

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH ZERO-KNOWLEDGE PROTOKOLŮ

DIPLOMOVÁ PRÁCE

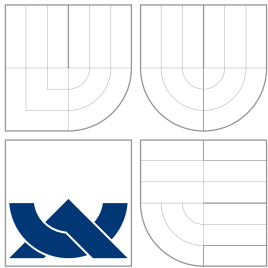
MASTER'S THESIS

AUTOR PRÁCE

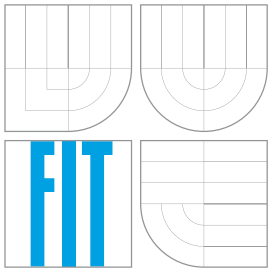
AUTHOR

Bc. JAN ŠAFÁŘ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH ZERO-KNOWLEDGE PROTOKOLŮ

DESIGN OF ZERO-KNOWLEDGE PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN ŠAFÁŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL OČENÁŠEK, Ph.D.

BRNO 2010

Abstrakt

Práce se zabývá automatizovanými metodami pro návrh bezpečnostních protokolů a jejich využitím pro návrh zero knowledge protokolů, nebo protokolů, kde je možné využít ZK protokoly jako subprotokoly. Konkrétně je kladen důraz na využití kompoziční metody. Práce také ukazuje možnou implementaci dané metody.

Abstract

Thesis introduces automated methods of protocol design and their usability for zero knowledge protocol design or protocols, where ZK protocols are used as subprotocols. Especially composition method is described more in depth. Thesis shows also a sample implementation of this method

Klíčová slova

zero knowledge, návrh bezpečnostních protokolů, automatizovaný návrh protokolů

Keywords

zero knowledge, security protocol design, automated protocol design

Citace

Jan Šafář: Návrh zero-knowledge protokolů, diplomová práce, Brno, FIT VUT v Brně, 2010

Návrh zero-knowledge protokolů

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně, pod vedením Ing. Pavla Očenáška Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal

.....
Jan Šafář
24. května 2010

© Jan Šafář, 2010.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1	Úvod	3
1.1	Motivace	3
1.2	Struktura dokumentu	3
2	Bezpečnostní protokoly	4
2.1	Úvod	4
2.2	Techniky bezpečnostních protokolů	5
2.2.1	Kryptografie	5
2.2.2	Hashování	6
2.2.3	Nonce a časová razítka	6
2.3	Formální popis	6
2.3.1	Tradiční zápis	6
2.3.2	Gramatika	7
2.4	Příklady protokolů	8
2.4.1	Yahalom	8
2.4.2	Needham-Shroeder asymetrická verze	8
2.4.3	Otway Rees	9
2.5	Závěr	9
3	Návrh bezpečnostních protokolů	10
3.1	Úvod	10
3.2	Analytický přístup	11
3.3	Automatický přístup	11
3.3.1	Návrh pomocí logiky	12
3.3.2	Evoluční návrh	14
3.4	Kompoziční návrh	15
3.4.1	Primitiva	18
3.4.2	Kompozice	20
3.4.3	Rozšíření metody	21
3.5	Závěr	21
4	Zero knowledge protokoly	22
4.1	Úvod	22
4.2	Interaktivní dokazovací systém	23
4.3	Další ZK vlastnosti	23
4.4	Ukázka - Feige-Fiat-Shamir důkaz identity	23
4.5	Závěr	24

5	Návrh ZK protokolů	25
5.1	Úvod	25
5.2	Vstupní a výstupní znalosti	25
5.3	Úpravy metod	26
5.3.1	Evoluční návrh	26
5.3.2	Kompoziční návrh	26
5.3.3	Návrh pomocí logiky	26
5.4	Závěr	26
6	Implementace	27
6.1	Úvod	27
6.2	Popis protokolu	27
6.2.1	Klauzule používané v definici	28
6.2.2	Generování unikátních čísel	29
6.2.3	Příklad definice protokolu	30
6.3	Popis primitiv	30
6.3.1	Příklad primitiva	31
6.3.2	Vytváření nových znalostí	33
6.4	Způsob prohledávání stavového prostoru	33
6.5	Spuštění programu a výstup	34
6.6	Srovnání s analytickým použitím metody	35
6.7	Benchmark	37
6.7.1	Lineární topologie	37
6.7.2	Fullmesh topologie	38
6.7.3	Dlouhodobé klíče	39
6.8	Závěr	40
7	Možnosti rozšíření	41
7.1	Nová primitiva	41
7.2	Provázání primitiv	41
7.3	Další optimalizace	41
8	Závěr	43

Kapitola 1

Úvod

1.1 Motivace

Tématem této práce je vytvořit přehled o metodách návrhu bezpečnostních protokolů, zvláště pak o automatických metodách, a možnostech těchto metod při návrhu zero knowledge protokolů, nebo protokolů kde se ZK protokoly vyskytují jako subprotokoly. Důraz je kladen na kompoziční metody. Práce také popisuje její možnou implementaci.

Zero knowledge protokoly jsou jedním z typů bezpečnostních protokolů, která se vyznačuje tím, že během komunikace dvou subjektů v síti nedojde k vyzrazení jakýchkoliv informací útočníkovi, které před během protokolu neznal.

1.2 Struktura dokumentu

Kapitola 'Bezpečnostní protokoly' se zabývá všeobecně bezpečnostními protokoly, jejich specifikací, způsoby zápisu a použití. V rámci kapitoly budou také uvedeny některé protokoly jako příklad.

V kapitole 'Návrh bezpečnostních protokolů' budou představeny metody návrhu bezpečnostních protokolů, s důrazem na kompoziční metodu.

Kapitola 'Zero knowledge protokoly' je věnována specifikaci zero knowledge protokolů, jejich možné implementaci a použití.

'Návrh zero knowledge protokolů' se zabývá možnostmi automatického návrhu protokolů jejichž součástí jsou zero knowledge protokolů a problémy, které při použití automatického návrhu nastávají.

Implementace kompoziční metody je představena v kapitole 'Implementace'. Další možnosti rozšíření metody jsou diskutována v kapitole 'Rozšíření implementace'.

Kapitola 2

Bezpečnostní protokoly

2.1 Úvod

Význam slova *protokol* vnímáme z pohledu informatiky jako množinu pravidel, podle které se odehrává komunikace mezi dvěma subjekty. Protokol může obsahovat definici následujících parametrů a procedur [16]:

- Detekce fyzického a logického spojení
- 'Handshake'
- Vyjednávání o parametrech spojení
- Operace před a po odeslání zprávy
- Formát jednotlivých zpráv
- Chování při obdržení neplatné zprávy
- Chování při výpadku spojení
- Způsob ukončení komunikace

Nejtypičtějším příkladem protokolů může být například IP, internet protocol, a nebo TCP, transmission control protocol, což jsou protokoly pomocí kterých se odehrává většina komunikace v internetu.

Bezpečnostní protokoly jsou jedna z mnohých skupin protokolů, která se zabývá, bezpečnou komunikací mezi několika subjekty. Pod pojmem bezpečnou můžeme rozumět, že komunikace splňuje některé z těchto vlastností:

- Autentičnost - účastník ví od koho zprávu dostal
- Integrita - účastník ví, že zpráva nebyla cestou změněna
- Důvěrnost - pouze oprávnění účastníci znají obsah zprávy

Bezpečnostní protokoly se používají nejčastěji pro bezpečný přenos zpráv, distribuci klíčů nebo autentizaci uživatelů.

2.2 Techniky bezpečnostních protokolů

K zajištění autentičnosti, integrity a důvěrnosti se používá v bezpečnostních protokolech několik technik [18]. V následujícím textu, budou uvedeny nejvýznamnější z nich.

2.2.1 Kryptografie

Kryptografie je disciplína, která se věnuje skrývání informací a jejich opětovnému odkrytí, většinou mluvíme o operacích šifrování a dešifrování. V současné době mluvíme o tzn. kryptografii moderní, kde bezpečnost zašifrované zprávy spoléhá na sílu klíče, který byl použit pro její zašifrování, na rozdíl od kryptografie klasické, kde byl kladen důraz na ukrytí šifrovacího algoritmu.

V současné době se používají dva typy kryptografie, symetrická a asymetrická, které se liší v typech použitých klíčů. Oba typy kryptografie lze použít pro zajištění důvěrnosti a autentičnosti.

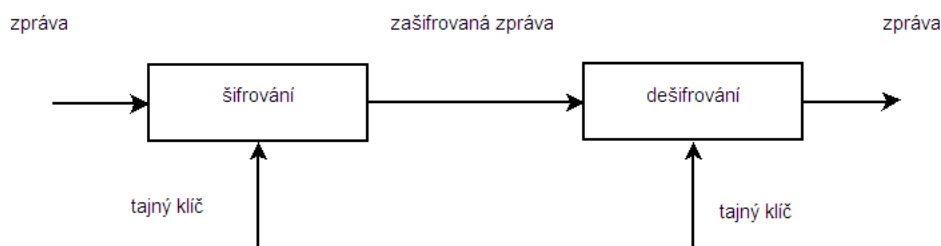
Symetrická kryptografie

Symetrická kryptografie využívá pro obě operace, šifrování i dešifrování, stejný tajný klíč, který musí znát obě strany, které se účastní komunikace. Z toho vzniká problém, jakým způsobem propagovat klíč ke všem zúčastněným stranám, tak aby se ho nikdo jiný nedozvěděl. Pokud je vyžadováno aby každá dvojice v rámci skupiny měla unikátní klíč, vzniká další problém, že množství klíčů roste s druhou mocninou velikosti skupiny.

Naopak výhodou tohoto způsobu šifrování je menší výpočetní náročnost při šifrování a dešifrování zpráv ve srovnání s asymetrickou kryptografií.

Pro klíč, který se používá v symetrické kryptografii, se používá označení tajný klíč, secret key. V současné době se za bezpečnou délku klíče v symetrické kryptografii považuje délka 256 bitů.

Algoritmus DES, data encryption standart, nebo AES, advanced encryption standart, jsou nejznámějšími zástupci šifrovacích algoritmů, které využívají symetrickou kryptografii.



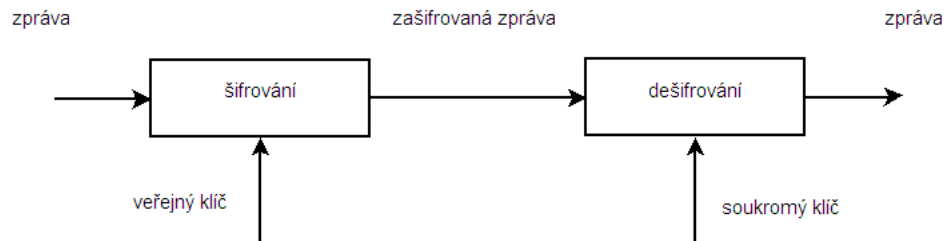
Obrázek 2.1: Diagram symetrické kryptografie

Asymetrická kryptografie

Asymetrická kryptografie, občas také nazývaná kryptografie veřejného klíče, využívá pro operace pár klíčů, soukromý a veřejný, secret a public (občas také private a shared). Zprávy zašifrované veřejným klíčem lze dešifrovat pouze klíčem soukromým a naopak. Klíče jsou konstruovány tak, aby bylo velmi náročné odvodit z veřejného klíče klíč soukromý, neboli v časovém horizontu kdy se bude klíč používat, je nemožné odvodit z veřejného klíče, klíč soukromý.

Účastníkům komunikace stačí aby zveřejnili své veřejné klíče a kdokoliv s nimi poté chce komunikovat zašifruje zprávu jejich veřejným klíčem a odešle. Svůj soukromý klíč účastníci nikdy nezveřejňují. Oproti symetrické kryptografii musí mít každý účastník ve skupině pouze jeden pár klíčů. Bezpečná délka klíče je v současné době 2048 bitů.

Asymetrickou kryptografií využívá například algoritmus RSA.



Obrázek 2.2: Diagram asymetrické kryptografie

2.2.2 Hashování

Hash funkce je funkce, která mapuje jakýkoliv vstup na výstup pevné velikosti, který se nazývá hash nebo message digest. Ideální kryptografická hash funkce má tyto vlastnosti:

- Je jednoduché vypočítat hash
- Je výpočetně nezávládnutelné najít zprávu s určitým hashem
- Je výpočetně nezávládnutelné modifikovat zprávu bez změny hashe
- Je výpočetně nezávládnutelné najít 2 různé zprávy se stejným hashem

Hash funkce se nejčastěji využívají pro zajištění integrity zpráv nebo uložení hesel. Nejznámější zástupci jsou SHA-1 a MD5.

2.2.3 Nonce a časová razítka

Nonce je nějaká hodnota, která je během protokolu použita pouze jednou. Používá se pro zajištění autentičnosti. Podobnou funkci má i časové razítko, které lze použít také pro kontrolu časového pořadí zpráv.

O nonce a časových razítkách často říkáme, že jsou fresh, tzn. že jsou nově vygenerovaná, nebyly 'nikdy' předtím použita.

2.3 Formální popis

2.3.1 Tradiční zápis

Pro zápis bezpečnostních protokolů se využívá nejčastěji zápis ve formě seznamu událostí [2]. Jednotlivé události zapisujeme ve tvaru:

$$P_1 \rightarrow P_2 : msg$$

Kde P_1 je odesílatel a P_2 příjemce zprávy s obsahem msg . Formálně můžeme říct, že událost je trojice (P_1, P_2, msg) .

V obsahu zprávy mohou být následující prvky:

- Účastníci A, B, \dots
- Symetrické klíče Kab, Ksa, \dots
Písmena za znakem K většinou určují účastníky, kteří by měli klíč sdílet
- Asymetrické klíče $KSa, KPa, KSb, KPb, \dots$
 KP je veřejný a KS soukromý klíč a poslední písmeno určuje účastníka, který vygeneroval tento pár klíčů
- Nonce Na, Nb, \dots
- Časové razítko Ta, Tb, \dots
- Funkci $f(), g(), \dots$
Ve zprávě je obsažena její návratová funkce a může se jednat o libovolnou funkci například hashovací funkci, +1 atd.
- Složení prvků (\dots, \dots)
- Zašifrování prvků $\{\dots, \dots\}K$
 K je klíč použitý k zašifrování zprávy

Příklad:

$$A \rightarrow B : \{Na, A, B\}Kab, h((Na, A, B))$$

..

2.3.2 Gramatika

Pro generování protokolů, které jsou definovány pomocí těchto pravidel lze použít následující gramatiku, jedná se o částečně upravenou gramatiku z [10].

$$G = (N, \Sigma, P, S)$$

- $N = \{ \langle \text{protocol} \rangle, \langle \text{event} \rangle, \langle \text{events}' \rangle, \langle \text{tuple} \rangle, \langle \text{atom} \rangle, \langle \text{atom}' \rangle, \langle \text{plain} \rangle, \langle \text{enc} \rangle, \langle \text{apply} \rangle, \langle \text{id} \rangle, \langle \text{key} \rangle, \langle \text{nonce} \rangle, \langle \text{func} \rangle \}$
- $\Sigma = \{ \text{ids} = \{A, B, \dots\}, \text{keys} = \{Kab, KPa, KSa, \dots\}, \text{funcs} = \{f(), \dots\}, \text{nonces} = \{Na, Nb, \dots\}, \text{timestamps} = \{Ta, Tb, \dots\}, \text{;}, \text{rightarrow}, \text{comma}, \text{()}, \text{\{ \}}, \text{newline} \}$
- $P = \{$
 - $\langle \text{protocol} \rangle \rightarrow \langle \text{event} \rangle \mid \langle \text{event}' \rangle \mid \epsilon$
 - $\langle \text{event}' \rangle \rightarrow \langle \text{event} \rangle \text{newline} \langle \text{event}' \rangle \mid \langle \text{event} \rangle$
 - $\langle \text{event} \rangle \rightarrow \langle \text{id} \rangle \text{rightarrow} \langle \text{id} \rangle : \langle \text{tuple} \rangle$
 - $\langle \text{tuple} \rangle \rightarrow \langle \text{atom} \rangle \mid \langle \text{atom}' \rangle$
 - $\langle \text{atom}' \rangle \rightarrow \langle \text{atom} \rangle \text{comma} \langle \text{atom}' \rangle \mid \langle \text{atom} \rangle$
 - $\langle \text{atom} \rangle \rightarrow \langle \text{enc} \rangle \mid \langle \text{plain} \rangle$

- $\langle \text{plain} \rangle \rightarrow \langle \text{id} \rangle \mid \langle \text{key} \rangle \mid \langle \text{nonce} \rangle \mid \langle \text{timestamp} \rangle \mid \langle \text{apply} \rangle \mid (\langle \text{tuple} \rangle)$
- $\langle \text{enc} \rangle \rightarrow \{ \langle \text{tuple} \rangle \} \langle \text{key} \rangle$
- $\langle \text{apply} \rangle \rightarrow \langle \text{func} \rangle (\langle \text{tuple} \rangle)$
- $\langle \text{id} \rangle \rightarrow \text{id}s$
- $\langle \text{key} \rangle \rightarrow \text{keys}$
- $\langle \text{func} \rangle \rightarrow \text{funcs}$
- $\langle \text{nonce} \rangle \rightarrow \text{nonces}$
- $\langle \text{timestamps} \rangle \rightarrow \text{timestamps}$
- $S = \{ \langle \text{protocol} \rangle \}$

Symbolsy *comma*, *rightarrow* a *newline* reprezentují čárku, šipku a nový řádek. Z důvodu přehlednosti byli uvedeni v textové podobě.

2.4 Příklady protokolů

2.4.1 Yahalom

Yahalom [3] je protokol určený pro autentizaci a distribuci symetrických klíčů s využitím důvěryhodného serveru.

$$A \rightarrow B : A, Na$$

$$B \rightarrow S : B, \{A, Na, Nb\}Kbs$$

$$S \rightarrow A : \{B, Kab, Na, Nb\}Kas, \{A, Kab\}Kbs$$

$$A \rightarrow B : \{A, Kab\}Kbs, \{Nb\}Kab$$

2.4.2 Needham-Shroeder asymetrická verze

Protokol zajišťuje vzájemnou autentizaci obou účastníků komunikace s pomocí důvěryhodného severu. Protokol využívá asymetrickou kryptografii. Byl poprvé představen v [11].

$$A \rightarrow S : A, B$$

$$S \rightarrow A : \{KPb, B\}KSs$$

$$A \rightarrow B : \{Na, A\}KPb$$

$$B \rightarrow S : B, A$$

$$S \rightarrow B : \{KPa, A\}KSs$$

$$B \rightarrow A : \{Na, Nb\}KPa$$

$$A \rightarrow B : \{Nb\}KPb$$

2.4.3 Otway Rees

Protokol [14] distribuuje nový symetrický klíč pro každou komunikaci. Nové klíče jsou generovány důvěryhodnou autoritou. Protokol lze využít i v případech kdy pouze jedna strana může komunikovat s důvěryhodným serverem.

$$A \rightarrow B : Nm, A, B, \{Na, Nm, A, B\}Kas$$

$$B \rightarrow S : Nm, A, B, \{Na, Nm, A, B\}Kas$$

$$S \rightarrow B : Nm, \{Na, Kab\}Kas, \{Nb, Kab\}Kbs$$

$$B \rightarrow A : Nm, \{Na, Kab\}Kas$$

2.5 Závěr

Kapitola popisuje prvky a techniky využívané v bezpečnostních protokolech. Seznamuje čtenáře také s typickým zápisem bezpečnostních protokolů, tak jak se používá v literatuře. Tento zápis je demonstrován na příkladech nejznámějších bezpečnostních protokolů. Náplní následující kapitoly jsou techniky, pomocí kterých se bezpečnostní protokoly navrhují.

Kapitola 3

Návrh bezpečnostních protokolů

3.1 Úvod

Návrh bezpečnostních protokolů spočívá ve vytvoření protokolu, který splňuje specifikované požadavky. Pro vytváření protokolů se používají dva přístupy analytický a automatický. Při analytickém způsobu návrhu návrhář vytváří protokol sám, bez použití počítačových aplikací, na základě zkušeností. Po dokončení návrhu se většinou protokol verifikuje pomocí verifikačních nástrojů a v případě, že je v nalezena nějaká chyba, která by mohla být využita útočníky, návrhář protokol opraví.

V případě automatizovaných metod návrhář pouze požadavky předá jako vstupní parametry počítačovému programu. Metoda mu poté vrátí navržený bezpečnostní protokol. Verifikace je v ale nutná i v tomto případě, protože pokud byly špatně definovány požadavky nemusí protokol splňovat zamýšlená kritéria. V případě, že jsou požadavky správné a i přesto automatická metoda nevrátí bezpečný protokol je nutné protokol upravit pomocí analytického přístupu.

Požadavky lze zadat ve formě znalostí [12], která mají jednotlivé subjekty, které se účastní komunikace, před a po běhu protokolu. Také se většinou specifikují i znalosti, které nesmí určití účastníci znát. Například

- Vstupní znalosti

$$A = \{B, Kas\}$$

$$B = \{Kbs\}$$

$$S = \{Kas, Kbs, Kab\}$$

- Výstupní znalosti

$$A = \{B, Kas, Kab\}$$

$$B = \{A, Kbs, Kab\}$$

$$S = \{Kas, Kbs, Kab\}$$

- Zakázané znalosti

$$A = \{Kbs\}$$

$$B = \{Kas\}$$

$$S = \{\}$$

$$I = \{K_{as}, K_{bs}, K_{ab}\}$$

Ve specifikacích se také často uvádí typ komunikačního kanálu mezi jednotlivými subjekty. Zda se jedná o jednosměrný či obousměrný kanál nebo bezpečný či nebezpečný, tzn. zda je možné komunikovat skrz kanál i bez šifrování zpráv aniž by byl útočník schopný zprávy odposlechnout. Popřípadě se specifikuje zda kanál vůbec existuje, viz. například protokol Otway Rees.

3.2 Analytický přístup

Protože analytický přístup je poměrně náročný a návrháři bez řádných zkušeností mohou při návrhu protokolu udělat závažné chyby, byly vytvořeny [1] určitá pravidla, které by měli napomáhat při tvorbě bezpečnostních protokolů.

1. Interpretace zprávy musí zcela záviset na obsahu
Příjemce zprávy musí být schopný rozlišit význam zprávy pouze na základě jejího obsahu, význam nesmí záležet na kontextu v kterém je zpráva doručena. Zabráníme tím možnému podvrhnutí zprávy.
2. Podmínky na provedení určité akce, po přijetí zprávy, musí být jasné dané
3. Identita účastníka musí být obsažena ve zprávě, pokud je nutná pro její interpretaci
Nesmí být možné zaměnit zprávy od jednoho účastníka za zprávy od jiného účastníka.
4. Použití šifrování musí mít jasně stanovený důvod
Šifrování je náročná operace a neměla by se využívat bezdůvodně. Pro každé použití šifrování je dobré mít specifikovaný důvod, proč je v dané části nutné.
5. Účastník, který vytvořil podpis již zašifrované zprávy nemusí znát její obsah.
6. Účastník, který vytvořil podpis zprávy a poté ji zašifroval, může znát její obsah.
7. Je nutné jasně stanovit předpoklady o nonce hodnotách.
8. Je nutné jasně specifikovat důvěryhodné vztahy
9. a další

Na základě těchto pravidel je možné dosáhnout při analytickém návrhu menšího počtu chyb. Problémem ale stále zůstává poměrně velká časová náročnost návrhu.

3.3 Automatický přístup

Z důvodů značné složitosti a chybovosti návrhu bezpečnostních protokolů začali vznikat různé automatizované metody návrhu, které by tuto obtížnou disciplínu zjednodušily [12].

V této části bude představeno několik automatických metod, které pracují na různých principech.

3.3.1 Návrh pomocí logiky

Simple authentication logic [4], dále jen SAL, vychází z BAN logiky [3]. Základem modelu jsou účastníci a kanály, skrz které si účastníci zasílají zprávy. Návrhář definuje vstupní požadavky pomocí formulí logiky jako předpoklady. Z těchto předpokladů, se pomocí syntetických pravidel vytvoří jednodušší formule. Po dokončení redukci předpokladů se z těchto jednoduchých formulí vytvoří zprávy protokolu. Problémem této metody je, že nedává přesný, algoritmický, předpis jakým způsobem zprávy protokolu vytvářet z této množiny redukováných formulí.

Kanály

V SAL jsou kanály charakterizovány pomocí dvou množin, množinou ‘čtenářů’ a množinou ‘písařů’, značíme je $r(C)$ a $w(C)$, kde C je kanál. Účastníci, kteří jsou v těchto množinách mohou přijímat/číst, respektive odesílat/zapisovat na daný kanál.

Příslušnost jednotlivých účastníku do jednotlivých množin kanálů je podmíněna znalostí jak z kanálu číst nebo do něj zapisovat. Tyto znalosti tvoří množinu, kterou označujeme jako C^r a C^w . Účastníci, kteří znají všechny prvky z množin C^r a/nebo C^w poté patří do množin $r(C)$ a/nebo $w(C)$.

Logika

Podobně jako každá logika se i SAL skládá ze základních formulí a inferenčních pravidel. K nim navíc logika obsahuje i opačná, již zmíněná, syntetická pravidla [4].

Základní formule

$P \triangleleft C(X)$: P vidí $C(X)$, aneb že někdo poslal zprávu X skrz kanál C , P ale nemusí umět zprávu X přečíst.

$P \triangleleft X \mid C$: P obdržel zprávu X z kanálu C , P vidí $C(X)$ a umí číst z kanálu C

$P \triangleleft X$: P obdržel X , někdo poslal zprávu X na kanálu, který P umí číst

$\#(X)$: X je fresh, X nebylo použito před tímto během protokolu

$P \sim X$: P někdy v minulosti řeklo X , nevíme přesně kdy

$P \parallel \sim X$: P nedávno, v tomto běhu protokolu, řeklo X

$P \models \phi$: P věří ϕ , kde ϕ je formule

Formule se dají skládat i pomocí logických spojek \wedge , \vee a \rightarrow .

Důvěryhodný a kompetentní účastník

Logika říká, že důvěryhodný účastník pro účastníka P je takový účastník, který pokud něco řekl, tak tomu sám věří. V jazyce logiky

$$P \models ((Q \parallel \sim \phi) \rightarrow (Q \models \phi))$$

Kompetentní účastník pro P je potom takový účastník, který pokud věří, že je něco pravda, tak je to pravda.

$$P \models ((Q \models \phi) \rightarrow \phi)$$

Inferenční pravidla

Na základě inferenčních pravidel je možné ze základních formulí vytvářet formule nové. Budou uvedeny jenom některé, podrobnější přehled je k dispozici v literatuře [4].

Pravidla vidění S1: Pravidlo říká, že pokud P vidí $C(X)$ a může kanál C číst, potom P rozpozná, že obdržel zprávu X skrz kanál C a vidí tuto zprávu.

$$\frac{P \triangleleft C(X), P \in r(C)}{P \models (P \triangleleft X \mid C), P \triangleleft X}$$

Pravidlo vidění S2: Pravidlo říká, že pokud účastník vidí složenou zprávu, potom vidí i jednotlivé části.

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

Syntetická pravidla

Z dříve uvedených inferenčních pravidel bylo odvozeno 9 syntetických pravidel, pomocí kterých se formule redukuje na jednodušší. Syntetická pravidla budou zapsány v následující formě:

$$\begin{aligned} G \\ \hookrightarrow G_1 \\ \hookrightarrow G_2/G_3 \\ \hookrightarrow \dots \end{aligned}$$

Tzn. pro dosažení cíle/předpokladu G je nutné splnit všechny podcíle G_1, G_2 nebo G_3, \dots .

Opět budou uvedeny pouze některá pravidla jako příklad.

Pravidlo Syn1: Aby P poznal, že zpráva X přišla skrz kanál C , je nutné aby P přijal $C(X)$ a uměl číst z kanálu C .

$$\begin{aligned} P \models (P \triangleleft X \mid C) \\ \hookrightarrow P \triangleleft C(X) \\ \hookrightarrow P \in r(C) \end{aligned}$$

Pravidlo Syn2: Aby P obdržel zprávu X , musí být zpráva X součástí jiné zprávy nebo musí tuto zprávu přijmout skrz nějaký kanál C .

$$\begin{aligned} P \triangleleft X \\ \hookrightarrow P \triangleleft (X, Y) / P \models (P \triangleleft X \mid C) \end{aligned}$$

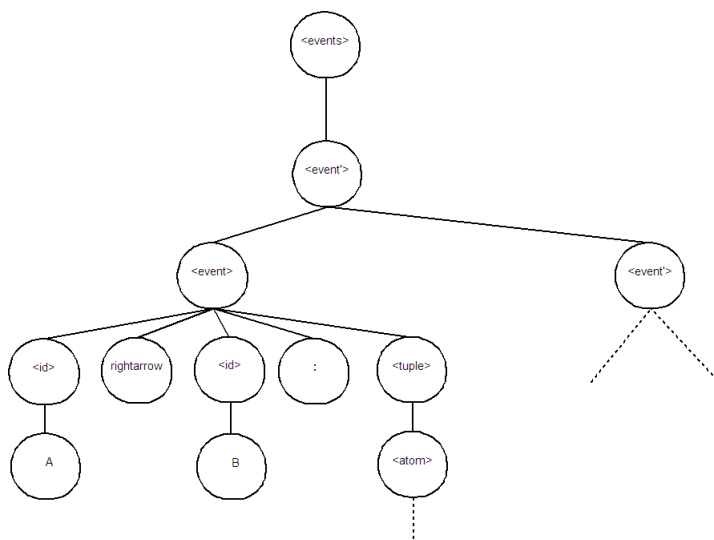
3.3.2 Evoluční návrh

Evoluční návrh [13] bezpečnostních protokolů využívá technik genetického programování. Základem genetického programování je populace jedinců, kde každý jedinec představuje možné řešení nějakého problému, v našem případě protokol. Z této populace je vybráno několik nejvhodnějších jedinců, kteří se stanou základem pro novou populaci. Nová populace se vytvoří pomocí mutací a křížením jedinců. A situace se opakuje dokud není nalezeno ideální řešení.

V dalších částech si rozebereme celý proces a jeho nejdůležitější části důkladněji.

Reprezentace protokolu

Protokol je reprezentován pomocí stromu, který vznikl aplikováním derivačních pravidel gramatiky, která byla uvedena dříve. Protokol je vlastně derivačním stromem věty, generované touto gramatikou. Příklad derivačního stromu je možné vidět na obrázku 3.1.



Obrázek 3.1: Příklad derivačního stromu

Evoluční algoritmus

Specifikace protokolu : Před spuštěním algoritmu je nutné specifikovat požadované vlastnosti, které má výsledný protokol mít. Jak už bylo uvedeno dříve, většinou se specifikují vstupní a výstupní vlastnosti a typy kanálů.

Vygenerování počáteční populace : Vytvoří se zcela náhodně určitý počet platných, ve smyslu že je lze vygenerovat uvedenou gramatikou, jedinců / protokolů. Tato populace je v prvním běhu současnou populací.

Výpočet fitness : Všechny protokoly jsou ohodnoceny pomocí fitness funkce. Čím lépe splňuje protokol specifikace, tím větší hodnotu fitness má.

Výběr jedinců : Z ohodnocené populace je vybráno několik jedinců. Metody výběru jsou různé, některé nejpoužívanější budou uvedeny později.

Křížení : Z vybraných jedinců jsou pomocí operace křížení vytvářeni jedinci dokud se nenaplňuje nová populace.

Mutace : Několik jedinců z nové populace je náhodně zmutováno.

Po zmutování nové generace se algoritmus vrací k výpočtu fitness a celý proces se opakuje. Algoritmus končí v případě, že je nalezen ideální jedinec nebo pokud uběhlo stanovený počet generací.

Způsob výběru jedinců

Ruletový výběr V ruletovém výběru je každému jedinci přidělena část prostoru o velikosti fitness. Poté se náhodně zvolí číslo z intervalu $< 0, \sum_{i=1}^N f_i >$, kde N je celkový počet jedinců a f_i je hodnota fitness jednoho jedince.

Ruletový výběr se touto způsobu říká z důvodu, že je možné si ho představit jako ruletové kolo, kde každému jedinci je přidělena určitá část kola. Čím větší má fitness tím větší šanci má se dostat do další generace.

Turnajový výběr Turnajový výběr ‘uspořádá’ několik turnajů o velikosti n mezi jedinci. Do turnaje jsou jedinci vybráni zcela náhodně. V každém má největší šanci uspět nejlepší jedinec, každý horší má šanci nižší. Konkrétně se dá šance na vítězství daného jedince vyjádřit vztahem $p(1-p)^{i-1}$, kde p je šance na vítězství prvního jedince a i je pozice jedince v seznamu seřazeném sestupně podle hodnoty fitness.

Úpravou velikosti turnaje a šanci na výběr nejlepšího jedince, můžeme vytvořit různé varianty turnajů, například deterministické ($p = 1$) nebo turnaje simulující náhodný výběr ($n = 1$).

Operace křížení a mutace

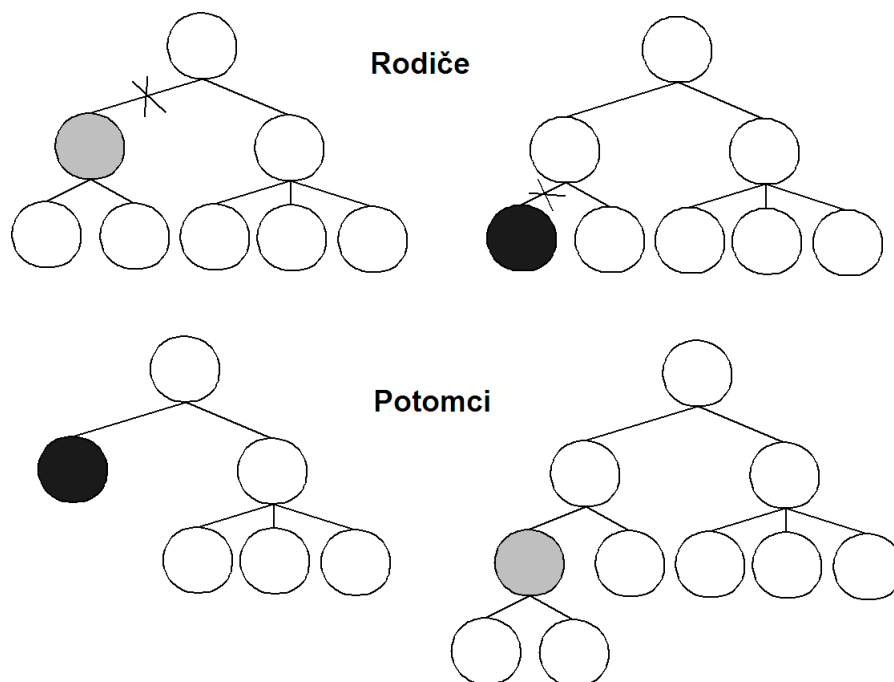
Při operaci křížení v genetickém programování, kde jsou jedinci reprezentováni pomocí stromu, se zvolí náhodně dva jedinci a v každém z nich se zvolí jeden podstrom. Tyto podstromy si jedinci vymění a vzniknou dva jedinci noví. Je ale nutné, zkontrolovat zda oba nově vytvořené jedinci jsou platným řešením. Operace je demonstrována na obrázku 3.2. Šedý a černý uzel značí kořeny měněných podstromů rodičů.

Mutací se jedinci odstraní jeden náhodný podstrom. Na jeho místě se poté vygeneruje nový platný podstrom. Příklad této operace je ukázán na obrázku 3.3. Je vidět, že na místě jednoprvkového podstromu, kde byl černý uzel kořenem, byl vygenerován nový strom. Šedý uzel je kořenem tohoto nového podstromu.

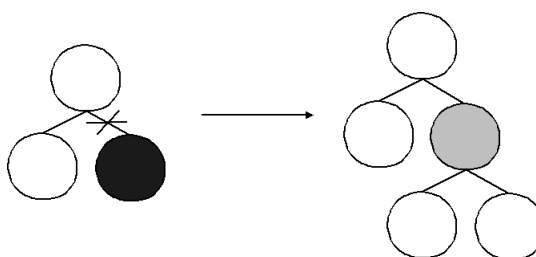
3.4 Kompoziční návrh

Návrh protokolu pomocí kompozice nelze přesně zařadit mezi automatické nebo analytické metody. Tak jak je navržena v [5] odpovídá spíše analytické metodě. Cílem této práce je ale implementace, nebo alespoň částečná implementace, této metody. Takže ji můžeme označit za analytickou metodu vhodnou pro automatické zpracování. Následující část se bude zabývat podrobnějším popisem návrhu protokolu s použitím kompozice.

Motivací pro vytvoření této metody bylo nabídnout systematický přístup, při vytváření bezpečnostních protokolů, který by nebyl postaven pouze na množině určitých doporučení.



Obrázek 3.2: Křížení v genetickém programování



Obrázek 3.3: Mutace v genetickém programování

Metoda využívá vlastnosti, že většina bezpečnostních protokolů se skládá z jednoduchých interakcí, typu dotaz-odpověď (challenge-response) a že složitější protokoly, se dají vytvořit kombinací těchto interakcí.

Označení kryptografických operací

Pro označení kryptografických operací bude v následujícím textu použito stejné značení jako v [5]. Toto značení bude také použito v implementaci.

$\{|m|\}_k$: Kryptografická operace zajišťující důvěrnost zprávy m s použitím klíče k

$[m]_k$: Kryptografická operace zajišťující integritu zprávy m pomocí klíče k

$\langle m \rangle_k$: Hašh zprávy m s využitím klíče k , pokud není klíč použit zapisujeme operaci jako $\langle m \rangle$

(a, b) : Konkaténace dvou částí zprávy

Pojmy použité v textu

V této části budou vysvětleny důležité pojmy, použité při popisu kompoziční metody nebo důležité pro pochopení fungování celé metody. Popis termínů, je zjednodušený. Podrobnější výklad jednotlivých termínů, lze najít v [5] v kapitole ‘Mathematical Preliminaries’. Tato část také slouží jako slovník, který shrnuje použité překlady jednotlivých termínů.

Množina proměnných a konstant : Značíme ji jako T

Množina klíčů : Množina klíčů, symetrických/asymetrických, krátkodobých/dlouhodobých. Množina klíčů je disjunktní s množinou proměnných a konstant. Označujeme ji symbolem K .

Množina termů : Sjednocení množiny klíčů s množinou proměnných a konstant. Termy si mohou účastníci protokolu mezi sebou zasílat. Značíme ji jako A

Množina základních termů : Sjednocení množiny klíčů a množiny proměnných.

Operace nad termy :

$\text{inv } K \rightarrow K$: Operace mapující klíč na jeho inverzní klíč. Asymetrický veřejný na asymetrický privátní a naopak. Symetrické klíče jsou mapovány sami na sebe.

$\text{key } A \rightarrow A$: Operace vytvoření klíče z termu, popřípadě termů.

$\text{conf } K \times A \rightarrow A$: Reverzibilní kryptografická operace zajišťující důvěrnost zprávy.

$\text{intg } K \times A \rightarrow A$: Reverzibilní kryptografická operace zajišťující integritu zprávy.

$\text{hash } K \times A \rightarrow A$: Ireverzibilní kryptografická operace.

$\text{join } A \times A \rightarrow A$: Spojení dvou termů. Jednotlivým termům říkáme **podtermy**.

Jednoduché termy : Termy, které nevznikli konkatencí termů. Opakem jsou termy **složené**.

Množinové operace : Základní operace nad množinami

$\text{Parts}(S)$: Množina všech podtermů termů v množině S .

$\text{Analz}(S)$: Množina všech podtermů termů v množině S . V případě, že term byl vytvořen nějakou reverzibilní kryptografickou operací, množina $\text{Analz}(S)$ obsahuje jeho podtermy pouze v případě, že obsahuje inverzní klíč, k této operaci.

$\text{Synth}(S)$: Množina všech termů, které lze vytvořit z množiny S pomocí operací nad termy

Ideál $I[S]$: Nejmenší množina termů, která obsahuje všechny pravky z S , je uzavřena vzhledem k operaci *join* a obsahuje termy, které vznikly použitím reverzibilních kryptografických operací na termy z $I[S]$ a inverzní klíč nepatří do $I[S]$. Jinými slovy je to nejmenší množina, kterou je nutné udržovat v tajnosti, aby nebylo možné zjistit žádný, ze základních termů S .

Koideál $C[S]$: Doplněk ideálu.

Spojovací množina \mathcal{B}_x : (Binding group) Množina účastníků protokolu spojených s tímto termem. Účastník, který term vytvořil mu přiřazuje spojovací množinu. Předpokládáme, že jednomu termu je přiřazena právě 1 spojovací množina.

Událost : Akce kterou může účastník protokolu ovlivňovat ostatní účastníky protokolu. Mezi tyto akce patří odeslání($e^+(x)$), přijmutí($e^-(x)$) a vygenerování termu((vx)). Dále pak srovnání termu vzhledem k předloze (pattern matching)($(x/p(x))$). Na pořadí události je kladeny 2 podmínky, odeslání termu musí předcházet vytvoření termu a přijmutí musí předcházet odeslání.

Nezávislé události : Jsou takové události, které na sobě nejsou časově závislé.

Vlákn :(Strand) Posloupnost událostí vykonaná jedním agentem.

Svazek :(Bundle) Množina vláken pro všechny agenty.

Trasa \mathcal{TR} :(Trace) Linearizace svazku. Svazek vláken je uspořádán do jednoho, tak, že je dodrženo pořadí událostí v jednotlivých vláknech i podmínky pořadí události.

Obsah trasy $m(\mathcal{TR})$: Sjednocení termů v událostech trasy.

Protokol P : Množina pravidel, která dovoluje agentům přidávat události do trasy. Pravidlo je dvojice (\mathcal{TR}, e) , která říká že pokud je současná trasa \mathcal{TR} je možné přidat událost e do \mathcal{TR} .

Běhy protokolu $Runs(P)$: Množina všech tras, kterou je možné daným protokolem vygenerovat.

Ochranná doména S_x :(Protective domain) Množina kterou je nutné držet v tajnosti aby nedošlo k vyzrazení x . Množina S_x obsahuje x a klíče účastníku ve spojovací množině B_x . O doméně dále říkáme že je I -kompatibilní pokud platí že $S_x \cap Parts(I) = \emptyset$. I jsou úvodní znalosti útočnicka.

Regulární protokol : Protokol je regulární, pokud dlouhodobá tajemství(klíče) nejsou obsahem nějaké jeho trasy.

Diskrétní trasa : Trasa je I -diskrétní pokud platí pro každou I -kompatibilní S_x $m(\mathcal{TR}) \subseteq C[S_x]$.

Diskrétní protokol : Každá trasa všech možných běhu protokolu je diskrétní.

3.4.1 Primitiva

Aby bylo možné kombinovat jednotlivé části, nazývané primitiva, do větších celků musí každé primitivum splňovat několik podmínek. V této části budou představeny jednotlivé primitiva společně s těmito podmínkami.

Základem konceptu primitiv jsou takzvané autentizační testy [9]. Autentizační test je dvojice zpráv (m_1, m_2) , taková, že m_2 vznikla netriviální kryptografickou operací, kterou nemohl provést útočník, ale mohl ji provést důvěryhodný účastník.

Rozlišujeme 3 typy těchto testů

1. **Odchozí** Pokud bylo x odesláno ve zprávě zašifrované klíčem k a poté přijato jako součást jiné zprávy, potom účastník, který zná klíč k musí být zodpovědný za odeslání této zprávy.
2. **Příchozí** Pokud bylo x odesláno jako součást nějaké zprávy, a poté bylo x přijato ve zprávě zašifrované klíčem k , potom účastník, který zná klíč k musí být zodpovědný za vygenerování této zprávy.

3. **Nevyžadané** Pokud je přijata zašifrovaná zpráva klíčem k , musel ji vygenerovat účastník, který tento klíč zná.

Tento koncept autentizačních testů, je ale příliš obecný a neposkytuje jednotlivým účastníkům dostatečné znalosti o běhu protokolu. Například z protokolu 3.1 nemůže účastník A jasně vyvodit co se stalo. Dešifroval účastník B zprávu a zasílá zpět odpověď? Snaží se s A komunikovat někdo jiný? Je shodou okolností N_a výsledkem komunikace s jiným účastníkem?.

$$\begin{aligned} A \rightarrow * &: \{ |N_a, A| \}_{K_B} \\ * \rightarrow A &: N_a \end{aligned}$$

Pro potřeby tvorby protokolů pomocí kompozice je použito zpřesnění autentizačních testů na testy dohody. Test dohody je takový autentizační test, který splňuje vlastnost dohody, tak jak je dále definována.

Dohoda : Protokol splňuje vlastnost dohody za těchto podmínek: Kdykoliv když iniciátor A dokončí během protokolu kde se domníval, že komunikuje s příjemcem B pak příjemce B prováděl protokol v domnění, že komunikuje s iniciátorem A . B v tomto běhu hrál roli příjemce a A roli iniciátora a každá taková komunikace ze strany A koresponduje s unikátní komunikací na straně B .

Pokud se vrátíme k 3.1 je vidět že běhy protokolů účastníka A a B spolu nekorrespondují. Použitím této podmínky pro jednotlivá primitiva je dosaženo toho, že je možné ze zpráv zjistit kdo prováděl transformaci, ale také za jakým účelem, nebo spíše, pro kterého účastníka je dané transformace určena.

V [5] jsou navrženy následující primitiva. Základem těchto primitiv je rozšíření autentizačních testů o spojovací množiny přenášených dat, \mathcal{B}_x . Tím je dosažena vlastnost dohody a tím se tyto autentizační testy stávají testy dohody. Důkazy této vlastnosti jsou uvedeny v téže publikaci.

$$\begin{aligned} 1: A \rightarrow B &: B, \{ |x, \mathcal{B}_x| \}_{k_{AB}}, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \\ B \rightarrow A &: A, \langle x, \mathcal{B}_x \rangle \end{aligned}$$

$$\begin{aligned} 2: A \rightarrow B &: B, \{ |x, \mathcal{B}_x| \}_{k_{AB}}, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \\ B \rightarrow A &: A, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \end{aligned}$$

$$\begin{aligned} 3: A \rightarrow B &: x, \mathcal{B}_x, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \\ B \rightarrow A &: A, \langle x, \mathcal{B}_x \rangle_{k_{AB}} \end{aligned}$$

Pro asymetrickou kryptografii lze použít podobná primitiva, s rozdílem, že musíme použít dva páry klíčů, (sk_*, vk_*) pro podpis a (ek_*, dk_*) pro šifrování zpráv.

$$\begin{aligned}
1 : A \rightarrow B : B, \{|x, \mathcal{B}_x|\}_{ek_B}, \langle x, \mathcal{B}_x \rangle_{sk_A} \\
B \rightarrow A : A, \langle x, \mathcal{B}_x \rangle \\
2 : A \rightarrow B : B, \{|x, \mathcal{B}_x|\}_{ek_B}, \langle x, \mathcal{B}_x \rangle_{sk_A} \\
B \rightarrow A : A, \langle x, \mathcal{B}_x \rangle_{sk_B} \\
3 : A \rightarrow B : x, \mathcal{B}_x, \langle x, \mathcal{B}_x \rangle_{sk_A} \\
B \rightarrow A : A, \langle x, \mathcal{B}_x \rangle_{sk_B}
\end{aligned}$$

Tímto je ukončena část, která se věnuje podmínkám, které musí platit v rámci jednotlivých primitiv. Následující sekce se zabývá podmínkami, které musí být splněny mezi primitivy, aby bylo možné je kombinovat

3.4.2 Kompozice

Aby bylo možné primitiva komponovat, je nutné aby se jednotlivá primitiva neovlivňovaly. Konkrétně by provedení jednoho primitiva nemělo narušit výsledek, který byl dosažen předchozím primitivem. O takovém výsledku, cíli, primitiva můžeme prohlásit, že je nedestruktivní. Cílem rozumíme autentizaci, nebo přenos dat, která mají zůstat tajemstvím.

Nedestruktivní cíl Pokud běh primitiva p_2 nenaruší předpoklady na kterých závisí cíl g_1 primitiva p_1 , potom můžeme říct, že cíl g_1 je nedestruktivní vzhledem k p_2 , tzn. běh p_2 na něho nemá vliv.

Pokud je cíl g_1 nedestruktivní v protokolu, který vznikl kombinací primitiv p_1 a p_2 , značíme jako $p_1 \otimes p_2$, a zároveň cíl g_2 je nedestruktivní v $p_1 \otimes p_2$, potom můžeme po primitivech prohlásit, že spolu neinterferují.

Zároveň musí být ale splněna vlastnost nekonstruktivity, tzn. že cíl protokolu nesmí být splněný, bez toho aby se uskutečnila platná událost, která ho má učinit platným. Pro demonstraci můžeme použít následující příklad protokolu:

$$\begin{aligned}
A \rightarrow * : \{|N_a, A|\}_{K_B} \\
B \rightarrow * : \{|N_a, A|\}_{K_C} \\
* \rightarrow A : N_a
\end{aligned}$$

Ačkoliv se neuskutečnila událost $B \rightarrow A : N_a$, autentizace mezi A a B je platná. Tento protokol nesplňuje vlastnost nekonstruktivity.

V [5] je dokázáno, že kompozice primitiv splňuje tyto vlastnosti pokud jsou dodrženy následující podmínky:

- Primitiva je regulární
- Všechny ochranné domény jsou I -kompatibilní.

- Žádná nonce není použita pro autentizaci dvou různých účastníků

Pouhé nenarušení cílů ostatních primitiv, ale není dostatečné pro korektní kompozici. Pro korektní kompozici je nutné, aby se všechny všechny cíle proběhly v rámci jednoho sezení. Toho se dá dosáhnout přidáním jednoznačného identifikátoru sezení, případně prvkem zvaným ‘binder’. Nejčastěji se jako ‘binder’ používá hash přenášených proměnných a jejich spojovacích množin.

3.4.3 Rozšíření metody

Metodu lze rozšířit o nová primitiva, pokud primitiva splňují podmínky z minulé části a navíc jsou to testy dohody.

3.5 Závěr

Kapitola představila několik různých přístupů k návrhu bezpečnostních protokolů. V následující části budou představen speciální typ bezpečnostních protokolů, Zero knowledge protokoly.

Kapitola 4

Zero knowledge protokoly

4.1 Úvod

Zero knowledge protokoly [15] [17] byly vytvořeny z důvodu, že u klasických bezpečnostních protokolů dochází k vyzrazení informací při každém běhu protokolu, které pokud se jich nashromáždí dostatečné množství mohou být využity například pro krádež identity. Při běhu zero knowledge (dále bude používána zkratka ZK) protokolu nedochází při běhu protokolu k úniku žádných informací.

Při běhu ZK protokolu se snaží účastník prokázat znalost určité informace druhému účastníkovi. Pro účastníky se používají zkratky P , z anglického prover, pro toho který se snaží ukázat že zná informace a V , z anglického verifier, pro toho který se snaží tuto znalost ověřit. Hlavní myšlenkou je možnost provést důkaz znalostí informací, bez toho aniž by se V dozvěděl cokoliv jiného než, že důkaz je pravdivý. Nejedná se o důkaz v matematickém smyslu, ale jedná se o dynamický proces mezi dvěma účastníky.

Aby bylo možné označit protokol za ZK, je nutné aby splňoval následující kritéria:

- V se nemůže z běhu protokolu dozvědět nic dozvědět
- P nemůže obelhat V
Pokud P tajemství nebo informace nezná, potom nesmí, nebo musí mít velmi malou pravděpodobnost, být schopný tuto znalost prokázat
- V nemůže obelhat P — I kdyby V podváděl a nedržel se striktně protokolu, nesmí být schopný se dozvědět cokoliv jiného, než že P zná nebo nezná informace.
- V nemůže předstírat nějaké třetí straně, že je P

Podobně jako současná moderní kryptografii využívají ZK protokoly pro udržení tajemství problémy jako je rozklad velkých čísel na prvočísla nebo knapsack problém. Oproti moderní kryptografii jsou ale ZK protokoly méně časově efektivní, tzn. jejich běh trvá delší dobu, což může být nevýhodné u RT aplikací, na druhou stranu ZK nejsou tak výpočetně náročné a lze je využít například ve vestavěných zařízeních, SMART kartách atd.

Nevýhodou ZK protokolů je jejich využitelnost, z důvodu, že nepřenáší žádné informace při běhu, nelze je použít pro přenos zpráv nebo distribuci klíčů. Lze je ale použít pro autentizaci uživatelů a objevují se případy, kde je možné je využít i pro potvrzení validity zadaných dat, bez toho aniž by účastník, který vyžaduje potvrzení, data znal.

4.2 Interaktivní dokazovací systém

ZK protokol lze chápat jako instanci interaktivních dokazovacích systémů [8], což jsou systémy ve kterých si P a V vyměňují zprávy ve formě výzva/odpověď v závislosti, které závisí na náhodě. Důkaz, který vznikne v tomto systému má pouze určitou šanci být pravdivý. Aby bylo možné tyto systémy označit za důkazy znalostí být platné následující podmínky, které jsou zobecněním předchozích kritérií:

- **Úplnost**

Důkaz znalosti je úplný, pokud čestný P a čestný V provádí protokol, potom uspějí s vysokou pravděpodobností.

- **Spolehlivost**

Důkaz znalostí je spolehlivý pokud existuje algoritmus A takový, že pokud nečestný účastník předstírající, že je P , může s nemalou pravděpodobností uspět v dokázání znalosti, potom A může být použito pro získání této znalosti. Pokud je tato znalost použita v dalších bězích protokolu má vysokou pravděpodobnost uspět. Neboli, pokud může někdo předstírat že je P a uspět, potom musí vlastnit znalost, která je ekvivalentní se znalostí P .

ZK důkaz(protokol) musí mít navíc ZK vlastnost, že všechny informace které si mezi sebou P a V pošlou, může získat V i bez pomoci P . Neboli, existuje nějaká procedura S , která může generovat ‘falešné’ běhy protokolu, které jsou nerozeznatelné od skutečných. Taková procedura se nazývá simulátor.

4.3 Další ZK vlastnosti

Pokud ZK vlastnost splňuje definici z předchozí části tak je označována jako perfektní ZK. Existují, ale ještě dvě méně striktní definice, které specifikují následující ZK vlastnosti:

- **Výpočetní ZK**

Existuje simulátor takový, že ‘falešné’ důkazy jsou výpočetně, s časovou složitostí ve třídě $O(P)$, nerozeznatelné od pravých.

- **Statistická ZK**

Existuje simulátor takový, že existuje nepatrná odchylka mezi pravděpodobnostní distribucí pravých a ‘falešných’ důkazů.

4.4 Ukázka - Feige-Fiat-Shamir důkaz identity

Důkaz identity Feige-Fiat-Shamir [7] je klasickým příkladem ZK protokolů a bude zde uveden pro aby čtenář získal konkrétnější představu o ZK protokolech. Důkaz je založen na problému nalezení odmocniny čísla v modulu n , kde je číslo, které vzniklo součinem dvou velkých prvočísel.

V příkladu jsou použity jako doposud zkratky P a V pro účastníky komunikace.

Příprava : Důvěryhodná třetí strana, nadále pouze T , vygeneruje číslo n , takové, že je součinem dvou prvočísel. T poté vygeneruje pár klíčů pro P . Veřejným klíčem je v pro které kvadratickým zbytkem v modulu n ($x^2 \equiv v \pmod{n}$) a soukromým klíčem bude nejmenší číslo s pro které platí $s = \sqrt{\frac{1}{v}} \pmod{n}$.

Běh protokolu :

1. P vygeneruje náhodné číslo r , takové že $GCD(r, n) = 1$
2. P vypočítá $x = r^2 \pmod n$ a x odešle V
3. V vygeneruje náhodný bit b a odešle ho P
4. Pokud $b = 0$, P pošle zpět r
jinak odešle $y = rs \pmod n$
5. Pokud V odeslal $b = 0$ potvrdí si, že $x \equiv r^2 \pmod n$ a že tedy P zná $\sqrt{x} = r$
nebo pokud $b = 1$ potvrdí si, že $x \equiv y^2 v \pmod n$ a že tedy P zná $\sqrt{\frac{x}{v}}$

Iterace : Protokol běží tolikrát dokud V není přesvědčen, že P opravdu zná s , popřípadě nezná. V každém běhu je ale nutné zajistit aby r bylo různé!

Přijmutí/odmítnutí : Po dostatečném množství iterací V odešle rozhodnutí o přijmutí nebo odmítnutí důkazu účastníkovi P .

V tradiční zápisu by mohl Feige-Fiat-Shamir důkaz identity vypadat takto:

$A \rightarrow B : Ncommit$

$B \rightarrow B : Nchallenge$

$A \rightarrow B : f(Ncommit, Nchallenge)$

$B \rightarrow B : Naccept/Nreject$

Můžeme vidět, že P může podvádět pokud s nezná, ale může se předem připravit pouze na jednu ze dvou variant bitu b , tzn. bude schopný správně odpovědět pouze na jednu z těchto možností, po několika pokusech je tedy pravděpodobné, že bude odhalen.

4.5 Závěr

Náplní kapitoly bylo seznámit čtenáře s zero knowledge protokoly jejich specifiky a využitím. V následující kapitole budou prozkoumány možnosti návrhu těchto protokolů pomocí automatických metod.

Kapitola 5

Návrh ZK protokolů

5.1 Úvod

Tradiční zápis Feige-Fiat-Shamir [7], tak jak byl uveden v minulé kapitole, lze obecně použít pro zápis jakéhokoliv jiného ZK protokolu. ZK protokol vždy bude mít toto pořadí zpráv, tzn. závazek, výzva, odpověď a konečné potvrzení nebo zamítnutí.

Hlavní úskalí návrhu samostatných ZK protokolů leží tedy spíše v oblasti vytváření matematických funkcí, které je možné v tomto typu protokolů použít. Tato práce by se ale neměla zabývat touto částí návrhu ZK, ale spíše otázkou jakým způsobem vytvářet komplexnější protokoly, ve kterých jsou použity ZK protokoly jako subprotokoly pro identifikaci jednotlivých účastníků.

Automatické metody uvedené v této práci nebyli původně uzpůsobené, pro návrh ZK protokolů, proto bude nutné je alespoň částečně upravit.

5.2 Vstupní a výstupní znalosti

Požadavky na ZK protokol se dají zapsat následujícím způsobem:

- Vstupní znalosti

$$\begin{aligned} A &= \{Ncommit, f(x, y)\} \\ B &= \{Naccept, Nchallenge\} \end{aligned}$$

- Výstupní znalosti

$$\begin{aligned} A &= \{Nchallenge, Naccept\} \\ B &= \{Ncommit, f(Ncommit, Nchallenge)\} \end{aligned}$$

Bohužel v těchto požadavcích nelze vyjádřit pořadí zpráv, že *Naccept* má strana *B* odeslat, až po přijmutí $f(Ncommit, Nchallenge)$ od strany *A*. Do požadavků tuto informaci musíme uvést jako poznámku, popřípadě rovnou do poznámky uvést, že dané subjekty se mají autentizovat pomocí ZK protokolů.

Nebo můžeme vytvořit nové výstupní znalosti:

ZKidentify(*x*): subjekt, který má tuto znalost ve výstupních znalostech, autentizoval pomocí ZK protokolů subjekt *x* v tomto běhu protokolu

ZKaccept(x) subjekt, který má tuto znalost ve výstupních znalostech, byl autentizován pomocí ZK protokolu subjektem x

Výstupní znalosti mohou poté vypadat následovně:

$$A = \{ZKaccept(B)\}$$

$$B = \{ZKidentify(A)\}$$

V případě pokud není protokol specifikován pomocí vstupních a výstupních znalostí, lze uvést, že cílem protokolu je provést autentizaci pomocí ZK protokolů.

5.3 Úpravy metod

5.3.1 Evoluční návrh

Při evolučním návrhu máme možnost upravit gramatiku, pomocí které generujeme protokoly, tak že do ní přidáme nonterminál $\langle ZK \rangle$, přepisovací pravidlo ve tvaru:

$$\langle \text{event} \rangle \rightarrow \langle \mathbf{ZK} \rangle$$

a přepisovací pravidla, která z nonterminálu $\langle ZK \rangle$ vytvoří posloupnost událostí, jejímž účelem bude autentizace dvou subjektů pomocí ZK protokolů.

Pokud by tento sled událostí byl ohodnocen větší hodnotou fitness, bylo by možné, že by tento sled algoritmus vyevolvoval sám, ale díky úpravám v gramatice můžeme zmenšit prohledávací prostor algoritmu.

5.3.2 Kompoziční návrh

V kompozičním návrhu [5] je možné vytvořit nové primitivum, pomocí kterého se nedosáhne dohody nad nějakými daty, ale dosáhneme autentizace(dohoda nad identitou uživatele) pomocí ZK protokolu.

Ve zprávách, které si oba účastníci odešlou nebudou obsažena žádná dlouhodobá tajemství a žádná nonce nebude použita pro více než jednoho účastníka(lze vytvořit unikátní commit, challenge i accept/reject nonce pro každého účastníka, který se bude chtít autentizovat pomocí ZK protokolů). Tím zajistíme splnění požadavků, kladená na nová primitiva

5.3.3 Návrh pomocí logiky

Do logiky je nutné přidat inferenční (a následně i syntetická) pravidla, která budou vyjadřovat možnou autentizaci pomocí ZK protokolů.

5.4 Závěr

V kapitole byly navrženy způsoby úpravy dříve popsanych automatických metod, tak aby mohly být využity ZK protokoly jako součást protokolů, které tyto metody navrhuji. Další část se bude zabývat samotnou implementací jedné z metod, konkrétně návrhem s využitím kompozice.

Kapitola 6

Implementace

6.1 Úvod

Pro implementaci byla vybrána kompoziční metoda [5] z důvodu snadné reprezentace ZK protokolu/autentizace jako jednoho primitiva. Evoluční algoritmus se zdál jako druhá nejlepší volba, ale úpravou gramatiky tak, že by se z jednoho nonterminálu generoval celý ZK subprotokol, by bylo dosaženo podobného výsledku jako použití primitiva v kompoziční metodě. A generování protokolů pomocí evolučních algoritmů bylo již zpracováno v [13]. Návrh pomocí logiky by mohl být poměrně komplikovaný, nově vytvořené inferenční a syntetická pravidla spojky by měli pouze rozšiřovat logiku ne ji ovlivňovat, a dokázání této vlastnosti nemuselo být jednoduché.

Jako implementační jazyk byl zvolen Prolog, z důvodu vhodného výpočetního modelu a dobré vyjadřovací schopnosti. Konkrétně byla implementace provedena ve variantě SWI-Prolog.

Nejprve bude popsáno jakým způsobem je definováno prostředí protokolu, účastníci, kanály, počáteční znalosti, a cíle protokolu. Tomuto tématu bude věnovaná část ‘Popis protokolu’. Následovat bude popis formátu primitiv, jakým způsobem jsou získávána konkrétní primitiva a přehled současně implementovaných primitiv. Poté bude představen způsob tvorby protokolu, jakým způsobem jsou primitiva vybírána pro splnění zadaných cílů.

V poslední části bude výstup implementované metody porovnán s analytickou metodou.

6.2 Popis protokolu

Popis protokolu byl zvolen tak, aby bylo možné ho zapsat co nejjednodušším způsobem a byl dostatečně srozumitelný. Vzhledem k implementaci v Prologu, byl popis protokolu zvolen jako seznam klauzulí, které definují počáteční stav databáze programu. Bohužel některé klauzule použité v definici jsou používány také v hlavním programu a z tohoto důvodu je nutné tyto klauzule definovat jako dynamické. Definice dynamické klauzule se provádí pomocí pravidla `dynamic`, která jako parametr přijímá název dynamické klauzule a její aritu, tzn. počet parametrů, které klauzule má. Na počátku každého definičního souboru musí být dynamicky definovány alespoň tyto dvě klauzule:

```
:-dynamic(actor/1).  
:-dynamic(bindGroup/2).
```

Ostatní klauzule lze dodefinovat jako dynamické dle potřeby.

6.2.1 Klauzule používané v definici

V této části budou představeny jednotlivé klauzule, které se mohou objevit v definici protokolu. Každá klauzule musí být v souboru na vlastním řádku a ukončena tečkou.

actor(a) Definuje **a** jako účastníka protokolu. Nikdo jiný než definovaní účastníci se nemohou účastnit výměny zpráv. V protokolu je definován speciální účastník **spy**, který se nemusí explicitně definovat ve specifikaci protokolu. Tohoto speciálního účastníka lze využít pro testování přenosu nešifrovaných zpráv.

sever(a) Upřesnění role účastníka **a**. Účastník má v protokolu roli důvěryhodného serveru. Role serveru je důležitá v některých primitivech.

initiator(a) Upřesnění role účastníka. Účastník **a** začíná běh protokolu, posílá první zprávu. **Musí být definováno!**

key(a, b, type, number) Definuje klíč, který je možné použít v rámci protokolu. Klíč je určen pro komunikaci mezi účastníky **a** a **b**. Klíč může nabývat 3. typů

secret Symetrický tajný klíč - účastníci **a** a **b** jsou různí

public Asymetrický veřejný klíč - účastníci **a** a **b** jsou shodní, jedná se o veřejný klíč daného účastníka

private Asymetrický privátní klíč - účastníci **a** a **b** jsou shodní, jedná se o veřejný klíč daného účastníka

Program nekontroluje zda k existuje platná dvojice asymetrických klíčů. Pouze při pokusu použít asymetrický klíč nebude schopný najít reverzní klíč a pravděpodobně nenajde žádný platný protokol. Hodnota **number** určuje číslo klíče, může existovat více klíčů pro 2 účastníky, v budoucnu je možné tuto vlastnost použít pro generování nových klíčů ze starších.

channel(a, b) Definuje existenci jednosměrného kanálu mezi účastníky **a** a **b**. Pro definici oboustranného kanálu je nutné definovat i kanál mezi účastníky **b** a **a**. Současná primitiva pracují pouze s oboustrannými kanály.

knows(a, x) Definice, že účastník **a** zná informaci **x**. Není nutné specifikovat, že účastníci znají ostatní účastníky, toto se považuje za automatické.

nonce(n) Způsob zápisu nonce, **n** by mělo být jedinečné číslo(způsob generování unikátních čísel bude popsán dále v textu). Není nutně explicitně definovat nonce hodnoty ve specifikaci protokolu, ty se v případě potřeby vytváří uvnitř primitiv. Pouze pokud chceme aby účastník znal určitou nonce je nutné toto definovat pomocí klauzule typu **knows(a, x)**.

bindGroup(x, List) Definice spojovací množiny pro znalost **x**. **List** je seznamem účastníků, kteří patří do této spojovací skupiny. Seznam má formát jako typický seznam v Prologu, ohraničen hranatými závorkami a jednotlivé prvky jsou odděleny čárkou např. [a, b, c]. Je **nutné** specifikovat spojovací pro **každou** znalost. Klíče se dají považovat za výjimku. Pokud pro klíč není specifikovaná spojovací množina je definována následujícím způsobem

- Symetricky tajný klíč - Účastníci v definice klíče + všichni kdo ho už znají

- Asymetrický veřejný klíč - Všichni účastníci protokolu
- Asymetrický soukromý klíč - Účastník v definici klíče

longterm(a) Znalost **a** je definována jako dlouhodobé tajemství. Z definice regulárních primitiv vyplývá, že takové znalosti nemohou být prvkem zpráv. Nejčastěji se používá pro dlouhodobé klíče.

goal(g) Definuje cíle protokolu. Jako cíl **b** je možné použít jakýkoliv cíl, který jsou použita primitiva schopná splnit. Typicky je možné použít následující cíle

- `knows(a, x)` - Znalost nějaké informace
- `auth(a, b, x)` - Autentizace účastníků **a** a **b** na základě informace **x**.
- `zkAuth(a, b)` - Autentizace pomocí ZK protokolů.

forbidden(g) Definuje zakázané stavy protokolu. Typicky obsahuje klauzule `knows`, pro znalosti, které by se ostatní účastníci neměli dozvědět.

6.2.2 Generování unikátních čísel

Při návrhu protokolu se často používají unikátní, jednoznačná čísla. Tuto funkčnost byla v definici protokolu zajištěna pomocí čítače unikátních čísel. Ten je definován pomocí pravidla ve tvaru `counterInc(unique, N)`, kde **N** je nové unikátní číslo. Bohužel vzhledem k použitému způsobu definice prostředí není generování čísel na první pohled zřejmé. Pouhou definici:

```
counterInc(unique, N).
knows(a, nonce(N)).
```

případně

```
counterInc(unique, N), knows(a, nonce(N)).
```

by bylo docíleno pouze redefinice pravidla `'` nebo `counterInc`. Pro správné generování unikátních čísel je nutné specifikovat pravidlo bez hlavičky, takové pravidlo se provede vždy při načtení souboru, ve kterém se vygeneruje nové unikátní číslo, které se použije v `nonce`, a dále je definována například znalost této `nonce`, popřípadě cíl závisící na této `nonce`. Následující příklad demonstruje jakým způsobem správně generovat unikátní čísla a způsob jejich následného využití.

```
:-
  counterInc(unique, N),
  bassert(nonce(N)),
  bassert(bindGroup(nonce(N), [a, b])),
  bassert(knows(a, nonce(N))),
  bassert(goal(knows(b, nonce(N))))).
```

Pravidlo `bassert(x)` vkládá daný term do databáze, jedná se o úpravu `assert` tak aby byl podporován `backtracking`, i když v případě definice protokolu je možné použít klasický `assert`.

V případě takto definovaných cílů a znalostí je nutné uvést, že se jedná o dynamické klauzule na počátku souboru, podobně jako definice.

6.2.3 Příklad definice protokolu

Na následujícím příkladu je vidět jednoduchá definice protokolu, jehož cílem je provést autentizaci uživatelů **a** a **b** na základě znalosti **x**.

```
:-dynamic(actor/1).
:-dynamic(bindGroup/2).

actor(a).
actor(b).

channel(a, b).
channel(b, a).

key(a, b, secret, 0).

knows(a, x).

knows(a, key(a, b, secret, 0)).
knows(b, key(a, b, secret, 0)).

initiator(a).

goal(auth(a, b, x)).
```

6.3 Popis primitiv

Vzhledem k tomu, že se primitiva používají pro různé účastníky a různé znalosti, bylo nutné vytvořit způsob jakým vytvořit konkrétní primitiva z určité předlohy. Byl proto zvolen univerzální tvar primitiv:

Název(Realizace, Primitivum)

Pod realizací rozumíme trojici, modelovanou jako seznam, dvou účastníků a informace/dat. Na základě této realizace je z obecného zápisu primitiva vygenerováno konkrétní, kde roli iniciátora a respondenta hrají účastníci z realizace. V některých případech je možné aby respondent a iniciátor byli stejný účastník, např. při vytváření nových dat.

Konkrétní primitivum obsahuje informace potřebné pro jeho začlenění do protokolu, konkrétně obsahuje 6 typů informací, kde každá je reprezentována jedním seznamem.

Předpoklady Definují, které fakta musí být pravdivá aby bylo možné dané primitivum provést. Nejčastěji obsahuje informace co který účastník znát má a co nemá. Všechny takto zadané podmínky by měli být nějakým způsobem splnitelné, tzn. je možné změnit jejich platnost.

Podmínky, které musí být platné před použitím primitiva a nelze změnit jejich platnost, např. existence účastníků, existence kanálů mezi účastníky, je vhodné uvést do těla pravidla generování primitiva a ne do seznamu předpokladů. S výjimkou testování zda účastník je členem spojovací množiny informace. Definice spojovací množiny se

totiž provádí při vytvoření informace a při inicializace primitiva z realizace, informace ještě nemusí existovat.

Preaktivní účastníci Definuje účastníky, kteří musí být aktivní před během primitiva. Ve většině případu tento seznam obsahuje pouze iniciátora primitiva. Aktivitou rozumíme, že účastník nedávno, přijal nebo zasílal nějaké informace.

Akce Definují jednotlivé zprávy, kteří si mezi sebou účastníci vymění. Akce mají formát `send(a, b, Data)`, kde **a** je odesílatel a **b** je příjemce a **Data** jsou seznamem prvků obsažených ve zprávě. V současné verzi mohou být prvky zprávy následující klauzule:

- Účastníci
- Znalosti
- Funkce
- `conf(klíč, Data)` Data zašifrovaná klíčem. Klíč je nutné uvést. Data jsou v tomto případě také seznamem prvků.
- `hash(klíč, Data)` Hash dat. Klíč je v tomto případě volitelný. Pokud nechceme použít klíč, uvedeme místo něj prázdný řetězec `''`. Data jsou opět seznamem prvků.

Podmínky V rámci vytváření primitiv rozlišujeme dva druhy podmínek. Podmínky, které musí být platné při plnění předpokladu primitiva a podmínky, které musí být platné po provedení primitiva. Každou z těchto skupin dále dělíme na podmínky týkající se realizaci a ostatní. Hlavní úlohou podmínek realizací je zabránění tvoření nekonečných smyček. Do ostatních podmínek je možné uvést, které klade primitivum na další běh protokolu.

Postaktivní účastníci Jsou účastníci, kteří jsou aktivní po provedení primitiva. Ve většině případů obsahuje iniciátora i respondenta.

Důsledky Definuje změny v databázi po provedení primitiva. Většinou odráží to, že iniciátor, případně respondent se dozvěděli novou informací nebo že proběhla autentizace mezi dvěma účastníky. V případě odeslání nešifrované zprávy je vhodné zde také uvést, že se tyto informace dozvěděl účastník **spy**.

Tyto informace jsou dostatečné pro generování protokolu pomocí primitiv.

V současné době jsou implementována primitiva pro odeslání informace v zašifrované a nezašifrované formě. Tyto primitiva také poskytují autentizaci uživatelů. Dále primitivum pro vyžádání informace od respondenta a vyžádání zaslání informace pro celou spojovací množinu dané informace, což je vhodné pro distribuci klíčů. Posledním primitivem je autentizace pomocí zero knowledge protokolů.

6.3.1 Příklad primitiva

Pro demonstraci jak vypadá konkrétní definice primitiva bude použito primitivum, které realizuje odeslání zašifrované informace uživateli.

```
1: prim_1([A, B, X],  
         [PreCond, PreActive, Actions, Constrains, PostActive, PostCond]) :-
```

```

2:  A \= spy, B \= spy,
3:  actor(A), actor(B), A \= B,
4:  channel(A, B), channel(B, A),
5:  not(longterm(X)),
6:  getKeysPair(A, B, [EncKey, DecKey, SigKey, VerKey]),
7:  list_to_set([knows(A, X), knows(A, EncKey), knows(A, SigKey),
               knows(B, DecKey), knows(B, VerKey), not(knows(B, X)),
               (bindGroup(X, BGX), member(B, BGX))],
               PreCond),
8:  PreActive = [A],
9:  Actions = [send(A, B, [B, conf(EncKey, [X, BGX]), hash(SigKey, [X, BGX])]),
              send(B, A, [A, hash(, ' , [X, BGX])])],
10: Constrains = [[], [[B, _, X], [_, B, X]], [], []],
11: PostActive = [A, B],
12: PostCond = [knows(B, X), auth(A, B, X)].

```

První řádek je definicí primitiva, je vidět že primitivum má dva parametry tak jak byli definovány v předchozí kapitole, tedy prvním parametrem je realizace a druhý slouží jako návratová hodnota inicializovaného primitiva.

Na následujících třech řádcích, 2 - 4, jsou testovány podmínky, které musí být platné a jejichž platnost nelze změnit, tedy že první dva parametry realizace jsou existující účastníci a ani jeden z nich není účastníkem **spy**, metoda nedovoluje přímou komunikaci s tímto účastníkem, tento účastník pouze odposlouchává zprávy odeslané v otevřené podobě.

Pátý řádek testuje zda informace, která se má mezi dvěma účastníky přenášet není definována jako dlouhodobé tajemství. Jak už bylo v řečeno v části zabývající se popisem jednotlivých klauzulí v definičním souboru, takovéto informace nesmí být prvky zasílaných zpráv.

Na řádku 6 jsou použitím pravidla `getKeysPair` získány klíče potřebné pro komunikaci mezi dvěma účastníky primitiva. Je nutné získat 2 páry klíčů, jeden pro šifrování a druhý pro podpis. Následující řádek, číslo 7, odstraňuje redundantní cíle, protože je možné se pro některé z těchto klíčů používá klíč stejný. V případě symetrické kryptografie je možné že všechny 4 klíče jsou stejné.

Řádek 8 definuje účastníka, který musí být aktivní, před použitím primitiva. Podobně řádek 11 definuje účastníky, kteří mohou být aktivní po provedení primitiva.

Seznam `Actions`, na řádku 9, definuje zprávy které si jednotliví účastníci vymění mezi sebou. V tradičním zápisu by bylo možné toto primitivum zapsat tímto způsobem:

$$\begin{aligned}
A \rightarrow B : & \quad B, \{X, \mathcal{B}_X\}_{EncKey}, \langle X, \mathcal{B}_X \rangle_{SigKey} \\
B \rightarrow A : & \quad A, \langle X, \mathcal{B}_X \rangle
\end{aligned}$$

Na řádku 12 jsou uvedeny důsledky, které má provedení primitiva na současný stav databáze. Je vidět, že účastník **B** se dozví novou informaci **X**. Zároveň se provede jednosměrná autentizace mezi účastníky **A** a **B**. Konkrétně **A** autentizuje účastníka **B**. Případně by bylo možné uvést, že účastník **spy** zná obě tyto zprávy, v současné implementaci slouží účastník **spy** pro kontrolu přenosu nezašifrovaných zpráv. I když tuto funkcionalitu lze rozšířit, i když větší relevanci má spíše při verifikaci protokolu, než při jeho návrhu.

Řádek 11 obsahuje podmínky na realizace, tak aby nevznikali při vytváření protokolu nekonečné smyčky, kdy by se používalo stále stejné primitivum pro splnění určitých cílů,

popřípadě by vznikali různé závislosti mezi cíli a předpoklady, které by vedly ke vzniku smyček. Toto téma bude podrobněji rozebráno v dalších částech.

6.3.2 Vytváření nových znalostí

Vytváření nových zpráv nebo znalostí ze současných znalostí účastníka není v současné době implementováno. Nové znalosti se mohou tvořit pouze transformací některých ze znalostí. Tyto transformace je možné provést právě jednou. Opakované transformace se neuskuteční, protože při dotazu, zda účastník zná tuto novou znalost, poté vždy uspěje. Pokud by bylo povoleno vytváření jakýchkoliv nových znalostí je pravděpodobné, že by se vyhledávání mohlo dostat do nekonečné smyčky.

Nekonečné smyčky a způsob jejich řešení, je ale spíše tématem další části, která se zabývá jakým způsobem je prohledáván stavový prostor řešení a jak bylo zabráněno tomu aby vznikali nekonečné smyčky i když při jiných příležitostech jako je vytváření nových zpráv.

6.4 Způsob prohledávání stavového prostoru

Program se snaží postupně plnit zadané cíle. Jako prvotní cíle jsou zvoleny cíle, které jsou definovány ve specifikaci protokolu. Program se snaží současné cíle postupně splnit pomocí nějakého primitiva. Pokud je nalezeno primitivum, které plní daný cíl, jeho předpoklady se stávají novými současnými cíli. Po splnění všech předpokladů je primitivum přidáno do protokolu a znalosti jsou přidány do databáze Prologu, tak jak jsou definovány v seznamu důsledku primitiva. Program pokračuje tímto způsobem dokud nejsou splněny všechny cíle. Primitiva, ze kterých program vybírá vhodné, jsou definovány při spuštění programu. Všechny primitiva jsou nedestruktivní, proto není třeba zkoumat zda se po přidání nějakého primitiva změnili nějaké předpoklady na kterých záviselo některé z primitiv předchozích.

Na primitiva jsou, ale kladeny v rámci vytváření protokolu i další dodatečné podmínky. Žádná z omezujících podmínek nesmí být v žádném stavu protokolu platná. Pokud je nalezeno primitivum, které splňuje určitý cíl, ale jeho provedením se stane nějaká z omezujících podmínek platnou, je nutné zvolit primitivum jiné.

Primitiva také na sebe musí logicky navazovat, k tomuto účelu slouží seznamy preaktivních a postaktivních uživatelů. Preaktivní účastník, každého primitiva musí také patřit do množiny postaktivních účastníků minulého primitiva. V případě, že se jedná o první primitivum v protokolu, musí být preaktivní účastník definován jako iniciátor komunikace ve specifikaci protokolu. Pokud nejsou splněny tyto podmínky návaznosti, tak generovaný protokol není platný.

Prohledávání prostoru [6] nevyužívá žádné náhodné proměnné a je tedy deterministické.

V minulých částech bylo otevřeno téma vzniku nekonečných smyček při vytváření protokolu. Pokud by existovalo zadání protokolu, kde by část účastníků tvořilo kružnici, definovanou jako v teorii grafu, kde účastníci jsou uzly a kanály orientované hrany, je možné že by nastal stav, kdy by se účastníci na kružnici pokoušeli zjistit nějakou informaci od ostatních účastníků na kružnici i v případě, že by tuto informaci měl nějaký účastník, který na kružnici neleží. Vytvořil by se tak nekonečný sled dotazů, na tu stejnou informaci.

Program se tomuto chování snaží zabránit pomocí omezení, které jsou kladeny na realizace. Každé primitivum definuje omezující podmínky, které musí platit při plnění jeho předpokladů. Pokud bylo zvoleno primitivum, které má zajistit aby se účastník **a** dozvěděl informaci **x**, je jisté že by **a** tuto informaci neměl znát před provedením tohoto primitiva,

tn. by se neměl vyskytovat jako příjemce v realizacích primitiv s informací **x** a také by se tuto informaci neměl pokoušet distribuovat dál. V omezujících podmínkách takového primitiva budou tedy realizace `[a, _, x]` a `[_, a, x]`. Podtržítka je v tomto případě zástupný znak pro libovolnou proměnou, v tomto případě libovolný účastník.

Nyní bude představen způsob spuštění programu a formát výstupu.

6.5 Spuštění programu a výstup

Program nemá žádné speciální rozhraní a využívá možnosti interpretu Prologu. Pro spuštění programu je tedy nutné načíst soubor s hlavním programem pomocí klauzule `consult`, ta jako jediný parametr přijímá cestu k souboru v jednoduchých uvozovkách. Případně v případě OS Windows stačí dvojklik na soubor.

Po načtení hlavního programu je nutné inicializovat čítač unikátních čísel. To se provede klauzulí `prepareDB.`, která se navíc stará o vymazání starých znalostí z databáze, v případě že již byl nějaký protokol dříve generován. Po inicializaci databáze je nutné buď nadefinovat protokol, popřípadě načíst soubor s definicí protokolu pomocí výše uvedené klauzule `consult`.

Provedením těchto je program připraven ke generování protokolů. Generování je spuštěno příkazem `createProtocol(Ps, C, FP)`. Jednotlivé parametry mají následující význam:

- **Ps** Seznam názvů primitiv, které se mohou použít ke generování protokolu
- **C** Dvojice seznamů dodatečných podmínek
- **FP** Finální protokol

Interpret Prologu po vygenerování protokolu vyzve uživatele, zda chce hledat další možný protokol nebo jsme se současným řešením spokojeni. Protokol je potvrzen stlačením **Enter** nebo znaku `.`(tečka). V případě, že uživatel chce vygenerovat další z možných protokolů stlačí klávesu `;`(středník). V případě že interpret nemohl nalézt žádné další řešení vypíše na standardní výstup `false`.

Takto vygenerované protokoly jsou ale v dost nečitelné formě. O převod do čitelné podoby a následně i o tisk se stará pravidlo `printProtocol`, to jako jediný parametr přijímá vygenerovaný protokol. Tisk je proveden na standardní výstup. Formát výstupu je podobný je podobný standardnímu zápisu protokolů s přihlédnutím na konvence zápisu šifrovaných a hashovaných zpráv tak jak jsou zapsány v [5]. Následující kód ukazuje jak prakticky spustit generování protokolu.

```
prepareDB,  
consult(dbFile),  
createProtocol([, prim_1, prim_send, prim_createRequest], [[], []], P),  
printProtocol(P).
```

V současné době je implementována pouze jedna z optimalizací výstupu. Konkrétně se jedná o spojování zpráv pokud mají obě stejného odesilatele a příjemce a následují v protokolu za sebou. Optimalizace pracuje pouze se seznamem akcí, ten lze získat z protokolu aplikováním klauzule `getActionsAll`, samotná optimalizace se pak provede klauzulí `mergeActions`, která jako první parametr přijímá seznam akcí a druhý parametr je optimalizovaný seznam akcí. Demonstrace použití této optimalizace je uvedena níže.

```

prepareDB,
consult(dbFile),
createProtocol([], prim_1, prim_send, prim_createRequest), [[], []], P),
getActionsAll(P, As),
mergeActions(As, MAs),
printActions(MAs).

```

Aplikace také nabízí vyhledání všech nejkratších možných protokolů, které splňují danou definici. K tomu slouží pravidlo `getShortestProtocols`, to má stejné parametry jako `createProtocol`. V pravidlu je využita optimalizace slučování zpráv. Problém s tímto pravidlem je, že ačkoliv jedno řešení implementace vyhledá poměrně rychle, vyhledání všech možných řešení je poměrně časově náročné.

V poslední části je provedeno srovnání implementované metody s metodou analytickou na vybraném protokolu.

6.6 Srovnání s analytickým použitím metody

V této části bude na zadání specifikace protokolu demonstrováno, zda analytická metoda i metoda implementovaná produkují podobné výsledky. Současně část poslouží jako příklad výstupu programu. Bude vytvořen scénář jehož cílem bude distribuce klíčů mezi dvěma subjekty s pomocí distribučního serveru. Specifikace pro aplikaci bude mít následující tvar.

```

actor(a).
actor(b).
actor(s).
server(s).

```

```

channel(a, b).
channel(b, a).

```

```

channel(a, s).
channel(s, a).

```

```

channel(b, s).
channel(s, b).

```

```

key(a, b, secret, 0).
key(a, s, secret, 0).
key(b, s, secret, 0).

```

```

knows(s, key(a, b, secret, 0)).
knows(s, key(a, s, secret, 0)).
knows(s, key(b, s, secret, 0)).
knows(a, key(a, s, secret, 0)).
knows(b, key(b, s, secret, 0)).

```

```

initiator(a).

```

```

goal(knows(a, key(a, b, secret, 0))).
goal(knows(b, key(a, b, secret, 0))).

forbidden(knows(s, x)).
forbidden(knows(spy, x)).
forbidden(knows(spy, key(a, b, secret, 0))).
forbidden(knows(spy, key(a, s, secret, 0))).
forbidden(knows(spy, key(b, s, secret, 0))).

```

Pokud budeme postupovat podle analytické metody, jedná se vlastně o kombinaci dvou primitiv, kde A a B odešlou požadavek serveru, který jim poté zašle klíče. S tím, že by měl odpovědět po té co od obou účastníků přijme první zprávu. Primitiva vypadají takto:

$$\begin{aligned}
A \rightarrow S &: \mathcal{B}_x, \{|a, \mathcal{B}_x|\}_{k_{AS}}, \langle a, \mathcal{B}_x \rangle_{k_{AS}} \\
S \rightarrow A &: A, \{|x|\}_{k_{AS}}, \langle \mathbf{c}_{SA} \rangle_{k_{AS}}
\end{aligned}$$

Hodnota proměnné a je obecně nonce a je pro různá pro každého účastníka. Hodnota x může být různá, ale vzhledem k vytváření protokolu pro distribuce klíčů bude hodnota x stejná pro oba účastníky a bude nabývat hodnoty jejich klíče k_{AB} . Protokol tedy bude tvořen 4 zprávami, kde první zpráva bude vyslána od účastníka A serveru S . Vzhledem k tomu, že server odpovídá, až po přijmutí obou prvních zpráv musí následovat odeslání zprávy účastníkem B . Poté server rozešle klíč. Protokol tedy vypadá následovně:

$$\begin{aligned}
1: A \rightarrow S &: A, B, S, \{|a, A, B, S|\}_{k_{AS}}, \langle a, A, B, S \rangle_{k_{AS}} \\
2: B \rightarrow S &: A, B, S, \{|b, A, B, S|\}_{k_{BS}}, \langle b, A, B, S \rangle_{k_{BS}} \\
3: S \rightarrow A &: A, \{|k_{AB}\}_{k_{AS}}, \langle a, b, A, B, S \rangle_{k_{AS}} \\
4: S \rightarrow B &: B, \{|k_{AB}\}_{k_{BS}}, \langle a, b, A, B, S \rangle_{k_{BS}}
\end{aligned}$$

Vidíme, ale že zprávy 1. a 2. na sobe logický nenavazují, B neví, že s ním chce účastník A komunikovat. Tato chyba se dá odstranit přidáním primitiva, během kterého bude B informován, popřípadě se dá zvolit varianta kdy A odešle první zprávu účastníkovi B namísto serveru S . Účastník B obě zprávy spojí a odešle S . Protokol tedy změní svůj tvar na:

$$\begin{aligned}
1: A \rightarrow B &: A, B, S, \{|a, A, B, S|\}_{k_{AS}}, \langle a, A, B, S \rangle_{k_{AS}} \\
2: B \rightarrow S &: A, B, S, \{|a, A, B, S|\}_{k_{AS}}, \langle a, A, B, S \rangle_{k_{AS}}, \{|b, A, B, S|\}_{k_{BS}}, \langle b, A, B, S \rangle_{k_{BS}} \\
3: S \rightarrow A &: A, \{|k_{AB}\}_{k_{AS}}, \langle a, b, A, B, S \rangle_{k_{AS}} \\
4: S \rightarrow B &: B, \{|k_{AB}\}_{k_{BS}}, \langle a, b, A, B, S \rangle_{k_{BS}}
\end{aligned}$$

V tomto tvaru již protokol splňuje všechny náležitosti. Pokud tento protokol porovnáme s výsledkem vygenerovaným pomocí implementované automatické metody, je vidět, že implementovaná metoda je poněkud méně efektivní co do počtu zpráv.

$$\begin{aligned}
1: a \rightarrow s &: s, \{|req(a, reqI(0, Kab0)), a, b, s|\}_{Kas0}, \\
&\langle req(a, reqI(0, Kab0)), a, b, s \rangle_{Kas0}
\end{aligned}$$

```

2: s -> a: a, <req(a, reqI(0, Kab0)), a, b, s>
3: a -> b: b, reqI(0, Kab0), a, b, s, <reqI(0, Kab0), a, b, s>
4: b -> a: a, <reqI(0, Kab0), a, b, s>
5: b -> s: s, {|req(b, reqI(0, Kab0)), a, b, s|}Kbs0,
        <req(b, reqI(0, Kab0)), a, b, s>Kbs0
6: s -> b: b, <req(b, reqI(0, Kab0)), a, b, s>
7: s -> a: a, {|Kab0|}Kas0, <req(a, reqI(0, Kab0)), Kab0, a, b, s>Kas0
8: s -> b: b, {|Kab0|}Kbs0, <req(b, reqI(0, Kab0)), Kab0, a, b, s>Kbs0

```

Tento rozdíl od analytické přístupu vznikl na základě faktu, že implementovaná metoda v současné době pracuje s primitivou jako s celky a neumí generovat takzvaná ‘entangled’, propojená, primitiva. Propojená primitiva jsou taková primitiva, kde zprávy jednoho primitiva jsou mezi časově umístěny mezi zprávami druhého primitiva, tohoto propojení si můžeme všimnout u analytické varianty. Zprávy každého primitiva, ale musí být řazeny ze sebou tak jak jsou definovány v samostatných primitivech, tzn. druhá zpráva nemůže být nikdy před první.

Dalším vlivem je způsob implementace primitiva, jehož úkolem je rozeslání vyžadované znalosti. Ta vyžaduje od každého člena spojovací množiny znalosti, aby serveru odeslal požadavek ve formě `req(účastník, znalost)`, aby se ale zabránilo tomu, že by si účastníci mohli generovat tyto požadavky, bez toho aniž by věděli, že někdo jiný ve skupině tuto znalost požaduje, bylo zavedeno primitivum, které umožňuje generování těchto požadavků pouze ze znalostí `regI`, ta je vytvořena na účastníkovi, který žádá rozeslání informace. Tento požadavek, ale není odeslán v rámci stejného primitiva, ve kterém je požadovaná informace přijata, tím se vytváří teoreticky zbytečné zprávy 2 a 6, které slouží jako potvrzení přijetí požadavku. Jako potvrzení přijetí může totiž sloužit i odeslání dané znalosti.

Dalším problémem, je že metoda neumí přeměrovat tok, tzn. nelze odeslat data od *a* pro *s*, skrz účastníka *b* (zpráva 1), tak aby se účastník *b* tyto znalosti nedozvěděl. I přes tyto problémy, ale metoda byla schopná vygenerovat řešení, které je poměrně podobné metodě analytické.

6.7 Benchmark

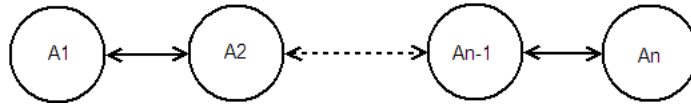
Důležitým faktorem je i rychlost generování protokolů v závislosti na počtu účastníků a počtem primitiv. V této části bude otestována rychlost generování protokolů v lineární a fullmesh topologii. Testování bude provedeno na jednoduchém protokolu, jehož úkolem je přenést znalost od jednoho účastníka k druhému, za podmínky, že odesílaná znalost nebude vyzrazena účastníkovi `spy`.

Všechny testy byly provedeny na notebooku HP typu nc6120 s procesorem Intel Pentium M 1,86Ghz s 768MB paměti RAM, pokud není uvedeno jinak. Časy uvedené v tabulkách budou v ve formátu `m:ss`, v případě časů větších než 60 minut bude použit formát `h:mm:ss`, časy menší jako 1 vteřina nebyly z důvodu možných nepřesností měřeny a budou zapsány jako `<0:01`.

6.7.1 Lineární topologie

Jednotliví účastníci jsou seřazeni za sebou a mezi každými dvěma účastníky, kteří následují za sebou, existuje oboustranný komunikační kanál. Každý účastník zná klíče pro komunikaci

se svými sousedy. Všechny klíče jsou definovány jako dlouhodobé. Topologie je demonstrována na obrázku 6.1.



Obrázek 6.1: Lineární topologie

Byla použita primitiva pro přenos zašifrované a nezašifrované zprávy. Po nalezení řešení byl klauzulí `fail` vyvolán backtracking, tzn. vyhledání dalšího řešení. Změřený čas je časem, kdy metoda nemůže najít žádné další řešení. Tabulka 6.1 shrnuje provedená měření. Sloupec *Počet* označuje počet účastníků.

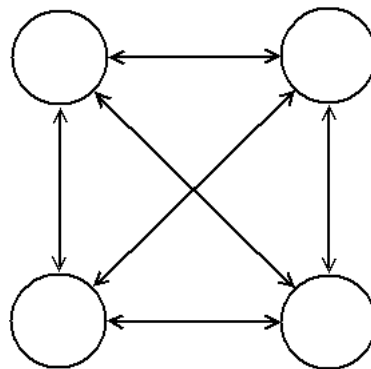
Počet	Čas
10	<1s
20	<1s
50	<1s
100	<1s

Tabulka 6.1: Výsledky měření v lineární topologii

Je vidět, že počet účastníků nemá při generování jednoduchého protokolu v lineární topologii velký význam. Hlavním faktorem je zde pravděpodobně existence pouze jednoho řešení. Lze předpokládat, že doba řešení by rostla s počtem účastníků.

6.7.2 Fullmesh topologie

Full mesh topologie 6.2, tzn topologie kde mezi každou dvojicí účastníků existuje kanál, poskytuje větší množství cest mezi dvěma uživateli, konkrétně $P_n = 1 + (n - 2)P_{n-1}$, kde n je počet účastníků (musí platit podmínka $n \geq 2$). Účastníci znají klíče pro komunikaci s libovolným účastníkem a všechny byly definovány jako dlouhodobé.



Obrázek 6.2: Fullmesh topologie pro $n=4$

Byly provedeny 2 typy měření. První při zachování podmínek tak jak byly definovány na začátku sekce, tedy bezpečný přenos informace x skrz síť. Druhé měření bylo rozšířeno o přenos informace y zpět k prvnímu účastníkovi. Naměřené výsledky jsou prezentovány v

Počet účastníků	Počet různých cest	Čas 1	Čas 2
3	2	<0:01	<0:01
4	5	<0:01	<0:01
5	16	<0:01	<0:01
6	65	<0:01	0:12
7	326	0:01	7:20
8	1957	0:12	6:39:01
9	13700	1:52	neměřeno
10	109601	18:46	neměřeno

Tabulka 6.2: Výsledky měření ve fullmesh topologii

tabulce 6.2. Sloupec *Počet různých cest* definuje počet různých cest mezi 2 účastníky, *čas 1* a *čas 2* jsou časy experimentů.

Čas řešení pro první případ viditelně vykazuje lineární závislost vzhledem k počtu možných cest mezi dvěma uzly ve fullmesh topologii. Vzhledem k tomu že počet možných cest přímo koresponduje s počtem protokolů, které lze vygenerovat (vzhledem k použitým primitivům), lze prohlásit že doba řešení je lineárně závislá na počtu možných variant, které mohou při generování vzniknout.

O druhém experimentu lze teoreticky říct, že celkový počet řešení daného protokolu bude součinem počtu řešení jednotlivých podcílů. I když v případě sedmi účastníků, kdy se posílají obě zprávy, kdy lze předpokládat že počet řešení bude přibližně 10^5 (326^2) je čas vygenerování všech řešení o víc jak polovinu menší než v případě 10 účastníků, kdy se informace odesílá pouze jedním směrem, i když počet možných řešení je přibližně stejný. To může být způsobeno tím, že se pracuje s dvěma menšími stavovými prostory oproti jednomu většímu.

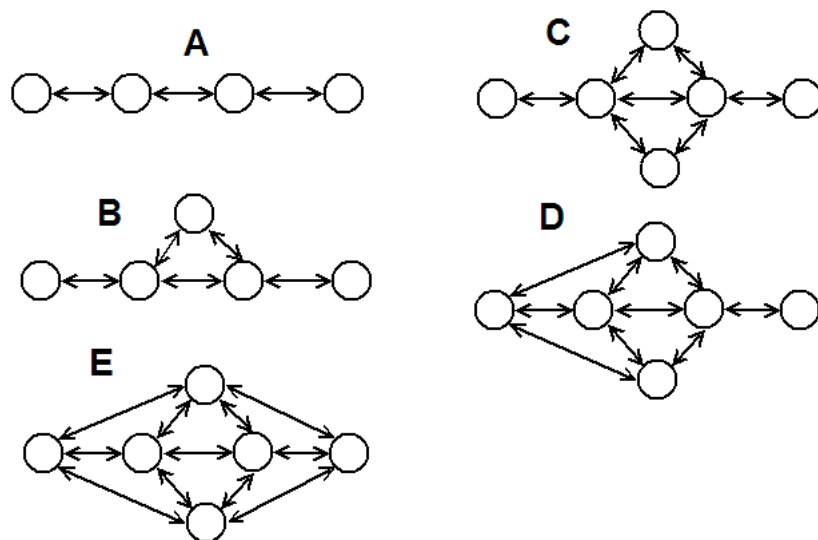
6.7.3 Dlouhodobé klíče

V rámci testování rychlosti bylo prozkoumán i vliv definice klíčů jako dlouhodobých. V případě kdy nejsou klíče definovány jako dlouhodobé je možné je přenášet pomocí mezi jednotlivými účastníky, což výrazně zvyšuje množství možných zpráv, které si mohou mezi sebou jednotliví účastníci poslat a tím pádem i vzroste množství variant nedokončených protokolů, které mohou při generování vzniknout. Na rozdíl od minulých testů, tyto nedokončené protokoly nemusí být validním řešením.

Měření bylo provedeno na několika různých topologiích. Vždy se jednalo o přenos zprávy z nejlevějšího uzlu do nejpravějšího. Topologie pro jednotlivé testy jsou znázorněny na obrázku 6.3.

Rozdíl v době řešení mezi protokoly, které měli definované klíče jako dlouhodobé a těmi které je jako dlouhodobé definovány neměli, byl poměrně výrazný. Pro topologii D nebyly nalezeny všechny řešení ani po 10h. Pro topologii E měření nebylo provedeno. Tyto výsledky jsou shrnuty v tabulce 6.3. Sloupec *Čas S* vyjadřuje čas v případě kdy jsou klíče definovány jako dlouhodobé, *Čas Bez* je čas řešení kdy klíče nebyli definovány jako dlouhodobé.

Podobný výsledek byl zaznamenán i při generování příkladu, který byl použit pro srovnání automatické metody s analytickou. Doba generování příkladu byla přibližně 4 hodiny v případě kdy klíče nebyly definovány jako dlouhodobé. V případě kdy klíče byli definované jako dlouhodobé se tato doba zkrátila na 21 vteřin, což je zrychlení přibližně 700 krát (Protokol byl generován na notebooku s procesorem Core 2 Duo 2,4Ghz s 2GB RAM).



Obrázek 6.3: Topologie použité při testování dlouhodobých klíčů

Topologie	Čas S	Čas Bez
A	<0:01	<0:01
B	<0:01	0:01
C	<0:01	10:58
D	<0:01	>10:00:00
E	<0:01	neměřeno

Tabulka 6.3: Výsledky měření dlouhodobých klíčů

Kompletní specifikace protokolu má tedy významný vliv na rychlost generování metody. Vyplývá to z toho, že pokud je znalost, která je použita v realizaci primitiva, definována jako dlouhodobá, je prohledávání této větve generovaného stromu rychle ukončeno. V případě kdy klíče nejsou definovány jako dlouhodobé metoda zkouší i případy, kdy jsou jednotlivé klíče přeneseny jiným účastníkům, což vede na poměrně rozsáhlé generované stromy, ačkoliv ve většině případů tyto stromy nepovedly na validní řešení.

6.8 Závěr

Program implementuje metodu návrhu pomocí kompozice, tak jak je popsána v [5]. V této kapitole bylo popsáno jakým způsobem definovat prostředí a cíle protokolu, jakým způsobem se provádí vyhledávání vhodných primitiv a techniky používané pro zabránění tvoření nekonečných smyček při vyhledávání. Implementovaná metoda bylo také srovnána s analytickou variantou. Také byly provedeny testy rychlosti generování protokolů pomocí implementované metody. V následující kapitole budou diskutovány možnosti rozšíření současné implementace.

Kapitola 7

Možnosti rozšíření

Současná implementace nabízí několik možností k rozšíření. Samozřejmost je možnost definovat nová primitiva a tím rozšiřovat metodu o nové funkce. Dalším možným rozšířením je možnost generovat provázaná primitiva. Tyto možnosti budou v této části popsány.

7.1 Nová primitiva

Metodu lze snadno rozšířit o nová primitiva a cíle. Primitiva musí splňovat podmínky, které byli definovány v kapitole věnované kompozičnímu návrhu. Primitivum je poté nutné definovat v implementaci pomocí klauzule, která jako první parametr přijímá realizaci a druhý parametr slouží jako návratová hodnota.

Pokud primitivum implementuje i nové cíle, je nutné je specifikovat i ve vyhledávací funkci. V té lze i omezit i primitiva, která budou k plnění tohoto cíle použita.

Například je možné implementovat například primitiva, která jsou schopná odesílat zprávy po jiných kanálech, případně skrz jiné účastníky.

7.2 Provázání primitiv

Jak bylo ukázáno při srovnání analytické a implementované metody provázání běhů primitiv může zredukovat počet potřebných zpráv, které je nutné použít pro realizaci protokolu. Otázkou zůstává jakým způsobem toto provázání realizovat. V současné době aplikace při generování protokolu vytváří strom závislostí mezi jednotlivými primitivy a každé primitivum má své předpoklady a důsledky. Zavedením provázání by bylo nutné definovat předpoklady a důsledky pro jednotlivé zprávy primitiva. Tím by ale výrazně vzrostla složitost vyhledávání. A i po zavedení předpokladu pro zprávy by bylo nutné najít pravidla, podle kterých by se vážali jednotlivé subprotokoly, které řeší jednotlivé cíle nebo předpoklady.

7.3 Další optimalizace

Dalším možným způsobem optimalizace protokolu je konkatenace zpráv se stejným příjemcem a odesílatelem. V současné době je tato optimalizace použita pouze na zprávy, které v protokolu následují hned po sobě. Bylo by možné tuto optimalizaci posunout na úroveň primitiv, kde by bylo možné slučovat i primitiva s podobnou realizací (stejným odesílatelem a příjemcem), které následují za sebou.

Bylo by také možné změnit způsob vyhledávání primitiv i formát samotných primitiv. V současné době se vytváří strom předpokladu pro jednotlivé cíle, přechodem k reprezentaci protokolu jako orientovaného grafu, kde by jednotlivé uzly byly znalosti, cíle a odeslané zprávy by bylo možné rozšířit stavový prostor protokolů, které metoda umí navrhnout. Hrany by byly orientovány od předpokladu k cíli.

Primitiva by byla reprezentována také pomocí grafu a operace přidání primitiva k protokolu by byla reprezentována sjednocením stávajícího grafu s grafem primitiva. Validním grafem protokolu by byl takový graf, který by splňoval tyto podmínky

- Je acyklický
- Všechny cíle jsou dosažené.
- Graf neobsahuje zakázané uzly.

Pokud by graf nebyl acyklický některé znalosti nebo předpoklady by závisely sami na sobě, což je nežádoucí stav. Dosažený cíl, nebo obecněji uzel, lze definovat rekurzivně pomocí dvou pravidel:

- Uzel reprezentuje úvodní znalost

nebo

- Všechny předpoklady uzlu jsou dosažené

Nad takovým grafem lze poté vytvořit topologické uspořádání, tj. graf kde jsou uzly seřazeny za sebou a hrany směřují pouze jedním směrem. Pokud by se v tomto uspořádání vynechají všechny uzly, které nereprezentují zaslání zprávy, lze získat přepis zpráv zasláných v protokolu. Nad těmito zprávami je nutné provést několik testů, například zda některý z účastníků neodesílá zprávy bez vyzvání, případně nějakým způsobem reflektovat tyto podmínky do grafu protokolu.

Tento způsob implementace by poskytoval větší škálu možných protokolů a grafová reprezentace by mohla být vhodnější pro formální verifikaci protokolu. ZK protokoly/autentizace by bylo možné opět realizovat pomocí primitiva.

Kapitola 8

Závěr

Práce se zabývá možností jakými navrhovat protokoly jejichž součástí jsou ZK protokoly. Práce popisuje některé metody návrhy bezpečnostních protokolů a ukazuje možné úpravy, kterými je možné ZK protokoly zakomponovat do těchto metod. Práce klade důraz na metodu návrhu pomocí kompozice.

Byla navržena aplikace, která implementuje danou metodu. Aplikace je schopná navrhnout jednoduché protokoly z popisu prostředí a cílů, které má protokol dosáhnout. Metoda nabízí dobrou rozšiřitelnost z pohledu možnosti přidávat nové primitiva, která mohou řešit nové požadavky. Pomocí primitiv byla implementována i autentizace pomocí ZK protokolů.

Implementovaná metoda byla srovnána s analytickou metodou a bylo ukázáno, že produkuje podobné výsledky. S tím, že automatická metoda je neefektivní z pohledu počtu zpráv. Metoda je tedy vhodná pro automatické generování protokolů. Dále byly navrženy možnosti rozšíření současné implementace.

Literatura

- [1] Abadi, M.; Needham, R.: Prudent Engineering Practice for Cryptographic Protocols. 1995.
- [2] Bella, G.: *Formal correctness of security protocols*. Springer, 2007, ISBN 3-540-68134-5.
- [3] Burrows, M.; Abadi, M.; Needham, R.: A logic of authentication. *ACM Transactions on Computer Systems*, ročník 8, 1990: s. 18–36.
- [4] Buttyan, L.; Staamann, S.; Wilhelm, U.: A Simple Logic for Authentication Protocol Design. In *In 11th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998, s. 153–162.
- [5] Choi, H.-J.: Security protocol design by composition. Technická zpráva, University of Cambridge, Leden 2006.
- [6] Clarke, E.; Grumber, O.; Peled, D.: *Model Checking*. MIT Press, 1999, ISBN 0-26-203270-8.
- [7] Fiege, U.; Fiat, A.; Shamir, A.: Zero knowledge proofs of identity. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, New York, NY, USA: ACM, 1987, ISBN 0-89791-221-7, s. 210–217.
- [8] Goldwasser, S.; Micali, S.; Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, ročník 18, č. 1, 1989: s. 186–208, ISSN 0097-5397.
- [9] Guttman, J. D.; Thayer, F. J.: Authentication Tests. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Washington, DC, USA: IEEE Computer Society, 2000, ISBN 0-7695-0665-8, str. 96.
- [10] Jacquemard, F.: Security Protocols Open Repository.
<http://www.lsv.ens-cachan.fr/Software/spore/index.html>.
- [11] Needham, R.; Schroeder, M.: Using encryption for authentication in large networks of computers. *Commun. ACM*, ročník 21, č. 12, 1978: s. 993–999, ISSN 0001-0782.
- [12] Očenášek, P.: *Automated Design of Authentication and Key Distribution Protocols*. Dizertační práce, VUT Brno, 2010.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9143
- [13] Očenášek, P.; Švéda, M.: An Approach to Automated Design of Security Protocols. In *Proceedings of the International Conference on Networking (ICN 2006)*, IEEE

Computer Society, 2006, ISBN 0-7695-2522-0, str. 4.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8064

- [14] Otway, D.; Rees, O.: Efficient and timely mutual authentication. *SIGOPS Oper. Syst. Rev.*, ročník 21, č. 1, 1987: s. 8–10, ISSN 0163-5980.
- [15] Rosen, A.; Goldreich, O.: *Concurrent Zero-Knowledge*. Springer, 2006, ISBN 3-540-32938-2.
- [16] Ryan, P.; Schneider, S.: *The modelling and analysis of security protocols: the csp approach*. Addison-Wesley Professional, 2000, ISBN 0-201-67471-8.
- [17] Simari, G. I.: *A Primer on Zero Knowledge Protocols*. 2002.
- [18] Stallings, W.: *Cryptography and network security: principles and practice*. Prentice Hall, 1999, ISBN 0-13-869017-0.