VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

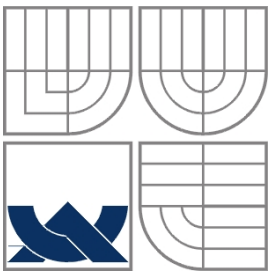
EVOLUČNÁ KNIŽNICA PRE PODPORU NÁVRHU KOMUNIKAČNÝCH PROTOKOLOV

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

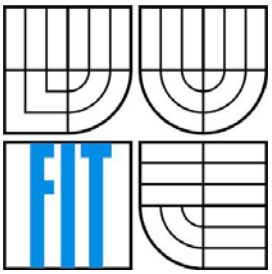
AUTOR PRÁCE
AUTHOR

MARTIN SAMEŠ

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EVOLUČNÍ KNIHOVNA PRO PODPORU NÁVRHU KOMUNIKAČNÍCH PROTOKOLŮ

EVOLUTIONARY LIBRARY FOR THE COMMUNICATION PROTOCOLS DESIGN

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN SAMEŠ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PAVEL OČENÁŠEK

BRNO 2007

Zadání diplomové práce/3354/2006/xsames02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Sameš Martin**
Obor: Výpočetní technika a informatika
Téma: **Evoluční knihovna pro podporu návrhu komunikačních protokolů**
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s technikami automatizovaného návrhu komunikačních protokolů. Zaměřte se především na bezpečnostní protokoly.
2. Nastudujte principy evolučních algoritmy. Zaměřte se zejména na genetické programování genetické algoritmy a simulované žíhání.
3. Diskutujte rozdíly v použití jednotlivých přístupů při automatizovaném návrhu protokolů. Zaměřte se zejména na bezpečnostní protokoly.
4. Zvolený přístup (po dohodě s vedoucím práce) implementujte do knihovny funkcí pro podporu návrhu komunikačních protokolů. Implementaci proveďte tak, aby bylo možné využít již existujících knihoven pro manipulaci s protokoly a heuristickými technikami.
5. Výsledky demonstrujte na příkladech návrhů jednoduchých bezpečnostních protokolů. Diskutujte možnosti dalšího rozšíření.

Literatura:

- Očenášek Pavel, Švéda Miroslav: An Approach to Automated Design of Security Protocols, In: Proceedings of the International Conference on Networking (ICN 2006), Los Alamitos, US, IEEE CS, 2006, p. 4, ISBN 0-7695-2522-0
- Očenášek Pavel: Verifikace bezpečnostních protokolů : diplomová práce, Brno, CZ, FIT VUT, 2003, p. 54
- Další literatura dle pokynů vedoucího práce.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Očenášek Pavel, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Martin Sameš**
Id studenta: 22703
Bytem: Bernolákova 1, 974 05 Banská Bystrica
Narozen: 20. 09. 1982, Zvolen
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Evoluční knihovna pro podporu návrhu komunikačních protokolů
Vedoucí/školitel VŠKP: Očenášek Pavel, Ing.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Potreba vyvíjať nové a nové bezpečnostné protokoly, ktoré zodpovedajú stanoveným požiadavkám vedie k automatizácií ich návrhu a následne lepšiu možnosť verifikácie. Táto práca sa zaoberá možnosťou automatizovaného návrhu protokolov s využitím genetických algoritmov. Porovnaním jednotlivých prístupov a následným vytvorením evolučnej knižnice pre podporu návrhu bezpečnostných protokolov.

Klíčová slova

Bezpečnostné protokoly, dizajn, automatizované metódy, evolučné algoritmy

Abstract

Developement and verification of new security protocols, which meets the requirements, needs automated techniques. This work deals with the possibility of using evolutionary approach in design of security protocols. By showing and comparing different methods and using some of them to create evolutionary library for support in developement of new communication protocols.

Keywords

Security protocols, design, automated methods, genetic algorithms

Citace

Sameš Martin: Evolučná knižnica pre podporu návrhu komunikačných protokolov. Brno, 2007, diplomová práce, FIT VUT v Brně.

Evolučná knižnica pre podporu návrhu komunikačných protokolov

Prohlášení

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing.Pavla Očenáška. Uviedol som všetky literárne pramene a publikácie, z ktorých sem čerpal.

.....
Martin Sameš
20.05.2007

Poděkování

Chcel by som sa poďakovať Ing.Pavlovi Očenáškovi za pomoc a cenné odborné rady pri riešení DP, Karlovi Tomáškovi za spoluprácu.

© Martin Sameš, 2007.

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

Obsah

Obsah 1
1 Úvod 3
2 Bezpečnostné protokoly a techniky automatizovaného návrhu 4
2.1 Bezpečnostné protokoly 4
2.1.1 Základné pojmy 4
2.1.2 Rozdelenie bezpečnostných protokolov 5
2.1.3 Príklady jednoduchých autentizačných protokolov 5
2.2 Návrh s využitím jednoduchej logiky 7
2.2.1 Základné pravidlá logiky 7
2.2.2 Príklad návrhu protokolu použitím syntetizovaných pravidiel: 8
2.3 Návrh pomocou odvodenia, derivačný systém 9
2.3.1 Príklad vytvorenia derivačného stromu, odvodenia protokolov 10
2.4 Návrh pomocou evolučných algoritmov 12
3 Vybrané evolučné a príbuzné algoritmy 13
3.1 Genetické algoritmy ako základ EA 13
3.1.1 Všeobecný postup genetického návrhu 13
3.2 Genetické programovanie 15
3.3 Gramatická evolúcia 17
3.3.1 BNF a GE 17
3.4 Simulované žíhanie 20
3.5 Evolučná knižnica GAlib 21
3.5.1 Základné prvky programu 21
3.5.2 Dôležité vlastnosti a triedy 24
4 Knižnica funkcií pre podporu návrhu komunikačných protokolov 25
4.1 Analýza a návrh 25
4.1.1 Výber existujúcich knižníc a ich využitie spolu s knižnicou funkcií 25
4.1.2 Od špecifikácie požiadaviek až po hľadaný protokol 26
4.2 Implementácia 29
4.2.1 Voľba programovacieho jazyka a platformy 29
4.2.2 Použité knižnice 29
4.2.3 Inicializácia 29
4.2.4 Proces GA 30
4.2.5 Podporné funkcie 32
4.3 Testy a získané výsledky 33

4.3.1 Fitness protokolu v závislosti na počte generácií....33
4.3.2 Závislosti na veľkosti populácie, počtu inštrukcií, generácií, dobe trvania a výslednej fitness protokolu....37
5 Záver....39
Literatúra....40
Zoznam príloh....42

1 Úvod

História sietí siaha do obdobia studenej vojny, kedy koncom 60. rokov vznikla prvá sieť, ktorá spájala univerzity a výskumné centrá. Sieť sa používala predovšetkým na armádne a vedecké účely. Keďže počet účastníkov bol malý, spoliehalo sa na ich lojalitu a diskrétnosť. Zabezpečeniu sa nevenovala veľká pozornosť. Autentizácia komunikujúcich strán prebiehala na základe hesla, ktoré sa šírilo sieťou nechránené. S postupnou komercializáciou a prenikaním sietí do všetkých odvetví v 80 a hlavne 90. rokoch sa zvyšovali požiadavky na zabezpečenie komunikácie a protokoly používané od počiatku sietí sa stali nedostatočnými. Potreba vyvíjať stále nové komunikačné (bezpečnostné) protokoly viedla k snahe automatizovať ich návrh a verifikáciu na základe definovaných požiadaviek. V dnešnej dobe existuje mnoho techník, ktoré sa uplatňujú pri vývoji a následnej verifikácií protokolov. Medzi najzaujímavejšie a stále čoraz viac aplikované na reálne problémy patria evolučné algoritmy (EA).

Možnosti ich využitia pri automatizovanom návrhu komunikačných protokolov [1][19] sa stali motiváciou pre moju prácu. Cieľom ktorej je, na základe evolučných metód, vytvorenie knižnice funkcií spolupracujúcej s už existujúcimi knižnicami pre manipuláciu s protokolmi a heuristickými technikami tak, aby umožnila generovanie protokolu, spĺňajúceho definované požiadavky.

V prvej časti diplomovej práce sa zaoberám bezpečnostnými protokolmi a rôznymi metódami používanými pri ich automatizovanom návrhu. Princípom genetických algoritmov, genetického programovania, gramatickej evolúcie, simulovaného žíhania a logiky BAN. Porovnaniu a výberom vhodných techník pre implementáciu podpornej knižnice funkcií.

V druhej časti sa venujem analýze problému, možným riešeniam, implementácií jednotlivých funkcií a testovaniu navrhnutej knižnice na jednoduchej špecifikácií protokolu pre rôzne nastavenia počiatočných hodnôt genetického algoritmu.

Záver obsahuje zhodnotenie dosiahnutých výsledkov a možnosti ďalšieho rozšírenia.

2 Bezpečnostné protokoly a techniky automatizovaného návrhu

2.1 Bezpečnostné protokoly

Protokol je množina pravidiel a dohôd, ktoré definujú komunikačný rámec medzi dvoma a viac subjektmi. Môžu to byť koncoví užívatelia, procesy, alebo počítačové systémy.

V kryptografických protokoloch je časť minimálne jednej správy zašifrovaná. Kryptografické a bezpečnostne orientované komunikačné protokoly sa používajú na vytvorenie bezpečnej komunikácie nad nezabezpečenými otvorenými sieťami a distribuovanými systémami. Tieto protokoly používajú kryptografické techniky na dosiahnutie cieľových vlastností ako sú dôveryhodnosť, autentizácia subjektov a služieb, integrita, poradie, aktuálnosť správ a distribúcia kryptografických kľúčov.

2.1.1 Základné pojmy

Kryptografia

Pomáha k zabezpečeniu komunikácie, aby útočník nemohol bez znalosti kľúčov získať originálne správy.

Kryptografia využíva dva symetrické procesy: šifrovanie a dešifrovanie. Šifrovanie sa podľa použitých šifrovacích algoritmov a kľúčov rozdeľuje na symetrické a asymetrické. Pri symetrickom šifrovaní sa používajú symetrické kľúče, čo znamená že obe komunikujúce strany používajú jeden zhodný kľúč. Asymetrické šifrovanie je založené na použití dvoch párových kľúčov. Každý subjekt má svoj verejný a privátny kľúč. V tomto prípade je možné nielen správy šifrovať, ale tiež podpisovať.

Asymetrická kryptografia má oproti klasickej, symetrickej veľkú výhodu v tom, že nie je nutné sa dohodnúť na tajnom zdieľanom kľúči pred zahájením utajenej komunikácie. Na druhej strane však hrozí podvrhnutie verejného kľúča (dajú sa voľne šíriť verejné kľúče s nepravou identitou). Tomu zamedzujú certifikačné autority, ktoré potvrdzujú pravosť verejných kľúčov komunikujúcich subjektov.

Nonce

Často označovaný ako "keksík", ide o náhodne vygenerované číslo (napr. časové razítko...).

Relačný kľúč

Ide o kľúč, ktorý sa používa k zabezpečeniu komunikácie, vytvára sa na začiatku a len pre danú reláciu.

2.1.2 Rozdelenie bezpečnostných protokolov

Symetrické protokoly:

Používajú symetrické šifrovacie algoritmy (rovnaký kľuč pre šifrovanie aj dešifrovanie) napr. šifrovací algoritmus DES. Správu zašifruje odosielateľ kľúčom, ktorý pozná aj prijímateľ, ten ju následne dešifruje.

Pri tomto type komunikácie sa väčšinou vyskytuje tretia strana, dôveryhodný server – KDC /KTC (*Key Distribution Center / Key Translation Center*), ktorý zabezpečuje výmenu kľúčov medzi komunikujúcimi subjektmi bezpečnou formou, obvykle po zvláštnom zabezpečenom komunikačnom kanáli. Medzi symetrické protokoly patrí napríklad Otway-Rees, Yahalom...

Asymetrické protokoly:

Používajú asymetrické šifrovacie algoritmy, napr. RSA., kde máme dvojicu kľúčov, verejný a privátny. Privátny kľúč reprezentuje identitu jeho vlastníka, verejný kľúč je voľne k dispozícií ostatným subjektom, ktorí sa podieľajú na komunikácií.

Medzi asymetrické protokoly patrí napr. IKE (Internet Key Exchange protocol) používaný v IPv6, SET (Secure Electronic Transaction protocol) používaný v bankovníctve spoločnosťami VISA a MasterCard, Needham-Schroeder autentizačný protokol..

Z hľadiska využitia bezpečnostných protokolov [11]**:**

- pre bezpečny prenos dát (IPSec, SSL/TSL, SSH, WTSL, SASL...)
- autentizačné protokoly pre autentizáciu, distribúciu kľúčov, šifrovanie prenášaných dát medzi komunikujúcimi subjektami (Wide Mouth Frog, Diffle Helman, Needham-Schroeder, Otway Rees, Yahalom, Kerberos ...), tvoria hlavnú časť bezpečnostných protokolov
- pre bankový sektor (SET...)
- pre zabezpečenie dát na úrovni aplikácií (HTTPS, S-HTTP, S/MIME, SFTP …).

2.1.3 Príklady jednoduchých autentizačných protokolov

Needham-Schroeder, symetrická verzia, symbolický zápis, KDC:

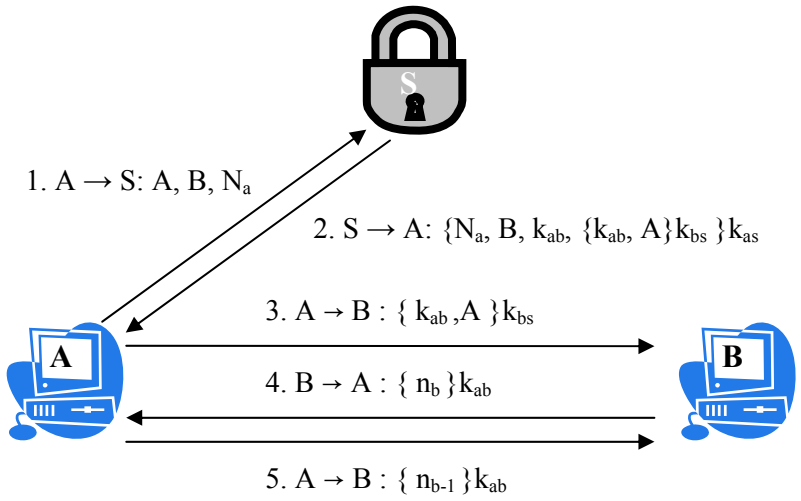
(1) $A \rightarrow S : A, B, N_a$

(2) $S \rightarrow A : \{N_a, B, k_{ab}, \{ k_{ab}, A \}k_{bs} \}k_{as}$

(3) $A \rightarrow B : \{ k_{ab}, A \}k_{bs}$

(4) B → A : { n_b }k_{ab}

(5) A → B : { n_{b-1} }k_{ab}



Obr. 2.1 Komunikácia prostredníctvom protokolu *Needham-Schroeder*

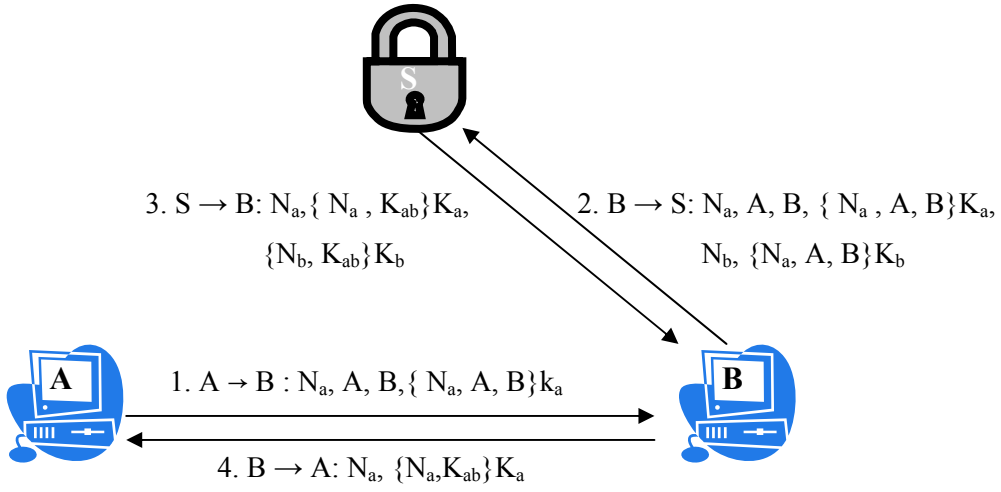
Otway-Rees, symetrický, symbolický zápis, KTC:

(1) A → B : Na, A, B, { Na, A, B }Ka

(2) B → S : Na, A, B, { Na, A, B }Ka , Nb, { Na, A, B }Kb

(3) S → B : Na, {Na, Kab }Ka , { Nb, Kab }Kb

(4) B → A : Na, { Na, Kab }Ka



Obr. 2.2 Komunikácia prostredníctvom protokolu *Otway-Rees*

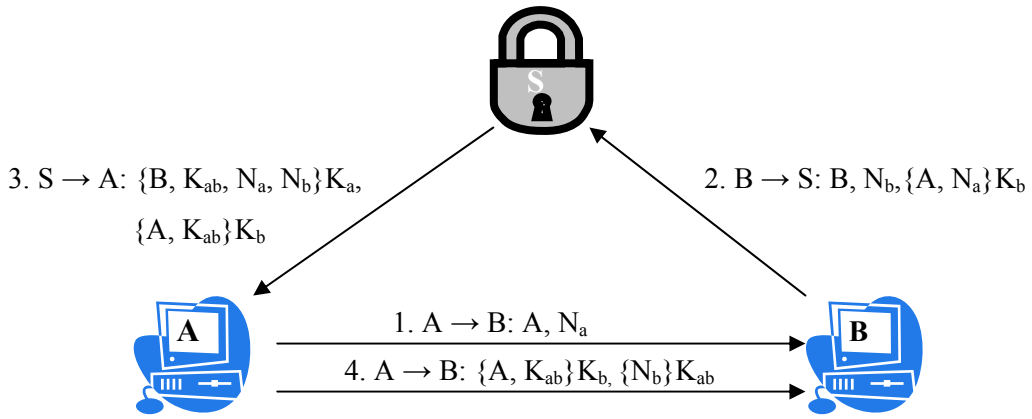
Yahalom, symetrický, symbolický zápis, KTC:

(1) A → B : A, Na

(2) B → S : B, Nb, { A, Na }Kb

(3) S → A : { B, Kab, Na, Nb }$_{Ka}$, { A, Kab }$_{Kb}$

(4) A → B : { A, Kab }$_{Kb}$, { Nb }$_{Kab}$



Obr. 2.3 Komunikácia prostredníctvom protokolu *Yahalom*

2.2 Návrh s využitím jednoduchej logiky

Predstavuje vrstvový prístup k vytváraniu protokolov od abstraktného popisu až po implementačnú úroveň [14]. Používa logiku BAN [13] (tvorcov Burrows, Abadi, Needham) pre analýzu autentizačných protokolov, ktorá vznikla za účelom reprezentácie vzťahov medzi jednotlivými subjektmi. Je postavená na množine znalostí (všetkých správ, ktoré sú známe subjektu A) a množine predpokladov (viera subjektu A, že daný subjekt B na opačnej strane má danú množinu znalostí). Každá operácia v protokole mení časť množín znalostí a predpokladov.

Obsahuje 7 základných pravidiel, podľa ktorých môžme špecifikovať protokol, následne pomocou teorémov a syntetických pravidiel odvodiť systematicky korektnú verziu protokolu. Rovnakým spôsobom sa dá verifikovať protokol.

2.2.1 Základné pravidlá logiky

1. $P \triangleleft C(X)$: *P* vidí správu *C(X)*. Niekto poslal správu *X* cez kanál *C*, *P* si toho všimol, ale ak nemôže čítať kanál *C*, potom nemôže rozpoznať správu ani konkrétny kanál.
2. $P \triangleleft X \mid C$: *P* vidí správu *X* cez kanál *C*. *P* dostal správu *X* cez kanál *C*. Toto je možné v prípade, ak niekto poslal správu a P má práva na čítanie kanálu *C*.
3. $P \triangleleft X \mid$ *P* vidí *X*. Niekto poslal správu *X* cez kanál, ktorý je *P* schopný čítať.
4. *#(X): X* je čerstvé. Nebolo použité nikdy pred tým. Platí napr. pre nonce.
5. $P \mid\sim X$: *P* raz poslal správu *X*. Niekedy v neznámom čase poslal správu *X*.
6. $P \mid\mid\sim X$: *P* nedávno poslal *X*. Znamená, že *P* poslal správu v aktuálnom behu protokolu.
7. $P \mid\equiv \phi$ *P* verí vo formulu ϕ. *P* verí že ϕ je pravdivé. To znamená že síce nemusí byť pravdivé, ale *P* tomu verí.

Pomocou týchto pravidiel, dedukčných pravidiel a teorémov sa vytvoria syntetizované pravidlá, ktoré sa použijú na konštrukciu protokolu.

2.2.2 Príklad návrhu protokolu použitím syntetizovaných pravidiel:

Príklad dedukčného pravidla videnia správy (S1): Ak *P* dostane správu *X* cez kanál, a *P* môže čítať tento kanál, potom *P* rozpozná, že správa prišla cez kanál *C* a môže ju prečítať.

(S1):

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

Syntetizované pravidlo bude potom vyzerať nasledovne:

(Syn1):

$P \mid\equiv (P \triangleleft X \mid C)$

$\Rightarrow P \triangleleft C(X)$

$\Rightarrow P \in r(C)$

Syntetizované pravidlá sa ďalej použijú na konštrukciu protokolu:

Štandardná definícia *Woo a Lam* protokolu:

(1) $B \rightarrow A : n_B$

(2) $A \rightarrow B : \{n_b\ k_{as},$

(3) $B \rightarrow S : \{A,\ \{n_b,\}k_{as},\ \}k_{bs}$

(3) $S \rightarrow B : \{n_b\}k_{bs}$

- prepísaná do logiky:

$B \triangleleft A$

$A \triangleleft N_b$

$B \triangleleft C_{as}(N_b)$

$S \triangleleft C_{bs}(A, C_{as}(N_b))$

$B \triangleleft C_{bs}(N_b)$

- nasleduje definícia predpokladov pre zvolený protokol:

(A1) $A \in r(C_{as})$.....*A* môže čítať kanál C_{as}

(A2) $S \in r(C_{as})$.....*S* môže čítať kanál C_{as}

(A3) $A \mid\equiv (w(C_{as}) = \{A, S\})$…….....*A* verí že len *A* a *S* môže zapisovať do kanálu C_{as}

(A4) $S \mid\equiv (w(C_{as}) = \{A, S\})$…….*S* verí že len *A* a *S* môže zapísovať do kanálu C_{as}

(A5) $B \in r(C_{bs})$.....*B* môže čítať z kanálu C_{bs}

(A6) $S \in r(C_{bs})$.....*S* môže čítať z kanála C_{bs}

(A7) $B \mid\equiv (w(C_{bs}) = \{B, S\})$…….....*B* verí že len *B* a *S* môžu zapisovať kanálu C_{bs}

(A8) $S \mid\equiv (w(C_{as}) = \{B, S\})$… ……*S* verí že len *B* a *S* môžu zapisovať kanálu C_{bs}

(A9) $A \mid\equiv ((S \mid\sim \phi) \rightarrow \{S \mid\equiv \phi\})$……..*A* verí, že *S* je legitímny

(A10) $A \mid\equiv ((S \mid\equiv (B \mid\sim X)) \rightarrow \{B \mid\sim X\}$....*A* verí že *S* je kompetentné rozhodnúť či *B* poslal správu

(A11) $B \mid\equiv ((S \mid\sim \phi) \rightarrow \{S \mid\equiv \phi\})$……...*B* verí, že *S* je legitímny

(A12) $B \mid\equiv ((S \mid\equiv (A \mid\sim X)) \rightarrow \{A \mid\sim X\})$…*B* verí že *S* je kompetentné rozhodnúť či *A* poslal správu

(A13) $B \mid\equiv \#(N_b)$……....*B* verí že nonce N_b je čerstvé

Cieľom protokolu je presvedčiť, že B hovorí naozaj s A, tento cieľ sa dá dosiahnuť len vtedy ak bude platiť nasledovná formula: B |≡ (A ||~ (B, Nb)), ktorá znamená že B verí, že A práve poslal správu Nb pre príjemcu B. Vychádza sa teda z požadovanej formule cieľa a použijú sa syntetizované pravidlá. Postupným aplikovaním vzniknú pravidlá, ktoré sú jedným z predpokladov pre normalizovaný protokol, alebo sú priamo časťami správ protokolu. Výsledný protokol bude tvaru:

$A \triangleleft (B, Nb)$

$S \triangleleft Cas(B, Nb)$

$B \triangleleft Cbs(A \mid\sim (B, Nb))$

Ktorý by sa dal prepísať do pôvodnej podoby ako :

(1) $A \rightarrow B : A$

(2) $B \rightarrow A : B, n_b$

(3) $A \rightarrow B : \{B, n_b,\}k_{as}$

(3) $B \rightarrow S : A, \{B, n_b\}k_{as}$

Princípy logiky BAN sú dobre využiteľné ku generovaniu a k analýze vygenerovaných protokolov [18]. Tým sa stáva jedným zo základných kameňov v procese automatizovaného návrhu.

2.3 Návrh pomocou odvodenia, derivačný systém

Niektoré skupiny protokolov sa dajú odvodiť od „základných“ protokolov systematicky a tým sa dajú skladať aj ich vlastnosti [12]. Napríklad *Diffle-Hellman key exchange* protokol, ktorý sa stará

a o zdieľaný kľúč a *Challenge-Response* protokol, ktorý sa stará o autentizáciu komunikujúcich subjektov, ale už nie o kľúč. Ich kompozíciou je možné získať výhodné vlastnosti ich oboch.

Princíp spočíva v tom, že každému protokolu sa priradí komponenta, zdokonalí sa a transformuje na logickú formulu, ktorá ho vyjadruje. Odvodenie stojí na troch základných stavebných blokoch a operáciách pre konštrukciu nových: *Kompozícií*, *transformácií*, a *zdokonalení*.

Kompozícia skladá základné stavebné prvky protokolu do väčších celkov. *Transformácia* pracuje nad jedným protokolom. Ide v podstate o prepisovaciu operáciu, ktorá môže modifikovať niekoľko krokov protokolu presunutím údajov v správe do inej správy, kombinovať kroky, alebo pridať kroky. Napr. premiestniť údaje zo správy protokolu do skoršej správy protokolu medzi dvomi rovnakými komunikujúcimi stranami. *Zdokonalenie* sa stará o časti správ, napr. nonce nahradí jej zašifrovanou verziou, pritom nemení štruktúru protokolu ani počet správ.

2.3.1 Príklad vytvorenia derivačného stromu, odvodenia protokolov

Diffle-Hellman protokol slúži na ustanovenie tajného kľúča medzi dvoma stranami:

A: vygeneruje náhodnú hodnotu a a vypočíta g^a

B: vygeneruje náhodnú hodnotu b a vypočíta g^b

Na vopred dohodnutom základe z.

Chalenge-Response zas poskytuje autentizáciu:

$A \rightarrow B:m$

$B \rightarrow A: SIG_B(m)$

Kompozícia potom bude vyzerať nasledovne:

$A \rightarrow B: m$

$B \rightarrow A: SIG_B(m)$

$B \rightarrow A: n$

$A \rightarrow B: SIG_A(n)$

- jednoducho sa zložia 2x za sebou kroky *Chalenge-Response* protokolu

Transformácia 1:

$A \rightarrow B: m$

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

$A \rightarrow B : SIG_A(n)$

- zjednoduší sa krok 2 a 3 do jeného výsledného kroku 2

Transformácia 2:

$A \rightarrow B: m$

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

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

- pridá sa nonce n a m do oboch krokov 2 a 3

Kompozícia s Diffle-Hellman:

$A \rightarrow B: g^a$

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

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

- použil sa nonce na zašifrovanie

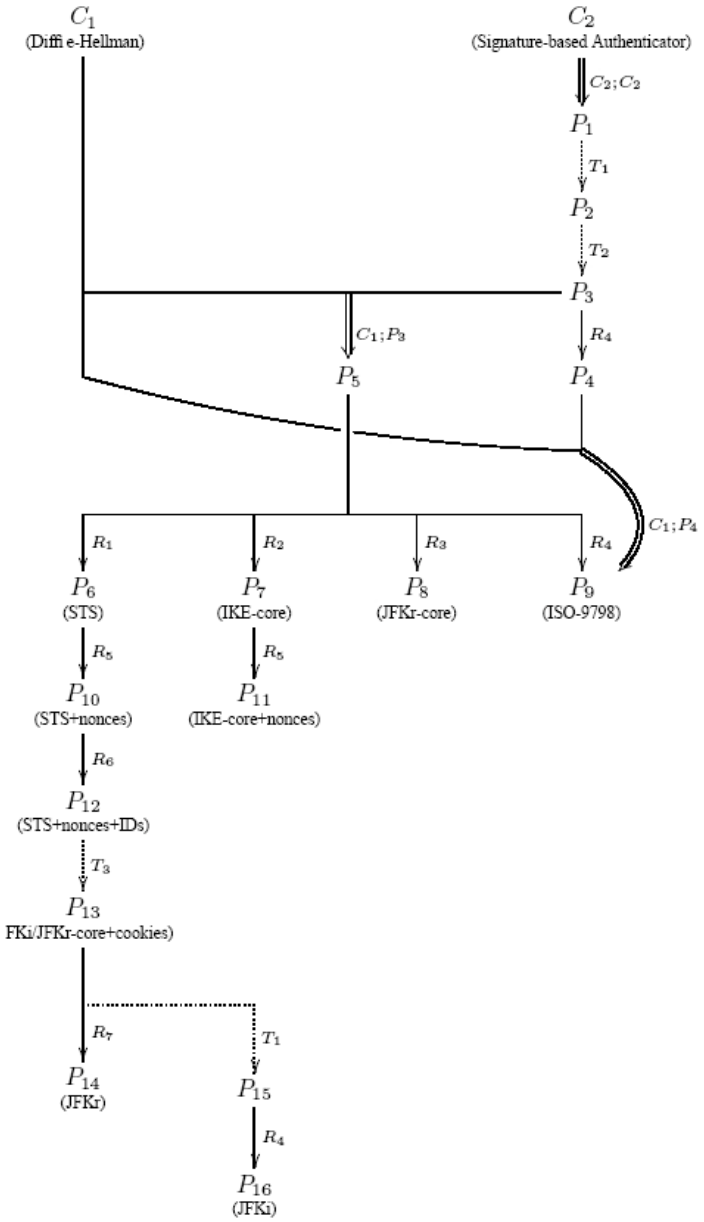
Postupne zdokonalením vznikne *STS (Station To Station)* protokol, kde K je zdieľaný kľúč odvodený z *Diffe-Hellman*, kvôli ochrane identity voči pasívnym útokom:

$A \rightarrow B: g^a$

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

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

Postupným aplikovaním *kompozície, transformácie* a *zdokonalenia* sa príde až k IKE a JFK ako znázorňuje Obr.2.4.



Obr.2.4 Derivačný graf pomocou aplikovania *kompozície, transformácie a zdokonalenia*

Získal sa tak derivačný graf *STS* protokolov. Návrh pomocou odvodzovania je vhodný pre *Key Exchange* protokoly, *STS* protokoly, *IKE*, *JFKi*, *JFKr.*

2.4 Návrh pomocou evolučných algoritmov

S pomedzi automatizovaných techník využívaných pri návrhu protokolu patrí medzi tie najmladšie. Založená je na simulácií prirodzeného kolobehu života v prírode a javí sa byť najperspektívnejšou v kombinácií s logikou BAN. Technikám založeným na evolučnom princípe sa podrobne venujem v nasledujúcej kapitole.

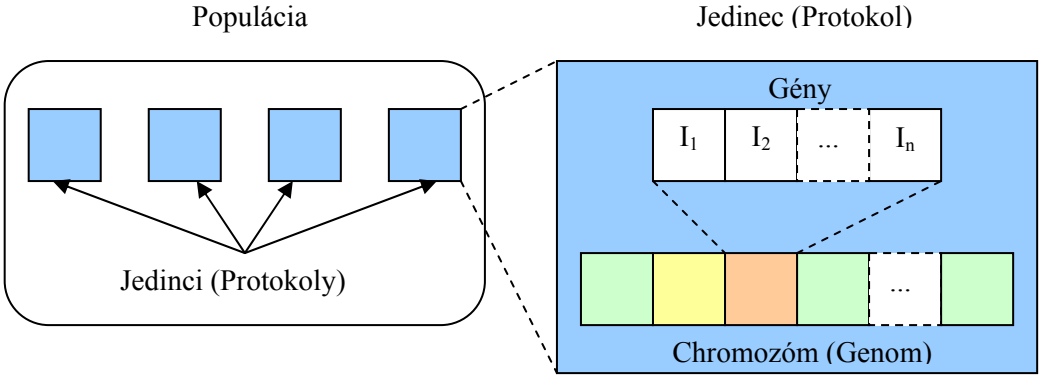
3 Vybrané evolučné a príbuzné algoritmy

3.1 Genetické algoritmy ako základ EA

Genetické algoritmy (GA) [15][16] vznikli v polovici 50. rokov minulého storočia, naplno sa však začali rozvíjať až v polovici 70. rokov, kedy John Holland definoval ich základné princípy a vlastnosti.

Simulujú pravidlá prirodzeného vývoja v prírode, kde sa v cykle z počiatočnej populácie krížením a mutáciou vyvinie populácia s lepšími (požadovanými) vlastnosťami.

Z pohľadu genetických algoritmov je protokol jedinec zakódovaný sekvenciou inštrukcií. Sekvencia inštrukcií protokolu je reprezentovaná chromozómom premenlivej dĺžky, na ktorý sa aplikujú genetické operátory (kríženie a mutácia). Ukazovateľom kvality vlastností je funkcia fitness.



Obr. 3.1 Štruktúra populácie: Protokol (jedinec) zakódovaný ako sekvencia inštrukcií (I_1 ... I_n).

V definícií GA je požadovaný chromozóm pevnej dĺžky. Existujú však upravené formy genetických algoritmov ako napr. Genetické Programovanie, ktoré rátajú s premenlivou dĺžkou reťazca.

3.1.1 Všeobecný postup genetického návrhu

a) Špecifikácia protokolu

Na začiatku návrhu protokolu je nutné definovať základné požadované vlastnosti ako napr.: koľko subjektov sa bude zúčastňovať komunikácie, ktoré základné inštrukcie a kryptografické operácie sa budú používať, ktoré časti správ sa budú môcť prenášať počas komunikácie a ktoré nie, a hlavne aké bude poradie inštrukcií pre výmenu správ, ktoré budú mať vplyv na zmenu množín znalostí

a predpokladov. Čím lepšia špecifikácia požadovaných vlastností, množín znalostí a predpokladov, tým bezpečnejší protokol dostaneme na konci evolučného procesu.

b) Vytvorenie počiatočnej populácie

Náhodne sa vygeneruje toľko protokolov ako je požadované pre počiatočnú populáciu, každý protokol - jedinec je reprezentovaný ako reťazec (chromozóm = sekvencia inštrukcií protokolu).

c) Výpočet fitness

Vypočíta sa fitness hodnota jednotlivých protokolov, simulovaním ich behu podľa množiny znalostí a predpokladov.

d) Spĺňa požadované kritériá?

Na základe predchádzajúceho ohodnotenia každého protokolu sa dá určiť či sa s v množine protokolov nachádza jedinec s počiatočnými požadovanými vlastnosťami. Ak áno algoritmus je na konci, ak nie pokračuje. Podmienky ukončenia algoritmu môžu byť rôzne. Populácia môže konvergovať k jedinému riešeniu, alebo môže obsahovať dostatočne dobré riešenia, alebo dosiahneme maximálny možný počet generácií.

e) Voľba rodičov

Protokoly s najlepšou fitness hodnotou sa vyberú pre kríženie.

f) Kríženie

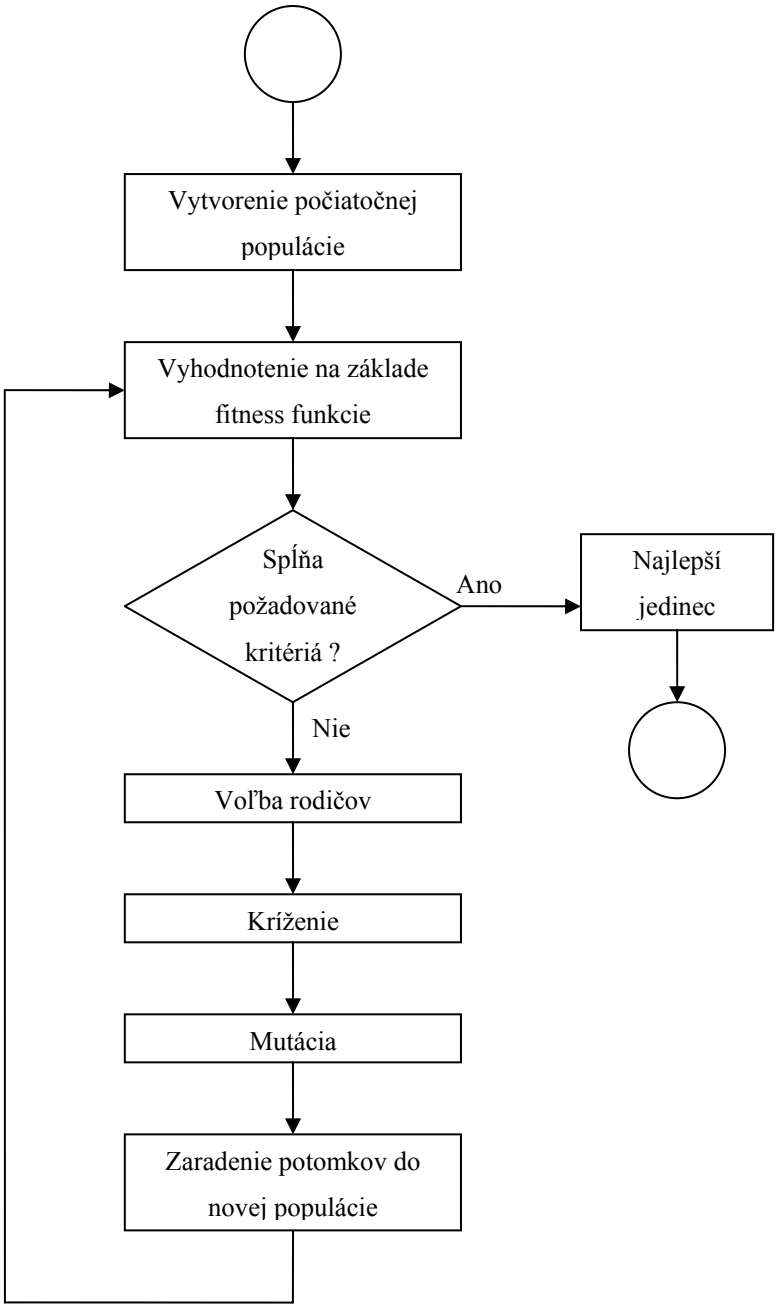
Vybraným jedinci si vymenia časti ich genetickej informácie. Voľba správneho miesta v chromozómoch na kríženie je veľmi dôležitá, ovplyvní hodnotu fitness v nasledujúcej generácií.

g) Mutácia

Aby sa vyhlo uviaznutiu populácie v istom bode, vykonajú sa náhodné zmeny v inštrukciách, pričom sa tým ovplyvnia obe množiny znalostí a predpokladov.

h) Nástup novej generácie

Jedincami, ktorí takto vzniknú sa nahradí stará populácia a celý proces evolúcie sa opakuje. Forma náhrady závisí od použitej metódy, je možné nahradiť celú populáciu, časť populácie... atď..

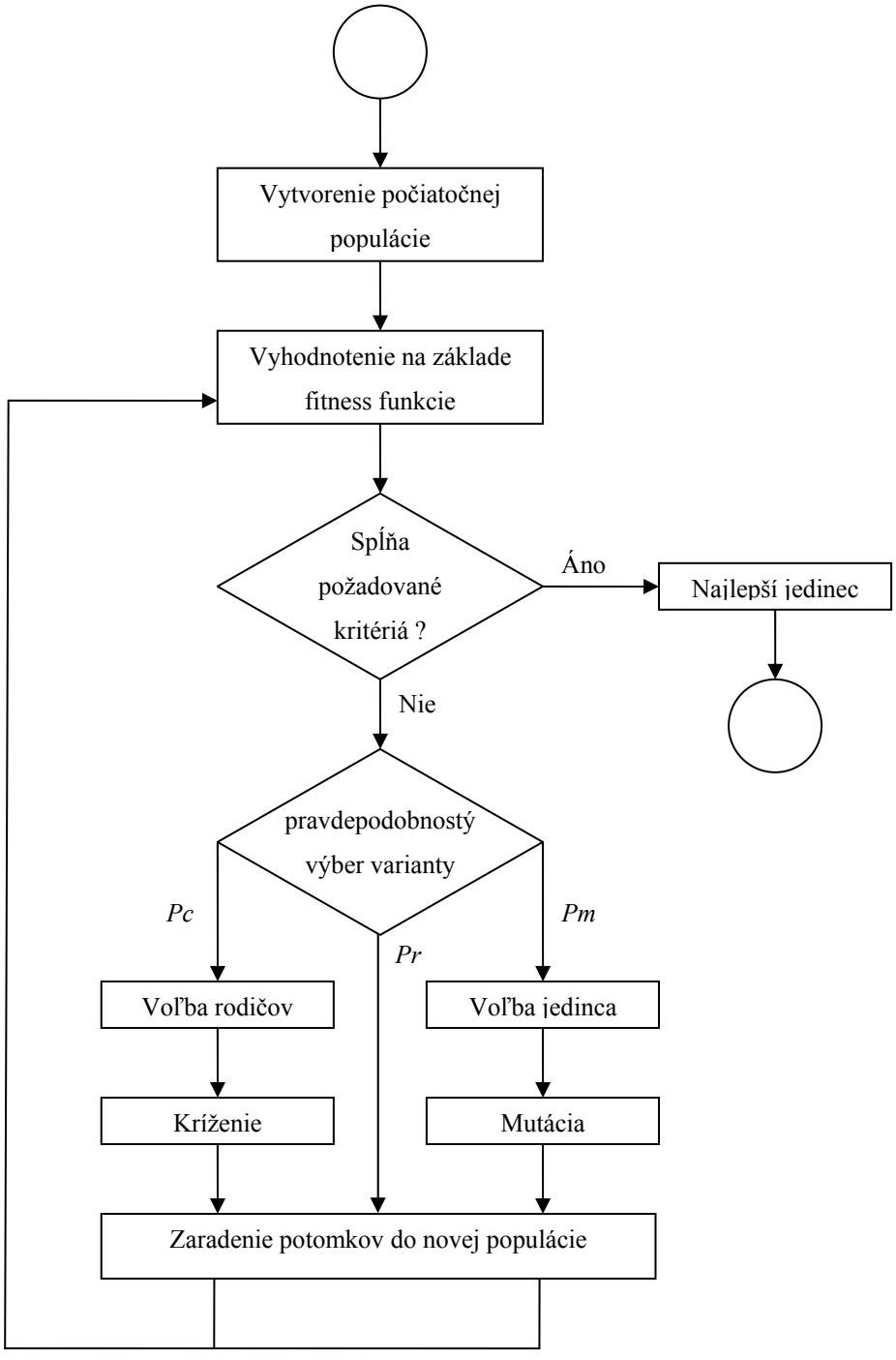


Obr. 3.2 Priebeh Genetického Algoritmu

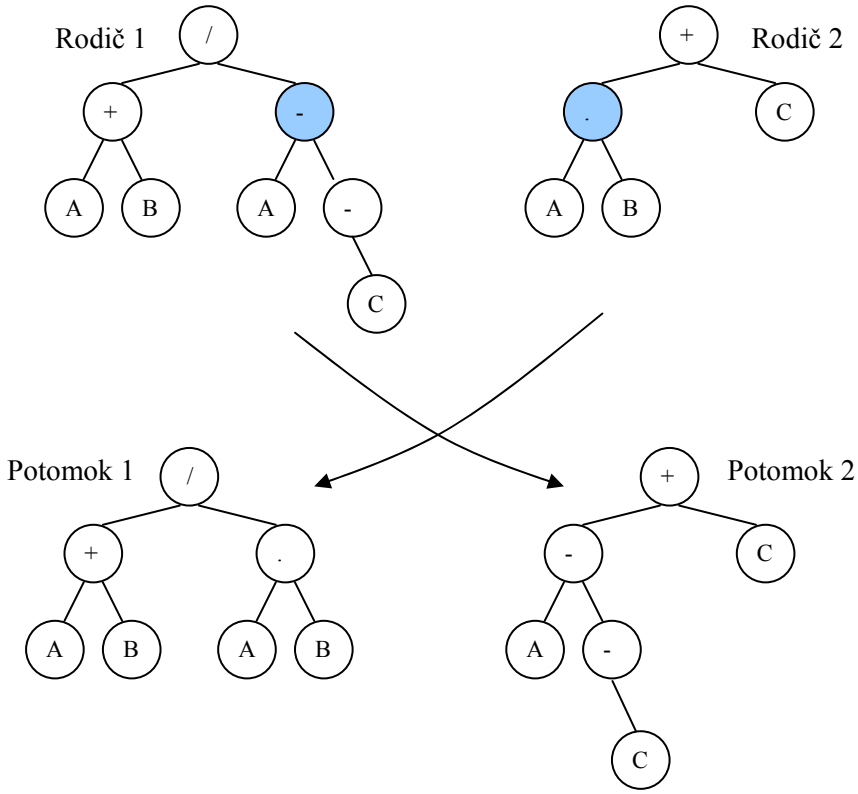
3.2 Genetické programovanie

Genetické programovanie vzniklo v 80. rokoch minulého storočia a o jeho vznik sa najviac zaslúžil John Koza [3]. Genetické programovanie (GP) je špecializovaná forma genetického algoritmu, kde genotyp = fenotyp, čo znamená že sa pracuje priamo s jedincami a nie s ich kódovanou reprezentáciou. Na rozdiel od GA nie je vyžadovaná rovnaká dĺžka chromozómov, čo je vhodné u protokolov, pretože sa skladajú z chromozómov premenlivej dĺžky.

GP reprezentuje riešenia hierarchickým spôsobom (strom), kde sa zamieňajú (krížia) medzi sebou jednotlivé časti (podstromy) v náhodne zvolených uzloch. Má zmenený operátor mutácie, ktorý vyberie náhodne uzol stromu a zmaže všetko pod ním. Následne ho nahradí náhodne vygenerovaným podstromom. Oproti GA dochádza k výberu operácie či už kríženia, mutácie alebo prekopírovania jedinca zo starej populácie do novej, na základe zvolenej pravdepodobnosti *Pc* (pre kríženie), *Pm* (pre mutáciu), *Pr* (pre kopírovanie).



Obr. 3.3 Priebeh algoritmu u Genetického Programovania



Obr. 3.4 Princíp kríženia v GP

Pri výmene podstromov by sa malo pomocou heuristiky rozlišovať, ktoré z nich je možné medzi sebou zameniť bez toho, aby sa zmenila sémantika inštrukčnej sekvencie [1].

3.3 Gramatická evolúcia

Gramatická evolúcia (GE) [7][8] je variantou GP. Používa kombináciu chromozómu s premenlivou dĺžkou a zápis gramatiky jazyka vo forme BNF (Backus Naurovej Forme) produkčných pravidiel. Na rozdiel od klasického použitia, napr. bezkontextovej gramatiky v prekladačoch (parser) ku kontrole syntaktickej správnosti programu, sa gramatika v GE využíva na generovanie programu. Tým je zaručená syntaktická správnosť.

3.3.1 BNF a GE

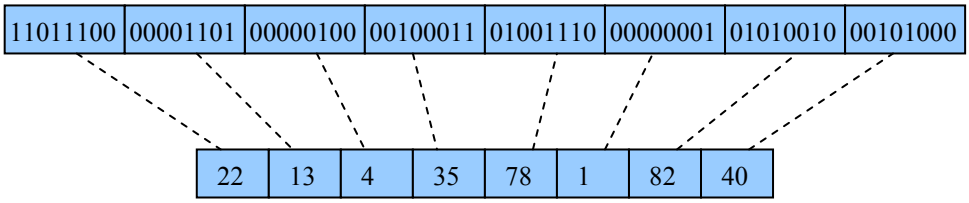
Backus Naurova Forma (BNF) je zápis pomocou, ktorého sa vyjadruje gramatika jazyka vo forme produkčných pravidiel. Pozostáva z terminálov, ktoré predstavujú jednotlivé časti jazyka a nonterminálov, ktoré sa môžu rozvinúť na viac nonterminálov alebo terminálov. Je to štvorica $\{N, T, P, S\}$, kde N je množina nonterminálov, T je množina terminálov, P je množina produkčných pravidiel, ktoré mapujú nonterminály na terminály a S je štartovací symbol, kde $S \in N$.

Rôzne skupiny protokolov môžu byť reprezentované pomocou bezkontextovej gramatiky, ktorá je ekvivalentom BNF.

Príklad zápisu bezkontextovej gramatiky pre symetrické autentizačné protokoly :

```
N = {start, message, next, krypt, genkey, nonce, sub}
T = {send, :, ,, {, }, key, N, N-1, a, b, s}
S = <start>
P :
        <start> ->  <sub> send <sub> : <message>
        <message> -> <sub> <next>
                    | <nonce> <next>
                    | <krypt> <next>
                    | <genkey> <next>
        <next> ->   , <message>
                    | e
        <krypt> ->  { <message> } <genkey>
        <genkey> -> key <sub> <sub>
                    | key <sub>
        <nonce> ->  N <sub>
                    | N-1 <sub>
        <sub> ->    a
                    | b
                    | s
```

V GE na rozdiel od GP dochádza k (de)kódovaniu z genotypu na fenotyp. Každú inštrukciu protokolu predstavuje chromozóm, ktorý pozostáva z binárnych čísiel, génov - kodónov (8 bitových blokov). Gény obsahujú čísla pre výber pravidiel z definovanej gramatiky.



Obr. 3.5 Chromozóm jedinca (binárna reprezentácia) a jeho mapovanie, kde každý gén reprezentuje náhodné číslo, ktoré sa použije pri dekódovaní z genotypu na fenotyp.

Princíp:

Príklad pre jeden nonterminál, ktorý produkuje 4 rôzne pravidlá gramatiky:

```
<message> ->  <sub> <next>    (0)
            | <nonce> <next>  (1)
            | <krypt> <next>  (2)
            | <genkey> <next> (3)
```

Z chromozómu náhodne vygenerovaných génov sa vyberie prvý gén a prevedie sa jeho binárna hodnota na do desiatkovej sústavy, následne sa spočíta pseudo náhodné číslo pre výber pravidla na základe vzorca:

```
Pravidlo = (hodnota génu) MODULO (počet pravidiel pre daný nonterminál)
```

V tomto príklade je číslo prvého génu = 22, pričom pre daný nonterminál je možné vyberať zo 4. pravidiel. Pomocou vzorca 22 MODULO 4 = 2 sa získa pravidlo, ktoré nahradí nonterminál `<message>` t.j. pravidlom č.2 `<krypt> <next>`. Postupne sa takto zľava generujú a prechádzajú gény chromozómu, ktoré určujú výber jednotlivých pravidiel z gramatiky, až pokiaľ nenastane jedna nasledujúcich situácií:

a) Vygeneruje sa kompletná inštrukcia, t.j. všetky nonterminály sa transformovali na terminály definované v gramatike.
b) Dosiahne sa koniec chromozómu, v tom prípade sa vyvolá baliaci operátor, kde výsledok vráti na začiatok chromozómu a čítanie génov pokračuje od znovu, až kým sa nedosiahne horná hranica reprezentujúca maximálny počet baliacich udalostí počas procesu mapovania jedinca. T.j. maximálny počet zabalení.
c) V prípade, že sa objaví hranica počtu baliacich operácií a jedinec ešte stále nie je kompletne namapovaný, mapovací proces sa zastaví a jedincovi je pridelená najnižšia možná hodnota fitness.

GE používa ustálený výmenný mechanizmus, takže dvaja rodičia vyprodukujú dvoch potomkov, kde najlepší vymenia najhorších jedincov v aktuálnej populácií na základe fitness. Využívajú sa štandardné genetické operátory ako *kríženie*, *mutácia* (aplikovaná s pravdepodobnosťou *Pm* na každý bit chromozómu), špecifické ako *kopírovací operátor* a *orezávací operátor*. *Kopírovací operátor* zduplikuje náhodne číslo/a génov a vloží ich na poslednú pozíciu génu v chromozóme. *Orezávací operátor* oreže časti chromozómu, ktoré sa nepoužili pri mapovaní z genotypu na fenotyp, takzvané *introny*.

Nevýhody GE pre protokoly sú v tom, že je možné vygenerovať len jednotlivé inštrukcie protokolu, ktoré nevieme ohodnotiť samostatne v kontexte iných inštrukcií. Môžeme z nej však použiť gramatiku ako základ pre definovanie gramatických pravidiel pre rôzne typy komunikačných protokolov, čím sa položia mantinely pri vytváraní populácie a zaručí sa ich syntaktická správnosť.

3.4 Simulované žíhanie

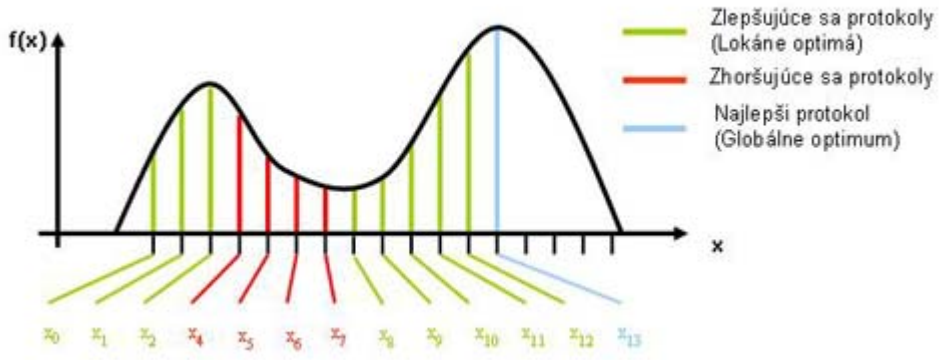
Je heuristická technika (SA), ktorá sa začala uplatňovať v 80. rokoch 20. storočia pri hľadaní globálneho minima (maxima) [17]. Vychádza z fyzikálneho princípu žíhania tuhého telesa k odstráneniu vnútorných defektov a pnutí.

Fyzikálny princíp :

Teleso najprv sa zahreje na vysokú teplotu, tým sa umožní atómom prekonať lokálne energetické bariéry a dostať sa do rovnovážnych polôh. Následne sa teleso pomaly ochladzuje, čím sa fixujú atómy v rovnovážnych polohách. Pri konečnej teplote žíhania (nižšej ako je počiatočná) sú atómy v rovnovážnych polohách a teleso neobsahuje vnútorné defekty ani pnutia.

Aplikovaný na protokoly:

Požadovanú teplotu T môže predstavovať napr. hodnota fitness protokolu. Na rozdiel od fyzikálneho princípu, kde sa začínalo s vysokou teplotou T a postupne sa znižovala, tu je postup opačný. Teplota T začína nízko (maximalizačná formulácia) a postupne sa zvyšuje so skúmaním lokálneho okolia. Na začiatku sú akceptované všetky protokoly bez rozdielu. Zo zvyšujúcou sa teplotou T sú už protokoly s nízkou fitness akceptované len s istou pravdepodobnosťou, kým protokoly z vysokou fitness sú akceptované stále. Postupne sa takto dosiahne globálne optimum, tj. protokol s najlepšou fitness.



Obr. 3.6 Princíp Simulovaného Žíhania

3.5 Evolučná knižnica GAlib

V dnešnej dobe existuje viacero knižníc, ktoré podporujú rôzne evolučné algoritmy. Medzi najznámejšie patria GAlib [4], Beagle [5] a EO [6]. V mojej práci som si zvolil knižnicu GAlib.

Genetic Algorithm library je knižnica pre C++ určená pre prácu s GA. Na začiatku GA je inicializovaná štartovacia populácia, táto populácia obsahuje určitý počet riešení, ktoré sú v každom cykle (generácií) vyhodnotené fitness funkciou. Následne nastáva reprodukcia, kedy sú lepšie riešenia vyberané s väčšou pravdepodobnosťou ako horšie riešenia a sú na ne aplikované genetické operátory. Najlepší jedinci, alebo aj všetci z reprodukovaných sú vybraní do ďalšej generácie a celý cyklus sa opakuje, kým nenarazíme na ukončovacie kritérium resp. požadované riešenie.

Výhodou GAlibu je možnosť pomerne jednoducho predefinovať podľa vlastných potrieb akékoľvek operátory alebo parametre a preto je vhodná ako základná knižnica pre automatizovaný návrh komunikačných protokolov. S využitím vlastných operátorov a funkcií.

3.5.1 Základné prvky programu

Objekt genetického algoritmu

Najdôležitejším prvkom v GAlibe je objekt genetického algoritmu, ktorý zahŕňa všetky kontrolné premenné GA vrátane pravdepodobnosti operátorov, priradenia samotných funkcií reprezentujúcich operátory, ukončovacie kritérium a hlavne je v ňom obsiahnutá definícia typu samotného algoritmu. Typy algoritmov sú nasledovné štyri:

GASimpleGA

Je jednoduchý štandardný algoritmus, ktorý v každej generácií používa úplne novú populáciu, ale je možné zvoliť tzv. elitizmus, kedy sa najlepší jedinec z predchádzajúcej populácie prenesie do nasledujúcej.

GASteadyStateGA

U tohto typu algoritmu je možné pracovať s prekrývajúcou sa populáciou, kde je možné určiť aká časť populácie bude naradená.

GAIncrementalGA

V inkrementálnom GA je možné definovať spôsob, akým sa má včleňovať nová populácia do aktuálnej. Potomkovia môžu nahradzovať svojich rodičov, náhodne vybrané jedince alebo jedince sebe najviac podobné.

GADemeGA

Vo štvrtom a poslednom základnom type je paralelne vyvíjaných niekoľko populácií s použitím *steady-state* algoritmu. V každej generácií je niekoľko jedincov z jednej populácie presunutých do niektorej z ďalších.

Pre vyššie uvedené typy sa musí nadefinovať príslušný hlavičkový súbor v našom programe. Existuje aj možnosť vytvoriť svoj vlastný typ GA. Z hľadiska automatizovaného návrhu protokolov však zatiaľ budú stačiť základné typy GA.

Dôležitou súčasťou knižnice sú štatistiky a vlastnosti, ktoré sa dajú dobre využívať k ladeniu algoritmu a zobrazovaniu všetkých potrebných informácií o tom ako prebehol výpočet.

GA vyžaduje pre prácu kódovanú reprezentáciu (genotypy), ktoré sa potom mapujú do skutočnosti a získajú sa tak skutočné riešenia (fenotypy). Každá štruktúra obsahujúca jediné riešenie sa nazýva genóm (v našom prípade genóm reprezentuje protokol a inštrukcie sú jeho gény). Je veľmi dôležité, aby bola štruktúra čo najmenšia, pretože sa tým znižuje celková doba výpočtu. Musí však reprezentovať všetky dôležité aspekty problému.

Reprezentácia

V GAlibe je k dispozícií široká škála preddefinovaných štruktúr ako je napríklad pole, strom, binárny reťazec, zoznam. Používa šablóny, čiže je možné definovanie rôznych typov. Pre každú štruktúru, ktorú chceme používať sa musí uviesť príslušný hlavičkový súbor, podobne ako tomu bolo pri voľbe typu GA.

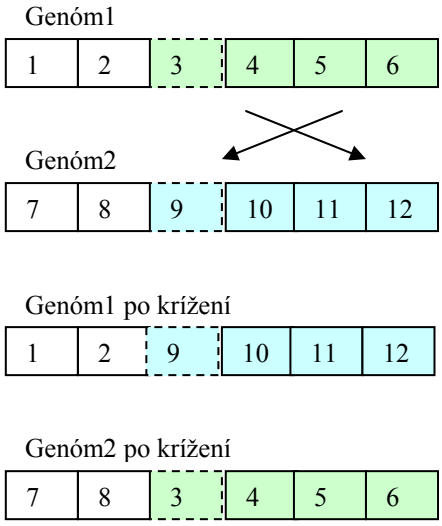
Čím sú dátové typy a štruktúry genómu zložitejšie, tým väčší pozor je treba dávať pri definovaní genetických operátorov.

Definícia genetických operátorov

Pre genóm existujú tri základné genetické operátory: *inicializácia, kríženie a mutácia.* V GAlibe sú tieto operátory preddefinované, no je možné ich upraviť podľa vlastných potrieb. Predstavujú ďalší krok pred samotným spustením výpočtu.

Inicializácia je operátor, ktorý sa volá len na začiatku GA a nevytvára žiadne nové genómy, ale implantuje genetický materiál do už existujúcich v počiatočnej populácií. Zvyčajne sú to náhodné hodnoty, s prihliadnutím na zmysluplnosť pre daný typ riešeného problému.

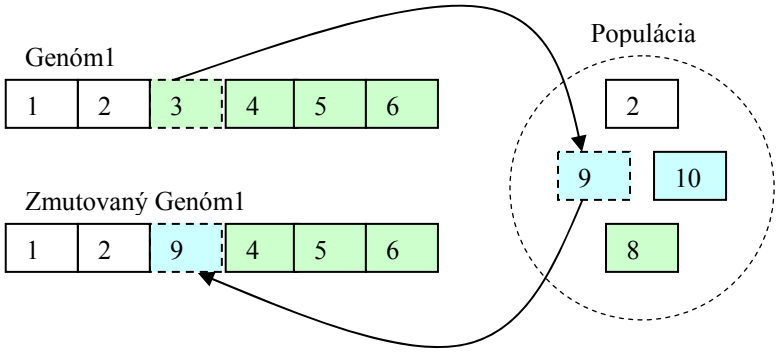
Kríženie je najpoužívanejší genetický operátor a zvyčajne sa jeho pravdepodobnosť pohybuje okolo 70%, ale sú aj prípady kedy sa vynecháva a dáva sa prednosť mutácií. Máme k dispozícií niekoľko typov krížení. Základné *jednobodové,* ktoré funguje tak, že sa náhodne vyberie bod kríženia u dvoch rodičovských jedincov a druhá časť genómu za bodom kríženia sa prehodí.



Obr. 3.7 Princíp jednobodového kríženia

V GAlibe okrem štandardného *jednobodového* operátoru kríženia existujú aj iné možnosti, ako je napríklad *viacbodové* kríženie, alebo kríženie *viacerých rodičov*. Operátor je definovaný pre rôzne dátové typy.

Operátor ***mutácie*** slúži k modifikácií už existujúceho genetického materiálu, alebo dokonca zameneniu za úplne nový. Je to spôsob akým sa dá pohnúť z lokálneho minima, čo je problém ktorý sa často vyskytuje u zložitejších úloh v GA. Môže vyzerať tak, že dôjde k prehodeniu dvoch náhodných génov, výmene dvoch podstromov alebo k vygenerovaniu nového kusu genómu. Nemusí byť vždy prospešná a môže pôsobiť deštruktívne. Preto sa nastavuje obvykle na jednotky percent, aby k nej nedochádzalo príliš často a nedegradovala GA.



Obr.3.8 princíp mutácie

Výhodou GAlibu je to, že používa ukazatele na funkcie k priraďovaniu operátorov genómu a preto je možné meniť operátory za behu algoritmu. Nie je nutné potom ani definovať novú triedu pri zmene chovania vstavaných typov genómu.

Stanovenie objektívnej funkcie

Objektívna funkcia slúži k ohodnoteniu jedinca v rámci populácie a vracia jeho čistú hodnotu. Fitness je potom výsledkom objektívnej funkcie a použitej škálovacej schémy. Tieto dve hodnoty treba rozlišovať. Dá sa totiž zvoliť mapovanie objektívnej hodnoty na hodnotu fitness priamo, alebo použiť niektorú zo škálovacích schém, ktorá transformuje objektívnu funkciu na hodnotu fitness, aby bolo možné čo najlepšie určiť vhodnosť daného jedinca k páreniu. Škálovacia schéma GAlibu je primárne nastavená na *lineárne škálovanie*, no k dispozícii sú ešte 4 ďalšie metódy vrátane priameho mapovania bez škálovania.

Je preto dôležité správne definovať objektívnu funkciu, ktorá má zásadný vplyv na výsledok celého GA.

3.5.2 Dôležité vlastnosti a triedy

GAPopulation

Je objekt, ktorý funguje ako kontajner na genómy. Obsahuje vlastné funkcie na inicializáciu a vyhodnocovanie a tiež udržuje ďalšie štatistické údaje o svojej populácií ako sú napr.: genóm s maximálnou, minimálnou, priemernou fitness, rôzne odchýlky atď..

V populácií je tiež definovaná metóda výberu, ktorá súvisí s reprodukciou a škálovacia schéma. Metód výberu genómov z populácie je v GAlibe viac druhov. Primárne nastavenie predstavuje *ruletový výber* , kde jedinci z vyššou hodnotou fitness majú väčšiu šancu na výber do nasledovnej populácie. Je možné zvoliť niektorú z ďalších metód. Bližší popis je zahrnutý v dokumentácií ku knižnici GAlib.

GAStatistics

Je veľmi užitočný objekt, ktorý ako už názov napovedá udržuje všetky informácie o tom čo sa deje behom celého algoritmu a to nielen na jeho konci ale i počas jednotlivých krokov. Pomocou neho je možné zbierať dôležité údaje, ako napr. zisťovať či sa už dosiahlo ukončovacie kritérium, počet populácií, generácií atď..

4 Knižnica funkcií pre podporu návrhu komunikačných protokolov

4.1 Analýza a návrh

Cieľom knižnice je definovať metódy (funkcie), tak aby umožnili návrh protokolu využitím spomenutých techník a už existujúcich knižníc. Po analýze jednotlivých evolučných prístupov sme dospeli k záveru, že najlepšou metódou vedúcou k automatizovanému vytvoreniu protokolu bude kombinácia viacerých techník.

4.1.1 Výber existujúcich knižníc a ich využitie spolu s knižnicou funkcií

Cieľom knižnice funkcií je v spolupráci s už existujúcimi knižnicami umožniť automatizovaný návrh komunikačných protokolov na evolučnom princípe.

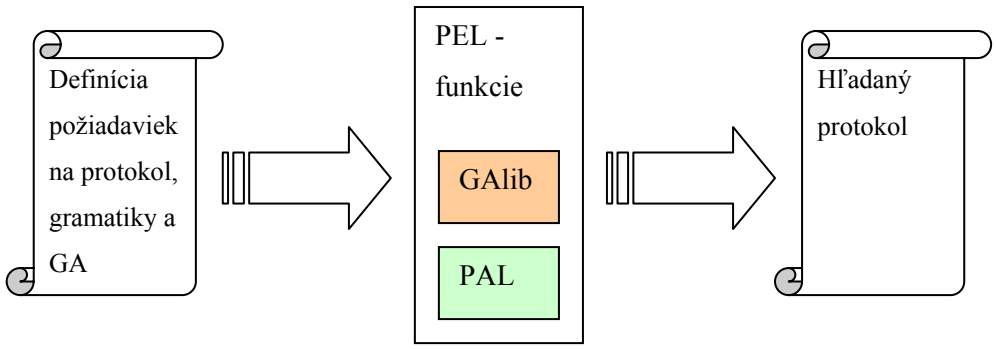
Z knižníc pre GA je vhodná napr. GAlib a z knižníc pre prácu s protokolmi existuje vyvíjaná aplikačná knižnica pre podporu návrhu bezpečnostných protokolov, ďalej len PAL (Protocol Aplication Library), kolegom Karlom Tomáškom, v spolupráci s mojou knižnicou funkcií PEL (Protocol Evolution Library). PEL bude využívať a doplňovať služby PAL a GAlibu.

GAlib som popísal v predchádzajúcej kapitole, a preto by som rád priblížil aj základy knižnice PAL, ktorá vznikla v spolupráci s PEL.

Aplikačná knižnica pre podporu návrhu bezpečnostných protokolov (PAL)

Ide o knižnicu ktorá využíva princípy logiky BAN a dokáže odsimulovať komunikáciu medzi subjektmi v rámci protokolu. Čím sa stala dôležitým nástrojom na hodnotenie protokolov vygenerovaných pomocou GA. Jej súčasťou je generátor protokolu podľa vopred definovaných požiadaviek, ktorý je zas dôležitý pri vytváraní počiatočnej populácie v GA.

Práve generátor sa stal kľúčovým stredobodom našej pozornosti, pretože sme potrebovali vytvoriť náhodné, ale pritom zmysluplné inštrukcie pre protokol. Po neúspešnej ceste za snubne vyzerajúcou GE sme sa rozhodli využiť výhody jej gramatickej časti a preniesli sme gramatiku pre skupiny protokolov na generátor. Tým sa získali mantinely pri vytváraní inštrukcií a zaručili ich syntaktickú správnosť. Bude však nutné definovať pravidlá gramatiky pre každú skupinu protokolov (napr. symetrické autentizačné, asymetrické autentizačné...). Zatiaľ, pre jednoduchosť, na overenie funkčnosti pracujeme len s bezkontextovou gramatikou pre symetrické autentizačné protokoly.



Obr. 4.1 Spolupráca knižníc pri automatizovanom návrhu komunikačných protokolov

4.1.2 Od špecifikácie požiadaviek až po hľadaný protokol

Špecifikácia protokolu:

Základom pre dosiahnutie kvalitného protokolu je správna definícia pravidiel gramatiky (pre danú skupinu protokolov), požiadaviek na komunikáciu (počiatočné, koncové znalostí a predpoklady, zakázané inštrukcie) a voľba parametrov pre samotný GA (požadovaná fitness, počet inštrukcií v protokole, veľkosť populácie, počet generácií).

Vytvorenie počiatočnej populácie a mapovanie do GA:

Na základe predaných parametrov sa pomocou gramatiky vygeneruje počiatočná populácia a odpovedajúca kódovaná reprezentácia pre GA.

PEL bude mať na starosti inicializáciu generátoru a populácie, ktorú naplní vyžitím knižnice PAL a bude sa ďalej starať o mapovanie z genotypu na fenotyp a späť.

Práve mapovanie a voľba kódovania neboli celkom jednoznačné, no nakoniec som využil možnosť reprezentácie pomocou čísla, ktoré by v sebe zahrňovalo číslo protokolu aj číslo inštrukcie. Celočíselné delenie maximálnym počtom inštrukcií v protokole potom určí číslo protokolu, a zvyšok po delení bude určovať číslo inštrukcie. Napr. pre protokol so 100 inštrukciami, kde hodnota prvého génu v genóme je 101, vyjadruje protokol číslo 1 a inštrukciu číslo 2. Pri opačnom mapovaní sa protokol násobí max. počtom inštrukcií a pričíta sa index aktuálnej inštrukcie protokolu. Môže však nastať problém v prípade, že by sa generovalo väčšie množstvo protokolov ako je rozsah čísla daného jeho typom. V našom prípade na testovacie účely zatiaľ postačí typ `int`.

Po vygenerovaní a namapovaní populácie je možné naštartovať GA.

Genetický algoritmus nad protokolmi:

Tu nastupuje využitie knižnice GAlib, ktorá pracuje s kódovanou verziou protokolu v rámci PEL.

Pri inicializácií počiatočnej populácie je nutné ohodnotiť jednotlivé genómy, pre ďalšie pokračovanie v GA.

Výpočet fitness pomocou simulácie:

Fitness je srdce GA, pretože sa od nej odvíja celý evolučný proces. Pre správny výpočet objektívnej hodnoty genómu, je potrebné nadefinovať vlastnú funkciu, ktorá dekóduje vybraný genóm z populácie a odsimuluje využitím PAL. Získajú sa tak hodnoty pre výpočet výslednej fitness. Vzorec pre výpočet fitness by mal vystihovať ciele protokolu, ktoré chceme dosiahnuť.

Zvolil som nasledovných 6 parametrov pre výslednú hodnotu fitness::

1. (požadované / získané znalosti) = *pz*, typu float v intervale (0 - 1),
2. + (požadované / získané predpoklady) = *pp* typu float v intervale (0 -1),
3. + (((max. počet + celkový počet inštrukcií) – aktuálny počet) / (max. počet + celkový počet)) = *pi* typu float v intervale (0 - 1),
4. + ((aktuálny počet – duplicitné inštrukcie) / aktuálny počet) = *pd* typu float v intervale (0 - 1),
5. – (zakázané inštrukcie / aktuálny počet) = *pn* typu float v intervale (0 - 1) ,
6. + ((aktuálny počet – zlé inštrukcie) / aktuálny počet) = *pe* typu float v intervale (0 - 1).

Každej spočítanej hodnote sa priraďujú váhy, určujúce na ktorú z hodnôt kladieme dôraz a je pre nás podstatná. Z hľadiska protokolu som váhy na začiatok nastavil takto:

váhu *w_pz = 2 - znalosti sú to čo chceme, aby si subjekty vymenili,*

váhu *w_pe = 1 - zlé inštrukcie nemajú v protokole miesto, ale môžu pomôcť pri krížení,*

váhu *w_pp = 1 - predpoklady sú dôležité kvôli autentizácií,*

váhu *w_pn = 1 - zakázané inštrukcie by nemali subjekty dostať*

váhu *w_pi = 1 - hoci chceme mať čo najkratší protokol, neznamená že dlhší nie je lepší*

váhu *w_pd = 0,5 - duplicitné inštrukcie môžu pomôcť pri krížení*

S váhami by však malo byť možné experimentovať, tzn. mať možnosť prestaviť ich počiatočné hodnoty. Celková fitness podľa uvedeného vzorca a váh dáva aktuálne rozpätie hodnôt od 0 po 5,5.

Po odsimulovaní a ohodnotení protokolu, bude potrebné pre jednotlivé genómy zapísať štatistické údaje do súboru vo formáte `csv`, pre prehľad o tom kam nám spel celý GA, prípadne kde uviazol. Hodnoty, ktoré sa budú zapisovať by mali byť: číslo genómu, genóm v kódovanej podobe, jeho fitness a počet génov.

Kríženie genómov v populácií s „inteligenciou“:

Tvorí pľúca celého GA. Štandardný GA pracuje s genómom pevnej dĺžky a preto bolo nutné pozmeniť štandardnú funkciu, aby pracovala pre náš problém a rátala s genómami premenlivej dĺžky. S tým sa odvíja aj operátor kríženia, ktorý je síce možné vyberať náhodne, no bude sa musieť

kontrolovať maximálna dĺžka najkratšieho rodičovského genómu, aby sa nekrížilo v bode kde nie sú gény.

Operátor kríženia rozhádže pôvodne správne zoradené správy protokolu, tým sa zmenia množiny znalostí a predpokladov. Preto je vhodné priniesť inteligenciu, ktorá upraví potomka do správneho stavu a doplní prípadne inštrukcie z rodiča, ktoré nesú chýbajúcu informáciu. V literatúre venovanej genetickým algoritmom [4] sa názory na inteligenciu zanesenú do vnútra GA rôznia. V našom prípade je to však nutné z hľadiska ďalšieho vývoja protokolu.

Mutácia génu v genóme:

Operátor mutácie sa dá implementovať dvomi spôsobmi.

Prvý spôsob je založený na výmene génu v genóme náhodne vybraným génom z už existujúcej populácie. Aj zmena inštrukcií a poradia prinesie nový protokol, ktorý by inak krížením nemusel vzniknúť vo vymedzenom čase. Tu by mutácia fungovala ako akýsi urýchľovač procesu kríženia.

Druhý spôsob počíta s vytvorením množiny génov, z ktorej by sa vyberal náhodný gén pre zámenu. Týmto by sa do protokolu zaniesla úplne nová informácia, ktorá sa v pôvodnej populácií nemusela vyskytovať.

Krok v GA:

V každom kroku, okrem kontroly na ukončovacie kritérium, sa bude udržiavať štatistika vývoja populácie v priebehu GA. Výsledky pre každú populáciu sa zapíšu do súboru vo formáte `csv` a budú obsahovať nasledovné hodnoty: číslo generácie, minimálnu, priemernú a maximálnu fitness, čas existencie populácie a počet genómov v populácií.

Ukončenie GA:

Algoritmus je u konca v prípade, že sa našlo požadované riešenie, alebo dosiahol maximálny počet generácií. Táto skutočnosť sa dá kontrolovať po každom kroku GA volaním a porovnávaním hodnôt zo štatistiky GAlibu. Výsledný genóm sa dá ešte zoptimalizovať, formou kontroly na prebytočné inštrukcie pomocou PAL a získať tak jeho minimálnu verziu.

Výsledky sa zapíšu do súboru vo formáte `txt`, a mali by obsahovať:
pôvodnú populáciu protokolov a parametrov, z ktorých sa GA spúšťal, výsledný genóm, jeho mapovanú verziu (fenotyp), čas trvania GA a jeho optimalizovanú verziu.

4.2 Implementácia

4.2.1 Voľba programovacieho jazyka a platformy

Programovací jazyk som zvolil C/C++, aby bolo možné vyžiť už existujúce knižnice pre prácu s protokolmi a GA. Projekt by mal byť prenositeľný na operačné systémy Unix/Linux a Windows. V operačnom systéme Windows som využil kompilátor pod balíkom MicrosoftVisual Studio 2003, v operačnom systéme Linux kompilátor z balíka GCC.

4.2.2 Použité knižnice

Najprv je nutné definovať knižnice ktoré sa budú používať v hlavičkovom súbore `PEL.h`. Využívam dve a to PAL a GAlib pričom z GAlibu je potrebné definovať aj typ genómu, s ktorým sa bude pracovať. Pre náš problém som zvolil čo najjednoduchšiu reprezentáciu genómu pomocou jednorozmerného poľa typu `int`.

```
//evolučná knižnica
#include <ga/GASimpleGA.h>      // typ použitého genetického algoritmu
#include <ga/GA1DArrayGenome.h> // typ použitého genómu

//aplikačná knižnica
#include "protocol.h" // pre prácu s protokolmi
#include "cgen.h"     // pre generovanie pravidiel protokolu
```

4.2.3 Inicializácia

Aby bolo možné kombinovane využívať funkcie GAlibu a PAL knižnice v spojení s PEL funkciami, nadefinoval som globálne objekt generátor `cgen generator` a pole ukazateľov na typ protokol `cprotocol** Population`, ktoré predstavuje počiatočnú populáciu vygenerovaných protokolov (fenotypy).

O inicializáciu generátoru z PAL sa stará funkcia `GeneratorInit(generator)`, ktorá mu predá pravidlá gramatiky, podľa ktorej vygeneruje jednotlivé pravidlá protokolu.

Po inicializácií generátoru nasleduje vytvorenie počiatočnej populácie pomocou funkcie `Population PopulationInit(cgen &generator,int nProt,int nInstr)`, ktorá volá funciu `cprotocol* ProtocolInit(cprotocol *simprot,cgen &generator)`, pre zvolený počet protokolov a inštrukcií. Nastavenie každého protokolu vyžaduje definovanie znalostí, predpokladov a zakázaných inštrukcií pre každý subjekt. Po inicializovaní zapíše, pre štatistiku, vytvorenú populáciu do textového súboru `vysl.txt`.

Globálne konštanty, definované v `PEL.h` potrebné k správnemu fungovaniu GA sú nasledovné:

```
int popsize = 10;             //počet protokolov v populácií
```

```
int glength = 20;              //počiatočný počet inštrukcií v protokole
const int maxinstr = 100;      //maximálny počet inštrukcií
int ngen = 100;                //maximálny počet generácií
float pmut = 0.01;             //pravdepodobnosť mutácie
float pcross = 0.7;            //pravdepodobnosť kríženia
int w_knowl = 2;               //váha znalostí
float w_expect = 1;            //váha predpokladov
float w_instr_count = 1;       //váha počtu protokolov
float w_instr_dupl = 0.5;      //váha duplicitných inštrukcií
float w_instr_denied = 1;      //váha zakázaných inštrukcií ktoré subjekty
nemajú dostať
float w_badmut = 1;            //váha neplatných inštrukcií protokolu
```

Všetky parametre sa dajú zmeniť z príkazovej riadky pri spúšťaní programu. Dôležité sú hlavne `popsize`, `glength`, a `ngen`.

Pre zvolené hodnoty sa potom inicializuje genóm pomocou funkcie `void initializer(GAGenome &g)`, kde sa vytvorí odpovedajúca kódovaná verzia protokolu podľa vzorca: `gén(i) = ( ( id * maxinstr ) + i ) )` , kde `id` je index protokolu v rámci populácie a `i` je index génu.

4.2.4 Proces GA

Po inicializácií resp. ešte v rámci nej sa volá modifikovaná funkcia v PEL pre výpočet fitness celej populácie genómov.

Funkcia fitness

```
float fitness(GAGenome &g);
```

pracuje s aktuálnym genómom a využíva funkcií z PEL a PAL. Vytvorí si vlastný pomocný protokol, do ktorého namapuje genóm a odsimuluje. Zo získaných hodnôt potom spočíta a vráti fitness genómu. Medzi najdôležitejšie funkcie patria:

- mapovanie genómu na pomocný protokol:
  ```
  String MapRule(genome,population, i);//vráti fenotyp z genotypu
  ```
- simuláciu pomocného protokolu:
  ```
  bool TestInstr(pom,knowl);          //test znalosti na inštr.
  int insert_instr(pom);              //vloženie inštr. do prot.
  run();                              //simulácia behu protokolu
  ```
- po simulácií získanie hodnôt pre celkovú fitness
  ```
  float TestExpect();                 //predpoklady
  float TestKnowledge();              //znalosti
  float TestDeniedExpect();           //zakázané inštrukcie
  int DuplInstrCounter(protocolsim);  //duplicitné inštr.
  ```
- pre zápis do štatistiky
  ```
  string GenomeToString(genome);      //gény v string podobe
  ```

Pri výpočte fitness hodnoty dochádza k pretypovaniu z `int` na `float` a z `float` na `double`. Na jej konci zapíše výsledky zo simulácie protokolu pre daný gén do súboru `partout.csv`.

Funkcie „inteligentného“ kríženia

```
int crossover
(const GAGenome &p1, const GAGenome &p2,GAGenome *o1, GAGenome *o2);
```

v PEL je táto štandardná funkcia upravená pre prácu s rôznymi dĺžkami genómov, pomocou vektorov. Kríženie prebieha najprv „nanečisto“, s využitím dočasných potomkov, ktorí sa po úpravách priradia riadnym potomkom.

Pomocou funkcie GAlibu `GARandomInt(1, minlength)` sa vyberie bod kríženia z intervalu danom od 1 po maximálnu dĺžku genómu kratšieho z rodičov a zamenia sa gény v dočasných potomkoch.

Takto vygenerovaných potomkov ešte treba upraviť pomocou funkcie

```
vector<int> crossIntel
(int gsize, vector<int> &v, cgen &generator, Population);
```

pretože, sa môže stať, že z funkčných rodičov vzniknú po krížení nefunkční potomkovia, ktorých zmeneným inštrukciám chýbajú znalosti na to aby boli vykonané v rámci protokolu. Chýbajúce informácie obsahujú pôvodní rodičia. Funkcia sa preto snaží získať potrebné inštrukcie, ktoré nesú žiadané informácie, od rodičov pred bodom kríženia a doplniť ich. Využíva mapovanie z genotypu na fenotyp na zistenie funkčnosti potomka. Simuluje získaný fenotyp pomocou PAL funkcií. Ak test na inštrukciu počas simulácie neprejde, znamená to, že inštrukcia nemá dostatok informácií na to, aby sa vykonala. V takom prípade sa volá funkcia objektu generátoru `set<int> InsertInstructions (*protocolchild, *population[divres.quot-1], knowl, p);`
ktorá vráti množinu indexov potrebných inštrukcií z rodiča. Pravidlá z množiny sa vložia do protokolu a odsimulujú. Po úspešnej simulácií sa spätným mapovaním prevedú inštrukcie na genotypy a vložia do genómu pred inštrukciu, ktorá nemala dostatočné informácie. Získa sa tým opäť funkčný potomok.

Takto opravených dočasných potomkov priradí reálnym a tých vráti do procesu GA. Pravdepodobnosť kríženia sa dá nastaviť, v mojej implementácií je táto hodnota nastavená na 70%.

Funkcia mutácie:

```
int mutator(GAGenome &g, float pmut);
```

je implementovaná vo verzii s náhodným výberom genómu a génu z už existujúcej populácie pomocou funkcie `int GARandomInt()`. V cykle prechádza celým aktuálnym genómom a aplikuje mutáciu na jednotlivé gény. Využíva pri tom funkciu `GABoolean GAFlipCoin(pmut)`, ktorá s určitou pravdepodobnosťou zvolí zámenu génu. Pravdepodobnosť mutácie je nastavená na 1%, aby sa nevyskytovala príliš často a nedeformovala protokol.

4.2.5 Podporné funkcie

Mapovanie a hodnotenie protokolu:

```
void MapBestIndividual
(const GAGenome &g, cgen &generator, Population);
```

Zabezpečuje mapovanie genómu na výsledný protokol, odsimuluje ho a vypíše inštrukcie spolu s hodnotami tvoriacimi fitness vrátane celkovej fitness do súboru `vysl.txt`. Využíva simulačné funkcie z PAL, `insert_instr a run`, mapovacie a testovacie funkcie z PEL `MapRule` a `DuplInstrCounter`.

Mapovanie pravidla:

```
string MapRule
(const GAGenome &g, Population, int gene);
```

Stará sa o prevedenie génu na inštrukciu z populácie. Pomocou celočíselného delenia získa číslo protokolu a zvyšok po delení určí index hľadanej inštrukcie. Parameter `gene` udáva index pravidla, ktoré sa bude mapovať.

Počet duplicitných inštrukcií v protokole

```
int DuplInstrCounter(cprotocol *simprot);
```

Zráta počet rovnakých inštrukcií pre simulovaný protokol, porovnaním reťazcov.

Konverzia genómu do textovej podoby

```
string GenomeToString(const GAGenome &g);
```

Prevedie genóm a jeho hodnoty génov do vhodnej reťazcovej podoby pre výpis.

Optimalizácia protokolu

```
void BestProtOptim
(const GAGenome &g, cgen &generator, Population);
```

Optimalizácia protokolu využíva funkcie pre simuláciu a výpočet fitness. Základom je funkcia `cprotocol* Optimalization(*cprotocol)`, ktorá vráti upravený vstupný simulovaný protokol. Opätovne ohodnotí vrátený optimalizovaný protokol a zapíše aj s novými hodnotami fitness do súboru `vysl.txt`.

Odstránenie rovnakých inštrukcií

```
cprotocol* SameInstrRemover(cprotocol *simprot, cgen &generator);
```

Funguje na rovnakom princípe ako funkcia vracajúca počet duplicitných sa inštrukcií, s tým rozdielom že vráti upravený protokol.

4.3 Testy a získané výsledky

Knižnicu som testoval pre rôzne konfigurácie nastavení parametrov GA a protokolu.

Na operačných systémoch Unix/Linux i Windows. Z množstva získaných výsledkov, ktoré sú súčasťou priloženého CD, prezentujem podstatnú časť, kde je možné vidieť prínos a časovú náročnosť GA pri návrhu komunikačných protokolov.

Špecifikácia technického vybavenia použitého k testovaniu:

OS Windows

IBM ThinkPad T42

Procesor: Pentium M 735, 1.7 GHz, 2MB L2-Cache

Pamäť: 512 MB DDR, 333MHz

OS: Windows XP Professional SP2

OS Linux

IBM ThinkPad T42

Procesor: Pentium M 735, 1.7 GHz, 2MB L2-Cache

Pamäť: 512 MB DDR, 333MHz

OS: Mandriva Linux 2007

a školský server *merlin* (merlin.fit.vutbr.cz)

Procesor(y): 2xOpteron 2216

Pamäť: 4GB RAM

OS: Red Hat v.9

4.3.1 Fitness protokolu v závislosti na počte generácií

Ide o porovnanie vývoja maximálnej, priemernej a minimálnej fitness počas priebehu GA, kde každá hodnota vyjadruje stav genómov v populácií počas jednotlivých generácií. Graf č.1 reprezentuje simuláciu na OS Linux server *merlin*, pre nasledovné nastavenie počiatočných parametrov:

```
popsize = 100 - počet protokolov v populácií
glength = 20 - počiatočný počet inštrukcii v protokole
maxinstr = 100 - maximálny počet inštrukcií v protokole
ngen = 100 - maximálny počet generácií
pmut = 0.01 - pravdepodobnosť mutácie
pcross = 0.7 - pravdepodobnosť kríženia
w_knowl = 2 - váha znalostí
w_expect = 1 - váha predpokladov
w_instr_count = 1 - váha počtu protokolov
w_instr_dupl = 0.5 - váha duplicitných inštrukcií v protokole
w_instr_denied = 1 - váha zakázaných inštrukcií v protokole
```

```
w_badmut = 1 - váha neplatných inštrukcií v protokole
```

Maximálna možná hodnota fitness, ktorá reprezentuje splnenie všetkých požiadaviek na protokol, sčítaním všetkých kladných váh dá číslo 5,5. Použitá gramatika, nastavenie počiatočných množín znalostí, predpokladov, zakázaných inštrukcií a komunikačných kanálov sú nasledovné:

Gramatika pre 3 komunikujúce subjekty:

Pravidlá:

```
generator.AddRule("<start>=<sub>-><sub>:<message>");
generator.AddRule("<gennonce>=, <nonce>");
generator.AddRule("<gennonce>= ");
generator.AddRule("<gennonce>=, <message>");
generator.AddRule("<message>=<gensub>");
generator.AddRule("<message>=<krypt><gennonce>");
generator.AddRule("<message>=<kmessage><gennonce>");
generator.AddRule("<kmessage>=<krypt><gennonce>");
generator.AddRule("<kmessage>=<gensub><gennonce>");
generator.AddRule("<kmessage>=<genkey><gennonce>");
generator.AddRule("<krypt>={<kmessage>}<key>");
```

Komunikačné kanály:

```
generator.AddChannel("A", "S", unsecure);
generator.AddChannel("A", "B", unsecure);
generator.AddChannel("B", "S", unsecure);
generator.AddChannel("B", "A", unsecure);
generator.AddChannel("S", "S", unsecure);
generator.AddChannel("S", "B", unsecure);
```

Znalosti subjektu A: `A, B, S, KAS, NA;`

Znalosti subjektu B: `A, B, S, KBS, NB;`

Znalosti subjektu S: `A, B, S, KAB, KAS, KBS, NS;`

Predpoklady subjektu A: `B: KAS, NA; S: NA ;`

Predpoklady subjektu B: `A: KBS, NB; S: NB;`

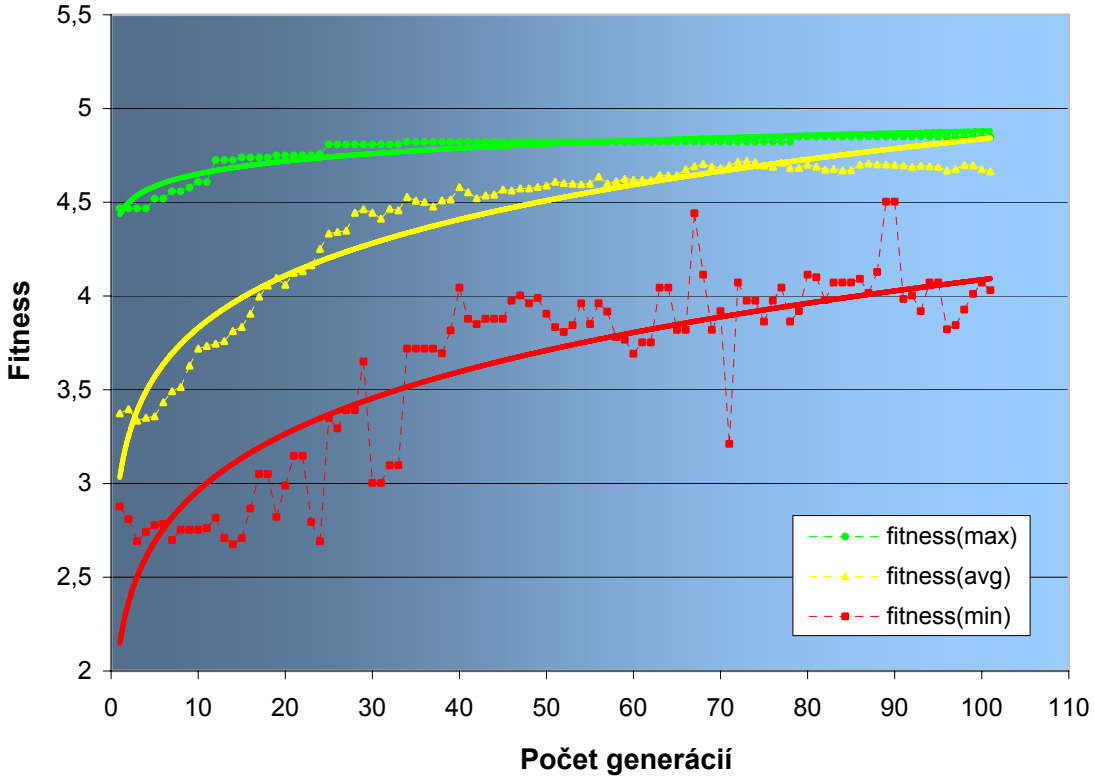
Predpoklady subjektu S: `A: KAB; B: KAB;`

Ciele subjektov: `A: KAB; B: KAB;`

Zakázané inštrukcie subjektu A: `B: KAS;`

Zakázané inštrukcie subjektu B: `A: KBS;`

Na základe počiatočných nastavení sa vygenerovala počiatočná populácia 100 protokolov o 20. inštrukciách a prebehol GA pre 100 generácií s nasledovnými výsledkami.



Graf č.1: Závislosť hodnoty fitness od počtu generácií

Z grafu je možné vyčítať, že pre zvolené nastavenia sa maximálna hodnota fitness od 40. generácie zmení len nepatrne a od 80 generácie sa už nemení vôbec. Priemerná hodnota sa približuje k maximálnej, to znamená, že postupne sa vytvorí populácia protokolov s fitness blížiacej sa max. hodnote. Minimálna fitness odzrkadľuje vývoj minimálnej hodnoty genómu v populácií.

Výsledný najlepší protokol pred a po optimalizácií vyzerá nasledovne:

```
Resulting genotype:
3100 3916 8202 1203 3104 3105 3106 1007 1008 1009 1010 319 6212 3313 6313
3715 2516 6605

Resulting fenotype (protocol):
    1. A->B:S
    2. S->A:A
    3. B->A:B, NB, NB
    4. A->B:{S}KAS, NA, NA
    5. A->S:S, NA
    6. S->B:A
    7. B->S:S, NB, NB
    8. B->S:S
    9. B->S:A, NB, A
```

10. S->A:{KAB}KAS
11. B->A:S
12. S->B:{KAB}KBS
13. A->S:S
14. B->A:{B}KBS
15. A->S:{S, NA}KAS
16. B->A:A
17. S->B:{A}KBS
18. B->S:A

Total time for generating best solution: 31 seconds
Fitness of the best: 4.85
Percentual fitness: 88.1818%
Knowledge: 1
Expectations: 0.5
Double instr.: 0
Denied instr.: 0
Bad instr.: 0

Optimized version:
1. B->A:B, NB, NB
2. A->B:{S}KAS, NA, NA
3. A->S:S, NA
4. B->S:S, NB, NB
5. B->S:A, NB, A
6. S->A:{KAB}KAS
7. S->B:{KAB}KBS
8. A->S:{S, NA}KAS

Fitness after optimization: 4.93333
Percentual fitness: 89.697%
Knowledge: 1
Expectations: 0.5
Double instr.: 0
Denied instr.: 0
Bad instr.: 0

4.3.2 Závislosti na veľkosti populácie, počtu inštrukcií, generácií, dobe trvania a výslednej fitness protokolu

Skúmané hodnoty dávajú ucelený obraz o úspešnosti a náročnosti výpočtu pomocou GA aj v závislosti na OS. Nastavenie parametrov je okrem uvedených zmien rovnaké ako v predchádzajúcom príklade.

Platforma OS Linux server merlin (3 subjekty)				
Počet inštrukcií	**Počet protokolov**	**Počet generácií**	**Maximálna fitness**	**Doba generovania (v sekundách)**
15	10	100	3,0529	2
15	20	100	4,1855	3
15	50	100	4,1195	8
15	100	100	4,8695	22
15	200	100	4,2029	43
15	300	100	4,8695	66
20	10	100	4,6833	2
20	20	100	4,825	7
20	50	100	4,8	15
20	100	100	4,85	31
20	200	100	4,8333	64
20	300	100	4,8333	91
25	10	100	4,5132	3
25	20	100	4,7383	9
25	50	100	4,7684	20
25	100	100	4,7679	42
25	200	100	4,848	96
25	300	100	4,832	134
30	10	100	4,5141	3
30	20	100	4,7897	9
30	50	100	4,7675	25
30	100	100	4,7412	53
30	200	100	4,8128	95
30	300	100	4,823	144

Tabuľka č.1 Maximálna fines a časová náročnosť v závislosti rôznych parametrov GA a protokolu pre OS Linux a 3 komunikujúce subjekty

Z empiricky zistených hodnôt sa dá odvodiť, že najlepšie hodnoty fitness sú pre 20 inštrukcií na protokol a stačí populácia o max. 200 protokoloch. Potom už stúpa časová náročnosť a hodnoty fitness sú buď horšie alebo sa zlepšujú len nepatrne. Hodnoty sú tiež závislé od generátorom náhodne vytvorenej populácie. V prípade, že sa podarí vygenerovať dobrú populáciu už na začiatku, stačí aj menšia populácia s menším počtom inštrukcií a menším počtom generácií.

V porovnaní so simuláciou v OS Windows, pracuje GA výrazne pomalšie ako v OS Linux čo ukazuje tabuľka porovnania pre vybrané hodnoty.

Časové porovnanie vývoja protokolu v závislosti na OS				
Počet inštrukcií	**Počet protokolov**	**Počet generácií**	**Maximálna fitness**	**Doba generovania (v sekundách)**
Platforma OS Windows IBM ThinkPad T42				
20	100	100	4,8333	1190
Platforma OS Linux IBM ThinkPad T42				
20	100	100	4,7833	43

Tabuľka č.2 Časová náročnosť vývoja protokolu v závislosti na OS a použitého kompilátoru

Príčinou pre taký veľký časový rozdiel, pri rovnakom nastavení parametrov je pravdepodobne v knižnici GAlib, ktorej primárna podpora je pre OS Unix/Linux. Na OS Windows je pre použitý kompilátor pod Microsoft Visual Studio 2003 nutné nastavovať množstvo parametrov, ktoré zrejme v konečnom výsledku spomaľujú celý program.

Simulácia pre viaceré subjekty zvyšuje náročnosť výpočtu, pretože narastá počet inštrukcií protokolu a tým aj časová a pamäťová náročnosť.

5 Záver

Vytvorená knižnica funkcií v spolupráci s už existujúcimi knižnicami umožňuje skúmanie prínosu evolučných algoritmov pri návrhu komunikačných protokolov.

Ide o časovo a na technické prostriedky náročný proces, ktorý nie vždy skončí úspešne. Môže uviaznuť v lokálnom maxime, alebo pri náhodne nevhodne vygenerovanej počiatočnej populácií zdegradovať a skončiť na protokole o jednej inštrukcií. Toto sú však ojedinelé prípady, s ktorými som sa počas mojej práce s GA stretol.

Z výsledkov testovania na jednoduchých autentizačných symetrických protokoloch je zrejmé, že je možné pomocou GA získať protokoly, ktoré spĺňajú zvolené požiadavky zhruba na 90% pri použitom pevnom nastavení váh. Oproti prvotne vytvorenej populácií sa maximálna hodnota fitness zvýši v priemere o 7%. Na konci však vznikne množina protokolov, ktoré oproti prvotnej populácií majú podstatne vyššiu hodnotu fitness čo dokumentuje priemerná hodnota, ktorá sa zvýši v priemere o 23%. Z tejto populácie je potom možné vyberať nielen najlepšieho jedinca, ale viacerých s rovnakou hodnotou fitness a skúsiť použiť na nich niektorú z optimalizačných techník. Prínos GA je teda v dvoch rovinách, jednak umožňuje získať najlepšie riešenie, no zároveň vytvára aj skupinu dostatočne dobrých riešení, s ktorou sa dá ďalej pracovať.

Možností ďalšieho rozšírenia a skúmania evolúcie na protokoloch, je nespočetne veľa, v závislosti od nastavení rôznych parametrov, až po testovanie rôznych typov genetických algoritmov (prekrývajúce sa populácie, paralelné sa vyvíjajúce populácie, rôzne možnosti krížení …). Je možné využiť služieb iných knižníc pre podporu GA ako je GAlib a testovať rôzne skupiny protokolov.

Literatúra

[1] Očenášek Pavel, Švéda Miroslav: *An Approach to Automated Design of Security Protocol,* In: Proceedings of the International Conference on Networking (ICN 2006), Los Alamitos, US, IEEE CS, 2006, p. 4, ISBN 0-7695-2522-0

[2] Očenášek Pavel: *Verifikace bezpečnostních protokolů,* Brno, CZ, FIT VUT, 2003, p. 54

[3] Koza J. R.: *Genetic Programming: On Programming Computers by Means of Natural Selection and Genetics*, MIT Press, 1992

[4] GAlib A C++ Library of Genetic Algorithm Components, URL: http://lancet.mit.edu/ga/, (september 2006)

[5] Open BEAGLE Evolutionary Computation Framework, URL: http://beagle.gel.ulaval.ca/index.html, (január 2007)

[6] EO Evolutionary Computational Framework, URL: http://eodev.sourceforge.net/, (január 2007)

[7] Ryan C., Collins J.J., O'Neill M.: *Grammatical Evolution: Evolving Programs for an Arbitrary Language,* Lecture Notes in Computer Science 1391, First European Workshop on Genetic Programming, 1998

[8] Gramatical Evolution Tutorial, URL: http://www.grammaticalevolution.org/tutorial.pdf, (október 2006)

[9] O'Neill M., Ryan C.: *Grammatical Evolution*. IEEE Transactions on Evolutionary Computation, 5(4), pp 349-358, 2001

[10] Song Dawn, Perrig Adrian, Phan Doantam: *AGVI - Automatic Generation, Verification, and Implementation of Security Protocols*, 13th Conference on Computer Aided Verification CAV, 2001

[11] Petr Hadaš: *Bezpečnostní protokoly – teorie a praxe: Bakalářská práce*, Brno, CZ, FIT VUT, p. 34, 2006

[12] Anupam Datta, Ante Derek, John C. Mitchell, Dusko Pavlovic: *A Derivation System for Security Protocols and its Logical Formalization*, Computer Science Dept., Stanford University, Stanford, CA, 2003

[13] Burrows M., Abadi M., Needham R.: *A Logic of Authentication*, ACM Transactions on Computer Systems, 8(1), p. 18-36, 1990

[14] Levente Buttyán, Sebastian Staamann, Uwe Wilhelm: *A Simple Logic for Authentication protocol Design*, Swiss Federal Institute of Technology, Institute for computer Communications and Applications, EPFL-DI-ICA, Lausanne, Switzerland 1998

[15] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989

[16] Holland J.: *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[17] Kirkpatrick S., Gelatt C. D., Vecchi M. P.: *Optimization by Simulated Annealing*, Science, Vol 220, Number 4598, p. 671-680, 1983

[18] John A Clark: *Protocols are Programs Too: Using GAs to Evolve Secure Protocols*, Presentation to SEMINAL Workshop in Bath (2.3.2000), URL: http://www-users.cs.york.ac.uk/%7Ejac/PublishedPapers/Presentations/SEMINAL.ppt, (apríl 2007)

[19] Chen Hao, Clark John and Jacob Jeremy: *Automatic Design of Security Protocols*, Computational Intelligence, Volume 20, Number 3, pp 503 – 516, Special Issue on Evolutionaty Computing in Cryptography and Security, August 2004

Zoznam príloh

Príloha 1. CD