View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ELTE Digital Institutional Repository (EDIT)

Eötvös Loránd Tudományegyetem

Informatikai Kar

Komputeralgebra Tanszék

Többrésztvevős biztonságos számítások

Témavezető
dr. Burcsi Péter
docens

Szerző
Fábián Gergely
Programtervező informatikus Bsc.

Budapest, 2018

Témabejelentő

Tartalomjegyzék

1 Bevezetés

A szakdolgozatom témájának a kriptográfiát, azon belül pedig a többrésztvevős biztonságos számításokat választottam. A terület legkorábbi és talán legismertebb példájára, a milliomos problémára, és annak egy variánsára, a szocialista milliomos problémára mutat egy megoldást a program. A kriptográfia mindig is érdekelt, de sajnos a BSc-s tanulmányaim során nem sok szó esett a témáról, ezért döntöttem úgy a szakdolgozatom keretében fogok foglalkozni vele.

2 Felhasználói dokumentáció

2.1 Többrésztvevős biztonságos számítások

A többrésztvevős biztonságos számítások [1] (secure multi-party computation), a kriptográfia egy részterülete, ahol a cél egy többparaméteres függvény értékének kiszámítása, úgy hogy az egyes résztvevők csak egy-egy paramétert ismernek, és azt nem kívánják a többi résztvevő előtt felfedni.

A többrésztvevős biztonságos számítások 1982-ben jelentek meg először amikor Andrew Yao bemutatta a milliomos problémát, aminek egy megvalósítását a szakdolgozat is tartalmazza. Eredetileg két résztvevős biztonságos számításoknak hívták (secure two-party computation) a területet, mivel a milliomos probléma két résztvevővel foglalkozik. Nem sokkal később általánosították a problémát, és így lett többrésztvevős.

A terület azt feltételezi, hogy a számításban résztvevő felek nem tudnak találni egy megbízható kívülállót találni, akivel megoszthatnák a titkaikat, és anélkül, hogy bárkivel megosztaná őket, kiszámolná az eredményt, és csak azt mondaná el az összes résztvevőnek.

Például legyen a három résztvevőnk Alice, Bob és Charlie (a kriptográfiában megszokott elnevezéseket használom a résztvevőkre), akik meg szeretnék tudni, hogy kinek van a legmagasabb fizetése, anélkül, hogy elmondanák egymásnak, hogy mennyit keresnek. A fizetésüket jelöljük ***x***, ***y***, ***z***-vel. Matematikailag formalizálva ki kell számolniuk a következőt:

$$F(x,y,z) = max(x,y,z)$$

Ha lenne egy megbízható kívülálló (mondjuk, hogy van egy Tony nevű közös barátjuk, akiről tudják mindhárman, hogy tud titkot tartani), el tudnák neki mondani a fizetésüket, ő kiszámolná a maximumát az értékeknek, és ezt a számot elmondaná nekik. A többrésztevős biztonságos számításoknak az a célja, hogy megadjon egy olyan protokollt, ahol a résztvevő felek csak egymással váltanak üzeneteket, mégis megtudják az ***F(x,y,y)*** függvény eredményét, anélkül, hogy felfednék ki mennyit keres, vagy egy kívülállóra, Tonyra, kéne támaszkodniuk. Semmivel több információhoz nem juthatnak a protokoll során, mintha együttműködnének egy lefizethetetlen, tökéletesen megbízható Tonyval.

Ebben az esetben a résztvevők annyi információhoz juthatnak amennyit a saját inputjukból, és a függvény eredményéből kitalálhatnak. [1]

2.2 Szocialista milliomos probléma

A szocialista milliomos probléma [2] a milliomos probléma egy egyszerűsített változata. A probléma arról szól, hogy két szocialista milliomos meg akarja tudni, hogy a vagyonuk egyenlő-e anélkül, hogy megmondanák egymásnak, hogy mennyi pénzük van.

A probléma megoldását gyakran használják kriptográfiai protokollként arra, hogy megbizonyosodjanak egy távoli kommunikációs partner személyazonosságáról, és kivédjék a *man-in-the-middle* támadásokat. A beszélgetés előtt megegyeznek egy jelszóval, és a szocialista milliomos problémával analóg módon a kommunikáció kezdetén összehasonlítják a jelszavakat, hogy egyeznek-e anélkül, hogy felfednének a jelszóból bármit is. [2]

2.2.1 Motiváció

Alice és Bob rendelkezik egy-egy titkos számmal, legyen ez a két szám $\boldsymbol{x}$ illetve $\boldsymbol{y}$. Alice és Bob azt szeretné megtudni, hogy az $\boldsymbol{x} = \boldsymbol{y}$ egyenlőség teljesül-e, úgy, hogy a másik fél ezen kívül semmi mást nem tud meg a másik számáról.

Egy passzív támadó, aki csak belehallgat a kommunikációs csatornába semmit nem tudhat meg az $\boldsymbol{x}$ és $\boldsymbol{y}$ számokról, még azt sem, hogy az egyenlőség teljesül-e.

Ha valamelyik fél nem őszinte, és eltér a protokolltól, akkor sem tudhat meg több információt annál, hogy teljesül-e az egyenlőség.

Egy aktív támadó, aki képes tetszése szerint belenyúlni Alice és Bob üzenetváltásába (tehát man-in-the-middle támadást hajt végre), nem szerezhet több információt, mint egy passzív támadó, illetve nem tudja befolyásolni a protokoll végkimenetelét azon kívül, hogy meghiúsítja azt.

Tehát a protokoll alkalmas arra, hogy két a kommunikáló fél rendelkezik-e ugyanazzal a titkos információval. [2]

2.3 Milliomos probléma

A milliomos probléma [3] (Yao's millionaires' problem) egy klasszikus többrésztvevős számítási probléma, ami arról szól, hogy két milliomos meg szeretné tudni, hogy melyikük a gazdagabb, úgy, hogy nem fedik fel egymás előtt a vagyonuk méretét. A problémát Andrew Yao mutatta be 1982-ben, megteremtve ezzel a témakör alapjait.

A probléma analóg a következő, sokkal általánosabb problémával: van két számunk $\boldsymbol{a}$ és $\boldsymbol{b}$ és az a cél, hogy megoldjuk az $\boldsymbol{a \geq b}$ egyenlőtlenséget $\boldsymbol{a}$ és $\boldsymbol{b}$ értékének felfedése nélkül.

A kriptográfiában ez egy nagyon fontos probléma, aminek a megoldása használt az e-kereskedelemben, illetve az adatbányászatban. A kereskedelmi alkalmazásokban időnként össze kell hasonlítani titkosított számokat biztonságosan. [3]

2.4 Célközönség

A különböző protokollok különböző problémákat hivatottak megoldani, a szocialista milliomos protokoll egy titkos számmal történő autentikációra használható, a milliomos protokoll adatbányászathoz, vagy különböző üzleti alkalmazásokhoz hasznos.

2.5 Rendszerkövetelmények

2.5.1 Hardver

Dual-core 1GHz vagy annál gyorsabb processzor, 2 GB RAM.

2.5.2 Szoftver

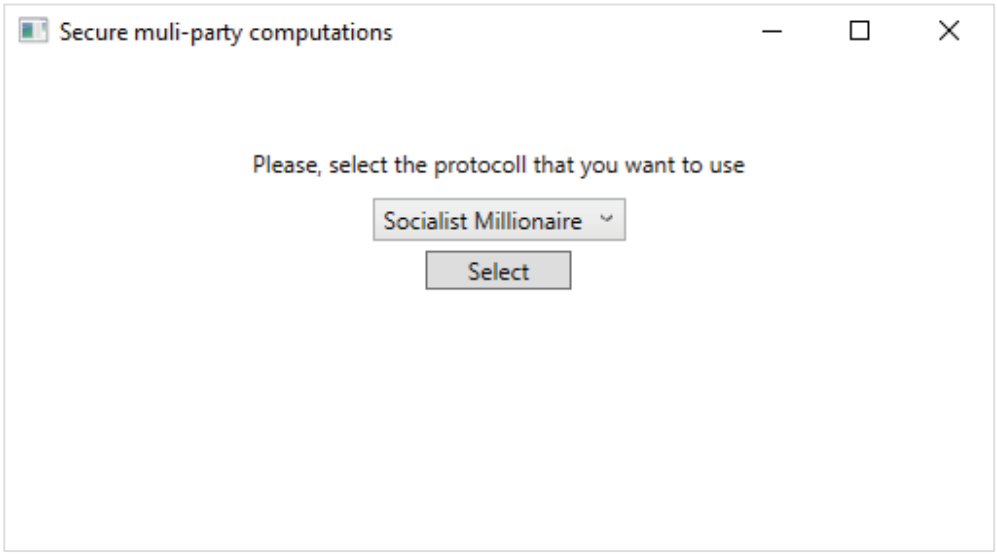
Windows 7, vagy annál újabb operációs rendszer. Ezen kívül telepítve kell, hogy legyen a Microsoft .NET Framework 4.5.2-es verziója.

2.6 Telepítés

A lemezen található *mpc.zip* tömörített állomány kicsomagolása után az *mpc.exe*-vel futtatható az alkalmazás. A *zip* állomány emellett tartalmaz példa konfigurációs fájlokat is, hogy rögtön működőképes legyen a program.

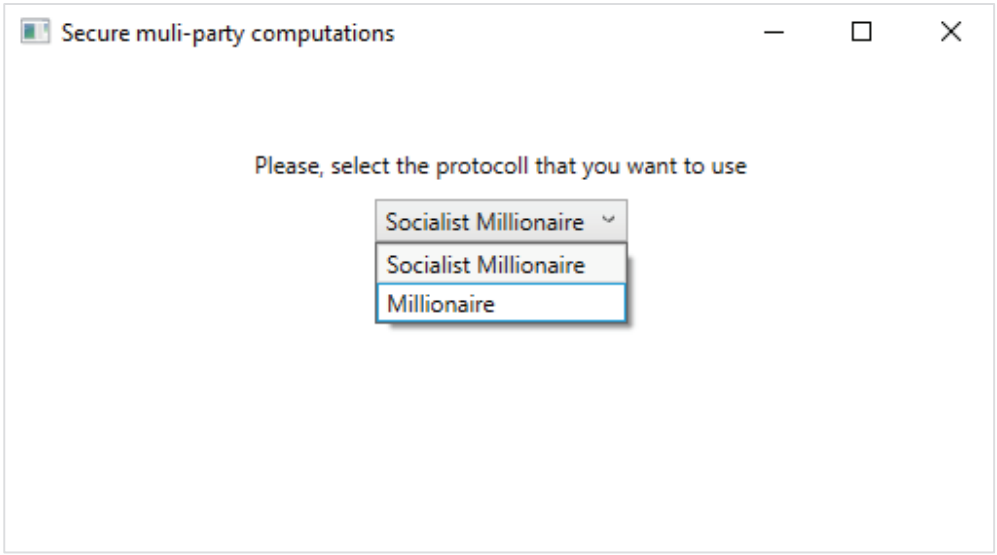
2.7 Használati útmutató

Az alkalmazás indítása után a *Secure multi-party computations* című ablak jelenik meg, ahol kiválasztható hogy melyik protokollt szeretnénk használni.



1. ábra: Kezdőképernyő

A legördülő menüből válogathatunk a protokollok közül. Alapértelmezetten a szocialista milliomos van kiválasztva, ha ezt szeretnénk használni, akkor semmi teendőnk. Az összes többit megtaláljuk a menüben.

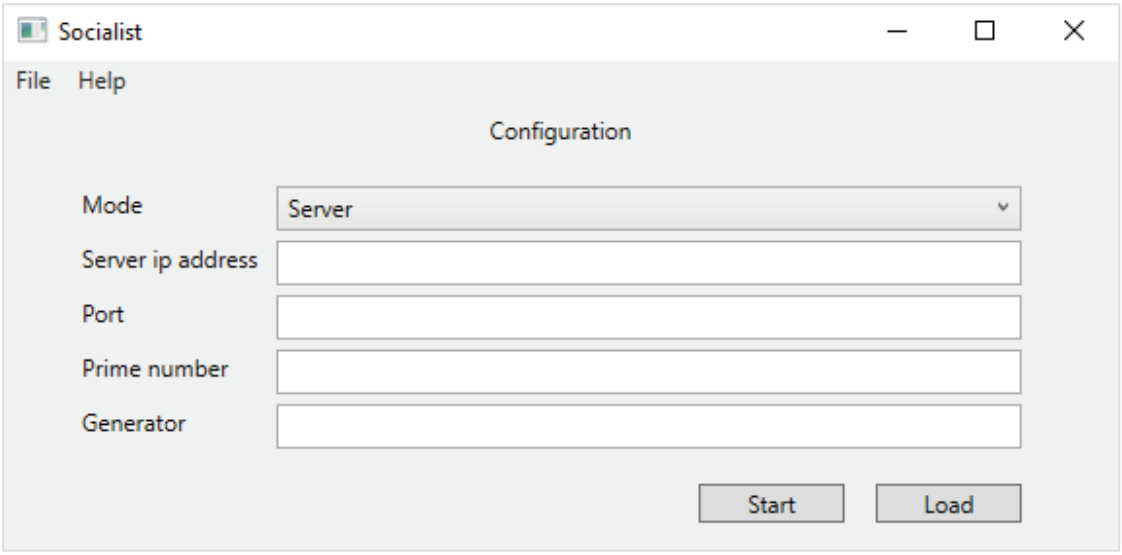


2. ábra: Legördülő menü

Miután kiválasztottuk a nekünk megfelelő opciót, a ***select*** feliratú gombra kattintva haladhatunk tovább.

2.7.1 Szocialista milliomos protokoll

Ha ezt a protokollt szeretnénk használni, először fel kell konfigurálnunk a megfelelő adatokkal. A megjelenő ablak ezt hivatott elősegíteni.



3. ábra: A szocialista milliomosprotokoll konfigurációs ablaka

A mezőket kézzel is kitölthetjük, vagy a ***load*** gombra kattintva egy előre megírt konfigurációs fájlból is betölthetjük a paramétereket. A konfigurációs fájlal szemben a következő alak az elvárás:

mode:<mód>
server_ip:<szerver ip címe>
port:<portszám>
prime:<prímszám>
generator:<generátor elem>

A ***mode*** paraméter azt határozza meg, hogy az általunk indított program szerver vagy kliens módban fusson. Ahhoz, hogy a protokoll sikeresen menjen végbe a két fél között, az egyik félnek szerver módban kell futnia, a másiknak kliens módban. A végeredmény szempontjából semmilyen különbség sincs a két fél között. A szocialista milliomos protokoll

aszimmetriája és a tcp aszimmetriája miatt van szükség a két fél ilyesfajta megkülönböztetésére. Hogy ez pontosan mit takar, azt a fejlesztői dokumentációban részletesebben is tárgyalom, a használat szempontjából mindössze annyi a fontos, hogy a két kommunikáló fél egyike szerver, másik kliens módban fusson.

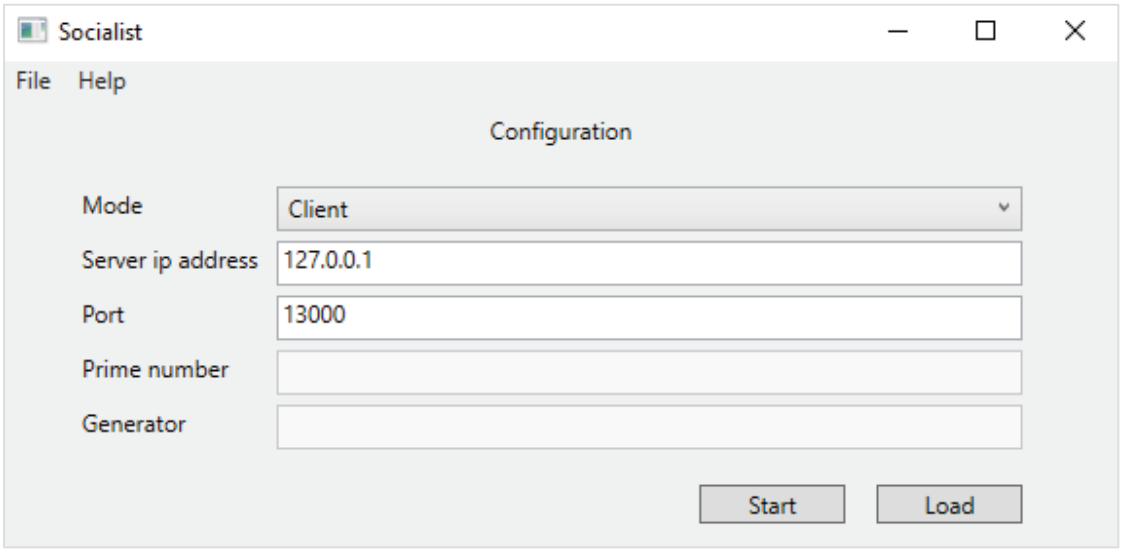
A ***server_ip*** a szerverként futó alkalmazás IP címe kell, hogy legyen mindkét félnél.

A ***port*** paraméterrel tudja a szerver beállítani, hogy melyik porton fogja várni a kliens fél csatlakozását. Ezt a számot közölnie kell a klienssel, mert csak akkor lesz sikeres a csatlakozása a szerverhez, ha azon a porton próbál csatlakozni, amit a szerver meghatározott.

A ***prime*** mezőben egy publikus kulcsot kell megadni. A biztonságos kommunikáció érdekében a kulcsnak egy minél nagyobb prímszámnak kell lennie. A programhoz mellékelt példa konfigurációs fájl egy 1536 bit hosszú prímszám szerepel. Ezen dokumentum írásakor ez elég hosszú, ahhoz, hogy a biztonságosan tudjon működni a protokoll, de fontos megjegyezni, hogy ahogy az idő halad és a számítógépek teljesítménye nő egyre nagyobb prímszámokra lesz szükség. Ez a szám egy publikus kulcs, így semmivel nem gyengíti a protokollt, ha harmadik fél megtudja ezt a számot, mivel csak titkosításra lehet használni, dekódolásra nem.

A ***generator*** paraméter a $(\mathrm{Z} / \mathrm{Z}_{prime}) *$ multiplikatív csoport egy generátor eleme. A generátor elemnél az a fontos, hogy minél nagyobb legyen a rendszáma. Ez a szám is publikus, tehát a titokban tartásától nem függ a protokoll biztonsága.

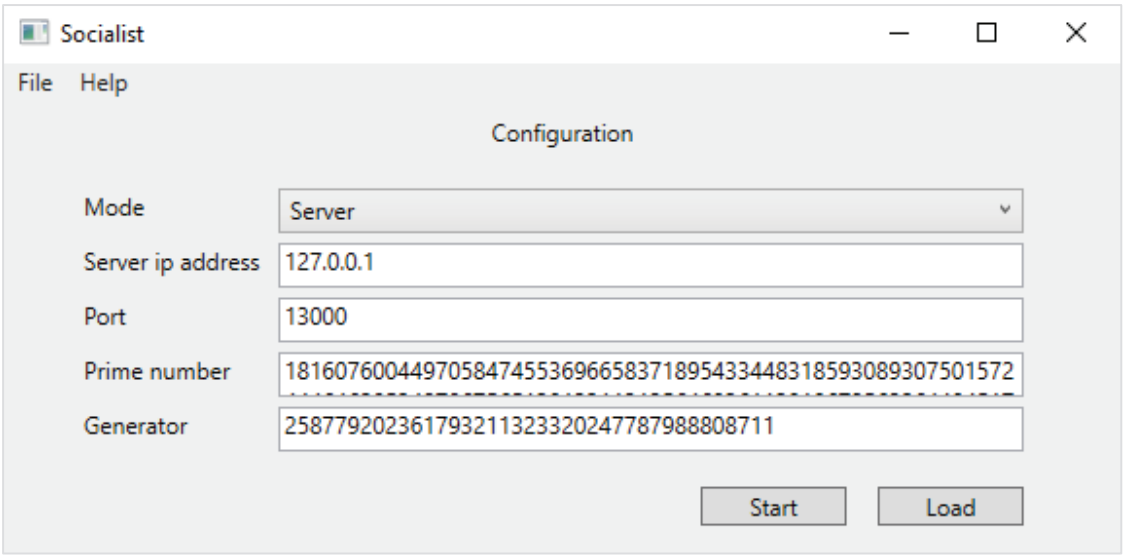
A prímszám, illetve a generátor elem nem fontos, hogy be legyen állítva a kliensél, mivel a csatlakozás után a szerver átküldi a kliensnek a számokat, és ezután a szerver értékeit fogják használni a kommunikáció további részében.



4. ábra: A kliens által egy teljesen jól kitöltött konfigurációs ablak

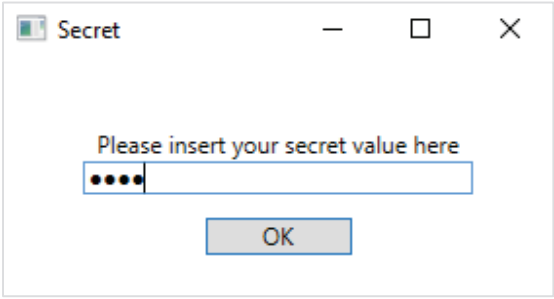
Ahogy a 4. ábrán látszik is, amennyiben a mód opciónál klienst választunk, a prímszámot és a generátor elemet tartalmazó szövegdoboz kiszürkül és használhatatlanná válik, mivel ilyenkor semmi jelentősége sincs, annak, hogy mit írunk ebbe a két mezőbe, ha üresen hagyjuk úgy is minden rendben fog működni.

A számunkra megfelelő konfiguráció beállítása után a start gombbal lehet elindítani a protokollt.



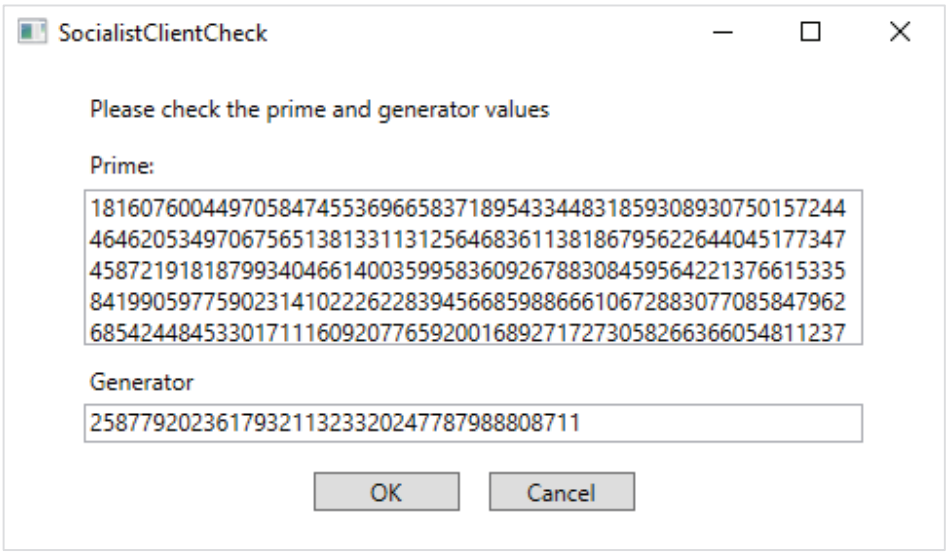
5. ábra: Egy teljesen kitöltött szerver konfiguráció

Az indítás után a program bekéri a titkot (a számot, amit össze akarunk hasonlítani). Miután ezt megadtuk a szerver esetében elkezd várni, hogy egy kliens csatlakozzon, a kliens pedig megkísérli a csatlakozást a szerverhez.



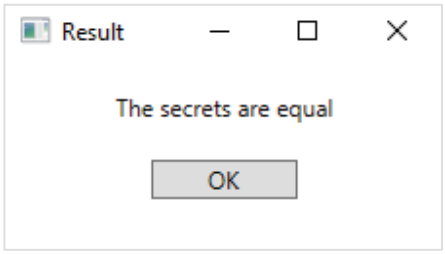
6. ábra: Titok bekérése

A csatlakozás után, amikor a kliens megkapja a prímet és a generátor elemet, van lehetősége ellenőrizni őket, és csak akkor folytatódik a protokoll, ha elfogadta. Ez azért fontos, mert, ha a szerver szándékosan gyenge prímet ad meg akkor nem lesz elég biztonságos az üzenetváltás. A számok szövegdobozban jelennek meg, hogy könnyen ki lehessen másolni őket.



7. ábra: A kliens ebben az ablakban látja a szerver által küldött számokat

Miután a kliens elfogadta a szerver által küldött számokat lezajlik a protokoll, és a program mindkét félnél kiírja a végeredményt.



8. ábra: Eredmény képernyő

Ezután az ***ok*** gomb megnyomása után visszakerülünk a kezdő képernyőre.

2.7.2 Milliomos protokoll

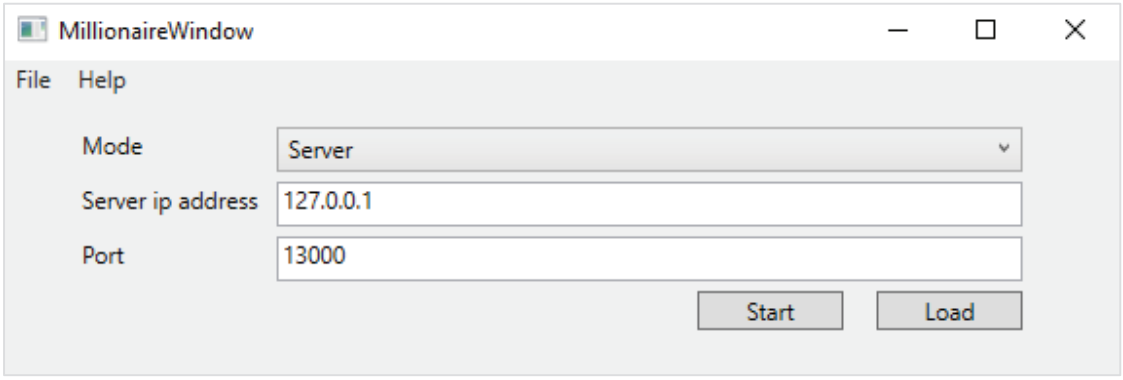
A protokoll kiválasztása után a konfigurációs ablakba kerülünk, ahol beállíthatjuk, hogy szerver, vagy kliens módban szeretnék futtatni az alkalmazást, illetve a csatlakozási adatokat adhatjuk meg.

A beállítandó paraméterek a következők:

Mode: itt lehet kiválasztani, hogy az általunk indított alkalmazás szerverként vagy kliensként működjön a kommunikáció során. Az egyik résztvevőnek szervernek kell lennie, a másiknak pedig kliensnek. Ez a hálózati kommunikáció és a protokoll aszimmetriája miatt szükséges.

Server ip: a szerverként futó alkalmazás IP címe. A szerver itt adja meg, hogy hol fog hosztolni, kliens pedig itt tudja beállítani, hogy hova csatlakozzon.

Port: egy szabad port, ahol a szerver várja a kliens csatlakozását. Ugyanaz a szám kell, hogy beírva legyen a kliensnél és a szervernél is.

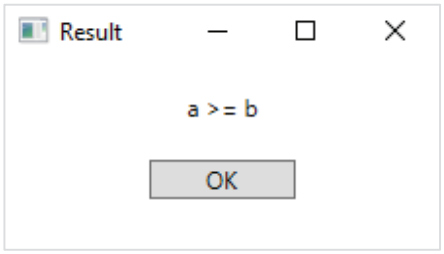


9. ábra: A milliomos protokoll kitöltött konfigurációs ablaka

Miután beállítottunk mindent megfelelően a ***start*** gombra tudjuk elindítani a protokollt.

Ezután a program bekéri a titkot (az összehasonlítani kívánt számot). Miután ezt megadtuk a szerver elkezd várni a kliens csatlakozására, a kliens pedig megpróbál csatlakozni a szerverhez.

A csatlakozás után elindul a protokoll, innentől a felhasználónak semmi tennivalója, azon kívül, hogy megvárja, hogy befejeződjenek a számítások, és a program kiírja a végeredményt.



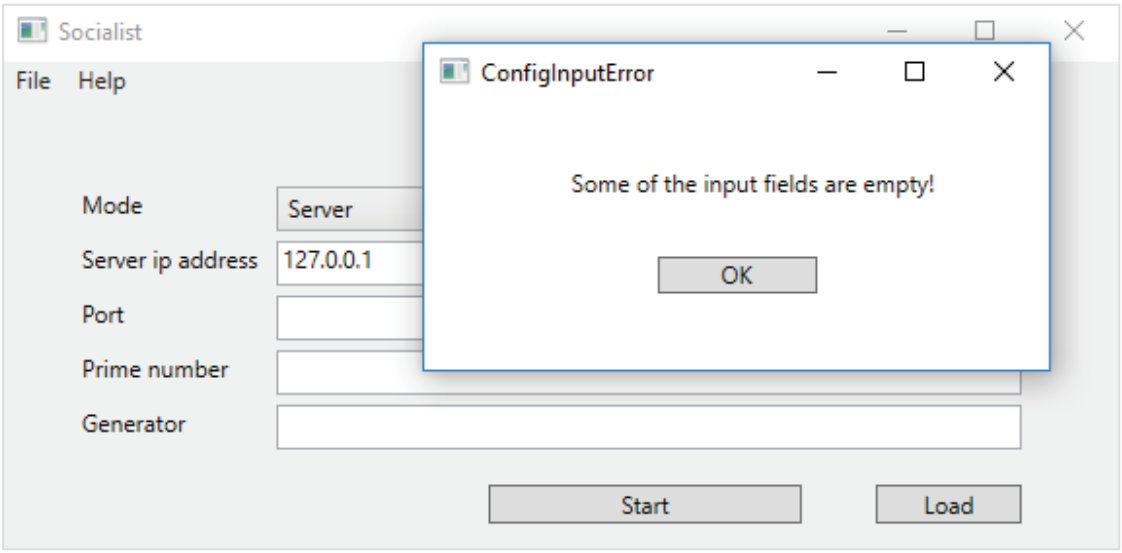
10. ábra: Eredmény képernyő

Ezután az ***ok*** gomb megnyomása után visszakerülünk a kezdőképernyőre.

2.8 Hibaüzenetek

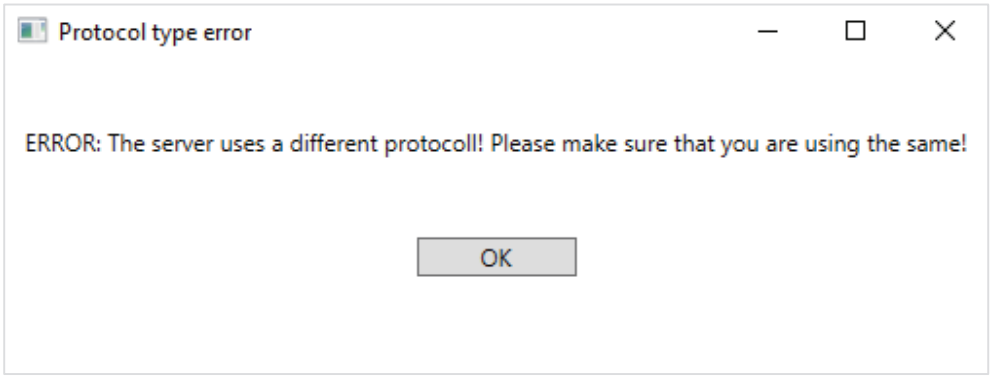
Ebben a részben a lehetséges előforduló hibaüzenetekről és megoldásukról lesz szó.

Ha bármelyik protokoll konfigurációs ablakában nem töltjük ki az összes mezőt, de megnyomjuk a **start** gombot, akkor egy felugró ablakban hibaüzenetet kapunk. Ez azért van, mert az összes használható mező kitöltése kötelező (némelyik protokollnál vannak olyan mezők, amiket a kliensnek nem szükséges kitölteni, de ezek a mezők jól láthatóan kiszürkülnek ha a kliens módot választjuk ki, és a használati útmutatóban ezek a mezők külön meg vannak említve).



1. ábra: Ha nem töltünk ki minden mezőt, ezt a hibaüzenet fog megjelenni

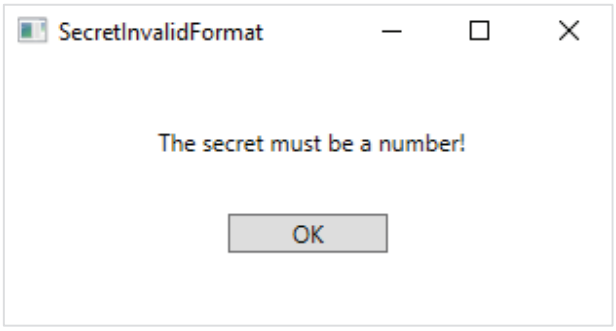
Hogyha a szerverre egy másik protokollra beállított kliens csatlakozik akkor mindkét fél kap egy hibaüzenetet, mivel csak két azonos protokollt használó alkalmazás tud együtt működni.



2. ábra: Így néz ki a kliensoldali hibaüzenet, ha nem egyezik a protokoll

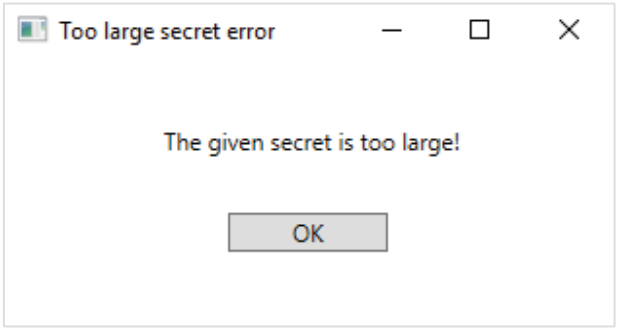
Amikor megadjuk a titkunkat, akkor is többféle hibaüzenetet kaphatunk, hogyha valamit elrontunk.

Hogyha elgépelünk valamit, és a beírt érték nem egy szabályos szám lesz (például 123 helyett 123w-t ütünk be), akkor hibát fogunk kapni, mivel a titoknak egy számnak kell lennie.



3. ábra: Hogyha nem szabályos formátumú számot ütünk be, ezt a hibát fogjuk kapni

Az is hibát eredményez a titok megadásánál, hogyha túl nagy számot gépelünk be. Az általunk megadott számnak kisebbnek kell lennie, mint 2^{32}. Amennyiben ennél nagyobb számot próbálunk megadni hibát kapunk.



4.ábra: Túl nagy szám esetén ezt a hibát kapjuk

Konfigurációs fájl beolvasásakor és mentésekor is előfordulhat, hogy hibát kapunk. Általában ez amiatt lehet, hogy nem megfelelő formátumú fájlt próbálunk betölteni. Mentéskor az idézheti elő a problémát, hogy nincs írási jogunk a megadott mappában, emiatt máshova kell menteni a fájlt.

Az is előfordulhat, hogy megszakad a kapcsolat kommunikáció közben a két fél között, ekkor kiír valami hálózati hibát. Ilyenkor győződjünk meg róla, hogy csatlakozva vagyunk az internethez, majd indítsuk újra a programot, és próbáljuk meg még egyszer hátha ezúttal nem megy el az internet.

3 Fejlesztői dokumentáció

3.1 Követelményelemzés

3.1.1 Feladat leírása

A programnak két kriptográfiai protokollt kell tudnia megvalósítania. Mindkét protokoll a többrészvevős biztonságos számítások témaköréből kerül ki, mindegyik két résztvevős, tehát az alkalmazásnak képesnek kell lennie hálózatban csatlakoznia más kliensekhez. Mindezt úgy, hogy a felhasználók adatai végig biztonságban vannak.

A szocialista milliomos protokoll feladata megmondani, hogy a két résztvevő titkos számai egyenlőek-e, mindezt úgy, hogy egyikük se tud meg semmit a másiknál lévő számról, és egy passzív – hallgatózó – illetve egy aktív, a kommunikációs csatorna közepébe beékelődött támadó se tud meg semmit a két felhasználó titkáról.

A milliomos protokoll nagyon hasonló az előzőhöz, annyi különbséggel, hogy itt nem csak egyenlőség vizsgálatra van szükség, hanem meg kell tudni mondania, hogy melyik szám a nagyobb. A biztonsági előírások teljesen megegyeznek az előző feladatéval.

A programnak rendelkeznie kell egy egyszerű felhasználói felülettel, hogy kényelmesebb legyen a használata.

Emellett biztosítani kell lehetőséget arra, hogy a felhasználó képes legyen elmenteni, illetve betölteni a beállításait.

3.1.2 Felhasználói történetek

Felhasználói történetek segítségével fejtsük ki egy kicsit részletesebben a funkciókat.

Nézzük meg a szocialista milliomos protokoll egy felhasználási módját egy felhasználói történeten keresztük:

Mint felhasználó, *szeretném* összehasonlítani a nálam lévő titkos kulcsot, *azért, hogy* megbizonyosodjak róla, hogy tényleg ő az, akivel kommunikálni szeretnék

1. *Amennyiben* a kezdőképernyőn vagyok,
   ha kiválasztom a szocialista milliomos protokollt,
   akkor az alkalmazás tovább lép a konfigurációs ablakra.
2. *Amennyiben* átkerültem konfigurációs ablakra,
   ha a ***load*** gombra kattintva betöltöm az adatokat a konfigurációs fájlból,
   akkor megjelennek az általam várt adatok a képernyőn.
3. *Amennyiben* megjelentek az általam várt adatok a képernyőn,
   ha megnyomom a ***start*** gombot,
   akkor a program bekéri tőlem titkos kulcsot.
4. *Amennyiben* a program bekéri tőlem a titkos kulcsot,
   ha beírom,
   akkor elindul a protokoll.
5. *Amennyiben* elindul a protokoll,
   ha megvárom amíg befejeződik,
   akkor a program kiírja a végeredményt.
6. *Amennyiben* a program kiírja a végeredményt,
   ha az ***ok*** gombra kattintok,
   akkor visszakerülök a kezdőképernyőre.

Ebből látszik nagyjából a program struktúrája egy átlagos használat során, a kezdőképernyőn kiválasztjuk, hogy melyik funkciót akarjuk használni, utána jönnek a beállítások, amiket beírhatunk kézzel, vagy betölthetünk egy már létező config fájlt, majd indítás után kell beírni a titkot (azért olyankor, mert titkos adatot nem biztonságos egy config fájlban tárolni, ezért tartom fontosnak elkülöníteni). Ezután végbemegy a protokoll, kiíródik az eredmény, majd annak nyugtázása után visszakerülünk a kezdő képernyőre.

A következő példán keresztül a programon belül történő navigálást próbálom bemutatni, aztán a használatot.

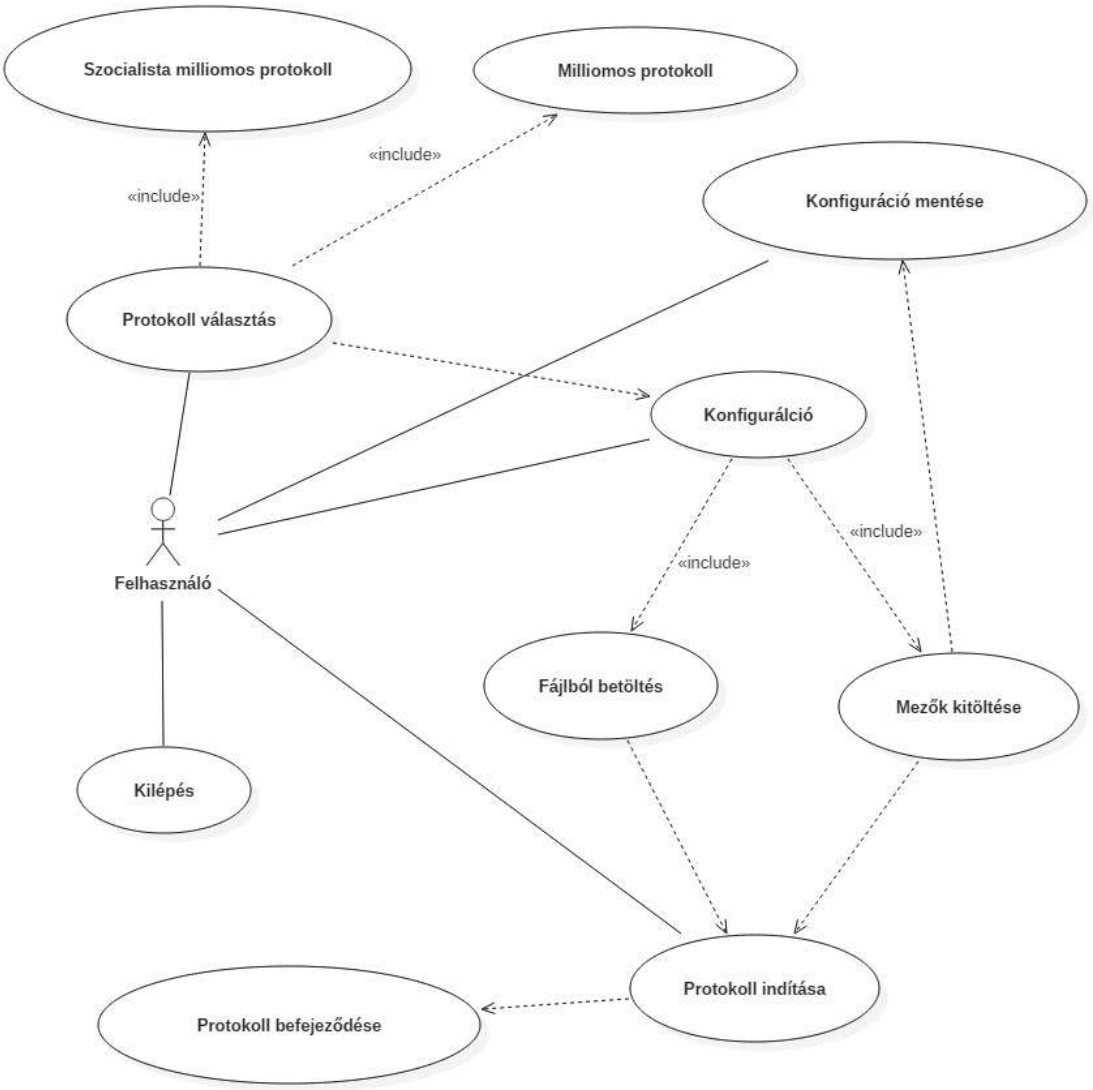
Mint felhasználó szeretném összehasonlítani a fizetésem a kollégámmal (de nem akarjuk közölni egymással konkrét számokat), *azért, hogy* megbizonyosodjunk róla, hogy melyikünk munkája értékesebb a cégnek.

1. *Amennyiben* a kezdőképernyőn vagyok,
   ha kiválasztom a szocialista milliomos protokollt,
   akkor az alkalmazás tovább lép a konfigurációs ablakra.
2. *Amennyiben* a konfigurációs ablakon vagyok,
   ha megnyomom a ***help*** menüpontot,
   akkor megjelenik egy rövid leírás a protokollról.
3. *Amennyiben* megjelent a rövid leírás, és rájövök, hogy nekem a milliomos protokoll kell,
   ha megnyomom a protokoll választás menüpontot,
   akkor visszakerülök a kezdőképernyőre.
4. *Amennyiben* a kezdőképernyőn vagyok,
   ha kiválasztom a milliomos protokollt,
   akkor a program tovább lép a konfigurációs ablakba.
5. *Amennyiben* a konfigurációs ablakon vagyok,
   ha kitöltöm az összes mezőt,
   akkor indításra kész a protokoll.
6. *Amennyiben* indításra kész a protokoll,
   ha megnyomom a menüben a ***save config*** pontot,
   akkor elmenti a beállításaimat, és később be tudom tölteni.
7. *Amennyiben* elmentettem a beállításaimat és indításra kész a protokoll,
   ha megnyomom a ***start*** gombot,
   akkor a program bekéri tőlem a fizetésemet.
8. *Amennyiben* a program bekéri tőlem a fizetésemet,
   ha megadom neki,
   akkor a program elkezdi a kommunikációt a másik féllel.
9. *Amennyiben* a program elkezdte a kommunikációt a másik féllel,
   ha megvárom amíg végez,
   akkor kiírja a képernyőre a végeredményt.
10. *Amennyiben* a program kiírta a végeredményt,
    ha az ***ok*** gombra kattintok,
    akkor a visszakerülök a kezdőképernyőre.

Ebből látszik, hogy tud a felhasználó körbe nézni a programban, és hol kaphat valaki segítséget, ha elakadt, vagy nem biztos a dolgában. Illetve látszik, hogy elvárás a programmal szemben, hogy képes legyen elmenteni a felhasználó által beírt beállításokat egy config fájlba, hogy a következő alkalommal egy kattintással be lehessen tölteni mindent (a titkon kívül, amit biztonsági okokból nem szabad a beállítások között tárolni).

3.1.3 Használati esetek

A használati eset diagrammal mutatjuk be, hogy miként működik együtt a rendszer a felhasználóval.



3.2 Felhasznált algoritmusok

3.2.1 Szocialista milliomos probléma megoldása

A problémára több megoldás is létezik, ebben a konkrét esetben az Off-the-Record Messaging Protcolt [5] használom.

A protokoll elkezdése előtt a két fél megegyezik egy $\boldsymbol{p}$ prímben, és egy $\boldsymbol{h}$ számban, ami bármelyik nem egység eleme a $(\boldsymbol{Z}/_{\boldsymbol{p}}\boldsymbol{Z})^*$ multiplikatív csoportnak. Célszerű minél nagyobb prímszámot választani a kommunikáció biztonsága érdekében. A példa fájlban egy 1536 bites prímszám van, az ajánlott méret legalább ekkora, de a nagyobb sem árt. A $\boldsymbol{h}$ számnak is érdemes egy minél nagyobb rendű generátor elemet megválasztani. Érdemes még megjegyezni hogy minden számítást modulo $\boldsymbol{p}$-ben értelmezünk, tehát a $(\boldsymbol{Z}/_{\boldsymbol{p}}\boldsymbol{Z})^*$ multiplikatív csoporton belül.

A $\langle h|a,b\rangle$-vel fogjuk jelölni a az $\boldsymbol{a}$ és $\boldsymbol{b}$ számok *Diffie-Helman-Merkle [6]* kulcscseréjét, ami $\boldsymbol{h^{ab}}$-t eredményül mindkét félnek:

- Alice kiszámolja $\boldsymbol{h^a}$-t, majd elküldi Bobnak, aki kiszámolja $\boldsymbol{(h^a)^b = h^{ab}}$-t.
- Bob kiszámolja $\boldsymbol{h^b}$-t, majd elküldi Alicenak, aki kiszámolja $\boldsymbol{(h^b)^a = h^{ba}}$-t.

$\boldsymbol{h^{ab} = h^{ba}}$, mivel a szorzás asszociatív $(\boldsymbol{Z}/_{\boldsymbol{p}}\boldsymbol{Z})^*$-ban.

A protokoll biztonságának egy részre véletlen generált titkokon alapszik, amikhez fontos a megfelelő kriptográfiailag biztonságos véletlenszám generátor használata. Emellett figyelni kell arra, hogy a protokoll sérülékeny arra, hogyha valamelyik fél szándékosan valamelyik véletlenszámot nullának választja meg (később látni fogjuk hogy ezeket a számokat $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$-val jelöljük). Emiatt a *Diffie-Helman* csere közben meg kell vizsgálni, hogy a kapott $\boldsymbol{h^a}$, $\boldsymbol{h^b}$, $\boldsymbol{h^\alpha}$, $\boldsymbol{h^\beta}$ valamelyike egy-e (ha igen, akkor azonnal meg kell szakítani a protokollt, mert nem biztonságos). Ezen felül teljesülnie kell annak, hogy $\boldsymbol{P_a \neq P_b}$ és $\boldsymbol{Q_a \neq Q_b}$.

	Alice	Csatorna	Bob
1	Titok x Véletlen generált: a, α, r	Publikus p, h	Titok: y Véletlen generált: b, β, s
2		Biztonságos csere: g = $\langle h\|a,b\rangle$	

3		Biztonságos csere: $\gamma = \langle h \mid \alpha, \beta \rangle$	
4	Ellenőrizni kell, hogy $h^b \neq 1$ és $h^\beta \neq 1$		Ellenőrizni kell, hogy $h^a \neq 1$ és $h^\alpha \neq 1$
5	$P_a = \gamma^r$ $Q_a = h^r g^x$		$P_b = \gamma^s$ $Q_b = h^s g^y$
6		Nem biztonságos csere: P_a, Q_a, P_b, Q_b	
7		Biztonságos csere: $g = \langle Q_a Q_b^{-1} \mid \alpha, \beta \rangle$	
8	Ellenőrizni kell, hogy $P_a \neq P_b$ és $Q_a \neq Q_b$		Ellenőrizni kell, hogy $P_a \neq P_b$ és $Q_a \neq Q_b$
9	Meg kell vizsgálni: $c = P_a P_b^{-1}$		Meg kell vizsgálni: $c = P_a P_b^{-1}$

Tudjuk, hogy:

$$P_a P_b^{-1} = \gamma^r \gamma^{-s} = \gamma^{r-s} = h^{\alpha\beta(r-s)}$$

Tehát:

$$c = (Q_a Q_b^{-1})^{\alpha\beta}$$

$$= (h^r g^x h^{-s} g^{-y})^{\alpha\beta} = (h^{r-s} r^{x-y})^{\alpha\beta}$$

$$= (h^{r-s} h^{ab(x-y)})^{\alpha\beta} = h^{\alpha\beta(r-s)} h^{\alpha\beta ab(x-y)}$$

$$= (P_a P_b^{-1} h^{\alpha\beta ab(x-y)})$$

Mivel a véletlenszerűen választott értékeket mindkét fél titokban tartja a másik előtt, ezért egyikük sem tudja elérni, hogy $P_a P_b^{-1}$ és c egyenlő legyen, kivéve, ha x és y megegyeznek. Ekkor ugyanis $h^{\alpha\beta ab(x-y)} = h^0 = 1$, tehát $c = P_a P_b^{-1}$.

Emiatt, ha a számítások végén a $c = P_a P_b^{-1}$ egyenlőség teljesül, akkor az x = y egyenlőség is teljesül, egyébként x ≠ y. [2]

3.2.2 Milliomos probléma megoldása

A program Ioannidis és Ananth 2003-as megoldása alapján íródott. [7]

A probléma megoldásához használni fogjuk az oblivious protocol-t (azon belül az 1-2 oblivious variánsát). [4] Ennek az átviteli módnak az a lényege, hogy a küldőnek van két adata S_0 és S_1, a címzett pedig választ egy $i \in \{0,1\}$-et, és a küldő elküldi neki S_i-t, úgy hogy a címzett nem tud meg semmit $S_{(1-i)}$-ről a küldő pedig semmit nem tudja meg i értékét, tehát nem tudja, hogy a címzett a két adat közül melyiket kapta meg.

Alice által birtokolt titkot *a*-val jelöljük, Bob titka pedig *b*. Feltesszük azt hogy a titkok olyan számok amiknek a bináris reprezentációja nem hosszabb valamilyen $d \in N$ számnál. Emellett van egy *k* számunk, ami az oblivious transferben használt privát kulcs hossza.

1. lépés: Alice létrehoz egy $d \times 2$-es A mátrixot, ami k-bites számokat tartalmaz. Ezen kívül választ két véletlen számot, u-t és v-t, úgy, hogy $0 \leq u < 2k$ és $v < k$, és *elég nagy*.

2. lépés: K_{ijl} fogja jelölni a K_{ij} pozícióban található k-bites szám l. bitjét, ahol $l=0$ jelöli a legkisebb helyiértékű bitet. Emellett a_i-vel jelöljük az a szám i. bitjét.

Alice a következőket teszi:

Minden i-re, ahol $1 \leq i \leq d$:

1. Minden $j \geq v$-re K_{i1j}-t és K_{i2j}-t egy random bitre állítjuk.
2. Ha $a_i = 1$, akkor legyen $l = 1$, egyébként $l = 2$. Minden j-re ahol $0 \leq j \leq 2 \cdot i - 1$ állítsuk K_{ilj}-t random bitre.
3. Legyen $m = 2 \cdot i$. Ekkor legyen $\mathbf{K_{il(m+1)}} = 1$ és $K_{ilm} = a_i$.
4. $1 \leq i < d$ esetén S_i egy véletlenszerűen választott k-bites szám és S_d egy másik k-bites szám lesz, ahol az utolsó két biten kívül az összes bitet véletlenszerűen választjuk meg. Az utolsó két bit kiszámítása pedig a következő módon történik: $S_{d(k-1)} = 1 \oplus \oplus_{j=1}^{d-1} S_{j(k-1)} \oplus \oplus_{j=1}^{d} K_{j1(k-1)}$ és $S_{d(k-2)} = 1 \oplus \oplus_{j=1}^{d-1} S_{j(k-2)} \oplus \oplus_{j=1}^{d} K_{j1(k-2)}$, ahol $\oplus$-al jelöljük a bitenkénti XOR operátort.
5. $l = 1,2$ esetén legyen $K'_{il} = leftrot(K_{il} \oplus S_i, u)$. Ahol $\mathrm{leftrot}(x, t)$ jelöli a bitenkénti balra tolását x-nek u-val.

3. lépés: Alice átküldi Bobnak 1-2 oblivious transfer segítségével K'_{il} számokat, úgy, hogy $l = b_i + 1$ és b_i az ***i***. bitje ***b***-nek ∀i: $0 \leq i \leq d$-re.

Az oblivious transfer[4] a következőképp működik:

Jelölje m_0, illetve m_1 a küldendő üzeneteket, amikből Bob csak az egyiket kaphatja meg, Alice pedig nem tudhatja meg, hogy Bob melyiket kapta meg.

Emellett az átvitel kezdetekor generálunk egy RSA kulcspárt, aminek a publikus részét átküldjük Bobnak. Jelölje ***D*** a titkos kulcsot, ***E*** a publikus kulcsot, és ***N*** a modulust.

	Alice			Bob	
	Titkos	Publikus		Publikus	Titkos
1	m_0, m_1				
2	D	N, E	⟹	N, E	
3		x_0, x_1	⟹	x_0, x_1	
4					k, b
5		v	⟸	$v = (x_b + k^E) \bmod N$	
6	$k_0 = (v - x_0)^d \; mod \; N$ $k_1 = (v - x_1)^d \; mod \; N$				
7		$m'_0 = m_0 + k_0$ $m'_1 = m_1 + k_1$	⟹	m'_0, m'_1	
8					$m_b = m'_b - k$

1. Alice kiválasztja a két titkot, amiből az egyiket szeretné eljuttatni Bobnak. Legyen a két titok m_0 és m_1.
2. Alice legenerál egy RSA kulcspárt, *D* lesz a titkos kulcs, *E* a publikus kulcs, és *N* a modulus. *E*-t és *N*-t elküldi Bobnak.
3. Alice generál két véletlen számot: x_0-t és x_1-et, majd elküldi őket Bobnak.
4. Bob kiválasztja, hogy melyik üzenetet szeretné megkapni, tehát kiválaszt egy $b \in \{0,1\}$-t majd, generál egy véletlenszerű k számot.
5. RSA-val lekódolva elküldi x_b-t és k-t. Tehát kiszámolja a $v = (x_b + k^E) \; mod \; N$ számot, és elküldi ezt Alice-nak.

6. Alice ezután kiszámolja a $k_0 = (v - x_0)^d \ mod \ N$-t és a $k_1 = (v - x_1)^d \ mod \ N$ számokat. Ezek közül az egyik pont k lesz, de Alice nem tudja, hogy melyik az (és nem is tud rájönni sehogy).
7. Alice kiszámolja $m'_0 = m_0 + k_0$ és $m'_1 = m_1 + k_1$ számokat, majd átküldi őket Bobnak, aki ebből csak az egyiket fogja tudni dekódolni, a másik semmilyen hasznos információt nem tartalmaz számára.
8. Bob kiszámolja a $m_b = m'_b - k$ számot.

Az átvitel végén Bobnál lesz a két kódolt üzenet, m'_0 és m'_1. Ebből csak azt tudja megfejteni, amelyikhez tartozó ***b***-t választotta korábban. Tehát a Bob által választott üzenet $m_b = m'_b - k$ lesz. [4]

A milliomos protokoll esetében a két titok a K'_{i1} és a K'_{i2} lesz, ezekből Bob csak az egyiket fogja megkapni, és az, hogy melyiket az a titkától függ. Ha $b_i = 0$, akkor az első üzenetet (tehát K'_{i1}-et), egyébként a másikat kapja meg.

4. lépés: Alice elküldi Bobnak az $\mathrm{N} = \mathrm{rot}(\bigoplus_{j=1}^{d} S_j, \mathrm{u})$ számot.

5. lépés: Bob ezután kiszámolja a bitenkénti XOR-ját a 3. illetve a negyedik lépésben kapott számoknak. Ezután Bob elkezdi végig menni a kapott bitsorozaton balról jobbra, amíg nem talál elég nagy 0 sorozatot. Nevezzük el ***c***-nek az első bitet ami a 0 sorozattól jobbra van (***c*** nem 0). Hogyha a ***c*** jobb oldali szomszédja 1, akkor a $\geq$ b, egyébként a $<$ b. [7]

3.3 Tervezés

Miután körvonalazódott, hogy mi az elvárás a programmal szemben a funkcionalitások, és a felhasználóval való interakció terén, elkezdhetjük konkrétabban megtervezni az alkalmazást.

Azt tudjuk, hogy szükség van egy egyszerű grafikus felhasználói felületre. Ezt a .NET-es WPF tökéletesen megoldja, nagyon gyorsan és kényelmesen össze lehet rakni benne a célnak megfelelő felületet.

A programnak tudnia kell a hálózaton kommunikálni, erre a TCP protokoll tökéletesen megfelel, viszont akkor a hálózati kommunikáció szempontjából meg kell különböztetnünk egy szervert és legalább egy klienst. Ez sem jelent problémát, mivel a kriptográfiai protokolljaink, amiket használunk, amúgy is aszimmetrikusak, ezért mindenképp lesznek

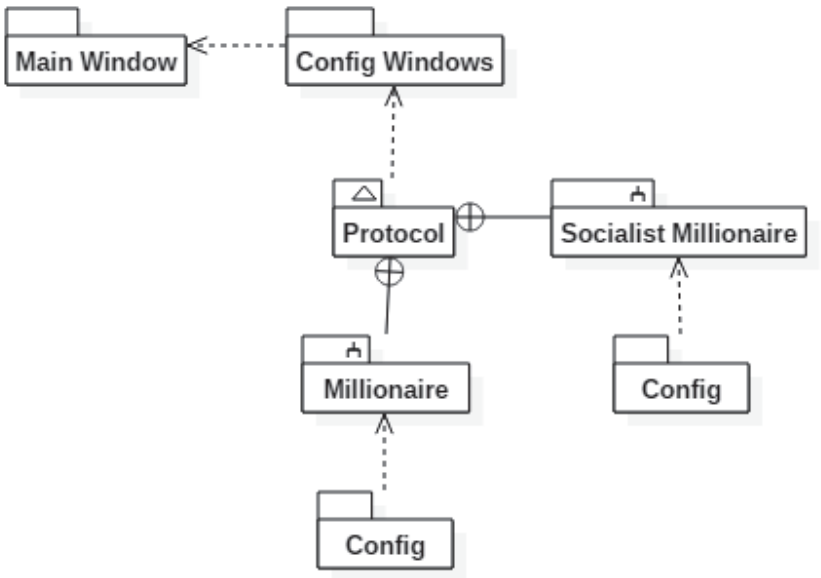
megkülönböztetett szerepek. Emiatt egyszerűen azt mondjuk, hogy az algoritmusban amelyik félt Alice-nek nevezzük az lesz a szerver, Bob pedig a kliens lesz.

Ezen felül azt szeretnénk, hogyha a beállításainkat nem kéne minden alkalommal begépelni, hanem képesek lennénk elmenteni, illetve betölteni azokat, tehát a programnak rendelkeznie kell egy egyszerű perzisztenciával. Mivel a beállítások egyszerű szövegekkel reprezentálható adatok, ezért tökéletesen megfelel, ha az adatokat egy szöveges fájlba írja a program, illetve onnan kiolvassa.

Mivel a program rendeltetésszerű használata első ránézésre nem feltétlenül egyértelmű, azért fontos, hogy a felhasználó könnyen hozzájuthasson a szükséges információkhoz, viszont mindezt úgy kell elérni, hogy a felhasználói felületet nem bonyolítjuk túl felesleges dolgokkal. A megvalósítás közben fontos ezt is észbentartani.

3.3.1 Szoftver architektúra

A program háromrétegű MVP architektúrában íródott. A megjelenítésért a **Window** osztályok felelnek, a model a **Server**, illetve **Client** osztályokban található a perzisztencia pedig a **Config** osztályokban van.

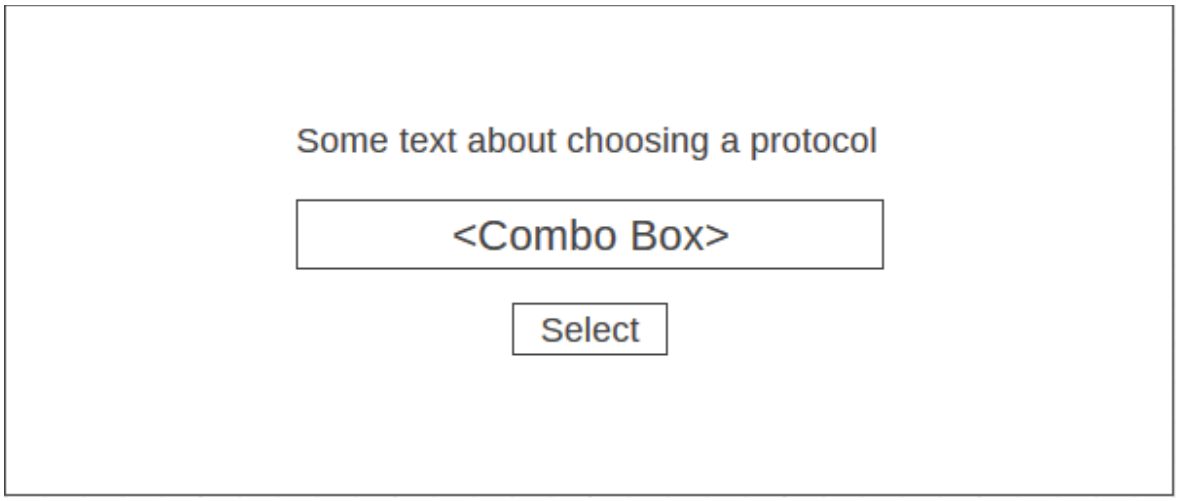


A **Main Window** csomag egyetlen osztályt tartalmaz, a *MainWindow*t, ez tartalmazza a fő képernyő megjelenítéséért felelős kódot. A **Config Windows** csomagban vannak a *SocialistWindow*, és a *MillionaireWindow* osztályok, ami a két protokoll konfigurációs ablakának megjelenítését végzi. Ebből a két csomagból áll össze a *View*.

A **Protocol** csomag tartalmazza az alkalmazás *Model* részét. Ennek két alcsomagja a **Millionaire** és **Socialist Millionaire** csomagok. Ezekben vannak a megfelelő protokollhoz tartalmazó *Client*, illeve *Server* osztályok.

3.3.2 Felhasználói felület

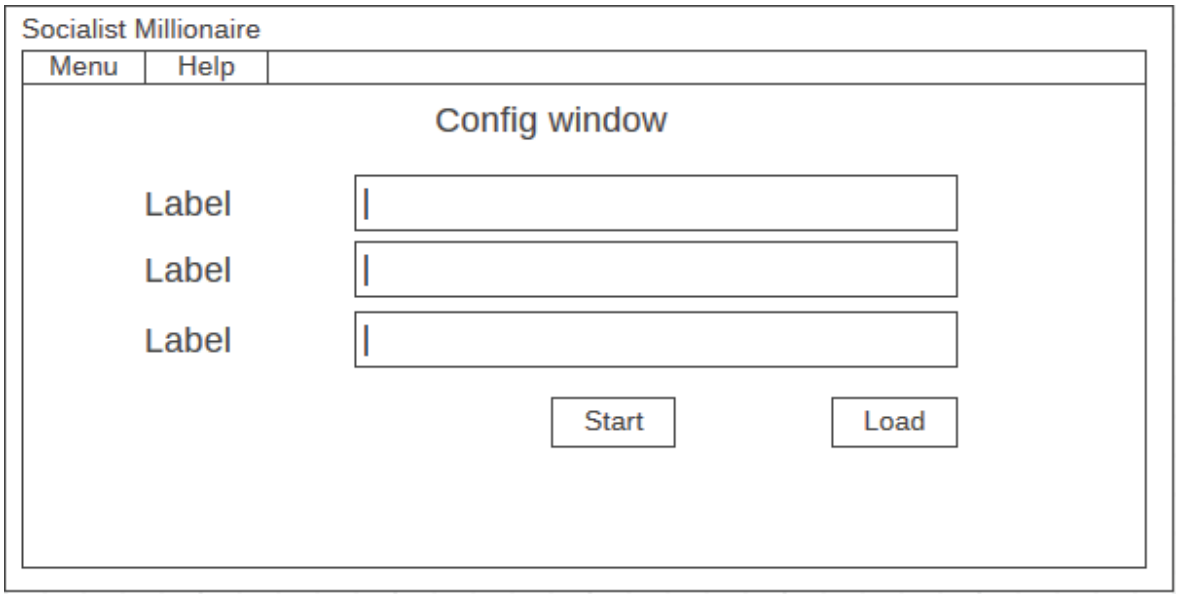
Mivel a program három különböző funkcióból áll, a kezdőképernyő egy protokoll választó felület, ahol csak annyi történik, hogy a felhasználó kiválasztja, hogy melyikre funkcióra van szüksége, minden más csak a kiválasztás után jelenik meg.



A képernyő tetején található egy szöveg, alatta egy Combo Box amivel ki lehet választani, hogy a három protokollból melyiket szeretnénk használni, a Combo Box alatt, pedig egy ***select*** feliratú gomb, amivel véglegesítjük a kiválasztást.

Miután kiválasztottuk a protokollt a konfigurációs ablak nyílik meg. Itt van lehetőségünk *config* fájlból betölteni a beállításokat, vagy helyben kézzel kitölteni, hogy milyen beállításokkal szeretnénk, hogy fusson a program. Ezek a konfigurációs képernyők nem sokban különböznek egymástól, a lényeg, hogy van felül egy kis menü egy pár dologgal, egy külön help menüponttal ami leírja a fontosabb tudnivalókat (a felhasználói dokumentációhoz

hasonló módon), és pár Text Box-al amikbe a konfigurációs paraméterek kerülnek, egy ***load***, illetve egy ***start*** gombbal.



Nyilván a címkék helyére a különböző paraméterek nevei kerülnek majd, és protkollonként változó mennyiségű paraméter miatt nem pont három text box lesz, hanem amennyire szükség van, de hasonló elrendezéssel.

Emellett kelleni fog egy ablak a titok bekérésére. Ezt az ablakot lehetséges kihagyni, mivel a titkot bekérhetjük a konfigurációs ablakban is, de úgy gondolom, hogy ez egy sokkal fontosabb adat a többinél, ezért inkább kiemeltem egy külön ablakba.

Mivel egyszerű funkciót tölt be (egy adat bekérése), ezért nem is kell szinte semmi az ablakra. Viszont, mivel egy kényes információról beszélünk, azért a normál *text box* helyett, célszerűbb *password boxot* használni a titok bekérésére.

Egy másik hasonlóan egyszerű, de annál fontosabb ablak az eredmény ablak, ami akkor jelenik meg, hogyha befejeződik a protokoll, és kiírja az eredményt.

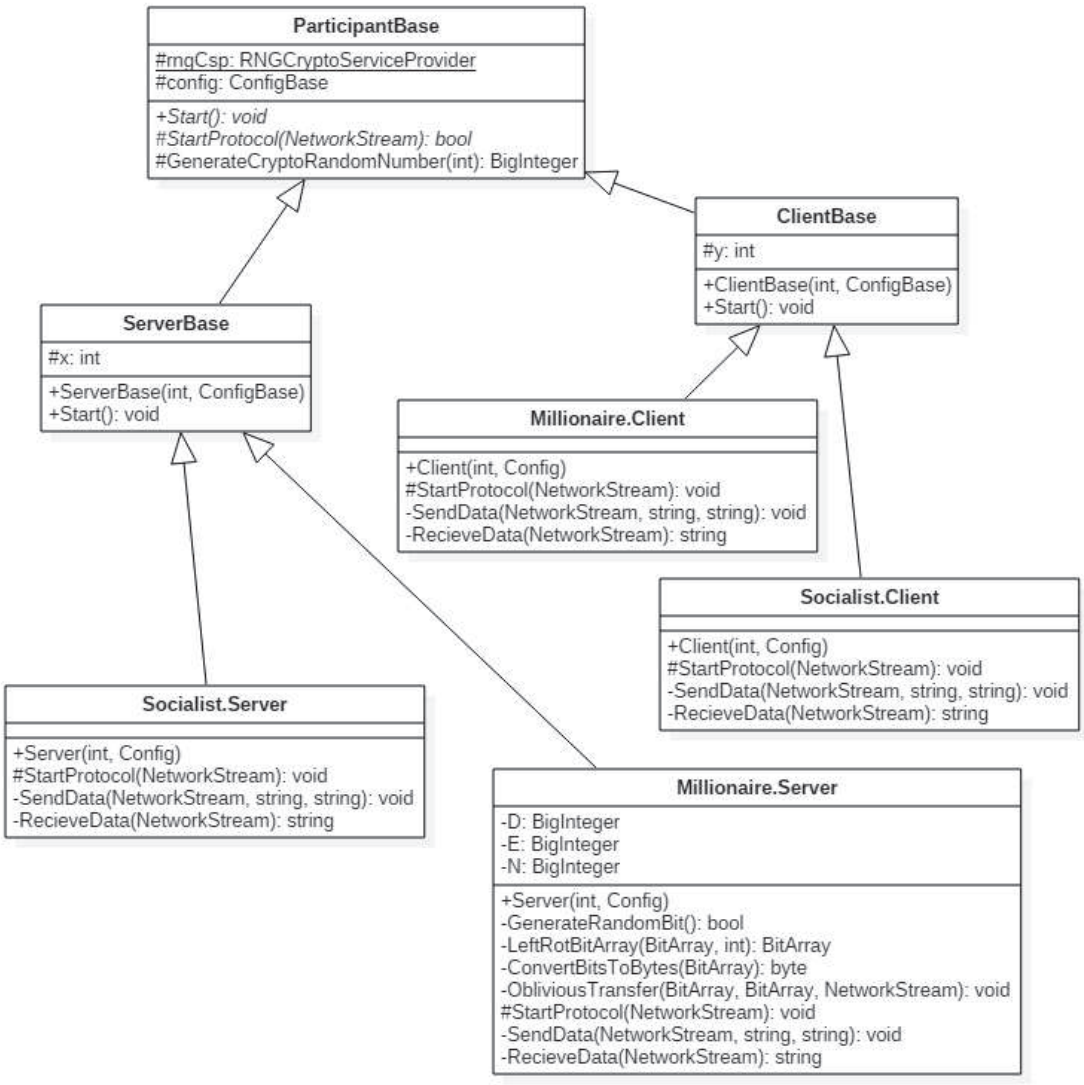
A ***ResultLabel*** helyére megy a számítások eredménye, aminek a lehetséges értéke protokollonként változó.

Mivel a program futása alatt a háttérben rengeteg számítás, és információ csere történik, ezért a felhasználói felület ablakai mellett végig fut egy konzol ablak is, ami logolja az információcserét a kommunikációs felek között. Nem a legelegánsabb megoldás, de szerintem egy ilyen jellegű program esetében fontos, hogy végig átláthatóan működjön. A konzol ablakot lehet helyettesíteni az adatok logfájlba való kiírásával is, de szerintem a konzol a jobb megoldás, mivel így futás közben végig látszik minden, ami történik.

3.4 Megvalósítás

Ahogy már korában is írtam a megjelenítésért a *Window* osztályok felelnek. A felhasználói felület minden egysége egy-egy külön ablak, ezek a WPF-es *Window* ősosztályból származnak mind.

Ahhoz, hogy csökkentsem a kódismétlést írtam több absztrakt osztályt is. A kommunikációban résztvevő feleket reprezentáló osztályok a *ParticipantBase* ősosztályból származnak. Itt vannak definiálva a *Client* és a *Server* osztályok közös függvényei. Ebből származnak a *ClientBase* illetve a *ServerBase* osztályok, amik összefoglalják protokoll függetlenül külön a szerverek és külön a kliensek közös tulajdonságait. A *ClientBase* és a *ServerBase* osztályokból származnak a konkrét *Server*, illetve *Client* osztályok, amikben már az adott protokoll tényleges megvalósítása szerepel.



A *ParticipantBase* osztály tartalmazza a *RNGCryptoServiceProvider* statikus osztály egy példányának inicializálását, amit a az összes protokoll használ a kriptográfiailag biztonságos véletlenszámok generálásához. Ennek az osztálynak a felhasználása rögtön meg is jelenik a *GenerateCryptoRandomNumber* függvényben, ami annyit csinál, hogy kap egy ***b*** egész számot, és visszaad egy ***b*** byte hosszú *BigInteger* típusú véletlen számot. Emellett az osztály leírja, hogy minden protokollban résztvevő osztálynak implementálnia kell Start, illetve egy *StartProtocol* metódusokat. Az előbbi publikusan meghívható és ez jelzi, hogy el kell kezdeni előkészíteni, illetve végrehajtani a protokollt. A *StartProtocol* az a védett metódus, ami a tényleges protokoll kódját tartalmazza, és *Client* illetve a *Server* osztály akkor hívja

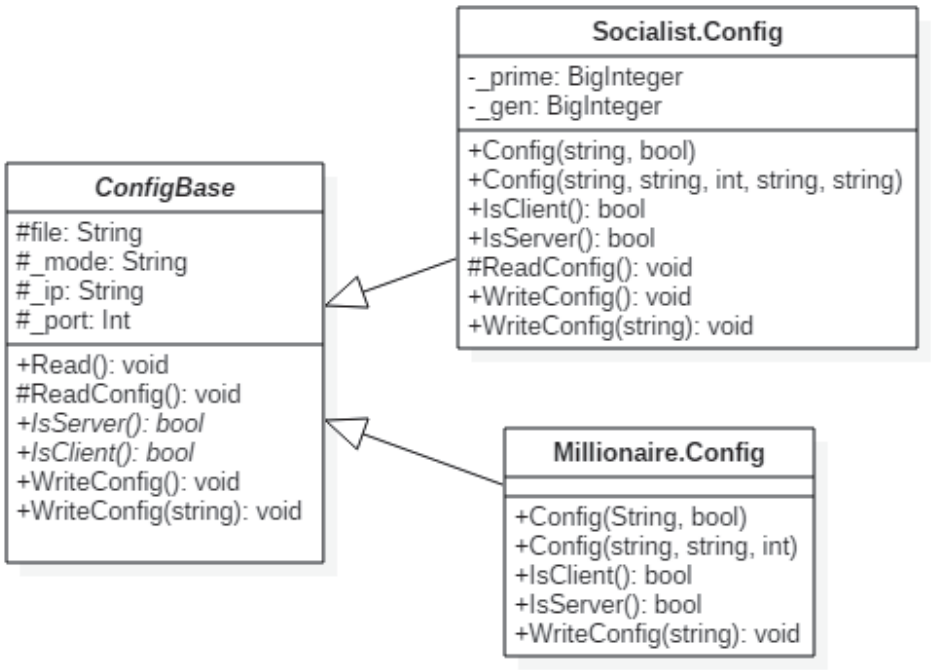
meg ezt, hogyha minden elő van készítve és a kliens és a szerver sikeresen csatlakozott egymáshoz.

A *ServerBase*, illetve a *ClientBase* osztályok tartalmazzák a *Start* függvény implementációját, mivel az összes kliens, illetve összes szerver ugyanazon előkésztési lépéseken kell, hogy keresztülmenjen. A szerver megnyitja a socketjét a megadott porton, és várja a klienst, a kliens pedig csatlakozik a megadott ip-re és portra. Ha ez befejeződött akkor kerül meghívásra a *StartProtocol* metódus.

A szocialista milliomos protokoll szerver és kliens osztálya meglehetősen egyszerű, a *StartProtocol*-ban van a lényegi kód, a *SendData*, illetve a *RecieveData* intézi az információátvitelt a két fél között.

A milliomos protokoll kliense is hasonlóan egyszerű módon épül fel, viszont a szerver lényegesen összetettebb, és több függvényből áll, mint bármi más a programban. A számolásaihoz rengeteg segédfüggvényre volt szükség. A *GenerateRandomBit*, a *GenerateCryptoRandomNumber* függvény felhasználásával előállít egy véletlenszerű bitet. A *LeftRotBitArray* egy bit tömböt tol el balra a paraméterként megadott számú pozícióval. A *ConvertBitsToBytes* függvény a paraméterként kap egy bit tömböt, és egy azzal azonos értékű byte tömböt ad vissza. Az *oblivious transfer* pedig az *1-2 oblivious transfer protokoll* [4] küldő oldali megvalósítását tartalmazza.

A perzisztenciának is készítettem egy *ConfigBase* ősosztályt, mivel a két protokollnak beállításai csak kevésben térnek el. Ebből az ősosztályból származnak a tényleges konkrét beállításokat tartalmazó *Config* osztályok, amik segítségével felkonfigurálhatjuk a protokollokat, illetve ezek tartalmazzák a beállítások fájlból történő kiolvasásához, és lementéséhez szükséges kódot is.



A *ConfigBase* definiálja a másik két osztály alapszerkezetét. Tartalmazza a közös mezőket, amik minkét protokollhoz szükségesek. Ezek a *mode*, hogy kliens vagy szerver szerepet tölt-e be a program, az *ip*, ami a szerver IP címét határozza meg, illetve a *port*, ami a szerver által használt port számot írja le.

A *ReadConfig* függvény intézi az adatok fájlból való kiolvasását. A szocialista milliomos protokoll *Config* osztálya felüldefiniálja ezt a függvényt, mivel a szervernek az alap három adatnál több dologra van szüksége. Ez látszik is *Socialist*.*Config* osztály mezőiben is, mivel itt van két plusz *BigInteger* típusú adat.

Az *IsClient*, illetve az *IsServer* függvények megmondják, hogy a *Config* osztály adott példánya milyen módba van állítva.

A *WriteConfig* metódus a mentésért felelől, kiírja a beállított adatokat a paraméterként megadott fájlba.

3.4.1 Implementációs döntések

Ebben a részben mutatom be, hogy az implementáció közben milyen, a modelltől eltérő döntéseket hoztam meg.

Első körben, az absztrakt algoritmusoktól minden olyan esetben, ahol az indexelés nem 0-tól kezdődött eltértem, mivel a C# alapértelmezetten a tömb adatszerkezetet 0-tól kezdi indexelni, ezért nekem sokkal egyszerűbb volt, hogy áttranszformáltam minden intervallumot 0 kezdetűvé. Tehát ha a Milliomos protokoll $1 \leq i \leq d$ intervallumon ment végig, akkor a programban a *for ciklus* $0 \leq i \leq d$ tartományt járja be.

3.5 Továbbfejleszthetőség

A projekt szerintem nagyon egyszerűen tovább fejleszthető több irányba is.

A legegyértelműbb lehetőség további protokollokkal kiegészíteni a programot. A meglévő kódhoz mindössze annyit kell hozzátenni, hogy a kezdőoldal felhasználói felületén található *combo boxba* belerakni az új lehetőségeket, és a kezelőablak kódjában a ***select*** gomb kattintás eseményében található elágazást ki kell pár ággal egészíteni, amikben az új protokollokhoz kapcsolódó ablakokat megnyitjuk. Innentől tudunk dolgozni az eddigi kódtól teljesen függetlenül (a legtöbb esetben a meglévő *Client*, *Server*, illetve *Config* ősosztályok kódja hasznos lehet, de nem szükséges használni).

Egy másik lehetséges – és az előző iránnyal nem feltétlen ellentétes – opció, a protokollok kódját kiemelni egy könyvtárrá, amit más projektek tudnak használni. Természetesen itt is ki lehet egészíteni további kriptográfiai protokollokkal. Ebben az esetben a felhasználói felület elveszti a jelentőségét. Ebben az esetben az osztályszerkezeten célszerű kicsit módosítani, illetve a kliens-szerver módokkezelését is lehet, hogy újra kell gondolni. Ez az irány kevésbé egyértelmű kihívásokat rejt magában, mint a másik, de szerintem mindenképpen egy működőképes, és hasznos alternatíva lehet.

Esetleg a protokollok köré lehet írni egy olyan programot, ami használja is őket, tehát, hogy a protokoll ne csak magában legyen, hanem a program egy felhasználását is tartalmazza.

Természetesen az általam felsoroltakon túl is lehetnek egyéb irányok amerre fejleszthető a program, ebben a fejezetben csak egy rövid kitekintést szerettem volna tenni, hogy mit lehet még esetleg kezdeni ezzel a projekttel a jövőben.

3.6 Tesztelés

Ez a fejezet tartalmazza a program tesztelési tervét.

Mivel a tervezési részben bemutatott felhasználói történetek egyúttal egy funkcionális tesztelési útvonalat is bemutatnak, ezért első körben ezek mentén kezdjük el a kész szoftver tesztelését.

Elvárás	Tényleges esemény
Amennyiben a kezdőképernyőn vagyok, *ha* kiválasztom a szocialista milliomos protokollt, *akkor* az alkalmazás tovább lép a konfigurációs ablakra.	Protokollválasztás után a ***select*** gombra kattintva ténylegesen átkerültünk a szocialista milliomos prtokoll konfigurációs ablakára.
Amennyiben a konfigurációs ablakon vagyok, *ha* a ***load*** gombra kattintva betöltöm az adatokat a konfigurációs fájlból, *akkor* megjelennek az általam várt adatok a képernyőn.	A ***load*** gombra kattintva megjelent egy párbeszédablak, amivel ki tudtam választani, hogy melyik fájlt szeretném betölteni. Miután kiválasztottam az általam előkészített config fájlt, a program tényleg betöltötte az általam várt adatokat, és kiírta őket a megfelelő *text boxokba*.
Amennyiben minden be van állítva minden, *ha* megnyomom a ***start*** gombot, *akkor* a program bekéri tőlem a titkos kulcsot.	A ***start*** gomb megnyomása után megjelent egy ablak, ahol egy *password boxba* be lehetett írni a titkos kulcsot.
Amennyiben beírom a titkos kulcsot, *ha* megnyomom az ***ok*** gombot, akkor elindul a protokoll.	Beírtam a titkot, megnyomtam az ***ok*** gombot, és a program a konzol ablakban elkezdte kiírni a hálózati kommunikáció során cserélt adatokat.
Amennyiben elindult a protokoll, és kliens oldali módban vagyok, ha a szerver elküldi a kódolásra használt prímszámot, illetve generátor elemet, akkor kiírja ezeket a program és megvárja amíg elfogadom őket.	Kliens módban futtattam a protokollt, és rögtön az elején kiírta a program a két számot, és addig nem ment tovább amíg le nem okéztam őket.
Amennyiben elindul a protokoll, *ha* megvárom amíg befejeződik, *akkor* a program kiírja a végeredményt.	A protokoll indulása után 1-2 másodperccel megjelent egy ablak, ami kiírta az eredményt.

Amennyiben a program kiírja a végeredményt, *ha* az ***ok*** gombra kattintok, *akkor* visszakerülök a kezdőképernyőre.	Az eredményablakban az ***ok*** gombra kattintás után megjelent a kezdőképernyő.

A tervezéshez képest egy kicsit változtattam a felhasználói történetek szövegén, de a lényeg nem változott sehol. A szöveg azért változott, mert szerintem így jobban tükrözi a ténylegesen elkészített szoftver működését. Nyilván a tervezési fázis alatt még nem voltam tisztában pontosan mindennel, hogy miként is fog kinézni. A felhasználói történetek jelentése nem változott meg, csak a szöveg lett kicsit pontosítva.

Emellett kiegészítettem egy pluszt esettel az eredeti történetet (amikor a kliens a protokoll elején ellenőrzi a szerver által prezentált számokat).

Elvárás	Tényleges eredmény
Amennyiben a konfigurációs ablakban vagyok, *ha* megnyomom a ***help*** menüpontot, *akkor* megjelenik egy rövid leírás a protokollról.	A konfigurációs ablakban található ***help*** gomb megnyomása után megnyílik egy fájl, ami a protokollok rövid leírását tartalmazza.
Amennyiben a konfigurációs ablakban vagyok, *ha* rákatintok a ***back to select window*** menüpontra, *akkor* visszakerülök a kezdőképernyőre.	A konfigurációs ablakban található ***back to select window*** menüpontra kattintás után visszakerültem a kezdőképernyőre.
Amennyiben a kezdőképernyőn vagyok, *ha* kiválasztom a milliomos protokollt, *akkor* a program tovább lép a protokoll konfigurációs ablakára.	A milliomos protokoll kiválasztása után amikor megnyomtam a ***start*** gombot, átkerültem a megfelelő konfigurációs ablakba.
Amennyiben a konfigurációs ablakon vagyok, *ha* kitöltöm az összes mezőt, *akkor* indításra kész a protokoll.	Kitöltöttem kézzel az összes beállítást, és el tudtam indítani a protokollt, a ***start*** gomb megnyomásával.
Amennyiben ki van töltve az összes szükséges beállítás, *ha* megnyomom a	A kitöltött beállításokat el tudtam menteni egy fájlba a ***save config*** menüpont használatával.

menüben a ***save config*** mezőt, *akkor* a program elmenti a beállításaimat.	
Amennyiben elmentettem a beállításaimat, *ha* később megnyomom a ***load*** gombot és kiválasztom a fájlt, amibe mentettem a korábbi adatokat, *akkor* rendben betölti és megjeleníti őket a program.	Mentés után kitöröltem mindent a program *input fieldjeiből*, majd betöltöttem a fájlt, ahova elmentettem korábban az adatokat, és sikeresen betöltötte azokat az adatokat, amiket megadtam.

Ezzel lefedtük a program kényelmi funkcióit, illetve a különböző navigálási opciókat.

Következő lépés meggyőződni arról, hogy a protokollok implementációi is helyesek. Mivel mindkét protokoll nagyban épít a véletlen generált számok használatára, ezért fontos, hogy minden tesztet többször elvégezzünk. Ezért minden tesztet öt alkalommal is el fogok végezni, és minden esetet többféle számmal is ki fogok próbálni, hogy a lehető legnagyobb valószínűséggel ki tudjam szűrni az esetleges hibákat.

3.6.1 Szocialista milliomos protokoll

Első körben teszteljük a legegyszerűbb esetet: kis számot, amik nem egyenlőek és még túl közel sincsenek egymáshoz.

Alice titka	Bob titka	Elvárás	Eredmény
1	29	Nem egyenlő	5/5
35	2	Nem egyenlő	5/5
15	94	Nem egyenlő	5/5
77	32	Nem egyenlő	5/5

Ezután teszteljük ugyanezt az esetet nagyobb számokkal

Alice titka	Bob titka	Elvárás	Eredmény
93421	82301	Nem egyenlő	5/5
12001	4344	Nem egyenlő	5/5
2018	516	Nem egyenlő	5/5

1345	80312	Nem egyenlő	5/5

A következő tesztesetekben olyan számaik lesznek a résztvevőknek, amik nem túl nagyok, viszont nagyon közel vannak egymáshoz.

Alice titka	Bob titka	Elvárás	Eredmény
2	3	Nem egyenlő	5/5
43	42	Nem egyenlő	5/5
143	141	Nem egyenlő	5/5
199	197	Nem egyenlő	5/5

Most pedig olyan eseteket vizsgálunk, ahol a titkok nagyok, de aránylag közel vannak egymáshoz.

Alice titka	Bob titka	Elvárás	Eredmény
23410	23411	Nem egyenlő	5/5
98722	98721	Nem egyenlő	5/5
42999	43000	Nem egyenlő	5/5
56720	56714	Nem egyenlő	5/5
2377	2378	Nem egyenlő	5/5

Miután megvizsgáltuk azokat az eseteket, ahol nem egyezik meg a két titok, térjünk rá azokra, ahol megegyezik a két szám, de még nem kifejezetten nagyok a számok.

Alice titka	Bob titka	Elvárás	Eredmény
42	42	Egyenlő	5/5
3	3	Egyenlő	5/5
95	95	Egyenlő	5/5
278	278	Egyenlő	5/5

Az utolsó tesztesetek olyanok, ahol a két titok egyenlő és viszonylag nagyobb számok.

Alice titka	Bob titka	Elvárás	Eredmény
29671	29671	Egyenlő	5/5
99810	99810	Egyenlő	5/5
75622	75622	Egyenlő	5/5
100944	100944	Egyenlő	5/5

A tesztesetek alapján arra következtethetünk, hogy a szocialista milliomos protokoll helyesen van implementálva.

3.6.2 Milliomos protokoll

A milliomos protokoll egy kicsit összetettebb tesztelés szempontjából, mivel előfordulhat, hogy a protokoll nem ad eredményt, mivel nagyon ritkán előfordulhat, hogy a program nem talál elég hosszú nulla sorozatot, ezért nem tudja egyértelműen megmondani a végeredményt.

Az protokoll tesztelésének első fázisában nézzünk meg kis számokat, amik viszonylag távol állnak egymástól.

Alice titka	Bob titka	Elvárás	Eredmény
2	23	a <= b	5/5
4	38	a <= b	5/5
19	1	a >= b	5/5
102	57	a >= b	5/5
234	500	a <= b	5/5
11	41	a <= b	5/5
78	60	a >= b	5/5

Következőnek vizsgáljuk meg az egymástól viszonylag távol álló nagy számokat.

Alice titka	Bob titka	Elvárás	Eredmény
1034	789	a >= b	5/5
1002	1201	a <= b	5/5
890124	865012	a >= b	5/5

67508	60508	a >= b	5/5
29010	31100	a <= b	5/5
6092	7840	a <= b	5/5
54054	60060	a <= b	5/5

Majd ellenőrizzük le, hogy ugyanígy működik kicsi, egymáshoz közeli számokkal is.

Alice titka	Bob titka	Elvárás	Eredmény
1	2	a <= b	5/5
4	3	a >= b	5/5
92	90	a >= b	5/5
201	203	a <= b	5/5
348	350	a <= b	5/5
411	410	a >= b	5/5
637	636	a >= b	5/5

Ezután nézzük meg, hogy közeli nagy számokra is jó eredményt ad.

Alice titka	Bob titka	Elvárás	Eredmény
1024	1023	a >= b	5/5
1908	1909	a <= b	5/5
6001	6002	a <= b	5/5
8710	8708	a >= b	5/5
57793	57794	a <= b	5/5
120566	120665	a <= b	5/5
346091	346092	a <= b	5/5

Végül vizsgáljuk meg az egyenlőséget. Itt az az elvárás, hogy mindkét eredményt megkapjuk, nagyjából hasonló alkalommal. Az eredmény oszlopban az x/y azt mutatja meg, hogy hányszor jött ki az y számú tesztből, hogy a >= b.

Alice titka	Bob titka	Elvárás	Eredmény
42	42	Változó	1/5
2456	2456	Változó	2/5
19876	19876	Változó	3/5

A 15 tesztből 6 alkalommal lett a >= b az eredmény, ami nagyjából megfelel az elvárásainknak.

A tesztelés során amikor nem jött ki végeredmény, azt figyelmen kívül hagytam, és újra futtattam azt az esetet. Ez egy teljesen normális működés, és az algoritmus velejárója. Ezt ki lehetne szűrni (de inkább csak lecsökkenteni az előfordulás gyakoriságát), hogyha kisebb nulla sorozat esetén is detektálna eredményt, ezzel viszont azt kockáztatnám, hogy néha rosszul ítélné meg az eredményjelző bit helyzetét. Elméletben a jelenlegi hossznál is előfordulhat, hogy rossz bitet néz meg a program, de ennek az esélye elhanyagolható (ehhez az kell, hogy az eredményt adó nulla sorozat előtt, és vele nem összeérve keletkezik a véletlenszerűen választott bitekből egy legalább 28 bit hosszú nulla sorozat).

4 Irodalomjegyzék

[1]: https://en.wikipedia.org/wiki/Secure_multi-party_computation 2018.05.04

[2]: https://en.wikipedia.org/wiki/Socialist_millionaires 2018.05.04

[3]: https://en.wikipedia.org/wiki/Yao%27s_Millionaires%27_Problem 2018.05.04

[4] https://en.wikipedia.org/wiki/Oblivious_transfer#1%E2%80%932_oblivious_transfer 2018.05.15

[5] https://en.wikipedia.org/wiki/Off-the-Record_Messaging 2018.05.23

[6] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#Generalization_to_finite_cyclic_groups 2018.05.23

[7] https://en.wikipedia.org/wiki/Yao%27s_Millionaires%27_Problem#The_protocol_of_Ioannidis_&_Ananth[3] 2018.05.23