VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

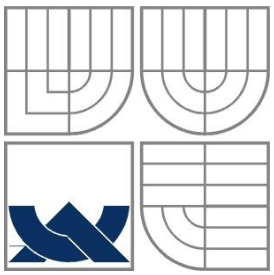
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METODY NÁVRHU PLATEBNÍCH PROTOKOLŮ

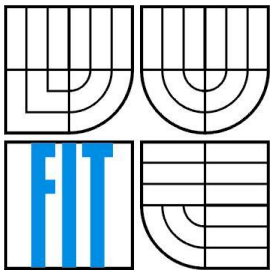
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. PETER MATÚŠKA
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METODY NÁVRHU PLATEBNÍCH PROTOKOLŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. PETER MATÚŠKA
AUTHOR

VEDOUCÍ PRÁCE — ING. PAVEL OČENÁŠEK, PH.D.
SUPERVISOR

BRNO 2011

NA TOMTO MIESTE BUDE ZADANIE DIPLOMOVEJ PRÁCE

Abstrakt

Tato práce analyzuje existující přístupy v návrhu bezpečnostních a platebních protokolů. Popisuje jejich návrh použitím BAN logiky a použitím derivačního systému. Speciální pozornost je věnována metodě kompozice, která je založena na vytváření složitějších protokolů z menších celků a je demonstrována na návrhu nákupní procedury protokolu SET. Metoda je v rámci práce automatizována a implementována v jazyce C++, co umožňuje návrháři na základě jeho požadavků vygenerovat množinu kandidátních protokolů, které může dále použít jako základ v další fázi návrhu.

Abstract

This paper analyses some existing approaches in security and payment protocol design. It describes protocol design using simple BAN logic and using derivation system. Special attention is paid to composition method, which is based on the design of complicated protocols from small parts called primitives and it is demonstrated on design of purchase procedure of SET protocol. This method was automated and implemented in C++ language, which allows designer to generate set of candidate protocols according to his needs and this set can be further used for next phase of protocol design process.

Klíčová slova

Metody návrhu, bezpečnostní protokoly, platební protokoly, platební systém, metoda kompozice

Keywords

Design methods, security protocols, payment protocols, payment system, composition method

Citace

Matúška Peter: Metody návrhu platebních protokolů, diplomová práce, Brno, FIT VUT v Brně, 2011

Metody návrhu platebních protokolů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Pavla Očenáška, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Peter Matúška
15.5.2011

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Pavlovi Očenáškovi, Ph.D. za odbornou pomoc, trpělivost a cenné rady a svoji rodině a přítelkyni za podporu při psaní práce.

Obsah

1 Úvod

Už od pradávna pri prvých prenosoch informácií vznikla potreba túto informáciu nejakým spôsobom zabezpečiť proti zneužitiu v akomkoľvek slova zmysle. Medzi hlavnými kritériami bolo zabezpečiť, aby sa k správe nedostala neoprávnená osoba, bola doručená, nedošlo k jej zmene a bol jednoznačný odosielateľ. V druhej polovici dvadsiateho storočia a nástupom počítačových sietí sa na prenos informácií vytvoril nový kanál – internet. Zo začiatku boli do siete napojené len vojenské a vedecké inštitúcie a zabezpečenie v sieťach bolo minimálne. Všetky dôverné informácie sa šírili v podstate v otvorenej forme. Až v osemdesiatych a začiatkom deväťdesiatych rokov sa celosvetová sieť, v tom čase už verejne známa ako internet, dostala na dosah širokej verejnosti. Pri tak obrovskom počte užívateľov už je nevyhnutné, aby boli prenášané informácie zabezpečené po všetkých stránkach. Obzvlášť dôležitá je bezpečnosť pri práci s financiami. Nápad elektronického nakupovania dotiahol do úspešného konca už v roku 1979 Michael Aldrich. Elektronická komercia (anglicky „electronic commerce“ alebo tiež „e-commerce“) je v podstate využitie internetu na realizáciu obchodných transakcií. Ruku v ruke s elektronickým nákupom a používaním kreditných kariet šiel aj vývoj komunikačných, bezpečnostných a platobných protokolov. A práve návrhom týchto protokolov sa budem venovať v tejto práci.

V nasledujúcej kapitole rozoberiem problematiku platobných systémov a protokolov a uvediem čitateľa do základných pojmov a princípov súvisiacich s bezpečnosťou v tejto oblasti. Objasním obecný model platobného systému, entity zúčastnené v tomto systéme a požiadavky ktoré sa na tieto entity kladú.

V tretej časti potom popíšem niektoré existujúce metódy návrhu bezpečnostných a platobných protokolov. Patrí sem napríklad metóda jednoduchej logiky, ktorej základom je logika BAN, alebo metóda návrhu pomocou derivačného systému. Metódy budem ilustrovať na príklade návrhu bezpečnostného protokolu.

Štvrtá kapitola je určená výhradne metóde kompozície, ktorej sa budem venovať aj vo zvyšku práce. Metóda funguje princípom spájania základných stavebných blokov do väčších celkov, čím je možné rozdeliť zložitejší protokol na menšie celky, ktoré sú jednoduchšie na návrh a tie následne pri dodržaní určitých pravidiel spojiť do výsledného celku. V závere kapitoly ukážem, ako je možné touto metódou navrhnúť platobný protokol.

Ďalšia kapitola sa venuje automatizácii metódy kompozície. Objasním základné myšlienky a ciele, ktoré by mala splniť implementácia tejto metódy a aké výsledky je od nej možné očakávať. Podrobnejšie popíšem návrh niektorých kľúčových komponentov programu, ako sú vstupno-výstupný formát, logika niektorých kľúčových algoritmov a v neposlednom rade použitie programu samotného.

V nasledujúcej kapitole potom budem demonštrovať výsledky získané pri návrhu platobného protokolu. Detailne popíšem prácu s programom a popíšem najdôležitejšie časti protokolov získaných na jeho výstupe.

Siedma kapitola je venovaná možnostiam ďalšieho vývoja programu a situáciám, ktoré sa pri kompozícii protokolov vyskytujú a bolo by možné ich určitým spôsobom zaintegrovať do implementovaného algoritmu a tým celý proces o niečo zoptimalizovať.

Táto diplomová práca nadväzuje na semestrálny projekt, z ktorého získané znalosti boli použité pri tvorbe kapitol 2 a 3, kde analyzujem existujúce metódy návrhu bezpečnostných protokolov a poslúžila aj ako základ pre kapitolu 4, kde sa zaoberám teoretickým základom metódy kompozície. Práve analýza týchto metód bola náplňou spomínaného semestrálneho projektu.

2 Platobné systémy

Táto kapitola popisuje model platobného systému, vysvetľuje dôležité pojmy z oblasti elektronickej komercie a poukazuje na riziká a bezpečnostné otázky, ktoré sa vynárajú pri elektronickom obchodovaní.

2.1 Elektronická komercia

Tento pojem sa čoraz častejšie dostáva medzi širokú verejnosť. Väčšina ľudí za ním však vidí len nakupovanie tovaru prostredníctvom internetovej siete. Ale táto problematika je ďaleko rozsiahlejšia. M. H. Sheriff [1] rozdeľuje elektronickú komerciu na štyri oblasti. Podľa zúčastnených strán, predmetu obchodu a vzťahov medzi zúčastnenými stranami sú to tieto kategórie:

Business-to-business – používa sa aj skratka „B2B“. Jedná sa o obchodné transakcie medzi dvomi firmami. Väčšinou sú to dlhodobo stabilné vzťahy. Pod týmto pojmom sa rozumejú prepojenia medzi finančnými inštitúciami, napríklad bankami.

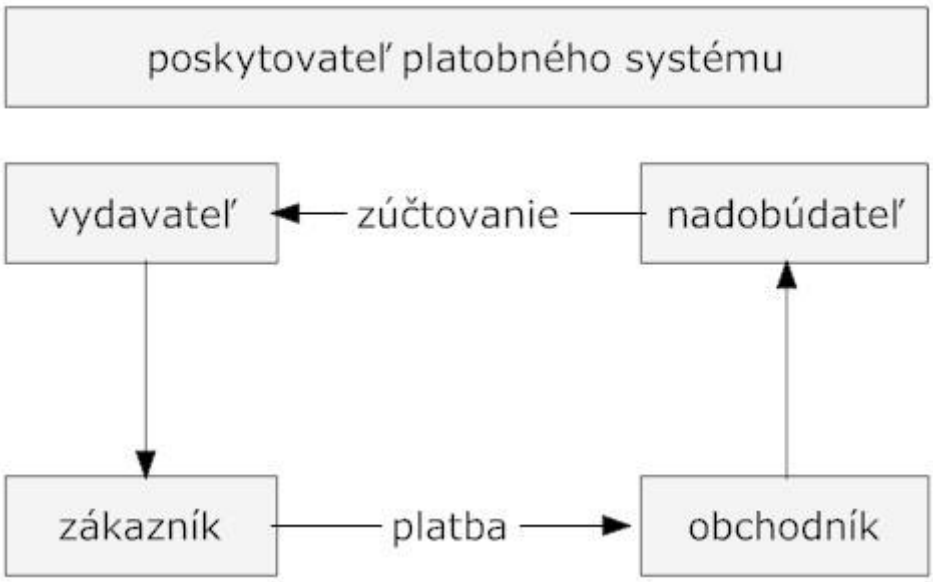
Business-to-customer – tiež známe pod skratkou „B2C“ alebo tiež vzťah firmy a zákazníka. Ako z názvu vyplýva, táto oblasť popisuje koncového zákazníka, ktorý nakupuje prostredníctvom internetu. Nákup je realizovaný rôznymi webovými aplikáciami a internetovými obchodmi, to znamená na diaľku, bez kontaktu zúčastnených strán.

Susedská alebo tiež kontaktná komercia (z anglického „Neighborhood commerce“) – zahrňuje osobné stretnutie zúčastnených strán. Napríklad nákup platobnou kartou v supermarkete.

Peer-to-peer (skratka „P2P“) – priamy obchod medzi dvomi entitami bez ďalších účastníkov. Napríklad prevod peňazí od jedného človeka k druhému.

2.2 Obecný model platobného systému

Obecný model platobného systému znázorňuje obrázok 1. Typicky je platobný systém riadený poskytovateľom platobného systému (Payment System Provider), ktorý je spojený s niekoľkými bankami. Na jednej strane banka hrá úlohu vydavateľa platobnej karty (Issuer) pre zákazníka (Customer) a na strane druhej hrá úlohu nadobúdateľa (Acquirer) platobných príkazov pre obchodníka (Merchant). Keď sa banka chce stať vydavateľom platobnej karty, musí podpísať kontrakt s poskytovateľom platobného systému a dostať pridelené svoje identifikačné číslo BIN (Bank Identification Number). Zákazník, ktorý si u vydavateľa zadováži platobnú kartu, tiež získa svoje identifikačné číslo, PIN (Personal Identification Number). Obchodník, ktorý chce poskytovať elektronické platby, musí na druhej strane uzavrieť zmluvu s bankou, ktorá bude hrať úlohu nadobúdateľa a inkasovať platby od zákazníkov. Predpokladá sa, že zákazník použije počítač na použitie platobného protokolu a na druhej strane obchodník použije zabezpečené zariadenie, ktoré umožní komunikovať platobným protokolom. Zúčtovanie (clearing) medzi bankami sa vykoná na už existujúcej finančnej sieti medzi bankami.



Obrázok 1: Model platobného systému

Pri elektronickej platbe je dôležitý proces autorizácie, ktorý musí overiť, že platobná karta je platná a môže byť použitá na úhradu elektronickou cestou. Ďalej je potrebné overiť, že údaje o zúčtovaní, ktoré uviedol držiteľ karty, sa zhodujú s údajmi u vydavateľa. Nemenej dôležitý je samotný proces zaplatenia, kde sa údaje o transakcii spolu s autorizačnými údajmi posielajú od zákazníka obchodníkovi.

2.3 Bezpečnostné požiadavky na platobný systém

Medzi hlavné bezpečnostné ciele pri elektronickom obchode patrí zabezpečenie komunikačnej siete medzi nakupujúcim zákazníkom a obchodníkom, medzi obchodníkom a jeho bankou a v neposlednom rade medzi bankou obchodníka a bankou zákazníka.

V literatúre [1] sa často uvádzajú tieto hlavné požiadavky na komunikáciu prostredníctvom otvorenej počítačovej siete:

Zabrániť každému, kto nie je účastníkom komunikácie, čítať, meniť alebo inak nepozorovane zasahovať do komunikácie zúčastnených strán.

Znemožniť napodobenie alebo sfalšovanie platobného príkazu a generovanie falošných správ. Napríklad nečestný obchodník nesmie za žiadnych okolností byť schopný použiť údaje, ktoré predtým získal od zákazníka, na vytvorenie falošného nákupu.

Zaistiť zákonné požiadavky, ako napríklad možnosť stornovania nákupu, utajenie osobných údajov zákazníka, ktoré boli pri nákupe získané a prípadné riešenie vzniknutých konfliktov.

Zaistiť dostupnosť služieb tak, ako boli zjednané v zmluve.

Všetkým zákazníkom poskytovať rovnako kvalitné služby bez ohľadu na to, kde sa momentálne nachádzajú.

Z týchto požiadaviek vyplýva niekoľko pojmov, ktoré sa bežne používajú v oblasti komunikácie v počítačových sieťach. Sú to pojmy dôvernosť, integrita, identifikácia, autorizácia, kontrola prístupu a nepopierateľnosť. Dôvernosť znamená, že komunikáciu môže čítať len zúčastnená osoba. Integrita zaručuje, že správy v komunikácii neboli nijakým spôsobom zmenené treťou stranou. Identifikácia je overenie totožnosti účastníka komunikácie. Autorizácia je umožnenie, alebo naopak neumožnenie prístupu k službám užívateľovi na základe jeho identifikácie. Často sa tento pojem definuje aj ako udelenie oprávnení alebo kontrola prístupu. A vďaka nepopierateľnosti nemôže

účastník komunikácie poprieť, že odoslal danú správu. Častou požiadavkou na komunikačné systémy je aj účtovateľnosť, a teda možnosť zaznamenávania všetkých akcií účastníka komunikácie a ich spätná dohľadanie.

3 Metódy návrhu bezpečnostných a platobných protokolov

V tejto kapitole uvediem niektoré používané metódy návrhu protokolov. V kapitole 3.1 sa zaoberám metódou, ktorá používa k návrhu jednoduchú logiku založenú na BAN [8] logike. Uvádzam v nej základné pravidlá logiky a ich aplikovanie na protokol „Woo and Lam“ [9]. V kapitole 3.2 popisujem návrh protokolu pomocou derivačného systému. V úvode vysvetľujem princíp tejto metódy a následne na príklade ukážem deriváciu protokolov rodiny STS[1]. A konečne v kapitole 3.3 uvádzam stručný prehľad metódy kompozície, ktorá je vhodná okrem iného aj na návrh platobných protokolov, preto sa jej ďalej podrobne venujem samostatne v kapitole 4.

3.1 Použitie jednoduchej logiky

Tento prístup využíva logiku BAN, ktorej autormi sú páni Burrows, Abadi a Needham [8]. Je to akýsi vrstvový prístup k návrhu protokolu, kde začíname od abstraktného popisu a končíme až pri implementácii. Základnými kameňmi logiky sú dve množiny – množina znalostí a množina predpokladov. Množina znalostí je množina všetkých správ, ktoré sú známe subjektu A a množina predpokladov obsahuje znalosti, o ktorých subjekt A predpokladá, že sú známe subjektu B. Jednotlivé operácie v protokole potom menia obsah práve týchto dvoch množín. Subjekty, ktoré komunikujú, konajú práve podľa toho, čomu veria (v angličtine sa často používa termín „belief and action“). BAN logikou nie je možné dokázať, že protokol je nebezpečný. Ale ak sa nám nepodarí nájsť dôkaz o tom, že je bezpečný, je to prinajmenšom podozrivé.

3.1.1 Notácia a základné pravidlá BAN logiky

V tejto podkapitole objasním základy logiky BAN tak, ako ich popísali vo svojej práci autori [8]. Ako som už naznačil, subjekty konajú podľa toho, čomu veria. Preto prvým základným pravidlom bude pravidlo: **A**lica verí tvrdeniu **P**. Formálne sa to zapíše ako $A \mid \equiv P$, čo čítame ako „**A** verí **P**“. Tvrdenie „**A**lica verí, že $\mathbf{K_{AS}}$ je dobrý kľúč na komunikáciu s **S**“ by sme zapísali ako $A \mid \equiv A \overset{K_{AS}}{\longleftrightarrow} S$ a čítali ako „**A** verí, že $\mathbf{K_{AS}}$ je dobrý kľúč pre **A** a **S**“. **S** vo väčšine prípadov bude zastávať úlohu dôveryhodného serveru alebo certifikačnej autority. Ak **A**lica verí, že **S** je dôveryhodný, aby vytvoril dobrý kľúč na komunikáciu s **B**obom, napíšeme to ako $A \mid \equiv S \Rightarrow A \overset{K_{AB}}{\longleftrightarrow} B$. Čítame to ako „**A** verí, že **S** má právo vytvárať dobré kľúče pre **A** a **B**“.

Keď **A**lica dostane správu, zapíšeme to ako $A \lhd P$ a čítame „**A** vidí **P**“. Vidieť ale neznamená veriť, kým nevieme, kto správu poslal. Ak **A**lica poslala správu, ktorá obsahuje tvrdenie **P**, zapíšeme to ako $A \mid \sim P$ a hovoríme, že „**A**lica niekedy tvrdila **P**“. **A**lica to však mohla tvrdiť veľmi dávno, takže potrebujeme vyjadriť čerstvosť jej správy. Zápis $\sharp\,(P)$ značí že **P** je čerstvé (fresh).

Základné pravidlá BAN logiky teda môžeme zhrnúť do nasledujúceho zoznamu:

$P \mid \equiv X$	P verí X
$P \lhd X$	P vidí X

[1] Station-to-Station - http://en.wikipedia.org/wiki/Station-to-Station_protocol

$P \mid\sim X$	P niekedy tvrdilo X
$\#(P)$	P je čerstvé
$P \Rightarrow X$	P má právomoc nad X
$P \overset{K}{\leftrightarrow} Q$	K je dobrý kľúč na komunikáciu medzi P a Q
$\overset{K}{\rightarrow} P$	P má K ako verejný kľúč
$P \overset{X}{\rightleftharpoons} Q$	X je tajomstvo, ktoré poznajú len P a Q
$\{X\}_Y$	X je skombinovaná (zašifrovaná) s (tajným) Y

V BAN logike sa používa notácia $\frac{P}{Q}$. Znamená to, že ak P je pravdivé, tak aj Q je pravdivé. Takže ak Alica verí X a platí $\frac{X}{Y}$, potom bude veriť aj Y. Zo základných pravidiel sú odvodené napríklad nasledujúce postuláty:

$\frac{P \mid\equiv P \overset{K}{\leftrightarrow} Q, P \lhd \{X\}_K}{P \mid\equiv Q \mid\sim X}$ - ak P verí, že K je dobrý kľúč na komunikáciu medzi P a Q a P vidí X zašifrované pomocou K, potom P verí, že Q odoslal správu X

$\frac{P \mid\equiv Q \mid\sim X, P \mid\equiv \#(X)}{P \mid\equiv Q \mid\equiv X}$ - ak P verí, že Q odoslal správu X a P verí, že X je čerstvé, tak P verí, že Q verí X

$\frac{P \mid\equiv Q \Rightarrow X, P \mid\equiv Q \mid\equiv X}{P \mid\equiv X}$ - ak P verí, že Q má právomoc nad X a P verí, že Q verí X, potom P verí X

Existujú aj ďalšie postuláty odvodené zo základných pravidiel logiky, ale ich odvodenie ponechám na čitateľovi. Dôležité je poznamenať, že použitím týchto postulátov je možné dokázať, že konkrétny účastníci komunikácie veria, že môžu používať konkrétne kľúče na komunikáciu. Ak tento dôkaz zlyhá, ukazuje to možnosť útoku na protokol.

3.1.2 Návrh protokolu užitím jednoduchej logiky

Návrhom protokolu pomocou logiky sa vo svojej práci [2] zaoberajú L. Buttyán, S. Staamann a U. Wilhelm. Táto podkapitola kapitola sa opiera o poznatky práve z tejto práce. Predpokladajme, že distribuovaný systém je množina účastníkov (užívatelia, počítače, procesy) a komunikačných kanálov. Účastníci medzi sebou komunikujú podľa pravidiel vopred definovaného protokolu za účelom dosiahnuť nejaký cieľ, napríklad poskytovať službu. Komunikácia je založená na posielaní správ po komunikačných kanáloch. Kanál môžeme považovať za zariadenie, ktoré má svoje prístupové možnosti. Je charakteristický tým, že má množinu účastníkov, ktorí môžu čítať a množinu účastníkov, ktorí môžu zapisovať do kanála (readers, writers). To znamená účastníkov, ktorí môžu dostávať správy cez kanál a účastníkov, ktorí môžu posielať správy cez kanál. Ak označíme komunikačný kanál ako C, odpovedajúce množiny potom označíme ako r(C) a w(C). Subjekt P je čitateľom (má právo dostávať správy) kanála C zapíšeme ako P = r(C). Obdobne by sme zapísali možnosť subjektu P zapisovať (posielať správy) do kanála. Kombináciou BAN logiky a použitím komunikačných kanálov dostaneme množinu pravidiel:

- $P \lhd C(X)$ – P vidí C(X). Niekto poslal správu X kanálom C, ale ak P nemá právo čítať kanál C, P nerozumie správe X
- $P \lhd X \mid C$ – P vidí X cez kanál C. P dostal správu X kanálom C. Niekto poslal správu X a P môže čítať kanál C.
- $P \lhd X$ – P vidí X. Niekto poslal správu X kanálom, ktorý P môže čítať

- $\#(X)$ – X je čerstvé. X sa ešte nevyskytlo v doterajšom priebehu komunikácie. Platí väčšinou pre nonce
- $P \mid\sim X$ – P niekedy povedal X. Nevieme presne kedy to bolo, ale stalo sa to.
- $P \mid\mid\sim X$ – P nedávno povedal X
- $P \mid \equiv X$ – P verí X. Verí, že X je pravdivé, ale to nemusí byť pravdivé.

Buttyán, Sebastian a Wilhelm [2] uviedli niektoré interferenčné pravidlá a na nich postavené teorémy. Napríklad pravidlo čerstvosti $\frac{P \mid\equiv(Q \mid\sim X), P \mid\equiv \#(X)}{P \mid\equiv (Q \mid\mid\sim X)}$, ktoré hovorí, že ak P verí, že Q niekedy povedal X a súčasne P verí, že X je čerstvé, tak potom P verí, že Q nedávno povedal X. A napríklad teorém $\frac{P \mid\equiv((Q \mid\mid\sim \phi)\rightarrow \phi), P \mid\equiv(Q \mid\mid\sim \phi)}{P \mid\equiv \phi}$ hovorí, že ak P verí, že Q je čestný a P verí, že Q nedávno povedal ϕ, kde ϕ je tvrdenie, potom P verí ϕ. Z takýchto interferenčných pravidiel a teorém sa reverzáciou odvodzujú syntetické pravidlá, ktoré sa potom použijú pri vytváraní protokolu. Napríklad pre vyššie uvedený teorém by sa syntetické pravidlo odvodilo nasledovne: Aby P mohol uveriť tvrdeniu ϕ, musí uveriť, že Q je čestný a kompetentný a že Q nedávno povedal ϕ. Zapíše sa to takto:

$$
\begin{aligned}
&P \mid\equiv \phi \\
&\quad \hookrightarrow P \mid \equiv (Q \mid\mid\sim \phi) \\
&\quad \hookrightarrow P \mid \equiv ((Q \mid\mid\sim \phi) \rightarrow \phi)
\end{aligned}
$$

Ako ukážku zhotovenia protokolu pomocou logiky som zvolil autorizačný protokol „Woo and Lam“ [2][9]. Symbolický zápis protokolu je nasledovný:

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Zapísané pomocou logiky:

1. $B \lhd A$
2. $A \lhd N_B$
3. $B \lhd C_{AS}(N_B)$
4. $S \lhd C_{BS}(A, C_{AS}(N_B))$
5. $B \lhd C_{BS}(N_B)$

Musia platiť nasledujúce predpoklady:

1. $A \in r(C_{AS})$	A môže čítať z kanálu C_{AS}
2. $S \in r(C_{AS})$	S môže čítať z kanálu C_{AS}
3. $A \mid\equiv (w(C_{AS}) = \{A, S\})$	A verí, že len A a S môžu zapisovať do kanálu C_{AS}
4. $S \mid\equiv (w(C_{AS}) = \{A, S\})$	S verí, že len A a S môžu zapisovať do kanálu C_{AS}
5. $B \in r(C_{BS})$	B môže čítať z kanálu C_{BS}
6. $S \in r(C_{BS})$	S môže čítať z kanálu C_{BS}
7. $B \mid\equiv (w(C_{BS}) = \{B, S\})$	B verí, že len B a S môžu zapisovať do kanálu C_{BS}

8. $S \mid \equiv (w(C_{BS}) = \{B, S\})$ S verí, že len B a S môžu zapisovať do kanálu C_{BS}
9. $A \mid \equiv \big((S \mid\mid \sim \phi) \longrightarrow (S \mid \equiv \phi)\big)$ A verí, že S je čestný
10. $A \mid \equiv \Big(\big(S \mid \equiv (B \mid \sim X)\big) \longrightarrow (B \mid \sim X)\Big)$ A verí, že S vie rozhodnúť, či B poslal X
11. $B \mid \equiv \big((S \mid\mid \sim \phi) \longrightarrow (S \mid \equiv \phi)\big)$ B verí, že S je čestný
12. $B \mid \equiv \Big(\big(S \mid \equiv (A \mid \sim X)\big) \longrightarrow (A \mid \sim X)\Big)$ B verí, že S vie rozhodnúť, či A poslal X
13. $B \mid \equiv \#(N_B)$ B verí, že N_B je čerstvé

Cieľom protokolu je ubezpečiť B, že komunikuje naozaj s A. Aj na základe zverejnených útokov [3,4] môžeme tento cieľ špecifikovať ako $B \mid \equiv \big(A \mid\mid \sim (B, N_B)\big)$, čo znamená, že B verí, že A nedávno poslalo nonce N_B subjektu B. Pomocou syntetických a interferenčných pravidiel a teorém popísaných v [2] sa dopracujeme k forme:

1. $A \lhd (B, N_B)$
2. $S \lhd C_{AS}(B, N_B)$
3. $B \lhd C_{BS}\big(A \mid \sim (B, N_B)\big)$

Symbolický zápis odvodeného protokolu bude:

1. $A \rightarrow B : A$
2. $B \rightarrow A : B, N_B$
3. $A \rightarrow B : \{B, N_B\}_{K_{AS}}$
4. $B \rightarrow S : A, \{B, N_B\}_{K_{AS}}$
5. $S \rightarrow B : \{A, B, N_B\}_{K_{BS}}$

Takto odvodený protokol je oproti originálnemu „Woo and Lam“ protokolu odolný aj voči útokom publikovaných v [3][4].

3.2 Návrh pomocou derivačného systému

Hlavná myšlienka tohto prístupu spočíva v tom, že sa vyberú dva (prípadne viac) protokolov, vyberú sa niektoré ich vlastnosti a skombinujú sa v novo vytvorenom protokole. Ten má tak vlastnosti oboch pôvodných protokolov, čo sa môže hodiť. Postup vo svojej práci [5] popísali A. Datta, A. Derek, J.C. Mitchell a D. Pavlovic a vychádza z nej táto kapitola.

Systém na derivovanie bezpečnostných protokolov sa skladá zo základných stavebných blokov nazývaných komponenty a množiny operácií, ktorými sa z pôvodných protokolov vytvorí nový protokol. Existujú tri typy operácií a to skladanie, transformácia a vylepšenie. Operácia skladanie zoberie dva protokoly a nejakým spôsobom ich spojí do jedného. Operácia vylepšenie sa vzťahuje vždy na konkrétnu správu jedného protokolu. Napríklad nahradenie nešifrovaného textu za šifrovaný je vylepšenie. Nemení sa pri tom ani počet správ ani štruktúra daného protokolu. Transformácia sa podobne ako vylepšenie vzťahuje na jeden protokol. Na rozdiel od vylepšenia sa však v tomto prípade jedná o operáciu, ktorou sa môže zmeniť až niekoľko krokov protokolu, presunúť údaje z jednej správy do druhej, kombinovať kroky či vložiť kroky nové. Napríklad presun údajov zo správy protokolu do inej správy (medzi rovnakými účastníkmi) je transformácia.

3.2.1 Príklad návrhu protokolu pomocou derivačného systému

Datta a Derek [5] vysvetľujú vo svojej práci derivačný systém na derivovaní rodiny protokolov STS (Station-To-Station), ktoré slúžia na výmenu kľúčov. Patria sem napríklad protokol IKE[2] alebo protokoly JFKi a JFKr[3], ktoré by mali nahradiť protokol IKE. Medzi základné vlastnosti protokolov rodiny STS patrí utajenie kľúča, vzájomná autorizácia, ochrana voči DoS útokom, ochrana identity a efektivita výpočtu. Deriváciu budem ilustrovať na postupnom použití jednotlivých operácií a komponent. Na deriváciu STS protokolov sú použité dva základné komponenty. Prvý je komponent K1, vybraný z protokolu Diffie-Hellman [10]. Tento protokol poskytuje spôsob, akým si dve komunikujúce strany môžu ustanoviť zdieľaný kľúč, ktorý nie je odhaliteľný pasívnym útočníkom. Bezpečnosť kľúča závisí na sile použitého šifrovacieho algoritmu. Komponent (K1), ktorý je použitý pri derivácii, obsahuje len túto výpočtovú časť protokolu Diffie-Hellman. V rámci nej teda nedochádza k žiadnemu zasielaniu správ medzi účastníkmi. Predpokladajme, že Alice (A) a Bob (B) sú účastníci komunikácie. Komponent potom môžeme vyjadriť takto:

A: vygeneruje náhodnú hodnotu a a vypočíta g^a
B: vygeneruje náhodnú hodnotu b a vygeneruje g^b

Druhý komponent (K2) je postavený na princípe autorizácie typu Challenge-Response. Jedná sa o rodinu protokolov, kde jeden subjekt má výzvu („challenge“) a druhý mu na základe nej pošle validnú odpoveď („response“). Je založená na podpise a používa sa na jednosmernú autorizáciu.

$$A \rightarrow B: m$$
$$B \rightarrow A: SIG_B(m)$$

Predpokladá sa, že m je čerstvá hodnota alebo keksík (anglicky „nonce“) a že Alice disponuje verejným kľúčom Boba a teda vie overiť podpis. Použitím kompozície, zložením komponentu K2 dva krát za seba, dostávam sekvenciu:

$$A \rightarrow B: m$$
$$B \rightarrow A: SIG_B(m)$$
$$B \rightarrow A: n$$
$$A \rightarrow B: SIG_A(n)$$

Transformáciou, spojením dvoch správ, ktoré tesne za sebou odosiela Bob, do jednej správy, dostaneme sekvenciu:

$$A \rightarrow B: m$$
$$B \rightarrow A: n, SIG_B(m)$$
$$A \rightarrow B: SIG_A(n)$$

Opäť je možné použiť transformáciu a pridať do podpisov nonce m a n:

$$A \rightarrow B: m$$

[2] IKE – Internet Key Exchange, RFC 2409 http://tools.ietf.org/html/rfc2409
[3] JFK – Just Fast Keying

$$B \rightarrow A: n, SIG_B(n, m)$$
$$A \rightarrow B: SIG_A(m, n)$$

V tomto momente je vhodné použiť komponent K1 a pridať do systému aj časť protokolu Diffie-Hellman:

$$A \rightarrow B: g^a$$
$$B \rightarrow A: g^b, SIG_B(g^b, g^a)$$
$$A \rightarrow B: SIG_A(g^a, g^b)$$

Odvodením z Diffie-Hellman je možné získať kľúč K, ktorým sa zašifrujú podpisy účastníkov - operácia vylepšenie. Vznikne tak protokol STS:

$$A \rightarrow B: g^a$$
$$B \rightarrow A: g^b, E_K(SIG_B(g^b, g^a))$$
$$A \rightarrow B: E_K(SIG_A(g^a, g^b))$$

Tento protokol je vďaka zdieľanému Diffie-Hellman tajnému kľúču odolný voči útokom „man in the middle“, kde útočník je tajným prostredníkom komunikácie. Postupne je možné použiť ďalšie transformácie a vylepšenia a dopracovať sa tak aj k protokolom ako JFKi alebo JFKr.

3.3 Metóda kompozície

Predstavil ju vo svojej práci Choi [6]. Táto metóda sa zaoberá návrhom protokolov po častiach. Bezpečnostné vlastnosti, ktoré od protokolu požadujeme, sa rozdelia na samostatné časti, ktoré sa potom riešia jednotlivo. Tieto časti je potom možné vhodným spôsobom spojiť do jedného celku (protokolu) tak, že sa nebudú ich jednotlivé prvky navzájom ovplyvňovať. Medzi typické časti, ktoré je možné riešiť samostatne, patrí napríklad autorizácia, utajenie alebo nepopierateľnosť.

Ako som už spomenul, metóda kompozície je založená na skladaní protokolov z častí. Tieto časti sa volajú tiež primitívy. Jedná sa o časti a mechanizmy, ktoré môžeme nájsť v známych bezpečnostných protokoloch a princípoch, ako napríklad autorizácia typu „challenge and response“. Tieto primitívy boli navrhnuté tak, aby nekolidovali jedna s druhou, čo ich predurčuje k tomu, aby na ich základe bolo možné použitím určitých pravidiel vybudovať nový protokol. Inými slovami, táto metóda umožňuje rozdeliť komplexný protokol na jednotlivé jednoduchšie časti – komponenty, ktoré sa dajú jednoduchšie navrhnúť a verifikovať. Výhoda tohto prístupu je obdobná výhodám modulárneho programovania.

Túto metódu je možné aplikovať na širokú škálu prostredí. Okrem iných ju môžeme uplatniť aj pri návrhu platobných protokolov. Podrobným popisom tejto metódy a návrhom platobného protokolu na jej základe sa budem zaoberať v kapitole 4.

4 Návrh protokolu metódou kompozície

V tejto kapitole sa budem zaoberať podrobnejšie návrhom platobného protokolu metódou kompozície, ktorú predstavil Choi [6] a ktorej stručný prehľad som uviedol v závere kapitoly 3.

Bez ohľadu na to, ktorú metódu návrhár použije pri tvorbe protokolu, musí si byť presne vedomý toho, aké ciele a požiadavky si na nový protokol kladie. Hlavnými cieľmi sú vo väčšine prípadov autorizácia a utajenie. Ďalšou požiadavkou môže byť napríklad nepopierateľnosť.

Predpokladajme teda, že protokol, ktorý budeme navrhovať, musí splniť nejakú vopred danú množinu cieľov. Existuje mnoho spôsobov, ako ich implementovať. Väčšina bežných metód nerieši tieto problémy jeden po druhom, ale snaží sa ich riešiť súčasne, čo nie je jednoduché a s narastajúcim počtom cieľov sa náročnosť ešte zvyšuje. Naopak metóda kompozície rieši tieto problémy individuálne, jeden po druhom. Hľadať riešenie len jednej požiadavky je pre dizajnérov omnoho jednoduchšie, než hľadať riešenie, ktoré by splnilo všetky požiadavky naraz. Každý problém je implementovaný nezávisle použitím mechanizmov, ktoré sú overené a navzájom sa neovplyvňujú a tieto mechanizmy sú potom spojené do jedného celku, ktorý zaručí, že sa všetky požiadavky splnia naraz.

Prístup je obdobný modulárnemu programovaniu. Umožňuje zložitejšie ciele vytvárať z jednoduchých cieľov, čo zjednodušuje verifikáciu a garantuje tak správnosť protokolu získaného týmto prístupom.

4.1 Základné pojmy

Táto kapitola pokrýva teoretické základy potrebné na pochopenie metódy kompozície a jej následnú automatizáciu. Choi [6] vo svojej práci zachádza v teórii ešte o niečo ďalej a všetky jeho tvrdenia a teórie dokazuje matematickou cestou, ale to už je mimo rozsah tejto práce a nie je to nevyhnutné pre zvládnutie tejto metódy. Preto v nasledujúcich podkapitolách uvediem len tie fakty, o ktoré sa budem opierať pri implementácii.

4.1.1 Správy a termy

Nech A je množina správ, ktoré si medzi sebou posielajú komunikujúce subjekty. Jednotlivé prvky tejto množiny sa nazývajú termy. V zásade sa termy rozdeľujú na tri druhy – konštantné termy, variabilné termy a šifrovacie kľúče.

Označme množinu konštantných termov ako $C \subseteq A$ a množinu variabilných termov ako $V \subseteq A$. Do množiny C patria termy, ktoré nie sú závislé na konkrétom behu protokolu, ako napríklad mená subjektov alebo konštantné čísla. Množina V obsahuje premenlivé termy, napríklad časové známky (timestamp) alebo keksík (nonce). Platí, že C a V sú disjunktné.

Množina kľúčov $K \subseteq A$, ktorá je disjunktná s množinou $C \cup V$, obsahuje privátne alebo verejné kľúče a krátkodobé alebo dlhodobé kľúče. Verejný kľúč subjektu S sa značí ako $pub(S)$, privátny ako $prv(S)$ a symetrický ako $shr(S)$. Zdieľaný symetrický kľúč, ktorý sa používa na komunikáciu medzi subjektmi A a B, sa značí ako K_{AB}, pár podpisový a overovací kľúč subjektu A ako (sk_A, vk_A) a pár šifrovací a dešifrovací kľúč subjektu A ako (ek_A, dk_A). Kľúč obecne označujeme ako k.

Na dosiahnutie utajenia určitého termu x, prípadne množiny termov, sa používa šifrovanie a značí sa to ako $\{|x|\}_k$. Ďalším dôležitým prvkom je hašovanie. Jedná sa o nevratnú operáciu, ktorá pretransformuje term na inú hodnotu buď s použitím kľúča, alebo bez neho. Značíme to ako $\langle x \rangle_k$ prípadne $\langle x \rangle$.

4.1.2 Udalosti

Predpokladajme, že komunikujúci subjekt má svoj interný stav a komunikuje po verejnom kanáli. Na základe svojho stavu sa subjekt rozhoduje, akú akciu následne vykoná. Tieto akcie sa terminológii kompozičnej metódy nazývajú tiež udalosti. Poznáme tieto typy udalostí:

$e^+(x)$	term x je odosielaný subjektom
$e^-(x)$	term x je prijímaný subjektom
(vx)	term x je generovaný subjektom
$(x/p(x))$	term x je porovnávaný so vzorom $p(x)$

Udalosti je možné indexovať názvom subjektu, ktorého sa udalosť týka, napríklad $e_A^+(x)$. $e^+(x)$ a $e^-(x)$ sa nazývajú externé udalosti a (vx) a $(x/p(x))$ interné udalosti. Tieto sa dejú v rámci jedného subjektu a ostatné subjekty o nich nevedia. (vx) môže byť napríklad vygenerovanie keksíku a $(x/p(x))$ môže byť napríklad dešifrovanie šifrovanej správy. Vždy, keď jeden zo subjektov vygeneruje nejaký nový term x (nastane udalosť (vx)), musí byť jasné, pre ktoré subjekty je tento term určený a term je na tieto subjekty viazaný. Skupina týchto subjektov sa volá tiež väzobná skupina (z anglického výrazu „binding group“) a označuje sa β_x.

Dôležitým faktorom pri práci s udalosťami je ich poradie. Ak udalosť e_i nastala pred udalosťou e_j, značíme to ako $e_i < e_j$. V rámci metódy kompozície platia nasledujúce pravidlá:

1. $e^+(x) < e^-(x)$
2. $(vx) < e^+(x)$

Znamená to, že každý term musí byť najprv odoslaný a až potom prijatý a že každý term musí byť najprv vygenerovaný a až potom odoslaný. A teda udalosti vpravo nemôžu nastať bez udalostí vľavo. Dve udalosti sú na sebe nezávislé ak platí $e_i \not< e_j \wedge e_j \not< e_i$.

4.2 Stavebné prvky - primitívy

V tejto kapitole uvediem základné primitívy tak, ako ich definuje vo svojej práci Choi [6]. Ďalej vysvetlím, ktoré vlastnosti musí mať každá z nich, aby bolo možné ich v rámci kompozície použiť. A v neposlednom rade ozrejmím problematiku pridávania ďalších nových primitív medzi existujúce tak, aby boli dodržané vlastnosti potrebné pre ich kompozíciu.

Primitívy by sa mohli rozdeliť na dve skupiny podľa toho, či sú určené pre symetrickú alebo asymetrickú kryptografiu. Rozdiel však bude len v použitých kľúčoch, takže striktným rozdelením na tieto dve podskupiny sa nebudem ďalej zaoberať. Na tomto mieste ale uvediem ako symetrické, tak i asymetrické verzie základných primitív. Vzhľadom k tomu, že v oblasti platobných protokolov sa takmer výhradne používa asymetrická kryptografia, budem ďalej pracovať len s týmito verziami primitív.

Primitívy pre symetrickú kryptografiu:

Typ 1:	$A \rightarrow B$:	$B, \{\lvert x, \beta_x \rvert\}_{k_{AB}}, \langle x, \beta_x \rangle_{k_{AB}}$
	$B \rightarrow A$:	$A, \langle x, \beta_x \rangle$
Typ 2:	$A \rightarrow B$:	$B, \{\lvert x, \beta_x \rvert\}_{k_{AB}}, \langle x, \beta_x \rangle_{k_{AB}}$
	$B \rightarrow A$:	$A, \langle x, \beta_x \rangle_{k_{BA}}$

Typ 3: $A \rightarrow B$: $x, \beta_x, \langle x, \beta_x \rangle_{k_{AB}}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{k_{BA}}$

Primitívy pre asymetrickú kryptografiu sa líšia len v tom, že používajú asymetrické šifrovanie. Tu sú:

Typ 1: $A \rightarrow B$: $B, \{|x, \beta_x|\}_{ek_B}, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle$

Typ 2: $A \rightarrow B$: $B, \{|x, \beta_x|\}_{ek_B}, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{sk_B}$

Typ 3: $A \rightarrow B$: $x, \beta_x, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{sk_B}$

V zápisoch, ktoré som práve uviedol, sa objavuje prvok β_x. Jedná sa o tzv. väzobný prvok (z anglického výrazu „term binder“) a je to usporiadaná množina subjektov, ku ktorým je naviazaný daný term (v tomto prípade je to term x). Napríklad, ak subjekt A vygeneruje nonce x pri komunikácii so subjektom B, potom x bude naviazaný na subjekt A ako iniciátora a subjekt B ako respondenta. Zapíšeme to ako $(x; \beta_x)$, kde $\beta_x = (A, B)$. Keďže sa jedná o usporiadanú množinu subjektov, na poradí subjektov A a B záleží. Dvojica $(x; \beta_x)$ sa nazýva tiež parametrom primitívy. Term x nemusí byť vždy jeden. V týchto prípadoch sa x nahradí za vektor $\vec{x}$.

4.3 Spájanie a pridávanie nových primitív

Z hľadiska metódy kompozície je protokol definovaný ako množina primitív, ktorá má pevne dané usporiadanie a je definovaná svojim parametrom. Parametrom primitívy sa rozumie dvojica: variabilný term (prípadne množina termov) a jeho väzobná množina.

Aby bolo možné primitívy skladať dohromady, musíme si overiť, že medzi nimi nedôjde k interferencii. Inými slovami, ciele každej primitívy musia zostať naplnené aj po spojení týchto primitív. Nech $p_1(\omega_1)$ a $p_2(\omega_2)$ sú primitívy s parametrami ω_1 a ω_2 a nech $g_1(\omega_1)$ a $g_2(\omega_2)$ sú ciele, ktoré tieto primitívy naplňujú, keď sú použité nezávisle na sebe. Keď ich však použijeme súčasne, musíme si overiť, že nedôjde k interferencii. Ak p_2 prebehne pred p_1 a p_2 nenaruší žiadne predpoklady, ktoré potrebuje k naplneniu svojho cieľa p_1, potom g_1 nie je ovplyvnený spustením p_2 a hovoríme, že je nedeštruktívny. Ak toto tvrdenie platí aj pre g_2, primitívy sa navzájom neovplyvňujú a môžu byť spojené dohromady. Niektoré z tých, ktoré vyššie spomínané vlastnosti splňujú, sú uvedené v predchádzajúcej kapitole.

Definícia 1: Ak spustenie $\boldsymbol{p_2}$ nenaruší predpoklady, ktoré potrebuje $\boldsymbol{p_1}$ k naplneniu svojho cieľa, potom cieľ $\boldsymbol{g_1}$ je **nedeštruktívny**, teda spustenie $\boldsymbol{p_2}$ nemá žiadny efekt na $\boldsymbol{g_1}$.

Ďalším dôležitým pojmom je nekonštruktívnosť. Hovoríme, že g_1 je nekonštruktívny vtedy, keď nemôže nastať bez splnenia cieľa g_2.

Definícia 2: Ak splnenie cieľa $\boldsymbol{g_1}$ nemôže nastať bez splnenia cieľa $\boldsymbol{g_2}$, potom cieľ $\boldsymbol{g_1}$ je **nekonštruktívny**, teda sa nemôže splniť sám o sebe.

Príklad nedeštruktívneho cieľa je napríklad utajenie. V prípade, že nejaká primitíva má ako cieľ utajenie, nesmie existovať primitíva, ktorá by zabránila dosiahnuť tento cieľ. Príkladom nekonštruktívneho cieľa je autorizácia. Je zrejmé, že táto nemôže nastať sama o sebe, bez naplnenia niektorých predchádzajúcich cieľov. Subjekt nemôže byt autorizovaný „z ničoho nič".

Dôležitou súčasťou kompozície je vytváranie nových primitív a ich pridávanie medzi existujúce. Takto vytvorená primitíva musí spĺňať určité podmienky, aby po jej pridaní medzi ostatné bolo stále bezpečné spájať tieto komponenty do väčších celkov. Nech $S = \{p_1, p_2, \dots, p_n\}$ je množina primitív, ktoré môžme bezpečne používať na kompozíciu. To znamená, že každá kompozícia, ktorá vznikne z tejto množiny primitív, musí byť nedeštruktívna a nekonštruktívna. Nech p je primitíva, ktorá zabezpečuje utajenie g_1 a autorizáciu g_2. Utajenie je nedeštruktívny cieľ (musí platiť pre všetky kroky protokolu danej primitívy) a autorizácia je nekoštruktívny cieľ (môže nastať, len ak nastane špecifická udalosť alebo množina udalostí v rámci danej primitívy). Takáto primitíva, ktorá splní tieto dve požiadavky je vhodný kandidát na pridanie do množiny S. Prvou podmienkou, ktorú musí nová primitíva splniť, je podmienka súhlasu. Táto podmienka značí, že pre každý cieľ primitívy musí byť účastníkovi jasné, kto je jeho partnerom pre dosiahnutie daného cieľa. Keby účastník nemal partnera, mohol by cieľ splniť sám, čo by porušilo podmienku nekonštruktivity. Ďalšou podmienkou je, že nová primitíva musí byť regulárna. To znamená, že žiadne dlhodobé tajomstvo (napríklad šifrovací kľúč) sa nesmie použiť ako súčasť správy daného protokolu. Ďalšou podmienkou je diskrétnosť. Znamená to, že primitíva musí presne vedieť, s ktorým účastníkom zdieľa určité tajomstvo. Poslednou podmienkou, ktorú musí nová primitíva splniť, aby mohla byť pridaná medzi ostatné existujúce primitívy, je tá, že každý nonce (keksík) sa môže použiť na autorizáciu maximálne jedného účastníka.

Ciele, ktoré spĺňajú jednotlivé primitívy protokolu vzniknutého kompozíciou, musia byť navzájom nerozdeliteľné. Najjednoduchším spôsobom, ako zaistiť túto vlastnosť, je použiť nejaké unikátne číslo a pridať ho do správ všetkých primitív. V protokoloch, kde vystupujú len dva subjekty, je každý vygenerovaný term viditeľný pre oba subjekty. Preto v tom to prípade naozaj postačí pridať do správ takéto unikátne číslo. V prípade, že v protokole je zúčastnených viac subjektov, nebude takáto úprava postačovať.

Predpokladajme, že protokol $P = p_i \otimes p_j$ je kompozícia dvoch primitív $p_i(\vec{x}_i, \beta_{\vec{x}})$ a $p_j(\vec{y}_j, \beta_{\vec{y}})$ a $g_i(\vec{x}_i, \beta_{\vec{x}})$ a $g_j(\vec{y}_j, \beta_{\vec{y}})$ sú ciele, ktoré tieto primitívy naplňujú. Keďže protokol je usporiadaná množina primitív, musia aj ciele byť usporiadané. Predpokladajme, že $g_i(\vec{x}_i, \beta_{\vec{x}}) \leq g_j(\vec{y}_j, \beta_{\vec{y}})$. Aby oba ciele zaručene nastali v jednom behu protokolu, musí sa do tohto protokolu pridať tzv. väzobný prvok (z anglického výrazu „binder"), ktorý nesie tzv. väzobnú informáciu potrebnú na prepojenie daných dvoch primitív. Obecne sa táto informácia definuje ako zjednotenie parametrov oboch primitív. V tomto prípade by teda vyzerala nasledovne: $(\vec{x}, \vec{y}, \beta_{\vec{x}} \cup \beta_{\vec{y}})$. Ľubovoľný term, ktorý nesie túto informáciu, sa môže stať väzobným prvkom. Napríklad $\langle \vec{x}, \vec{y}, \beta_{\vec{x}} \cup \beta_{\vec{y}} \rangle$. V prípade, že je to potrebné, môže sa ľubovoľný term zahašovať, napríklad $(\langle \vec{x} \rangle, \vec{y}, \beta_{\vec{x}} \cup \beta_{\vec{y}})$, $(\vec{x}, \langle \vec{y} \rangle, \beta_{\vec{x}} \cup \beta_{\vec{y}})$ či $(\langle \vec{x} \rangle, \vec{y}, \langle \beta_{\vec{x}} \cup \beta_{\vec{y}} \rangle)$. Obecne sa preferuje použiť hašovanie, pretože sa tak bezpečne udržia utajené termy, ktoré majú byť utajené.

V prípade, že navrhujeme protokol, v ktorom vystupujú len dva subjekty, bude na previazanie primitív stačiť, aby práve jeden so subjektov pripojil k svojej správe väzobný prvok, ktorý je popísaný vyššie. Odporúča, aby to však nebola posledná správa tejto kompozície, pretože to môže za určitých podmienok znížiť bezpečnosť a zvýšiť chybovosť daného protokolu. Detailne sa týmto problémom zaoberá vo svojej práci autor metódy Choi [6].

Situácia je o niečo zložitejšia, ak navrhujeme protokol, kde je zúčastnených viac subjektov. V prípade, že chceme k dvom spojeným primitívam pripojiť tretiu, budeme musieť zjednotiť parameter tretej primitívy s väzobným prvkom prvých dvoch spojených primitív. Bližšie sa tomuto problému budem venovať pri implementácii knižnice, ktorá bude do určitej miery automatizovať túto funkcionalitu.

4.4 Návrh platobného protokolu

Táto kapitola popisuje základné obecné požiadavky na platobný protokol a ukazuje, ako je možné tento protokol implementovať metódou kompozície.

4.4.1 Bezpečnostné požiadavky na platobný protokol

Účastníci platobného protokolu sú typicky štyria: vydavateľ karty, nadobúdateľ, zákazník a obchodník. Na každého z nich sú kladené viaceré bezpečnostné požiadavky, ktoré je potrebné dodržať pri návrhu protokolu. Uvediem ich v nasledujúcich podkapitolách a v ďalšom texte sa ne budem odkazovať pri návrhu samotného protokolu.

4.4.1.1 Požiadavky vydavateľa karty a nadobúdateľa

Predpokladá sa, že vydavateľ platobnej karty a nadobúdateľ komunikujú medzi sebou na už existujúcej finančnej sieti, ktorá je z nášho pohľadu bezpečná. Preto obe entity budeme v ďalšom texte považovať za jednu a tak budeme definovať aj požiadavky, ktoré si tieto entity kladú.

N1: Dôkaz o tom, že transakcia bola autorizovaná zákazníkom. Ak chce nadobúdateľ zaúčtovať platbu konkrétnej kreditnej karte, musí mať jednoznačný dôkaz o tom, že majiteľ kreditnej karty platbu skutočne autorizoval. Tento dôkaz nesmie byť za žiadnych okolností opakovane použitý, napríklad na zaplatenie inej transakcie. Musí obsahovať minimálne zaplatenú čiastku, menu, popis tovaru, údaje o obchodníkovi a dodaciu adresu, prípadne spôsob doručenia tovaru.

N2: Dôkaz o tom, že transakcia bola autorizovaná obchodníkom. Ak chce nadobúdateľ autorizovať platbu danému obchodníkovi, musí mať jednoznačný dôkaz o tom, že tento obchodník o túto platbu sám požiadal.

4.4.1.2 Požiadavky obchodníka

O1: Dôkaz o tom, že transakcia bola autorizovaná nadobúdateľom (bankou). Obchodník potrebuje jednoznačný dôkaz o tom, že jeho banka (nadobúdateľ) autorizovala danú platbu. Tento dôkaz musí zahŕňať minimálne inkasovanú čiastku, údaj o dátume a čase a nejaký jednoznačný identifikátor transakcie. Väčšinou sa jedná o unikátne číslo.

O2: Dôkaz o tom, že zákazník autorizoval transakciu. Predtým než obchodník požaduje autorizáciu od nadobúdateľa, musí mať jednoznačný dôkaz o autorizácii transakcie zákazníkom.

4.4.1.3 Požiadavky zákazníka

Z1: Nie je možné vytvoriť neautorizovanú platbu. Kreditná karta zákazníka nesmie byť použitá na platbu bez toho, aby sa použil PIN kód, číslo kreditnej karty a tajný podpisový kľúč zákazníka. Aby to bolo možné, musí nadobúdateľ byť čestný a jeho tajný kľuč nesmie byť známy útočníkovi. Ďalej musí zákazník mať možnosť dokázať, že neautorizoval platbu, ak by náhodou útočník odhalil tajný kľúč nadobúdateľa.

Z2: Dôkaz o tom, že transakcia bola autorizovaná nadobúdateľom. Zákazník môže vyžadovať tento dôkaz. Nie je síce kritický, ale dá sa považovať za slušnosť.

Z3: Potvrdenie od obchodníka. Zákazník môže požadovať dôkaz o tom, že obchodník, ktorý mu tovar ponúkal, zinkasoval za neho platbu a prisľúbil jeho doručenie.

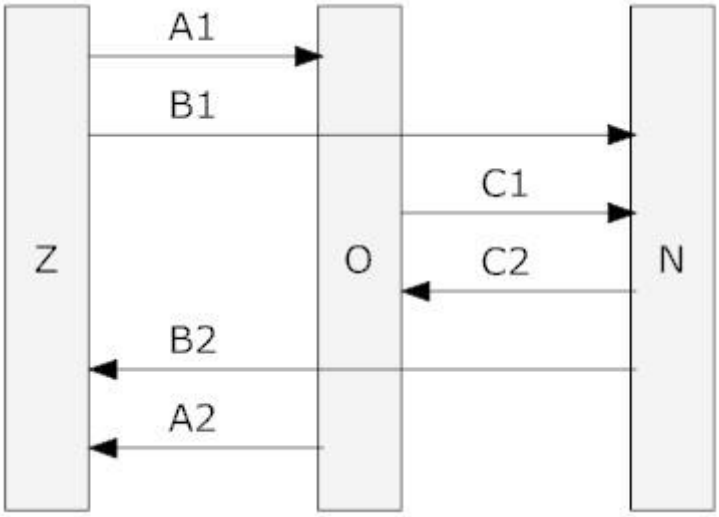
4.4.1.4 Voliteľné požiadavky

P1: Súkromie. Zákazník môže vyžadovať, aby nikto okrem zúčastnených strán nevedel, že niečo kupoval.

P2: Anonymita. Zákazník aj obchodník môžu požadovať anonymitu v platobnom styku ako ochranu pred sledovaním útočníkom.

4.4.2 Kompozícia protokolu

Protokol má troch účastníkov: obchodník O, zákazník Z a nadobúdateľ N. Obecne označíme účastníka symbolom U. Každý z týchto účastníkov disponuje dvomi pármi kľúčov: (ek_U, dk_U) na šifrovanie a dešifrovanie, (sk_U, vk_U) na podpisovanie a verifikáciu. Každý účastník vlastní k svojim verejným kľúčom aj odpovedajúce certifikáty $Cert(U, ek_U)$. Zákazník Z okrem toho samozrejme má kreditnú kartu a disponuje údajmi potrebnými pre autorizovanie platby.



Obrázok 2: Procedúra nákupu

Typický nákup môže prebiehať napríklad nasledovne: zákazník si prezerá tovar ponúkaný obchodníkom, napríklad na internetovej stránke a rozhodne sa niečo si zakúpiť. Pošle teda objednávku obchodníkovi. Informácie o tejto objednávke môže okrem toho zaslať aj nadobúdateľovi, aby neskôr mohol nadobúdateľ overiť korektnosť údajov, ktoré získa od obchodníka. Aby bola dodržaná požiadavka **P1**, nemôže obchodník objednávku autorizovať sám, takže ju pošle ďalej nadobúdateľovi. Ten na základe údajov, ktoré prijal predtým od zákazníka, platbu autorizuje a vygeneruje autorizačný kód. Ten potom pošle obchodníkovi a zákazníkovi. Obchodník dokončí objednávku zaslaním príslušného tovaru a potvrdenia zákazníkovi. Celú procedúru znázorňuje obrázok 2.

Celý problém môžeme rozdeliť na tri nezávislé protokoly, z ktorých každý má práve dvoch účastníkov: jeden medzi zákazníkom a obchodníkom (protokol A), druhý medzi zákazníkom a nadobúdateľom (protokol B) a nakoniec tretí medzi obchodníkom a nadobúdateľom (protokol C). Správy príslušných protokolov sú vyznačené v obrázku 2. Postupnosť správ môže byť aj iná. Napríklad správy C_1 a B_1 môžu byť poslané v náhodnom poradí. Rovnako tak správy B_2 a A_2.

Pri implementácii protokolu je potrebné voliť primitívy podľa toho, kde potrebujeme doručiť určité tajomstvo. V prípade protokolu A potrebuje zákazník Z tajne doručiť objednávku obchodníkovi O, ale obchodník zákazníkovi žiadne tajomstvo poslať nepotrebuje. Preto zvolíme nasledujúcu primitívu, kde U= (Z, O, N):

$$p_1: \quad A_1 \quad Z \rightarrow O: U, \{|\vec{x}_1, U|\}_{ek_O}, \ \langle \vec{x}_1, U \rangle_{sk_Z}$$
$$A_2 \quad O \rightarrow Z: Z, \langle \vec{x}_1, U \rangle_{sk_O}$$

Vektor $\vec{x}_1$ obsahuje údaje úzko spojené s objenávkou. Mal by obsahovať napríklad čiastku, dátum a čas, počet a druh tovaru, informácie o obchodníkovi, dodaciu adresu alebo meno. Aby bolo možné transakciu v budúcnosti vyhľadať, bude $\vec{x}_1$ obsahovať aj unikátny identifikátor objednávky, napríklad číslo transakcie. Nech $T = (\vec{x}_1, U)$ je popis objednávky, ktorý obsahuje všetky tieto informácie. Primitíva p_1 – protokol A:

$$p_1: \quad A_1 \quad Z \rightarrow O: U, \{|T|\}_{ek_O}, \ \langle T \rangle_{sk_Z}$$
$$A_2 \quad O \rightarrow Z: Z, \langle T \rangle_{sk_O}$$

Situácia pri tvorbe protokolu B je obdobná ako v predchádzajúcom prípade. Zákazník potrebuje poslať tajomstvo (údaje o objednávke, prípadne nejaké extra informácie) nadobúdateľovi a ten mu v odpovedi už žiadne tajomstvo neposiela.

$$p_2: \quad B_1 \quad Z \rightarrow N: U, \{|\vec{y}_2, U|\}_{ek_N}, \ \langle \vec{y}_2, U \rangle_{sk_Z}$$
$$B_2 \quad N \rightarrow Z: Z, \langle \vec{y}_2, U \rangle_{sk_N}$$

Údaje vo vektore $\vec{y}_2$ sú úzko spojené s objednávkou a platbou. Obsahujú okrem indentifikačných údajov aj číslo kreditnej karty, PIN kód a dátum expirácie karty. Nech $P = (\vec{y}_2, \mathrm{U})$, potom bude primitíva implementujúca protokol B nasledovná:

$$p_2: \quad B_1 \quad Z \rightarrow N: U, \{|P|\}_{ek_N}, \ \langle P \rangle_{sk_Z}$$
$$B_2 \quad N \rightarrow Z: Z, \langle P \rangle_{sk_N}$$

Aby bolo možné tieto primitívy skombinovať, musí sa do oboch pridať väzobný prvok, ktorému by obaja účastníci O a N rozumeli a na základe ktorého by si mohli priradiť danú správu do správneho protokolu . Spojovací prvok S môže mať v tomto prípade napríklad tvar $S_1 = (\langle P \rangle, \langle I \rangle)$. Vyžaduje si to však zaslanie ďalších termov príslušným účastníkom. Výsledná kompozícia bude nasledovná:

$$p \quad A_1 \quad Z \rightarrow O: U, \{|T|\}_{ek_O}, \ \langle T \rangle_{sk_Z}, \langle S_1 \rangle_{sk_Z}, \langle P \rangle_{sk_Z}$$
$$B_1 \quad Z \rightarrow N: U, \{|P|\}_{ek_N}, \ \langle P \rangle_{sk_Z}, \ \langle S_1 \rangle_{sk_Z}, \langle T \rangle_{sk_Z}$$
$$B_2 \quad N \rightarrow Z: Z, \langle P \rangle_{sk_N}$$
$$A_2 \quad O \rightarrow Z: Z, \langle T \rangle_{sk_O}$$

Posledný protokol, C, môže byť postavený na primitíve, kde $\vec{z}_3$ je nonce:

$$p_3: \quad C_1 \quad O \rightarrow N: U, \{|\vec{z}_3, U|\}_{ek_N}, \ \langle \vec{z}_3, U \rangle_{sk_O}$$
$$C_2 \quad N \rightarrow O: O, \langle \vec{z}_3, U \rangle_{sk_N}$$

Aby sme mohli prepojiť primitívy p_2 a p_3, opäť musíme zaviesť do protokolu určitý spojovací prvok, ktorý bude známy účastníkom O a N. Kedže P musí byť utajené pred O a T musí byť utajené pred N, spojovací prvok bude $S_2 = (\langle T\rangle, \langle P\rangle, \vec{z}_3)$. Vznikne protokol:

$$
\begin{array}{lll}
p & A_1 & Z \to O\colon U, \{|T|\}_{ek_O},\ \langle T\rangle_{sk_Z}, \langle S_1\rangle_{sk_Z}, \langle P\rangle_{sk_Z} \\
 & B_1 & Z \to N\colon U, \{|P|\}_{ek_N},\ \langle P\rangle_{sk_Z},\ \langle S_1\rangle_{sk_Z}, \langle T\rangle_{sk_Z} \\
 & C_1 & O \to N\colon U, \{|\vec{z}_3, U|\}_{ek_N},\ \langle \vec{z}_3, U\rangle_{sk_O}, \langle S_2\rangle_{sk_O} \\
 & C_2 & N \to O\colon O, \langle \vec{z}_3, U\rangle_{sk_N} \\
 & B_2 & N \to Z\colon Z, \langle P\rangle_{sk_N} \\
 & A_2 & O \to Z\colon Z, \langle T\rangle_{sk_O}
\end{array}
$$

Vďaka S_2 vie obchodník O aj nadobúdateľ N, že $p_3(\vec{z}_3, U)$ patrí k rovnakému sedeniu ako $p_1(T)$ a $p_2(P)$. Z túto informáciu nepotrebuje.

V kapitole 4.3.1 som uviedol požiadavky jednotlivých účastníkov komunikácie. Teraz uvediem, akým spôsobom boli naplnené.

N1: implementované pomocou $\langle S_1\rangle_{sk_Z}$
N2: implementované pomocou $\langle S_2\rangle_{sk_O}$
M1: zatiaľ neimplementované
M2: implementované pomocou $\langle S_1\rangle_{sk_Z}$
C1: implementované pomocou $\langle S_1\rangle_{sk_Z}$
C2: implementované pomocou $\langle P\rangle_{sk_N}$
C3: implementované pomocou $\langle T\rangle_{sk_O}$

Aby bola splnená aj podmienka **M1**, musíme správu trochu upraviť:

$$\langle \vec{z}_3, U\rangle_{sk_N} \Longrightarrow \langle \langle T\rangle, \{|P|\}_{ek_N}, \langle \vec{z}, U\rangle\rangle_{sk_A}$$

Požiadavka **P1** je implementovaná samotným protokolom a požiadavka **P2** sa naplní napríklad vtedy, keď zákazník použije namiesto pravého mena pseudonym, ktorý bude známy len banke (nadobúdateľovi a vydavateľovi platobnej karty). Získaný protokol je ešte možné optimalizovať odstránením niektorých redundantných prvkov a spojením iných prvkov. Výsledný protokol implementuje nákupnú procedúru protokolu SET a bude mať tvar:

$$
\begin{array}{lll}
p' & M_1 & Z \to O\colon U, \{|T|\}_{ek_O},\ \{|P|\}_{ek_N}, \langle S_1\rangle_{sk_Z}, \langle T\rangle_{sk_Z}, \langle P\rangle_{sk_Z} \\
 & M_2 & O \to N\colon U, \{|P|\}_{ek_N},\ \{|\vec{z}_3, U|\}_{ek_N},\ \langle S_1\rangle_{sk_Z}, \langle T\rangle_{sk_Z}, \langle S_2\rangle_{sk_O} \\
 & M_3 & N \to O\colon O, \langle \langle T\rangle, \{|P|\}_{ek_N}, \langle \vec{z}, U\rangle\rangle_{sk_A}, \langle P\rangle_{sk_N} \\
 & M_4 & O \to Z\colon Z, \langle T\rangle_{sk_O}, \langle P\rangle_{sk_N}
\end{array}
$$

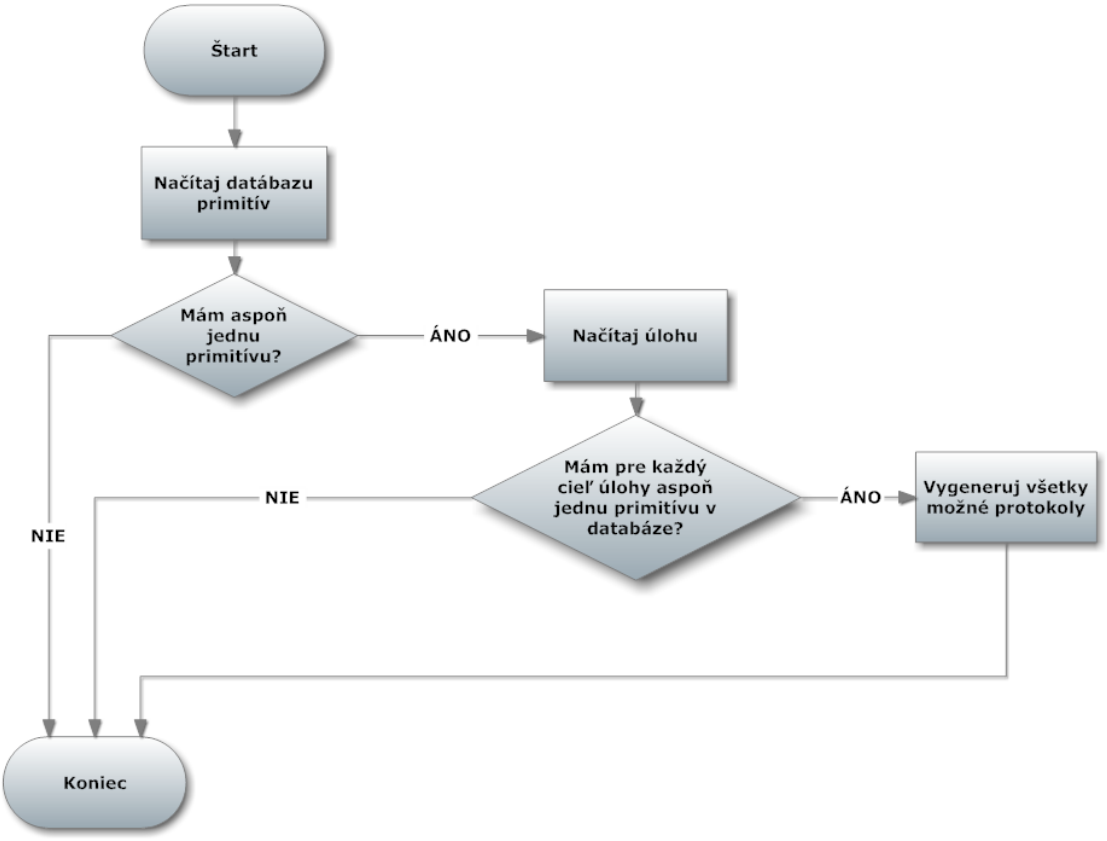
5 Implementácia

V tejto kapitole detailne popíšem návrh a implementáciu knižnice pre podporu návrhu protokolov metódou kompozície, naznačím hlavnú myšlienku a ciele implementácie, špecifikujem formát vstupu a výstupu a podrobne rozoberiem návrh samotnej knižnice.

5.1 Základná myšlienka a ciele

V prvom rade je dôležité si uvedomiť, že v praxi je plne automatický návrh protokolu pomerne ťažko realizovateľný. Návrhárovi protokolu ale uľahčí prácu aj nástroj, ktorý mu na základe jeho požiadaviek dokáže pripraviť aspoň približný návrh či niekoľko návrhov, ktoré by mu poslúžili ako základ pri navrhovaní určitého protokolu. Preto hlavným cieľom môjho úsilia bude program, ktorý dokáže na základe zadaných cieľov metódou kompozície poskladať zo základných komponentov (primitív) požadovaný protokol, alebo aspoň jeho čo najpresnejší návrh. Vzhľadom k tomu, že pri spájaní primitív dohromady existuje v podstate vždy viac možností, ako tieto komponenty spojiť, bude výsledných protokolov samozrejme viac. Toto množstvo ovplyvní aj počet základných primitív, ktoré pri kompozícii použijeme. Čím viac ich bude, tým viac kombinácií môžeme získať.

Idea je teda taká, že návrhár určitým spôsobom špecifikuje svoje požiadavky na protokol, ktorý chce navrhnúť a predá ich programu na vstupe. Ten z dostupnej databázy stavebných prvkov (primitív) vyberie všetky tie, ktoré splnia dané podmienky a poskladá z nich metódou kompozície všetky možné protokoly. Výstupom bude potom množina protokolov. Celý princíp znázorňuje diagram na obrázku 3.



Obrázok 3: Workflow diagram navrhovaného programu

Na vstupe program dostane čo najpresnejšie špecifikované požiadavky na hľadaný protokol a databázu primitív. Ako prvý krok sa spracuje databáza primitív. V prípade, že máme aspoň jeden prvok v databáze, pokračuje sa ďalej a spracuje sa zadanie. Následne prebehne overenie, že pre každý cieľ, ktorý je v úlohe definovaný, existuje aspoň jedna primitíva v databáze. V prípade, že tomu tak nie je, nemá cenu pokračovať ďalej, pretože nie je možné splniť všetky ciele. Ak ale tieto primitívy existujú, program pokračuje a vygeneruje všetky možné protokoly na základe metódy kompozície. Tieto sú potom predané návrhárovi na výstupe. Všetky etapy, ktoré som práve spomenul, podrobnejšie rozoberiem v ďalších kapitolách.

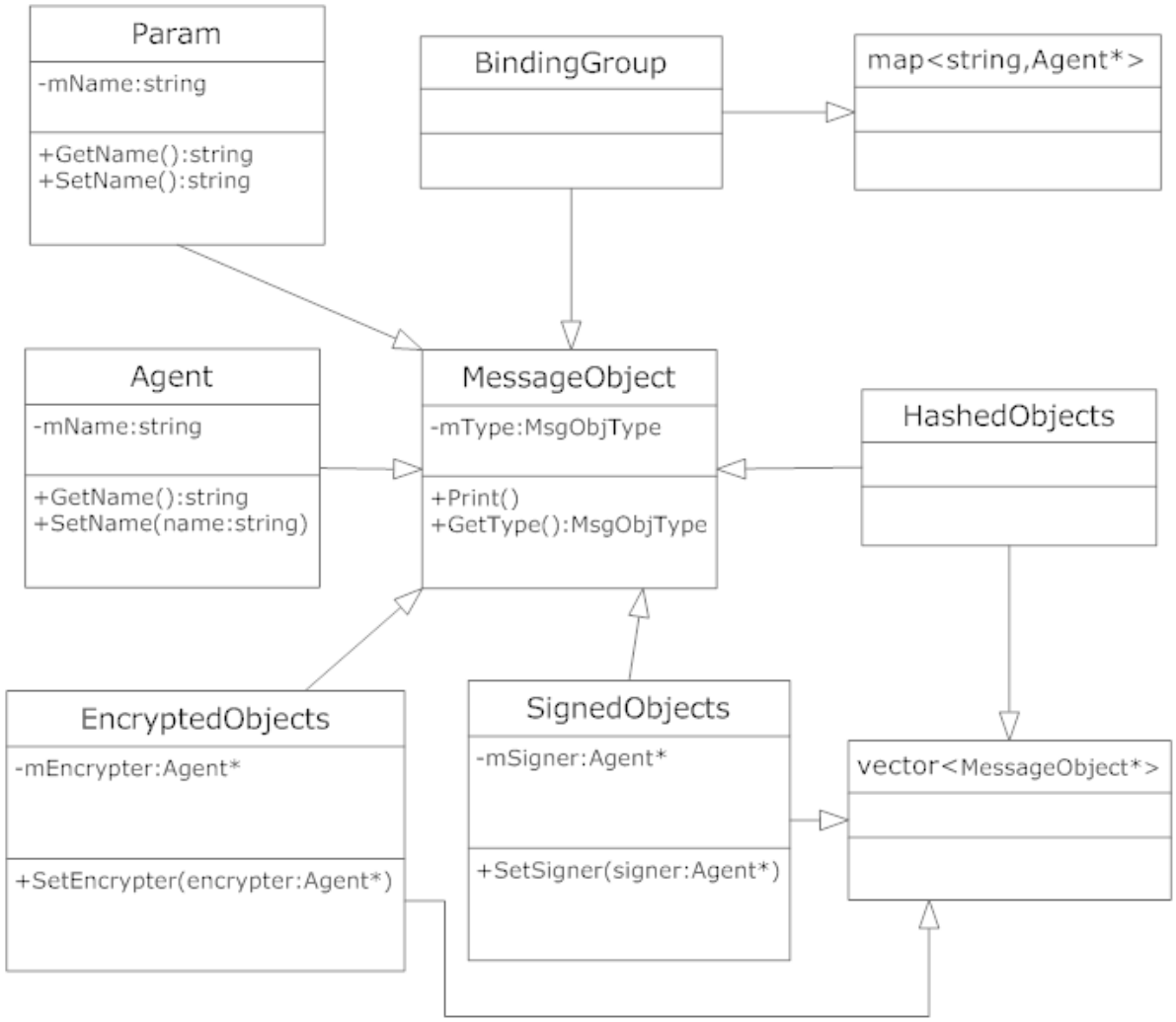
Ďalším prínosom pre návrhára by bol výstup, ktorý by mohol jednoducho použiť ako vstup do ďalších nástrojov používaných na prácu s protokolmi ich verifikáciu a podobne. Preto ďalším mojim cieľom bude použiť výstupný formát, ktorý sa bežne používa na zápis protokolov a to jazyk CAS+. Bližšie o tomto formáte pojednáva kapitola 5.5.

5.2 Databáza primitív

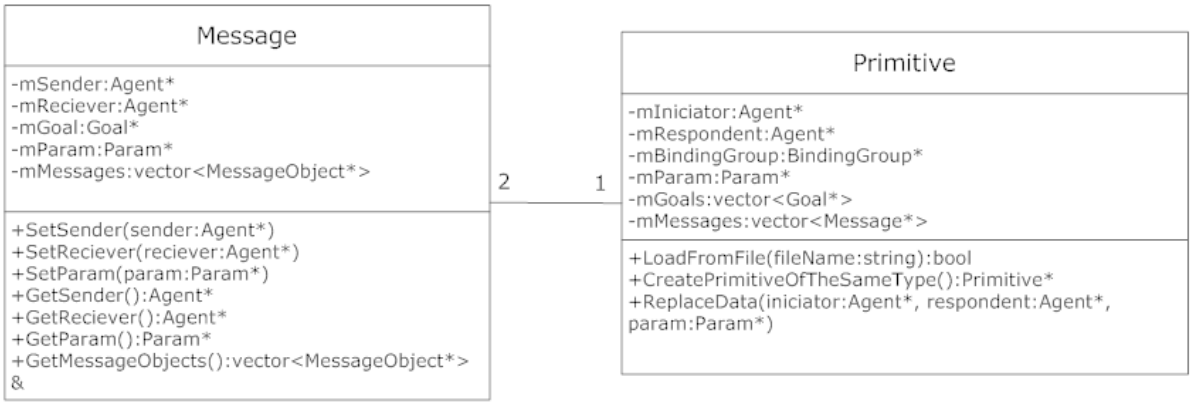
Základom metódy kompozície je databáza primitív, z ktorých sa budú skladať zložitejšie protokoly. Preto aj základom môjho programu bude práve takáto databáza.

Pri navrhovaní tejto komponenty je dôležité si uvedomiť, čo o každej primitíve potrebujem vedieť a čo takáto primitíva bude obsahovať. V tomto prípade môžem s určitosťou povedať, že každá primitíva bude popisovať komunikáciu medzi práve dvomi subjektmi (iniciátor a respondent) a bude obsahovať práve jeden parameter (dáta, ktoré pošle iniciátor respondentovi a ten ho následne na základe týchto dát autorizuje). Ďalšia vec, ktorú určite potrebujem o primitíve vedieť, sú ciele, ktoré splňuje. Na základe tejto vlastnosti budem totiž hľadať vhodné primitívy pri kompozícii protokolu. Cieľov, ktoré splňuje jedna primitíva, je obvykle toľko, koľko je v nej správ. Inak povedané, každá správa plní určitý cieľ. Správy sú súčasne posledná vec, ktorú potrebujem ku každej primitíve uložiť.

Na základe princípov, ktoré som popísal v kapitolách 4.2 a 4.3 a ako ich popísal Choi vo svojej práci [6], má každá primitíva parameter. Preto aj môj návrh smeroval k tomu, aby som tento parameter mohol jednoducho meniť a dosadiť za neho príslušné hodnoty na základe protokolu, ktorý budem hľadať. Ďalším dôležitým faktom je, že súčasťou zaslanej správy môže byť akákoľvek entita – parameter a jeho väzobná skupina, subjekty, haše a šifrované alebo podpísané objekty. Hierarchiu tried, ktorá popisuje tieto entity, znázorňuje obrázok 4. Všetky entity, ktoré sa v správe vyskytujú, majú spoločného predka, ktorým je trieda `MessageObject`. Tie, ktoré v sebe môžu obsahovať vnorené ďalšie entity, ako napríklad šifrované (`EncryptedObjects`) alebo podpísané objekty (`SignedObjects`), majú ako ďalšieho predka ešte štandardný kontajner príslušného typu. Triedy popisujúce subjekty a parametre (`Agent` a `Param`) majú okrem zdedených vlastností ešte ďalší atribút navyše a to svoje meno. Väzobná skupina `BindingGroup` je množina subjektov, ku ktorým sa viaže parameter primitívy. Šifrované objekty majú ako ďalší atribút subjekt, ktorý objekty zašifroval a podpísané objekty majú ako ďalší atribút subjekt, ktorý ich podpísal. Každá správa teda bude obsahovať odosielateľa, prijímateľa, množinu objektov, ktoré sú predmetom správy a nakoniec cieľ, ktorý daná správa plní. Definície tried reprezentujúcich primitívu a správu sú popísané na obrázku 5. Primitíva má práve dve správy a každá správa patrí do práve jednej primitívy. Pre lepšiu čitateľnosť som v triedach neuviedol všetky metódy, len niektoré základné. Bežnou súčasťou takmer každej triedy sú metódy na nastavovanie a získavanie privátnych premenných (metódy `Set()` a `Get()`).



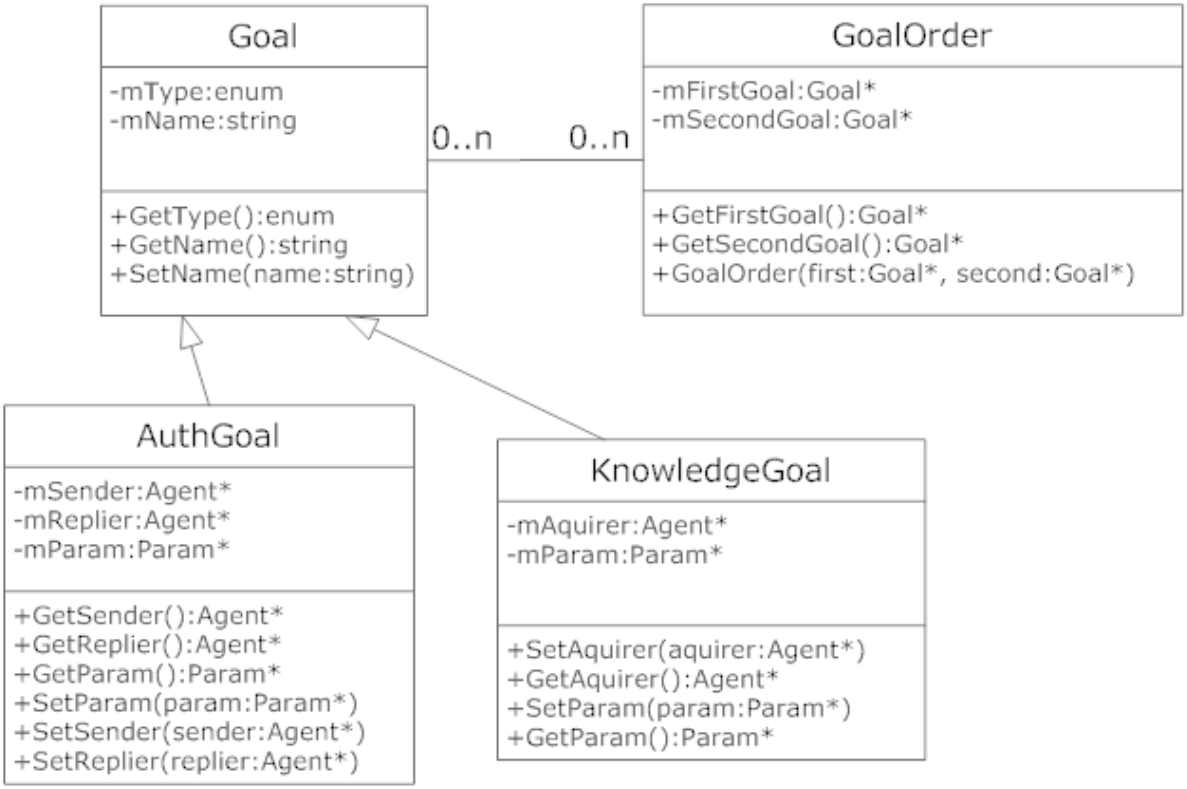
Obrázok 4: Diagram tried prvkov správy



Obrázok 5: Diagram tried primitívy a správy

V rámci celého programu budem uvažovať dva typy cieľov. Prvý vyjadruje ten cieľ, že určitý subjekt nadobudne určitú vedomosť a druhý, že určitý subjekt bol autorizovaný druhým subjektom na základe dát ktoré mu poslal. Štruktúru navrhnutých tried popisuje obrázok 6. Každý cieľ má svoje jedinečné pomenovanie a v závislosti od typu cieľa potom aj ďalšie atribúty. V prípade autorizačného cieľa to sú oba zúčastnené subjekty a príslušný parameter a v prípade znalostného cieľa je to subjekt, ktorý znalosť nadobudne a potom parameter reprezentujúci danú znalosť. Na tomto obrázku je ešte

trieda, pomocou ktorej budem pracovať s poradím cieľov (`GoalOrder`). Tá bude potrebná pre špecifikovanie poradia cieľov v zadaní úlohy, ktorú bude program riešiť. Poradie cieľov nemusí mať definované žiadne ciele (na poradí vôbec nemusí záležať) a cieľ nemusí byť súčasťou nejakého poradia.



Obrázok 6: Diagram tried cieľov

Kvôli prehľadnosti som sa rozhodol, že každá primitíva bude v samostatnom súbore a programu sa na vstup dodá iný súbor, v ktorom nebude nič iné, len zoznam primitív patriacich do tejto databázy. Takto bude možné jednoduchým spôsobom možné vymieňať databázy primitív, prípadne jednotlivé primitívy, ktoré chceme v danom momente použiť. Formát, v akom som sa rozhodol primitívy ukladať, má textovú podobu a je veľmi jednoduchý. Vysvetlím ho na príklade nasledujúcej primitívy:

$A \rightarrow B$: $B, \{|x, \beta_x|\}_{ek_B}, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle$

Prepísaná do formátu, ktorý budem používať v implementácii, bude mať tvar:

```
# primitive type1
[goals]
g1: knowledge: resp=par
g2: auth: init= resp: par

[messages]
init->resp: resp, {par,b_grp}EK_resp, <par,b_grp>SK_init; g1
resp->init: init, <par,b_grp>; g2
```

Do príkladu som zámerne pridal aj ukážku komentáru, ktorý budem vo všetkých mojich súboroch používať a bude ním každý riadok, ktorý začína špeciálnym znakom #. V hranatých zátvorkách sú označené začiatky sekcií, v ktorých sú definované ciele a správy. Komunikujúce subjekty sú označené ako `init` (začína komunikáciu) a `resp` (odpovedá), parameter primitívy ako `par` a jeho väzobná skupina ako `b_grp`. Definícia cieľa začína vždy jeho unikátnym názvom, pokračuje typom a končí cieľom samotným. V tomto prípade bude prvý cieľ vyjadrovať to, že respondent sa dozvie, alebo iným slovom naučí, parameter, ktorý dostane od iniciátora komunikácie. Druhý cieľ potom vyjadruje, že respondent autorizuje iniciátora na základe parametru. Pri každej primitíve budem predpokladať, že každý účastník komunikácie disponuje dvomi pármi kľúčov – šifrovací a dešifrovací kľúč a podpisový a verifikačný kľúč. Šifrované správy určené pre iniciátora (resp. respondenta) sú v zložených zátvorkách, ktoré sú nasledované symbolom `EK_init` (resp. `EK_resp`) a správy podpisované iniciátorom (resp. respondentom) sú v ostrých zátvorkách, za ktorými nasleduje symbol `SK_init` (resp. `SK_resp`). Objekty, ktoré boli zahašované, sú v ostrých zátvorkách bez ďalšieho označenia. Správa je potom definované odosielateľom a prijímateľom, za nimi nasleduje zoznam objektov danej správy a na konci za bodkočiarkou je cieľ, ktorý daná správa plní.

Trieda, ktorá zahŕňa celú databázu primitív, je vo svojej podstate veľmi jednoduchá. Nesie názov `PrimitiveDatabase` a objekt tejto triedy poskytuje okrem načítaných primitív ešte metódy na vyhľadávanie primitív podľa zadaného cieľa.

5.3 Požiadavky na výsledný protokol

V tejto kapitole popíšem, akým spôsobom sa budú definovať požiadavky na protokol, ktorý chceme navrhnúť a rozoberiem jednotlivé detaily tohto popisu.

Predtým, než návrhár začne navrhovať určitý komunikačný protokol, musí mať čo najlepšiu predstavu o cieľoch daného protokolu, o znalostiach jednotlivých subjektov komunikácie a o bezpečnostných požiadavkách na tento protokol. Aby mnou napísaný program dokázal čo najviac uľahčiť prácu potenciálnemu návrhárovi, musí mu umožniť zadefinovať jeho požiadavky čo najpríjemnejším a čo najjednoznačnejším spôsobom. Zvolil som textový formát, ktorý je štruktúrou v podstate zhodný s formátom na ukladanie primitív. V popise požiadaviek na protokol ale samozrejme budú iné sekcie, ktoré odpovedajú odpovedajúcim údajom potrebným na špecifikovanie protokolu. Budú to údaje o komunikujúcich subjektoch a ich počiatočných znalostiach, o cieľoch hľadaného protokolu a o poradí týchto cieľov. Predpokladajme, že chceme navrhnúť úplne jednoduchučký protokol, ktorý bude mať len dvoch účastníkov a základné ciele. Nech A a B sú subjekty, p je parameter, ktorý je zároveň počiatočnou znalosťou subjektu A a nech cieľom bude, aby subjekt B získal znalosť o parametri p a aby autorizoval subjekt A na základe tohto parametru. Popis požiadaviek na takýto protokol bude nasledovný:

```
[agents]
A, B

[knowledge]
A: p

[goals]
g1: knowledge: B = p
```

```
g2: auth: A = B : p

[goal_order]
g1<g2
```

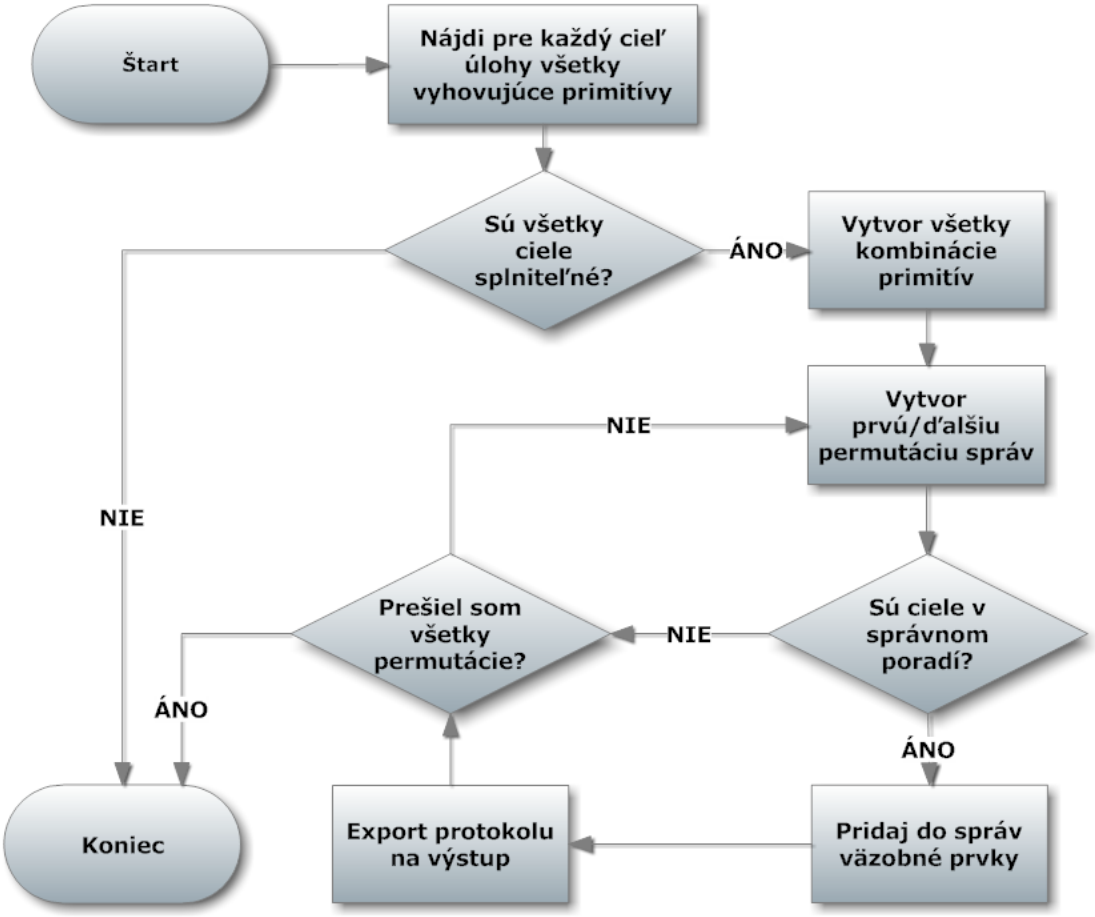
Prvá sekcia definuje komunikujúce subjekty, druhá definuje ich počiatočné znalosti, tretia ciele rovnakým spôsobom, ako tomu bolo pri primitívach a nakoniec posledná sekcia definuje poradie, v akom ciele musia nastať. V tomto prípade požadujeme, aby cieľ `g1` nastal pred cieľom `g2`. Týmto spôsobom je možné veľmi presne špecifikovať hľadaný protokol a znížiť tak počet kandidátov, ktorých program vygeneruje. Protokoly vhodné na platobné účely a použiteľné v tejto oblasti budú samozrejme zložitejšie a budem sa nimi zaoberať v kapitole 6.

Trieda, ktorá pokrýva túto oblasť, nesie názov `Task` a nie je ničím zložitá. Objekt tejto triedy drží v sebe údaje o definovanej úlohe – mená subjektov, ich znalosti, ciele, ktoré sa majú splniť a poradie týchto cieľov.

5.4 Generovanie protokolov

Máme k dispozícii databázu primitív, máme zadanú úlohu a môžeme sa teda pustiť do generovania samotných protokolov. Implementáciu tohto procesu objasní práve táto kapitola.

Jadrom celej implementácie je metóda kompozície a teda vytváranie zložitých protokolov spájaním jednoduchých častí. Tento proces je znázornený diagramom na obrázku 7 a mohli by sme ním nahradiť časť generovania protokolov, ktorá je v diagrame na obrázku 3.



Obrázok 7: Workflow diagram procesu generovania protokolov

Prvým krokom po inicializácii databázy primitív a načítaní úlohy, ktorú chceme kompozíciou vyriešiť, je overenie, že pre každý cieľ, ktorý je v úlohe zadefinovaný, máme v databáze aspoň jednu primitívu, čo ho dokáže naplniť. V každej z nich sú následne identifikátory subjektov a parameter nahradené za tie, ktoré sú v zadaní úlohy. V prípade, že čo i len jeden cieľ nie je možné splniť, program okamžite končí a v generovaní protokolov sa nepokračuje. Ak je všetko v poriadku a každý cieľ je možné splniť, proces pokračuje ďalej.

Druhým krok je vytvoriť z primitív, ktoré sme získali v prvom kroku, všetky kombinácie. Predpokladajme, že máme v databáze dve primitívy a v úlohe zadefinujeme tri ciele. Ďalej predpokladajme, že každý tento cieľ môžeme splniť oboma primitívami z databázy. Vznikne nám teda $2^3 = 8$ kombinácií. Tieto kombinácie sú základom pre generovanie výstupných protokolov.

Ďalším krokom je vytvorenie všetkých permutácií správ, ktoré môžeme získať z daných kombinácií primitív. Každá primitíva, ktorú používam v mojej implementácii, má práve dve správy. Ak nadviažem na príklad z predchádzajúceho odstavca, tak pri ôsmich kombináciách troch primitív, kde každá má dve správy, bude všetkých permutácií správ $8.(3.2)! = 5760$. Samozrejme ani zďaleka nie všetky permutácie budú vyhovovať zadanej úlohe.

V nasledujúcej fáze sa postupne prechádzajú vytvorené permutácie správ a vyberajú sa z nich tie, ktoré vyhovujú danému zadaniu. Ako som popísal v kapitole 5.2, každá správa v primitíve splňuje určitý cieľ. V zadaní úlohy, ktoré som popísal v kapitole 5.3 som vysvetlil, ako sa definuje poradie cieľov, ktoré chceme vo výslednom protokole dodržať. Preto sa v tomto kroku procesu každá permutácia správ, ktorá neplní ciele v danom poradí, zahodí a generuje sa ďalšia.

Ak máme permutáciu, ktoré ma ciele v správnom poradí, môžeme ju považovať za výsledok a pustiť sa do spájania jednotlivých správ dohromady. V terminológii metódy kompozície to znamená pridať do každej primitívy väzobný prvok. V kapitole 4.3 som vysvetlil, že vzniká zjednotením parametrov tých dvoch primitív ktoré chceme spojiť. Toto platí v prípade, že spájame dohromady dve primitívy. V prípade, že chceme k dvom spojeným primitívam pripojiť tretiu, budeme musieť zjednotiť parameter tretej primitívy s väzobným prvkom prvých dvoch spojených primitív. Prvým krokom je vždy spojenie prvých dvoch primitív, kde postačí ako väzobný prvok zjednotenie parametrov. Následne vždy pripojujeme postupne ďalšie a ďalšie primitívy a väzobným prvkom bude zjednotenie predchádzajúceho väzobného prvku a parametru novej primitívy. Princíp bude prezentovaný na príkladoch v kapitole 6.

Každá permutácia, ktorá splnila všetky podmienky zadané v úlohe a boli do jej správ vložené väzobné prvky na previazanie primitív dohromady, je nakoniec exportovaná vo na výstup, ktorý je v prípade mojej knižnice súbor a jeho formát popisuje nasledujúca kapitola.

Proces, ktorý som popísal v tejto kapitole, implementačne pokrýva trieda s názvom `Composer`. Má ako členské premenné databázu primitív (`PrimitiveDatabase`) a úlohu, ktorú bude plniť (`Task`). Zavolaním metódy `GenerateProtocols` sa vygenerujú všetky čiastkové protokoly, ktoré sa následne rozvinú do všetkých permutácií v metóde `ExportResults` a vyexportujú na výstup.

5.5 Formát výstupu

Ako už napovedá názov tejto kapitoly, budem sa v nej zaoberať formátom výstupu programu, ktorý bude implementovať podporu návrhu protokolov metódou kompozície.

V kapitole 5.1 som naznačil, že mojim cieľom je použiť formát, ktorý by bol kompatibilný s ďalšími dostupnými nástrojmi na prácu s komunikačnými protokolmi. Vhodným kanditátom, ktorý tieto požiadavky splňuje je jazyk CAS+, ktorý predstavili vo svojej práci R. Saillard a T. Genet [7].

Vzhľadom k tomu, že výsledkom bude vždy pomerne početná množina protokolov, rozhodol som sa, že nebudem ukladať každý z nich do samostatného súboru, ale uložím všetky za sebou do jedného súboru oddelené tak, aby to bolo na prvý pohľad pochopiteľné pre užívateľa. Protokoly budú popísané jazykom CAS+, takže budú pripravené na prípadné ďalšie spracovanie.

Uvažujme príklad z kapitoly 5.3. Výstupný súbor by mohol vyzerať napríklad nasledovne:

```
protocol 1; % public key
identifiers
A, B : user;
p : number;
EK_A, SK_A, EK_B, SK_B : public_key;
Hash : function;

messages
1. A -> B: B,{p,A,B}EK_B,{p,A,B}SK_A'
2. B -> A: A,Hash(p,A,B)

knowledge
A: B, p, EK_A, SK_A, EK_B, Hash;
B: A, EK_B, SK_B, EK_A, Hash;

session_instances
[A:A_inst, B:B_inst];

intruder_knowledge

goal
secrecy_of p [B,A];
A authenticates B on p

#####################################################

protocol 2; % public key
identifiers
A, B : user;
p : number;
EK_A, SK_A, EK_B, SK_B : public_key;
Hash : function;

messages
A -> B: p,A,B,{p,A,B}SK_A'
B -> A: A,{p,A,B}SK_B'

knowledge
A: B, p, EK_A, SK_A, EK_B, Hash;
B: A, EK_B, SK_B, EK_A, Hash;

session_instances
```

```
[A:A_inst, B:B_inst];

intruder_knowledge

goal
secrecy_of p [B,A];
A authenticates B on p
```

Tento výstup bol získaný použitím databázy, ktorá obsahovala len dve primitívy. Čitateľa by mohli na prvý pohľad zaujať určité fakty, ktoré sa vo výstupnom protokole objavili a ktoré by som mal objasniť. V prvom rade je to predpoklad, že každý subjekt pozná svoje dva páry kľúčov. Jeden pár na šifrovanie (verejný) a dešifrovanie (privátny) a druhý pár na podpisovanie (privátny) a verifikáciu (verejný). Preto sa tieto kľúče objavili aj v definícii identifikátorov aj v definícii počiatočných znalostí výsledného protokolu. A druhým faktom je, že predpokladám, že každý subjekt pozná a používa jednu rovnakú hašovaciu funkciu. Táto je tak isto uvedená aj v definícii identifikátorov aj v počiatočných znalostiach subjektov.

Vzhľadom k tomu, že tento príklad je triviálne jednoduchý, je možné ho implementovať len jednou primitívou a v podstate ku kompozícii ani nedošlo. V tomto prípade som sa ale snažil ukázať formát výstupu. Zložitejšie príklady predvediem pri návrhu platobných protokolov v kapitole 6. Bližší popis výstupného formátu a teda jazyka CAS+ zahŕňa nasledujúca kapitola.

5.5.1 Jazyk CAS+

Tento jazyk popisuje komunikačný protokol v šiestich častiach. V nich sú postupne deklarované identifikátory (komunikujúce subjekty, kľúče, funkcie a posielané dáta), správy, počiatočné znalosti subjektov, inštancie jednotlivých identifikátorov, znalosti útočníka a ciele.

V prvej časti, označovanej kľúčovým slovom `identifiers`, sú definované všetky identifikátory, ktoré v protokole nejakým spôsobom vystupujú. Jazyk definuje päť typov týchto identifikátorov – užívateľ (`user`), symetrický kľúč (`symmetric_key`), verejný kľúč (`public_key`), funkcia (`function`) a číslo (`number`). Typ číslo neznamená, že sa jedná len o číslo, ale je to označenie obecne pre akékoľvek užitočné dáta, ktoré sa budú v rámci protokolu posielať. Typ funkcia označuje jednocestnú hašovaciu funkciu. To znamená, že nedokážeme z hašu $F(X)$ spätne získať prvok X. V prípade verejného kľúča K sa jeho súkromná časť (privátny kľúč) označuje ako K'.

Druhá sekcia je definovaná kľúčovým slovom `messages`. Obsahuje zoznam posielaných správ vo formáte $(i.S_i \rightarrow_i R_i : M_i)_{1 \leq i \leq n}$, kde i je číslo kroku, S_i je odosielateľ, R_i je príjemca a M_i je množina identifikátorov definovaných v prvej časti protokolu. Pre zápis šifrovania pomocou kľúčov sa používa zápis $\{A\}_K$, kde A je množina objektov zašifrovaná kľúčom K. Pre zápis hašovacej funkcie sa používa zápis $F(A)$, čo značí haš množiny objektov A spočítaný pomocou hašovacej funkcie F. V rámci tejto časti sa v jazyku CAS+ používajú ešte ďalšie funkcie a konštrukcie, ale v rámci tejto práce si vystačím s vyššie uvedenými.

Tretia časť, označovaná kľúčovým slovom `knowledge`, definuje počiatočné znalosti všetkých komunikujúcich subjektov. Sú tu všetkým subjektom priradené všetky tie identifikátory definované v prvej časti, ktoré sú tomuto subjektu známe pred začiatkom protokolu. Predpokladá sa, že každý subjekt pozná sám seba, takže sa v tomto zozname neuvádza.

V časti `session_instances` sú uvedené inštancie jednotlivých identifikátorov z prvej časti. Jedná sa o podobný princíp ako triedy a objekty v objektovom programovaní. V prvej časti sú deklarované identifikátory (triedy) a v tejto časti inštancie (objekty).

Časť `intruder_knowledge` je podobná časti `knowledge`, avšak popisuje znalosti potenciálneho útočníka.

A na koniec časť `goals` definuje ciele, ktoré by mal daný protokol splniť a ktoré chceme verifikovať. Jazyk CAS+ definuje tri typy cieľov – utajenie (`secrecy`), autorizáciu (`authentication`) a slabú autorizáciu (`weak authentization`). Utajenie sa vzťahuje k jednému identifikátoru a množine užívateľov, ktorý musia byť deklarovaný v prvej časti protokolu. Identifikátor značí hodnotu, ktorá má byť utajená a užívatelia uvedený v tomto cieli sú tí, ktorí majú právo sa toto tajomstvo naučiť. Autorizácia sa vzťahuje k dvom užívateľom a identifikátoru a to tak, že prvý užívateľ autorizuje druhého na základe hodnoty identifikátoru.

5.6 Programovací jazyk a použitie programu

Celý projekt som vyvíjal v programovacom jazyku C++ v prostredí Microsoft Visual Studia 2008. Nie sú v ňom použité žiadne externé knižnice. V rámci testovania a ladenia som použil aj školský server Merlin, kde bol v čase implementovania nainštalovaný kompilátor g++ (GCC) vo verzii 4.4.6. Program som validoval aj nástrojom valgrind, pričom neboli zistené žiadne chyby ani neuvoľnená pamäť.

Výsledný program pracuje výhradne v príkazovom riadku a pre korektný beh vyžaduje tri parametre. Prvým je cesta k textovému súboru, ktorý reprezentuje databázu primitív, Druhý je textový súbor s definovanou úlohou a tretím parametrom je výstupný súbor, kde sa uložia všetky vygenerované protokoly za seba podľa špecifikácie jazyka CAS+.

6 Návrh protokolu implementovanou knižnicou

Táto kapitola demonštruje použitie implementovanej knižnice pri návrhu platobného protokolu. Rozhodol som sa implementovanú knižnicu demonštrovať na hľadaní platobného protokolu, ktorý je obdobný platobnému protokolu v práci Choia [6] a ktorého teoretický návrh metódou kompozície som popísal v kapitole 4.4. Môj zámer je porovnať teoretický výsledok a výsledok mojej implementácie.

6.1 Definícia úlohy

Podobne, ako v kapitole 4.4, aj tu budem uvažovať, že najčastejšie sú účastníci platobného protokolu traja. Bude to obchodník, zákazník a banka. Predpokladáme, že medzi bankou obchodníka a zákazníka už existuje zabezpečená sieť a budem teda tieto dva subjekty modelovať spoločne ako jeden. Úloha bude znázorňovať nákup tovaru prostredníctvom internetového obchodu.

Nákupná procedúra bude mať nasledujúci tvar:

1. Zákazník si vyberie tovar, ktorý chce kúpiť, potvrdí objednávku a rozhodne sa ju zaplatiť napríklad platobnou kartou. V tom momente sa obchodníkovi vygeneruje správa, ktorá bude obsahovať údaje o kupovanom tovare. Súčasne sa pošle správe aj banke a tá bude obsahovať okrem ceny tovaru a ďalších údajov o objednávke (dátum, počet kusov atď.) aj osobné údaje zákazníka vzťahujúce sa k platobnej karte. Je to napríklad PIN kód, dátum platnosti karty, číslo karty a podobne.
2. Obchodník na základe prijatej objednávky vygeneruje údaje, ktoré sa k nej vzťahujú a pošle ich banke. Táto správa bude obsahovať napríklad cenu tovaru, dátum, počet objednaných kusov tovaru a podobne. Mali by to byť údaje, ktoré si banka dokáže spojiť s údajmi od zákazníka.
3. Keď banka dostane údaje od obchodníka aj od zákazníka (nezáleží na tom, v akom poradí) a údaje nejakým spôsobom overí a priradí si ich k sebe, pošle obchodníkovi aj zákazníkovi odpoveď, ktorá im potvrdí správnosť údajov. Je dobré poznamenať, že banka vystupuje v tomto prípade ako dôveryhodný subjekt. Ostatní dvaja účastníci komunikácie teda predpokladajú, že keď dostanú kladnú odpoveď od banky, znamená to že boli obaja overený bankou.

Keď mám presnú predstavu o nákupnej procedúre, môžem sa pustiť do definície úlohy vo formáte, ktorý je vstupom implementovaného programu. Prvá sekcia popisuje všetky subjekty zúčastnené v komunikácii. Označme zákazníka ako `C`, obchodníka ako `M` a banku ako `B`. Zápis vo vstupnom bude nasledovný:

```
[agents]
C, M, B
```

V ďalšej sekcii budú definované počiatočné znalosti jednotlivých subjektov. Predpokladám, že každý subjekt pozná sám seba a všetky zúčastnené subjekty. Rovnako predpokladám, že všetci účastníci poznajú svoje páry kľúčov na šifrovanie a dešifrovanie a podpisovanie a verifikáciu. Preto tieto fakty do definície neuvádzam. Dôležité je ale uviesť to, že zákazník vie čo si chce objednať a tak isto vie svoje osobné údaje vzťahujúce sa k platobnej karte. Obchodník zase vie údaje, ktoré je

potrebné poslať banke spolu s objednávkou, aby si ho banka mohla overiť. Označme údaje o objednávke ako `OI`, údaje potrebné pre zaplatenie kartou (PIN, platnosť karty atď.) ako `PI` a údaje obchodníka ako `MI`. Zápis tejto sekcie úlohy bude mať tvar:

```
[knowledge]
C: PI, OI
M: MI
```

Nasledujúca sekcia bude popisovať ciele, ktoré chceme daným protokolom splniť. Chcem aby obchodník dostal údaje o objednávke `OI` a aby na základe nich potvrdil zákazníkovi prijatie a zaplatenie objednávky. Tento cieľ bude mať nasledujúci zápis:

```
[goals]
g1: knowledge: M = OI
g2: auth: C = M : OI
```

Ďalej požadujem, aby sa do banky dostali informácie o platbe zákazníka `PI` a údaje potrebné na overenie obchodníka `MI`. Na základe týchto údajov dostane obchodník aj zákazník potvrdenie od banky. Tieto ciele zapíšem nasledovne:

```
g3: knowledge: B = PI
g4: auth: C = B : PI
g5: knowledge: B = MI
g6: auth: M = B : MI
```

Posledná sekcia slúži na definovanie poradia, v akom sa jednotlivé ciele majú splniť. Lepšie povedané, nedefinuje presné poradie všetkých cieľov, ale definuje určité podmienky, ktoré musia byť splnené. Určite chceme aby, zákazník dostal potvrdenie od obchodníka až potom, ako obchodík dostane od zákazníka objednávku. Rovnako potvrdenie od banky by mal zákazník dostať až potom, ako banka dostane od neho údaje o platbe. No a samozrejme aj overenie obchodníka bankou nastane až potom, ako banka obdrží jeho údaje. Tieto základné ciele vyjadrím podmienkami:

```
[goal_order]
g1<g2
g3<g4
g5<g6
```

Na presnejšiu špecifikáciu protokolu však potrebujem pridať niekoľko ďalších podmienok. Chcem, aby banka potvrdila údaje aj obchodníkovi aj zákazníkovi až potom, ako obdrží od obidvoch týchto účastníkov potrebné údaje. Ďalej chcem, aby obchodník potvrdil zákazníkovi objednávku až potom, ako dostane on sám potvrdenie od banky. No a nakoniec požadujem, aby obchodník poslal svoje údaje do banky až potom, ako dostane objednávku od zákazníka. Po pridaní všetkých týchto cieľov bude zadanie úlohy kompletné a uložím ho do súboru `task.txt`, ktorý bude mať nasledujúci tvar:

```
[agents]
C, M, B
```

```
[knowledge]
C: PI, OI
M: MI

[goals]
g1: knowledge: M = OI
g2: auth: C = M : OI
g3: knowledge: B = PI
g4: auth: C = B : PI
g5: knowledge: B = MI
g6: auth: M = B : MI

[goal_order]
g1<g2
g3<g4
g5<g6
g6<g2
g3<g6
g5<g6
g1<g5
g5<g4
```

6.2 Použité primitívy

Autor metódy kompozície Choi [6] vo svojej práci definuje viacero primitív. V tejto podkapitole uvediem tie, ktoré som použil pri návrhu platobného protokolu popísaného v kapitole 6.1.

Rozhodol som sa vložiť do databázy štyri primitívy. Pri použití programu ich potom budem z databázy postupne odoberať, aby som ukázal vplyv počtu primitív na počet výsledných protokolov a tiež ich vplyv na dobu výpočtu. Tu sú teda použité primitívy:

Typ 1: $A \rightarrow B$: $B, \{|x, \beta_x|\}_{ek_B}, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle$

Typ 2: $A \rightarrow B$: $B, \{|x, \beta_x|\}_{ek_B}, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{sk_B}$

Typ 3: $A \rightarrow B$: $x, \beta_x, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{sk_B}$

Typ 4: $A \rightarrow B$: $\beta_x, \{|x, \beta_x|\}_{ek_B}, \langle x, \beta_x \rangle_{sk_A}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{sk_B}$

Program na vstupe vyžaduje tieto primitívy v špeciálnom formáte. Prvá primitíva bude mať tvar:

```
[goals]
g1: knowledge: resp=par
```

```
g2: auth: init=resp : par

[messages]
init->resp: resp, {par, b_grp}EK_resp, <par, b_grp>SK_init; g1
resp->init: init, <par, b_grp>; g2
```

Vo všetkých ďalších bude sekcia `goals` rovnaká, pretože všetky plnia rovnaké ciele. K ďalším primitívam preto kvôli prehľadnosti uvediem len ich sekcie `messages`:

```
[messages]
init->resp: resp, {par, b_grp}EK_resp, <par, b_grp>SK_init; g1
resp->init: init, <par, b_grp>SK_resp; g2

[messages]
init->resp: par, b_grp, <par, b_grp>SK_init; g1
resp->init: init, <par, b_grp>SK_resp; g2

[messages]
init->resp: b_grp, {par, b_grp}EK_resp, <par, b_grp>SK_init; g1
resp->init: init, <par, b_grp>SK_resp; g2
```

Každá z nich bude uložená v samostatnom súbore. Názvy týchto súborov potom uložím do jedného textového súboru, ktorý bude spolu s definíciou úlohy ďalším vstupným parametrom programu. Obsah databázového súboru pomenovaného ako `primitives.txt` bude teda nasledovný:

```
primType1.txt
primType2.txt
primType3.txt
primType4.txt
```

6.3 Výsledné protokoly a štatistiky programu

V tejto kapitole ukážem výsledky generovania protokolu špecifikovaného v kapitole 6.1. Protokoly budem generovať postupne s jednou, dvomi, tromi a nakoniec štyrmi primitívami v databáze. Celý proces vykonám na školskom serveri Merlin, kde bol v čase testovania procesor Quad-Core AMD Opteron™ Processor 2387 a 16 GB operačnej pamäte. Program bol preložený s nastaveným optimalizovaním `-O2`.

Na výsledkoch programu budem podrobnejšie ilustrovať, ako vytvorený program pracuje a prečo vznikli také kompozície ako vznikli. Spustenie programu vykonám príkazom:

```
~$ composition primitives.txt task.txt output.txt
```

Pre prvý beh programu ponechám v databáze len prvú zo štyroch primitív. Výsledkom je deväť vygenerovaných protokolov a priemerný čas jedného behu z desiatich spustení je $13\ ms$. Počet protokolov nie je veľký, pretože som zadal ciele úlohy pomerne jednoznačne a máme k dispozícii jedinú primitívu, takže kombinácií nevznikne mnoho. Nasleduje ukážka jedného z výsledkov:

```
protocol 7; % public key
identifiers
C, M, B : user;
OI, PI, MI : number;
EK_C, SK_C, EK_M, SK_M, EK_B, SK_B : public_key;
Hash : function;

messages
1. C -> B: B, {PI,B,C,M}EK_B, {PI,B,C,M}SK_C',
           {Hash(PI),Hash(OI),B,C,M}SK_C', {OI}SK_C'
2. C -> M: M, {OI,B,C,M}EK_M, {OI,B,C,M}SK_C',
           {Hash(OI),Hash(PI),B,C,M}SK_C', {PI}SK_C'
3. M -> B: B, {MI,B,C,M}EK_B, {MI,B,C,M}SK_M',
           {Hash(OI),Hash(PI),Hash(MI),B,C,M}SK_M'
4. B -> C: C,Hash(PI,B,C,M)
5. B -> M: M,Hash(MI,B,C,M)
6. M -> C: C,Hash(OI,B,C,M)

knowledge
C: M, B, PI, OI, EK_C, SK_C, EK_M, EK_B, Hash;
M: C, B, MI, EK_C, EK_M, SK_M, EK_B, Hash;
B: C, M, EK_C, EK_M, EK_B, SK_B, Hash;

session_instances
[C:C_inst, M:M_inst, B:B_inst];

intruder_knowledge

goal
secrecy_of OI [M,C];
C authenticates M on OI
secrecy_of PI [B,C];
C authenticates B on PI
secrecy_of MI [B,M];
M authenticates B on MI
```

Zo zadania vieme, že nezáleží na poradí v akom zákazník pošle údaje o objednávke a o platbe patričným subjektom. Rovnako tak nezáleží na poradí štvrtej a piatej správy. Práve tieto a niektoré ďalšie možnosti vo výsledku tvoria deväť protokolov. V každej správe, kde boli pridané väzobné prvky, ktoré nie sú súčasťou originálnej primitívy, ale museli byť do protokolu pridané za účelom previazania primitív do jedného celku, som tieto prvky podčiarkol. V tomto prípade sa celý protokol zostavoval z troch čiastkových protokolov (každý subjekt komunikuje s každým). Z definície väzobného prvku vyplýva, že sa bude jednať o zjednotenie parametrov spojovaných primitív a ich väzobných skupín podpísané privátnym kľúčom odosielateľa. Kvôli bezpečnosti a utajeniu sa však nepoužíva parameter priamo, ale iba jeho haš. Zjednotenie týchto údajov pre prvú a druhú primitívu je $\{Hash(PI), Hash(OI), B, C, M\}SK\_C'$. Program však pri kontrole zistil, že banka nemá znalosti o objednávke OI a obchodník naopak nemá znalosti o platbe PI, takže by nemali tento väzobný prvok voči čomu overiť. Preto musia byť do týchto správ pridané ďalšie potrebné objekty, aby väzobný prvok bolo možné korektne overiť. Pri pripojovaní tretej primitívy je situácia už trošku iná. Do tejto primitívy sa ako väzobný prvok pridá zjednotenie väzobného prvku predošlých primitív a parameter novej primitívy. Celá množina bude podpísaná podpisovým kľúčom obchodníka. Jej tvar bude $\{Hash(OI), Hash(PI), Hash(MI), B, C, M\}SK\_M'$. V tomto prípade prijímateľ, banka, má vedomosť

o všetkých objektoch väzobného prvku, takže si ich dokáže porovnať a zasielanie ďalších objektov v správe nie je nutné.

Pri vytváraní sekcie počiatočných znalostí, sa program riadi jednoduchými pravidlami: každý subjekt ma vedomosť o každom subjekte, každý pozná hašovaciu funkciu, každý pozná svoj šifrovací a dešifrovací kľúč, podpisový a verifikačný kľúč a každý pozná šifrovacie (verejné) kľúče ostatných subjektov. A samozrejme jednotlivé subjekty majú vo svojich znalostiach tie svoje dáta, ktoré sú predmetom nimi posielaných správ (údaje o objednávke, o platbe a o obchodníkovi).

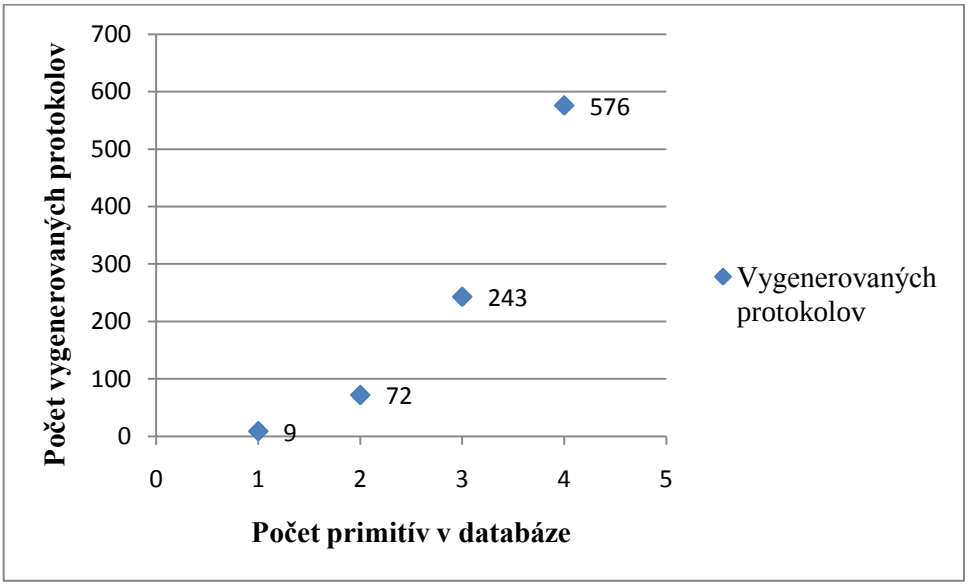
Pri generovaní cieľov program postupuje tak, že pre každý cieľ v zadaní úlohy, ktorý definuje nadobudnutie znalosti o dátach nejakým subjektom, vygeneruje do výstupného protokolu cieľ, ktorý požaduje utajenie týchto dát pred všetkými subjektmi, ktoré nie sú v cieli ďalej uvedené. V tomto prípade chcem, aby údaje o objednávke boli známe len zákazníkovi a obchodníkovi, aby údaje o platbe boli známe len zákazníkovi a banke a aby údaje o obchodníkovi boli známe len jemu a banke. Ďalšie ciele vo výsledných protokoloch sú obdobné cieľom v zadaní. Chcem aby sa zákazník autorizoval u obchodníka na základe údajov o objednávke, aby sa zákazník autorizoval v banke na základe platobných údajov a aby sa obchodník autorizoval v banke na základe jeho údajov.

Za povšimnutie stojí ešte sekcia o vedomostiach potenciálneho útočníka. Túto sekciu ponechávam úmyselne prázdnu a jej vyplnenie je na návrhárovi, ktorý bude chcieť otestovať konkrétnu situáciu.

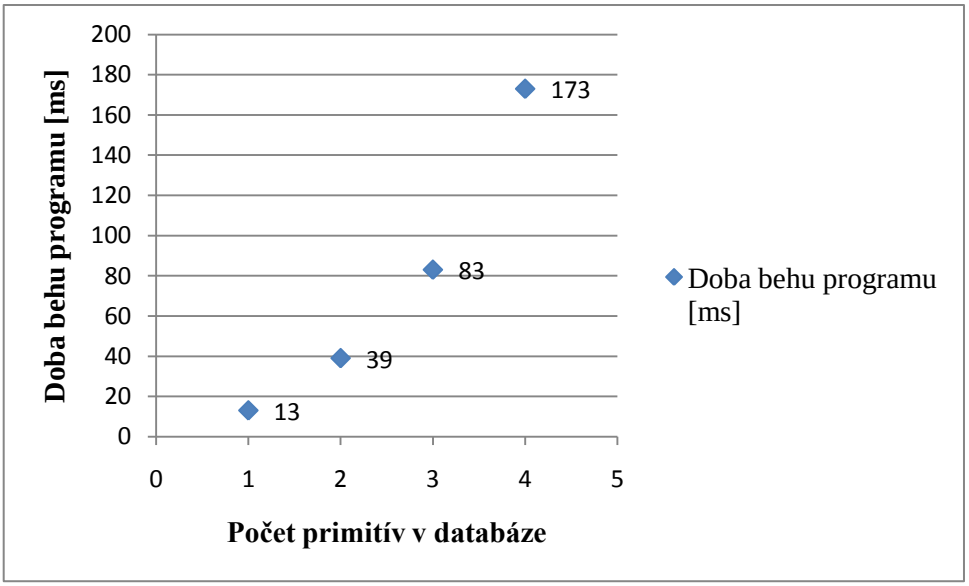
Následne som test opakoval postupne s dvomi, tromi a štyrmi primitívami v databáze. Počet vygenerovaných protokolov a dobu behu programu v jednotlivých prípadoch ilustruje tabuľka 1. Závislosť počtu vygenerovaných protokolov na počte primitív znázorňuje graf na obrázku 8 a závislosť doby trvania behu programu na počte primitív znázorňuje graf na obrázku 9.

Počet primitív v databáze	1	2	3	4
Vygenerovaných protokolov	9	72	243	576
Doba behu programu [ms]	13	39	83	173

Tabuľka 1: Štatistika generovania protokolov



Obrázok 8: Závislosť počtu vygenerovaných protokolov na počte primitív



Obrázok 9: Závislosť doby behu programu na počte primitív v databáze

Z grafov je zrejmé, že aj počet protokolov aj doba trvania má tendenciu rásť exponenciálne s počtom primitív v databáze. V prípade doby trvania sa však stále jedná o pomerne malé hodnoty. Extrémne veľký počet výsledných protokolov ale asi nebude pre návrhára prínosom, preto je potrebné si uvážiť, ktoré primitívy sa do koncepcie hľadaného protokolu hodia najlepšie a použiť pri generovaní len tieto. Prípadne je tu možnosť ešte detailnejšie špecifikovať zadanie úlohy, aby pri kompozícii nevzniklo také veľké množstvo rôznych kombinácií.

6.4 Záverečné úpravy

Protokoly získané metódou kompozície by nemali byť považované za finálne. Mali by slúžiť ako základ pri navrhovaní výsledného protokolu. V tejto kapitole naznačím, ako by takáto finalizácia mohla vyzerať.

Jeden z vygenerovaných protokolov v predchádzajúcej kapitole obsahuje nasledujúcu sekvenciu správ:

```
1. C -> B: B, {PI,B,C,M}EK_B, {PI,B,C,M}SK_C',
           {Hash(PI),Hash(OI),B,C,M}SK_C', {OI}SK_C'
2. C -> M: M, {OI,B,C,M}EK_M, {OI,B,C,M}SK_C',
           {Hash(OI),Hash(PI),B,C,M}SK_C', {PI}SK_C'
3. M -> B: B, {MI,B,C,M}EK_B, {MI,B,C,M}SK_M',
           {Hash(OI),Hash(PI),Hash(MI),B,C,M}SK_M'
4. B -> C: C,Hash(PI,B,C,M)
5. B -> M: M,Hash(MI,B,C,M)
6. M -> C: C,Hash(OI,B,C,M)
```

Pri bližšom pohľade na tieto správy môžeme usúdiť, že niektoré z nich by bolo možné spojiť dohromady a znížiť tak ich počet v protokole. Jednou z možností je pokúsiť sa odstrániť komunikáciu medzi zákazníkom a bankou. Zákazník by mohol údaje o platbe poslať priamo obchodníkovi. Keďže sú zašifrované šifrovacím kľúčom banky, obchodík ich nedokáže dešifrovať a bude ich musieť poslať banke. Tá potom obdrží aj údaje o platbe aj údaje o obchodníkovi priamo od obchodníka. V odpovedi potom pripojí haš platobných údajov (spojenie správ 4 a 5), ktorému opäť obchodník nerozumie a prepošle ho zákazníkovi. Upravený protokol by mohol mať nasledujúci tvar:

```
1. C -> M: M, {OI,B,C,M}EK_M, {PI,B,C,M}EK_B, {PI,B,C,M}SK_C',
           {OI,B,C,M}SK_C', {Hash(PI),Hash(OI),B,C,M}SK_C'
2. M -> B: B, {MI,B,C,M}EK_B, {PI,B,C,M}EK_B, {MI,B,C,M}SK_M',
           {PI,B,C,M}SK_C',
           {Hash(OI),Hash(PI),Hash(MI),B,C,M}SK_M'
3. B -> M: M, Hash(PI,B,C,M), Hash(MI,B,C,M)
4. M -> C: C, Hash(OI,B,C,M), Hash(PI,B,C,M)
```

V správach by sa po takýchto úpravách niekedy mohli objaviť určité redundantné prvky, ktoré je v závere vhodné odstrániť.

6.5 Porovnanie teoretického a automatizovaného návrhu

Automatizovaný návrh dáva návrhárovi určitý nástroj na získanie množiny protokolov, na ktorých môže založiť svoj teoretický návrh. Znamená to, že automatický návrh metódou kompozície, ktorý som v rámci tejto práce implementoval, sa bez finálneho zásahu a doladenia návrhára zrejme nezaobíde. Vo vygenerovaných protokoloch môžu vznikať určité situácie, ktoré takéto doladenie vyžadujú. Napríklad situácia, keď v protokole nastanú dve po sebe idúce správy s rovnakými odosielateľmi a adresátmi. Tieto správy by bolo možné spojiť do jednej a znížiť tak ich celkový počet. Iným prípadom je vhodne presmerovať niektoré správy tak, aby sa z protokolu vylúčila komunikácia medzi niektorými subjektmi, čo vedie k prehľadnejšiemu a kompaktnejšiemu protokolu. Tieto a niektoré ďalšie faktory metódy kompozície podrobnejšie rozvediem v kapitole 7, kde sa budem zaoberať možnosťami ďalšieho vývoja programu.

7 Možnosti ďalšieho vývoja

Vo svojej práci som implementoval základný princíp metódy kompozície a demonštroval som použitie implementovaného programu pri návrhu platobného protokolu. Program však má určitý priestor na vylepšenia. Je dôležité si uvedomiť niekoľko faktov. Na zostavovanie protokolov som používal primitívy, ktoré vždy mali práve jeden parameter. Pri kompozícii niekedy vznikajú protokoly, v ktorých dve po sebe idúce správy majú rovnakého odosielateľa a rovnakého prijímateľa. V prípade protokolov s viacerými subjektmi sa objavovali správy, ktoré by bolo možné poslať adresátovi nepriamo, prostredníctvom iného subjektu. Všetky tieto aspekty vytvárajú určitý priestor na vylepšenie.

7.1 Spájanie správ medzi rovnakými subjektmi

Predpokladajme, že by v protokole vznikla situácia, kde by za sebou nasledovali dve primítivy nasledovne:

$A \rightarrow B$: $B, \{|x, \beta_x|\}_{k_{AB}}, \langle x, \beta_x \rangle_{k_{AB}}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{k_{BA}}$
$B \rightarrow A$: $A, \{|y, \beta_y|\}_{k_{BA}}, \langle y, \beta_y \rangle_{k_{BA}}$
$A \rightarrow B$: $B, \langle y, \beta_y \rangle_{k_{AB}}$

Prostredné dve správy je možné spojiť do jednej a znížiť tak celkový počet správ:

$A \rightarrow B$: $B, \{|x, \beta_x|\}_{k_{AB}}, \langle x, \beta_x \rangle_{k_{AB}}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{k_{BA}}, \{|y, \beta_y|\}_{k_{BA}}, \langle y, \beta_y \rangle_{k_{BA}}$
$A \rightarrow B$: $B, \langle y, \beta_y \rangle_{k_{AB}}$

Pričom nedôjde k porušeniu poradia, v ktorom v daných primitívach budú naplnené ich ciele. Správa uprostred zároveň obsahuje dva parametre x a y. Pri spájaní správ sa však komplikuje situácia pri spájaní primitív pomocou väzobného prvku.

7.2 Presmerovanie správ

Ďalším možným vylepšením programu by bolo riešenie situácií, kde je možné vhodne využiť presmerovanie určitých správ za účelom zníženia počtu medzi sebou komunikujúcich subjektov. Predpokladajme tento zjednodušený prípad:

$A \rightarrow B$: $B, \{|x, \beta_x|\}_{k_{AB}}, \langle x, \beta_x \rangle_{k_{AB}}$
$A \rightarrow C$: C, $\{|z, \beta_z|\}_{k_{AC}}, \langle z, \beta_z \rangle_{k_{AC}}$
$B \rightarrow C$: C, $\{|y, \beta_y|\}_{k_{BC}}, \langle y, \beta_y \rangle_{k_{BC}}$
$C \rightarrow B$: $B, \langle y, \beta_y \rangle_{k_{CB}}$
$C \rightarrow A$: $A, \langle z, \beta_z \rangle_{k_{CA}}$
$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{k_{BA}}$

V tomto prípade by bolo možné úplne odstrániť komunikáciu medzi subjektmi A a C. Subjekt A totiž môže poslať parameter z priamo subjektu B, pretože je zašifrovaný kľúčom, ktorý tento subjekt nepozná a teda by nenarušil utajenie parametru z. Len by ho preposlal subjektu C spolu s jeho ďalšími údajmi. Ten by následne vygeneroval odpovede pre oba subjekty A aj B, ale poslal by ich obidve subjektu B. Ten by zašifrovanej odpovedi pre subjekt A opäť nerozumel a preposlal by mu ju spolu so svojou odpoveďou. Po úpravách by sekvencia správ mohla vyzerať nasledovne:

$A \rightarrow B$: $B, \{|x, \beta_x|\}_{k_{AB}}, \langle x, \beta_x \rangle_{k_{AB}}, \{|z, \beta_z|\}_{k_{AC}}, \langle z, \beta_z \rangle_{k_{AC}}$

$B \rightarrow C$: C, $\{|y, \beta_y|\}_{k_{BC}}, \langle y, \beta_y \rangle_{k_{BC}}, \{|z, \beta_z|\}_{k_{AC}}, \langle z, \beta_z \rangle_{k_{AC}}$

$C \rightarrow B$: $B, \langle y, \beta_y \rangle_{k_{CB}}, \langle z, \beta_z \rangle_{k_{CA}}$

$B \rightarrow A$: $A, \langle x, \beta_x \rangle_{k_{BA}}, \langle z, \beta_z \rangle_{k_{CA}}$

7.3 Ďalšie vylepšenia

Medzi ďalšie vylepšenia by som mohol uviesť napríklad pridanie užívateľského rozhrania, kde by návrhár videl aké primitívy má v databáze, prípadne ich mohol pridávať a upravovať. Užitočným by mohla byť aj podpora ďalších výstupných formátov prípadne export výsledných protokolov tak, aby každý bol v samostatnom súbore. Tieto vylepšenia sa týkajú ale hlavne komfortu pri práci s programom, nie sú ničím prínosným pre metódu kompozície ako takú.

8 Záver

V úvode práce, v kapitole 2, som sa venoval platobným systémom. Vysvetlil som pojem elektronická komercia, jej typy a použitie. Ďalej som zobrazil model obecného platobného systému, popísal jeho entity a funkciu týchto entít v platobnom styku. V závere kapitoly som načrtol požiadavky, ktoré sú kladené na bezpečnú komunikáciu po otvorenej sieti v rámci platobného styku.

Kapitola 3 je venovaná metódam používaným na návrh komunikačných protokolov. Použitie jednoduchej logiky sa opiera o BAN logiku. Jej základom sú dve množiny: množina znalostí a množina predpokladov a interferenčné pravidlá. Ich použitím sa modifikuje obsah oboch množín. Pomocou logiky je možné modifikovať existujúce protokoly a odstrániť prípadné ich nedostatky, či navrhnúť protokoly nové. Techniku som prezentoval na vylepšení protokolu „Woo and Lam“.

Derivačný systém na návrh protokolov je založený na základných komponentoch a operáciách nad týmito komponentmi. Sú to operácie kompozícia, vylepšenie a transformácia. Kompozícia je skladanie komponent za seba. Vylepšenie je nahradenie časti správy inou časťou správy, napríklad nahradenie keksíku jeho zašifrovanou podobou, pričom sa nemení počet správ ani štruktúra protokolu. Transformácia na druhej strane mení štruktúru protokolu. Správy sa môžu kombinovať, môžu sa presúvať dáta z jednej správy do inej, prípadne pridávať správy nové. Techniku som ilustroval deriváciou protokolov rodiny STS z dvoch základných komponentov: Diffie-Hellman a autorizácie typu Challenge-Response. Deriváciou je možné vytvoriť napríklad protokoly IKE či JFK, ktoré sa používajú na distribúciu kľúčov.

V závere kapitoly 3 a v celej kapitole 4 sa venujem kompozičnej metóde návrhu protokolov, ktorá môže dobre poslúžiť aj pri návrhu platobných protokolov. Hlavnou výhodou tejto metódy pri návrhu platobných protokolov je, že zložitý problém sa dekomponuje na jednoduché čiastkové problémy, ktoré je možne riešiť samostatne a jednoduchšie verifikovať dostupnými nástrojmi. Protokoly sa zostavujú zo základných stavebných blokov – primitív. Tieto sa potom spájajú do väčších celkov a spĺňajú tak komplexnejšie požiadavky na protokol. Každá primitíva musí byť navrhnutá tak, aby boli splnené presne dané pravidlá, ktoré umožňujú neobmedzene tieto primitívy komponovať a tvoriť tak zložitejšie celky. Tieto požiadavky sú napríklad nedeštruktívnosť a nekonštruktívnosť cieľov, ktoré daná primitíva implementuje. Každá z týchto primitív implementuje určité ciele, ktorých poradie nesmie byť pri kompozícii narušené. V poslednej časti kapitoly 4 sa zaoberám konkrétnym návrhom nákupnej procedúry platobného protokolu SET. Protokol sa rozdelí na tri čiastkové protokoly medzi jednotlivými účastníkmi komunikácie (komunikuje každý s každým) a tie sa riešia ako samostatné problémy tak, aby boli splnené uvedené požiadavky jednotlivých účastníkov komunikácie. Sú to vždy zákazník, obchodník a banka (nadobúdateľ a vydavateľ platobnej karty) Primitívy sa vyberajú podľa toho, ktoré informácie je potrebné utajiť. Pri výslednom protokole si môžeme byť istý, že je korektný, ak sú korektné všetky primitívy použité na jeho kompozíciu.

Kapitola 5 popisuje spôsob akým som implementoval danú metódu v jazyku C++. Výsledkom je program, ktorý pracuje v príkazovom riadku a vyžaduje na vstupe tri parametre – databázu primitív, zadanie úlohy, ktorú chceme kompozíciou vyriešiť a nakoniec výstupný súbor, kde budú uložené všetky výsledné protokoly. Pre zápis primitív a definíciu úlohy som si zvolil textový formát, ktorý je veľmi podobný jazyku CAS+, ale niektoré konštrukcie som nahradil za jednoduchšie. Na výstupe sú potom protokoly vygenerované v samotnom jazyku CAS+, čo umožňuje ich jednoduché použitie pri ďalšom spracovaní v iných dostupných programoch na prácu s protokolmi, ako napríklad

program SPAN[4]. Základná logika programu je taká, že sa vygenerujú všetky permutácie možných protokolov, na základe primitív, ktorými je možné zvolené ciele naplniť, a potom sa z týchto permutácií vyberú len tie, ktoré vyhovujú všetkým podmienkam špecifikovaným v zadaní.

Následne som demonštroval použitie programu na návrhu platobného protokolu. Rozhodol som sa použiť podobný model, na akom metódu teoreticky demonštroval autor metódy Choi [6], aby bolo možné výsledky jednoduchšie porovnať. Súčasťou tohto protokolu sú tri subjekty – zákazník, obchodník a banka, pričom každý z nich komunikuje navzájom s každým. Pri použití štyroch primitív som získal metódou kompozície až viac ako 500 rôznych možných protokolov. Ukázal som, že počet vygenerovaných protokolov rastie exponenciálne s počtom primitív v databáze, takže vyšší počet primitív by už nebol pre návrhára protokolu prínosom, ale naopak komplikáciou, pretože by musel vyberať z veľmi veľkej množiny. Ako lepší sa javí prístup, kde by sa dopredu vyberie len jedna ale dve primitívy, ktoré by najlepšie vyhovovali zadanej úlohe a následne sa len tieto použili na generovanie protokolov.

V poslednej časti práce potom naznačujem možnosti na vylepšenie implementovaného algoritmu, ktoré by viedli v niektorých prípadoch k zmenšeniu počtu správ v jednotlivých protokoloch, prípadne k transformácii troch protokolov medzi tromi účastníkmi na dva protokoly.

Na záver by som charakterizoval výsledný program ako prostriedok, ktorý sa nachádza na rozhraní teoretického a automatizovaného návrhu protokolu. Dokáže na základe špecifikovaných požiadaviek a cieľov vygenerovať návrhárovi množinu protokolov, ktoré sú ďalej vďaka použitému CAS+ jazyku vhodné na ďalšie automatizované spracovanie pomocou iných dostupných nástrojov, ako napríklad programu SPAN, kde je možné tieto protokoly ďalej filtrovať podľa rôznych kritérií prípadne ich verifikovať a overiť tak ich správnosť.

[4] SPAN – Security Protocol Animator http://www.irisa.fr/celtique/genet/span/

Literatúra

[1] SHERIF, M. *Protocols for Secure Electronic Commerce*. 2nd edition. CRC Press, 2004.

[2] BUTTYÁN, L; STAAMANN, S; WILHELM, U. A Simple Logic for Authentication Protocol Design. *In 11th IEEE Computer Security Foundations Workshop*. 1998

[3] ABADI, M; NEEDHAM, R. Prudent engineering practice for cryptographic protocols. *In Proceedings of the IEEE CS Symposium on Research in Security and Privacy*. 1994, s. 122-136.

[4] DEBBABI, M; MEJRI, M; TAWBI, N; YAHMADI, I. A new algorithm for the automatic verification of authentication protocols: From specifications to flaws and attack scenarios. *In DIMACS Workshop on Designand Formal Verification of Security Protocols*. 1997.

[5] DATTA, A; DEREK, A; MITCHELL, J; PAVLOVIC, D. A Derivation System for Security Protocols and its Logical Formalization. *In Proceedings of the 16th IEEE Computer Security Foundations Workshop*. 2003

[6] CHOI, H. *Security protocol design by composition.* 2006. Technical Report. University of Cambridge.

[7] SAILLARD, R; GENET, T. *CAS+*. 2011

[8] BURROWS, M; ABADI, M; NEEDHAM, R. A logic of authentication. *ACM Transactions on Computer Systems*. 1990.

[9] WOO, T; LAM S. Authentication for distributed systems. *Computer*. 1992.

[10] DIFFIE, W; HELLMAN, M. New directions in cryptography. *IEEE Transactions on Information Theory*. 1976.

Zoznam príloh

Príloha 1. CD - Zdrojové súbory, projektový súbor pre Microsoft Visual Studio 2008, unix makefile, ukážková databáza stavebných prvkov a ukážkové príklady použitia