VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KRYPTOGRAFICKÉ PROTOKOLY PRO CRYPTOOL

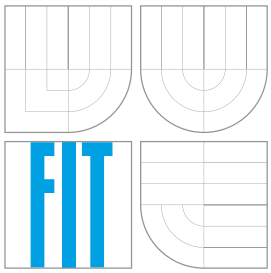
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ SKOWRONEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KRYPTOGRAFICKÉ PROTOKOLY PRO CRYPTOOL

CRYPTOGRAPHICS PROTOCOLS FOR CRYPTOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ SKOWRONEK
AUTHOR

VEDOUCÍ PRÁCE — MARTIN HENZL, Ing. Mgr.
SUPERVISOR

BRNO 2012

Abstrakt

Práce je zaměrena na kryptografické protokoly a jejich implementaci do programu CrypTool. Cílem práce je určit, které kryptografické protokoly by bylo vhodné do programu CrypTool implementovat a následně jejich implementaci provést takovým způsobem, aby jejich demonstrace byla co nejnázornější a nejvhodnější pro pochopení jejich konceptu. V souvislosti se zadáním této práce byly vybrány protokoly Dining Cryptographers, Coin Flipping a Zero-Knowledge. Jako další vhodné protokoly byly přidány Oblivious Transfer a Yao´s Millionaire Problem. V práci je podrobně popsán koncept vybraných protokolů a postup při jejich implementaci.

Abstract

This work is focused on cryptographic protocols and their implementation in the CrypTool. Goal of this work is to settle which cryptographics protocols are appropriate to implement and to find the way how to implement them most illustratively and usably for educational purposes. These protocols were chosen according to the work assigment: Dining Cryptograhpers, Coin Flippin and Zero-Knowledge. As other appropriate protocols were added: Yao´s Millionaire Problem and Oblivious Transfer. Detailed concept of chosen protocols and their implementation process are described in this work.

Klíčová slova

CrypTool, Dining Cryptographers, Zero Knowledge, Flip Coin, Yao´s millionaire problem, Oblivious Transfer, kryptografický protokol

Keywords

CrypTool, Dining Cryptographers, Zero Knowledge, Flip Coin, Yao´s millionaire problem, Oblivious Transfer, cryptographic protocol

Citace

Ondřej Skowronek: Kryptografické protokoly pro CrypTool, bakalářská práce, Brno, FIT VUT v Brně, 2012

Kryptografické protokoly pro CrypTool

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Martina Henzla

.....
Ondřej Skowronek
15. května 2012

Obsah

Kapitola 1

Úvod

V dnešní době informačních technologií má téměř každý člověk zkušenosti s elektronickou komunikací. Je běžnou praxí používat internet za účelem vzdělávání nebo i jen k zábavě. Používání internetu s sebou ovšem nese i určitá rizika. Když chceme někomu poslat nějakou informaci, ne vždy využíváme přímé spojení, ke kterému by nikdo jiný neměl přístup.

V mnoha případech se naše zpráva předává přes bezpočet serverů, než dojde ke chtěnému příjemci. A tohle je příčina, proč může být používání internetu nebezpečné. Nikdy přesně nevíme, komu se námi poslané zprávy dostanou. Citlivé informace v našich zprávách mohou být zneužity.

Kvůli tomuto riziku je proto nutné komunikaci zabezpečit, aby se nikdo nechtěný nemohl k posílaným informacím dostat a zneužít je. Tím, jak zabezpečit, a znemožnit tak neoprávněným osobám přístup k našim informacím. se zabývá věda zvaná kryptografie. Kryptografie se převážně zabývá kryptografickými systémy, které se používají pro šifrování a dešifrování zpráv. [16]

Tato věda je neustále ve vývoji, protože žádný kryptografický systém není dokonalý, lidé stále nacházejí nové způsoby, jak zabezpečení prolomit, a proto je nutno vyvíjet nové a dokonalejší kryptografické systémy. Spousta lidí nemá ponětí o tom, jak ani ty nejzákladnější kryptografické systémy fungují.

Neméně důležitou součástí kryptografie jsou kryptografické protokoly. S těmito protokoly se setkáváme v každodenním životě, přesto jejich běžní uživatelé nejsou seznámeni s tím, jak tyto protokoly fungují. Cílem této práce je změnit tento fakt a seznámit širokou veřejnost s těmito protokoly. [11] Pro tento nelehký úkol bude využit program CrypTool.

Program CrypTool je interaktivní výukový program, který se zabývá kryptografií. Tento program má podobný cíl jako tato bakalářská práce. Snaží se své uživatele seznámit s kryptografií. Toto seznámení probíhá takovou formou, kdy si uživatel může nechat demonstrovat nejrůznější situace, kterými se kryptografie zabývá. [6]

Tento program je v neustálém vývoji a zatím zde byly implementovány převážně kryptografické systémy. Kryptografické protokoly zde mají slabé zastoupení, tudíž uživatelé tohoto programu se toho o kryptografických protokolech moc nedozví. Z tohoto důvodu zde budou, v rámci této bakalářské práce, implementovány nové kryptografické protokoly.

Nejprve bude třeba určit, které protokoly jsou už zde implementovány a které protokoly by bylo vhodné zde implementovat. Při implementaci těchto protokolů půjde převážně o to, aby byly co nejsrozumitelnější uživatelům CrypToolu. Pro dané protokoly bude třeba vytvořit vhodnou dokumentaci, která pomůže uživatelům se v těchto protokolech zorientovat.

Důležitým cílem této práce bude spojit se s vývojáři programu CrypTool a nabídnout jim zde vytvořené kryptografické protokoly. Pokud se vývojářům CrypToolu budou tyto

vytvořené kryptografické protokoly líbit, stanou se tak součástí další stabilní verze CrypToolu. Všichni uživatelé, kteří si stáhnout novou verzi programu CrypTool, si tak budou moci tyto kryptografické protokoly vyzkoušet a snadno je pochopit.

Zvyšování povědomí o kryptografickým protokolech je užitečné pro celou společnost, protože čím více bude zájemců, zabývajících se kryptografií a jejími protokoly, tím bude rychlejší jejich vývoj a elektronická bezpečnost poroste.

Druhá kapitola se bude zabývat převážně programem CrypTool. Čtenáři s ním budou podrobně seznámeni, dozvědí se, jak se s tímto programem pracuje a jak probíhá jeho vývoj. Budou zde podrobně popsány základní souvislosti, které je potřeba znát při vysvětlování implementace následujících protokolů.

Ve třetí kapitole se budu věnovat kryptografickým protokolům a seznámím čtenáře s tím, co to vůbec kryptografický protokol je. Také se v této kapitole budu věnovat analýze programu CrypTool, abych zjistil, které protokoly zde chybí a které by bylo užitečné implementovat. Budou zde zmíněny vybrané protokoly a důvody, proč je vhodné tyto protokoly do programu CrypTool naimplementovat.

V dalších kapitolách se budu věnovat už jednotlivým kryptografickým protokolům, kde vysvětlím koncept těchto protokolů, popíši návrh implementace protokolu do programu CrypTool a jak jsem protokol implementoval. Budou zde popsány postupy, které budou aplikovány, aby implementace těchto protokolů byla účinně demonstrovatelná a pochopitelná pro všechny.

Závěrem budou shrnuty všechny kapitoly a budou sumarizovány výsledky implementace jednotlivých protokolů. Budou zde zhodnoceny cíle této práce a bude popsána snaha dostat tyto protokoly k tvůrcům CrypToolu.

Kapitola 2

CrypTool

Jak už bylo naznačeno v úvodu, CrypTool je výukový program, snažící se širokou veřejnost seznámit se základními, ale i pokročilými kryptografickými systémy. CrypTool je opensource program, což znamená, že jeho šíření je dovoleno, a je dokonce podporováno samotnými tvůrci. CrypTool je vyvíjen od roku 1998 a mezitím se na tomto vývoji podílely stovky programátorů. Existuje současně několik používaných verzí CrypToolu CrypTool 1.4.x, JCrypTool, CrypTool online a CrypTool 2. CrypTool 1.4.x je základní verze CrypToolu, napsaná v jazyce *C++* pro operační systém Windows, byla vyvíjena od začátku projektu a nyní je v porovnání s CrypTool 2 poněkud zastaralá. JCrypTool je speciální verze CrypToolu, která je napsána v jazyce Java, takže základním rozdílem této verze od ostatních je to, že umožňuje multiplatformnost. CrypTool online je verze, která je provozována přes webový server, a tak je přístupnější všem uživatelům, kteří by si chtěli CrypTool vyzkoušet. Tato bakalářská práce se ovšem bude zabývat verzí CrypTool 2. CrypTool 2 má, na rozdíl od CrypTool 1, velmi interaktivní prostředí a na něj je soustředěna většina pozornosti vývojářů CrypToolu.

CrypTool 2 je napsán v jazyce *C#*, proto je bohužel spustitelný pouze v operačním systému Windows, kde jsou nainstalovány knihovny Nett. Pokud se budu v následujícím textu zmiňovat o CrypToolu, bude tím myšlena verze CrypTool 2, nebude-li řečeno jinak. [17]

2.1 Vývoj CrypTool

Tento projekt je open-source, tudíž každý, kdo se chce podílet na vývoji, se může přidat. Musí se ale domluvit s tvůrci programu CrypTool. Tvůrci programu vítají zájem ostatních programátorů přidat do jejich programu nové komponenty, ale nemohou přidávat úplně vše co kdo naprogramuje. Než vypustí stabilní verzi, musí vše náležitě otestovat a ověřit její správnost. Aby byla komunikace mezi vývojáři co nejpraktičtější a mohli se takto zároveň podílet na vývoji několika verzí zároveň, je používán systém sdílení verzí Apache Subversion. [19]

2.2 Práce s programem CrypTool

Práce s programem CrypTool je velmi intuitivní. Koncept tohoto programu je takový, že uživatel má v pracovním panelu širokou nabídku pluginů, které přetáhnutím může vložit do hlavního okna a využít tak jejich funkčnost. Plugin je základním kamenem pro práci

v programu CrypTool. Práce vývojářů, kteří chtějí přispět a rozšířit program CrypTool, tak většinou spočívá v tom, že vytvářejí nové pluginy.

2.3 Plugin

Každý plugin sice pracuje jinak, ale přesto mají určité stejné vlastnosti. Plugin si můžeme představit jako krabičku, která má určité vstupy a výstupy. Každý vstup a výstup má své jméno a datový typ. Počet vstupů a výstupů záleží na konkrétním pluginu. Uživatel si tedy vloží do hlavního okna pluginy, se kterými by rád pracoval a poté propojí jejich vstupy a výstupy mezi sebou.

Až si uživatel poskládá pluginy podle svého přání, může zmáčknout tlačítko "spustit"v horním panelu a podívat se, jaké hodnoty si pluginy mezi sebou posílají. Při zmáčknutí tlačítka "spustit"se aktivují ty pluginy, které nemají žádné vstupy, pošlou hodnoty ze svých výstupů pluginům, na které jsou napojeny, a aktivují je.

Pluginy mohou mít své nastavení, které ovlivňuje jejich chování a tak i jejich výstupy. Například, plugin pro matematické operace má v nastavení, jakou matematickou operaci má pro své vstupy vykonávat, proto může být výstupem součet jejich vstupů, rozdíl vstupů atd... Mimo nastavení mohou mít pluginy své tělo, do kterého se dají zapisovat data, které mají tyto pluginy posílat, popř. do těla pluginů se uloží vstup, který tyto pluginy přijímají. [14]

Při implementaci kryptografických protokolů se budu snažit využívat pluginů, které už jsou v programu CrypTool implementovány, a to z toho důvodu, abych nevytvářel nové zbytečné věci. Funkčnost těchto zabudovaných pluginů nebudu detailněji popisovat, proto, aby i čtenář, který nemá zkušenosti s tímto programem, mohl pochopit jak jednotlivé mnou implementované protokoly v CrypTool fungují, popíši zde pluginy, které budou zmiňovány v dalším textu. U těchto pluginů nebudu popisovat celou jejich funkčnost, ale pouze důležité části, které je nutno znát.

TextInput

Toto je základní pomocný plugin. Do tohoto pluginu je možné zapisovat text, který se při spuštění hlavního okna odešle ostatním pluginům. Plugin nemá žádný vstup, jen výstup *Text* datového typu string.

TextOutput

Plugin sloužící pro uživatele, aby ten zjistil, co se posílá z výstupu nějakého pluginu. Má jeden vstup *Input data* datového typu *Object*. Na těle tohoto pluginu se zobrazuje to, co mu bylo posláno na jeho vstup.

NumberInput

Plugin podobný pluginu *TextInput* s tím rozdílem, že do těla tohoto pluginu je možné zapisovat pouze číslice. Jeho výstupem *Number Output* se posílá jeho obsah, tento výstup je datového typu *BigInteger*.

BooleanInput

Plugin pro posílání logických hodnot. U tohoto pluginu se dá nastavit, jestli je jeho hodnota *false* nebo *true*. Tato hodnota je i graficky ztvárněna ikonkou zelené nebo červené žárovky. Tato hodnota se poté odesílá jeho jediným výstupem *Output* datového typu *Boolean*.

BooleanOutput

Tento plugin slouží pro znázornění přijatých hodnot datové typu *Boolean*, kdy se podle přijatých hodnot změní ikonka tohoto pluginu na zelenou nebo červenou žárovku. Tento plugin má pouze vstup *Input* datového typu *Boolean*.

PrimeGenerator

Plugin sloužící na generování náhodných prvočísel. Tento plugin nemá žádné vstupy a pouze jeden výstup *BigInteger Output* datového typu *BigInteger*. V pluginu lze nastavit nejvyšší prvočíslo, které lze vygenerovat či velikost generovaného prvočísla v počtech bitů.

NumberOperations

Plugin sloužící pro základní matematické operace. V nastavení jde nastavit, jestli má tento plugin sčítat, odčítat, násobit, dělit či vykonávat operaci modulo. Plugin má dva důležité vstupy *x Input* a *y Input*, datového typu *BigInteger*. Do těchto vstupů patří čísla, která budou použita v matematických operacích. Výsledek těchto operací se ukládá do vstupu *Output* datového typu *BigInteger*.

StringOperations

Obdobný plugin jako *NumberOperations*, akorát tento plugin slouží na provádění operací s řetězci. Důležité vstupy tohoto pluginu jsou *String One* a *String Two*. Tento plugin lze nastavit na nejrůznější řetězcové operace, já jsem však z těchto operací využil pouze konkatenaci řetězců. Výsledek operací se ukládá na výstup *Output String*.

IncDec

Plugin s jedním vstupem *Input* datového typu *Integer*. Tento plugin slouží na připočítávání nebo odčítání. V nastavení lze nastavit, zda se bude přičítat či odčítat, a o kolik. Upravené číslo se ukládá na výstup *Output*, který je datového typu *Integer*.

StreamComparator

Plugin pro porovnávání dvou vstupů datového typu *ICryptoolStream*. Vstupy se jmenují *Stream One* a *Stream Two*. Plugin zjistí, jestli vstupy jsou stejné a výsledek uloží jako logickou hodnotu na výstup *Comparator Achievement*.

2.4 Navrhování implementace

Při návrhu bylo třeba vymyslet, jak co nejpodobněji protokoly zabudovat do programu CrypTool. Protože při implementaci protokolů lze pouze vytvářet nové pluginy, byl jsem

vázán možnostmi, které pluginy skýtají. Upravovat program v jádře tohoto programu by bylo velmi nepraktické, protože tento kód je velmi obsáhlý.

Dalším faktem je to, že tvůrci CrypToolu neznámým vývojářům neumožňují upravovat jádro, neboť veškeré změny v jádře musí být pečlivě promyšleny, protože tyto změny ovlivní příliš mnoho lidí. Z tohoto důvodu bych tyto změny nemohl nabídnout oficiálním tvůrcům, což je jedním z cílů této bakalářské práce.

Navrhování protokolů je velmi složitá činnost, je vždy třeba vyřešit, který návrh je nejjasnější a pro uživatele programu CrypTool tedy nejpochopitelnější. Je zde třeba dát si pozor, aby pluginy nezapouzdrovaly příliš mnoho procesů.

Zapouzdřenost procesů je při vývoji aplikací chtěná věc, protože se snažíme, aby se uživatel pro něj nedůležitými informace mohl zaobíra co nejméně. Je optimální, pokud jsou všechny nezbytné věci provedeny aplikací. Zde je taková velká míra abstrakce neužitečná z tohoto důvodu, protože při implementaci pluginu není důležité, aby jeho použití bylo jednoduché, ale aby si při práci s tímto pluginem uživatel uvědomil, jak tento protokol vůbec funguje.

2.5 Šablona

V programu CrypTool je možné uložit obsah hlavního okna, který lze později nahrát zpátky. Pluginy pro kryptografické protokoly mají specifické užití pouze v samotném protokolu. Poskládání pluginů dohromady je pro uživatele bez znalosti daného protokolu velmi složité. Proto vytvořím pro každý protokol šablonu, aby si uživatel mohl nahrát do hlavního okna mnou složené pluginy, a tak viděl, jak by měl dotyčný protokol vypadat. Po nahrání šablony si uživatel může udělat změny, jaké chce, a pozorovat, jaké to má důsledky na funkčnost protokolu.

2.6 Dokumentace

Při přidávání nových pluginů do programu CrypTool je nestačí pouze správně naprogramovat. Uživatelům musí být jasné jak pracují, aniž by si četli zdrojový kód. Pro všechny pluginy je třeba vytvořit podrobnou dokumentaci. Dokumentace musela být napsána v angličtině.

Dokumentace se skládá z několika částí, první části je seznámení uživatele CrypToolu s pluginem, v případě implementace pluginů kryptografických protokolů patřilo do této části popis konceptu protokolu.

Další části je užití, zde je popsáno jak plugin funguje a jak s ním má uživatel zacházet. V této části se popisují vstupy a výstupy pluginy. Také je zde objasněno nastavení pluginů a k čemu slouží.

Každý správná dokumentace k pluginu také musí odkazy na stránky, z kterých autor vycházel při implementaci těchto pluginů. Odkazy v dokumentaci pluginů jsou shodné s odkazy v citacích.

2.7 Ikony

Každému pluginu lze nastavit jeho ikona. Tato ikona nijak neovlivní běh pluginů, slouží pouze k estetickým účelům. Vytváření vlastních ikon je zbytečné, protože lze využít ikon,

které jsou k dispozici zdarma. Pro nalezení těchto ikon jsem využil tuto webovou stránku http://www.iconfinder.com/free_icons

Kapitola 3

Kryptografické protokoly

3.1 Úvod do kryptografických protokolů

Kryptografické protokoly jsou způsoby komunikace mezi dvěma a více účastníky. Tyto protokoly se od ostatních komunikačních protokolů liší tím, že úkolem tohoto typu protokolu je zabránit vyzrazení nějaké soukromé informace. Proto se kryptografické protokoly zpravidla využívají mezi účastníky, kteří si navzájem nedůvěřují.

Kryptografické protokoly se dají použít nejen v komunikaci přes internet. Využití kryptografických protokolů je velmi široké, spadá sem vytváření certifikátů pravosti, anonymní odesílání zpráv, šifrované internetové komunikace a další. [10]

3.2 Analýza programu CrypTool

Abych mohl implementovat do programu CrypTool nějaký protokol, musel jsem nejdříve zanalyzovat, které protokoly už do něj byly implementovány. Program CrypTool se soustředí převážně na vytváření kryptografických systémů, proto je zde implementováno jen pár protokolů. Tyto protokoly jsou Diffle Herman Merkle key exchange, Smart Card protokol a Wep protokol. Mezi chybějícími protokoly, které ještě nebyly v CrypTool nikým implementovány, byly protokoly *Dining Cryptographers*, *Coin Flipping* a *Zero Knowledge*. Tyto protokoly jsou dle mého názoru pro implementaci vhodné.

Vybrat protokoly vhodné k implementaci nebylo jednoduché, bylo třeba si nastudovat charakteristiky jednotlivých protokolů a určit jestli je implementace v programu CrypTool pro tento protokol možná. Kryptografické protokoly mohou být značně abstraktní, takže vymyslet správnou formu implementace je nejednoduché. Proto jsem se snažil vybrat takové protokoly, které nejsou příliš abstraktní a pracuje se tam s reáliemi, které lze jednoduše prezentovat pluginy.

Další protokol, který jsem se rozhodl implementovat, je *Yao´s millionaire problem*. Tento protokol jsem si vybral z toho důvodu, že je zajímavý a je snadno demonstrovatelný v programu CrypTool. Navíc pro tento protokol se využívá RSA kryptografického systému, který je již v programu zabudován. Mohu tak demonstrovat využití stávajících pluginů pro vytváření nových věcí.

Posledním, pátým protokolem, který jsem vybral pro implementace, je protokol *Oblivious transfer 1-n*. Tento protokol jsem vybral kvůli tomu, že si samotní tvůrci programu CrypToolu přáli tento protokol do tohoto programu zabudovat. [2]

Souhrn implemenetovaných protokolu je takový:

- Dining Cryptographers
- Coin Flipping
- Zero Knowledge
- Yao's Millionaire Problem
- Oblivous transfer

Kapitola 4

Dining Cryptographers

4.1 Úvod

Kryptografové jdou do restaurace na večeři. Sednou si ke kulatému stolu, objednají si jídlo a povečeří. Po obědě k ním přistoupí číšník a oznámí jim, že oběd byl již za všechny zaplacen. Kryptografové respektují právo ostatních na anonymitu, ale jsou zvědaví, jestli za oběd zaplatil jeden z nich a nebo za oběd zaplatila asociace kryptografů. Jak to zjistit řeší následující protokol. Tento protokol se používá pro anonymní komunikaci mezi více jak dvěma účastníky. [4]

4.2 Koncept

Tento protokol probíhá ve dvou krocích.

V prvním kole si každý kryptograf hodí tajně mincí a zapamatuje si výsledek. Hozené panně přiřadí logickou hodnotu *true* a hozenému orlovi logickou hodnotu *false*. Tuto hodnotu poté pošeptá kolegovi kryptografovi po jeho levé straně. Tyto dvě hodnoty si zapamatuje, protože budou použity v následujícím kole.

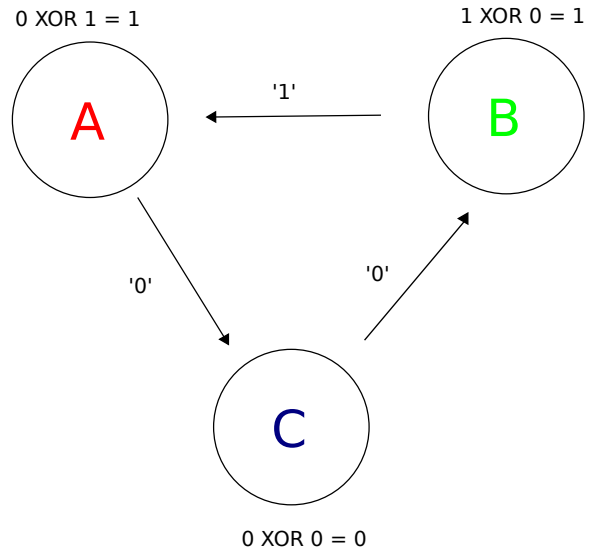
V druhém kole kryptografové, kteří neplatili, veřejně ohlásí výsledek logické operace *XOR* mezi těmito dvěma hodnotami, které si uchovávají. Pokud kryptograf platil, tak místo logické operace *XOR* použije logickou operaci *OR*. Když všichni kryptografové znají vzniklé hodnoty od ostatních kryptografů, tak mezi všemi těmito hodnotami použijí znovu logickou operaci *XOR*. Pokud bude výsledkem logická hodnota *true*, znamená to, že jeden z kryptografů platil, jinak to znamená, že neplatil nikdo. [8]

Na 4.1 a 4.2 lze pozorovat jak vypadají různé situace při placení.

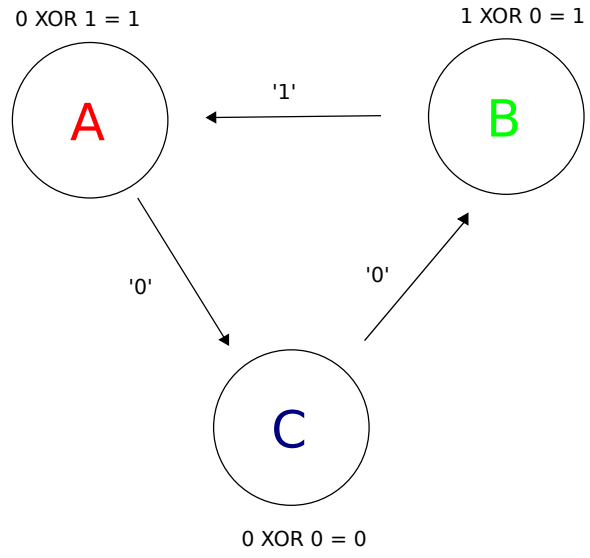
4.3 Návrh implementace

Prvním návrhem implementace tohoto protokolu bylo to, že bude pouze jeden plugin, který bude reprezentovat každého kryptografa a žádné další pluginy vytvářet nebude potřeba. Po krátké úvaze jsem si uvědomil, že pokud by existoval pouze jeden plugin, většina procesů znázorňujících práci tohoto protokolu by byla příliš zapouzdřena.

Druhým návrhem bylo, že pro představu ukázky házení mincí jednotlivých kryptografů by bylo velmi názorné, kdyby se vytvořil plugin simulující hod mincí, který by se až teprve napojil na jednotlivého kryptografa. Uživatel by si tak mohl vyzkoušet odebrat tento plugin a dát tam pevné hodnoty, aby si tak ověřil, že u tohoto protokolu nezáleží na hodnotě hozené



Obrázek 4.1: Simulace neplatících kryptografů



Obrázek 4.2: Simulace jednoho platícího kryptografa

mince, ale že tento princip náhodnosti pouze umožňuje anonymitu tohoto protokolu, protože při tomto faktoru náhodnosti nemůže nikdo očekávat žádné konkrétní hodnoty, které by odhalily, zda kryptograf platil nebo ne.

Třetím návrhem bylo, že bude pouze plugin na házení mincí a plugin znázorňující jednotlivého kryptografa. K pluginu kryptografa se pouze připojí plugin pro házení mincí a kryptografové se mezi sebou vhodně propojí a budou mít určité výstupy, ze kterých

by šlo vyčíst, jestli některý z propojených kryptografů zaplatil nebo ne. Tento návrh by byl dostatečně názorný pro pochopení tohoto protokolu, ale vytváření smyček v programu CrypTool s sebou obnáší komplikace, a to že je třeba určit začátek smyčky, tudíž jeden kryptograf by byl odlišný od ostatních kryptografů, což by se zbytečně lišilo od konceptu protokolu.

Nakonec jsem se rozhodl tento protokol implementovat tak, že jsem vytvořil pluginy celkem tři. Třetí protokol bude vyhodnocovat údaje od kryptografů a určí, zda jeden z nich skutečně zaplatil.

4.4 Implementace

Byly vytvořeny dva pluginy související s protokolem a jeden protokol, který lze využít i pro jiné účely(*Flip coin plugin*)

Flip coin plugin

Tento plugin se dá využít i mimo tento protokol. Znázorňuje jednotlivé hody mincí. Na začátku demonstrace se spustí generátor náhodných čísel, který určí hodnotu od nuly do jedné. Jednička znázorňuje pannu a nula znázorňuje orla. Tento plugin nemá žádné vstupy a pouze jeden výstup *Flip Coin*. Tento výstup má hodnotu datového typu *boolean* a znázorňuje číselné vyjádření hodu mincí.

Dining cryptographer

Tento plugin modeluje jednotlivého večeřícího kryptografa. Tento plugin má jeden atribut *Paid* datového typu *boolean*. Tento atribut se dá nastavit před spuštěním demonstrace protokolu a indikuje, jestli tento konkrétní kryptograf platil nebo ne.

Plugin má dva vstupy, prvním vstupem je *FlipCoin* a napojuje se na něj *Flip coin* plugin, tento vstup znázorňuje konkrétního kryptografa. Druhým vstupem je *SharedSecret* a do toho vstupu přichází sdílené tajemství od kryptografa po levici, který mu tak sděluje informaci o jeho hodu.

Na tento se vstup se také typicky váže *Flip Coin* plugin. Pokud kryptografovi přijdou hodnoty z obou dvou jeho vstupů, tak je zpracuje a vloží je na svůj výstup *Result*. Tyto hodnoty jsou zpracovány následovně: Pokud má kryptograf v nastavení, že neplatil, provede pouze logický *XOR* obou vstupů. Pokud kryptograf zaplatil, pak se provede logický *OR*. Všechny vstupy a výstupy mají datový typ *boolean*.

Dining final

Toto je plugin, který kalkuluje konečný výsledek tohoto protokolu. Nemá žádné speciální nastavení a má jeden vstup. Jeho vstup je datového typu *boolean* a měl by být napojen na výstupy jednotlivých kryptografů (plugin *Dining cryptographer*). Na tyto vstupy použije logickou operaci *OR* a výsledek se uloží na výstup *Result*. Výstup je také datového typu *boolean*. Pokud je výsledek *0*, znamená to, že žádný kryptograf nezaplatil, pokud je výsledek *1*, jeden z kryptografů platil.

4.5 Šablona

Šablona vytvořená pro tento protokol se skládá ze tří pluginů *Flip coin*, ze tří pluginů *Dining cryptographer*, jednoho pluginu *Dining final* a jednoho *BooleanOutput*. Plugin *BooleanOutput* patří mezi základní nástroje programu CrypTool. Pluginy *Flip coin* jsou napojeny na pluginy *Dining cryptographer*, a to prvním vývodem na svého kryptografa do vstupu *FlipCoin* a potom druhým vývodem do vstupu kryptografa vpravo do jeho vstupu *SharedSecret*. Výstupy kryptografů vedou všechny do pluginu *Dining final*, jehož výstup *Result* je vyveden do pluginu *BooleanOutput*. Po spuštění demonstrace lze vidět, podle hodnoty uložené v *BooleanOutput*, zda jeden z kryptografů zaplatil.

Kapitola 5

Coin Flipping

5.1 Úvod

Tento protokol umožňuje komunikaci dvou znepřátelených stran, které si navzájem nedůvěřují. Tyto dvě strany se dohadují o nějakém problému a nemohou to vyřešit kompromisem. Zároveň mohou komunikovat pouze zasíláním zpráv. Například to mohou být dva rozvedení manželé, kteří se po telefonu nemohou dohodnout, komu bude po rozvodu patřit auto.

Proto se tyto dvě strany rozhodnou, že svůj spor vyřeší hodem mincí. Jedna strana hodí mincí, druhá tipuje, jestli byla hozena panna nebo orel. Pokud tipuje správně, vyhrává, jinak tuto stranu čeká prohra. A zde je jádro problému, protože když tyto strany spolu komunikují pouze pomocí zasílání zpráv, nemá druhá strana jistotu, že ta první strana poctivě odhalí, co si na minci hodila. Zde je třeba použít Coin Flipping protokol, aby žádná strana nemohla podvádět a došlo tak ke spravedlivému rozsouzení. [9]

5.2 Koncept

Je mnoho variant tohoto protokolu, já jsem si vybral právě tuto, protože mi přišla jako velmi názorná a vhodná k implementování.

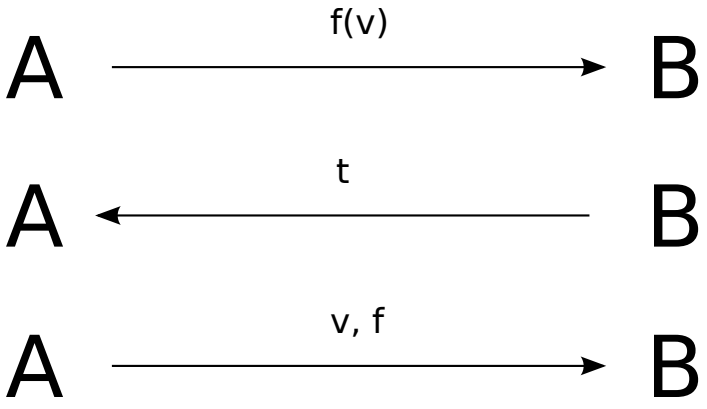
Jedna ze stran si hodí mincí, podívá se na výsledek a ten zašifruje jednosměrnou šifrovací funkcí. Je důležité, aby tato jednosměrná šifrovací funkce byla dostatečně složitá a hlavně její výsledky byly nepředvídatelné. Zašifrovaný výsledek pošle první strana druhé.

Druhá strana potom tipuje výsledek hodu mincí. Protože nezná šifrovací funkci, nemůže tedy předvídat, jaký tento výsledek je. Svůj tip pošle druhé straně a ta tento tip porovná se skutečností. Pokud druhá strana tipovala správně, vyhrává, pokud ne, tak prohrává. Výsledek pošle první strana druhé straně.

Druhá strana poté požaduje po první straně, aby jí poslala jednosměrnou šifrovací funkci, kterou zašifrovala výsledek hodu, aby si mohla ověřit, že první strana nepodváděla.

Proto je potřeba, aby šifrovací funkce měla nepředvídatelné výsledky, aby první strana nemohla podvádět. V opačném případě by si mohla vytvořit dvě šifrovací funkce, které by jí vracely stejný výsledek, ale jedna by měla jako vstup hodnotu pro pannu a druhá hodnotu pro orla. A až podle tipu druhé strany, by se rozhodla, kterou z těchto funkcí by druhé straně poslala. Druhé straně by tedy připadalo, že první strana nepodváděla, protože by měla funkci, která by jí pokaždé vracela šifrovaný výsledek. Schéma tohoto protokolu jde vidět na 5.1. Značky A a B reprezentují jednotlivé strany. "V"je hodnota mince hozená

stranou A, "T"je tip hodnoty hozené mince a "f"je jednosměrná šifrovací funkce. [3]



Obrázek 5.1: Schéma coin flipping protokolu

5.3 Návrh implementace

Je třeba správně navrhnout, které pluginy by prezentovaly jednotlivé strany. Strany jsou dvě a každá má značně odlišné chování, proto bude potřeba pro každou stranu navrhnout plugin zvlášť, protože obě dvě strany budou mít různé vstupy a výstupy. Počítám s tím, že, abych prezentoval hod mincí a popř. tip jedné strany, bude využit *Flip coin* plugin. Tento plugin bude ale využit pouze jako demonstrační šablona, ke vstupům jednotlivých hodů totiž bude možné připojit pevné hodnoty, aby si uživatel mohl demonstrovat správnost protokolu při různých hodnotách. Je možné, že bude nutno vytvořit plugin, který se postará o společnou komunikaci dvou stran, protože tento protokol probíhá ve více kolech, a tak by mohly vzniknout komplikace.

Tento návrh, vytvořit pluginy pro stranu A a pro stranu B, jsem nakonec zavrhl, protože nemůže být plugin, který by reprezentoval veškeré chování strany A, jelikož komunikace mezi stranami A a B probíhá ve více kolech a vytváření zpětných smyček je v programu CrypTool komplikované. Navíc, pokud by existovaly pouze dva pluginy s velkým počtem vstupů a výstupů, bylo by to pro uživatele velmi nepřehledné. Dokonce jsem se rozhodl, že implementovat speciální plugin pro začátek komunikace těchto dvou stran, kdy strana A pouze hází mincí a B tipuje výsledek, jde nahradit už hotovým pluginem *Flip Coin*.

Protokol jsem také kvůli implementaci v CrypTool trochu upravil, protože v CrypToolu nelze elegantně implementovat posílání šifrovací funkce. Proto jsem se rozhodl, že k řetězci 0 nebo 1, značící hodnotu hodu, připojím náhodný řetězec, který bude sloužit jako klíč. Druhá strana si proto nebude moci hod ověřit, dokud nedostane tento náhodný řetězec.

Rozhodl jsem se, že vytvořím pluginy prezentující stranu A při vytvoření klíče a jeho spojení s hodnotou hodu, a druhou fázi protokolu taktéž pro stranu A, kdy se určuje, která strana zvítězila.

5.4 Implementace

Implementovaly se dva nové pluginy (*CoinFlipping1* a *CoinFlipping2*)

CoinFlipping1

Vstupem tohoto pluginu je *CoinFlip* datového typu *boolean*. V nastavení je parametr určující klíč. Tento klíč si může uživatel jakkoliv nastavit. Tento plugin vezme hodnotu hodu, konvertuje ji na řetězec *0* nebo *1*. Plugin provede konkatenaci tohoto řetězce s řetězcem klíče a výsledek uloží na výstup *HashInput*. Ve výstupu *Key* je uložena hodnota klíče.

CoinFlipping2

Tento plugin reprezentuje druhou fázi protokolu pro stranu A. Má tři vstupy: *CoinFlipA* ,*CoinFlipB*, oba datového typu *boolean*, a vstup *Key*. Vstup *CoinFlipA* reprezentuje hodnotu, kterou si hodila strana A na začátku protokolu a *CoinFlipB* reprezentuje hodnotu mince, která byla zaslána straně A od strany B, značící tip hodnoty mince. Vstup *Key* prezentuje klíč použitý v prvním kroku.

Výstupy tohoto pluginu jsou *Success*, *CoinResult* a *Key*. *Success*, a *HashFunctionEnable* jsou datového typu *boolean* a *CoinResult*je datového typu *string*. *CoinResult* je výsledek hodu mincí strany A, který je určen k poslání straně B. Tato hodnota je v datovém typu *string*, protože tato hodnota se bude zřetězovat s klíčem, přičemž zřetězený celek použije strana B k ověření pravosti. *Success* je hodnota značící úspěšnost tipu strany B. Výstup *Key* je hodnota klíče použitého v prvním kroku. Tato hodnota se sice neupravuje, ale přesto se nezměněná vkládá a posílá dál z tohoto pluginu. Je to sice zbytečné, protože bych mezikrok přes tento plugin mohl vynechat, ale přesto jsem se to rozhodl udělat tímto způsobem, abych demonstroval, že strana B může dostat klíč teprve, až bude strana A v druhé fázi protokolu.

U tohoto pluginu lze nastavit, zda se strana A chová čestně nebo ne. Pokud je u pluginu nastaveno nečestné chování, tak strana A vždy hlásí, že strana B prohrála (hodnota ve výstupu *Success* je vždy *false*), ignoruje svůj skutečný hod (*CoinFlipA*) a oznamuje, že si hodila opačný hod, než strana B tipuje (hodnota *CoinResult* je vždy opačná než hodnota *CoinFlipB*. Pokud se strana A chová čestně, tak porovnává hodnoty vstupů *CoinFlipA* a *CoinFlipB* a do výstupu *Success*, dává výsledek porovnání, jestli tedy strana B tipovala správně a vyhrává. Do výstupu *CoinResult* se ukládá hodnota z vstupu *CoinFlipA*.

Některé naimplementované pluginy znázorňující jednosměrné funkce mají vstupy datového typu *ICrypToolStream*, přesto jsem zvolil typ výstupu jako datový typ *string*. Nastavil jsem to tak z toho důvodu, že tyto dva datové typy jsou mezi sebou navzájem kompatibilní a také proto, že *ICrypToolStream* je zvláštní datový typ, který slouží především k posílání velkého množství znaků, nejčastěji čteného z nějakého souboru, ale tento plugin posílá pouze krátké, jednoznakové řetězce, takže by použití tohohle datového typu bylo zbytečné.

5.5 Šablona

Byla vytvořena šablona, kterou si uživatelé programu CrypTool mohou prohlédnout a demonstrovat si tak chod tohoto protokolu. Šablona obsahuje plugin *CoinFlipping1* reprezentující stranu A a její hod. Výstup *BoolValue* je vyveden do vstupu *CoinFlip* od pluginu *CoinFlipping1*.

V tomto pluginu je již nastaven výchozí klíč, který se dá snadno upravit a jehož hodnota není pro chod protokolu podstatná. Z pluginu je výstup *ValuedKey* vyveden do vstupu pluginu znázorňující jednosměrnou šifrovací funkci. K tomuto účelu jsem použil plugin *SHA*. Výstup *Hashed value* pluginu *SHA* byl připojen do vstupu *Stream-One* defaultního plu-

ginu *StreamComparator*. Druhý výstup *Key* pluginu *CoinFlipping1* je vyveden do pluginu *CoinFlipping2* a jeho vstupu *Key*.

Tipování hodu mincí, který provádí strana A, jsem taktéž reprezentoval pluginem *FlipCoin*. Zde jsem ale využil pouze výstup *BoolValue*, který jsem propojil se vstupem *CoinFlipA* pluginu *CoinFlipping2*.

Plugin *CoinFlipping2* prezentuje druhý krok strany A, kdy přijal tip od strany B a nyní určuje výsledek jejich sporu, přičemž straně B posílá klíč k jednosměrné šifrovací funkci. Plugin je nastaven na čestné chování. Výstup *Success* je napojen na plugin *BooleanOutput*, jehož hodnota znázorňuje výsledek vyhlášený stranou A. Výstup *Key* a *FlippedCoinValue* jsou připojeny na plugin *StringOperation*, který je nastaven na konkatenaci.

Výsledek konkatenace se posílá z výstupu *OutputString* do pluginu druhého pluginu *SHA*. Výstup *HashedValue* je zaveden do stejného pluginu *StreamComparator* jako první plugin *SHA*, ale do vstupu *Stream-Two*. Výstup *Comparator Achievement* je propojen s pluginem *BooleanOutput*. Tento plugin znázorňuje, zda strana A nepodváděla.

Kapitola 6

Zero-Knowledge

6.1 Úvod

Jsou dvě strany A a B, které si navzájem nedůvěřují. Strana A prohlašuje, že zná nějaké určité tajemství nebo informaci, kterou by strana B ráda věděla. Strana A proto požaduje po straně B, aby jí zaplatila a až poté jí toto tajemství předá. Protože strana B straně A nevěří, že opravdu takové tajemství zná, proto po straně A vyžaduje, aby jí to tajemství předvedla.

Strana A ovšem nechce toto tajemství straně B předat, dokud nedostane zaplaceno, protože kdyby strana B toto tajemství znala, neměla by už důvod proč dát straně B peníze. A zde nastává problém, kterým se zabývá Zero-Knowledge protokol, jak přesvědčit stranu B, že strana A opravdu zná toto tajemství, aniž by straně B toto tajemství prozradila. [7]

6.2 Koncept

Zero-Knowledge protokol předpokládá, že strana A zná nějakou informaci, která jí umožňuje vrátit správný výstup, pokaždé když jí strana B pošle nějaký vstup. Strana B proto posílá straně A rozdílné vstupy a ověřuje její výstupy, dokud strana A nepošle špatný výstup nebo dokud strana B neusoudí, že strana A opravdu danou informaci zná.

Princip Zero-Knowledge protokolu lze ukázat na následující situaci. Jsou dva lidé: Alice a Bob. Alice prohlašuje, že zná tajné heslo, které odemyká kouzelné dveře v jeskyni, která má dvě chodby a ty chodby jsou zezadu propojeny právě těmito kouzelnými dveřmi. Bob, jak už bylo naznačeno v úvodu protokolu, Alici nevěří, že toto heslo zná, a proto požádá Alici, aby mu znalost tohoto tajemství demonstrovala.

Dohodnou se, že Alice půjde do jedné ze dvou chodeb a Bob na ni pak zavolá, ze které chodby má vyjít. Alice tedy například půjde do první chodby a Bob na ni zavolá, ať vyjde z druhé chodby. Pokud by Alice neznala tajné heslo, tak z druhé chodby vyjít nemůže a vyšlo by tak najevo i Bobovi, že Alice mu lže. Pokud by ale Bob požadoval po Alici, ať vyjde z první, tak by Alice mohla z této chodby vyjít, aniž by musela znát tajné heslo. Proto je třeba tento protokol několikrát opakovat, aby se pravděpodobnost této možnosti výrazně omezila. Proto tento protokol může Bobovi pouze dokázat, že Alice toto tajemství nezná, ale nelze pomocí něj s naprostou jistotou určit, že toto tajemství Alice zná. [12]

6.3 Návrh implementace

Zero-Knowledge protokol by byl nejlépe demonstrovatelný na odhalování tajemství, kdy strana A by znala jeden určitý Hammiltonův graf, strana B by znala cestu a požadovala by po straně A, aby jí dokázala, že ví, jak ten určitý Hammiltonův graf vypadá. Bohužel tento návrh jsem zavrhl, protože v programu CrypTool neexistuje žádná struktura, která by reprezentovala Hammiltonův graf.

Zajímavé by bylo prezentovat Zero-Knowledge protokol tak, že by obě dvě strany využívaly plugin nějaké šifrovací funkce. Strana A by přijímala od strany B hodnoty, které by měla vkládat do té určité šifrovací funkce a pak by straně B posílala zpátky výsledky, které by si strana B sama ověřila tím, že by do té stejné šifrovací funkce tyto hodnoty vložila.

Tento návrh jsem zavrhl ze dvou důvodů. Kdyby strana A danou funkci neznala, velmi těžko by předstírala znalost tajemství nějakým náhodným výsledkem, protože by neznala rozsah správných výsledků, takže šance, že by si správně tipla hodnotu, kterou by poslala zpátky B, byla téměř nulová, proto by při demonstraci tohoto protokolu nevynikla možnost opakovaného testování.

Druhým důvodem, proč byl tento návrh mnou zavržen, je skutečnost, že tento protokol předpokládá, že strana A má nějakou informaci, kterou strana B nevlastní, proto by bylo nelogické, kdyby obě strany, A i B vlastnily stejnou šifrovací funkci, takhle by hlavní myšlenka tohoto protokolu nebyla jasná.

Nakonec jsem se rozhodl, že použiji návrh, který bude zaměřen na opakované testování a dá straně A možnost přídstírat znalost tajemství, a přesvědčit tak stranu B o tom, že dané tajemství opravdu zná, pokud strana B nebude požadovat dostatečné množství testování. V tomto návrhu jsem vycházel hodně z konceptu, který jsem k tomuto protokolu popsal. Strana B si může zvolit z kolika možností se bude vybírat správný výsledek a kolik celkem provede pokusů, aby si ověřila, že strana A opravdu dané tajemství zná. Mezi těmito pluginy bude muset být vytvořena zpětná vazba, aby tento protokol mohl být proveden vícekrát, podle nastaveného počtu pokusů. Nebude zde tedy vytvořeno klasické schéma, kdy jdou určité vstupy k jednomu pluginu, ten je zpracuje a pak pošle pluginy k dalšímu pluginu, a tak je každý jednotlivý plugin aktivován pouze jedenkrát.

V tomto návrhu se nebude řešit, jakým způsobem strana A získá správný výsledek, ale půjde zde pouze nastavit, že dané tajemství opravdu zná.

Strana B bude straně A posílat hodnoty z daného rozsahu množiny výsledků a bude kontrolovat, zda bude strana A na tyto hodnoty správně reagovat. Pokud bude mít strana A nastaveno, že dané tajemství zná, vždy bude posílat, že ověření proběhlo v pořádku, pokud bude mít nastaveno, že dané tajemství nezná, potom bude mít možnost tipovat, které číslo jí strana B pošle, a pokud bude mít štěstí a tipované číslo se bude shodovat s číslem od strany B, tak potom ověření také proběhne kladně.

Pro objasnění k popsanému konceptu protokolu podotýkám, že poslaným číslem strana B říká straně A, ze které chodby má strana A vyjít. Strana A, pouze pokud má štěstí, že si vygeneruje správné číslo (tedy že zašla do správné chodby) nebo zná tajemství na otevření průchodu, tak může vyjít z požadované chodby a poslat straně B zpátky stejné číslo.

Na tomto návrhu půjde názorně demonstrovat, jak moc se mění úspěšnost strany A blafovat, že dané tajemství zná, podle počtu pokusů a rozsahu množiny výsledků.

6.4 Implementace

Byly naimplementovány dva pluginy reprezentující jednotlivou stranu A nebo stranu B. Tyto pluginy jsou *Zero Knowledge Checker* a *Zero Knowledge Checked*.

Zero Knowledge Checker

Zero Knowledge Checker plugin reprezentuje stranu B. Tedy stranu, která nezná to určité tajemství a požaduje po druhé straně A, aby jí ukázala, že toto tajemství opravdu zná. Tento plugin má jeden vstup *Input*. Tímto vstupem plugin přijímá poslané výsledky od strany A. Plugin má dvě nastavení. První nastavení *AmmountOfAttempts* značí, kolik pokusů chce strana B udělat, aby si ověřila, že strana A zná dané tajemství. Druhým parametrem tohoto pluginu *AmmountOfOptions* je rozsah množiny hodnot výsledků. Tento plugin se spouští tolikrát, kolik má nastaveno pokusů. Pokaždé náhodně vygeneruje hodnotu větší nebo rovnu 0 ,a menší, než je nastaven rozsah možných výsledků. Ke generování náhodných čísel je použita třída *BigIntegerHelper*, která je součástí knihoven vytvořených speciálně pro program CrypTool. Menší nevýhodou použití této knihovny je, že nepracuje se standardními datovými typy *integer*, ale používá speciální datový typ *BigInteger*. Vygenerované náhodné číslo se poté odešle výstupem *OutputRandom*. Výstupem *AmmountOfOption* se posílá počet variant možných výsledků. Při prvním spuštění pluginu se nastavuje do výstupu *Success*, který je datového typu *Boolean*, hodnota *true*. Plugin ovšem po každém spuštění, aktivovaným došlou hodnotou na vstup *Input*, kontroluje došlou hodnotu, jestli se shoduje s naposledy odeslanou hodnotou. Pokud budou tyto dvě hodnoty rozdílené, znamená to, že strana A dané tajemství nezná, a proto se nastaví hodnota výstupu *Success* jako *false*.

Zero Knowledge Checked

Tento plugin reprezentuje stranu A, která zná určité tajemství a snaží se to dokázat, nebo dané tajemství nezná a snaží se přesvědčit stranu B, že ho ve skutečnosti ho zná. Má dva vstupy, *Input* a *AmmountOfOptions*. Oba dva tyto vstupy jsou datového typu *BigInteger* a to z toho důvodu, že se zde pro generování náhodných čísel využívá třídy *BigIntegerHelper*. Tento plugin má nastavení určující, jestli strana A opravdu zná dané tajemství. Pokud strana A tajemství zná, tak všechna čísla, která jí dojdou přes vstup *Input* posílá nezměněná přes vstup *Output*. Proto strana B vyhodnotí všechny její výstupy jako správné. Pokud ale strana A dané tajemství nezná, tak si pouze vygeneruje náhodné číslo od 0 do hodnoty dané vstupem *AmmountOfOptions*, které udává počet variant výsledků. Pokud si strana B vygeneruje stejné číslo jako to, které jí bylo posláno, pak si bude strana B myslet, že tajemství opravdu zná.

6.5 Šablona

Vytvořil jsem šablonu pro znázornění tohoto protokolu. Hlavním jádrem této šablony jsou dva pluginy *Zero Knowledge Checked* a *Zero Knowledge Checker*. Tyto dva pluginy reprezentují jednotlivé strany A a B. Do pluginu *Zero Knowledge Checker* jsou zapojeny do jeho vstupu *Input* dva pluginy a to *Zero Knowledge Checked* a *Integer input*. Toto zvláštní zapojení dvou pluginů do jednoho vstupu je použito z důvodu vytvoření zpětné vazby mezi pluginy *Zero Knowledge Checker* a *Zero Knowledge Checked*, která demonstruje opakované ověřování, že strana A opravdu tajemství zná.

Kdyby byl plugin *Integer input* vynechán, tak by se žádný z ostatních pluginů neaktivoval. Hodnota tohoto pluginu je irelevantní, plugin *Zero Knowledge Checker* při prvním aktivování hodnotu vstupu *Input* nevyužívá.

Výstupy pluginu *Zero Knowledge Checker*, *AmmountOfOptions* a *OutputRandom* jsou vyvedeny do vstupů pluginu *Zero Knowledge Checked*, *AmmountOfOptions* a *Input*. Vstupem *AmmountOfOptions* je strana A informována, kolik je možných výsledků, což jí tak umožňuje předstírat znalost tajemství. Vstupem *OutputRandom* je demonstrováno posílání požadavků strany B pro stranu A, která, prezentována *Zero Knowledge Checked* pluginem, musí správně poslat odpověď svým výstupem *Output*. Po ukončení daného počtu pokusů posílá plugin *Zero Knowledge Checked* svým výstupem *Success* logickou hodnotu do pluginu *Boolean Output*, která signalizuje, zda strana A dané tajemství zná.

Kapitola 7

Yao Millionaire's Problem

7.1 Úvod

Tento protokol se zabývá problémem dvou milionářů. Tento probléme je následující: Jsou dva milionáři a chtěli by zjistit, který z nich dvou je bohatší. Oba dva si navzájem nedůvěřují a nechtějí tomu druhému milionáři prozradit, kolik peněz ve skutečnosti mají. A přitom neexistuje žádná třetí strana, které by oba dva milionáři důvěřovali natolik, aby jí prozradili, kolik mají peněz a ona je tak rozsoudila. Tento protokol následující problém řeší, milionáři se dozvědí, který z nich má více peněz a ani jednomu z nich nebude poskytnuta informace o skutečné částce. [1]

7.2 Koncept

Pro činnost tohoto protokolu se využívá asymetrický kryptografický systém. Vygenerují se privátní a veřejné klíče. Veřejný klíč dostanou obě dvě strany a privátní klíč dostane strana A. Pro snadné vysvětlení konceptu budeme považovat počet peněz strany A za I, počet peněz strany B za J. Dále se milionáři musí dohodnout, jaké bude maximum, které jeden z nich může mít. Většinou se bere počet peněz milionáře v miliónech, protože čím je možné maximum větší, tím je protokol početně složitější. Tento protokol probíhá obecně ve více krocích. [18]

Krok 1

Strana B si zvolí náhodné N-bitové číslo, které nazveme x. (Strana A později bude používat N/2 bitové číslo, takže velikost tohoto čísla je důležitá). Poté strana B vypočítá C. C je zašifrované číslo x kryptografickým algoritmem, kdy je použit veřejný klíč.

Krok 2

Strana B číslo C použije tak, že vypočítá C - J + 1. Výsledek poté pošle straně A

Krok 3

Strana A vygeneruje sérii čísel, Y1, Y2, Y3... až do zvoleného maxima peněz. Y1 je zde rozšifrované číslo C - J + 1, které jí poslala strana B. Strana A použije pro rozšifrování svůj

soukromý klíč. Y2 je rozšifrované číslo (C - J + 1) + 1, Y3 je rozšifrované číslo (C - J + 1) + 2...

Krok 4

Strana A vygeneruje náhodné prvočíslo číslo p, které má velikost N/2 bitů. A vypočítá čísla Z1, Z2, Z3... až do zvoleného maxima peněz. Z se vypočítá jako Y modulo p.

Krok 5

Strana A nyní pošle číslo P a všechna čísla Z straně B.

Krok 6

Strana B přijme čísla Z a vybere si J-té číslo Z. Poté vypočítá číslo G, kdy G je x modulo p. Poté porovná G s J-tým číslem Z. Pokud je J-té Z rovné číslu Z, potom je strana A bohatší než strana B, nebo má stejný počet peněz. Pokud se tato čísla nerovnají, je strana A chudší než strana B.

7.3 Návrh implementace

Tento protokol má jasné kroky, které se musí provést, aby se bezpečně zjistilo, která strana je bohatší. V tomto protokolu jsou čtyři důležité parametry, které je třeba před začátkem protokolu určit. Těmito parametry je počet peněz obou milionářů, maximální počet peněz, které může milionář mít a číslo N, které určuje bitovou délku náhodně vygenerovaných čísel použitých v protokolu. Tyto parametry by se měly vkládat do pluginů *NumberInput*, vývody z těchto pluginů budou vyvedeny do vstupů dalších pluginů, které tyto parametry budou využívat.

Vhodným asymetrickým kryptografickým systém pro tento protokol je RSA. RSA je spolehlivý asymetrický kryptografický systém, jehož výhodou je to, že je již v programu CrypTool implementován. [5] Pro využití RSA jsou v CrypTool dva pluginy, *RSA* a *RSA Key Generator*, které oba dva využiji při implementaci. *RSA Key Generator* bude použit pro vygenerování soukromých a veřejných klíčů.

V prvním kroku je nutno vygenerovat náhodné číslo, které by mělo N-bitovou délku, tento plugin CrypTool obsahuje, je to plugin *PrimeGenerator*. Je ale třeba v pozdějším kroku vygenerovat další číslo, které by mělo N / 2 bitovou délku a tak by při použití tohoto pluginu bylo třeba odděleně nastavovat N u prvního generátoru a N / 2 u druhého generátoru. Toto řešení, kdy by uživatel byl nucen nastavovat N a N / 2 zvlášť u obou generátorů, mi přišlo těžkopádné. Proto jsem se rozhodl, že budu modifikovat tento plugin a přidám mu výstup, který bude přijímat bitovou délku vygenerovaného čísla. Díky tomu bude moci uživatel nastavit do jednoho pluginu N a to bude použito v obou generátorech. Pro zbylou část prvního kroku využiji již zmiňovaný plugin *RSA*.

V druhém kroku vyřeším matematickou operaci C - J pomocí výchozího pluginu *NumberOperation*, který umí provést všechny základní matematické operace. Pro přičtení 1 využiji plugin *IncDec*. Zkombinováním obou těchto pluginů vyjde chtě C - J + 1.

Pro provedení třetího kroku budu muset vytvořit již samostatný plugin.

Ani čtvrtý krok nelze nahradit výchozími pluginy. Také zde nastává problém v tom, že mezi pluginy je třeba poslat sérii čísel, jejichž počet není předem dán. Tento problém je možno řešit tím, že se tato série čísel pošle v kontejneru, a nebo postupně. Rozhodl jsem

se, že čísla mezi těmito dvěma pluginy budu posílat postupně z toho důvodu, protože bych musel zapouzdřit RSA algoritmus přímo do pluginy, takto při postupném posílání mohu využít již vytvořeného *RSA* pluginu. Také je to dle mého názoru názornější pro uživatele.

Pátý krok bude taktéž součástí druhého pluginu. Posílání je zde znázorněno vazbami mezi pluginy.

Pro šestý krok je rovněž potřeba vytvořit samostatný plugin. Zde už je praktičtější posílat všechna čísla v kontejneru a ne postupně, protože se s čísly nemusí vykonávat žádná operace, kterou by bylo možné nahradit výchozím pluginem. Tento třetí plugin bude také ukazovat, jestli je první milionář bohatší nebo ne.

7.4 Implementace

Pro tento protokol jsem implementoval celkem čtyři pluginy, tři pluginy jsou spojené výhradně s protokolem a jeden plugin může mít využití i jinde.

PrimeGeneratorWithInput

Tento plugin je podobný pluginu *PrimeGenerator*. Jeho rozdílem je to, že generuje čísla pouze podle počtu bitů, kdy se tento počet bitů neurčuje v nastavení pluginu, ale používá se pro to vstup datového typu *integer*. Výstupem generátoru je číslo datového typu *BigInteger*, který obsahuje vygenerované náhodné číslo.

Yao1

Vstupy tohoto pluginu jsou *C* a *MaxMoney*. C značí hodnotu poslanou straně A v druhém kroku. *MaxMoney* je maximální hodnota peněz, kterou milionář může mít. V tomto pluginu je zapouzdřen 3. krok. Podle maximální hodnoty peněz se vygeneruje stejný počet čísel, prvním číslem je C, další čísla jsou vždy o jedno větší. Plugin posílá čísla postupně, aktivuje svůj výstup *Y* pro každé číslo.

Yao2

Tento plugin zapouzdřuje 4. a 5. krok. Jeho vstupy jsou *Y*, *MaxMoney*, *p* a *I*. Tento plugin očekává, že mu bude posláno tolik čísel na jeho vstup *Z*, jaká je hodnota vstupu *MaxMoney*. Po přijetí všech očekávaných čísel použije plugin na tato čísla operaci modulo *p*, kde *p* je hodnota vstupu *p*. Hodnota vstupu *I* určuje počet peněz milionáře A. Podle tohoto čísla se určuje, kterým číslům bude připočtena jedna. Budou to ta čísla, jejichž pořadové číslo je stejné nebo větší než *I*. Takto upravená čísla se uloží do kontejneru, který poté bude poslán výstupem *Z*.

Yao3

Plugin zapouzdřuje 6. krok. Vstupem *Z* je přijímán kontejner čísel od strany A. Dalšími vstupy jsou *J*, *x* a *p*. V pluginu je vybráno z kontejneru *J*-té číslo a je porovnáváno s číslem *x* modulo *p*. Pokud jsou tato dvě čísla rovna, je tím zjištěno, že strana B je bohatší, pokud jsou čísla rozdílná, strana A má více peněz. Skutečnost, jestli je strana B bohatší, vyjadřuje výstup *BIsRicher*, který je datového typu *boolean*.

7.5 Šablona

Byla vytvořena šablona pro znázornění tohoto protokolu. Tato šablona se skládá z pěti pluginů *NumberInput*. Dva pluginy jsou pro nastavení hodnot bohatství dvou milionářů, jeden pro číslo N a pro číslo určující maximální počet peněz. Pátý plugin je zde pouze pomocný a uživatel by neměl jeho hodnotu upravovat, protože pak protokol nemusí správně fungovat. V tomto protokolu je nastavena hodnota *2*.

Plugin s obsahem znázorňující číslo N je zaveden do pluginu *PrimeGeneratorWithInput*, jehož výstupem je náhodné N-bitové číslo x a také do pluginu *NumberOperation*. Plugin *NumberOperation* je nastaven na dělení a jeho druhým vstupem je již zmíněný pátý plugin s implicitně nastavenou hodnotou *2*. Výstupem *NumberOperation* pluginu bude tedy číslo *N / 2*. Číslo *N / 2* je zavedeno do druhého pluginu *PrimeGenerator*, jehož výstupem je číslo p. Náhodně vygenerované číslo x je vloženo do *RSA* pluginu, nastaveného na zašifrování. Klíče pro tento plugin jsou vygenerovány pluginem *RSA Key Generator*.

Zašifrované x neboli c je vloženo do vývodu pluginu, sloužícímu pro matematické operace, *NumberOperation*. Druhým vstupem je *NumberInput* plugin, obsahující počet peněz milionáře B. Plugin *NumberOperation* má na svém výstupu rozdíl těchto hodnot.

Výstup je zaveden do pluginu *IncDec*, který je nastaven na přičítání o hodnotu *1*.

Tato hodnota je vložena do vstupu C pluginu *Yao1*. Ostatními vstupy tohoto pluginu jsou *MaxMoney* propojený s pluginem *NumberInput* obsahující hodnotu maximálního počtu peněz. Výstup tohoto pluginu Y je zaveden do pluginu *RSA*, který je nyní nastaven na dešifrování, klíče jsou převzaty ze stejného pluginu *RSA Key Generator* jako u předešlého pluginu *RSA*.

Dešifrovaná zpráva je připojena na vstup pluginu *Yao2*, který má také vstup *MaxMoney* spojený se stejným pluginem *NumberInput*. *Yao2* má také vstup p, do kterého je vkládána hodnota p a vstup I, do kterého je vložen z *NumberInput* počet peněz milionáře A. Výstup Z je spojen se vstupem Z pluginu *Yao3*.

Dalšími vstupy *Yao3* jsou p a x, do kterých jsou vloženy stejnojmenné hodnoty. Čtvrtým vstupem je J, do kterého je vkládána hodnota značící počet peněz milionáře B. Vývod pluginu *BIsRicher* je zaveden do pluginu *BooleanOutput*, který udávázda je milionář B bohatší než milionář A.

Kapitola 8

Oblivious Transfer

8.1 Úvod

Tento protokol řeší následující problém. Jsou dvě strany A a B, strana A má určitý počet zpráv a strana B chce dostat jednu určitou zprávu, kterou má strana A. Strana A ale nechce, aby si strana B přečetla všechny její zprávy, pouze tu jednu, kterou chce. Strana B naopak nechce, aby strana A zjistila, kterou z těch zpráv si chce přečíst. Neexistuje zároveň žádná třetí strana, které by strana A chtěla poslat všechny zprávy a strana B chtěla prozradit, kterou zprávu chce. [15]

8.2 Koncept

Tento protokol jsem přetvořil z Oblivious transfer protokolu 1-2. Koncept je stejný, pouze místo dvou zpráv, je zpráv n.

Krok 1

Strana A vlastní zprávy *m0, m1, m2 ...*. Strana A vygeneruje klíče kryptografického systému RSA, N je veřejné modulo, e je veřejný exponent a d je soukromý exponent. Dále strana A vygeneruje náhodné zprávy *x0, x1, x2 ...*, zpráv x je stejně jako skutečných zpráv m, tyto zprávy x pošle straně B.

Krok 2

Strana B vytvoří číslo b, které vyjadřuje index chtěné zprávy (začíná se od 0). Vygeneruje číslo k, které zašifruje pomocí RSA kryptosystémem a to přičte k číslu x_b, výsledek v pošle straně A.

Krok 3

Strana A vypočítá sérii možných k tak, že dešifruje pomocí RSA kryptosystému za použití svého soukromého klíče $v \ - x$, pro každé x. Pomocí vzniklých k, kterých je stejný počet jako zpráv, pošle straně B všechny takto upravené zprávy: $n \ = m \ + k$.

Krok 4

Strana B použije své k, aby si přečetla zprávu, kterou požadovala. [13]

8.3 Návrh implementace

Vzhledem k tomu, že v protokolu je využit RSA algoritmus, budu muset použít plugin pro generování RSA klíčů *RSA Key Generator*. Rozhodl jsem se, že za zprávy budu považovat čísla a nikoliv znaky. Tento přístup je dle mého názoru názornější a vhodnější na implementaci, vzhledem k tomu, že v průběhu protokolu jsou zprávy součástí matematických operací. Také jsem se rozhodl, že bude názorné, pokud každá zpráva bude jeden *IntegerInput* plugin. Uživatel si tak vytvoří tolik zpráv, kolik bude chtít. Bude pouze muset nastavit, kolik těch zpráv je, aby příslušný plugin všechny zprávy očekával. Pro generování náhodných zpráv není žádný již hotový plugin, z toho důvodu pro to vytvořím také samostatný plugin. Plugin pro generování zpráv bude muset vygenerované zprávy posílat v kontejneru, protože zpráv bude neurčitý počet.

Plugin reprezentující první krok není třeba, pouze zde bude použit mnou vytvořený generátor náhodných čísel. Druhý krok jsem se rozhodl zapouzdřit do samostatného pluginu, a tento plugin bude využívat RSA klíče. Rozhodl jsem se do pluginu zapouzdřit RSA algoritmus, protože v tomto případě dle mého názoru nejde o příliš komplexní krok, kde by uživateli bylo skryto příliš mnoho procesů. U třetího a čtvrtého kroku je nutno také vytvořit speciální plugin, je to i z toho důvodu, že se zde manipuluje s více, předem neznámými počty čísel, a tak je třeba použít pro tato čísla kontejner a výchozí pluginy programu CrypTool s kontejnery nepracují.

8.4 Implementace

Pro účely tohoto protokolu byly vytvořeny celkem čtyři pluginy, z čehož jeden plugin lze využít i mimo protokol. Zbylé pluginy úzce souvisí s protokolem.

RandomMessageGenerator

Random message generator je plugin, který slouží pro generování náhodných zpráv. Jeho vstup *AmmountOfMessage* udává počet zpráv, které tento generátor vygeneruje. Generátor lze nastavit, aby generoval zprávy pouze do určité maximální hodnoty. Jednotlivá zpráva je datového typu *BigInteger*. Tyto zprávy se ukládají do kontejneru *List*, který se potom posílá z výstupu *Messagess*.

ObliviousTransfer1

Tento plugin prezentuje druhý krok protokolu vykonávaný stranou B. Jeho vstupy jsou x, b, e a N. Vstup x přijímá jako vstup náhodně vygenerované zprávy stranou A, uložené v kontejneru *List*. Vstup b značí index zprávy, kterou by strana B ráda dostala. Zbylé vstupy jsou pro klíče RSA kryptosystému. RSA šifrující algoritmus je zabudován přímo v pluginu.

Plugin také vygeneruje náhodné číslo k, jehož maximální hodnota je omezena nastavením.

Plugin z dodaných vstupů vypočítává hodnotu v. Tato hodnota se vypočítá pomocí rovnice $v = (x_b + k^e) \bmod N$. Hodnota v se poté ukládá na stejnojmenný výstup v. Na výstup k se uloží hodnota k.

ObliviousTransfer2

Tento plugin prezentuje třetí krok protokolu vykonávaný stranou A. Tento plugin má 6 vstupů. Vstup *message* očekává hodnotu zprávy. Tento plugin se aktivuje vícekrát, a to podle počtu zpráv. Plugin nejprve čeká, dokud mu nedojdou všechny zprávy, a až poté s nimi pracuje. Pomocí hodnoty vstupu *count* se zjistí, kolik je celkem zpráv a na kolik zpráv má tedy plugin čekat. Dalšími vstupy je x s kontejnerem náhodných zpráv, v s hodnotou z druhého kroku od strany B, e a N se soukromými klíči pro RSA kryptosystém. Plugin vypočte sérii možných k tak, že $k_i = (v - x_i)^d \bmod N$. Z těchto hodnot k je pouze jediná shodná s hodnotou k, kterou vygenerovala strana B. Proto strana A vytvoří sérii zašifrovaných zpráv n, kdy $n_i = n_i + k_i$. Zašifrované zprávy se ukládají do kontejneru *List* a pak se hromadně odesílají výstupem *cryptedMessages*.

ObliviousTransfer3

Plugin prezentuje závěrečný 4. krok protokolu vykonávaný stranou B. Jeho vstupy jsou b, hodnota chtěného indexu zprávy, k náhodně vygenerovaná hodnota v 2. kroku a *cryptedMessages* s kontejnerem zašifrovaných zpráv. Plugin získá chtěnou zprávu tak, že vypočte $m_b - k$. Tato zpráva se poté uloží na výstup *message*.

8.5 Šablona

Šablona demonstrující protokol vypadá následovně: byly vytvořeny dva pluginy *IntegerInput*, jeden udávající počet zpráv a druhý udávající hodnotu b, tedy indexu chtěné zprávy. Hodnota počtu zpráv byla nastavena jako 2 a číslo b jako 1. Plugin s hodnotou udávající počet zpráv je napojen na mnou vytvořený plugin *RandomMessageGenerator*, do jeho vstupu *AmmountOfMessage*. Hodnota toho vstupu bude udávat počet náhodně vygenerovaných zpráv.

Pro generování soukromých a veřejných klíčů jsem do šablony přidal plugin *RSA Key Generator*.

Plugin *IntegerInput* s hodnotou b je připojen na pluginy *ObliviousTransfer1* a *ObliviousTransfer3* do stejnojmenných vstupů b. Plugin *ObliviousTransfer1* má v dalších jeho vstupech N a e zapojené veřejné klíče od pluginu *RSA Key Generator*. V posledním vstupu x je mu posílán kontejner náhodně generovaných zpráv z pluginu *RandomMessageGenerator*.

Výstup v pluginu *ObliviousTransfer1* v je napojen do stejnojmenného vstupu pluginu *ObliviousTransfer2*. Tento plugin má ve vstupech N a d napojeny soukromé klíče z pluginu *RSA Key Generator*. Plugin *IntegerInput* s hodnotou o počtu zpráv je připojen ke vstupu pluginu *ObliviousTransfer2* do vstupu *count*. V tomto případě jsou do vstupu *message* zapojeny dva pluginy *IntegerInput* znázorňující samotné zprávy. Do vstupu x tohoto pluginu je zde také zaveden vývod z pluginu *RandomMessageGenerator*.

Výstup *cryptedMessages* pluginu *ObliviousttTransfer2* posílá kontejner zpráv do stejnojmenného vstupu pluginu *ObliviousttTransfer3*. Další jeho vstup k je spojen se stejnojmenným výstupem pluginu *ObliviousTransfer1*. Do vstupu b je zavedena hodnota indexu chtěné zprávy. Tento plugin dešifruje chtěnou zprávu a její hodnotu uloží na výstupu *message*. Ten je pro demonstraci propojen s pluginem *TextOutput*, aby po spuštění této šablony bylo vidět, která zpráva byla doručena.

Kapitola 9

Závěr

Tato práce se zabývá programem CrypTool. Čtenáři byli s tímto programem podrobně seznámeni. Bylo jim vysvětleno, co je to program CrypTool, v čem je tento program užitečný a jak se lze připojit k jeho vývoji.

Bylo zde objasněno, jak se s programem CrypTool pracuje a čeho se musí vývojáři vyvarovat, pokud se snaží implementovat nové komponenty do tohoto programu.

Program CrypTool byl analyzován a bylo zjištěno, které pluginy tomuto programu chybí a bylo by vhodné je implementovat. Bylo rozhodnuto, že mezi protokoly hodící se k implementaci do programu CrypTool jsou Dining Cryptographers, Coin Flipping, Zero-Knowledge, Yao´s Millionaire Problem a Oblivious Transfer.

Cíl této práce, to jest snažit se zvýšit povědomí o kryptografických protokolech, byl splněn. Všech pět protokolů bylo implementováno a připraveno pro použití. Těmito protokoly se práce podrobně zabývala, uvedla čtenáře do problematiky a byl v ní popsán jejich koncept. Je zde zároveň nastíněno, jak probíhala jejich implementace a jaké důvody vedly k tomu, proč byla implementace provedena zrovna tím daným způsobem. K implementaci každého protokolu byly dopracovány nezbytné prvky, které jsou důležité pro plné pochopení dotyčných protokolů.

Těmito prvky jsou šablona a dokumentace. Šablony zahrnují sestavené komponenty protokolů s vysvětlivkami, které objasňují, jak spadají jednotlivé komponenty do dotyčného protokolu.

Dokumentace byla vytvořena pro každou komponentu daného protokolu. V dokumentaci je popsáno, jak se s jednotlivými komponentami pracuje. Dokumentace také obsahuje úvod do problematiky protokolu.

Podařilo se navázat spojení s tvůrci programu CrypTool a nabídnout jim nové pluginy a šablony s kryptografickými protokoly. Tento cíl byl tedy rovněž splněn. Byl získán přístup do repositářů programu CrypTool. Implementované protokoly jsou zkoumány a testovány vývojáři programu CrypTool. Pravděpodobně tyto protokoly budou v nové stabilní verzi. Díky tomu se tato práce dostane ke všem zájemcům o kryptografii a rozšíří se vědomí o kryptografických protokolech.

I kdyby tyto protokoly nakonec ve stabilní verzi zpřístupněny nebyly, nalezne tato práce využití v ústavu inteligentních systémů, kde poslouží k snadnější výuce kryptografických protokolů.

Literatura

[1] AMIRBEKYAN, V., A. a ESTIVILL-CASTRO: Practical protocol for Yao's millionaires problem enables secure multi-party computation of metrics and efficient privacy-preserving k-NN for large data sets. *Knowledge and Information Systems*, ročník 21, č. 3, 2009: s. 327–363.

[2] BENE, T.: Kryptografické protokoly [online]. www.obluda.cz/iprednasky/15_proto.pdf, [cit. 2012-05-12].

[3] BLUM, M.: Coin Flipping by Telephone a Protocol for Solving Impossible Problems [online]. dm.ing.unibs.it/giuzzi/corsi/Support/papers-cryptography/Coin_flipping.pdf, [cit. 2012-05-07].

[4] CHAUM, D.: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability [online]. www.cs.cornell.edu/People/egs/herbivore/dcnets.html, [cit. 2012-05-07].

[5] DENNING, D.: Digital signatures with RSA and other public-key cryptosystems. *Communications of the ACM*, ročník 27, č. 4, 1984: s. 388–392.

[6] ESSLINGER, B.: CrypToolEin Open Source Projekt in der Praxis. *Datenschutz und Datensicherheit-DuD*, ročník 33, č. 3, 2009: s. 167–173.

[7] GOLDWASSER, S., MICALI, S. a RACKOFF, C.: The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, ACM, 1985, s. 291–304.

[8] JUELS, A.: Dining Cryptographers Revisited [online]. http://www.rsa.com/rsalabs/staff/bios/ajuels/publications/pdfs/dc-revisited.pdf, [cit. 2012-04-30].

[9] MAJI, H.; PRABHAKARAN, A., M. a SAHAI: On the Computational Complexity of Coin Flipping. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, IEEE, 2010, s. 613–622.

[10] MEADOWS, C.: Formal verification of cryptographic protocols: A survey. *Advances in Cryptology-ASIACRYPT'94*, 1995: s. 133–150.

[11] MENEZES, A. J.: *Handbook of applied cryptography*. Boca Raton: CRC Press, 1997, ISBN 08-493-8523-7, 780 s.

[12] MOHR A.: A Survey of Zero-Knowledge Proofs with Applications to Cryptography. Southern Illinois University at Carbondale Carbondale IL 62901.

[13] NEUŽILOVÁ, J. a. J. S.: Generalising Oblivious Transfers [online]. bezadis.ics.upjs.sk/old/CryptoSymposium/files/paper9.pdf, [cit. 2012-04-30].

[14] PRZYBYLSKI, S., A. WACKER, M. WANDER, F. ENKLER a P. VACEK: Plugin Developer Manual: How to build your own plugins for CrypTool 2.0. [online]. www.cryptool.org/trac/CrypTool2/browser/trunk/Documentation/PluginHowTo/HowToDeveloper.pdf, [cit. 2012-05-12].

[15] RABIN, M.: How to exchange secrets by oblivious transfer. Technická zpráva, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[16] RHEE, M.: *Internet security: cryptographic principles, algorithms and protocols.* Wiley, 2003, ISBN 04-708-5285-2.

[17] WWW Stránky: Cryptool Portal. www.cryptool.de, [cit. 2012-03-25].

[18] WWW Stránky: Solution to the Millionaire's Problem. www.proproco.co.uk/million.pdf, [cit. 2012-04-19].

[19] WWW Stránky: How to Contribute. www.cryptool.org/en/ct2-volunteer-en/288-how-to-contribute, [cit. 2012-04-30].

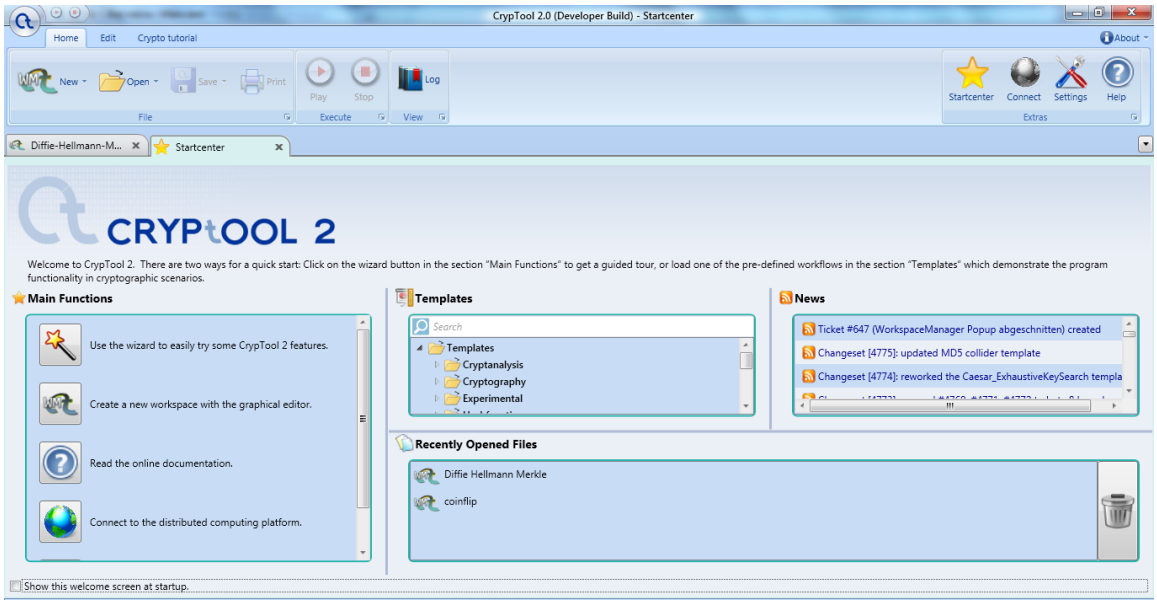
Příloha A

Obsah CD

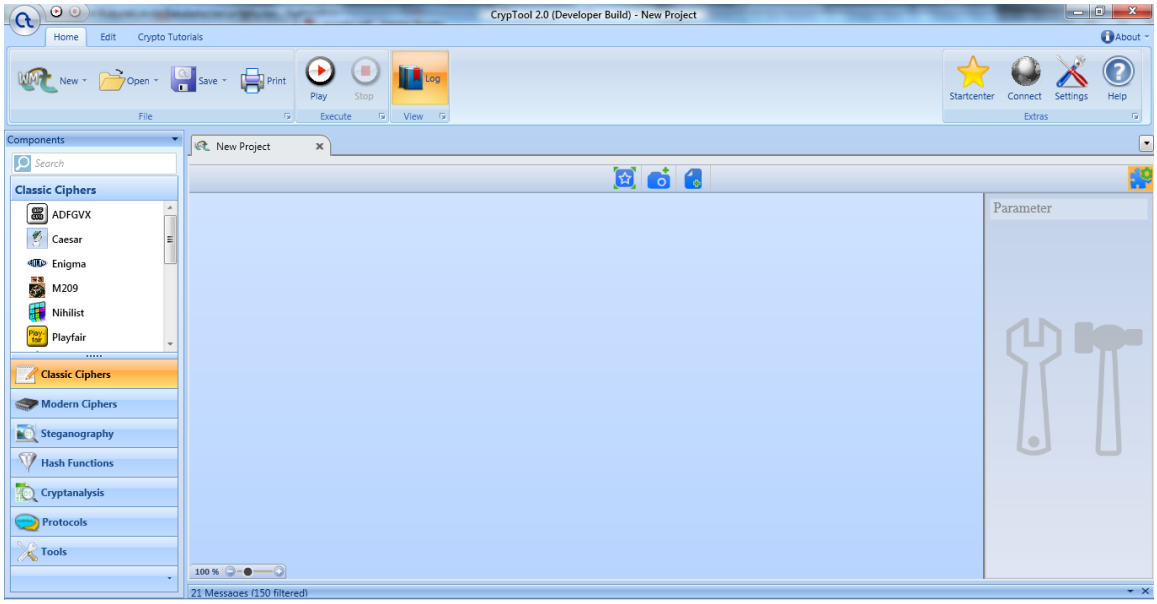
Cesta	Popis
`/cryptool/`	Zdrojové soubory programu CrypTool s kryptografickými protokoly
`/protocols/`	Zdrojové soubory pluginů jednotlivých kryptografických protokolů a šablony
`/tex/`	Zdrojové texty této práce
`/projekt.pdf`	PDF soubor s touto prací
`/manual.pdf`	Manul pro vývojáře CrypTool, je zde popsáno jak CrypTool přeložit a spustit

Příloha B

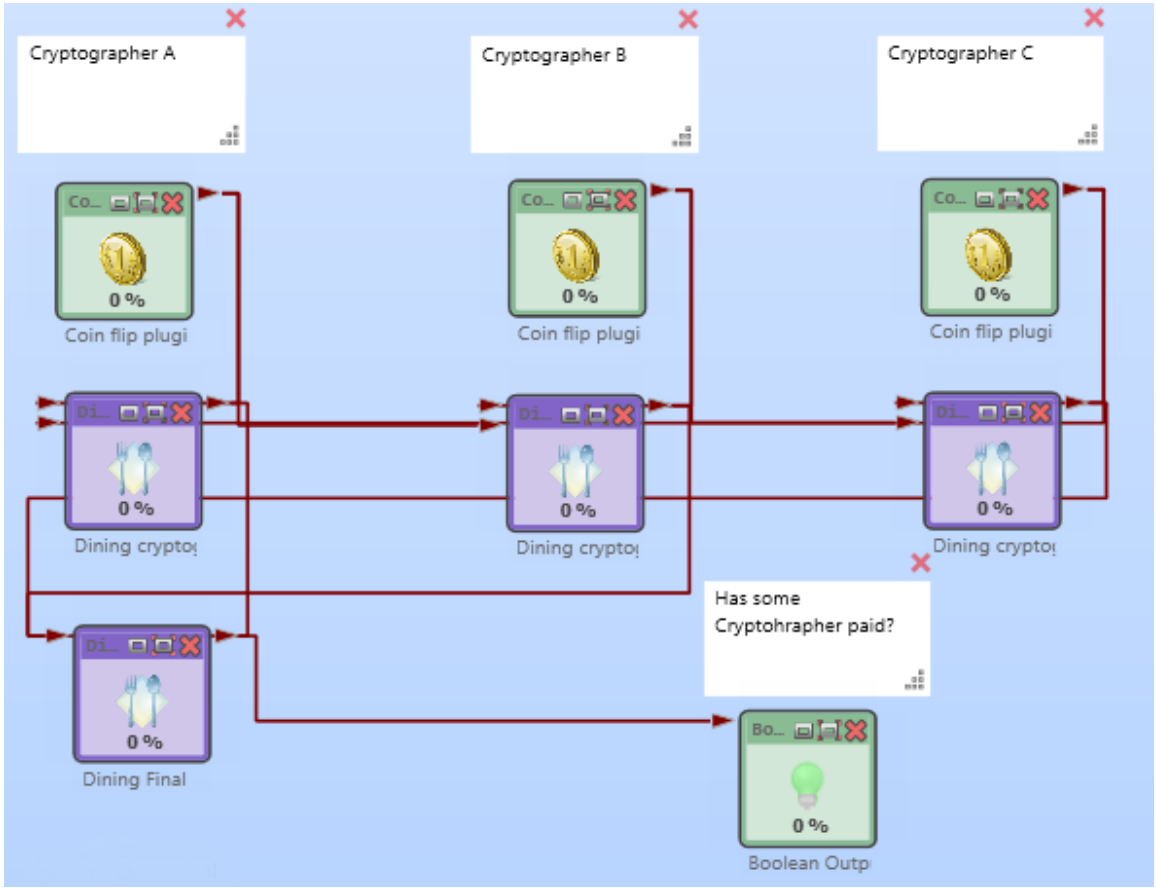
Screenshoty programu CrypTool



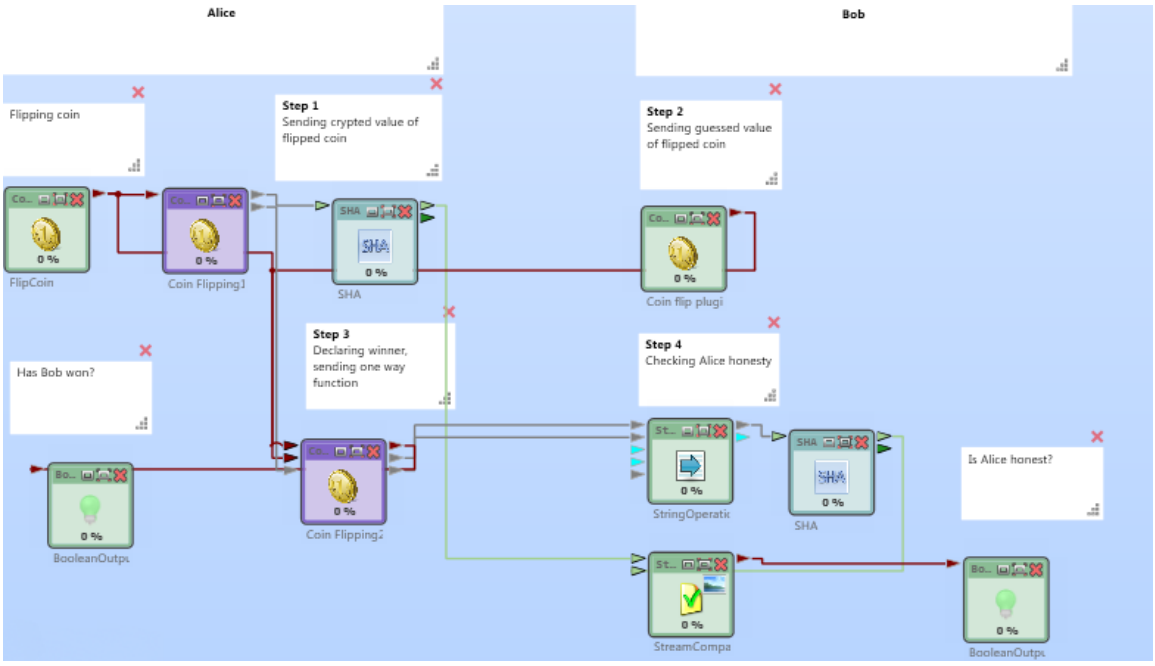
Obrázek B.1: Startcentrum



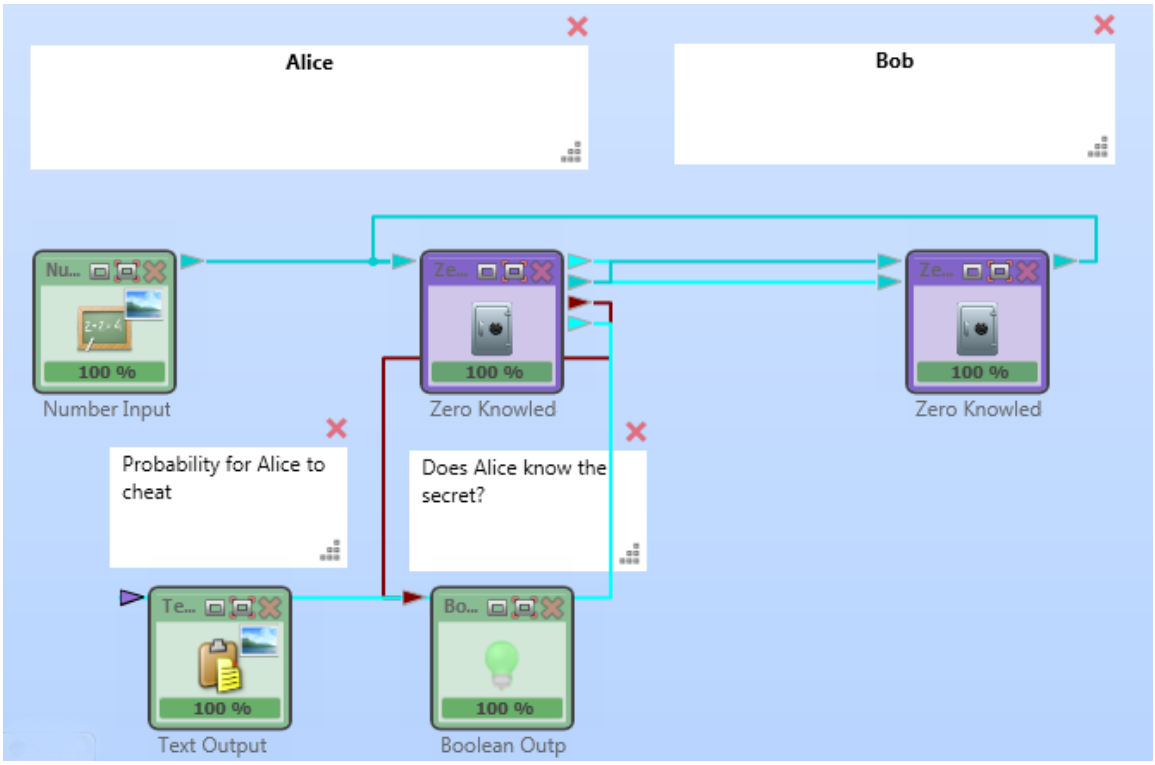
Obrázek B.2: Čistá pracovní plocha



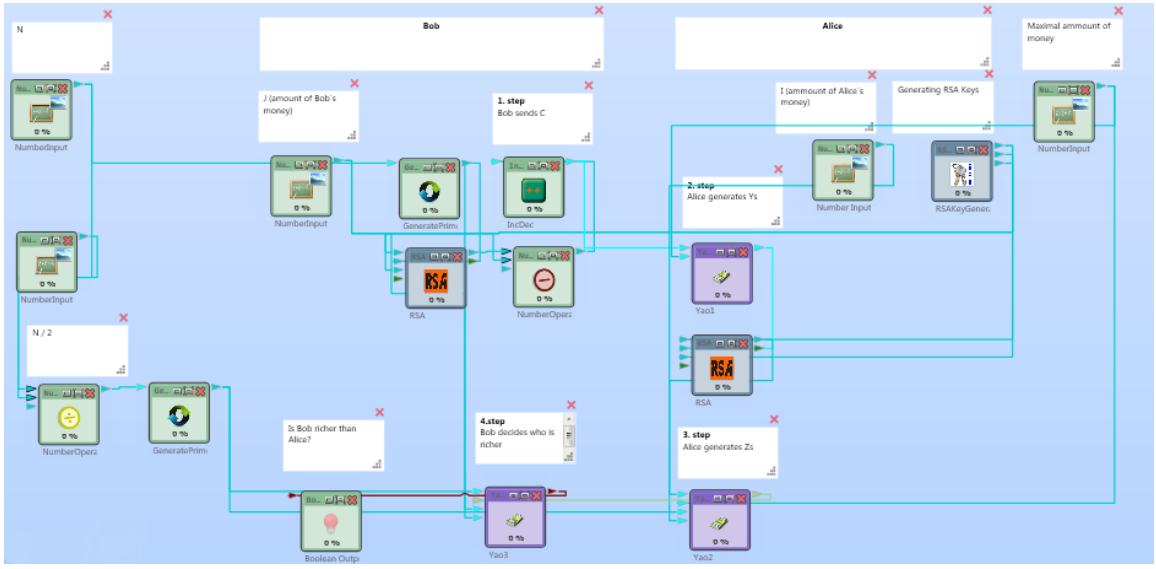
Obrázek B.3: Šablona Dining Cryptographers



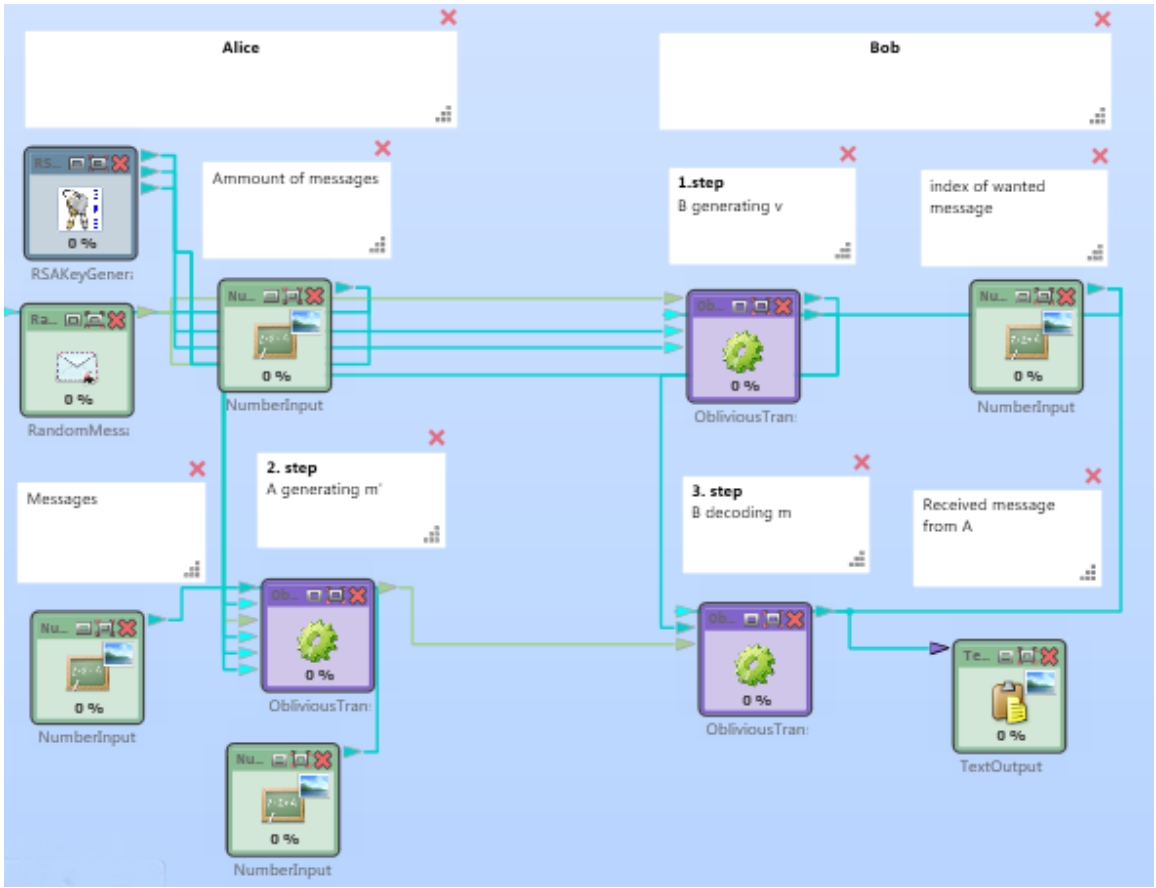
Obrázek B.4: Šablona Coin Flipping



Obrázek B.5: Šablona Zero-Knowledge



Obrázek B.6: Šablona Yao´s Millionaire problem



Obrázek B.7: Šablona Oblivious Transfer