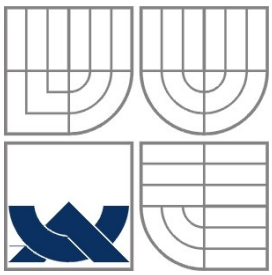
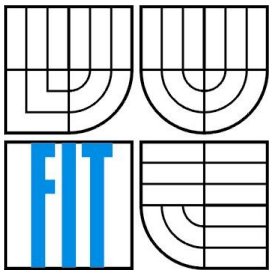


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DATABÁZE SPECIFIKACÍ BEZPEČNOSTNÍCH PROTOKOLŮ

SPECIFICATIONS DATABASE OF SECURITY PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

PETR HADAŠ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PAVEL OČENÁŠEK

BRNO 2008

Abstrakt

Diplomová práce popisuje čtyři nástroje určené pro verifikaci bezpečnostních protokolů Athena, Casper, Isabelle a Murphi. U každého je uvedena stručná charakteristika a část implementace protokolu Needham Schroeder. Součástí je také srovnání vybraných nástrojů. Druhá část práce podrobně popisuje nástroj Athena a uvádí příklady verifikovaných protokolů. U každého protokolu je uvedena specifikace komunikace, odhalený útok a výsledky vlastní verifikace. Na závěr práce porovnává výsledky verifikace s již publikovanými útoky.

Klíčová slova

Bezpečnostní protokoly, verifikace, autentizace, Murphi, Isabelle, Athena, Casper, Woo Lam, Needham Schroeder, Yahalom, Andrew Secure RPC

Abstract

This paper describes four tools for verification security protocols Athena, Casper, Isabelle and Murphi. Each tool is briefly characterized and implementation of protocol Needham Schroeder. One part of this paper is comparing of selected tools. The second part of this paper describes in detail a tool Athena and mentions examples of verified protocols. By each protocol is stated a specifications of communication, a detected attack and results of own verification. At the end compares this paper verification results with already publicated attacks.

Keywords

Security protocols, authentication, verification, Murphi, Isabelle, Athena, Casper, Woo Lam, Needham Schroeder, Yahalom, Andrew Secure RPC

Citace

Hadaš Petr: DATABÁZE SPECIFIKACÍ BEZPEČNOSTNÍCH PROTOKOLŮ. Brno, 2008, diplomová práce, FIT VUT v Brně.

Databáze specifikací bezpečnostních protokolů

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením ing. Pavla Očenáška. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Hadaš
15. 5. 2008

Poděkování

Děkuji svému vedoucímu ing. Pavlu Očenáškovi za veškerou pomoc při vypracování této práce. Dále pak rodině za podporu během celé doby studia.

Obsah

1 Úvod

Diplomová práce se zabývá bezpečnostními protokoly a způsoby jejich verifikace. Práci lze rozdělit na dvě základní části. První se zabývá porovnáním verifikačních nástrojů, druhá obsahuje detailní popis a analýzu protokolů pomocí nástroje Athena.

První část se tedy zabývá bezpečnostními protokoly a vybranými nástroji, které se používají k verifikaci těchto protokolů. Ještě před těmito kapitolami dokument seznamuje čtenáře se základními pojmy, které je nutné znát z důvodu následného správného pochopení této práce. Kapitola Bezpečnostní protokoly obsahuje seznam vybraných protokolů, které byly následně implementovány. Pro pozdější klasifikaci protokolů jsou v této kapitole uvedeny i typy protokolů. V části nazvané Verifikační nástroje jsou nejprve zmíněny všeobecné způsoby ověřování správnosti protokolů, nabídka nástrojů a uvádí se zde důvod pro výběr vybraných nástrojů. Dále se již věnuji vybraným verifikačním nástrojům. Po výpisu podrobností o nástrojích je uvedeno porovnání nástrojů Athena, Casper, Isabelle a Murphi a na základě porovnání vybrán nástroj Athena.

Druhá část práce se zabývá právě nástrojem Athena. Kapitola Athena – vybraný nástroj seznamuje čtenáře v širším kontextu s tímto nástrojem, ale také ukazuje příklad verifikace protokolu Woo Lam od začátku až po zpracování výsledků. Následující část popisuje jednotlivé protokoly. Uvádí zápis komunikace a známého útoku. U útoků je uveden detailnější popis pro snazší pochopení. Pod těmito údaji se nachází blok s vlastní implementací, kde se dozvíme, zda byla odhalena chyba a případně, která chyba to byla. Po specifikaci všech protokolů následuje závěrečné zhodnocení úspěšnosti nástroje a porovnání nalezených chyb s dříve publikovanými útoky.

Součástí práce je také návod pro instalaci a práci s nástrojem Athena, který pomůže při verifikaci vlastních protokolů. Na závěr jsou uvedeny ukázkové příklady několika implementovaných protokolů.

1.1 Základní pojmy

Bezpečnostní protokoly

Bezpečnostní protokoly (BP) určují způsob bezpečné komunikace skrze nezabezpečený prostor. BP lze rozdělit do dvou hlavních kategorií na [1] autentizační protokoly a protokoly pro výměnu klíčů.

Autentizační protokoly slouží k určení identity uživatele, chybou v tomto protokolu by se mohlo stát, že bude mylně komunikovat s útočníkem místo s právoplatným uživatelem.

Protokoly pro výměnu klíčů se používají k distribuci klíčů pro následné šifrování a dešifrování. Správná funkce tohoto protokolu zajistí důvěrnost. U distribuce klíčů se používá ve většině případů tzv. důvěryhodná třetí strana, která tyto klíče poskytuje.

Řada protokolů poskytuje obě dvě předchozí vlastnosti. Provádí jednak autentizaci uživatele a zároveň distribuuje klíče.

Bezpečnostní požadavky

S rozvojem internetu a nutnosti komunikace prostřednictvím nezabezpečené sítě, se vyvíjely i požadavky na bezpečnost. Z počátku byl kladen důraz na [2] důvěrnost, integritu a dostupnost. Dnes se musí k těmto bezpečnostním požadavkům přiřadit i odpovědnost, nepopiratelnost a spolehlivost.

Důvěrnost: zajištění soukromí dat. Data smí být schopen přečíst pouze odpovědný uživatel.

Integrita: data nesmí být během přenosu modifikována. Uživatelé jsou si jisti, kdo data upravil, ke kontrole dat se používá hashovací funkce, která odhalí, zda data nebyly změněny. V případě útoku musíme ovšem ochránit i hashovací řetězec, protože útočník může změnit jak data, tak hash. Pokud dojde k chybě na vedení (při přenosu) hash chybu rozpozná.

Dostupnost: musí být zaručena dostupnost služby v libovolné době. Útoky na přehlcení serveru množstvím požadavků. Ten poté nemůže odpovídat. Tato vlastnost není vždy nejdůležitější ovšem jsou případy, kdy výpadek způsobí kritický stav systému.

Odpovědnost: uživatel musí být zodpovědný za své chování.

Nepopiratelnost: uživatel nemůže popřít provedenou akci. Typicky se jedná o elektronickou poštu, kdy pošleme email a nemůžeme popřít, že jsme to udělali.

Typy útoků [9]

Útoky na protokoly můžeme rozdělit na pasivní a aktivní útok. Při pasivním útoku útočník pouze odposlouchává komunikaci a hledá data, která by mohl posléze zneužít. Naproti tomu aktivní útočník zasahuje do komunikace mezi účastníky přenosu.

Odposlouchávání

Pokud se může útočník dostat k odposlouchávání komunikace, nastává problém se soukromím. Data nejsou v bezpečí a může dojít k odcizení citlivých informací. Odposlouchání je snadněji proveditelné v bezdrátových sítích. Jedná se o pasivní útok.

Podvržení identity

Útočník se vydává za někoho jiného, a tím může získat od napadeného subjektu důležité informace.

Modifikace zprávy

Útočník přijme zprávu, modifikuje ji a přepošle dále.

Přerušení komunikace

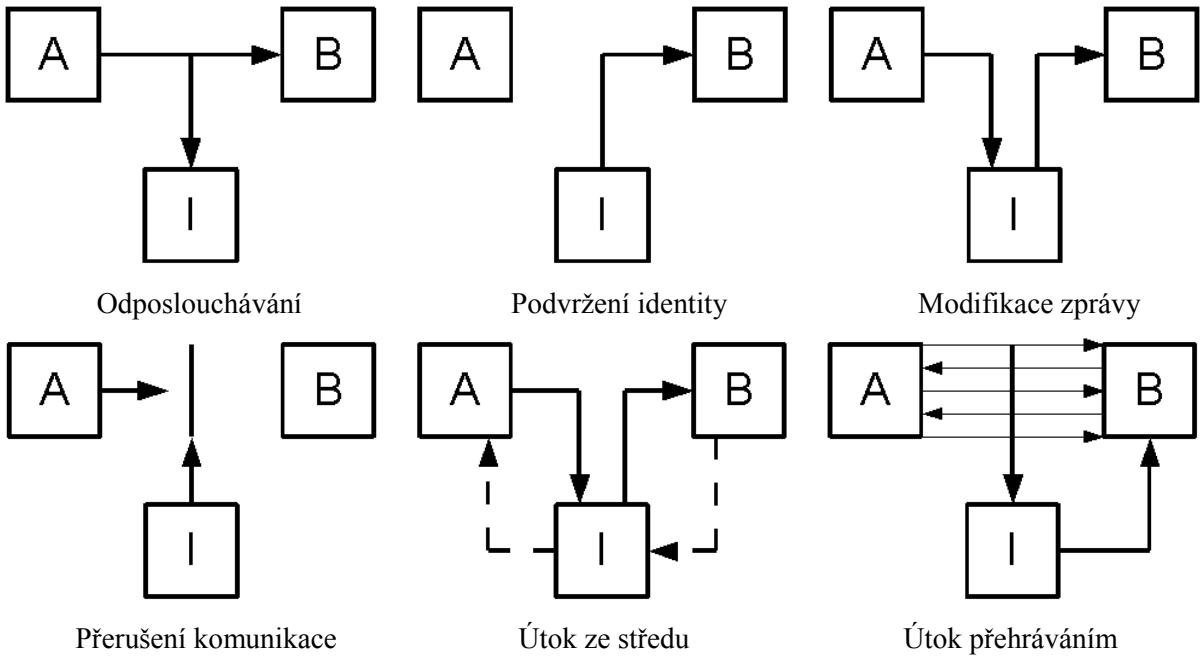
Útok na dostupnost. DoS.

Útok ze středu

Jedná se svým způsobem o dvojnásobné podvržení identity. Strany komunikace si myslí, že komunikují s protějškem, namísto toho komunikují přes útočníka.

Útok přehráváním

Nejprve útočník odposlouchává komunikaci a uchovává si důležité zprávy pro pozdější útok. Po získání potřebných informací může předstírat, že je právoplatný uživatel. Ochrana pomocí časových razítek.



2 Bezpečnostní protokoly

2.1 Seznam protokolů k implementaci

Vybrané protokoly jsem seřadil do několika kategorií podle [6] a [7]. Tato kapitola obsahuje seznam cca. 30 protokolů, které budou v letním semestru implementovány. Jednotlivé protokoly mají většinou vícero verzí, které upravují odhalené slabiny. Všechny verze jednotlivých protokolů zde nejsou uvedeny, jen nejvýznamnější, ovšem implementovat budu větší množství dostupných verzí.

Andrew Secure RPC
Andrew Secure RPC - BAN
Andrew Secure RPC - BAN konkrétní
Denning - Sacco
Denning - Sacco - modifikace Lowe
Kao Chow v.1
Kao Chow v.2
KSL
Needham Schroeder (symetrický klíč)
Needham Schroeder (symetrický klíč) rožšíření
Needham Schroeder (veřejný klíč)
Neumann Stubblebine
Neumann Stubblebine - Hwang
Otway Rees
Wide Mouthed Frog
Woo Lam Π
Woo Lam Π 1
Woo Lam Π 2
Woo Lam Π 3
Yahalom
Yahalom BAN

Jak je ze seznamu patrné při výběru protokolů byl kladen důraz na jednotlivé verze protokolu a snaha vystihnout jakým způsobem docházelo k rozvoji a pokroku u jednotlivých verzí, proti kterým útokům byly nové modifikace imunní, a kterým naopak neodolávaly. Porovnání je uvedeno za kapitolou věnovanou jednotlivým protokolům.

2.2 Typy protokolů

Pro porovnání a detailní pochopení funkce bezpečnostních protokolů je vhodné definovat kategorie, do kterých se testované protokoly rozdělí, a porovnají se jejich vlastnosti z hlediska zařazení:

Z hlediska počtu účastníků:

1. bez TTP – jedná se o protokoly, kde většinou komunikují pouze 2 strany iniciátor a příjemce mezi zástupce, zde patří např. Andrew Secure RPC
2. s TTP – ke komunikaci protokol vyžaduje důvěryhodnou třetí stranu, které všichni účastníci důvěřují, patří zde většina protokolů např. Yahalom, KSL.

Z hlediska použité kryptografie:

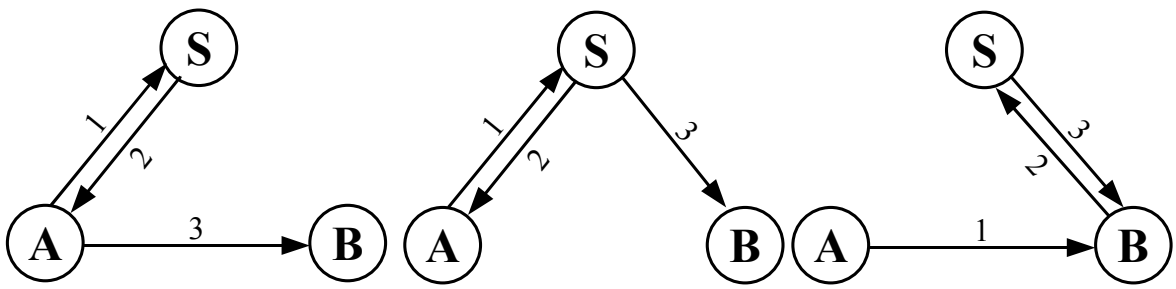
1. symetrický – používá se symetrická kryptografie, pouze x kopií stejného klíče, klíčem je možné provést šifrování i dešifrování, problém distribuce, většina protokolů
2. asymetrický – využívá soukromý a veřejný klíč, soukromý klíč zná pouze vlastník, veřejný klíč znají všichni, jedním se provádí šifrování, druhým dešifrování, šifrovat lze oběma.

Z hlediska generování klíče:

1. KDC (Key Distribution Center) – klíč relace v práci označován nejčastěji jako Kab vytvoří TTP, tato varianta je u vybraných protokolů nejčastější, např. Needham Schroeder
2. KTC (Key Translation Center) – TTP klíč negeneruje, relační klíč vytvoří jeden z účastníků, TTP pouze tento klíč distribuuje, příkladem může být např. Wide Mouthed Frog

Z hlediska způsobu komunikace[2]:

- push model – iniciátor komunikace nejdříve komunikuje se serverem, po obdržení zprávy kontaktuje sám účastníka, se kterým chce komunikovat. Používá se jak u KDC, tak u KTC případem je např. Denning - Sacco
- hybridní model – přímá komunikace mezi účastníky neprobíhá, zprávy se posílají prostřednictvím TTP používá se jak u KDC, tak u KTC, příkladem protokolu je např. Wide Mouthed Frog
- pull model – TTP kontaktuje až příjemce, tento model se používá pouze u KDC. Příkladem tohoto protokolu je Woo Lam.



Z hlediska funkce:

- jednosměrná autentizace – autentizuje se pouze jedna strana, v některých situacích postačující podmínka, kdy se např. autentizuje pouze server klientovi
- vzájemná autentizace – autentizuje se jak Alice k Bobovi, tak Bob k Alici. Oba jsou si jistí, se kterým účastníkem komunikují.
- distribuce klíče – oba účastníci si domluví čerstvý klíč (klíč relace), kterým komunikují. Je nutné zajistit, aby byl klíč opravdu aktuální, což je problém, který může vést k několika útokům.

2.3 Porovnání protokolů

Většina protokolů pracuje s tzv. Důvěryhodnou třetí stranou (Trusted Third Party), někdy označován také jako KDC (Key Distribution Center) nebo důvěryhodný server (trusted server). Jedná se o nezávislého účastníka, který sdílí tajemství (klíč) s oběma stranami komunikace. Tento subjekt musí být velmi dobře chráněn. Ve výčtu protokolů valná většina využívá tohoto důvěryhodného serveru. Jeho použitím se zvyšují bezpečnostní parametry celého systému. Příklad protokolu, který tento mechanizmus nepoužívá je Andrew Secure RPC.

Symetrické protokoly[6]

- ✔ každý účastník má pouze jeden dlouhý klíč
- ✔ symetrický klíč je relativně krátký
- ✔ jednoduché přidávání a ubírání dalších subjektů
- ✗ klíč může být během přenosu odhalen
- ✗ Důvěryhodná třetí strana je úzkým místem vzhledem k výkonu
- ✗ Důvěryhodná třetí strana musí uchovávat velké množství klíčů, pro každého uživatele

Asymetrické protokoly[6]

- ✔ nevyžaduje Důvěryhodnou třetí stranu neustále aktivní
- ✔ není třeba uchovávat veřejný klíč v tajnosti
- ✔ u velkých sítí je menší počet klíčů
- ✗ distribuce klíčů je pomalá v porovnání se symetrickým
- ✗ klíče jsou značně větší
- ✗ asymetrické klíče jsou čísla se speciálními matematickými vlastnostmi (nutno generátor)

Symetrické i asymetrické protokoly mají své pro i proti, je nutné se zamyslet, který je pro naše požadavky vhodný. Zda vyžadujeme rychlost nebo další bezpečnostní funkce, které nám poskytují např. Asymetrické protokoly.

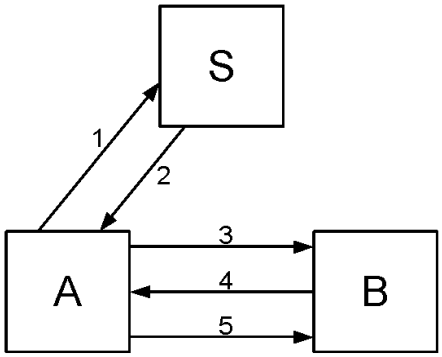
2.4 Příklad bezpečnostního protokolu

Needham-Schroeder

Tento autentizační protokol vymysleli britský vědec Roger Needham spolu s Michael D. Schroeder již v roce 1978. První tři části zajišťují výměnu klíče, poslední dvě zprávy jsou určeny k ověření uživatele, zda opravdu zná klíč. V tomto protokolu hraje významnou roli S (důvěryhodná třetí strana), která generuje klíč Kab. Spolu se S musí mít dohodnuty klíče oba účastníci, tyto klíče nesmí znát nikdo jiný.

Popis komunikace:

1. A => S: A, B, Na
2. S => A: {Na, B, Kab, {Kab, A}Kbs}Kas
3. A => B: {Kab,A}Kbs
4. B => A: {Nb}Kab
5. A => B: {dec(Nb)}Kab



Needham-Schroeder

Popis funkce

Nejprve pošle klient A, který chce zahájit komunikaci s klientem B, serveru S svou identifikaci a identifikaci PC klienta B. Dále připojí kontrolní číslo (nonce). Server mu následně vrátí zprávu se zašifrovaným klíčem, který znají pouze klient A a server S. Zpráva obsahuje serverem ustanovený klíč pro komunikaci mezi uživateli A a B. Tento klíč je v této zprávě obsažen ještě jednou, ovšem zašifrován klíčem Kbs, který zná pouze server S a uživatel B. Tento paket pošle klient A klientovi B, a tím se dozví klient B klíč pro vzájemnou komunikaci. Dále pak následují ještě pakety pro ověření identity uživatelů, zda uživatel, který nám poslal klíč Kab, jej opravdu zná.

Útok na protokol

Byl proveden autentizační útok pány Denning a Sacco. Předpokládá se, že útočník zaznamenává komunikaci a po nějaké době je schopen odhalit klíč Kab. Poté se bude vydávat v kroku 3 za A a zašle stejnou zprávu B. Tím se autentizuje.

3 Verifikační nástroje

3.1 Obecné informace

K provádění verifikace bezpečnostních protokolů je možné použít nepřeberné množství nástrojů. Každý má své výhody i nevýhody a je vhodné nejprve nastudovat jeho vlastnosti, před tím než jej použijeme. Při výběru se musí brát v zřetel, zda je program vhodný pro námi testovaný protokol, zda se jedná o komerční či volně dostupný nástroj a také jaká je rychlost provádění analýzy.

Pro provádění verifikace bezpečnostních protokolů je možné využít nesčetné množství metod. Tyto metody lze rozdělit do tří hlavních skupin, jsou to ověřování modelem (model checker), dokazování teorému (theorem provers) a nakonec dokazování pomocí formální logiky (formal logics).

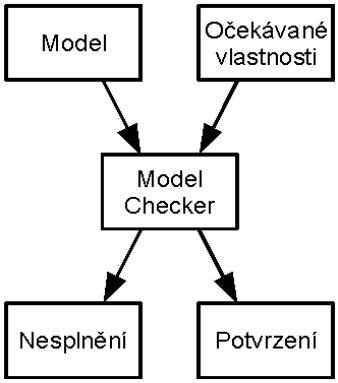
Ověřování modelem

Tato metoda prochází stavový prostor a kontroluje, zda jsou splněny požadavky. Jak ukazuje obrázek vstupem do modelu checkeru jsou dvě části model a specifikace. Model určuje stavový prostor, který se bude prohledávat. Specifikace určuje podmínky, které musí být při procházení tohoto stavového prostoru zaručeny. Tyto podmínky lze zapsat pomocí temporální logiky. Nástroje využívající verifikaci ověřováním modelem mohou mít specifikaci určenou několika možnými způsoby CTL, LTL …. Výsledkem by pokaždé mělo být, zda došlo ke splnění požadavků či zda jsme se během procházení stavového prostoru dostali do zakázaného stavu.

Při používání metody ověřování modelem se potýkáme s problémem rostoucího stavového prostoru. Procházení velkého stavového prostoru nám zvyšuje časovou náročnost a výsledek by nebyl například znám v rozumném čase, proto je nutné aplikovat redukci stavového prostoru. [8] Mezi redukční metody patří například abstrakce a kompozitní verifikace. Abstrakce redukuje stavový prostor tím, že se na systém nedíváme příliš konkrétně a tím pádem se nám zjednoduší model. Kompozitní verifikace vychází z předpokladu, že se dá systém rozdělit na menší podsystémy a tyto podsystémy verifikovat samostatně.

Nástroje:

PRISM, BRUTUS, KRONOS*, UPPAAL* SPIN a SMV.



Ověřování modelem (model checking)

Dokazování teorému

Theorem proving pracuje na principu odvozování nových tvrzení na základě dostupných odvozovacích pravidel ze zadaných axiomů. Systém využívající theorem proving vyžaduje přesný popis řešeného problému v logické formě. Tento zápis není snadný a vyžaduje značné úsilí od expertního uživatele. Systém se pokusí vyřešit specifikovaný problém. Tento přístup se používá jak pro softwarovou tak hardwarovou verifikaci.

Nástroje:

ISABELLE

Formální logika

Někdy také označována jako autentizační logika. Princip verifikace pomocí logiky [9] je takový, že se nejprve provede specifikace kroků protokolu a poté se definují požadované cíle. Po nadefinování těchto vlastností se spustí proces ověřování s nastavenými počátečními předpoklady, průběžně se kontroluje zda výsledek odpovídá požadovaným cílům.

Verifikace založena na logice, se skládá z těchto kroků [3]: Formalizace zpráv protokolu, specifikace počátečních předpokladů, specifikace cílů protokolu a aplikace logických požadavků. Největší zástupcem tohoto odvětví je BAN logika (Burrows, Abadi, Needham).

Nástroje:

AAPA

Testovaný protokol:

Needham Schroeder – testování vzájemné autentizace provedeno u všech nástrojů

A->B: {A,Na}Kb

B->A: {B,Na,Nb}Ka

A->B: {Nb}Kb

3.2 Athena

Jedná se o součást systému SyMP (Symbolic Model Prover). Tento nástroj se skládá ze dvou dokazovacích systémů standardního a Athena. Standardní kombinuje použití verifikace modelem a dokazování teorému a lze jej použít především k verifikaci hardwarových systémů. Dokazovací systém Athena je naproti tomu určen pouze na verifikaci bezpečnostních protokolů. Autorem systému Athena je Dawn Song z univerzity v Berkeley. Tento program je dostupný pouze pro platformu Unix. Verifikace je prováděna velmi rychle u běžných protokolů do jednotek sekund.

Protokolu se zapisuje pomocí tzv. Strand Spaces.[5] Každý protokol se skládá ze skupiny rolí (účastníků), kterým náleží sekvence akcí s příslušnými parametry. Vlastnosti protokolu, které chceme zkontrolovat, se zadávají pomocí výrokové logiky. Konkrétní účastník s definovanými parametry definuje vlákno (Strand).

Systém Athena se skládá ze tří částí APV, APG a ACG. Předmětem této práce je APV systém (Automatic Protocol Verifier) tato část se zabývá verifikací zapsaného protokolu. APG (Automatic Protocol Generator) je grafický generátor bezpečnostních protokolů. Dokáže generovat protokoly se dvěma a nebo třemi účastníky. Uživatel specifikuje bezpečnostní požadavky, požadavky na generování klíče atd. Jako poslední součást systému je ACG (Automatic Code Generator). Tento systém vygeneruje kód v programovacím jazyku Java, ze dvou vstupních souborů. První soubor bude obsahovat specifikaci protokolu ve formátu APV. Druhý soubor obsahuje pouze zjednodušený zápis protokolu.

Vstupní soubor

Zdrojový soubor musí začínat řetězcem begin a končit slovem end. Bezprostředně za úvodním begin následuje hlavička, která informuje o souboru. Zpravidla obsahuje název protokolu, který kontrolujeme. Samotný obsah souboru [4] je rozdělen na dvě části:

- Specifikace protokolu
- Specifikace vlastností

Specifikace protokolu

Soubor je vymezen klíčovými slovy begin a end. Jako první se zde nachází hlavička, která má pouze informativní charakter. Minimální forma hlavičky je „Protocol:“, je však dobrým zvykem vyplňovat celý název, pro přehlednost.

```
BEGIN
1:1 Protocol: NeedhamSchroederAsym
```

Za hlavičkou následuje charakteristika jednotlivých rolí. Na prvním řádku se nachází název role (P_A) se dvěma parametry první určuje identifikaci role a druhý počet kroků v specifikaci protokolu. Klíčovým slovem var začíná deklarace lokálních proměnných potřebných pro komunikaci.

Poté se již zaznamenává samotná komunikace. Na začátku mohou být dva druhy šipek, šipka -> určuje zprávu odeslanou, šipka <- zprávu přijatou. V příkladu jsou použity dva důležité prvky C a E. C představuje konkatenaci hodnot (C[a,b] = ab), E představuje šifrování (E{msg,key} zpráva msg je zašifrována klíčem key). Příkaz new zajišťuje generování nových hodnot nonce, potřebných k bezpečnosti. Jelikož se jedná o asymetrický protokol jsou zde použity k šifrování veřejné klíče. Ty se označují prefixem PUBKEY_ , podobně existují i prefixy pro ostatní typy klíčů PRIVKEY_ a SYMKEY_.

```
P_A(0,3){
VAR: P_A, P_B, NONCE_Na, NONCE_Nb;
-> : E{C[P_A,NONCE_Na],PUBKEY_P_B} | New(NONCE_Na);
<- : E{C[P_B,NONCE_Na,NONCE_Nb],PUBKEY_P_A};
-> : E{NONCE_Nb,PUBKEY_P_B};
}
```

Specifikace druhé role je takřka totožná s tou první. Liší se pouze v identifikátoru směru komunikace, protože tento účastník přenosu přijímá zprávy, které předchozí posílá a generuje své vlastní nonce.

```
P_B(1,3){
VAR: P_A, P_B, NONCE_Nb, NONCE_Na;
<- : E{C[P_A,NONCE_Na],PUBKEY_P_B};
-> : E{C[P_B,NONCE_Na,NONCE_Nb],PUBKEY_P_A} | New(NONCE_Nb);
<- : E{NONCE_Nb,PUBKEY_P_B};
}
End
```

Specifikace vlastností

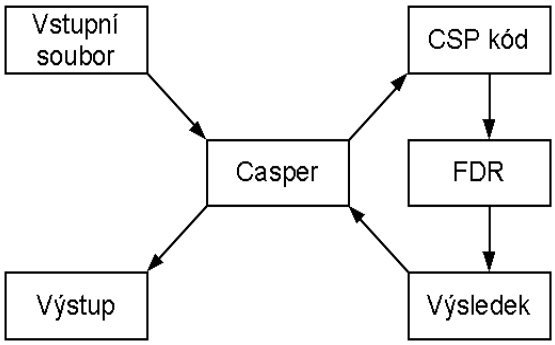
Jelikož příklad ukazuje autentizační protokol, který provádí vzájemnou autentizaci, máme ve specifikaci vlastností dvě podmínky VC. Tyto podmínky kontrolují, zda došlo k autentizaci nejprve P_A vzhledem k P_B a poté obráceně. Strand určuje svými parametry, stejně jako u role, identifikátor a krok jeho specifikace. První podmínka určuje, že po skončení komunikace obou rolí musí být shodné všechny hodnoty klíčů i nonce. Druhá ukazuje, že se musí rovnat i po 3 kroku odesilatele a 2 kroku příjemce.

```
Spec {
 VC.{Strand(1,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]}
 =>{Strand(0,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]},
 VC.{Strand(0,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]}
 =>{Strand(1,2)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]}
}
END
```

3.3 Casper

Jazyk Casper[10] vznikl na univerzitě v Oxfordu za účelem zjednodušení implementace verifikačních procesů pomocí procesní algebry CSP. Tento způsob zápisu kódu nabízí velké možnosti ovšem jeho zápis je příliš složitý a docházelo při implementaci k častým chybám. Proto vznikl jazyk Casper, aby usnadnil práci a poskytl stejnou vyjadřovací sílu. Jak už z předchozích úvodních vět vyplývá Casper překládá svůj kód do CSP kódu a ten následně zpracovává.

Casper používá pro verifikaci model checker FDR. Ten zpracovává vstupní CSP kód a kontroluje zda jsou splněny všechny požadované vlastnosti. Pokud dojde k chybě, kontrola objeví slabinu v protokolu, model checker oznámí cestu kterou došel k chybě a Casper tuto cestu převede do srozumitelnější podoby. Cesta koresponduje s útokem na protokol.



Vstupní soubor

Zdrojový soubor musí definovat samotný protokol a také systém, který jej kontroluje. Proto se soubor skládá ze dvou částí:

- definice protokolu (zasílání zpráv, předpokládané cíle)
- definice systému pro kontrolu (definování rolí, datových typů, schopnosti útočníka)

Definice protokolu

Hlavní částí definice protokolu je definice sekvence posílaných zpráv tzv. popis protokolu. Tato část je uvedena návěštím #Protocol description. Poté následují jednotlivé kroky komunikace. Konvence pro zapisování zašifrované zprávy je {m}{k}. Tento zápis znázorňuje, že zpráva m byla zašifrována klíčem k. Musíme určit jakým způsobem se dozvěděl A o B. V jazyku Casper je možné využít notaci, viděnou na druhém řádku příkladu, že se vynechá název odesilatele, což znamená, A informuje prostředí.

```
#Protocol description
 0.    -> A : B
 1.  A -> B : {na,A}{PK(B)}
```

```
2.  B -> A : {na,nb}{PK(A)}
3.  A -> B : {nb}{PK(B)}
```

Po popisu komunikace je nutné určit seznam použitých proměnných. Opět jsou uvozeny návěštím tentokrát s názvem #Free variables. Datové typy Agent určují aktéry komunikace, nonce představuje Nonce (zvyšuje bezpečnost, takřka v každém protokolu). PK a SK jsou funkce, které vracejí veřejný nebo soukromý klíč daného aktéra. Zápis v této části je pouze deklarativní definice funkcí je provedena později.

```
#Free variables
A, B : Agent
na, nb : Nonce
PK : Agent -> PublicKey
SK : Agent -> SecretKey
```

V další části jsou implementovány procesy. Každý uživatel je reprezentován jedním CSP procesem. Jako první je definováno jméno role, pod tímto názvem bude vystupovat CSP proces s parametry. První parametry označuje uživatele, se kterým je příslušný proces spojen, dále za klíčovým slovem knows následují informace, o kterých uživatel ví. Podle příkladu INITIATOR zná vlastní identitu A nonce na, veřejné klíče PK a svůj soukromý klíč SK(A).

```
#Processes
INITIATOR(A, na) knows PK, SK(A)
RESPONDER(B, nb) knows PK, SK(B)
```

Následuje zapsání požadavků na protokol tzv. Specifikace. Rozlišují se dva druhy požadavků na bezpečnost (Secret) a autentizaci (Agreement). První řádek z příkladu lze vyložit jako A předpokládá, že hodnotu na zná pouze A a B, ekvivalentně pro druhý řádek. Třetí řádek popisuje že A je autentizován k B a oba se dohodli na hodnotách na a nb.

```
#Specification
Secret(A, na, [B])
Secret(B, nb, [A])
Agreement(A, B, [na,nb])
Agreement(B, A, [na,nb])
```

Definice systému:

Druhá část souboru popisuje aktuální systém, který bude testován. Nejprve se definují proměnné, které budou ve verifikaci figurovat. V našem případě jsou to uživatelé a nonce. Definujeme nejen regulérní uživatele, ale i útočníka (Mallory) a jeho příslušný nonce.

```
#Actual variables
Alice, Bob, Mallory : Agent
Na, Nb, Nm : Nonce
```

Po proměnných se definují funkce. V našem případě jsou to funkce pro získání soukromého a veřejného klíče. V příkladu jsou označeny klíčovým slovem symbolic, které označuje, že Casper dosadí vlastní hodnoty. Z hlediska verifikace nás hodnoty klíčů nezajímají.

```
#Functions
symbolic PK, SK
```

Část definice samotného systému přiřazení aktuálních účastníku, kteří se podílejí na komunikaci.

```
#System
INITIATOR(Alice, Na)
RESPONDER(Bob, Nb)
```

Specifikace útočníka. Nejprve označíme útočníka a poté, co všechno útočník zná a co tím pádem může použít k útoku.

```
#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Mallory, Nm, PK, SK(Mallory)}
```

Výstupem souboru je odhalení chyby, kdy Casper vypíše útok ve stejném formátu jako je zadán popis komunikace protokolu. Pár větami vypíše i stav systému, hlavně informaci o tom, který údaj byl odhalen útočníkovi.

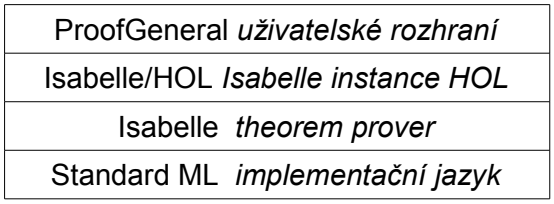
Stejně jako u prvního nástroje Athena, i zde existuje možnost generování zdrojového kódu ze zápisu protokolu. K tomuto účelu slouží COSP-J, ten vytváří z definice protokolu stejně jako v předchozím případě zdrojový kód v programovacím jazyku Java.

3.4 Isabelle

Tento nástroj je vyvíjen na univerzitách v Cambridge a Mnichově, pod vedením Larry Paulsona a Tobiase Nipkowa. Nástroj Isabelle dovoluje zapsat matematické formule ve formálním jazyku a je schopen provést důkaz o platnosti těchto formulí pomocí logického kalkulu. Isabelle má široké pole působnosti, není omezena pouze na bezpečnostní protokoly.

Syntax logiky zapsané v Isabelle je reprezentováno pomocí bezkontextové gramatiky. Zápis bezpečnostních vlastností se provádí v HOL (higher-order logic). Důkaz je generován pomocí Isabelle/HOL. HOL je programovací jazyk, který kombinuje funkcionální programování společně s

logikou. Protokol je definován jako množina traces. Trace má následující formát A sends msg to B. Jsou zde definovány operace, které jsou popsány níže a které rozšiřují množinu zpráv.



Architektura Isabelle

Typy účastníků komunikace (Agents)

- Server S : server
- Friend i : i je přirozené číslo reprezentující, reprezentuje běžného uživatele
- Spy : útočník

Typy operátorů

- parts : obsahuje celé zprávy i zašifrovaný text
- analz : obsahuje celé zprávy i zašifrovaný text, pokud zná klíč
- synth : *obsahuje zprávy, které může útočník získat odposloucháním*

Zpráva může obsahovat

- jméno agenta *A*, *B,...*
- nonces *Na, Nb, ...*
- klíče *Ka, Kb, Kab, ...*
- sloučené zprávy *{msg msg2}*
- hashovaná zpráva *Hash msg,*
- šifrovanou zprávu *Crypt msg*.

Rozlišují se dva typy nonce odhalitelné a neodhalitelné. Pomocí synth (analz msg) lze získat zprávy, které může odvodit útočník ze zprávy msg.

Typy událostí

- Says A B X : A posílá zprávu X agentovi B
- Notes A X : A interně ukládá zprávu X
- Gets A X : A přijímá zprávu X.

Modelování protokolu

U tohoto nástroje, na rozdíl od předchozích, nebude uveden celý zápis protokolu, kód zde napsaný je zaměřen pouze na podstatné části a na vysvětlení principu, implementace. Proměnná

ns_public obsahuje množinu všech dostupných traces. Na začátku je přidána prázdná cesta, druhé pravidlo určuje zprávy poslané útočníkem. Tyto dvě pravidla jsou běžné u všech běžných protokolů.

```
Nil:  "[] ∈ ns_public"
Fake: "[|evsf ∈ ns_public;  X ∈ synth (analz (spies evsf))|]
       ==> Says Spy B X  # evsf ∈ ns_public"
```

Následuje popis komunikace, posílání zpráv mezi účastníky.

```
      (*Alice initiates a protocol run, sending a nonce to Bob*)
NS1:  "[|evs1 ∈ ns_public;  Nonce NA ∉ used evs1|]
       ==> Says A B (Crypt (pubEK B) {Nonce NA, Agent A})
              # evs1  ∈  ns_public"

NS2:  "[|evs2 ∈ ns_public;  Nonce NB ∉ used evs2;
        Says A' B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs2|]
       ==> Says B A (Crypt (pubEK A) {Nonce NA, Nonce NB, Agent B})
             # evs2  ∈  ns_public"

NS3:  "[|evs3 ∈ ns_public;
        Says A  B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs3;
        Says B' A (Crypt (pubEK A) {Nonce NA, Nonce NB, Agent B})
           ∈ set evs3|]
       ==> Says A B (Crypt (pubEK B) (Nonce NB)) # evs3 ∈ ns_public"
```

3.5 Murphi

Murphi [11] je jazyk založený na principu podmínka / akce, který popisuje nedeterministický konečně stavový automat. Stav modelu je dán hodnotami všech globálních proměnných. Počáteční stav je tudíž dán inicializační hodnotou globálních proměnných. Přechody mezi stavy jsou dány pravidly (rules), toto pravidlo se skládá z boolovského výrazu a akce, která se vykoná při splnění výrazu. Tyto akce mohou měnit globální proměnné, čímž změní i stav. V jazyku Murphi jsou definovány procesy, kde každý proces se skládá z několika pravidel. Tyto procesy běží paralelně a komunikují spolu pomocí sdílených proměnných.

Důležitým rysem tohoto nástroje je možnost škálovatelnosti modelu. Při zkoumání protokolu se nám hodí, abychom mohli nejdříve prozkoumat protokol při málo účastnících, a tím zkontrolovat pouze kostru modelu, a poté postupně můžeme přidávat další prvky. Další důležitou vlastností je schopnost automatické symetrické redukce.

Prochází se stavový prostor, pokud se nalezne porušení pravidel, tak program vytiskne uživateli seznam stavů, jak došel k chybě. Způsob kontroly je podobný jako u FDR, ovšem liší se v několika vlastnostech. Rozdíly mezi FDR a Murphi jsou takové, že Murphi využívá sdílené proměnné a poskytuje mnoho podpůrných funkcí, jako symetrickou redukci, hash, reverzní pravidla, konstruktor pro opakování a dále také je zde možnost použít paralelní verzi Murphi. Syntax jazyka je založena na jazyku Pascal s prvky C.

Princip analýzy

Nejdříve je nutné provést definici protokolu. Zápis protokolu je na první pohled složitější než třeba u Atheny nebo Casperu. Musí se definovat formát zpráv, které budou použity z důvodu práce systému se sdílenými proměnnými. Po definici protokolu je nutné namodelovat útočníka. Ten je chápán jako součást systému, která může zahájit komunikaci s řádnými uživateli, smazat zprávu, odposlouchávat všechny zprávy, dešifrovat zprávu, pokud má k dispozici klíč, generovat zprávy použitím odposlechnutých zpráv a počátečních znalostí. Dále je nutné upřesnit požadavky na bezpečnost. Definují se stavy, ve kterých se může nacházet systém, aniž by došlo k porušení bezpečnosti.

Výkonnost systému

Systém provádí analýzu ve velmi krátkém čase. Problém nastává pokud verifikujeme složitější systémy s více uživateli, tím rapidně roste počet stavů. Jednoduché protokoly zkontroluje program řádově v minutách, u složitějších 5 a více uživatelů a složitý protokol i několik hodin.

iniciátorů	*odpovídačů*	*útočníků*	*sítí*	*stavů*	*čas*
1	1	1	1	1706	3.1s
1	1	1	2	40206	82.2s
2	1	1	1	17277	43.1s
2	2	1	1	514550	5761.1s

Needham-Schroeder (Murphi)[11]

Modelování protokolu

Zápis protokolu v jazyku Casper je značně obtížnější než v prvních dvou nástrojích, proto je zde uveden jen nástin práce v tomto nástroji. Pro ilustraci byl vybrán stejný protokol jako v předchozích čili zjednodušený Needham-Schroeder.

Nejprve se nastaví typy, které se budou v programu používat. Konstanty na začátku kódu určují kolik bude účastníků a kolik zpráv může být současně na síti. Výpis je značně zjednodušen, je zde pouze pro ilustrací. Ve skutečnosti je nutné nadefinovat i formát zprávy a ostatní uživatele, včetně útočníka.

```
const
  NumInitiators:   1;
  NetworkSize:     1;
type
  InitiatorId:  scalarset (NumInitiators);
  AgentId:      union {InitiatorId, ResponderId, IntruderId};
  InitiatorStates : enum {I_SLEEP, I_WAIT, I_COMMIT};
  Initiator : record
    state:     InitiatorStates;
    responder: AgentId;
```

```
  end;
var
  net: multiset[NetworkSize] of Message;
  ini: array[InitiatorId] of Initiator;
```

Zde je definice posílání první zprávy v protokolu Needham-Schroeder (A->B: {Na,A}Kb). Příkaz ruleset je ekvivalentem k příkazu for z jiných jazyků tj. cyklí uzavřenou smyčku postupně pro všechny hodnoty. Konkrétně v příkladu máme jenom jednoho iniciátora, tudíž se provede smyčka 1x, agenty máme ovšem 3, takže postupně provádíme pro iniciátora, respondenta a útočníka. Poté následuje pravidlo, při kterém se vykoná následná akce. První podmínka určuje, že iniciátor komunikace musí být ve stavu I_SLEEP (inicializační hodnota). Druhá podmínka určuje, že agent, se kterým chce iniciátor komunikovat, není on sám a poslední podmínka kontroluje, zda se již nevysílá nějaká zpráva.

```
ruleset i: InitiatorId do
  ruleset j: AgentId do
    rule "initiator starts protocol"
      ini[i].state = I_SLEEP &
      !ismember(j,InitiatorId) &
      multisetcount (l:net, true) < NetworkSize
    ==>
```

Po splnění podmínky se začne vykonávat požadovaná akce. Nejdříve se vytvoří zpráva, definice formátu zprávy není zde uvedena, ale je zřejmá z následné inicializace. Zpráva obsahuje položku mType, která určuje typ zprávy, v tomto případě pro zjednodušený NS protokol jsou zde tři typy. Tyto typy určují, která data se zasílají. Na závěr se nastaví stav iniciátora na čekání na odpověď.

```
    var
      outM: Message;
    begin
      undefine outM;
      outM.source := i;   outM.dest := j;   outM.key := j;
      outM.mType := M_NonceAddress;   outM.nonce1 := i;   outM.nonce2  := i;
      multisetadd (outM,net);
      ini[i].state := I_WAIT;
      ini[i].responder := j;
    end;
  end;
end;
```

Pro úplnost by muselo být doplněno ještě jedno pravidlo pro přijetí a odeslání zprávy. Tím by byly definovány zprávy zasílané od A. Poté by bylo třeba ještě definovat pravidla pro uživatele B a chování útočníka.

3.6 Porovnání nástrojů

Pro verifikaci bezpečnostních protokolů je v dnešní době k dispozici velké množství nástrojů. Každý má své pro i proti a jeho použití záleží na mnoha faktorech, od jaké vlastnosti chceme verifikovat, kolik času chceme obětovat učení se dalšího jazyka na popis, jak je náchylný k chybám při implementaci atp. Liší se také uživatelskou přívětivostí, většina nástrojů má pouze konzolové uživatelské rozhraní, s grafickým uživatelským rozhraním je možné se setkat hlavně v komerčních nástrojích. Devadesát procent všech nástrojů, hlavně nekomerčních, je určena pro operační systém linux.

Pokud mohu dle vlastní zkušenosti jazyky porovnat podle snadnosti práce s ním, tak bych pořadí viděl takto. Nejsnáze se mi pracovalo v nástroji Casper, dále pak v Athena, Murphi a nejobtížnější dle mého názoru byl jazyk Isabelle. Podle rozšíření a nasazení v praxi však považuji právě nástroj Isabelle za nástroj s největšími možnostmi. Casper se jeví jako vhodný nástroj s výbornými vlastnostmi, který zachovává výhody CSP a poskytuje značné zjednodušení zápisu. Všechny čtyři nástroje pocházejí z věhlasných světových univerzit. Liší se ovšem jejich podporou a dalším rozvojem. Jako nejmohutnější projekt s největší podporou a nejlepšími výsledky a stále pokračujícím vývojem je nástroj Isabelle, jednak výzkum probíhá na dvou univerzitách v mnohačlenných týmech a dále pak také nejnovější verze je z roku 2007 Isabelle 2007. Trochu horší podmínky jsou u nástroje Casper, ale tento nástroj také v roce 2007 zaznamenal nové aktualizace. Nástroj mírně zaostává v dostupných materiálech, ovšem samotný jazyk je dobře zdokumentován, jsou k dispozici i vzorové příklady specifikace protokolů. Systém Athena je vlastně práce jednoho člověka (Dawn Song), která vymyslela algoritmus a systém zaintegrovala do SyMP, který však již pět let setrvává ve verzi 0.3. Nástroj Murphi také již delší dobu nebyl aktualizován, vývoj probíhal v letech 1992 až 1999. Nechci tvrdit, že vývoj na těchto nástrojích již nepokračuje, že byly tyto programy špatné, naopak během mnoha let bylo jejich prostřednictvím ověřeno velké množství protokolů.

	Jednoduchost	Flexibilita	Možnosti	Prostředí	Tok	Komponenty
Athena	+	–	–	neměnné	Role	neformální
Casper	+	+	+	Explicitní	Role	formální
Isabelle	+–	+	+	neměnné	Trace	formální
Murphi	+	+ –	+ –	Implicitní	tok	deklarace

Tabulka vlastností [3]

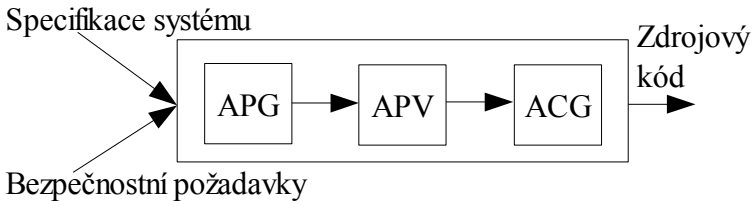
4 Athena – vybraný nástroj

4.1 O nástroji

Athena verifikuje bezpečnostní protokoly pomocí kontroly modelem, podobně jako jiné nástroje, například FDR, Murphi a Brutus. Tyto nástroje začínají z inicializačního stavu a procházejí všechny možné sekvence účastníků a útočníka. Nástroje vyhledají chyby v protokolech, ovšem mají dva problémy. Jedním je omezený počet účastníků a exploze stavového prostoru, kdy dochází k prudkému nárůstu kontrolovaných stavů a tím se komplikuje, či znemožňuje verifikace jistých protokolů. I když předem zmíněné nástroje používají nejrůznější metody k redukci stavového prostoru, stále je tato skutečnost problémem.

Nástroj je implementován v programovacím jazyku SML, který je popsán v následující podkapitole. Dnes již není tak hojně používaný, ovšem pro účel aplikace je více než vhodný. Další samostatnou podkapitolu jsem se rozhodl věnovat problematice metody SSM, které tento nástroj využívá a jejíž pochopení může zvýšit schopnost pochopit tento nástroj.

Athena představuje balíček třech komponent, které je možno svázat dohromady a používat propojeně. Jedná se o APV, APG a ACG. První zmíněný APV (Automatic Protocol Verifier) slouží k verifikaci bezpečnostního protokolu a tímto nástrojem se převážně zabývá tato práce. APG (Automatic Protocol Generator) slouží ke generování bezpečnostních protokolů, které vyhovují požadovaným specifikacím. Poslední součástí je ACG (Automatic Code Generator), který podle specifikace protokolu vytvoří kód v programovacím jazyku Java. Vzájemnou provázanost znázorňuje následující obrázek.



Systém Athena [13]

V následujících dvou odstavcích se věnuji systémům APG a ACG, které sice nejsou předmětem práce, ale souvisí s danou problematikou. APG má jako jediný ze zmíněné trojice grafické uživatelské rozhraní. Jelikož je napsán v jazyce Java je multiplatformní a neměl jsem problém spustit jej jak v systému Windows, tak v systému Linux. Obsluha nástroje je naprosto triviální a zvládne jí každý. V hlavním okně se nastaví parametry protokolu, a poté se již může spustit generování protokolů. Mezi vlastnosti, které můžeme specifikovat patří počet účastníků komunikace, jaký způsob autentizace bude proveden, typ klíčů (symetrický / asymetrický) a další. Po spuštění

generování se v adresáři aplikace vytvoří soubor, který obsahuje vygenerované protokoly. Nástroj umožňuje zápis více protokolů do jednoho souboru. Generování probíhá velmi rychle a není problém během minuty vygenerovat 200MB protokolů. Počet vygenerovaných protokolů nebo vytvořených dat se nikterak nezobrazuje, je tedy nutné hlídat velikost vytvořených dat a včasně proces ukončit.

Dále bych se stručně zmínil o ACG. Jedná se o konzolovou aplikaci, která je opět vytvořena v Javě a která generuje javovský kód. Jako vstup slouží dva soubory, které specifikují protokol. První soubor má tvar stejný, jako vstupní soubor použitý k verifikaci. Čili popis komunikace a zároveň specifikace vlastností, které by protokol měl splňovat. Soubor druhý je zkrácený zápis souboru a je ekvivalentní s výstupem z generátoru protokolů (APG). Výsledkem je sada souborů (vždy pro každého účastníka komunikace jeden), které obsahují třídy popisující chování dané protokolem.

4.2 SML

K implementaci jsem si vybral nástroj Athena, který je vytvořen pomocí překladače SML. Pro bližší pochopení tohoto nástroje by bylo dobré se zmínit i o tomto programovacím jazyku. Už jen z toho důvodu, že překladač je nutný ke spuštění této aplikace a je tedy vhodné si o něm říci pár slov.

Jedná se o rodinu funkcionálních jazyků ML, mezi další funkcionální jazyky například Haskell a Lisp. Funkcionální jazyky se používají převážně v akademických kruzích například pro modelování a formální verifikaci. Programovací jazyk Standard ML (SML) byl vyvinut v 80tých letech na univerzitě v Edinburghu. V roce 1997 byl modifikován na dnes používanou verzi s označením SML97. Od této doby bylo vyvinuto několik implementací (překladačů) mezi nejznámější patří Moscow ML, SML.Net, SML# a SML/NJ. Poslední zmiňovaný byl použit i při implementaci nástroje Athena. Následující část obsahuje příklady, které mohou být nápomocny při detailnějším zkoumání kódu verifikátoru.

```
val x = 3;                // definice proměnné x s hodnotou 3
fun mul2 x = x * 2;       // vytvoří funkci která násobí parametr 2
mul2(x);                  // zavolání funkce s parametrem x (výsledek 6)
use "program.sml";        // spustí program ze souboru program.sml
print "Zprava\n";         // vytisknutí textu na vystup
datatype turn = on|off;   //sml
structure StringKey = struct      // příklad struktury
     type ord_key = string
     val compare = String.compare
end
```

Nejdůležitější pro verifikaci je příkaz `use`, kterým se athena v sml spouští. V jazyku se explicitně neuvádějí typy, ale i přesto nabývají jistých typů a operace jsou typové. Například podmínka if umožňuje v podmínce použití pouze boolovské hodnoty. Jazyk umožňuje definovat nejrůznější struktury, pole, vektory, využívá rekurzi, výjimky, polymorfismus atd.

4.3 SSM

Athena využívá k verifikaci kontrolu modelem. K dokázání správnosti protokolů je použita metoda Strand Space Model (*SSM*) [32] autorů Thayer, Herzog a Guttman, kteří ve zmíněném dokumentu ukázali, jak je možné manuálně provést verifikaci. Athena představuje automatizaci tohoto počinu.

Strand space se skládá z množiny strandů (vláken). Každé vlákno je reprezentováno skupinou událostí jednotlivých účastníků. Každý legitimní účastník představuje jedno vlákno, pokud ovšem při simulaci dojde k souběžnému běhu více procesů jednoho účastníka, vytvoří se pro každý běh nové vlákno. Útočník je také reprezentován svými vlastními vlákny. Dalším pojmem důležitým při popisu modelu je tzv. bundle (balík). Ten obsahuje všechna vlákna potřebná k poslání a přijetí zprávy. Vlákna jak od legitimních účastníků, tak od útočníka. Vlákna je možné popsat jako sekvenci přijatých a vyslaných zpráv jednoho příjemce. Balík tedy určuje komunikaci mezi těmito vlákny. Pomocí těchto komponent a pravidel, která popisují verifikovanou vlastnost se provede verifikace.

Jelikož verifikace využívá kontrolu modelem, musí se nástroj potýkat také s problémem exploze stavů. Athena využívá několik metod k redukci stavového prostoru [12]. Prvním případem redukce je zamezení asynchronních přechodů mezi stavy, tím se značně omezí počet stavů. Struktury stavů a přechodů mezi nimi mají přesně určené vztahy, což má za důsledek efektivní reprezentaci stavů. Dalším způsobem redukce je použití reprezentace přechodů mezi stavy symbolicky místo explicitně uváděných hodnot, což zápis i reprezentaci značně zjednodušuje. Posledním vylepšením zaručujícím dostatečnou redukci stavového prostoru, je použití zpětného prohledávání. Výhodou oproti dopřednému prohledávání stavů je, že se nemusí začínat se všemi účastníky komunikace, ale vychází se pouze z jednoho vlákna a přidávají se pouze nezbytně nutná vlákna dána striktně omezenými přechody. K redukci stavového prostoru se také používá včasné prořezávání stavového prostoru, tj. omezení počtu nedosažitelných stavů, před jejich expanzí.

4.4 Specifikace vlastností protokolu

Pro správnou verifikaci je nutné určit, které vlastnosti budeme ověřovat. Musíme nutně dokonale chápat jak protokol funguje, a které bezpečnostní funkce nám zajišťuje. V tomto oddílu jsou popsány vlastnosti[13], které nám umožňuje Athena ověřovat. Zápis do zdrojového kódu je velmi podobný a je popsán v následující kapitole.

K popisu protokolu se používají vlákna strandy. Legitimní účastníci jsou reprezentováni typem s následující syntaxí Role[argumenty], např. Resp[A, B, Na, Nb]. A, B představují identifikace jednotlivých účastníků, v tomto případě Alice a Boba, proměnné Na a Nb obsahují hodnoty Nonce vygenerovaná od A a od B. Útočník je také reprezentován vláknem, které představuje počáteční znalosti o komunikaci. Útočník zná hodnoty A a B, ví tudíž kdo s kým hodlá komunikovat a může

tuto hodnotu podvrhnout. Dále pak zná hodnoty veřejných klíčů všech zúčastněných stran, tyto hodnoty jsou běžně dostupné. Útočník může zaznamenávat zprávy mezi legitimními účastníky a generovat zprávy nové. Nové zprávy mohou obsahovat inicializační hodnoty a nebo části zprávy, které se zaznamenaly. Tyto zprávy postupně vytvářejí nové vlákna použité při verifikaci.

Pro zkoumání vlastností protokolu se zaměříme převážně na autentizaci a utajení. Vlastnosti popisujeme v tomto formátu. V následující kapitole je popsáno, jak se tyto formule zapisují do zdrojového kódu.

A autentizuje B, unikátnost je dána čerstvostí nonce Nb.

$$\forall Cp.Cp : (Init[A, B, S, Na, Nb] \Rightarrow Resp[A, B, S, Na, Nb])$$

B autentizuje A, unikátnost je dána čerstvostí nonce Na.

$$\forall Cp.Cp : (Resp[A, B, S, Na, Nb] \Rightarrow Init[A, B, S, Na, Nb])$$

Distribuce klíče s autentizací pomocí serveru

$$\forall Cp.Cp : (Init[A, B, S, Na, Kab] \Rightarrow Srv[A, B, S, Na, Kab])$$
$$\forall Cp.Cp : (Resp[A, B, S, Nb, Kab] \Rightarrow Srv[A, B, S, Nb, Kab])$$

Distribuce klíče bez autentizace pomocí serveru

$$\forall Cp.Cp : (Init[A, B, S, Na, Kab] \Rightarrow Resp[A, B, S, Na, Kab])$$
$$\forall Cp.Cp : (Resp[A, B, S, Nb, Kab] \Rightarrow Init[A, B, S, Nb, Kab])$$

Utajení klíče (Θ symbolizuje množinu znalostí útočníka)

$$\forall Cp.Cp : (Init[A, I, S, Na, Kab] \Rightarrow Kab \notin \Theta)$$
$$\forall Cp.Cp : (Resp[I, B, S, Nb, Kab] \Rightarrow Kab \notin \Theta)$$

4.5 Popis jazyka

Vstupní soubor se dá rozdělit na dvě části. První popisuje chování protokolu a druhá vlastnosti, které se mají verifikovat. První část je rozdělena na úseky odpovídající jednotlivým účastníkům a jejich akcím. Účastníky komunikace (Principal) musíme pojmenovat `P_A`, `P_B` a `P_S`, sama autorka uvádí [13], možnost změny zápisu při změně parseru aplikace. Pro ukázkové aplikace toto však vůbec nevadí a je pouze nutné na toto omezení pamatovat.

Pro zápis veškerých hodnot je určena konvence, jejíž dodržení zvyšuje čitelnost kódu i pochopení případných problémů. Přehledná čitelnost kódu je více než nutná, z důvodu špatné sdílnosti překladače sml. Označení `P_X` již bylo zmíněno dříve, označuje legitimní účastníky komunikace. Nonce se označují logicky `NONCE_yx`, kdy yx představuje označení nonce, je vhodné volit hodnoty shodné s zápisem ve formálním tvaru například Nb, Ma .... Důležitou částí zápisu je zápis klíčů. Athena rozlišuje 4 druhy klíčů veřejný `(PUBKEY_P_X)`, soukromý `(PRIVKEY_P_X)`, symetrický `SYMKEY(P_X,P_Y)` a klíč relace `SESKEY_XYZ`. První dva klíče se používají při asymetrické kryptografii a symbol X prezentuje účastníka, kterému náleží. Symetrický klíč slouží k symetrickému šifrování a symboly X a Y určují, kteří účastníci tento klíč vlastní a tudíž mohou sním

šifrovat či dešifrovat. Poslední klíč se většinou generuje během běhu protokolu a označení XYZ může být libovolné, ovšem pro přehlednost by bylo vhodné opět využít stejného označení jako u formálního popisu protokolu např. Kab.

Pro bližší pochopení zápisu kódu je nutné uvést příklad, na kterém je možné demonstrovat zápis protokolu. Celý příklad je uveden v následující podkapitole.

```
P_B(1,5){
VAR: P_A, P_B, P_S, NONCE_Na, NONCE_Nb;
<- :  P_A;
-> :  NONCE_Nb | New(NONCE_Nb);
<- :  E{NONCE_Nb, PRIVKEY_P_A};
-> :  E{C[P_A,E{NONCE_Nb, PRIVKEY_P_A}], PRIVKEY_P_B};
<- :  E{NONCE_Nb, PRIVKEY_P_B};
}
```

První část obsahuje specifikaci jednoho účastníka v našem případě `P_B` (Boba). Jak již je zmíněno dříve, účastník je označen identifikátorem `P_B`, následující dva parametry označují 1. pořadové číslo účastníka, číslované od 0, v tomto případě 0 = `P_A` a 1 = `P_B`. Druhý parametr označuje počet zpráv, které účastník přijímá nebo odesílá, tj. počet řádků ve specifikaci komunikace daného subjektu. Následující řádek začínající řetězcem var, představuje deklaraci proměnných, které daný účastník používá. Zde je nutné uvést označení jednotlivých účastníků (`P_X`), nonce (`Nonce_Nx`) a pokud je distribuován klíč relace je nutné uvést i notaci `SESKEY_Kab`. Na začátku každého řádku představující přijmutí či odeslání zprávy se nachází šipka určující právě tok dat (→ poslání zprávy, ← příjem zprávy). Pokud při posílání zprávy dochází zároveň k vygenerování nové hodnoty například čerstvý nonce nebo klíč relace je nutné uvést za specifikaci zprávy např. klauzuli „`| new (Nonce_Nb)`“. Pokud je třeba vytvořit v jednom kroku více hodnot oddělují se čárkami.

Pro zápis pravidel je nutné se obeznámit se dvěmi nezbytně nutnými konstrukcemi a to zápis konkatenace a šifrování. Libovolný počet hodnot lze zřetězit pomocí zápisu `C[A,B,C,D,....]`, pokud potřebujeme popsat proces zašifrování zvolíme zápis `E{zpráva,klíč}`. Zpráva může obsahovat pouze jeden element, proto pokud je potřeba zašifrovat vícero hodnot je nutné nejdříve hodnoty zřetězit a následně zašifrovat. Po zápisu všech legitimních účastníků komunikace, následuje specifikace vlastností protokolu. Pravidla pro popis jsou uvedena v předcházející kapitole, následující příklad demonstruje zápis těchto pravidel v praxi.

```
VC.   {Strand(1,5)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]} =>
      {Strand(0,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]}
VC.   {Strand(1,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)],
      Strand(24,1)[(ANY_secretX,NONCE_Nb0)]} => |,
```

První dva řádky z vloženého kódu reprezentují tzv. autentizační formuli (ověřují zda Bob autentizuje Alici). Strand představují vlákna, kterým se věnuje kapitola o SSM. První parametr udává identifikaci uživatele uvedená ve specifikaci chování protokolu a druhý argument popisuje pořadí

zprávy. Následuje výčet parametrů, kterým se přiřadí názvy pomocí, kterých se analyzuje výstup a které se používají k verifikaci. Označení může být libovolné vyjma slova I, které označuje útočníka. Druhá bezpečnostní specifikace označuje kontrolu utajení. Kontroluje se zde zda útočník nezjistil hodnotu Nb0. Hodnota vlákna (Strand) 24 označuje útočníka. Jakákoliv hodnota účastníka větší nežli 10 je pokládána za útočníka.

4.6 Popis verifikace

V předchozí podkapitole bylo uvedeno, jakým způsobem se zapisuje verifikační kód v programu Athena. Tato část popisuje celkový popis verifikace od zadání kódu až po samotné zpracování výsledků. Pro ukázku verifikace byl vybrán protokol Woo Lam Π 3, který je popsán blíže v části zabývající se všemi protokoly.

Komunikace 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. $S \Rightarrow B : \{A, Nb\}Kbs$

Útok na protokol:

1. I(A)⇒B: A
2. B⇒I(A): Nb
3. I(A)⇒B: Nb
4. B⇒I(S): {A,Nb}Kbs
5. I(S)⇒B: {A,Nb}Kbs

Zdrojový kód Athena:

```
1:1 Protocol: WooLamPi3
  P_A(0,3){
  VAR: P_A,P_B,P_S,NONCE_Nb;
  -> : P_A;
  <- : NONCE_Nb;
  -> : E{NONCE_Nb,SYMKEY_(P_A,P_S)};
 }

  P_B(1,5){
  VAR: P_A,P_B,P_S,NONCE_Nb,ANY_X;
  <- : P_A;
  -> : NONCE_Nb | New(NONCE_Nb);
  <- : ANY_X;
  -> : E{C[P_A,ANY_X],SYMKEY_(P_B,P_S)};
  <- : E{C[P_A,NONCE_Nb], SYMKEY_(P_B,P_S)};
  }
```

```
  P_S(2,2){
  VAR: P_A,P_B,P_S,NONCE_Nb,ANY_X;
  <- : E{C[P_A,E{NONCE_Nb,SYMKEY_(P_A,P_S)}],SYMKEY_(P_B,P_S)};
  -> : E{C[P_A,NONCE_Nb], SYMKEY_(P_B,P_S)};
  }
End
Spec {
  VC. {Strand(1,5)[(P_A,A0),(P_B,B0),(P_S,S0),(NONCE_Nb,Nb0)]} =>
   {Strand(0,3)[(P_A,A0),(P_B,B0),(P_S,S0),(NONCE_Nb,Nb0)]}
}
END
```

Vstupní kód musí být ohraničen příkazy BEGIN a END. Následují 2 části programu specifikace komunikace a specifikace verifikovaných vlastností. Část popisující komunikaci mezi účastníky je umístěna mezi návěští `x:x Protocol: Nazev_Protokolu` a `End`. Číselné označení x:x nemá na funkci kódu žádný vliv, pouze v případě pokud v jednom souboru obsahuje více protokolů umožňuje tato notifikace protokoly odlišit. Název protokolu slouží taktéž pouze k identifikaci a nemá žádný sémantický význam.

Jednotliví účastníci jsou názorně označeni identifikátory P_A, P_B a P_S což odpovídá označení ve formálním zápisu. Při psaní kódu vzniká redundance, protože každá zpráva se zde uvádí dvakrát, poprvé u odesilatele a podruhé u příjemce. Detailnější popis jednotlivých akcí je uveden v předchozí kapitole. Na tomto konkrétním příkladu lze ukázat, jakým způsobem se používá direktiva new, kdy nonce Nb vygeneruje Bob.

Verifikační pravidlo je zde pouze jedno a kontroluje se zda protokol splňuje podmínky k jednocestné autentizaci. Zápis dvou strandů určuje vlákna, která kontrolujeme. První parametr identifikuje účastníka (číslo musí být shodné s označením při popisu chování protokolu), druhý poté označuje délku posloupnosti vlákna. V hranatých závorkách jsou uvedeny hodnoty, jejichž hodnota se bude kontrolovat, zda skutečně pokud Alice pošle nonce Nb tento nonce obdrží Bob a zda nedojde k nějaké změně během přenosu, což by indikovalo útok.

Výstup programu:

```
 checking spec < 1 > :
 ----false: number of states explored: 3
 inStrand
 state( sb(1, Strand (Sname 0, len: 5, Role 1)
),
  goalset: ,  env:
(AnyV Strand(0).X, Nonce NonceC Nb0):: (Nonce NonceV Strand(0).Nb, Nonce
```

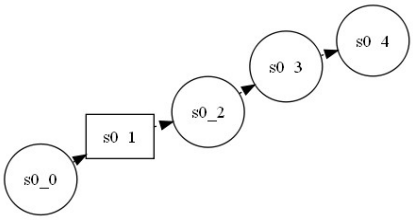
```
NonceC Nb0):: (Principals Pvar Strand(0).S, Principals PrincipalC S0):: (Principals Pvar Strand(0).B, Principals PrincipalC B0):: (Principals Pvar Strand(0).A,Principals PrincipalC A0),

 porderset: pset(Interm(Nonce NonceC Nb0, Node (1, Sname 0))
  ::Interm({Concat [Principals PrincipalC A0, Nonce NonceC Nb0]}_(Symkey (PrincipalC B0, PrincipalC S0)), Node (3, Sname 0)),
Subterm(Nonce NonceV TempName (Nb), Node (1, Sname 0)),
p->c(Node (1, Sname 0), Node (2, Sname 0))
 ::p->c(Node (1, Sname 0), Node (2, Sname 0))
 ::p->c(Node (3, Sname 0), Node (4, Sname 0))
 ::p->c(Node (3, Sname 0), Node (4, Sname 0))
 ::p->c(Node (2, Sname 0), Node (3, Sname 0))
 ::p->c(Node (1, Sname 0), Node (2, Sname 0))
 ::p->c(Node (0, Sname 0), Node (1, Sname 0))))
 = false
a counter example!
 Usrtime: 15 ms!
 Systime: 0 ms!
 GCtime: 0 ms!
 ~~~~~~~~~ True protocols:
  0
 ~~~~~~~~~ Exceed range protocols:
  0
 ~~~~~~~~~ Unification Problem protocols:
  0
 ~~~~~~~~~ Finish ~~~~~~~~
```

Pro analýzu výstupu bude dobré si definovat dvě veličiny subterm a interm. Term A je subtermem termu B, pokud A je obsažen v podstromu B.Term A je intermem termu B, pokud A je obsažen v B v podobě konkatenace. Př. Term A,B,C,{D,E}Kab má tyto subtermy A, B, C, D, E, Kab, zatímco seznam intermů vypadá takto A, B, C, {D,E}Kab.

Pro popis chyby je nutné popsat jednotlivé části. Blok uvozený řetězcem `state` popisuje vlákna, ve kterých dochází k chybě. Tři parametry označují nejdříve identifikaci vlákna sname, dále len délku řetězce a poslední argument role, určuje účastníka. Identifikace se shoduje se zápisem v specifikaci komunikace. Podle identifikace vlákna, v příkladě 0, se určí hodnoty proměnných svázaných s tímto vláknem. Hodnoty se nacházejí v bloku uvozeném env. Z příkladu je možné vyčíst tyto hodnoty Strand(0,3)[(P_A,A0),(P_B,B0),(P_S,S0),(NONCE Nb,Nb0),(NONCE Na,Na0), (Any_X,Nb0)]. Poslední zmíněný určuje náhradu hodnoty Any_X za hodnotu Nb, která představuje útok. Výstup z programu generuje posloupnost útoku (Dot soubor), který je možné zobrazit pomocí

programu Graphviz, který je opět multiplatformní. Ovládání programu je triviální, pouze se vybere soubor se vstupními daty a po zvolení formátu výstupu se vygeneruje výsledný graf.



Woo Lam Pi3 - útok

Na obrázku je zobrazen útok. Kruhem jsou znázorněny regulérní stavy, ve kterých se systém nachází. Obdélník určuje stav ovlivněný útočníkem. V konkrétním případě se jedná o změnu hodnoty {Nb}Kas na hodnotu Nb. Výsledný útok má tedy tvar:

1. I(A)⇒B: A
2. B⇒I(A): Nb
3. **I(A)⇒B: Nb**
4. B⇒I(S): {A,Nb}Kbs
5. I(S)⇒B: {A,Nb}Kbs

4.7 Symboly použity při popisu protokolů

A: účastník zahajující komunikace, nejčastěji klient označovaný jako Alice

B: účastník, se kterým chceme komunikovat (server nebo další klient), označovaný jako Bob

S: důvěryhodná třetí strana TTP (Trusted Third Parties), server kterému účastníci důvěřují

I: účastník, který se snaží narušit regulérnost komunikace mezi A a B. Útočník (Intruder).

I(A): útočník se vydává za Alici, analogicky I(B), I(S)

A->B: A posílá zprávu B

Na, Nb: nonce, náhodná hodnota, která je vygenerována účastníky a,b ....

Ma, Mb: ekvivalent Na a Nb, používá se při popisu útoku. Představuje Na a Nb v další relaci.

Kab: klíč sdílený mezi Alicí a Bobem

{Zpráva}Kab: zpráva je šifrována pomocí klíče Kab

dec(Na): tento obrat popisuje transformaci nonce na jinou hodnotu, zajišťuje další bezpečnostní prvek, zabraňuje útoku opakováním. Bývá nejčastěji uváděno +1 nebo -1, tato hodnota bývá vždy zašifrována. Používá se k zjištění čerstvosti klíče.

Kpa,Ksa : dvojice klíčů při použití asymetrické kryptografie Kpa (Public) veřejný klíč Alice, Ksa (Secret) soukromý klíč Alice.

5 Implementované protokoly

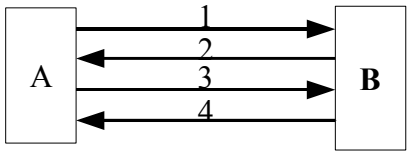
5.1 Andrew Secure RPC

Autoři: Mahadev Satyanarayanan

Publikováno: 1989, [19]

Popis: navržen pro souborový systém AFS, zajišťuje výměnu čerstvého sdíleného klíče bez TTP

1. A⇒B: A,{Na}Kab
2. B⇒A: {dec(Na),Nb}Kab
3. A⇒B: {dec(Nb)}Kab
4. B⇒A: {K'ab,N'b}Kab

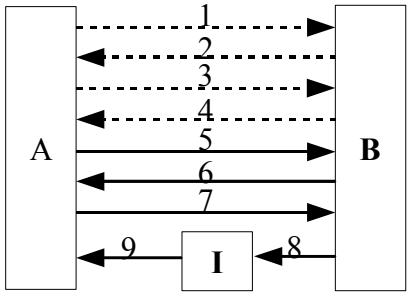


Útok na protokol:

Autoři: Michael Burrows, Martin Abadi, Roger Needham

Publikováno: 1989, [20]

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



Popis útoku:

Útok spočívá v prodloužení platnosti klíče. Útočník se vmísí mezi Alici a Boba a pokaždé, když bude Bob chtít poslat Alici nový klíč, podstrčí zprávu se starým klíčem, kterou zachytil při první výměně. Takto může útočník donekonečna prodlužovat platnost klíče, čímž poruší podmínku pro jeho čerstvost. Klíč však útočník nezjistí, ten musí určit jinak, ovšem s prodlužující dobou platnosti tohoto klíče se šance na odhalení zvětšují.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
B -> A : {K'ab, N'b}Kab
A -> B : A, {Ma}Kab
B -> A : {dec(Ma), Mb}Kab
A -> B : {dec(Mb)}Kab
I -> A : {K'ab, N'b}Kab
```

Počet stavů: 27

Doba verifikace: 47 ms

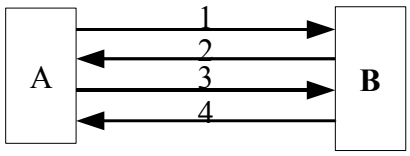
5.2 Andrew Secure RPC - BAN

Autoři: Michael Burrows, Martin Abadi, Roger Needham

Publikováno: 1989, [20]

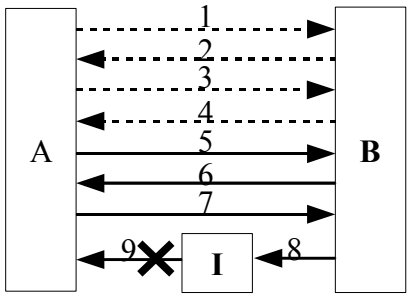
Popis: navržen pro souborový systém AFS, zajišťuje výměnu čerstvého sdíleného klíče bez TTP.

```
1. A⇒B: A,{Na}Kab
2. B⇒A: {dec(Na),Nb}Kab
3. A⇒B: {dec(Nb)}Kab
4. B⇒A: {K'ab,N'b,Na}Kab
```



Útok na protokol: Nenalezen (nepublikován)

```
1. A⇒B: A,{Na}Kab
2. B⇒A: {dec(Na),Nb}Kab
3. A⇒B: {dec(Nb)}Kab
4. B⇒A: {K'ab,N'b,Na}Kab
5. A⇒B: A,{Ma}Kab
6. B⇒A: {dec(Ma),Mb}Kab
7. A⇒B: {dec(Mb)}Kab
8. B⇒I(A): {K'ab,M'b,Ma}Kab
9. I(A)⇒B: {K'ab,N'b,Na}Kab
```



Popis útoku:

Tento protokol upravuje předešlou nalezenou chybu, publikovanou v předchozí podkapitole. Ochrana je jednoduchá a to přidáním hodnoty Nonce Na do zašifrované zprávy číslo 4. Tím se zajistí, že útočník nemůže přeposlat dříve zachycenou zprávu. Jak je patrné z formálního výpisu, nelze již použít zprávu odposlechnutou v prvním průchodu protokolu, přidáním nonce Na, které je pro každou relaci jiné. Při návrhu komunikace je tedy nutné připojit k novým vygenerovaným datům připojit i hodnotu, která by přesně identifikovala danou relaci a tudíž nemohla být zneužita při dalším běhu protokolu.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 16

Doba verifikace: 18 ms

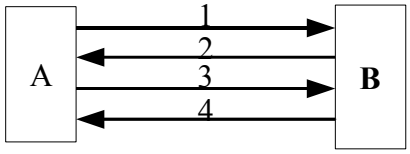
5.3 Andrew Secure RPC – BAN konkrétní

Autoři: Michael Burrows, Martin Abadi, Roger Needham.

Publikováno: 1990, [26]

Popis: zjednodušení původního AS RPC protokolu, zajišťuje výměnu sdíleného klíče bez TTP.

1. A⇒B: A,Na
2. B⇒A: {Na,K'ab}Kab
3. A⇒B: {Na}K'ab
4. B⇒A: Nb

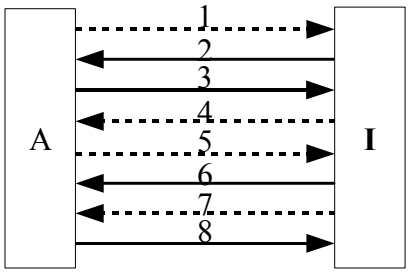


Útok na protokol:

Autoři: Gavin Lowe

Publikováno: 1996, [27]

1. A⇒I(B): A,Na
2. **I(B)⇒A: B,Na**
3. **A⇒I(B): {Na,K'ab}Kab**
4. I(B)⇒A: {Na,K'ab}Kab
5. A⇒I(B): {Na}K'ab
6. **I(B)⇒A: {Na}K'ab**
7. I(B)⇒A: Nx
8. **A⇒I(B): Nb**



Popis útoku:

Útok funguje na základě přeposílání upravených zpráv zpět k legitimnímu účastníkovi, který funguje jako orákulum (zprávy které mu jsou zaslány zašifruje a pošle zpět). Zjednodušeně řečeno se jedná o dva běhy protokolu zároveň, pokud chce Alice komunikovat s útočníkem, začne komunikovat útočník zároveň s Alicí.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.1. A -> I(B) : A, Na
b.1. I(B) -> A : B, Na
b.2. A -> I(B) : {Na, K'ab}Kab
a.2. I(B) -> A : {Na, K'ab}Kab
a.3. A -> I(B) : {Na}K'ab
b.3. I(B) -> A : {Na}K'ab
a.4. I(B) -> A : Ni
b.4. A -> I(B) : Nb
```

Počet stavů: 4

Doba verifikace: 16 ms

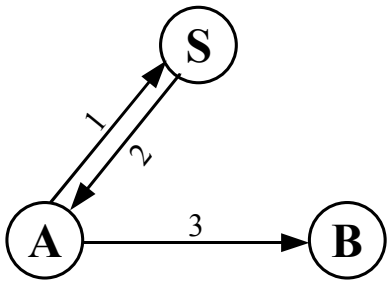
5.4 Denning-Sacco

Autoři: Dorothy Elizabeth Denning, Giovanni Maria Sacco

Publikováno: 1981, [21]

Popis: Vzájemná autentizace s použitím důvěryhodné třetí strany

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

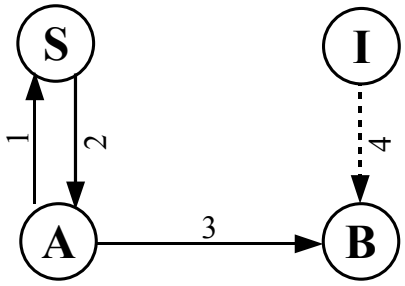


Útok na protokol:

Autoři: Gavin Lowe

Publikováno: 1997, [22]

```
1. A⇒S: A,B
2. S⇒A: {B,Kab,T,{Kab,A,T}Kbs}Kas
3. A⇒B: {Kab,A,T}Kbs
4. I(A)⇒B: {Kab,A,T}Kbs
```



Popis útoku:

Útočník přeposílá poslední zachycenou zprávu a tuto zprávu sám znovu přepošle Bobovi. Ten si myslí, že Alice se snaží navázat další spojení a útočníka autentizuje. Útočník sice nezná heslo, ale byl autentizován, čehož může dále využít. Oprava je možná pomocí přidání nonce handshaku jak navrhl Lowe ve své modifikaci.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
1. A -> S : A, B
2. S -> A : {B, Kab, T, {Kab, A, T}Kbs}Kas
3. A -> B : {Kab,A, T}Kbs
4. I(A) -> B : {Kab,A, T}Kbs
```

Počet stavů: 3

Doba verifikace: 16 ms

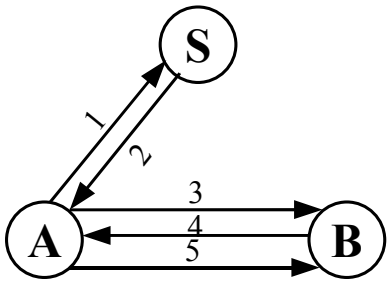
5.5 Denning-Sacco - Lowe modifikace

Autoři: Gavin Lowe

Publikováno: 1997, [22]

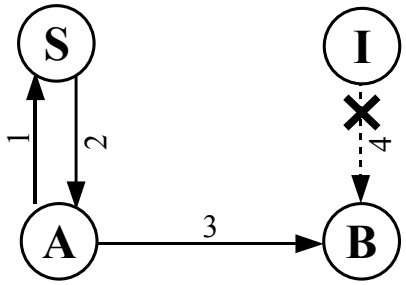
Popis: Upravená verze D-S, přidán nonce handshake

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: {dec(Nb)}Kab



Útok na protokol: Nenalezen (nepublikován)

1. A⇒S: A,B
2. S⇒A: {B,Kab,T,{Kab,A,T}Kbs}Kas
3. A⇒B: {Kab,A,T}Kbs
4. **I(A)⇒B: {Kab,A,T}Kbs**
5. **B⇒I(A): {Nb}Kab**
6. **I(A)⇒B: {dec(Nb)}Kab**



Popis útoku:

Útok na rozdíl od předchozí verze nelze provést, protože pro ověření uživatele je vyžadována znalost klíče Kab. Nestačí tudíž pouze přeposlat 4. zprávu, ale je nutná znalost klíče Kab pro zašifrování původní hodnoty nonce.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 3

Doba verifikace: 1 ms

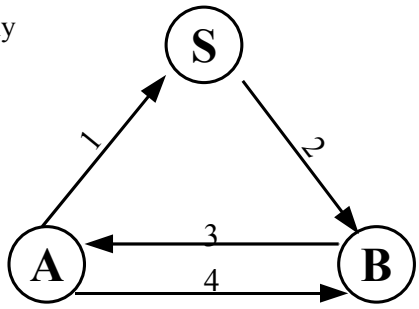
5.6 Kao Chow v.1

Autoři: I Long Kao, Randy Chow

Publikováno: 1995, [18]

Popis: Jednocestná autentizace s použitím důvěryhodné třetí strany

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

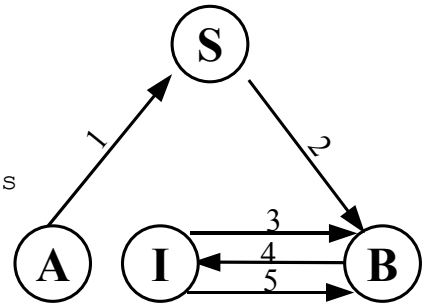


Útok na protokol:

Autoři: John Clark, Jeremy Jacob

Publikováno: 1997, [15]

1. A⇒S: A,B,Na
2. S⇒B: {A,B,Na,Kab}Kas,{A,B,Na,Kab}Kbs
3. I(S)⇒B: {A,B,Na,Kab}Kas,{A,B,Na,Kab}Kbs
4. B⇒I(A): {A,B,Na,Kab}Kas,{Na}Kab,N'b
5. I(A)⇒B: {N'b}Kab



Popis útoku:

Útočník odposlechne komunikaci. Předpokládáme, že uhodne klíč Kab. Poté se útočník vydává za server a zašle stejnou zprávu, jako odposlechl v bodě 2. Tím jsou oba dohodnuti na klíči a v posledním kroku, jelikož útočník zná klíč se může autentizovat.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.2. S -> B : {A, B, Na, Kab}Kas, {A, B, Na, Kab}Kbs
b.2. I(S) -> B : {A, B, Na, Kab}Kas, {A, B, Na, Kab}Kbs
b.3. B -> I(A) : {A, B, Na, Kab}Kas, {Na}Kab, N'b
b.4. I(A) -> B : {N'b}Kab
```

Počet stavů: 213

Doba verifikace: 1437 ms

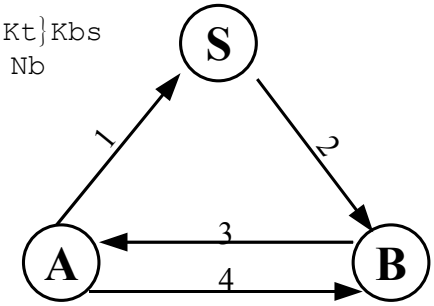
5.7 Kao Chow v.2

Autoři: I Long Kao, Randy Chow

Publikováno: 1995, [18]

Popis: Jednocestná autentizace s použitím důvěryhodné třetí strany

```
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
```

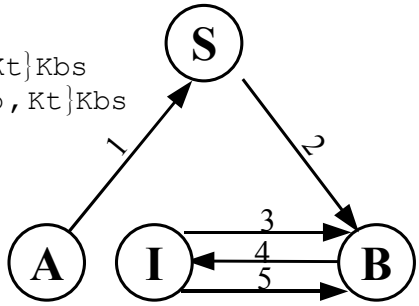


Útok na protokol:

Autoři: I Long Kao, Randy Chow

Publikováno: 1995, [18]

```
1. A⇒S: A,B,Na
2. S⇒B: {A,B,Na,Kab,Kt}Kas,{A,B,Na,Kab,Kt}Kbs
3. I(S)⇒B: {A,B,Na,Kab,Kt}Kas,{A,B,Na,Kab,Kt}Kbs
4. B⇒I(A): {A,B,Na,Kab,Kt}Kas,{Na}Kt,N'b
5. I(A)⇒B: {N'b,Kab}Kt
```



Popis útoku:

Zavedení nového klíče pouze pro autentizaci, odstraňuje problém z předchozí verze. Je však nutné dodržet, že klíč bude použit pouze k tomuto účelu. Jelikož se klíč Kt používá pouze jednou, na rozdíl od klíče relace Kab, kterým se šifrují data, je daleko obtížnější zjistit hodnotu Kt. Útok je velmi ztížen, ne však nemožný, proto se může stát, že při špatném zacházení s klíči se podaří útočníkovi zjistit hodnotu Kt, pomocí něhož následně může provést autentizaci.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.2. S -> B : {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs
b.2. I(S) -> B : {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs
b.3. B -> I(A) : {A, B, Na, Kab, Kt}Kas, {Na}Kab, N’b
b.4. I(A) -> B : {N’b, Kt}Kab
```

Počet stavů: 211

Doba verifikace: 1391 ms

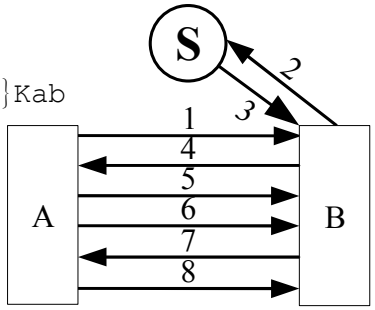
5.8 KSL

Autoři: Axel Kehne, Jürgen Schönwalder, Horst Langendörfer

Publikováno: 1992, [22]

Popis: Upravená verze Kerbera s použitím nonce

1. A⇒B: Na,A
2. B⇒S: Na,A,Nb,B
3. S⇒B: {Nb,A,Kab}Kbs,{Na,B,Kab}Kas
4. B⇒A: {Na,B,Kab}Kas,{Tb,A,Kab}Kbb,N'b,{Na}Kab
5. A⇒B: {N'b}Kab
6. A⇒B: Ma,{Tb,A,Kab}Kbb
7. B⇒A: Mb,{Ma}Kab
8. A⇒B: {Mb}Kab

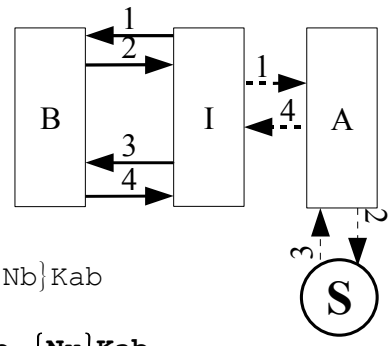


Útok na protokol:

Autoři: Gavin Lowe

Publikováno: 1996, [27]

1. I(A)⇒B: Nx,A
2. B⇒I(S): Nx,A,Nb,B
1. I(B)⇒A: Nb,B
2. A⇒S: Nb,B,Na,A
3. S⇒A: {Na,B,Kab}Kas,{Nb,A,Kab}Kbs
4. A⇒I(B): {Nb,A,Kab}Kbs,{Ta,B,Kab}Kaa,N'b,{Nb}Kab
3. I(S)⇒B: {Nb,A,Kab}Kbs,{Na,B,Kab}Kas
4. B⇒I(A): {Na,B,Kab}Kas,{Tb,A,Kab}Kbb,N''b,{Nx}Kab



Popis útoku:

Je možné provést stejný útok, jako v případě Neumann-Stubblebine, na autentizaci. Pro příklad byl vybrán jiný útok, který využívá souběžného spuštění více relací. Útočník zahájí první relaci po obdržení hodnoty Nb, začíná komunikovat s druhým účastníkem. Pomocí této hodnoty získá požadovaný výraz {Nb, A, Kab}Kbs. V posledním kroku získá hodnotu lístku {Tb, A, Kab}Kbb.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.1. I -> B : Mi, { Tb, A, Kab}Kbb
a.2. B -> I : Mb, {Mi}Kab
b.1. I -> B : Mb, {Tb, A, Kab}Kbb
b.2. B -> I : Mb', {Mb}Kab
a.3. I -> B : {Mb}Kab
```

Počet stavů: 26

Doba verifikace: 16 ms

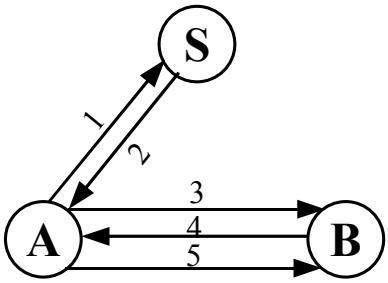
5.9 Needham Schroeder (symetrický klíč)

Autoři: Roger Needham, Michael Schroeder

Publikováno: 1978, [14]

Popis: Distribuce klíče a vzájemná autentizace s použitím důvěryhodné třetí strany

1. A⇒S: A,B,Na
2. S⇒A: {Na,B,Kab,{Kab,A}Kbs}Kas
3. A⇒B: {Kab,A}Kbs
4. B⇒A: {Nb}Kab
5. A⇒B: {dec(Nb)}Kab

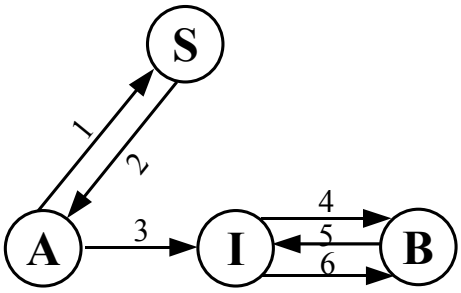


Útok na protokol:

Autoři: Dorothy Elizabeth Denning, Giovanni Maria Sacco

Publikováno: 1981, [15]

1. A⇒S: A,B,Na
2. S⇒A: {Na,B,Kab,{Kab,A}Kbs}Kas
3. A⇒I(B): {Kab,A}Kbs
4. I(A)⇒B: {Kab,A}Kbs
5. B⇒I(A): {Nb}Kab
6. I(A)⇒B: {dec(Nb)}Kab



Popis útoku:

Útočník odposlouchává komunikaci mezi účastníky A a B, důležitá je pro něj třetí zpráva, kterou si zapamatuje a použije při útoku. Útok začíná předstíráním že se jedná o Alici, pošle Bobovi stejné hodnoty jako poslal legitimní účastník v kroku 3. Ten se domnívá, že se jedná o novou relaci. Mezi těmito událostmi však mohl útočník klíč odhalit a poté již pro něj není problém přijmout zašifrovaný nonce Nb a vrátit zašifrovanou hodnotu upraveného nonce s zkompromitovaným klíčem.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 0

Doba verifikace: 0 ms

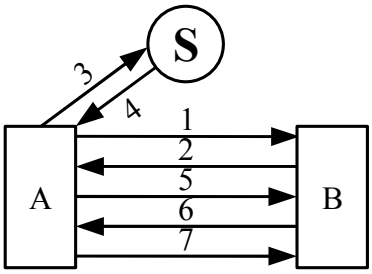
5.10 Needham Schroeder (symetrický klíč) amend.

Autoři: Roger Needham, Michael Schroeder

Publikováno: 1987, [29]

Popis: Distribuce klíče a vzájemná autentizace s použitím důvěryhodné třetí strany

```
1. A⇒B: A
2. B⇒A: {A,Nb}Kbs
3. A⇒S: A,B,Na,{A,Nb}Kbs
4. S⇒A: {Na,B,Kab,{Kab,Nb,A}Kbs}Kas
5. A⇒B: {Kab,Nb,A}Kbs
6. B⇒A: {Nb}Kab
7. A⇒B: {dec(Nb)}Kab
```

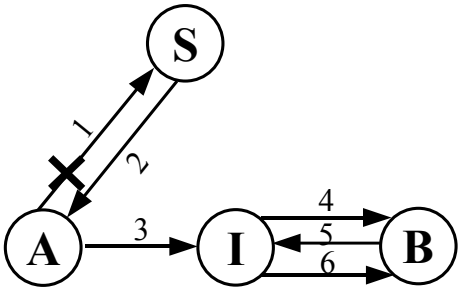


Útok na protokol: Nenalezen (nepublikován)

Autoři: Dorothy Elizabeth Denning, Giovanni Maria Sacco

Publikováno: 1981, [15]

```
1. A⇒S: A,B,Na
2. S⇒A: {Na,B,Kab,{Kab,A}Kbs}Kas
3. A⇒I(B): {Kab,A}Kbs
4. I(A)⇒B: {Kab,A}Kbs
5. B⇒I(A): {Nb}Kab
6. I(A)⇒B: {dec(Nb)}Kab
```



Popis útoku:

Tento protokol opravuje předešlou chybu přidáním úvodních dvou zpráv. Kdy útočník není schopen provést stejný útok. Ještě před možným útokem se provede nonce handshake, který zabrání zneužití následující části protokolu.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 0

Doba verifikace: 0 ms

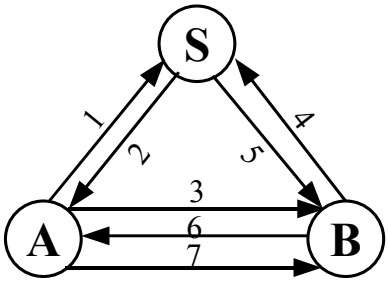
5.11 Needham Schroeder (veřejný klíč)

Autoři: Roger Needham, Michael Schroeder

Publikováno: 1978, [14]

Popis: Distribuce klíče a vzájemná autentizace s použitím důvěryhodné třetí strany a asymetrické kryptografie.

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

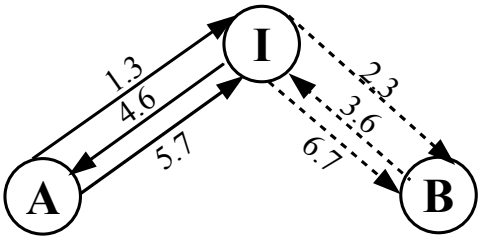


Útok na protokol:

Autoři: Gavin Lowe

Publikováno: 1995, [17]

1.3 A⇒I(B): {Na,A}Kpi
2.3 I(A)⇒B: {Na,A}Kpb
3.6 B⇒I(A): {Na,Nb}Kpa
4.6 I(B)⇒A: {Na,Nb}Kpa
5.7 A⇒I(B): {Nb}Kpi
6.7 I(A)⇒B: {Nb}Kpb



Popis útoku:

Útok opět využívá možnosti použít legitimního účastníka ke generování hodnot potřebných k další komunikaci. Komunikace je spuštěna ve dvou nezávislých relacích, kdy druhou vyvolává útočník. Přeposláním zprávy {Na,Nb}Kpa, obdrží útočník hodnotu Nb, kterou poté již může sám zašifrovat pomocí veřejného klíče Boba.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.3. A -> I : {Na,A}Kpi
b.3. I(A) -> B : {Na,A}Kpb
b.6. B -> I(A) : {Na,Nb}Kpa
a.6. I -> A : {Na,Nb}Kpa
a.7. A -> I : {Nb}Kpi
b.7. I(A) -> B : {Nb}Kpb
```

Počet stavů: 35

Doba verifikace: 64 ms

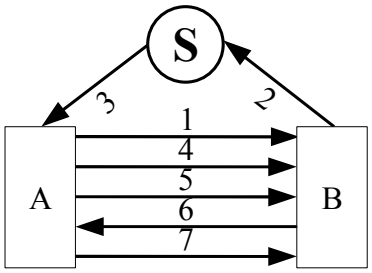
5.12 Neumann Stubblebine

Autoři: Clifford B. Neumann, Stuart G. Stubblebine

Publikováno: 1993, [25]

Popis: Distribuce klíče a vzájemná autentizace s použitím důvěryhodné třetí strany. Kroky 5, 6, 7 provádějí vzájemnou autentizaci, můžeme ji použít několikrát, dokud nedojde platnosti Tb.

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: Ma,{A,Kab,Tb}Kbs
6. B⇒A: Mb,{Ma}Kab
7. A⇒B: {Mb}Kab

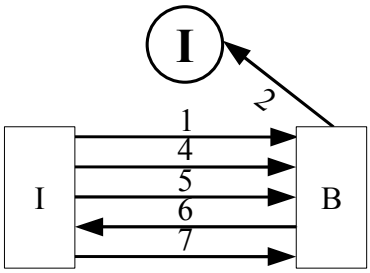


Útok na protokol:

Autoři: Tzonelih Hwang, Narn-Yoh Lee, Chuang-Ming Li, Ming-Yung Ko**,** Yung-Hsiang Chen

Publikováno: 1995, [24]

1. I(A)⇒B: A,Na
2. B⇒I(S): B,{A,Na,Tb}Kbs,Nb
4. I(A)⇒B: {A,Kab,Tb}Kbs,{Nb}Kab
5. I(A)⇒B: Ma,{A,Kab,Tb}Kbs
6. B⇒I(A): Mb,{Ma}Kab
7. I(A)⇒B: {Mb}Kab



Popis útoku:

Na tento protokol bylo publikováno mnoho útoků. Útočník se vydává jak za server tak za Alici. Útočník v 3. kroku použije jako zprávu {A, Kab, Tb}Kbs, zprávu kterou obdržel v kroku 2 {A, Na, Tb}Kbs. Z toho plyne, že klíč Kab bude mít hodnotu Na.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.1. I -> B : Ma, {A, Kab, Tb}Kbb
a.2. B -> I : Mb, {Ma}Kab
b.1. I -> B : Mb, {A, Kab, Tb}Kbb
b.2. B -> I : Mb', {Mb}Kab
a.3. I -> B : {Mb}Kab
```

Počet stavů: 18

Doba verifikace: 16 ms

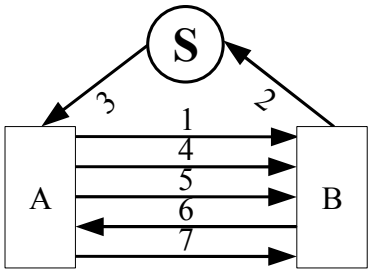
5.13 Neumann Stubblebine – Hwang

Autoři: Tzonelih Hwang, Narn-Yoh Lee, Chuang-Ming Li, Ming-Yung Ko**,** Yung-Hsiang Chen

Publikováno: 1995, [24]

Popis: Tato úprava protokol opravuje chybu popsánu v předešlém protokolu.

1. A⇒B: A,Na
2. B⇒S: B,{A,Na,Tb,Nb}Kbs
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: Ma,{A,Kab,Tb}Kbs
6. B⇒A: Mb,{Ma}Kab
7. A⇒B: {Mb}Kab

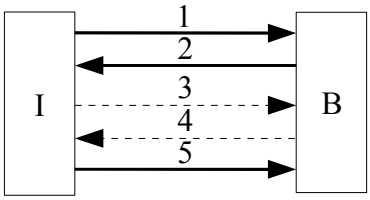


Útok na protokol:

Autoři: Tzonelih Hwang, Narn-Yoh Lee, Chuang-Ming Li, Ming-Yung Ko**,** Yung-Hsiang Chen

Publikováno: 1995, [24]

1. I(A)⇒B: Ma,{A,Kab,Tb}Kbs
2. B⇒I(A): Mb,{Ma}Kab
3. I(A)⇒B: Mb,{A,Kab,Tb}Kbs
4. B⇒I(A): Mb',{Mb}Kab
5. I(A)⇒B: {Mb}Kab



Popis útoku:

Jelikož kroky 5, 6 a 7 zůstávají stejné jako u nemodifikované verze zůstává z předešlé verze i jedna z jejich slabin. Jedná se o chybu autentizace, kdy při vyžadání zašifrování zprávy Mb, útočník podstrčí tuto hodnotu legitimnímu účastníkovi a ten tuto zprávu zašifruje. Útočník poté obdrženou zprávu použije v posledním kroku.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.1. I -> B : Ma, {A, Kab, Tb}Kbb
a.2. B -> I : Mb, {Ma}Kab
b.1. I -> B : Mb, {A, Kab, Tb}Kbb
b.2. B -> I : Mb', {Mb}Kab
a.3. I -> B : {Mb}Kab
```

Počet stavů: 26

Doba verifikace: 31 ms

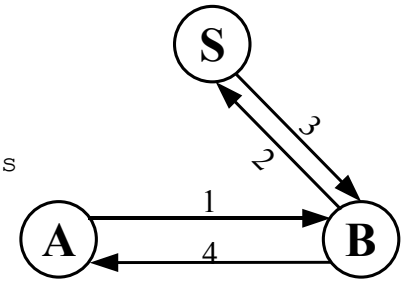
5.14 Otway Rees

Autoři: Dave Otway, Owen Rees

Publikováno: 1997, [30]

Popis: Distribuce klíče s pomocí TTP a symetrickými klíči

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

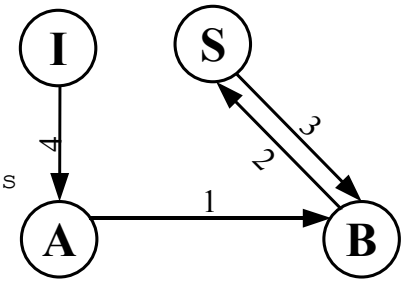


Útok na protokol:

Autoři: Gavin Lowe

Publikováno: 1997, [7]

1. A⇒I(B): M,A,B,{Na,M,A,B}Kas
2. B⇒S: M,A,B,{Na,M,A,B}Kas,{Nb,M,A,B}Kbs
3. S⇒B: M,{Na,Kab}Kas,{Nb,Kab}Kbs
4. I(B)⇒A: M,{Na,M,A,B}Kas



Popis útoku:

Útočník první zprávu odposlechne a poslední zprávu pošle sám. Útok je založen na faktu, že útočník místo zprávy {Na, Kab}Kas, pošle zprávu {Na,M,A,B}Kas. Alice zprávu dešifruje zkontroluje Na a hodnotu M, A, B pokládá za klíč Kab. Hodnoty M, A, B jsou známy tudíž útočník zná i klíč Kab.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 17

Doba verifikace: 156 ms

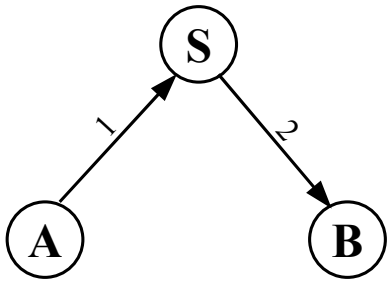
5.15 Wide Mouthed Frog

Autoři: Michael Burrows, Martin Abadi, Roger Needham

Publikováno: 1989, [20]

Popis: Distribuce sdíleného klíče

1. A⇒S: A,{Ta,B,Kab}Kas
2. S⇒B: {Ts,A,Kab}Kbs



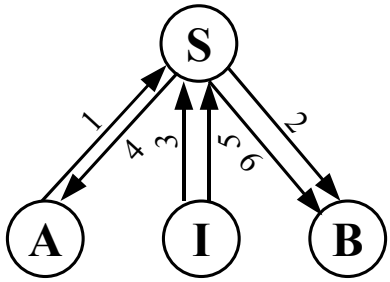
Útok na protokol:

Autoři: Gavin Lowe

Publikováno: 1997, [7]

1. A⇒S: A,{Ta,B,Kab}Kas
2. S⇒B: {Ts,A,Kab}Kbs
3. I(B)⇒S: B,{Ts,A,Kab}Kbs
4. S⇒A: {T's,B,Kab}Kas
5. I(A)⇒S: A,{T's,B,Kab}Kas
6. S⇒B: {T''s,A,Kab}Kbs

…



Popis útoku:

Útočník může neustále prodlužovat platnost klíče, přeposíláním dříve zachysených zpráv. Po obdržení, zprávy útočník ihned přepošle zprávu zpět serveru, který opět vygeneruje nový časové razítko čímž prodlouží platnost klíče. Příklad popisuje 3 běhy protokolu, útočník musí odposlouchávat veškerou komunikaci ze serveru, aby mohl tyto zprávy znovu přeposílat.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
I(B) -> S : B, {Ts, A, Kab}Kbs
S -> A : {T’s, B, Kab}Kas
```

Počet stavů: 1

Doba verifikace: 16 ms

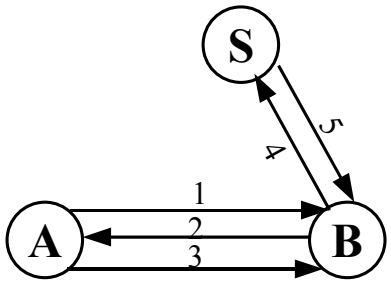
5.16 Woo Lam Π

Autoři: Simon S. Lam,Thomas Y. C. Woo

Publikováno: 1994, [16]

Popis: Jednocestná autentizace s použitím důvěryhodné třetí strany.

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

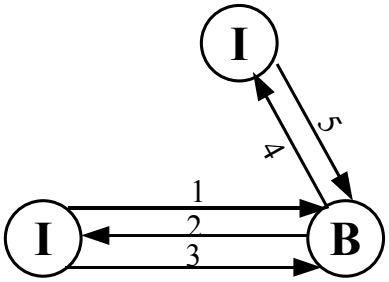


Útok na protokol:

Autoři: John Clark, Jeremy Jacob

Publikováno: 1997, [15]

```
1. I(A)⇒B: I
1. A⇒B: A
2. B⇒I(A): Nbi
2. B⇒A: Nb
3. I(A)⇒B: G
3. A⇒B: NbKas
4. B⇒S: {I,G}Kbs
4. B⇒S: {I,G}Kbs
5. S⇒B: x
5. S⇒B: {Nb}Kbs
```



Popis útoku:

Útočník se vydává za důvěryhodný server i účastníka A. Ve třetím kroku vrátí útočník uživateli stejnou hodnotu a pokud, tím docílí toho že hodnota Nb se bude rovnat {Nb}Kas, nemusí tudíž znát hodnotu klíče Kas ani Kbs, aby obelstil autentizaci.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
a.3. A -> I : {Na,A}KPi
b.3. I(A) -> B : {Na,A}KPb
b.6. B -> I(A) : {Na,Nb}KPa
a.6. I -> A : {Na,Nb}KPa
a.7. A -> I : {Nb}KPi
b.7. I(A) -> B : {Nb}KPb
```

Počet stavů: 35

Doba verifikace: 31 ms

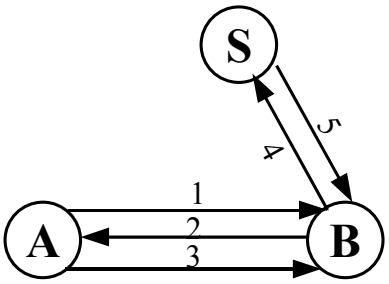
5.17 Woo Lam Π 1

Autoři: Simon S. Lam,Thomas Y. C. Woo

Publikováno: 1994, [16]

Popis: Jednocestná autentizace s použitím důvěryhodné třetí strany.

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

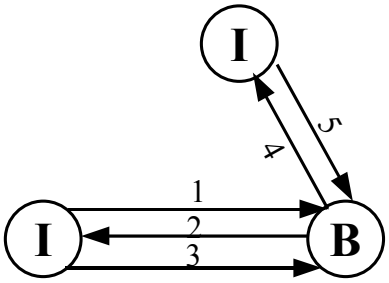


Útok na protokol:

Autoři: John Clark, Jeremy Jacob

Publikováno: 1997, [15]

```
1. I(A)⇒B: A
2. B⇒I(A): Nb
3. I(A)⇒B: Nb
4. B⇒I(S): {A,B,Nb}Kbs
5. I(S)⇒B: {A,B,Nb}Kbs
```



Popis útoku:

Útočník se vydává za důvěryhodný server i účastníka A. Ve třetím kroku vrátí útočník uživateli stejnou hodnotu a pokud, tím docílí toho že hodnota Nb se bude rovnat {A,B,Nb}Kas, nemusí tudíž znát hodnotu klíče Kas ani Kbs, aby obelstil autentizaci.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
1. I(A) -> B : A
2. B -> I(A) : Nb
3. I(A) -> B : Nb
4. B -> I(S) : {A, B, Nb}Kbs
5. I(S) -> B : {A, B, Nb}Kbs
```

Počet stavů: 3

Doba verifikace: 1 ms

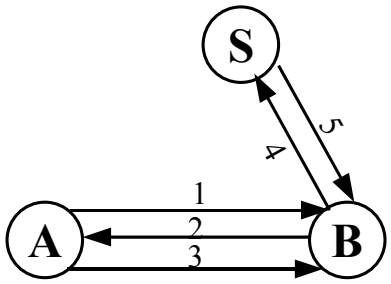
5.18 Woo Lam Π 2

Autoři: Simon S. Lam,Thomas Y. C. Woo

Publikováno: 1994, [16]

Popis: Jednocestná autentizace s použitím důvěryhodné třetí strany.

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

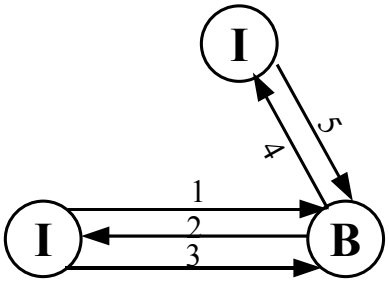


Útok na protokol:

Autoři: John Clark, Jeremy Jacob

Publikováno: 1997, [15]

```
1. I(A)⇒B: A
2. B⇒I(A): Nb
3. I(A)⇒B: Nb
4. B⇒I(S): {A,Nb}Kbs
5. I(S)⇒B: {A,Nb}Kbs
```



Popis útoku:

Útočník se vydává za důvěryhodný server i účastníka A. Ve třetím kroku vrátí útočník uživateli stejnou hodnotu a pokud, tím docílí toho že hodnota Nb se bude rovnat {A, Nb}Kas, nemusí tudíž znát hodnotu klíče Kas ani Kbs, aby obelstil autentizaci.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
1. I(A) -> B : A
2. B -> I(A) : Nb
3. I(A) -> B : Nb
4. B -> I(S) : {A, Nb}Kbs
5. I(S) -> B : {A, Nb}Kbs
```

Počet stavů: 3

Doba verifikace: 1 ms

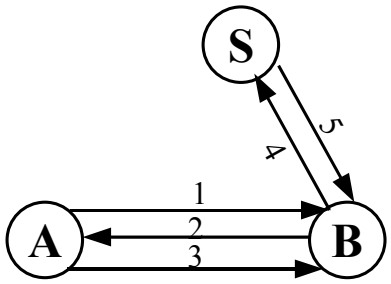
5.19 Woo Lam Π 3

Autoři: Simon S. Lam,Thomas Y. C. Woo

Publikováno: 1994, [16]

Popis: Jednocestná autentizace s použitím důvěryhodné třetí strany.

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. $S \Rightarrow B: \{A, Nb\}Kbs$

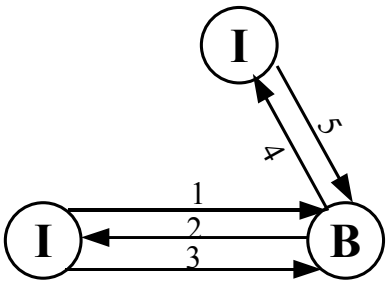


Útok na protokol:

Autoři: John Clark, Jeremy Jacob

Publikováno: 1997, [15]

1. I(A)⇒B: A
2. B⇒I(A): Nb
3. I(A)⇒B: Nb
4. B⇒I(S): {A, Nb}Kbs
5. I(S)⇒B: {A, Nb}Kbs



Popis útoku:

Útočník se vydává za důvěryhodný server i účastníka A. Ve třetím kroku vrátí útočník uživateli stejnou hodnotu a pokud, tím docílí toho že hodnota Nb se bude rovnat {Nb}Kas, nemusí tudíž znát hodnotu klíče Kas ani Kbs, aby obelstil autentizaci.

Výsledek verifikace Athena

Nalezena chyba: ✓

Chyba:

```
1. I(A) -> B : A
2. B -> I(A) : Nb
3. I(A) -> B : Nb
4. B -> I(S) : {A, Nb}Kbs
5. I(S) -> B : {A, Nb}Kbs
```

Počet stavů: 3

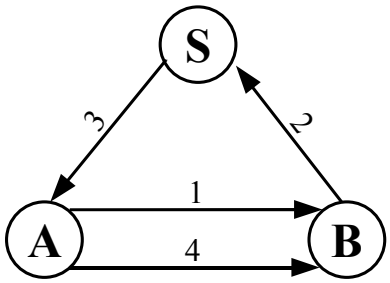
Doba verifikace: 16 ms

5.20 Yahalom

Autoři: Raphael Yahalom

Publikováno: 1989, [20]

Popis: Distribuce sdíleného klíče a vzájemná autentizace

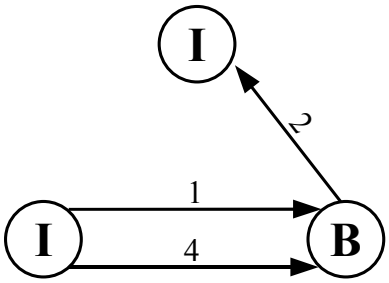


1. A⇒B: A,Na
2. B⇒S: B,{A,Na,Nb}Kbs
3. S⇒A: {B,Kab,Na,Nb}Kas,{A,Kab}Kbs
4. A⇒B: {A,Kab}Kbs,{Nb}Kab

Útok na protokol:

Autoři: John Clark, Jeremy Jacob

Publikováno: 1997, [7]



1. I(A)⇒B: A,Na
2. B⇒I(S): B,{A,Na,Nb}Kbs
4. I(A)⇒B: {A,Kab}Kbs,{Nb}Kab

Popis útoku:

Stejná chyba byla popsána již dříve u protokolu Otway Rees. Jelikož se útočník vydává jak za Alici, tak za TTP, neposílá se 3 zpráva. Útok spočívá ve skutečnosti, že útočník zná hodnotu {A, Na, Nb}Kbs a pokud má v dalším kroku poslat Bobovi zprávu {A, Kab}Kbs. Zašle místo ní zprávu {A, Na, Nb}Kbs, tudíž Bob bude pokládat za klíč hodnotu Na, Nb, kterou útočník zná.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 865

Doba verifikace: 2563 ms

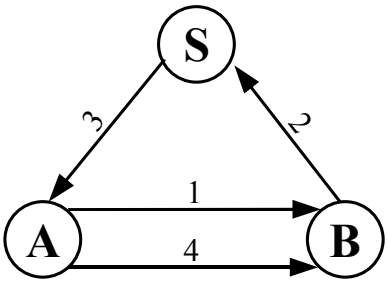
5.21 Yahalom - BAN

Autoři: Michael Burrows, Martin Abadi, Roger Needham

Publikováno: 1989, [20]

Popis: Zjednodušení předchozího protokolu, opět distribuce sdíleného klíče a vzájemná autentizace.

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

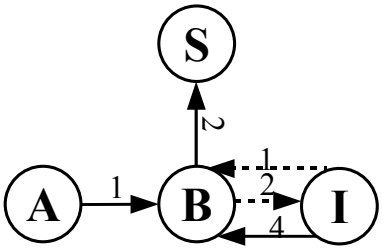


Útok na protokol:

Autoři: Paul Syverson

Publikováno: 1994, [31]

1. A⇒B: A,Na
2. B⇒S: B,Nb,{A,Na}Kbs
1. I(A)⇒B: A,Na,Nb
2. B⇒I(S): B,Nb,{A,Na,Nb}Kbs
4. I(A)⇒B: {A,Kab,Nb}Kbs,{Nb}Kab



Popis útoku:

Podobná chyba byla popsána již dříve u protokolu Otway Rees. Jelikož se útočník vydává jak za Alici, tak za TTP, neposílá se 3 zpráva. Po odposlechnutí prvních dvou zpráv zná útočník hodnoty A, B, Na, Nb. Toho využije k útoku zahájí znovu komunikaci s Bobem a předstírá, že je Alice. Místo A, Na pošle A, Na, Nb, čímž docílí, že Bob zašifruje klíčem Kbs všechny 3 hodnoty. Pokud tedy tuto zprávu útočník v posledním kroku přepošle, Bob nepozná chybu a klíč Kab bude mít hodnotu Na. Z těchto faktů tudíž plyne {A, Na, Nb}Kbs = {A, Kab, Nb}Kbs => Na = Kab.

Výsledek verifikace Athena

Nalezena chyba: ✗

Chyba:

```
Chyba nenalezena
```

Počet stavů: 3628

Doba verifikace: 12578 ms

6 Zhodnocení výsledků

6.1 Porovnání protokolů

Název protokolu	*TTP*	*K?C*	*Model*	*Chyba*	*Nalezena*	*Pub.*	*Útok*
Andrew Secure RPC	✗	✗	✗	✓	✓	1989	1989
Andrew Secure RPC - BAN	✗	✗	✗	✗	✗	1989	-
Andrew Secure RPC – BAN konkrétní	✗	✗	✗	✓	✓	1990	1996
Denning-Sacco	✓	KDC	push	✓	✓	1981	1997
Denning-Sacco - Lowe modifikace	✓	KDC	push	✗	✗	1997	-
Kao Chow v.1	✓	KDC	push	✓	✓	1995	1997
Kao Chow v.2	✓	KDC	push	✓	✓	1995	1995
KSL	✓	KDC	pull	✓	✓	1992	1996
Needham Schroeder (symetrický klíč)	✓	KDC	push	✓	✗	1978	1981
Needham Schroeder (symetrický klíč) a.	✓	KDC	push	✗	✗	1978	-
Needham Schroeder (veřejný klíč)	✓	KDC	hybrid	✓	✓	1978	1995
Neumann Stubblebine	✓	KDC	hybrid	✓	✓	1993	1995
Neumann Stubblebine – Hwang	✓	KDC	hybrid	✓	✓	1995	1995
Otway Rees	✓	KDC	pull	✓	✗	1997	1997
Wide Mouthed Frog	✓	KTC	hybrid	✓	✓	1989	1997
Woo Lam Π	✓	KDC	pull	✓	✓	1994	1997
Woo Lam Π 1	✓	KDC	pull	✓	✓	1994	1997
Woo Lam Π 2	✓	KDC	pull	✓	✓	1994	1997
Woo Lam Π 3	✓	KDC	pull	✓	✓	1994	1997
Yahalom	✓	KDC	hybrid	✓	✗	1989	1997
Yahalom - BAN	✓	KDC	hybrid	✓	✗	1989	1994

TTP : použití důvěryhodné třetí strany (Third Trusted Party)

K?C : pokud se jedná o TTP, zda TTP generuje klíč (KDC, KTC)

Model: pro TTP (push, pull, hybrid)

Chyba: výskyt chyby v protokolu

Nalezena: zda byla chyba nalezena verifikací v nástroji Athena

Pub.: rok ve kterém byl protokol prvně publikován

Útok: rok ve kterém byl poprvé publikován útok na protokol

6.2 Popis modifikací protokolů z hlediska útoku

Andrew Secure RPC

Andrew Secure RPC vznikl roku 1987 a byl dvakrát modifikován. První úpravu protokolu Andrew Secure RPC provedli pánové Burrows, Abadi a Needham. Podle nich byla také modifikace označena přívlastkem BAN. Toto vylepšení se týká i jiných protokolů, v této práci je zmíněn protokol Yahalom. Existuje i tzv. konkrétní varianta BAN modifikace, která na rozdíl od původní modifikace bezpečnost nezvyšuje, ovšem je vytvářena s důrazem na efektivní implementaci. Zprávy jsou zkráceny a šifruje se co nejméně nutné množství dat. Tuto konkrétní variantu BAN opravila modifikace Lowe, která zachovává stručnost a přidává pouze jediný bezpečnostní prvek.

První modifikace BAN ošetřuje chybu, kdy problém spočívá ve vygenerování celé nové zprávy v kroku 4. Tato zpráva neobsahuje žádnou informaci, která by byla spjata s danou relací a tudíž může dojít k útoku s použitím zprávy z jiné relace. Lowe modifikace opravuje chybu způsobenou, tentokrát ve zprávě 2, kde bez určení identity odesilatele je možné, donutit komunikující protistranu k vygenerování této zprávy.

```
1. A -> B : A, {Na}Kab
2. B -> A : {succNa, Nb}Kab
3. A -> B : {succNb}Kab
4. B -> A : {K'ab, N'b}Kab
```

Andrew Secure RPC

```
1. A -> B : A, {Na}Kab
2. B -> A : {succNa, Nb}Kab
3. A -> B : {succNb}Kab
4. B -> A : {K'ab, N'b, Na}Kab
```

Andrew Secure RPC BAN

```
1. A -> B : A, Na
2. B -> A : {Na, K'ab}Kab
3. A -> B : {Na}K'ab
4. B -> A : Nb
```

Andrew Secure RPC konkrétní BAN

```
1. A -> B : A, Na
2. B -> A : {Na, K'ab, B}Kab
3. A -> B : {Na}K'ab
4. B -> A : Nb
```

Andrew Secure RPC konkrétní BAN modif. Lowe

Woo Lam

Tato skupina protokolů od autorů Woo a Lam byla publikována v jediném článku roku 1994. Návrh obsahuje mnoho verzí jednak pro jednocestnou, tak pro vzájemnou autentizaci. Práce se zabývá pouze protokoly s jednocestnou autentizací s použitím symetrické kryptografie. Jednotlivé verze ovšem z pohledu vylepšení bezpečnosti přílišný vliv nemají.

Pro ilustraci porovnání verzí Woo Lam protokolu jsou zde uvedeny dvě varianty, verze 1 a 3. Rozdíl mezi verzemi 1 a 3 spočívá v redukci nadbytečných hodnot. Vyznačené hodnoty A a B identifikující účastníky komunikace, nikterak nepřispívají k bezpečnosti a tudíž je možné tyto hodnoty z protokolu odstranit. V tomto případě představují redundantní informaci, která je

redukována. Obě verze trpí stejnou slabinou, při nahrazení zprávy 3 hodnotou Nb získáme dosazením do kroku 4 hodnotu 5.

```
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
```

Woo Lam Pi3

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

Woo Lam Pi 1

6.3 Porovnání výsledků s jinými nástroji

Název protokolu	*Casper [33]*	*C & J [7]*	*AAPA [34]*	*Athena*
Andrew Secure RPC	✓	✓	✓	✓
Andrew Secure RPC - BAN	-	-	-	✗
Andrew Secure RPC – BAN konkrétní	-	-	-	✓
Denning-Sacco	✗	✗	✗	✓
Denning-Sacco - Lowe modifikace	-	-	-	✗
Kao Chow v.1	✗	✓	✓	✓
Kao Chow v.2	✗	✗	✓	✓
KSL	-	-	-	✓
Needham Schroeder (symetrický klíč)	✗	✓	✓	✗
Needham Schroeder (symetrický klíč) amend.	✗	✗	✗	✗
Needham Schroeder (veřejný klíč)	✓	✓	✗	✓
Neumann Stubblebine	✓	✓	✗	✓
Neumann Stubblebine – Hwang	-	-	-	✓
Otway Rees	✓	✓	✗	✗
Wide Mouthed Frog	✓	✓	✗	✓
Woo Lam Π	✓	✓	✗	✓
Woo Lam Π 1	✓	✓	✗	✓
Woo Lam Π 2	✓	✓	✗	✓
Woo Lam Π 3	✓	✓	✗	✓
Yahalom	✗	✓	✓	✗
Yahalom - BAN	-	-	-	✗

- \- informace nepublikována
- ✓ nalezena chyba
- ✗ nenalezena chyba

6.4 Typy útoků

Tato podkapitola popisuje několik útoků[7] na autentizační protokoly a jejich výskyt ve zvolených protokolech.

Útoky na čerstvost zprávy (Freshness Attacks)

Tento útok využívá zprávy zaslané v předchozí relaci, která obsahuje nový klíč a žádnou identifikaci relace. Útočník může za určitou dobu klíč, pomocí něhož se dříve komunikovalo, zjistit a následně, může zaslat zachycenou zprávu. Příjemce zprávy si bude myslet, že účastník chce komunikovat pomocí nového klíče, ten však bude již kompromitován a dojde k nesprávnému určení klíče a autentizaci. Útok na protokolu NS odhalili Denning a Sacco a navrhli změnu v protokolu.

Příklad chybných protokolů: Needham Schroeder (symetrický klíč)
Odhaleno Athena: ✗

Typové chyby (Type Flaws)

Mezi další možnou zranitelnost protokolu patří tzv. typové chyby. Jedná se o možnost chybné interpretace přijatých dat. Může se stát, že například útočník místo zprávy {A, K'ab}Kab podstrčí dříve získanou zprávu {A, Na}Kab, příjemce pokládá Na za nový klíč a domnívá se, že jej útočník nemůže znát. Tudíž k zjištění nového klíče nepotřebuje útočník znát hodnotu klíče Kab.

Příklad chybných protokolů: Andrew Secure RPC
Odhaleno Athena: ✓

Paralelní útoky (Parallel Session Attacks)

Útok spočívá v spuštění více relací souběžně, kdy zprávy z jedné relace jsou použity jako vstupy k druhé relaci. Příjemce bývá označován také jako orákulum (oracle), kdy mu můžeme podstrčit zprávu a on nám ji zašifruje.

Příklad chybných protokolů: Wide Mouthed Frog
Odhaleno Athena: ✓

7 Závěr

Tato diplomová práce měla za úkol získat potřebné informace z oblasti verifikace bezpečnostních protokolů, prozkoumat nejrůznější nástroje pro verifikaci a v jednom vybraném nástroji provést implementaci protokolů a zhodnotit výsledky s dříve publikovanými.

Prvním krokem bylo zvolit protokoly, které se později v rámci diplomové práce implementovaly, a provést jejich důkladnou analýzu nastudováním jejich vlastností a slabin. Na začátku práce jsou uvedeny typy protokolů, aby je bylo možné v závěrečném hodnocení rozdělit do kategorií. Protokoly jsem vybíral podle množství informací, které jsem o jednotlivých protokolech získal a dále jsem se soustředil na protokoly, u kterých byly uvedeny jejich slabiny, abych je mohl otestovat. Dále jsem vybral 4 programy pro verifikaci bezpečnostních protokolů, mezi kterými jsem se rozhodoval, ve kterém budu práci implementovat. Výběr těchto čtyř nástrojů jsem učinil na základě prostudování velkého množství nástrojů, které jsou obsahem [1]. Nejdříve jsem vybral nástroje, které nejsou komerční a je možné je využít pro studijní účely. Dále jsem již vybíral na základě popsaných vlastností. Zúžil jsem výběr na čtyři a tyto nástroje jsem čtenáři blíže popsal a vzájemně porovnal.

Implementaci jsem provedl pomocí nástroje Athena, který má potřebné schopnosti ke splnění požadovaných cílů. Jedním z důvodů mé volby bylo i to, že jazyk není tak známý a neexistuje k němu tolik materiálů jako k ostatním, což byla na jednu stranu výzva, na straně druhé ovšem také riziko. Největším úskalím, zapříčiněném nedostatečnou dokumentací programu, bylo správně interpretovat výstup z Atheny, která poskytuje mnoho podpůrných informací, které je nutné odfiltrovat. Pokud jsem vybral protokol k verifikaci, snažil jsem se i verifikovat jeho modifikace a tím ukázat, jakým způsobem se chyby v protokolech postupně eliminovaly, případně, které nové chyby vznikaly.

U vybraných protokolů jsem popsal komunikaci mezi účastníky a uvedl chyby, které se u protokolů vyskytují. Tyto chyby jsem se snažil posléze naimplementovat v nástroji Athena a připsat k jednotlivým protokolům získané výsledky. Na závěr porovnávám výsledky vlastní verifikace s publikovanými útoky.

I když se nepodařilo objevit všechny chyby publikované v protokolech, řekl bych, že si nástroj počínal při práci obstojně a i přes svou malou uživatelskou základnu může být přínosem při verifikaci bezpečnostních protokolů.

Literatura

[1] Ptáček M. *Specifikační jazyky a nástroje pro analýzu a verifikaci bezpečnostních protokolů*, diplomová práce, Brno, FIT VUT v Brně, 2007.

[2] Hanáček P., Staudek J. *Bezpečnost informačních systémů*, ÚSIS, Praha, 2000, ISBN 80-238-5400-3.

[3] Dojen R., Coffey T. "*Layered Proving Trees: A Novel Approach to the Automation of Logic-Based Security Protocol Verification*", ACM Transactions on Information and System Security (TISSEC), 2005.

[4] Song X. D. *Description on How to Use the System* [online]. 2002 [cit. 2007-12-11]. Dostupné z WWW: <http://www.ece.cmu.edu/~dawnsong/athena/manual.pdf> .

[5] Berezin S., Groce A. *SyMP: The User's Guide* [online]. 2000 [cit. 2007-12-13]. Dostupné z WWW: <http://citeseer.ist.psu.edu/466087.html>.

[6] *CCSDS Key Management Techniques White Book*, 2005.

[7] Clark A. J., Jacob L. J. *A survey of authentication protocol literature* [online]. 1997 [cit. 2007-12-19]. Dostupné z WWW: <http://citeseer.ist.psu.edu/clark97survey.html>.

[8] Smrčka A. *Úvod do formální verifikace* [online]. 2007 [cit. 2007-12-20]. Dostupné z WWW: <http://www.fit.vutbr.cz/~smrcka/fav/guide/>.

[9] Očenášek P. *Verifikace bezpečnostních protokolů*, diplomová práce, Brno, CZ, FIT VUT, 2003.

[10] Lowe G., Broadfoot P., Hui M.L. *Casper A Compiler manual* [online]. 2001 [cit. 2007-12-20]. Dostupné z WWW: <http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/>.

[11] Mitchell C.J, Mitchell M., Stern U. *Automated Analysis of Cryptographic Protocol using Murphi* [online]. 1997 [cit. 2007-12-26]. Dostupné z WWW: <http://verify.stanford.edu/dill/PAPERS/verification/MMS97.ps>.

[12] Song X. D. *Athena: A New Efficient Automatic Checker for Security Protocol Analysis*, 12th IEEE Computer Security Foundations Workshop, Mordano, Italy, 1999.

[13] Song X. D. An Automatic Approach for Building Secure Systems [online]. 2002 [cit. 2007-12-13].Dostupné z WWW:<http://www.ece.cmu.edu/~dawnsong/athena/thesis.pdf >.

[14] Needham R., Schroeder M. *Using encryption for authenticationin large networks of computers*. Communications of the ACM, 1978, s. 993 – 999.

[15] Denning D., Sacco G. *Timestamps in key distributed protocols.* Communication of the ACM, 1981, s. 533 – 535.

[16] Woo T. Y. C., Lam S. S. *A Lesson on Authentication Protocol Design.* Operating Systems Review, 1994, s. 24 – 37.

[17] Lowe G. *An attack on the Needham-Schroeder public key authentication protocol.*

Information Processing Letters. 1995, s. 131 – 136.

[18] Kao I L., Chow R. *An efficient and secure authentication protocol using uncertified keys.* Operating Systems Review. 1995, s. 14 – 21.

[19] Satyanarayanan M. *Integrating Security in a Large Distributed system*, ACM Transactions on Computer Systems, 1989.

[20] Burrows M., Abadi M., Needham R. *A logic of authentication*. Technical Report 39, Digital Systems Research Center, 1989.

[21] Denning D., Sacco G. *Timestamps in key distributed protocols*. Communication of the ACM, 1981, s. 533 – 535.

[22] Lowe G. *A family of attacks upon authentication protocols*. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[23] *Security protocols library* [online]. 2003 [cit. 2008-03-11]. Dostupné z WWW: <http://www.lsv.ens-cachan.fr/spore/complete.pdf>.

[24] Hwang T., Lee N., Li Ch., Ko M., Chen Y. *Two attacks on Neumann-Stubblebine authentication protocols*. Information Processing Letters, 1995.

[25] Neumann C., Stubblebine S. *A note on the use of timestamps as nonces.* Operating Systems Review, 1993.

[26] Burrows M., Abadi M., Needham R. *A logic of authentication.* ACM Transactions on Computer Systems, 1990.

[27] Lowe G. *Some new attacks upon security protocols.* In IEEE Computer Society Press, In Proceedings of the Computer Security Foundations Workshop VIII, 1996.

[28] Kehne A., Schönwalder J., Langendörfer H. *Multiple authentications with a nonce-based protocol using generalized timestamps*, 1992.

[29] Needham R., Schroeder M. *Authentication revisited*. Operating Systems Review, 1987.

[30] Otway D., Rees O. *Efficient and timely mutual authentication.* Operating Systems Review, 1987.

[31] Syverson P. *A taxonomy of replay attacks.* In Proceedings of the 7th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press. 1994, s. 131 – 136.

[32] Thayer J., Herzog J. C., Guttman J. D. *Strand spaces: Proving security protocols correct.* Journal of Computer Security, 1999.

[33] Donovan B., Norris P., Lowe G. *Analyzing a library of security protocols using Casper and FDR*. In Workshop on Formal Methods and Security Protocols, 1999.

[34] Brackin S. H. *Evaluating and improving protocol analysis by automatic proof.* In Proceedings of the 11th IEEE Computer Security Foundations Workshop, 1998.

Seznam použitých zkratek a symbolů

AAPA – (Automatic Authentication Protocol Analyzer) nástroj pro verifikaci

ACG – (Automatic Code Generator) součást Atheny generující spustitelný kód ze specifikace

Alice – zažitý výraz označující iniciátora komunikace, označován písmenem A

APG – (Automatic Protocol Generator) součást Atheny generující nové protokoly ze specifikace

APV – (Automatic Protocol Verifier) součást Atheny verifikující protokoly

AS – (Andrew Secure) bezpečnostní protokol

BAN – zkratka jmen autorů modifikace Burrows M., Abadi M. a Needham R.

Bob – zažitý výraz označující příjemce komunikace, označován písmenem B

C & J – (Clark and Jacobson) autoři [7]

CSP – (Communicating Sequential Processes) algebra používaná při dokazování

FDR – (Failures Divergence Refinement) verifikační nástroj využívající konečných automatů

HOL – (higher-order logic) představuje nástroje pro formální verifikační dotazování

Init – (Initiator) odesilatel zprávy

KDC – (Key Distribution Center) TTP generující klíč

KTC – (Key Translation Center) TTP negenerující klíč

NS – (Needham Schroeder) bezpečnostní protokol

Nonce – (Nonce) náhodná hodnota vygenerována účastníky sloužící ke kontrole čerstvosti

Resp – (Responder) příjemce zprávy

RPC – (Remote Procedure Call) protokol, který může spouštět služby přes síť

SML – (Standard ML) funkcionální programovací jazyk

SSM – (Strand Space Model) model pro analýzu kryptografických protokolů

SRV – (Server) označení pro TTP při zápisu verifikačních pravidel

TTP – (Trusted Third Parties) server, kterému účastníci důvěřují

Seznam příloh

Příloha 1. Manuál nástroje Athena

Potřebné programy:

- Standard ML of New Jersey - http://www.smlnj.org/
  (verze 110.0.7. http://www.smlnj.org/NEWS/110-README.html)
- Athena - (APV) http://www.ece.cmu.edu/~dawnsong/athena/
- Graphviz - http://www.graphviz.org/ (volitelná součást)

Instalace:

K spuštění verifikačního nástroje Athena je nutné získat několik komponent. Jedná se o překladač SML, konkrétně Standard ML of New Jersey ve verzi 110.0.7. (při použití novější verze se mohou objevit problémy). Dostupné jsou jak instalační soubory pro Unix, tak pro Windows. V systému Windows se musí přidat adresář s nainstalovaným SML do systémových proměnných. Pro spuštění programu Athena postačuje instalace verze APV určené k verifikaci. Soubory stačí rozbalit do libovolného adresáře a aplikace bude připravena k použití. Program Graphviz slouží ke grafickému znázornění výstupu. Opět se jedná o multiplatformní aplikaci s jejíž instalací nejsou žádné problémy.

Spuštění:

Program se spouští z prostředí SML. Z adresáře, kde se nachází rozbalený program APV, se příkazem SML spustí příkazový řádek programu Standard ML of New Jersey. Pro spuštění Atheny je nutné použít dva příkazy `use "shell.sml";` a `use "athena.sml";`, oba tyto soubory se nacházejí v aktuálním adresáři. Na konci souboru athena.sml se nachází příkaz, kterým se specifikuje soubor s verifikovaným protokolem. Příkaz pro verifikaci vypadá následovně:

```
val s = Athena.athena ("nazev",
"./vstup"::"./dot"::"./res"::"3"::"8000"::"false"::"true"::"true"::"false"
::"0"::["0"]);
```

První parametr pouze identifikuje zvolený protokol, dále následuje vstupní soubor dot, soubor s popisem grafického uspořádání stavů, soubor s výsledkem a další. Blíže je struktura popsána právě v souboru athena.sml. Příkaz lze ze souboru vyjmout a vepsat přímo do konzoly SML. Tento postup představuje ovšem úskalí, kdy dokud běží SML a byl verifikován jistý vstupní soubor, tak tento soubor nelze do ukončení SML měnit.

Pro grafickou prezentaci výstupu lze použít buď programu Graphviz, jehož ovládání je velmi intuitivní, nebo použití příkazu dot, který je schopen produkovat výstup ve formátu ps nebo png, např. dot -Tps dotVstup -o psVystup.

Příloha 2. Ukázka zdrojových textů

Andrew Secure RPC

```
BEGIN
1:1 Protocol: AndrewSecureRPC
  P_A(0,4){
    VAR: P_A, P_B, NONCE_Na, NONCE_Nb, NONCE_Nb2, SESKEY_Kab;

    -> : C[P_A, E{NONCE_Na, SYMKEY_(P_A,P_B)}] | New(NONCE_Na);
     <- : E{C[NONCE_Na,NONCE_Na,NONCE_Nb],SYMKEY_(P_A,P_B)};
     -> : E{C[NONCE_Nb,NONCE_Nb],SYMKEY_(P_A,P_B)};
     <- : E{C[SESKEY_Kab, NONCE_Nb2],SYMKEY_(P_A,P_B)};
  }
  P_B(1,4){
  VAR: P_A, P_B, NONCE_Na, NONCE_Nb, NONCE_Nb2, SESKEY_Kab;

    <- : C[P_A, E{NONCE_Na, SYMKEY_(P_A,P_B)}];
    -> : E{C[NONCE_Na,NONCE_Na,NONCE_Nb],SYMKEY_(P_A,P_B)} | New(NONCE_Nb);
    <- : E{C[NONCE_Nb,NONCE_Nb],SYMKEY_(P_A,P_B)};
    -> : E{C[SESKEY_Kab, NONCE_Nb2],SYMKEY_(P_A,P_B)}
      New(SESKEY_Kab,NONCE_Nb2);
  }
  End

  Spec {
    VC. {Strand(0,4)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Nb2,Nb1),
            (NONCE_Na,Na0),(SESKEY_Kab,Ses1)]} =>
        {Strand(1,4)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Nb2,Nb1),
            (NONCE_Na,Na0),(SESKEY_Kab,Ses1)]}
  }
  END
```

Denning Sacco

```
BEGIN
1:1 Protocol: DenningSacco
  P_A(0,3){
  VAR: P_A, P_B, P_S, SESKEY_Kab, DATA_T;

    -> :  C[P_A, P_B];
    <- : E{C[P_B, SESKEY_Kab, DATA_T, E{C[SESKEY_Kab, P_A, DATA_T],
            SYMKEY_(P_B,P_S)}], SYMKEY_(P_A,P_S)};
    -> :  E{C[SESKEY_Kab,P_A,DATA_T],SYMKEY_(P_B,P_S)};
  }
  P_B(1,1){
  VAR: P_A, P_B, P_S, SESKEY_Kab, DATA_T;

    <- :  E{C[SESKEY_Kab, P_A, DATA_T],SYMKEY_(P_B,P_S)};
  }
  P_S(2,2){
  VAR: P_A, P_B, P_S, SESKEY_Kab, DATA_T;

    <- :  C[P_A, P_B];
    -> : E{C[P_B, SESKEY_Kab, DATA_T, E{C[SESKEY_Kab,P_A,DATA_T],
            SYMKEY_(P_B,P_S)}], SYMKEY_(P_A,P_S)} | New(SESKEY_Kab);
  }
  End

Spec {
  VC. {Strand(1,1)[(P_A,A0),(P_B,B0),(P_S,S0),(DATA_T,T0),
            (SESKEY_Kab,Kab0)]} =>
      {Strand(0,3)[(P_A,A0),(P_B,B0),(P_S,S0),(DATA_T,T0),
            (SESKEY_Kab,Kab0)]},
  VC. {Strand(0,3)[(P_A,A0),(P_B,B0),(P_S,S0),(DATA_T,T0),
            (SESKEY_Kab,Kab0)]} =>
      {Strand(1,1)[(P_A,A0),(P_B,B0),(P_S,S0),(DATA_T,T0),
            (SESKEY_Kab,Kab0)]}
}
END
```

Needham Schroeder (veřejný klíč)

```
BEGIN
1:1 Protocol: NeedhamSchroederAsym
  P_A(0,3){
  VAR: P_A, P_B, NONCE_Na, NONCE_Nb;

  -> : E{C[NONCE_Na,P_A],PUBKEY_P_B} | New(NONCE_Na);
  <- : E{C[NONCE_Na,NONCE_Nb],PUBKEY_P_A};
  -> : E{NONCE_Nb,PUBKEY_P_B};
}

  P_B(1,3){
  VAR: P_A, P_B, NONCE_Nb, NONCE_Na;

  <- : E{C[NONCE_Na,P_A],PUBKEY_P_B};
  -> : E{C[NONCE_Na,NONCE_Nb],PUBKEY_P_A} | New(NONCE_Nb);
  <- : E{NONCE_Nb,PUBKEY_P_B};
  }

End

Spec {
  VC. {Strand(1,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]} =>
        {Strand(0,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]},
  VC. {Strand(0,3)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]} =>
        {Strand(1,2)[(P_A,A0),(P_B,B0),(NONCE_Nb,Nb0),(NONCE_Na,Na0)]}
      }

END
```

Woo Lam Π 1

```
BEGIN
1:1 Protocol: WooLamPi1
  P_A(0,3){
  VAR: P_A,P_B,P_S,NONCE_Nb;

    -> : P_A;
    <- : NONCE_Nb;
    -> : E{C[P_A,P_B,NONCE_Nb],SYMKEY_(P_A,P_S)};
  }
  P_B(1,5){
  VAR: P_A,P_B,P_S,NONCE_Nb,ANY_X;

    <- : P_A;
    -> : NONCE_Nb | New(NONCE_Nb);
    <- : ANY_X;
    -> : E{C[P_A,P_B,ANY_X],SYMKEY_(P_B,P_S)};
    <- : E{C[P_A,P_B,NONCE_Nb], SYMKEY_(P_B,P_S)};
  }
  P_S(2,2){
  VAR: P_A,P_B,P_S,NONCE_Nb,ANY_X;

    <- : E{C[P_A, P_B, E{C[P_A, P_B, NONCE_Nb], SYMKEY_(P_A, P_S)}],
            SYMKEY_(P_B,P_S)};
    -> : E{C[P_A,P_B,NONCE_Nb], SYMKEY_(P_B,P_S)};
  }
  End

Spec {
  VC. {Strand(1,5)[(P_A,A0),(P_B,B0),(P_S,S0),(NONCE_Nb,Nb0)]} =>
      {Strand(0,3)[(P_A,A0),(P_B,B0),(P_S,S0),(NONCE_Nb,Nb0)]}
}

END
```