klíčů. Je využíváno dočasných veřejných a soukromých klíčů, které jsou generovány vykonavateli protokolu, kdykoliv je potřeba udržovat/navázat autentizované spojení. Protokol je prováděn koncovými stanicemi (systémy) a jeho výhodou je jednoduchá škálovatelnost díky absenci centrálních ověřovacích prvků (serverů) a další podpůrné infrastruktury. PBK protokol byl vytvořen pro použití spolu s protokoly transportní a aplikační vrstvy.

Předtím než *Alice* začne navazovat spojení, vytvoří si veřejný a soukromý klíč, které bude používat v následujícím běhu protokolu (odtud název Purpose Built Keys). Dále si vytvoří PBID (Purpose Built ID) v podobě hashe z veřejného klíče *PKa*. PBID slouží k identifikaci příslušného iniciátora spojení a ke kontrole jeho veřejného klíče. Následně *Alice* naváže spojení zasláním své identity (IP adresa), veřejného klíče $KPa$[20] a PBID. *Bob* v roli příjemce uloží do tabulky IP adresu, PBID a veřejný klíč *KPa*.

Pokud chce *Alice* provést operaci vyžadující potvrzení identity, zašle žádost *X* šifrovanou soukromým klíčem *SKa* spolu s PBID. Po obdržení žádosti ověří *Bob* její pravost pomocí uloženého veřejného klíče *PKa* a vyšle challenge packet obsahující náhodné testovací číslo (nazveme nonce) *Nb*.

[20] Tento klíč může být zaslán při inicializaci spojení nebo kdykoliv poté, kdy je vyžadováno potvrzení identity iniciátora spojení. Pro jednodušší modelování ve Scytheru bude posílán v první zprávě při inicializaci spojení.

*Alice* zašifruje příslušné nonce *Nb* svým soukromým klíčem *Ksa* a pošle zpět. *Bob* po obdržení odpovědi zkontroluje *Nb* pomocí uloženého klíče *KPa*. Pokud hodnota *Nb* odpovídá předešlému challenge, je žádost schválena.

Po skončení komunikace jsou klíče zahozeny, existuje však pár vyjímek, kdy jsou klíče udržovány po více relací.

### Specifikace jednotlivých rolí v nástroji Scyther

```
# Iniciator – agent Alice
var Nb:  Nonce;
const X: Message;

send_1(A,B, A, pk(A), hash(pk(A)) );
send_2(A,B,{X}sk(A),hash(pk(A)) );
read_3(B,A, Nb);
send_4(A,B, {Nb}sk(A));
```

```
# Responder – agent Bob
const Nb: Nonce;
var X: Message;

read_1(A,B, A, pk(A), hash(pk(A) ));
read_2(A,B,{X}sk(A),hash(pk(A)) );
send_3(B,A, Nb);
read_4(A,B, {Nb}sk(A));
```

### Bezpečnostní požadavky

Protokol by měl zajistit utajení žádosti *X*.

### Specifikace bezpečnostních požadavků v nástroji Scyther

```
# Alice
claim_A1(A,Secret,X);
```

```
# Bob
claim_B1(B,Nisynch);
claim_B2(B,Niagree);
```

### Výsledek verifikace

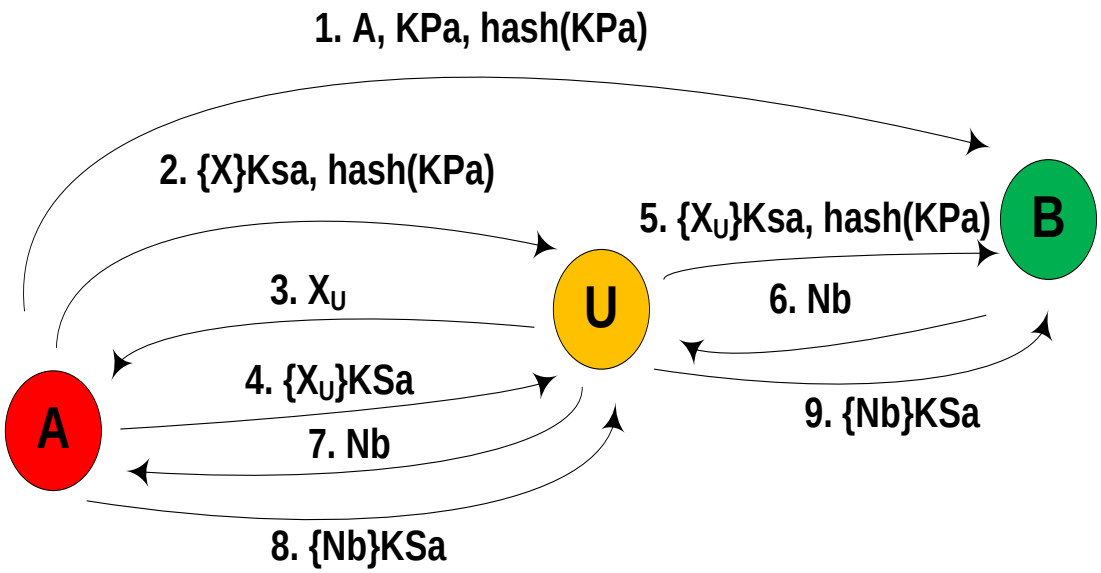
První problém, který zmíním, je část challenge/response v krocích 3 a 4. Zde *Alice* podepisuje přijatou výzvu v podobě *Nb*, aniž by věděla, od koho tato výzva přichází. Útočník si tak může nechat podepsat cokoliv potřebuje.

Toho lze využít k útoku **man–in-the-middle** - *Alice* začne běh protokolu standardní inicializací spojení posláním zprávy *A, KPa, hash(KPa)*. Následuje žádost o provedení operace *{X}KSa, hash(KPa)*, tuto zprávu však útočník odposlechne a pozdrží, načež odpoví *Alici* v přestrojení za *Boba* posláním své zprávy $X_U$. *Alice* nekontroluje původ zprávy, a tak ji podepíše svým soukromým klíčem *Ksa*. Útočník má nyní k dispozici podvrhnutou žádost $\{X_U\}KSa$ podepsanou legitimním agentem, ke které připojí *hash(KPa)* odposlechnutý v kroku 1 či 2. Výslednou zprávu zašle *Bobovi*, který ověří její platnost pomocí veřejného klíče uloženého v databázi. V případě kladné autentizace zahájí fázi challenge/response zasláním nonce *Nb Alici*. Toto nonce je opět odposlechnuto útočníkem a přeposláno tak, aby *Alice* odpověděla podepsaným *{Nb}KSa* na adresu útočníka. Protokol je dokončen doručením *{Nb}KSa Bobovi*, čímž útočník získal přístup k žádané operaci.

**Závěr:** Analýza v nástroji Scyther ukázala, že protokol PBK není schopen zabránit podvržení přenášeného tajemství *X*. Také čerstvost zpráv není nijak zaručena, proto nelze zabránit vkládání zpráv mimo pořadí a z toho vycházejícím útokům.

Protokol Purpose Built Keys je po verifikaci nástrojem Scyther považován za **nebezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 23.

**Grafické zobrazení útoku**




**Obrázek 11 - PBK protokol – útok man-in-the-middle**

## 6.11 Protokol Woo and Lam Pi f

**Zápis protokolu**

1. *A → B : A*
2. *B → A : Nb*
3. *A → B : {A, B, Nb}Kas*
4. *B→ S : {A, B, Nb, {A, B, Nb} Kas}Kbs*
5. *S → B : {A, B, Nb}Kbs*

**Popis**

Protokol Woo and Lam Pi f od autorů Thomase Y.C. Woo a Simona S. Lam [41] vznikl jako první návrh protokolu pro jednosměrnou autentizaci s využitím symetrických klíčů a důvěryhodného serveru. Ve verzi f byla posílena ochrana proti útoku **volbou otevřeného textu** (chosen plaintext attack), a to dvojitým šifrováním zprávy kroku 4 pomocí klíčů *Kas* a *Kbs*. Takovéto „vrstvené" šifrování vyjadřuje hierarchický vztah žádostí o překlad klíčů zaslaných na server.

Protokol je prováděn jednoduchým způsobem, *Alice* se ohlásí *Bobovi* s žádostí o navázání komunikace. *Bob* odpoví vytvořením nonce *Nb*. *Alice* následně vytvoří tiket pomocí svého symetrického klíče *Kas*. Tento tiket je konkatenován se zprávou od *Boba*, výsledek je znovu zašifrován klíčem *Kbs* a zaslán *Serveru*. Pokud souhlasí nonce *Nb* v tiketu a v části od *Boba*, *Server* potvrdí indetitu *Alice* potvrzující zprávou pro *Boba*.

**Specifikace jednotlivých rolí v nástroji Scyther**

| *# Iniciator – agent Alice* | *# Responder – agent Bob* | *# Role Serveru* |
|---|---|---|
| *var Nb: Nonce;* | *const Nb: Nonce;* | *var Nb: Nonce;* |
| | *var T: Ticket;* | |
| *send_1(A,B, A);* | | *read_4(B,S, {A,B,Nb,* |
| *read_2(B,A, Nb);* | *read_1(A,B, A);* | *{A,B,Nb}k(A,S)}k(B,S));* |
| *send_3(A,B, {A,B,Nb}k(A,S));* | *send_2(B,A, Nb);* | *send_5(S,B, {A,B,Nb}k(B,S));* |
| | *read_3(A,B, T);* | |
| | *send_4(B,S, {A,B,Nb,T}k(B,S));* | |
| | *read_5(S,B, {A,B,Nb}k(B,S));* | |

**Bezpečnostní požadavky**

Protokol pracuje správně, pokud *Bob* přijal zprávu od *Alice* v kroku 4 a zároveň v tom samém běhu obdrží potvrzující zprávu *{A, B, Nb}Kbs* v kroku 6. Důraz tedy bude kladen na správnou synchronizaci zpráv v protokolu.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *No claims* | *claim_B1(B,Nisynch);* | *claim_S1(S,Nisynch);* |
| | *claim_B2(B,Niagree);* | *claim_S2(S,Niagree);* |

**Výsledek verifikace**

Scyther odhalil slabinu v sychronizaci zpráv protokolu Woo and Lam Pi f. Problém je v kroku 3, kdy *Alice* zasílá *Bobovi* takzvaný tiket. *Bob* samozřejmě nezná obsah přijatého tiketu, a to spolu se znalostí nonce *Nb* poskytuje útočníkovi příležitost libovolně kombinovat odposlechnuté zprávy z jiných běhů protokolu. Pokud bychom zde řekli, že to je vše, co útočník zná, můžeme i tak protokol

Woo and Lam Pi f prohlásit za bezpečný. Špatná synchronizace zpráv v kroku 3 totiž nevede k úspěšnému útoku, pokud vezmeme v úvahu perfektní šifrovací metody.

**Závěr:** Protokol Woo and Lam Pi f je po verifikaci nástrojem Scyther považován za **bezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 24.

## 6.12 Protokol Woo and Lam Pi

### Zápis protokolu

1. $A \rightarrow B : A$
2. $B \rightarrow A : Nb$
3. $A \rightarrow B : \{Nb\}Kas$
4. $B \rightarrow S : \{A, \{Nb\} Kas\}Kbs$
5. $S \rightarrow B : \{Nb\}Kbs$

### Popis

Tento protokol vznikl zjednodušením původního Woo and Lam Pi f [41] a jeho dalších verzí (Woo and Lam Pi 1, 2, 3). Zajišťuje jednosměrnou autentizaci s využitím důvěryhodného serveru a symetrických klíčů.

Důkladnější popis spolu se známými útoky lze najít v části 4.11.

### Specifikace jednotlivých rolí v nástroji Scyther

| *# Iniciator – agent Alice* | *# Responder – agent Bob* | *# Role Serveru* |
|---|---|---|
| *var Nb: Nonce;* | *const Nb: Nonce;* | *var Nb: Nonce;* |
| | *var T: Ticket;* | |
| *send_1(A,B, A);* | | *read_4(B,S, {A, {Nb}k(A,S)}k(B,S));* |
| *read_2(B,A, Nb);* | *read_1(A,B, A);* | *send_5(S,B, {Nb}k(B,S));* |
| *send_3(A,B, {Nb}k(A,S));* | *send_2(B,A, Nb);* | |
| | *read_3(A,B, T);* | |
| | *send_4(B,S, {A,T}k(B,S));* | |
| | *read_5(S,B, {Nb}k(B,S));* | |

### Bezpečnostní požadavky

Požadavky jsou stejné jako pro protokol Woo and Lam Pi f.

### Specifikace bezpečnostních požadavků v nástroji Scyther

| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *No claims* | *claim_B1(B,Nisynch);* | *claim_S1(S,Nisynch);* |
| | *claim_B2(B,Niagree);* | *claim_S2(S,Niagree);* |

**Výsledek verifikace**

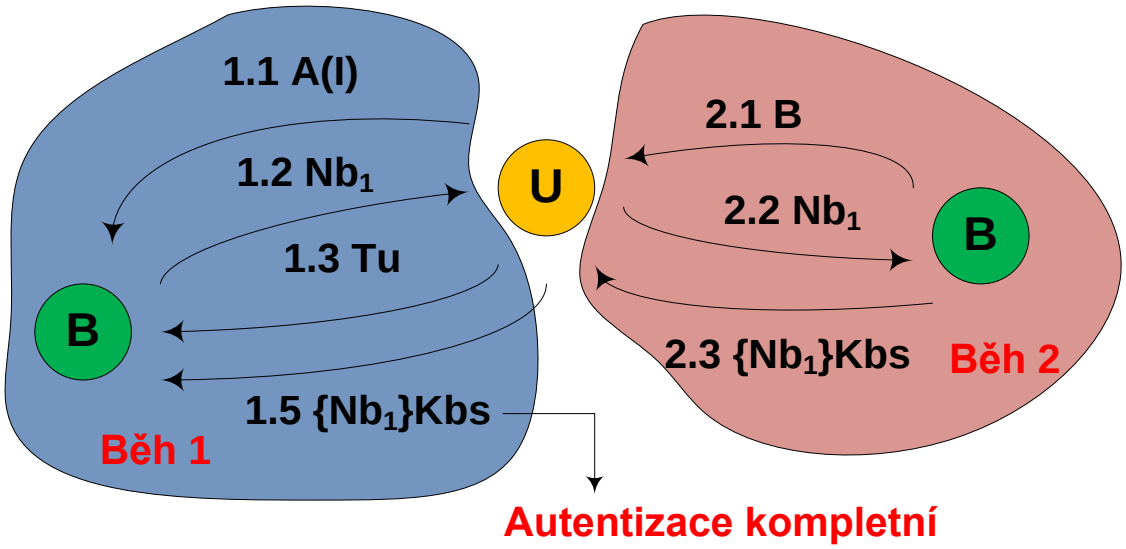
Zde se projevila chyba synchronizace přenesená z původního Woo and Lam Pi f protokolu popsaná v části 6.12. Útočník se autentizuje v přestrojení za jednoho z účastníků pomocí dvou paralelních běhů protokolu. Chyba je v kroku 5, kdy zpráva zaslaná *B* od serveru neobsahuje identitu iniciátora autentizace. Pokud se podíváme zpět na původní protokol Woo and Lam Pi f, právě existence *A* ve zprávě *{A, B, Nb}Kbs* zabránila úspěšnému útoku.

**Útok přehráním zprávy –** útočník se maskuje jako *Alice* a zasláním výzvy pro *Boba* začně běh 1. Ten odpoví posláním čerstvě vygenerovaného nonce $Nb_1$ . Mezitím *Bob* sám začíná běh 2 s jiným uživatelem. Útočník přepošle nonce $Nb_1$ do běhu 2, čímž získá zprávu $\{Nb_1\}Kbs$. Paralelně v běhu 1 útočník vytvoří tiket *Tu*. *Bob* přepošle tiket v další zprávě na server a vyčkává na odpověď. Ta je mu přeposlána z běhu 2 a tím je autentizace útočníka kompletní.

Pozn.: SPORE [21] neuvádí žádný známý útok, přesto jeden byl publikován [41]. Zde je útočník legitimním uživatelem systému s vlastním klíčem *Kus*. Náš útok se svým průběhem neliší, pouze není nutné, aby útočník byl uživatelem systému (díky posílanému tiketu v 3. kroku, kdy *Bob* neví, jakou identitu zpráva obsahuje).

**Závěr:** Protokol Woo and Lam Pi je po verifikaci nástrojem Scyther považován za **nebezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 25.

**Grafické zobrazení útoku**




**Obrázek 12 - Woo and Lam Pi – útok přehráním**

## 6.13 Protokol Yahalom

### Zápis protokolu

1. $A \rightarrow B : A, Na$
2. $B \rightarrow S : B, \{A, Na, Nb\}Kbs$
3. $S \rightarrow A : \{B, Kab, Na, Nb\}Kas, \{A, Kab\}Kbs$
4. $A \rightarrow B : \{A, Kab\}Kbs, \{Nb\}Kab$

### Popis

Protokol Yahalom [4] je jedním z rodiny protokolů pracujících na bázi symetrické kryptografie. Jeho úkolem je distribuce sdíleného relačního klíče mezi oba účastníky komunikace. Vydání tohoto klíče má na starosti třetí strana, vystupující zde jako důvěryhodný server.

Protokol je dokončen ve čtyřech zprávách. V první se ohlásí iniciátor komunikace agent *A* svému protějšku, se kterým chce komunikovat. Agent *B* složí zprávu pro důvěryhodný server, obsahující nonce *Na* a *Nb* pro ověření čerstvosti komunikace. To celé je zašifrováno symetrickým klíčem agenta *B* a serveru *S*. Třetí zpráva už obsahuje relační klíč *Kab*, agent *A* jej získá z první části po ověření čerstvosti pomocí *Na*. Druhá část je pro něj nečitelná, proto jí okopíruje do zprávy 4 a přidá potvrzení pro agenta *B*, že je to opravdu on, kdo inicioval celou komunikaci.

### Specifikace jednotlivých rolí v nástroji Scyther

```
# Iniciator – agent Alice
const Na: Nonce;
var Nb: Nonce;
 var Kab: RelKlic;

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

```
# Responder – agent Bob
const Nb: Nonce;
var Na: Nonce;
var Kab: RelKlic;

read_1(A,B, A,Na);
send_2(B,S, B, {A,Na,Nb}k(B,S) );
read_4(A,B, {A,Kab}k(B,S) , {Nb}Kab );
```

```
# Role Serveru
var Na,Nb: Nonce;
const Kab: RelKlic;

read_2(B,S, B, {A,Na,Nb}k(B,S) );
send_3(S,A, {B,Kab,Na,Nb}k(A,S),
{A, Kab}k(B,S) );
```

### Bezpečnostní požadavky

Protokol musí zajistit, aby relační klíč *Kab* nebyl v průběhu výměny nijak kompromitován. Všichni zúčastnění si musí být vědomi, s kým komunikují, a že se účastní stejného běhu protokolu.
Budeme tedy ověřovat bezpečnost relačního klíče *Kab* a obou nonce *Na*, *Nb*.

Pokud se podíváme na protokol detailněji, můžeme si všimnout jedné potenciální bezpečnostní slabiny. Tou je oddělení *Kab* a nonce *Nb* v kroku 4. Agent *B* totiž obdrží zprávu s relačním klíčem *Kab*, ale ta neobsahuje žádný důkaz o tom, že je aktuální. Aktuálnost nebo chcete-li čerstvost je potvrzena až s přijetím *Nb*. Pokud tedy útočník bude schopen získat nonce *Nb*, je tím

zkompromitován i klíč *Kab*. Proto při verifikaci budeme klást na stejnou úroveň jak otázku bezpečnosti klíče *Kab*, tak nonce *Nb*.

**Specifikace bezpečnostních požadavků v nástroji Scyther**

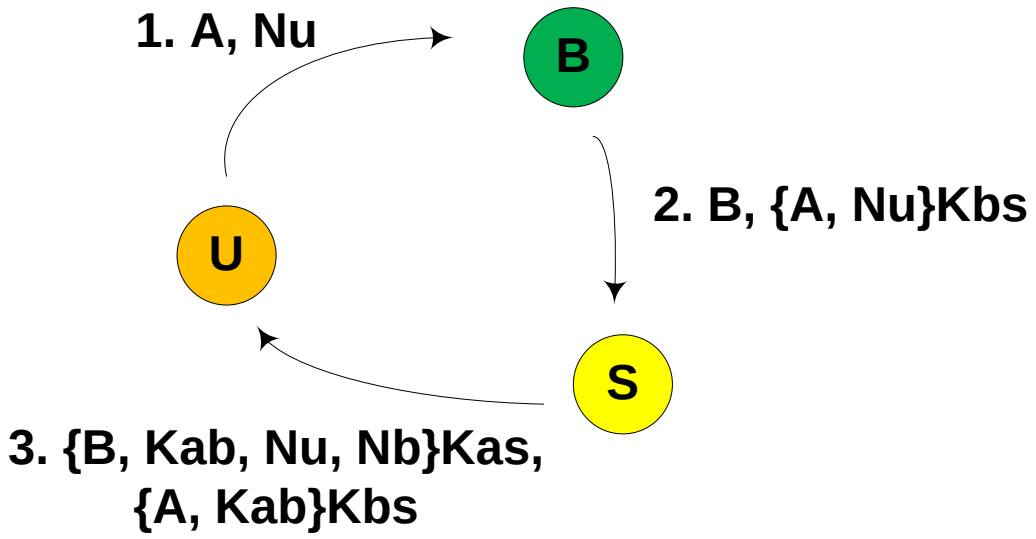
| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *claim_A1(A,Secret,Kab);* | *claim_B1(B, Secret,Kab);* | *claim_S1(S, Secret, Na);* |
| *claim_A2(A,Secret,Na);* | *claim_B2(B, Secret, Na);* | *claim_S2(S, Secret, Nb);* |
| *claim_A3(A,Secret,Nb);* | *claim_B3(B, Secret, Nb);* | |
| *claim_A4(A,Nisynch);* | *claim_B4(B,Nisynch);* | |
| *claim_A5(A,Niagree);* | *claim_B5(B,Niagree);* | |

**Výsledek verifikace**

Nástroj Scyther nenalezl žádný útok na relační klíč *Kab* a nonce *Nb*. Jediné co nemůže protokol zaručit, je bezpečnost nonce *Na*. Útočník je schopen toto nonce odposlechnout z prvního kroku protokolu a poté získat zpět zašifrované ze serveru. Může se tak maskovat jako iniciátor komunikace použitím starších zpráv, nicméně toto jednání nevede k získání jak *Kab*, tak *Nb*, protože útočník nezná dešifrovací klíč *Kas* či *Kbs*.

**Závěr:** Protokol Yahalom je po verifikaci nástrojem Scyther považován za **bezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 26.
Pozn.: Na protokol Yahalom nebyl zatím nalezen žádný útok kompromitující klíč *Kab*. Ověření bezpečnosti protokolu pomocí nástroje Isabelle lze najít zde [31].

**Grafické zobrazení útoku**




**Obrázek 13 - Protokol Yahalom – útok na nonce Na**

## 6.14 BAN modifikovaný protokol Yahalom

### Zápis protokolu

1. $A \rightarrow B : A, Na$
2. $B \rightarrow S : B, Nb, \{A, Na\}Kbs$
3. $S \rightarrow A : Nb, \{B, Kab, Na\}Kas, \{A, Kab, Nb\}Kbs$
4. $A \rightarrow B : \{A, Kab, Nb\}Kbs, \{Nb\}Kab$

### Popis

Tato verze protokolu Yahalom byla publikována autory Burrows, Abadi a Needham [4]. Po provedení analýzy BAN logikou, tito autoři navrhli změny vedoucí k odstranění náchylnosti původního protokolu vůči použití starých zpráv z kroku 4 (viz protokol Yahalom).

Hlavní změna přichází v tom, že nonce *Nb* není v kroku 2 posíláno jako šifrované tajemství, ale jako plain text. Už tady vidíme problém a tím je použití odposlechnutého nonce k útoku přehráním.
Jak už bylo řečeno, agent *B* posílá autentizačnímu serveru nešifrované nonce *Nb* a identitu agenta *A*. *Server* zopakuje *Nb*, vloží zprávu s relačním klíčem *Kab* spolu se zprávou pro agenta *B*. V posledním kroku agent *A* přepošle zprávu pro *B* a potvrdí svoji účast v současné komunikaci *Nb* šifrované obdrženým relačním klíčem.

### Specifikace jednotlivých rolí v nástroji Scyther

```
# Iniciator – agent Alice
const Na: Nonce;
var Nb: Nonce;
var Kab: RelKlic;

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

```
# Responder – agent Bob
const Nb: Nonce;
var Na: Nonce;
var Kab: RelKlic;

read_1(A,B, A,Na);
send_2(B,S, B, Nb, {A,Na}k(B,S) );
read_4(A,B ,{A,Kab,Nb}k(B,S),
{Nb}k(A,B) );
```

```
# Role Serveru
const Kab: RelKlic;
var Na,Nb: Nonce;

read_2(B,S, B,Nb, {A,Na}k(B,S));
send_3(S,A, Nb, {B, Kab, Na}k(A,S),
 {A, Kab, Nb}k(B,S));
```

### Bezpečnostní požadavky

Stejně jako v původní verzi protokolu [4] musí být zajištěn bezpečný přenos relačního klíče *Kab*. Dále budeme zkoumat, zda je možné získat nonce *Nb* při odposlechnutí běhu protokolu. I když došlo k modifikaci a nyní je posíláno nonce *Nb* spolu s relačním klíčem (absence *Nb* v tiketu šifrovaném *Kbs* u kroku 4 byla předmětem kritiky u originálního protokolu), je zkompromitování *Nb* pro bezpečnost protokolu stále velký problém. Pokud budeme uvažovat trochu dopředu a útok na nonce *Nb* se potvrdí, bude to znamenat, že útočník je schopen přehrát starší zprávy a tím podvrhnout relační klíč *Kab*.

Na druhé straně nebudeme zkoumat bezpečnost *Na*, jeho role se od původní verze protokolu nijak nezměnila, a proto lze předpokládat nalezení útoku na *Na*.

### Specifikace bezpečnostních požadavků v nástroji Scyther

| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *claim_A1(A, Secret, Kab);* | *claim_B1(B, Secret, Kab);* | *Claim_S1(S, Secret, Na);* |
| *claim_A2(A, Nisynch);* | *claim_B2(B, Nisynch);* | *Claim_S2(S, Secret, Nb);* |
| *claim_A3(A, Niagree);* | *claim_B3(B, Niagree);* | |
| *claim_A4(A, Secret, Na);* | *claim_B4(B, Secret, Na);* | |
| *claim_A5(A, Secret, Nb);* | *claim_B5(B, Secret, Nb);* | |

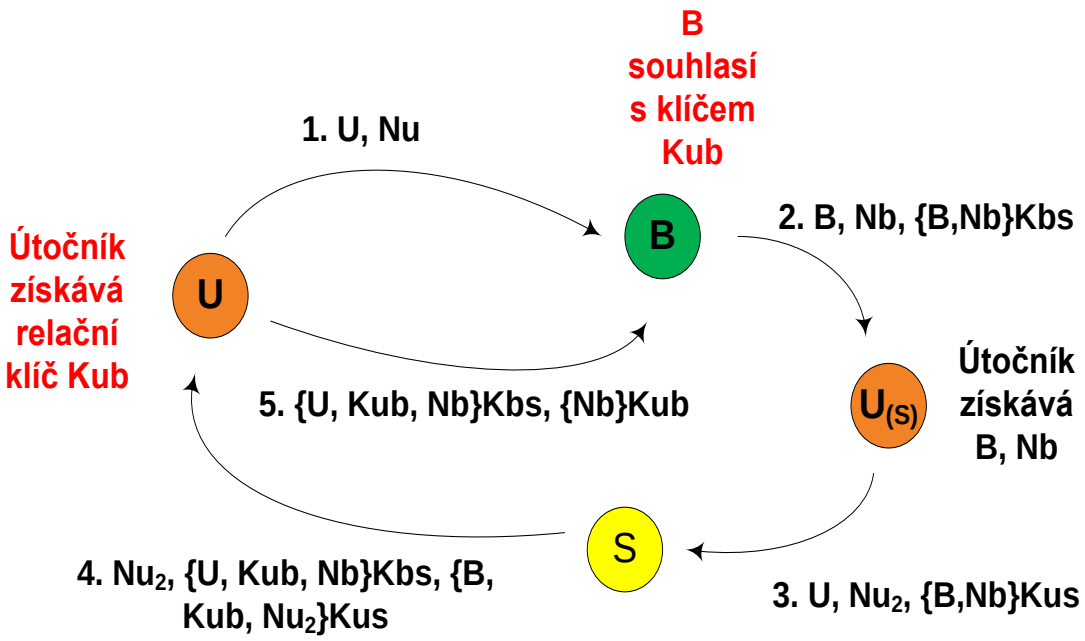
### Výsledek verifikace

Protokol je sice schopen zajistit bezpečnost relačního klíče *Kab*, ale bohužel díky přenosu nezašifrovaného *Nb* je útočník schopen odposlechnout toto nonce a tím získat předpoklad pro úspěšný útok přehráním.

**Útok přehráním s prokládáním zpráv** - útočník (označíme jako *U*) v prvním kroku zašle agentu *B* svoje nonce *Nu* a z jeho odpovědi *Serveru* získá odposlechnutím identitu *B* a nonce *Nb*. Tuto informaci zašifruje svým soukromým klíčem *Kus*, vygeneruje nové nonce $Nu_2$ a výslednou zprávu *U, $Nu_2$ ,{B, Nb}Kus* zašle důvěryhodnému serveru *S*. Ten podle pravidel protokolu vytvoří zprávu obsahující certifikáty šifrované soukromými klíči obou účastníků. Útočník takto jednoduše získá tajemství obsahující relační klíč *Kub*, který je šifrován jeho soukromým klíčem *Kus*, a je tedy schopen ho rozluštit. Nyní už jenom zbývá doručit *Kub* zpátky *B*, což se děje v posledním kroku. Díky tomu, že je čerstvost komunikace potvrzena nonce *Nb*, nemá agent *B* důvod nevěřit relačnímu klíči *Kub*.

**Závěr:** BAN modifikovaný Yahalom je po verifikaci nástrojem Scyther považován za **nebezpečný**. Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 27.

**Grafické zobrazení útoku**




**Obrázek 14 - BAN modifikovaný Yahalom – útok přehráním**

## 6.15 Lowova verze protokolu Yahalom

**Zápis protokolu**

1. *A → B: A, Na*
2. *B → S: (A, Na, Nb)Kbs*
3. *S → A: (B, Kab, Na, Nb)Kas*
4. *S → B: (A, Kab)Kbs*
5. *A → B: (A, B, S, Nb)Kab*

**Popis**

Garwin Lowe představil ve své práci [32] další modifikaci originálního protokolu Yahalom [4]. Lowe definuje několik podmínek, které musí být splněny, aby protokol byl bezpečný. Zmiňme některé podmínky, které jsou pro náš protokol nejdůležitější:

- *Odvoditelné identity (Identities inferable)* – šifrované části musí obsahovat všechny role v protokolu mezi svými volnými proměnnými. Příklad: V druhém kroku není nutno posílat *B, {A, Na, Nb}Kbs* (v originále), protože agent *S* je schopen poznat identitu *B* z použitého klíče *Kbs* (zpráva obsahuje všechny identity *A, Kbs = B,S*).

- *Žádná dočasná tajemství (No temporary secrets)* – při běhu protokolu se nevyužívá žádného dočasného tajemství, tzn. pokud nějaká část zprávy není považována za tajemství, je kdokoliv schopen se toto dozvědět, jakmile se objeví v běhu protokolu.

Nejdůležitější změnou je neposílání tiketu *{A, Kab}Kbs* iniciátorovi *Alici*, ale je rovnou poslán *Bobovi*. *Alice* tak nemusí přeposílat zprávy, které není schopna přečíst.
Poslední krok se řídí podmínkou *odvoditelných identit*. Jsou zaslány identity *A,B,S* , čímž se posilňuje zabezpečení protokolu (i když posílání *A, B* se může zdát zbytečně redundantní, někdy může vynechání identit vést k útoku na protokol).

### Specifikace jednotlivých rolí v nástroji Scyther

| *# Iniciator – agent Alice* | *# Responder – agent Bob* | *# Role Serveru* |
|---|---|---|
| *const Na: Nonce;* | *const Nb: Nonce;* | *const Kab: RelKlic;* |
| *var Nb: Nonce;* | *var Na: Nonce;* | *var Na,Nb: Nonce;* |
| *var Kab: RelKlic;* | *var Kab: RelKlic;* | |
| | | *read_2(B,S, {A,Na,Nb}k(B,S) );* |
| *send_1(A,B, A,Na);* | *read_1(A,B, A,Na);* | *send_3(S,A, {B,Kab,Na,Nb}k(A,S) );* |
| *read_3(S,A, {B,Kab,Na,Nb}k(A,S) );* | *send_2(B,S, {A,Na,Nb}k(B,S) );* | *send_4(S,B, {A,Kab}k(B,S));* |
| *send_5(A,B, {A, B, S, Nb}Kab );* | *read_4(S,B, {A,Kab}k(B,S));* | |
| | *read_5(A,B, {A, B, S, Nb}Kab);* | |

### Bezpečnostní požadavky

Protokol musí zachovat tajemství přenášeného relačního klíče *Kab*, stejně tak musí zajistit neprozrazení nonce *Nb*. Nonce *Na* není nijak chráněno a je stejně jako v předchozích verzích posíláno nezašifrované při inicializaci komunikace.

### Specifikace bezpečnostních požadavků v nástroji Scyther

| *# Alice* | *# Bob* | *# Server* |
|---|---|---|
| *claim_A1(A, Secret,Kab);* | *claim_B1(B, Secret,Kab);* | *claim(S, Secret, Nb);* |
| *claim_A2(A, Nisynch);* | *claim_B2(B, Nisynch);* | |
| *claim_A3(A, Niagree);* | *claim_B3(B, Niagree);* | |

### Výsledek verifikace

Na Lowův modifikovaný Yahalom protokol nebyl nalezen žádný útok.

**Závěr:** Lowův modifikovaný Yahalom je po verifikaci nástrojem Scyther považován za **bezpečný**.

# 7 Víceprotokolové útoky – Multiprotocol attacks

Analýza bezpečnostních protokolů pomocí automatických nástrojů přináší jak nové poznatky, tak časové zpřístupnění složitějších analýz u komplexnějších protokolů. Takto získaný čas a ušetřené úsilí je pak možné soustředit do jiných směrů a oblastí. V předchozí kapitole byl předveden nástroj Scyther pro automatickou analýzu bezpečnostních protokolů. Ukázal se jako dobrý prostředek k odhalování známých i neznámých útoků na jednotlivé protokoly.

Tím to ale nekončí. Těžko si lze představit, že na současné globalizované počítačové síti je používáno vždy jednoho bezpečnostního protokolu v izolaci. Díky stovkám miliónů počítačových stanic a serverů běží paralelně na dnešní počítačové síti nespočet různých typů bezpečnostních protokolů. Předchozí výzkumy se ale soustředily pouze na analýzu jednotlivých protokolů v izolovaném prostředí a bez vlivů okolí. Proto si nutně musíme položit otázku: Je možné, že bezpečnostní protokoly považované za bezpečné, mohou být úspěšně napadeny za pomoci ostatních bezpečnostních protokolů běžících souběžně na počítačové síti?

Právě tuto otázku se budeme snažit alespoň částečně zodpovědět v této kapitole. Pro testování byly vybrány bezpečnostní protokoly s podobnou strukturou posílaných zpráv, přičemž alespoň jeden protokol z testované dvojice je považován za bezpečný (podle dostupných zdrojů nebo analýzy v této diplomové práci). Zvolená testovací sada protokolů byla testována v nástroji Scyther s následujícími parametry (viz část 5.3.5):

- Max.number of runs – 10
- Matching type – typed matching
- Max. number of patterns per claim – 10

Uvedeny jsou jen příklady nalezených útoků na zvolenou sadu bezpečnostních protokolů. Zdrojový kód pro jednotlivé dvojice protokolů nebude uveden, je totiž výsledkem konkatenace jednotlivých protokolů. Celkové výsledky všech verifikací z kapitol 6 a 7 jsou uvedeny v tabulce na konci kapitoly 7.

## 7.1 Protokol Yahalom a Lowova verze protokolu Yahalom

**Zápis protokolu Yahalom**

1. *A → B : A, Na*
2. *B → S : B, {A, Na, Nb}Kbs*
3. *S → A : {B, Kab, Na, Nb}Kas, {A, Kab}Kbs*
4. *A→ B : {A, Kab}Kbs, {Nb}Kab*

**Zápis Lowovi verze protokolu Yahalom**

1. *A → B: A, Na*
2. *B → S: (A, Na, Nb)Kbs*
3. *S → A: (B, Kab, Na, Nb)Kas*
4. *S → B: (A, Kab)Kbs*
5. *A → B: (A, B, S, Nb)Kab*

**Popis protokolů**

Oba protokoly jsou považovány za bezpečné. Nástroj Scyther nenalezl po verifikaci jednotlivých protokolů v izolaci žádný možný útok, který by vedl k prozrazení relačního klíče *Kab*. Díky správné synchronizaci zpráv protokolů nebylo také možné získat dodatečné informace přehráním starších zpráv.

**Výsledek verifikace**

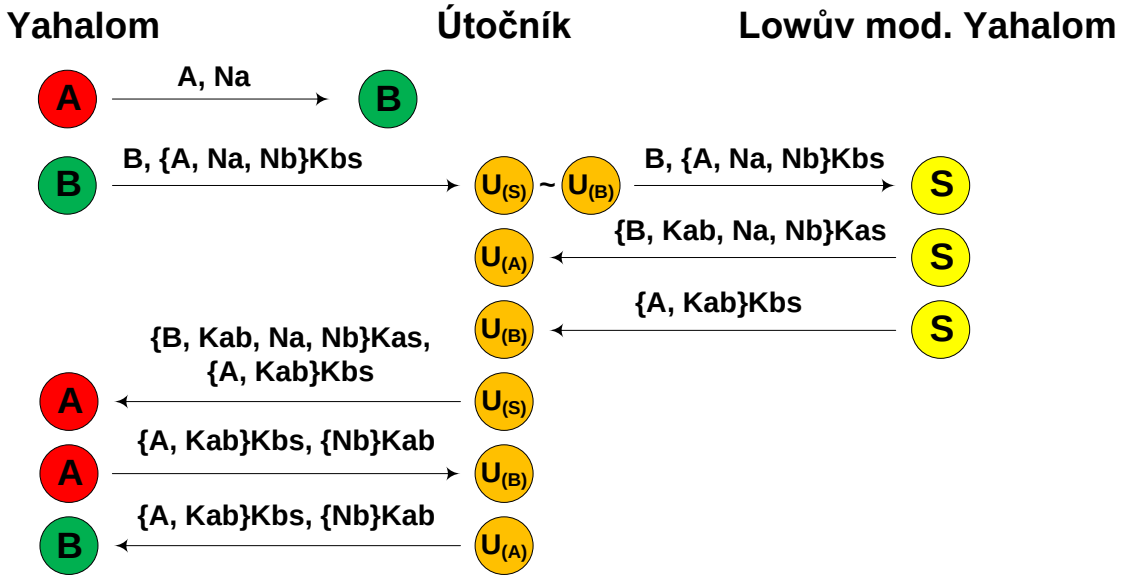
Používání obou verzí Yahalom protokolu může vést k úspěšnému **útoku přehráním.** Díky tomu, že oba protokoly využívají totožný typ zpráv, může útočník využít této skutečnosti k přeposlání starších zpráv a tím podstrčit agentům zkompromitovaný relační klíč.

**Útok přehráním** – *Alice* začíná protokol Yahalom posláním zprávy *A, Na Bobovi*. *Bob* z přijatého nonce vytvoří zprávu *B, {A, Na, Nb}Kbs* a zašle ji na server v protokolu Yahalom. Zprávu útočník zachytí a přepošle *Serveru,* který vykonává Lowův mod. Yahalom. *Server* přijaté informace použije k vytvoření dvou zpráv, první část zašle *Alici* a druhou *Bobovi*. Tyto dvě části útočník složí do jedné zprávy pro *Alici*, druhou část si nadále ponechá. *Alice* dokončí protokol zasláním zprávy s relačním klíčem *Kab* a nonce *Nb*. Útočník z ní může vybrat část zašifrovanou relačním klíčem *Kab* a připojit ji k zapamatované části *{A, Kab}Kbs* .

Shrneme-li výsledek, zjistíme, že útočník je schopen přehrát starší komunikaci v kroku 2. Tím získá znalost týkající se kontrolních nonce, které nemohou zaručit čerstvost zpráv v daném běhu protokolu. Když není zaručena čerstvost zpráv, otevírá to možnost k dalšímu využití starších zpráv v krocích 3 a 4. Útočník tak může podvrhnout relační klíč *Kab* účastníkům protokolu Yahalom.

Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 28.

**Grafické zobrazení útoku**




**Obrázek 15 - Útok na protokoly Yahalom a Lowův mod. Yahalom**

## 7.2 Protokol Yahalom a Woo and Lam Pi

**Zápis protokolu Yahalom**

1. $A \rightarrow B : A, Na$
2. $B \rightarrow S : B, \{A, Na, Nb\}Kbs$
3. $S \rightarrow A : \{B, Kab, Na, Nb\}Kas, \{A, Kab\}Kbs$
4. $A \rightarrow B : \{A, Kab\}Kbs, \{Nb\}Kab$

**Zápis protokolu Woo and Lam Pi**

1. $A \rightarrow B : A$
2. $B \rightarrow A : Nb$
3. $A \rightarrow B : \{Nb\}Kas$
4. $B \rightarrow S : \{A, \{Nb\} Kas\}Kbs$
5. $S \rightarrow B : \{Nb\}Kbs$

**Popis protokolů**

Na protokol Yahalom není v literatuře znám žádný útok, což bylo potvrzeno analýzou protokolu v části 6.13. Protokol Woo and Lam Pi obsahuje chybu v kroku 5, která umožňuje provést **útok přehráním**, viz část 6.12 nebo [41].

Pozn.: Protokol Woo and Lam Pi je modelován s využitím *tiketu*, proto můžeme kroky 3 a 4 psát 3. $A \rightarrow B : T$ , respektive 4. $B \rightarrow S : \{A, T\}Kbs$ .

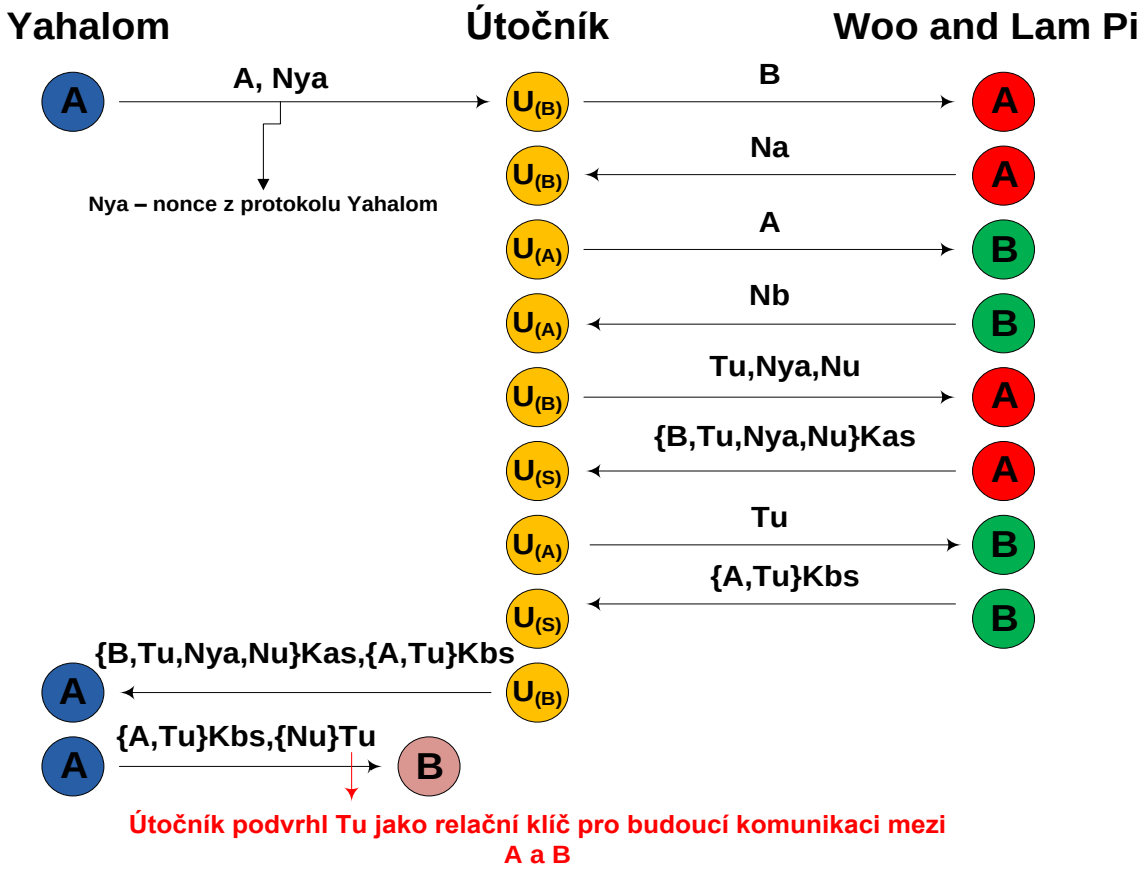
**Výsledek verifikace**

Pokud jsou paralelně prováděny protokoly Yahalom a Woo and Lam Pi, může to vést ke zkompromitování relačního klíče v protokolu Yahalom. Útočník je schopen podrvhnout protokolu Yahalom vlastní libovolně zvolený klíč, a tím může sledovat všechnu komunikaci mezi účastníky.

Útok **man-in-the-middle** – začněme protokolem Yahalom. *Alice* jako iniciátor zašle zprávu *A, Nya*. Mezitím útočník začne dva paralelní běhy protokolu Woo and Lam Pi. Po počáteční výměně nonce vytvoří útočník tiket *Tu* pro *Alici* a zprávu *Tu, Nya, Nu* pro *Boba*. Zprávy od obou agentů jsou zachyceny útočníkem a konkatenovány do výsledné zprávy *{B, Tu, Nya, Nu}Kbs, {A, Tu}Kas* , která je zaslána *Alici* v protokolu Yahalom. Ta očekává zprávu ve stejném tvaru, proto přijme podvržený tiket *Tu* jako relační klíč pro nadcházející spojení s *Bobem*. Ačkoliv se *Bob* vůbec nezůčastnil tohoto běhu protokolu Yahalom, je mu na konci zaslán zkompromitovaný relační klíč od *Alice* (útočník může jednoduše inicializovat běh protokolu Yahalom, na jehož konci *Bob* přijme zprávu s *Tu* od *Alice*).

Originální grafický výstup z nástroje Scyther je umístěn v příloze 3, obrázek 29.

**Grafické zobrazení útoku**




**Obrázek 16 - Útok na protokoly Yahalom a Woo and Lam Pi**

## 7.3 Prezentace výsledků analýzy bezpečnostních protokolů v nástroji Scyther

| Název protokolu | Pořadové číslo | Útoky | Nal. útoky v nás. Scyther[21] | Bezpečnost protokolu[22] | Multiprotokolové útoky[23] |
|---|---|---|---|---|---|
| **Adrew Secure RPC** | 1 | Útok přehráním | 3/0/0 | Nebezpečný | 1-2 |
| **BAN Andrew Secure RPC** | 2 | Není | 0/0/0 | Bezpečný | 2-1, 2-12 |
| **CHAPv2** | 3 | Není | 0/0/0 | Bezpečný | - |
| **CRAM – MD5** | 4 | Není | 0/0/0 | Bezpečný | - |
| **Denning – Sacco** | 5 | Násobný útok | 1/1/0 | Nebezpečný | 5-6 |
| **Lowe Denning - Sacco** | 6 | Není | 2/2/0 | Bezpečný | 6-5, 6-12, 6-14 |
| **EKE** | 7 | Man-in-the-middle | 2/2/0 | Nebezpečný | - |
| **Needham – Schroeder Public Key** | 8 | Man-in-the-middle | 2/4/0 | Nebezpečný | 8-9 |
| **Lowe Needham – Schroeder Public Key** | 9 | Není | 2/2/0 | Bezpečný | 9-8 |
| **PBK** | 10 | Man-in-the-middle | 1/2/0 | Nebezpečný | - |
| **Woo and Lam Pi f** | 11 | Není | 0/2/1 | Bezpečný | 11-12 |
| **Woo and Lam Pi** | 12 | Útok přehráním zprávy | 0/2/2 | Nebezpečný | 12-2 |
| **Yahalom** | 13 | Není | 1/1/1 | Bezpečný | 13-12,13-15, 13-15 |
| **BAN Yahalom** | 14 | Útok přehráním s prokládáním zpráv | 4/4/2 | Nebezpečný | - |
| **Lowe Yahalom** | 15 | Není | 0/0/0 | Bezpečný | 15-12, 15-15 |

**Tabulka 2. Prezentace výsledků**

[21] Počet nalezených útoků pro jednotlivé role, data vyjadřují výstup nástroje Scyther. Ve formátu A/B/S.
[22] Zhodnocení výsledků analýz izolovaných protokolů v kapitole 6.
[23] Vyjádřeno kombinací číselných označení jednotl. protokolů (př.: Yahalom a BAN Yahalom 14-15).

# 8 Závěr

Cílem mé diplomové práce byla analýza vybraných bezpečnostních protokolů pomocí zvoleného nástroje Scyther. Pro komplexnější pohled na danou problematiku jsem se věnoval vysvětlení některých pojmů z oblasti bezp. protokolů, spolu s úvodem do verifikačních logik. Jelikož analýza vybraných bezp. protokolů byla součástí praktické části diplomové práce, věnoval jsem teoretickému popisu relativně velký prostor. Případný čtenář nemusí být detailně seznámen s tématikou bezp. protokolů, považuji ji tedy za stěžejní v rámci teoretického úvodu.

Následuje hlubší analýza dostupných verifikačních nástrojů. Za cenný zdroj mi zde posloužil přehledně zpracovaný seznam verifikačních nástrojů z diplomové práce Ing. Michala Ptáčka [53], z něhož jsem záměrně opominul nástroje na dnešní dobu již poněkud zastaralé nebo nedostupné. Nakonec jsem vybral Isabelle pro jeho komplexnost, nástroj AVISPA pro jeho vedoucí postavení na poli automatické analýzy bezp. protokolů, a na závěr nástroj Scyther, který je relativně nový a neokoukaný.

Právě nástroj Scyther mi připadl jako vhodný kandidát pro analýzu bezpečnostních protokolů v praktické části. Díky svému relativně nedávnému zpřístupnění internetové komunitě neexistuje mnoho publikací na téma analýzy bezp. protokolů s využitím Scytheru. Jediné publikace o jeho použití pro analýzu bezp. protokolů pochází od autora Case Cremerse. Výsledky analýz ale už publikovány nebyly.

Scyther se ukázal jako schopný pomocník při analýze většího množství bezpečnostních protokolů. Díky ne příliš složitému programovacímu jazyku pro popis protokolů je uživatel schopen namodelovat příslušný protokol v otázce několika desítek minut. Všechno samozřejmě záleží na zkušenostech. Také grafický výstup znamená pro uživatele snadnější přístup k výsledkům analýz, přeci jenom dobrá představa o nalezeném útoku vede k rychlejšímu porozumění.

Ne vždy však Scyther poskytl jednoznačné výsledky, aby uživatel mohl říci, že zkoumaný protokol je bezpečný nebo naopak. Často je nutné prozkoumat nalezené útoky a porovnat s reálnou možností útočníka napadnout protokol s pomocí získaných znalostí.

K analýze jsem zvolil 15 různých bezpečnostních protokolů, z nichž 11 je čerpáno ze SPORE [21] a 4 jsou průmyslové protokoly uvedené v dokumentech RFC. Výběr protokolů nebyl čistě náhodný, volil jsem protokoly s různými vlastnostmi, jako je používání časových značek, inkrementů nonce nebo protokoly sloužící pro distribuci relačního klíče či k oboustranné autentizaci. Tím spolu s průmyslovými protokoly vznikl pestrobarevný vzorek protokolů, sloužících k demostraci možných útoků a slabin. Pro návrháře moderních bezp. protokolů mohou tyto analýzy sloužit k poučení se z minulých chyb a návrhu opravdu bezpečných protokolů.

V poslední kapitole uvádím příklady multiprotokolových útoků. Podle dostupných studií na internetu se zdá, že tato hrozba je poněkud opomíjena. Lidé se spokojí s ověřením bezpečnosti protokolů v izolovaném prostředí, ale už nepřikládají takovou váhu možnosti útoku na paralelně prováděné protokoly. V poslední době se ale zdá, že se multiprotokolové útoky dostávají do většího popředí zájmu. Důkazem toho je vyvíjení nástrojů jako je Scyther, které mohou pomoci multiprotokolové útoky analyzovat. V mé práci jsem se soustředil na analyzování dvojic bezpečnostních protokolů, z nichž vždy alespoň jeden je považován za bezpečný. Výsledkem je nalezení nemalého množství multiprotokolových útoků na bezpečné protokoly, běžící v sousedství jiných protokolů. Popsal jsem pouze dva příklady takovýchto útoků a zbytek uvedl v přehledové tabulce. Zde se nabízí prostor pro detailnější rozpracování dané problematiky s využitím výsledků této diplomové práce.

# 9 Literatura


[1] STALLINGS, W. *Cryptography and Network Security. Principles and Practice.* 2nd ed. New Jersey : Prentice-Hall, 1999. ISBN 0-13-869017-0.

[2] SCHNEIDER, S. A., RYAN, P. Y. A. *Modelling and Analysis of Security Protocols*. 1st ed. Addison Wesley, Boston, 2000, ISBN 0-201-67471-8.

[3] BISHOP, M. *Computer security: Art & Science*. Addison-Wesley, Boston, 2003, ISBN 0-201-44099-7.

[4] SINGH, S. *Kniha kódů a šifer*. 1st ed. Dokořán, Praha, 2003, ISBN 80-86569-18-7.

[5] ONDRÁČEK, D. *Databáze specifikací bezpečnostních protokolů*. [diplomová práce], FIT VUT v Brně, 2008.

[6] BURROWS, M., ABADI, M., NEEDHAM, R. A Logic of Authentication. *Technical Report 39* [online]. February 1989, [cit. 2009-11-17].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=77649>.

[7] KYNTAJA, T. A Logic of Authentication by Burrows, Abadi and Needham [online]. *Department of Computer Science, Helsinki University of Technology*, 1995, [cit. 2009-11-18].
Dostupné z WWW: <http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/ban.html>.

[8] SAFAVI-NAINI, R., MATHURIA, A. M., NICKOLAS, P. R. On the Automation of GNY Logic [online]. *Australian Computer Science Communications*, vol. 17, no. 3, December 1995, p. 370-379, [cit, 2009-11-22].
Dostupné z WWW: <http://eprints.kfupm.edu.sa/54733/1/54733.pdf>.

[9] COLOURIS, G., DOLLIMORE, J., KIDBERG, T. Archive Material from Edition 2 of Distributed Systems [online]. *Concepts and Design*, 1994, [cit. 2009-11-22].
Dostupné z WWW: <http://en.scientificcommons.org/42447582>

[10] GONG, L., NEEDHAM, R., YAHALOM, R. Reasoning about Belief in Cryptographic Protocols [online]. In *1990 IEEE Symposium on Security and Privacy,*1990, p. 234-248, [cit. 2009-11-24].
Dostupné z WWW: <http://eprints.kfupm.edu.sa/61143/>.

[11] ABADI, M. and TUTTLE, M. A Semantics for a Logic of Authentication [online]. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, ACM Press, August 1991, p. 201-216, [cit. 2009-11-22].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=112600.112618>.

[12] OORSCHOT, P.C. van. Extending Cryptographic Logics of Belief to Key Agreement Protocols [online]. In *Proceedings of the First ACM Conference on Computer and Communications Security*. November 1993, p. 232-243, [cit. 2009-11-24].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=168617>.

[13] SYVERSON, P. and OORSCHOT, P. C. van. A Unified Cryptographic Protocol Logic [online]. Technical Report 5540-227, *Naval Research Laboratory*, 1996, p. 1-29, [cit. 2009-11-26].
Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.7286>.
[14] KWON, T. and LIM, S. Automation-Considered Logic of Authentication and Key Distribution [online]. Lecture Notes. In *Computer Science*, Technical Report 442-457, Springer-Verlag, 2003, p. 1-15, [cit. 2009-11-26].
Dostupné z WWW: <www.springerlink.com/index/rxxw4kwa8ttg61ry.pdf>.
[15] LOWE, G. A Family of Attacks upon Authentication Protocols [online]. Technical Report 5, *Department of Mathematics and Computer Science*, *University of Leicester*, 1997, p. 1-11, [cit. 2009-11-26].
Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.9175>.
[16] DENNING, D., SACCO, G. Timestamps in Key Distributed Protocols [online]. *Communication of the ACM*, 1981, p. 533-535, [cit. 2009-11-28].
Dostupné z WWW: <http://faculty.nps.edu/dedennin/publications/TimestampsKeyDistribution.pdf>.
[17] WEIDENBACH, CH. Towards an Automatic Analysis of Security Protocols in First-Order Logic [online]. In *Harald Ganzinger*, *Proceedings of the 16th International Conference on Automated Deduction*, Springer-Verlag, 1999, vol. 1632, p. 378-382, [cit. 2009-11-29].
Dostupné z WWW: <http://www.springerlink.com/content/a5bd6kyj3994dklf/>.
[18] SHAIKH, S., BUSH, V. Analysing the Woo-Lam Protocol Using CSP and Rank Functions. *Journal of Research and Practice in Information Technology*, February 2006, vol. 38, no. 1, p. 19-29, [cit. 2009-12-1].
Dostupné z WWW: <http://www.jrpit.acs.org.au/jrpit/JRPITVolumes/JRPIT38/JRPIT38.1.19.pdf>.
[19] ANDERSON, R. J. *SecurityEngineering*: *Guide to Building Dependable Distributed Systems*. 2nd ed. Wiley & Sons, 2008. ISBN 978-0-470-06852-6, [cit. 2009-12-27].
Dostupné z WWW: <http://www.cl.cam.ac.uk/~rja14/book.html>.
[20] SATYANARAYANAN, M. Integrating Security in a Large Distributed System [online]. *ACM Transactions on Computer Systems*, 1989, p. 1-34, [cit. 2009-12-27].
Dostupné z WWW: <http://www.cs.cmu.edu/~coda/docdir/sec89.pdf>.
[21] *Security Protocols Open Repository* [online], [cit. 2009-12-27].
Dostupné z WWW: <http://www.lsv.ens-cachan.fr/spore>.
[22] *A Proof Assistant for Higher-Order Logic* [online], [cit. 2009-05-20].
Dostupné z WWW: <http://www.cl.cam.ac.uk/research/hvg/isabelle/dist/Isabelle/doc/tutorial.pdf>.
[23] *The Isabelle/Isar Reference Manual* [online], [cit. 2009-05-20].
Dostupné z WWW: <http://www.cl.cam.ac.uk/research/hvg/isabelle/dist/Isabelle/doc/isar-ref.pdf>.
[24] PAULSON, L. C. The foundation of a generic theorem prover [online]. *Journal of Automated Reasoning*, *Computer Laboratory*. University of Cambridge, 1989, p. 363-397, [cit. 2009-05-20].
Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.3928>.

[25] CREMERS, C. J. F. *Scyther tool* [online], [cit. 2009-12-30].
Dostupné z WWW: <http://people.inf.ethz.ch/cremersc/scyther/index.html>.
[26] CREMERS, C. J. F. *Scyther 1.0 User Manual DRAFT* [online]. April 2007, [cit. 2009-12-30].
Dostupné spolu s instalačním balíčkem z WWW: <http://people.inf.ethz.ch/cremersc/scyther/install-generic.html>.
[27] AVISPA. *v1.1 User Manual*. 2003 [online], [cit. 2009-12-30].
Dostupné z WWW: <http://www.avispa-project.org>.
[28] AVISPA. *Deliverable 2.1:The High Level Protocol Specification Language*. 2003 [online], [cit. 2009-12-30].
Dostupné z WWW: <http://www.avispa-project.org>.
[29] SAILLARD, R., GENET, T. *CAS+* [online], September 3, 2009, [cit. 2009-12-30].
Dostupné z WWW: <http://www.irisa.fr/celtique/genet/span/CAS_manual.pdf>.
[30] GLOUCHE, Y., GENET, T., HOUSSAY, E. User Manual for a Security Protocol ANimator for AVISPA [online]. *IRISA/Université de Rennes*, September 2008, [cit. 2009-12-30].
Dostupné z WWW: <http://www.irisa.fr/celtique/genet/span/#Download >.
[31] PAULSON, L. C. Two Formal Analyses of the Yahalom Protocol [online]. *Computer Laboratory, University of Cambridge*, 1997, p. 1-10, [cit. 2009-05-15].
Dostupné z WWW: <www.cl.cam.ac.uk/~lp15/papers/Auth/yahalom.pdf>.
[32] LOWE, G. Towards a Completeness Result for Model Checking of Security Protocols [online]. *Department of Mathematics and Computer Science, University of Leicester*, 1998, p. 1-10, [cit. 2009-05-15].
Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.8212>.
[33] RFC 2759. *Microsoft PPP CHAP Extensions, Version 2*. ZORN, G. [s.l.] : Microsoft Corporation, January 2000. 20 s, [cit. 2009-05-15].
Dostupné z WWW: <http://tools.ietf.org/html/rfc2759>.
[34] SCHNEIER, B., WAGNER, D. Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2) [online]. *Counterpane Systems*, October 1999, p. 1-8, [cit. 2009-05-15].
Dostupné z WWW: <http://www.schneier.com/paper-pptpv2.pdf>.
[35] EISINGER, J. Exploiting known security holes in Microsoft's PPTP Authentication Extensions (MS-CHAPv2) [online]. *University of Freiburg*, July 2001, p. 1-2, [cit. 2009-05-15].
Dostupné z WWW: <http://penguin-breeder.org/pptp/download/pptp_mschapv2.pdf>.
[36] HWANG, T., NARN, Y. L., CHUANG, M. L., MING, Y. K., YUNG, H. CH. Two attacks on neumann-stubblebine authentication protocols [online]. In *Information Processing Letters*, vol. 53, 1995, p. 103-107, [cit. 2010-05-20].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=202441>.
[37] BELLOVIN, S. M., MERRIT, M. Encrypted key exchange: Password-based proto-

cols secure against dictionary attacks [online]. In *1992 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 1992, p. 72-84, [cit. 2010-05-10].
Dostupné z WWW: <http://www.ece.cmu.edu/~adrian/630-f03/readings/bellovin-merritt.pdf>.
[38] RFC 1731. *IMAP4 Authentication Mechanisms*. MYERS, J. [s.l.] : Carnegie Mellon, December 1994. 6 s., [cit. 2010-05-10].
Dostupné z WWW: <http://tools.ietf.org/html/rfc1731>.
[39] RFC 2195. *IMAP/POP AUTHorize Extension for Simple Challenge/Response*. KLENSIN, J., CATOE, R., KRUMVIEDE, P. [s.l.] : MCI, 1997. 7 s., [cit. 2010-05-10].
Dostupné z WWW: <http://tools.ietf.org/html/rfc2195>.
[40] RFC 2104. *HMAC: Keyed-Hashing for Message Authentication*. KRAWCYK, H., BELLARE, M., CANETTI, R. [s.l.] : IBM, February 1997. 11 s., [cit. 2010-05-10].
Dostupné z WWW: <http://www.ietf.org/rfc/rfc2104.txt>.
[41] WOO, T. Y. C., LAM, S. S. A lesson on authentication protocol design [online]. *Operating Systems Review*, vol.28, 1994, p. 24-37, [cit. 2010-05-10].
Dostupné z WWW: <www.cs.utexas.edu/users/lam/Vita/Bpapers/WooLam94a.pdf>.
[42] BRADNER, S., MANKIN, A., SCHILLER, J. I. A Framework for Purpose-Built Keys (PBK) [online]. Network Working Group Internet Draft, In progress, June 2003, p. 1-5, [cit. 2010-05-12].
Dostupné z WWW: <http://tools.ietf.org/html/draft-bradner-pbk-frame-06>.
[43] LOWE, G. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, vol. 56, November 1995, p. 131-136, [cit. 2010-05-12].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=219895>.
[44] CREMERS, C. J. F., MAUW, S., VINK, E. P. Injective Synchronisation: an extension of the authentication hierarchy [online]. *Department of Mathematics and Computers, Eindhoven University of Technology*, May 2006, p. 1-30, [cit. 2010-05-16].
Dostupné z WWW: <http://people.inf.ethz.ch/cremersc/publications/index.html>.
[45] DOLEV, D., YAO, A. C. On the security of public-key protocols [online]. *Computer Science Department, Stanford University*, May 1981, p. 1-26, [cit. 2010-05-16].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=891726>.
[46] CREMERS, C. J. F. *Scyther - Semantics and Verification of Security Protocols* [Thesis], Technische Universiteit Eindhoven, 2006, [cit. 2010-05-16].
Dostupné z WWW: <http://people.inf.ethz.ch/cremersc/publications/index.html>.
[47] CREMERS, C. J. F. Verication of Multi-Protocol Attacks [online]. *Computer Science Report*, Eindhoven University of Technology, 2005, p. 1-9, [cit. 2009-05-15].
Dostupné z WWW: <http://people.inf.ethz.ch/cremersc/publications/index.html>.
[48] CREMERS, C. J. F., MAUW, S. Operational semantics of security protocols [online]. *Computer Science Report*, Eindhoven University of Technology, 2004, p. 1-5, [cit. 2009-05-15].
Dostupné z WWW: http://people.inf.ethz.ch/cremersc/publications/index.html.

[49] CREMERS, C. J. F., LAFOURCADE, P. Comparing State Spaces in Automatic Security Protocol Verification [online]. Technical Report 558, *Department of Computer Science, ETH Zurich*, 2007, [cit. 2009-05-15].
Dostupné z WWW: <http://people.inf.ethz.ch/cremersc/publications/index.html>.
[50] CREMERS, C. J. F., MAUW, S., VINK, E. P. Formal Methods for Security Protocols: Three Examples of the Black-Box Approach [online]. *Newsletter of the Dutch Association for Theoretical Computing Scientist*, vol. 7, 2003, p. 21-32, [cit. 2010-05-16].
Dostupné z WWW: <http://people.inf.ethz.ch/cremersc/publications/index.html>.
[51] NEEDHAM, R., SCHROEDER, M. Using encryption for authentication in large networks of computers [online]. *Communications of the ACM*, vol. 21, December 1978, p. 993-999, [cit. 2010-05-20].
Dostupné z WWW: <http://portal.acm.org/citation.cfm?id=359659>.
[52] OČENÁŠEK, P. *Verifikace bezpečnostních protokolů.* [diplomová práce], FIT VUT v Brně, 2003.
[53] PTÁČEK, M. *Specifikační jazyky a nástroje pro analýzu a verifikaci bezpečnostních protokolů*. [diplomová práce], FIT VUT v Brně, 2007.

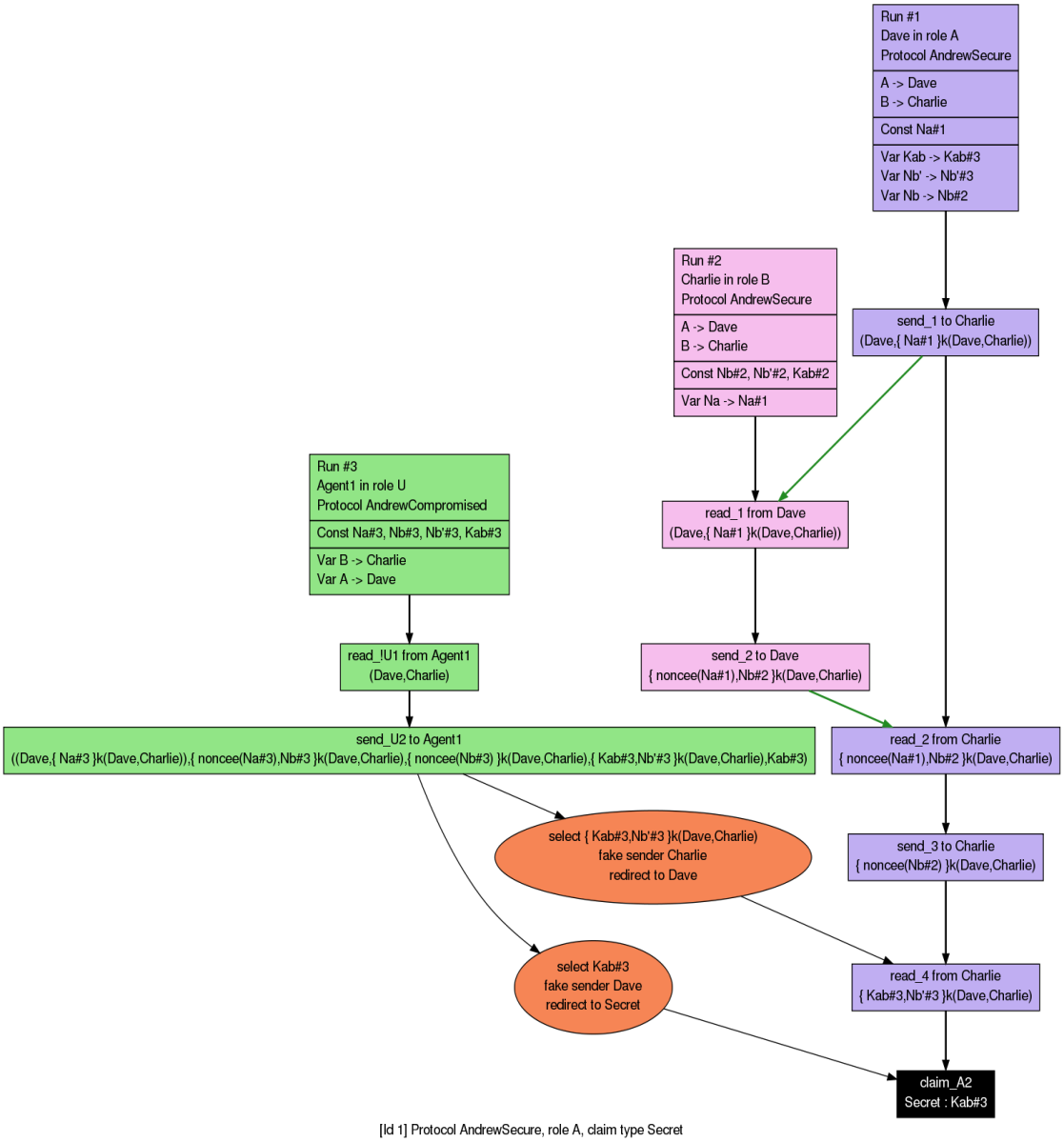
# Seznam příloh

# Seznam zkratek

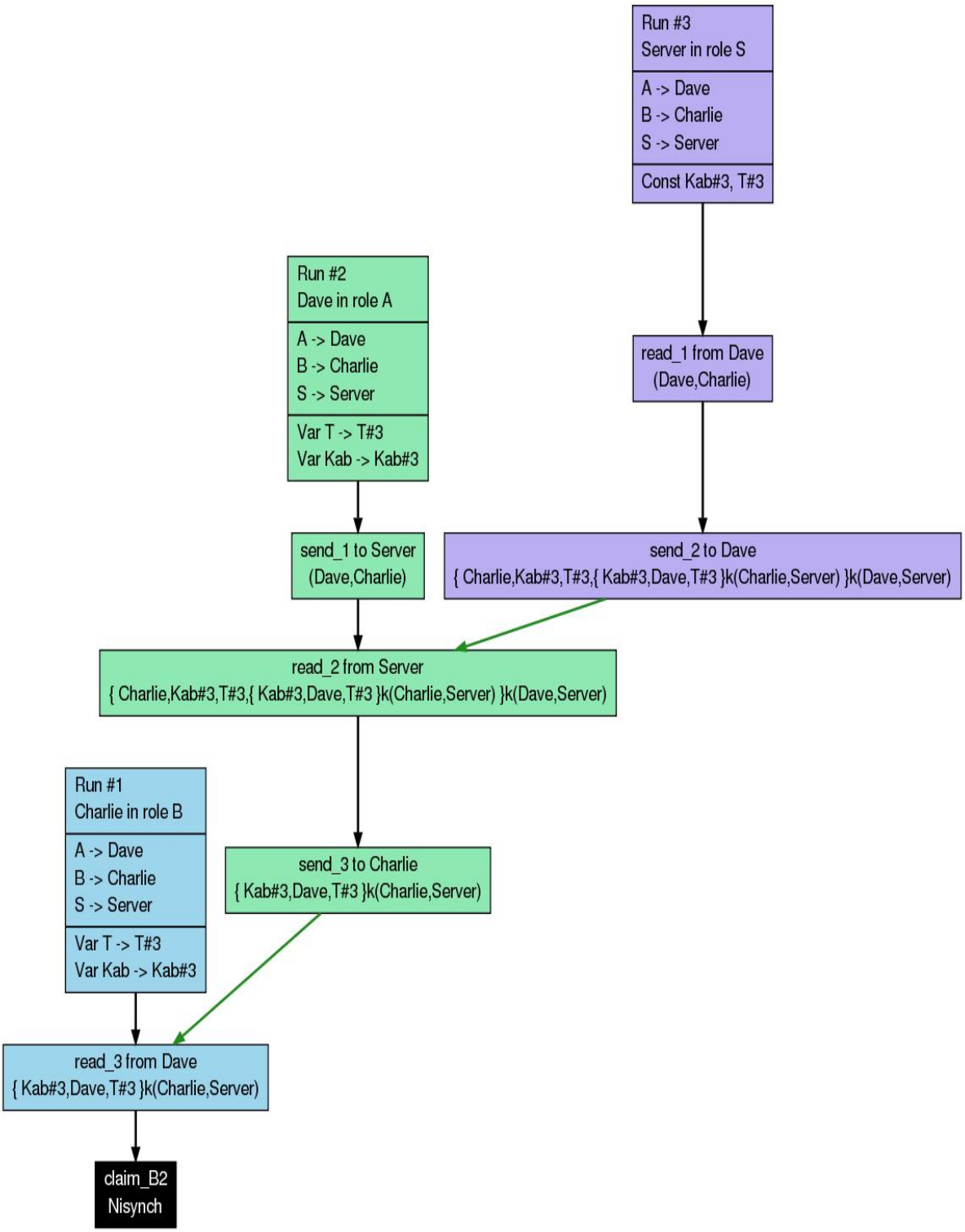
| | |
|---|---|
| **BAN** | Zkratka počátečních písmen přijmení autorů M. Burrows, M. Abadi, R. Needham |
| **GNY** | Zkratka počátečních písmen přijmení autorů L. Gong, R. Needham, R. Yahalom |
| **SVO** | Zkratka počátečních písmen přijmení autorů P. F. Sylverson, P. C. van Oorschot |
| **SV** | Signature verification |
| **ASVO** | Automatic considered SVO |
| **HOL** | High Order Logic |
| **SML** | Standard ML |
| **OCaml** | Objective Caml |
| **AVISPA** | Automated Validation of Internet Security Protocols and Applications |
| **HLPSL** | High Level Protocol Specification Laguage |
| **HLPSL2IF** | High Level Protocol Specification Laguage to Intermediate Format |
| **IF** | Intermediate Format |
| **OFMC** | On-the-Fly Model Checker |
| **CL-ATSE** | CL based Attack Searcher |
| **SATMC** | SAT based Model Checker |
| **TA4SP** | Tree Automata based on Automatic Approximations for the Analysis of Security Protocols |
| **SPAN** | Security Protocol Animator |
| **MSC** | Message Sequence Chart |
| **ETH** | Eidgenössische Technische Hochschule |
| **SPORE** | Security Protocols Online Repository |
| **SPDL** | Security Protocols Description Language |
| **IETF** | Internet Engineering Task Force |
| **MS-CHAPv2** | Microsoft Challenge-Handshake Authentication Protocol version 2 |
| **CRAM-MD5** | Challenge-Response Authentication Mechanism - Message Digest algorithm 5 |
| **MD5** | Message Digest algorithm 5 |
| **DES** | Data Encryption Standard |
| **SHA** | Secure Hash Algorithm |
| **IMAP** | Internet Message Access Protocol |
| **HMAC-MD5** | Hash-based Message Authentication Code - Message Digest algorithm 5 |
| **EKE** | Encrypted Key Exchange |
| **PBK** | Purpose Built Keys |
| **PBID** | Purpose Built Identification |
| **IP** | Internet Protocol |

# Seznam obrázků a tabulek

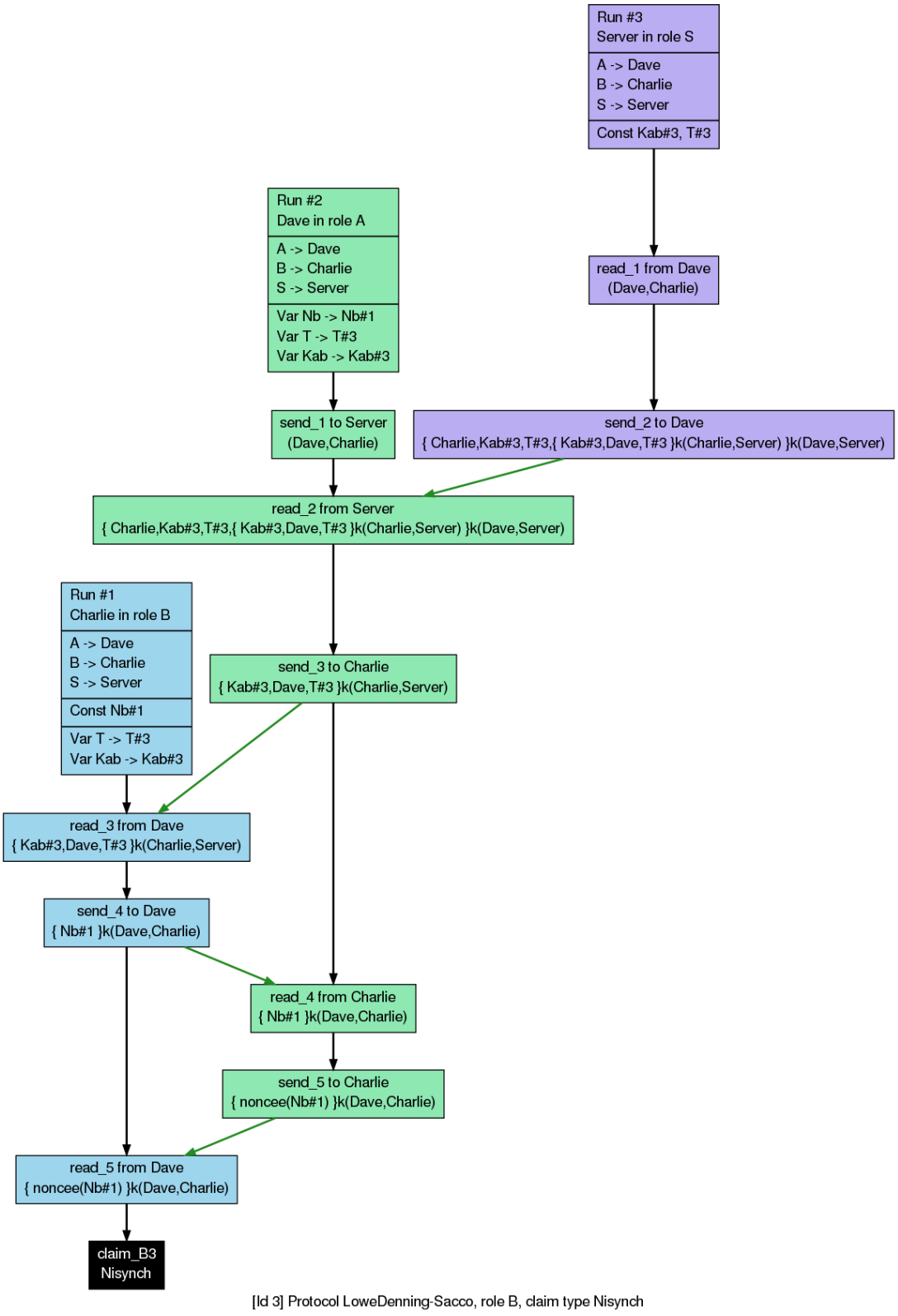
# Analyzované protokoly – grafický výstup nástroje Scyther



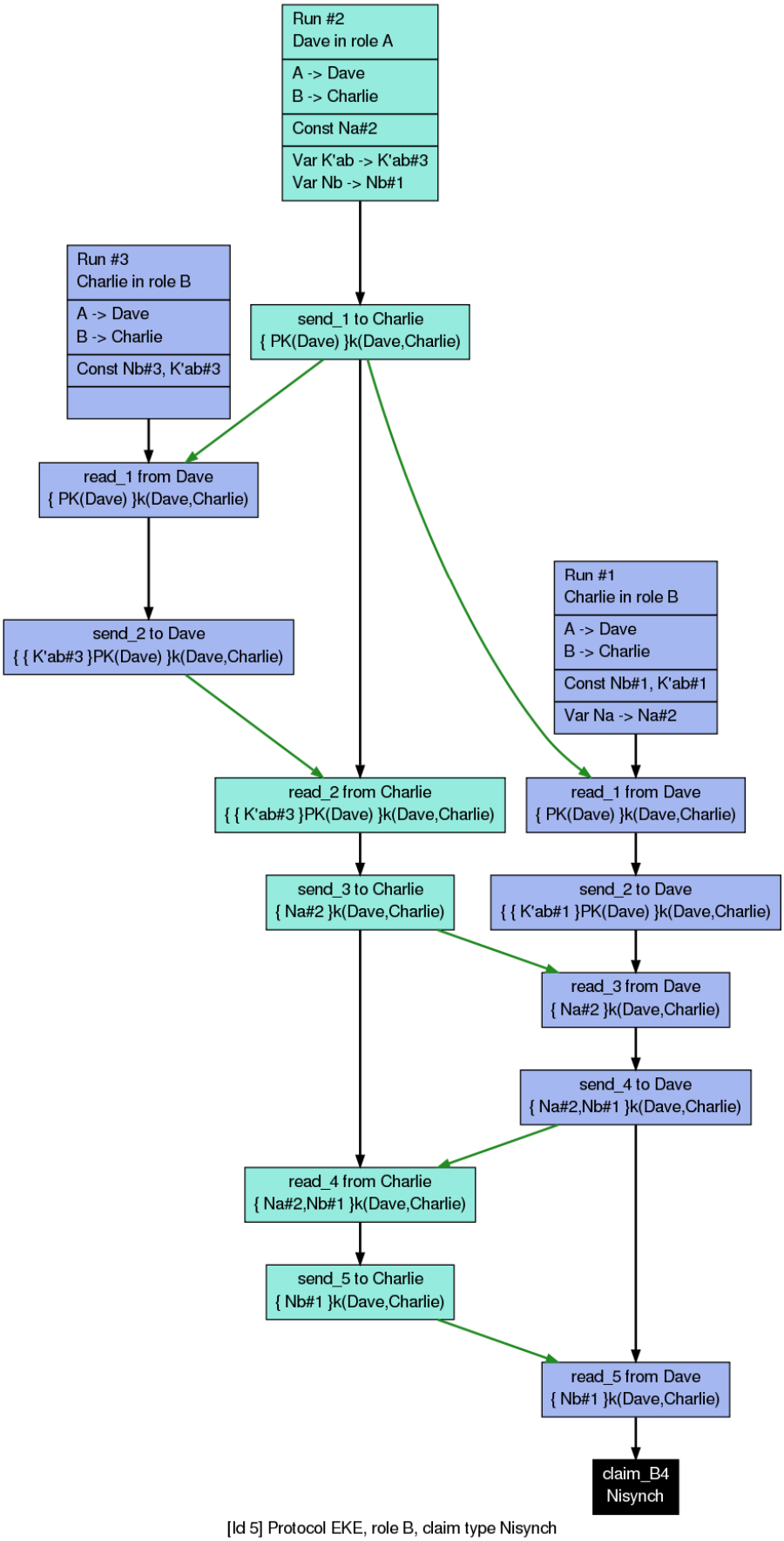

**Obrázek 17 - Andrew Secure RPC**



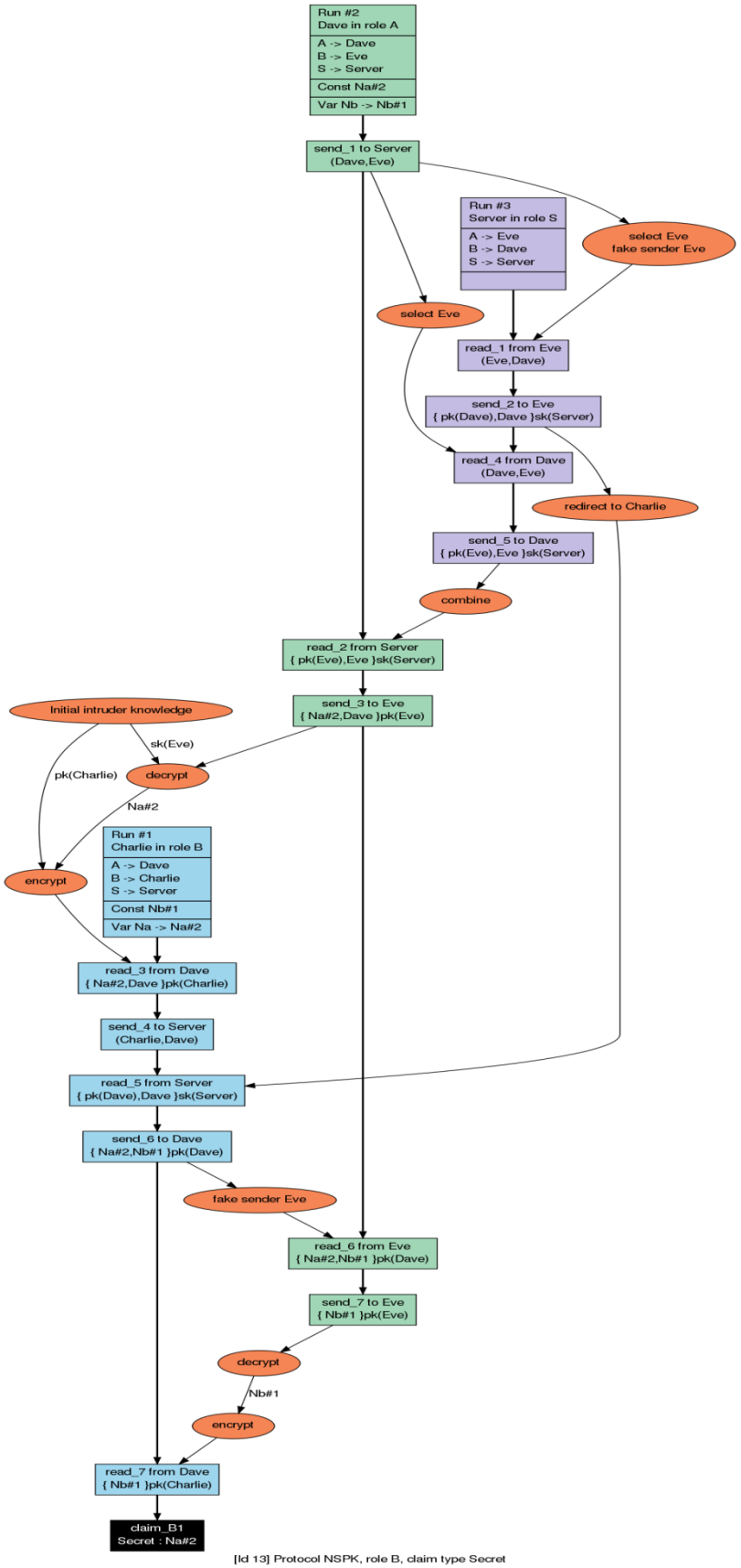

**Obrázek 18 - Dennig – Sacco shared key**



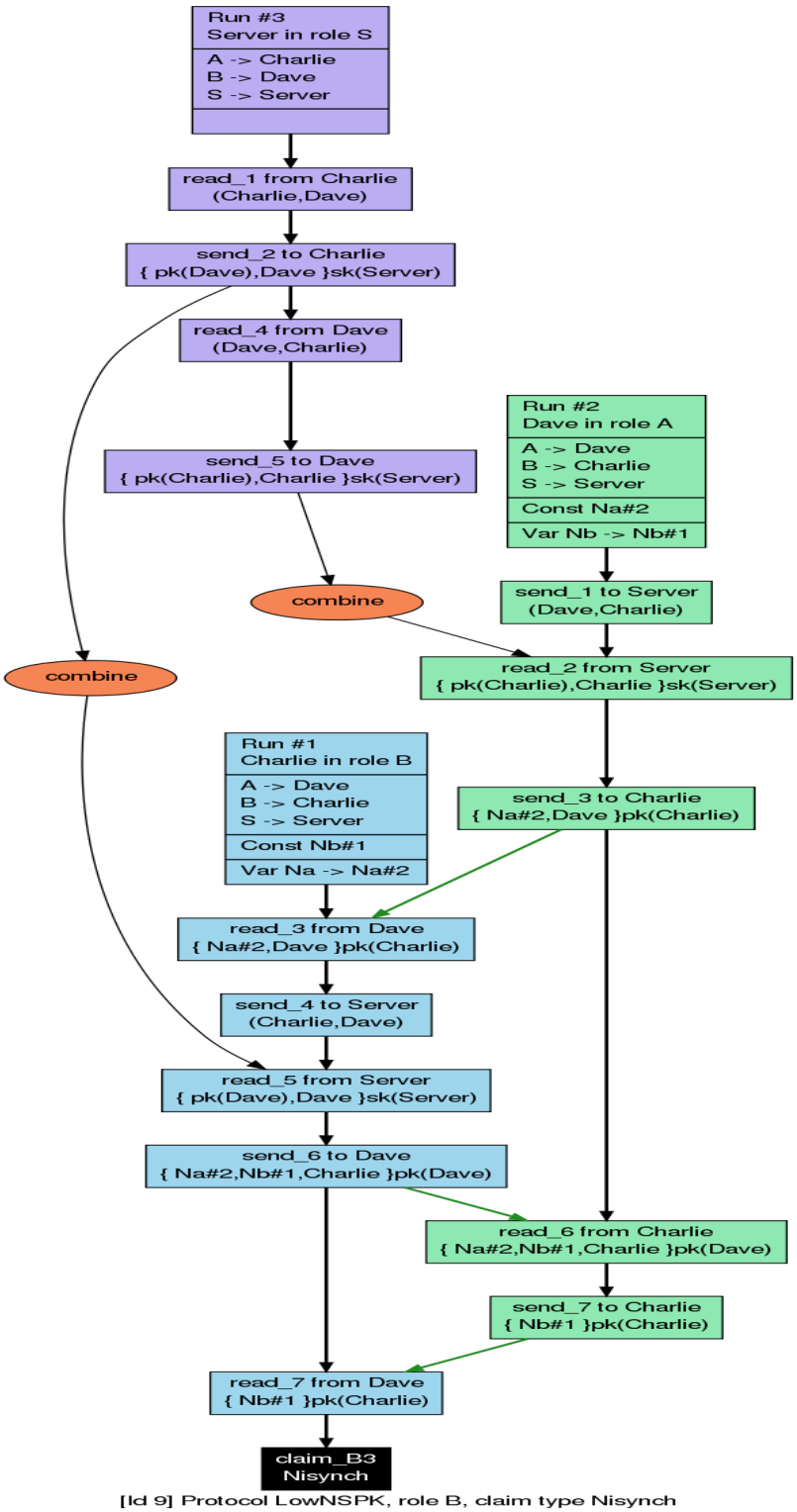

**Obrázek 19 - Lowův modifikovaný Denning - Sacco shaed key**



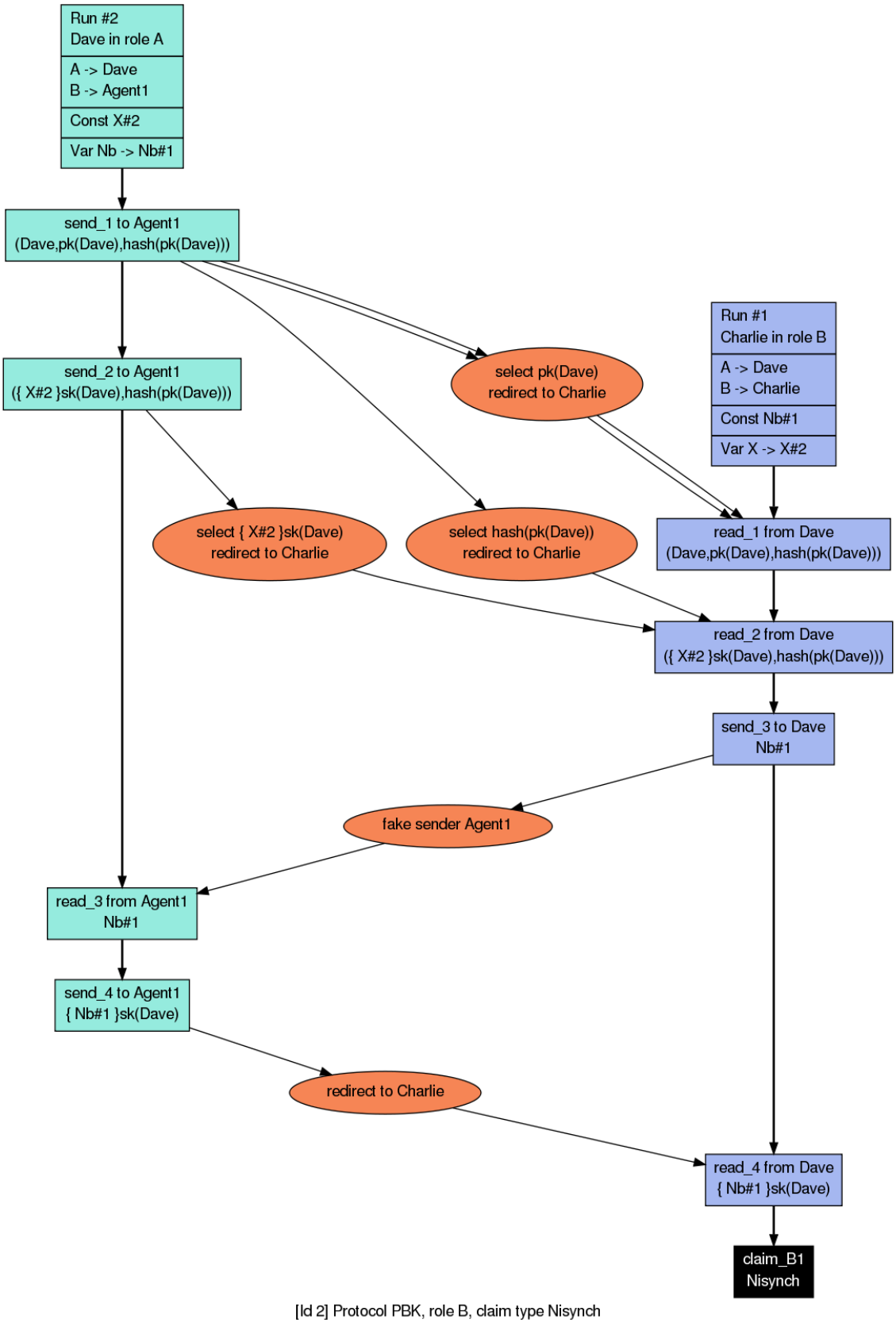

**Obrázek 20 - Encrypted Key Exchange**



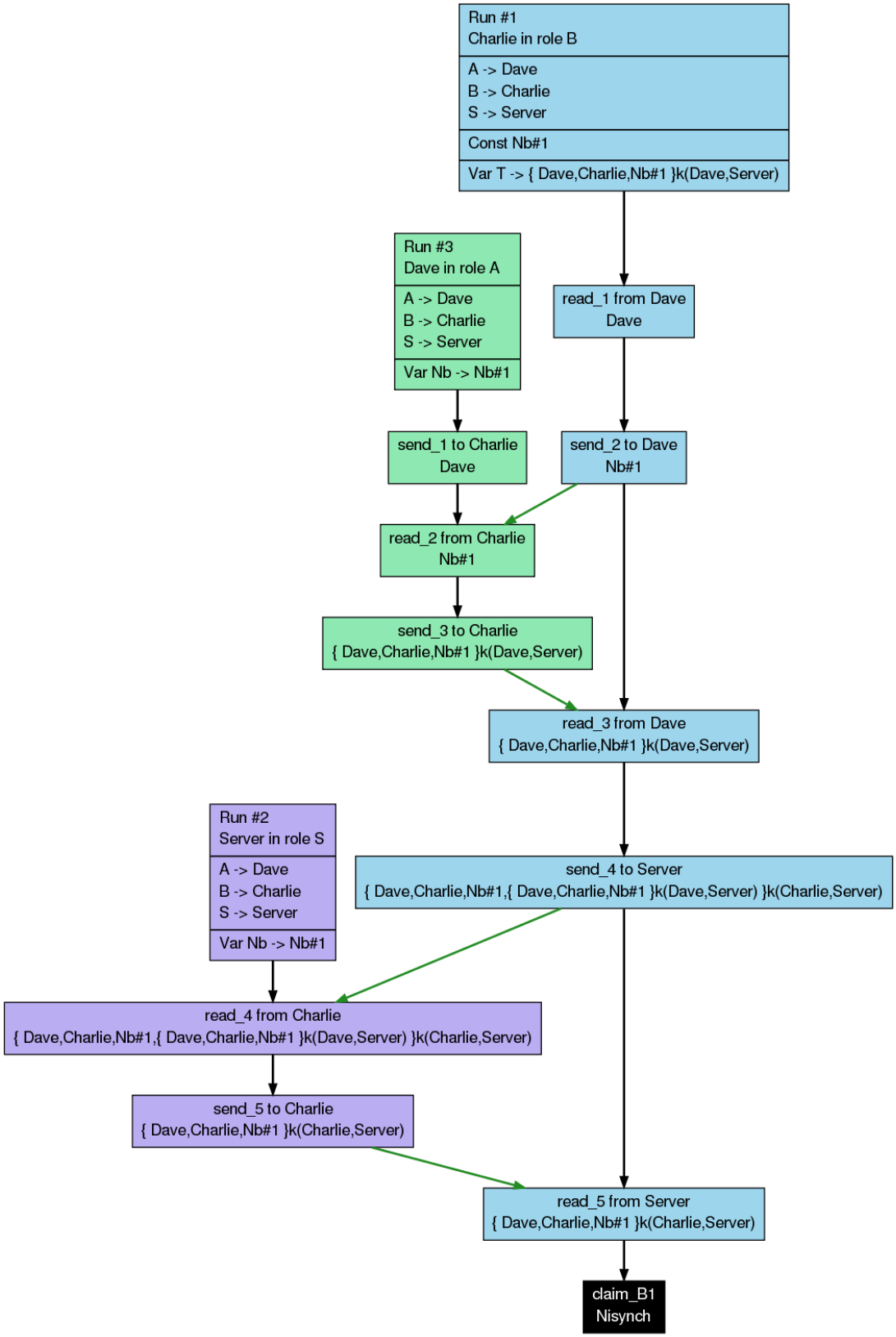

**Obrázek 21 - Needham – Schroeder Public key**



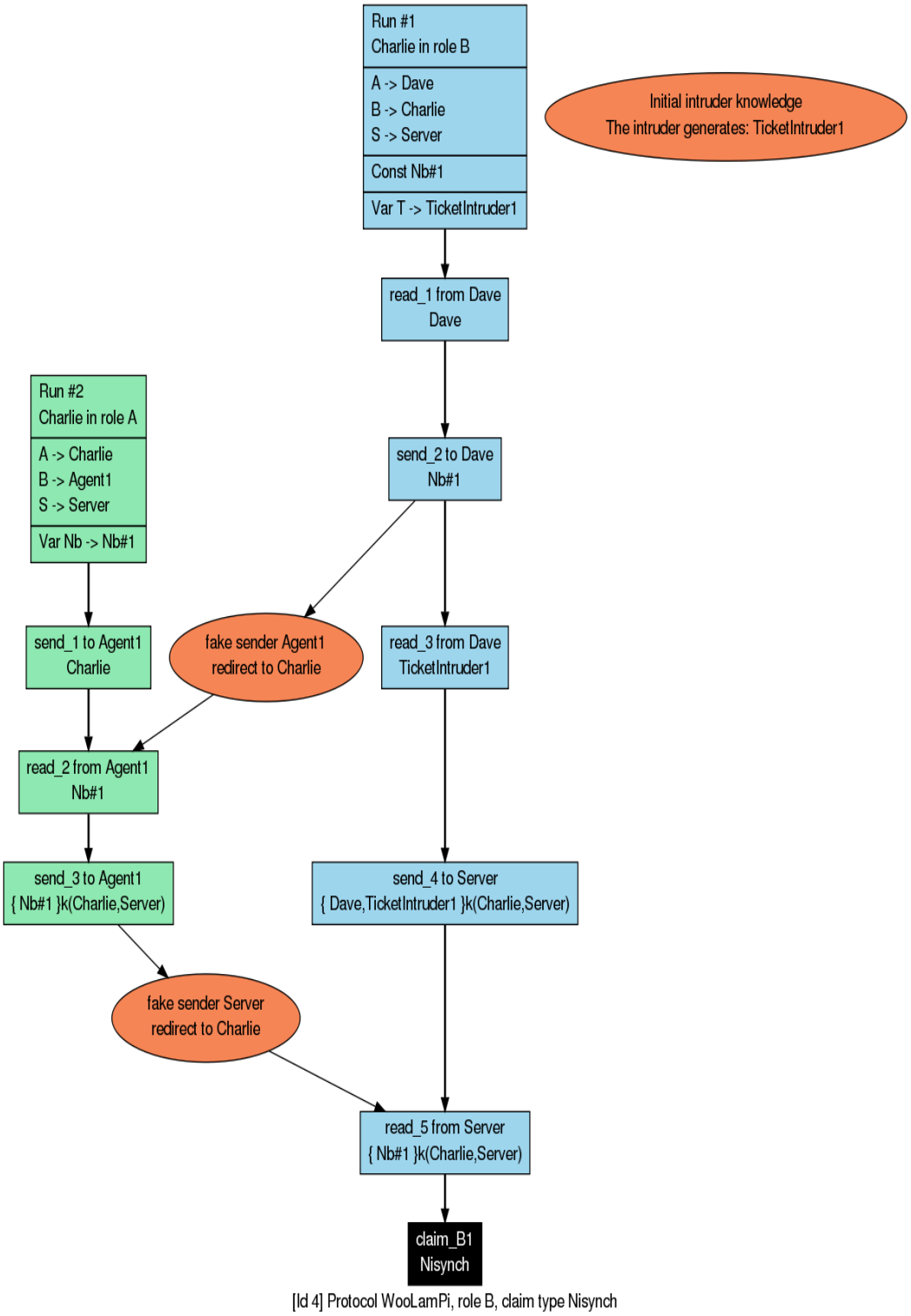

**Obrázek 22 - Lowe Needham - Schroeder Public key**



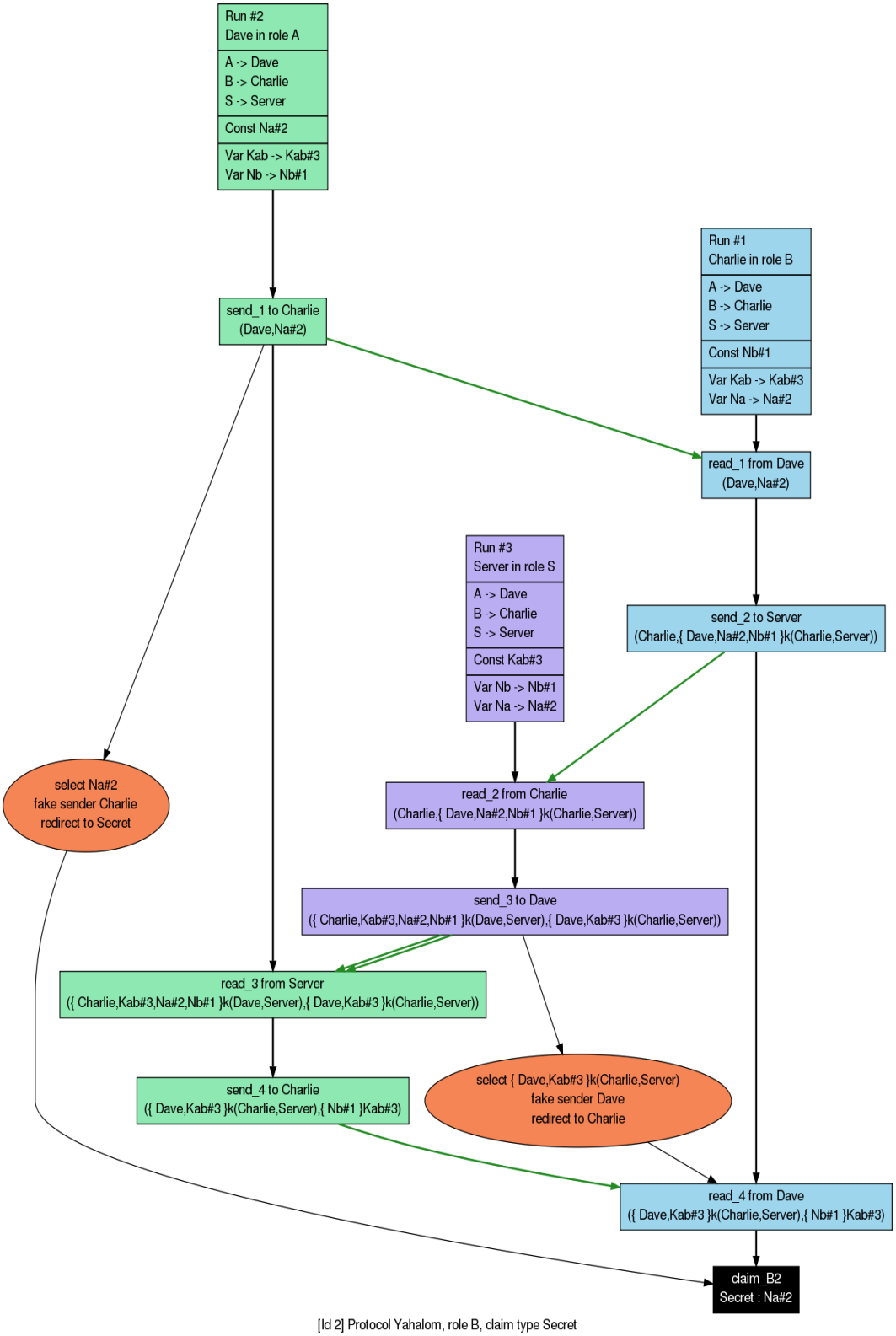

**Obrázek 23 - Purpose Built Keys**



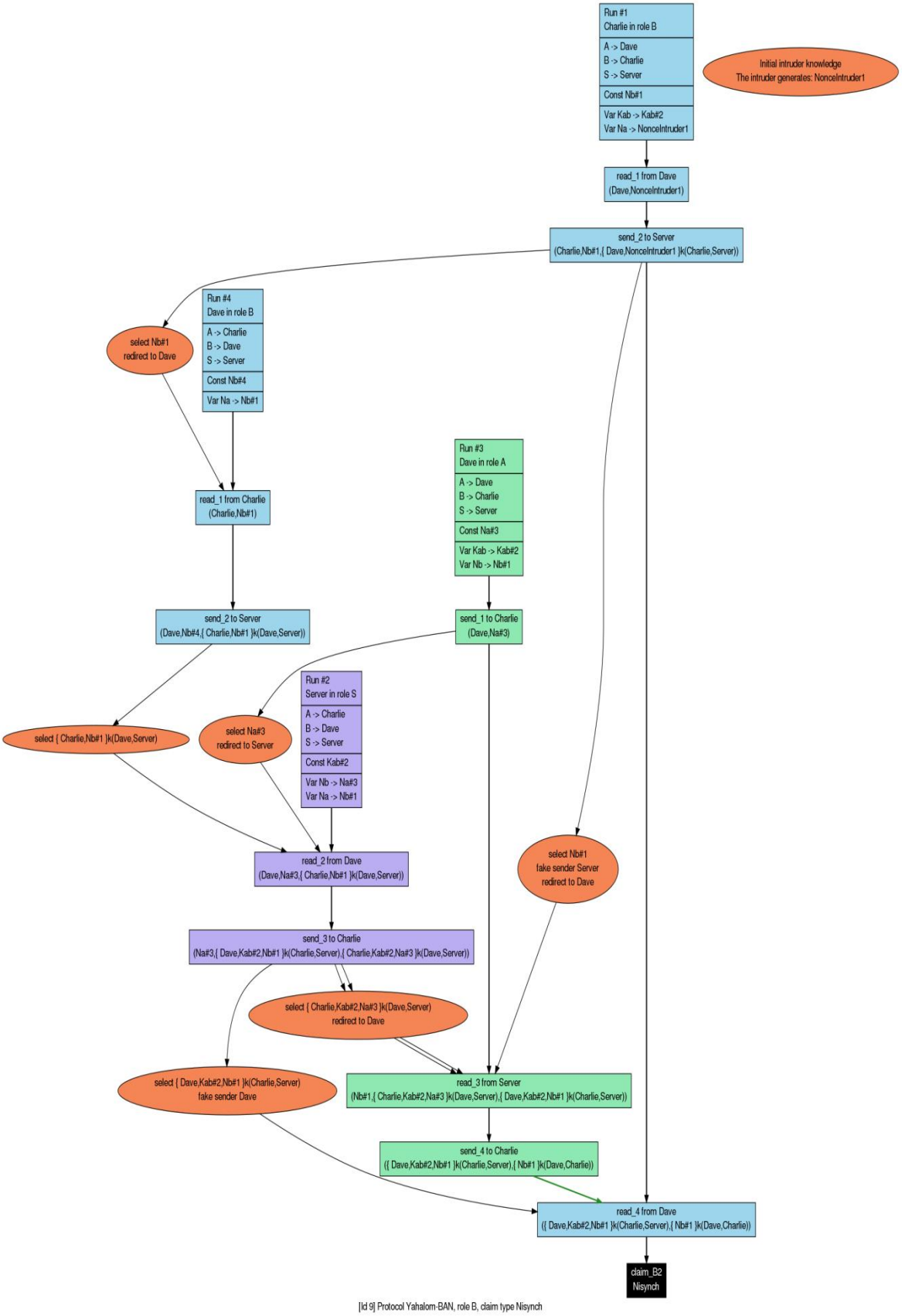

**Obrázek 24 - Woo nad Lam Pi f**



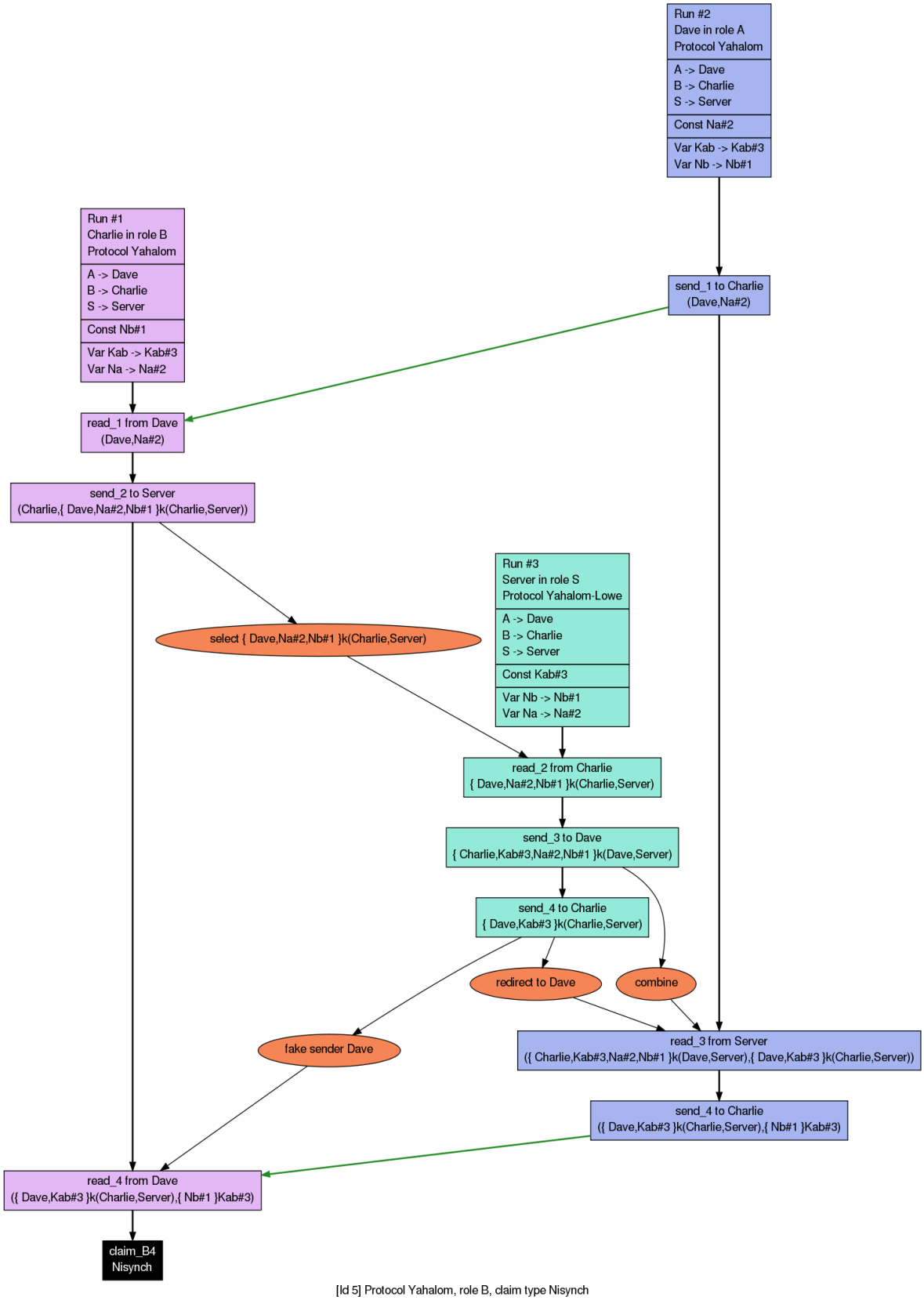

**Obrázek 25 - Woo and Lam Pi**



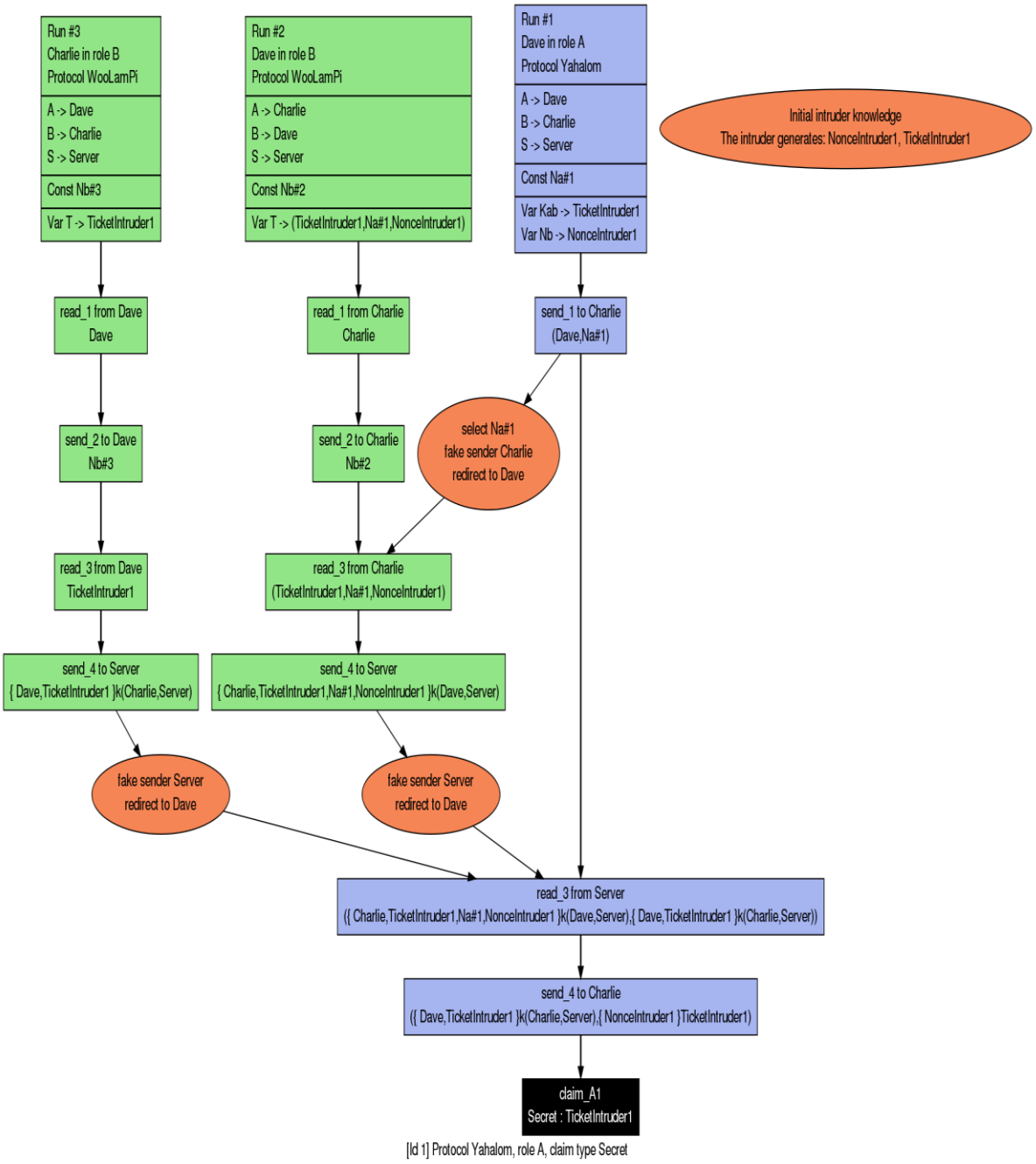

**Obrázek 26 – Yahalom**




**Obrázek 27 - BAN modifikovaný Yahalom**




**Obrázek 28 - Yahalom a Lowova verze Yahalom**




**Obrázek 29 - Yahalom a Woo and Lam Pi**