Török Viktor

Szteganográfia és kriptográfia

Szentes

2015

1. Bevezetés

A történelem tanulmányozása során az érdeklődő olvasó már az ókor idejéből származó feljegyzésekben is találkozhat olyan esetekkel, amikor két szemben álló oldal vezérei vagy politikusai el szerették volna egymás elől titkolni szándékaikat, ezért a fontosnak ítélt üzeneteiket titkosították, vagy úgy próbálták meg eljuttatni azokat a címzett részére, hogy az ellenséges oldal erről ne szerezzen tudomást. Ezt a tevékenységet több-kevesebb sikerrel már évezredek óta űzik a katonák és a diplomaták, és egy-egy titkosítóra általában jut a másik oldalon legalább egy olyan személy is, aki megpróbálja elfogni és megfejteni az ellenfél üzeneteit.

A titkosítással és annak visszafejtésével a kriptográfia és a kriptoanalízis [1], az adatrejtéssel pedig a szteganográfia és a szteganalízis [2] tudománya foglalkozik. Én az ELTE programozó matematikus szakán folytatott tanulmányaim során találkoztam először a kriptográfia néhány alapvető kérdésével, de a téma akkoriban nem keltette fel különösebben az érdeklődésemet. A GAMF mérnökinformatikus rendszergazda FOSZK képzésén hallgatott *Az informatikai biztonság alapjai* előadás szintén érintette, és érdekes formában tárgyalta mind a kriptológia, mind a szteganográfia alapjait, és ez a szakirodalom további tanulmányozására sarkallt. Végül szakdolgozati témaként is a szteganográfia és a hozzá szervesen kapcsolódó kriptográfia témakörét választottam. Jelen dokumentum a 2015 folyamán elkészült és sikeresen megvédett szakdolgozat egy néhány helyen módosított és kiegészített változatát tartalmazza.

A dolgozatban egy olyan program megtervezését és implementálását ismertetem, amely képes egy megadott, titkosítatlan szöveget titkosítani, és egy meghatározott formátumú képben, fotóban azt elrejteni az illetéktelenek elől. Kutatásaim során több olyan, már létező – fizetős és ingyen használható – szoftverrel találkoztam, amely képes elvégezni ezt a feladatot, de úgy gondolom, hogy mindenféleképpen tanulságos az alapvető kriptográfiai és szteganográfiai algoritmusok implementálása, és egy saját igényeknek megfelelő program létrehozása.

A szteganográfiai program Visual Studioban készült teljes változatát az alábbi címről lehet letölteni:

http://prog.lidercfeny.hu/szteganografia/szteganografia.zip

Ha valaki csak a futtatható változatra kíváncsi, akkor a következő címről töltheti le a programot:

http://prog.lidercfeny.hu/szteganografia/szteganografia_futtathato.zip

Ha valaki fel szeretné venni velem a kapcsolatot, akkor azt a kapitany@lidercfeny.hu e-mail címen keresztül teheti meg.

2. Az elméleti háttér áttekintése

2.1. A titkosítás alapkérdései

A titkosítási módszerek alkalmazására az általunk értékesnek tartott információk védelme érdekében van szükség [3]. Ezek az információk különféle adathordozókon és átviteli csatornákon találhatók meg.

Az értékes, védendő információk lehetnek katonai, diplomáciai vagy üzleti titkok, de a titkosítást magánemberek is felhasználhatják saját privát szférájuk védelmére.

Általában az a célunk, hogy egy adott információt úgy juttassunk el egy címzetthez, hogy annak tartalmához csak ő, mint jogosult személy férhessen hozzá [4]. Ebben a folyamatban legalább két résztvevő található, a feladó és a címzett, de általában feltételezhető, hogy van egy harmadik személy is, a támadó, aki hozzá szeretne jutni az üzenet tartalmához. Erre a feladatra két megközelítési módot vizsgálunk meg: a kriptográfiát és a szteganográfiát.

2.2. Kriptográfia és kriptoanalízis

A kriptográfia az üzenetek védelmét jelenti oly módon, hogy azok tartalmát az arra nem jogosult személyek számára értelmezhetetlenné tesszük [5], ennek eredményeképpen az üzenethez csak az arra jogosult fél férhet hozzá.

Azt az üzenetet, amelyet a feladó el szeretne juttatni a címzetthez, titkosítatlan formájában nyílt szövegnek nevezzük. Ezt a nyílt szöveget valamilyen algoritmussal titkosítjuk, ennek eredménye a titkosított vagy kriptoszöveg. A titkosított szöveg nyílt szöveggé való jogosult visszaalakítását megfejtésnek, a jogosulatlan visszaalakítását pedig visszafejtésnek vagy feltörésnek nevezzük. A titkosításhoz szükség van egy kulcsra, ami egy kiegészítő adat, és ennek ismeretében lehet megfejteni egy adott kriptoszöveget [6].

A kriptoanalízis a titkosított üzenet feldolgozását jelenti azzal a céllal, hogy a feltörő a titkosító kulcs előzetes ismerete nélkül meg tudja fejteni a titkosított üzenetet [7], azaz a kriptoanalízis alkalmazói általában a titok illetéktelen megfejtésére, feltörésére irányuló eljárásokkal foglalkoznak.

A kriptográfiát és a kriptoanalízist összességében kriptológiának hívjuk. A kriptológia kifejezés a szó rejtésének tudományát jelenti, és a görög krüptosz (rejtett), illetve a logosz (szó) szavakból származik [8].

2.3. Szteganográfia és szteganalízis

A szteganográfia a kriptológiához hasonlóan görög eredetű szó, jelentése rejtett írás, vagy használhatjuk helyette az adatrejtés kifejezést is [9]. A szteganográfia célja a kommunikáció elrejtése, ellentétben a kriptográfiával, ahol a támadó nagy valószínűséggel észreveszi, lehallgatja, majd ezek után fel is törheti, és módosíthatja is az üzenetet. A szteganográfiai módszerek a nyílt szöveget úgy rejtik el egy másik üzenetben, hogy a támadó nem is gyanítja, hogy a továbbított üzenet egy második üzenetet is tartalmaz [10]. Ebben az esetben az illetéktelen fél – szándékaink szerint – észre sem veszi az kettős üzenetküldés tényét.

Azt az objektumot, amibe beletesszük az elrejtendő információt, hordozónak vagy fedőinformációnak nevezzük. Azt az információt pedig, amit beleteszünk a hordozóba, nyílt szövegnek vagy üzenetnek hívjuk.

Az adatrejtéshez nem csak számítógépes technikákat lehet felhasználni, hanem például láthatatlan tintát, vagy mikropontokba rejtett írást is. Számítógép alkalmazása esetén szóba jöhet képek és hangállományok hordozóként történő felhasználása, melyek tartalmának kis mértékű módosításával úgy tudjuk elrejteni az eltitkolni szándékozott információt, hogy az eredeti hordozóhoz képest a változás észrevehetetlen lesz az emberi érzékelés (látás, hallás) számára.

A kritpográfia és a kriptoanalízis kapcsolatához hasonlóan a szteganográfiának is létezik egy ellentéte, amely a hordozókba ágyazott rejtett információk észlelésével foglalkozik, és amelyet szteganalízisnek nevezünk. A szteganalízis a rejtés tényének megállapításával foglalkozik. Miután sikeresen megtörtént az adatrejtés detektálása, az elrejtett üzenetet a támadónak még ki is kell nyernie a hordozóból [11].

2.4. A kriptográfia és a szteganográfia kapcsolata

A kriptográfiai és szteganográfiai algoritmusokat együttesen is fel lehet használni, melynek eredményéül kettős védelmet kapunk. Ez azt jelenti, hogy a nyílt szöveget először egy vagy több kriptográfiai módszerrel titkosítjuk, majd az így kapott kriptoszöveget elrejtjük egy hordozóban. Ebben az esetben a címzettnek ismernie kell a titkosításhoz felhasznált kulcsot is ahhoz, hogy megismerhesse az eredeti üzenet tartalmát. A kettős védelem megnehezíti a támadó dolgát, ugyanis ha esetleg észre is veszi, hogy egy adott hordozóban rejtett üzenet található, és azt ki is tudja belőle nyerni, akkor is csak egy titkosított szöveg birtokába jut, amit még vissza is kell fejtenie, ami a kulcs ismerete nélkül nehéz feladat lehet.

2.5. Támadásfajták

Egy információs rendszerhez való csatlakozás és a támadó lehetőségei alapján kétféle támadást különböztethetünk meg: a passzív és az aktív támadást [12].

A passzív támadás esetén a behatoló hozzájut az adatokhoz, képes a kommunikáció lehallgatására, adatokat gyűjthet, de megváltoztatni semmit nem tud, vagy nem akar. A támadás lényege az észrevétlen megfigyelés és adatgyűjtés. A történeti részben említett, a II. világháborúban az angolok által végzett adatgyűjtés tipikusan ilyen támadási forma volt, számukra létfontosságú volt, hogy a német fél ne sejtse meg, hogy ők képesek visszafejteni az Enigma segítségével titkosított üzeneteket.

Az aktív támadás esetén a támadó olyan módon csatlakozik a rendszerhez, hogy ott képes adatokat hamisítani, és a résztvevők nevében hamis üzeneteket küldeni, tranzakciókat indítani. A támadó ilyenkor beépül a kommunikáció folyamatába, az üzeneteket elnyeli, és az eredeti felek helyett válaszol mindkét irányba [13].

1. ábra. A passzív támadás modellje [14]

2.6. Történelmi példák

A történelem során a kriptográfiát és a szteganográfiát elsősorban katonai és diplomáciai üzenetek titkosítására, illetve elrejtésére használták.

Már az ókorból is ismerünk példákat az adatrejtési technikák alkalmazására. Hérodotosz, az egyik első ismert történész, aki a pater historiae (a történelem atyja) néven is ismert [15], A görög-perzsa háború című művében így írt:

„Arisztagorasz tehát nem tudta beváltani Artaphrenésznak tett ígéretét, egyre jobban szorongatták a sereggel kapcsolatos kiadások, bántotta a hadjárat kudarca, valamint Megabatésszel való meghasonlása, és félt, hogy elveszthetti az uralmat Milétoszban. Ilyen aggodalmaktól szorongattatva azon volt, hogy lázadást szervez. S véletlenül éppen akkor érkezett hozzá egy küldönc Hisztiaiosztól, és a fejbőrére írva azt az üzenetet hozta, hogy Arisztagorasz lázadjon fel a király ellen. Hisztiaiosz sehogy sem tudta biztonságosan megüzenni Arisztagorasznak, hogy szervezzen pártütést a király ellen, mert az utakat szigorúan ellenőrizték. Simára borotváltatta hát leghívebb szolgája fejét, ráírta az üzenetet,

megvárta, míg a szolga haja újra kinő, s amikor kinőtt, elküldte a szolgát Milétoszba, de semmilyen üzenetet nem bízott rá, csak azt, hogy ha megérkezik Milétoszba, kérje meg Arisztagoraszt, hogy nyírassa le a haját, és vizsgálja meg a fejét. Az írás, mint már mondtam, lázadásra buzdított. Ezt pedig azért tette Hisztiaiosz, mert nagyon búsult, hogy Szuszában tartják, és azt remélte, hogy ha Milétoszban lázadás tör ki, Dareiosz őt küldi el a tengerpartra. Ha azonban nem történik semmi, soha nem térhet vissza Milétoszba." [16]

A görög-perzsa háború hetedik könyvében is találhatunk egy ötletes megoldást arra, hogy hogyan lehetett a ma ismert elektronikus eszközök nélkül rejtett üzeneteket küldeni:

„Most pedig rátérek arra, ami elbeszélésemből korábban kimaradt. Először a lakedaimóniak értesültek róla, hogy a király Hellasz ellen készülődik. Ekkor küldtek el Delphoiba jóslatért, ahol a már ismertetett választ kapták. A hír pedig a következő, különös módon jutott el Lakedaimónba. A médekhez menekült Démaratosz, Arisztón fia véleményem szerint – amit a tények is támogatnak – nemigen kedvelte a lakedaimóniakat, így eltöprenghetünk rajta, hogy jóindulatból vagy kárörömből tette-e, amit tett. Amikor ugyanis Xerxész úgy döntött, hogy hadjáratot indít Hellasz ellen, a Szuszában időző Démaratosz elhatározta, hogy hírt küld róla a lakedaimóniaknak. Minthogy azonban az üzenet eljuttatására nem volt más módja – mert könnyen leleplezték volna –, a következőt eszelte ki. Vett egy két rétegből álló írótáblát, levakarta róla a viaszt, a fára felírta, hogy mit tervez a király, majd újra bekente a táblát viasszal, hogy az üzenetvivő ne keltse fel a táblával az utat őrzők gyanúját. Így jutott el a tábla Lakedaimónba. A lakedaimóniak nem tudtak mit kezdeni vele, míg végül – legalábbis úgy mesélték – Gorgó, Kleomenész leánya, Leónidasz felesége meg nem fejtette a rejtélyt: azt tanácsolta, hogy kaparják le a viaszréteget, s ott majd megtalálják az üzenetet, a fára írva. A lakedaimóniak megfogadták a tanácsot, elolvasták az üzenetet, aztán tudatták a többi hellénnel is. A hagyomány szerint így történt a dolog." [17]

Nem csak a görögök, hanem a rómaiak is használtak titkosítási módszereket. Gaius Suetonius Tranquillus A caesarok élete című történelmi művében így ír Iulius Caesar kulcsírásáról:

„Megvannak Ciceróhoz, valamint bizalmas embereihez magánügyekben írott levelei is,

melyekben ha titkos dolgot közölt, kulcsírást alkalmazott, vagyis úgy állította össze a betűk sorrendjét, hogy értelmes szót nem adtak ki; ha valaki ezt az írást meg akarja fejteni, az helyettesítsen minden betűt az ábécé sorrendjében három hellyel előtte állóval, tehát például a D-t az A-val, és így tovább." [18]

Az isteni Augustusról szóló könyvében megtalálhatjuk az előző módszer egy kis mértékben módosított változatát:

„Ha kulcsírást használt, A helyett B, B helyett C betűt és így sorban mindig a betűsor következő jegyét írta le, X helyett pedig két A-t." [19]

Az ókori időkre való visszatekintés után a XX. és a XXI. század történetéből szeretnék néhány olyan példát hozni, amelyek megmutatják, hogy milyen problémát okozhat az, ha az elküldött üzenetek illetéktelen, ellenséges kezekbe kerülnek, illetve az, ha a saját elhárításnak nem sikerül időben felfednie az ellenség szándékait.

A tannenbergi csata 1914-ben, az első világháború kezdetén, a keleti fronton zajlott le az orosz és a német csapatok között. Az orosz 1. és 2. hadsereg, illetve a német 8. hadsereg állt egymással szemben. Az élőerő létszámát tekintve jelentős, több, mint kétszeres orosz fölényről beszélhetünk. A csata az orosz túlerő ellenére döntő német győzelemmel zárult, melyben jelentős szerepet játszott a magasabb színvonalú, Paul von Hindenburg és Erich Ludendorff nevével fémjelzett császári hadvezetés, míg orosz részről az is nehezítette az együttműködést, hogy a két hadsereg parancsnokai, Alekszandr Szamszonov és Paul von Rennenkampf jószerivel még beszélni sem volt hajlandók egymással. Az orosz véres veszteség kb. 50 000 főre rúgott, és 90 000 fő hadifogságba került. Ezzel szemben a német oldal véres vesztesége kb. 10 000 fő volt [20]. A német győzelemben bizonyos – de nem túlértékelendő – mértékben közrejátszott, hogy a 8. hadsereg az orosz fél óvatlan rádióforgalmazása miatt rendszeresen fontos információk birtokába jutott. Az oroszok sok esetben mindenféle rejtjelzés nélkül küldték el rádión a szándékaikat taglaló üzeneteket, amelyeket a német híradó-alegységek különösebb probléma nélkül le tudtak hallgatni [21].

A midwayi csata a második világháború egyik döntő, tengeren vívott csatája volt, amelyet az Egyesült Államok és Japán jelentős tengeri erői vívtak 1942 júniusában. A japán csapatok az Egyesült Államok által birtokolt Midway-atollra kívántak csapást mérni,

méghozzá 4 repülőgép-hordozó, 7 csatahajó és számos kisebb hadihajó felvonultatásával [22]. A Nagumo tengernagy számára készített felderítő jelentés leszögezte, hogy „Az ellenség nem tud terveinkről” [23]. Ez a feltevés nem volt igaz, ugyanis az amerikai hírszerzés Joseph J.Rochefort fregattkapitány vezette csoportja képes volt a japánok által használt JN-25 rejtjelkulcs visszafejtésére. Ennek következtében már jó előre meg tudták mondani, hogy ellenséges támadás várható, viszont először nem tudták meghatározni, hogy mi lesz a konkrét célpont. Erre a helyre ugyanis a japánok AF-ként hivatkoztak, és amerikai részről csak valószínűsíteni lehetett, hogy Midway-ről van szó. Ezért utasították a midwayi támaszpontot, hogy adjanak le egy megtévesztő angol nyelvű rádióüzenetet rejtjelzés nélkül, amely arról szól, hogy elromlott a frissvíz-ellátó berendezésük. Két nappal később aztán lehallgattak egy japán rádióüzenetet, amely jelentette, hogy AF-nek kevés a friss vize [24]. Így az amerikai fél a többi üzenet birtokában meg tudta határozni, hogy hol és mikor várható a támadás, ezért erre a helyre tudta koncentrálni a rendelkezésére álló erőket. A csata súlyos japán vereséggel zárult: elsüllyedt 4 repülőgép-hordozó, elvesztettek 228 repülőgépet, jórészt a harcedzett és nehezen pótolható személyzettel együtt.

A II. világháború egy másik, ma már elég részletesen ismert epizódja az Enigma nevű elektromechanikus titkosító és visszafejtő berendezés feltörésének esete volt. Az Enigmát a német haderő használta kommunikációjának titkosítására. Nagy-Britannia számára létfontosságú volt, hogy le tudja hallgatni és vissza tudja fejteni az ellenség üzenetváltásait, többek között a német tengeralattjárók által a brit kereskedelmi flotta ellen vívott tonnaháborúban elért sikerek miatt, amely komolyan veszélyeztette a szigetország ellátását. Ehhez segítségükre volt a lengyel titkosszolgálatnak dolgozó három matematikus, Marian Rejewski, Jerzy Rúzycki és Henryk Zygalski munkássága [25], akik már az 1920-as évek végén megkezdték az Enigma akkor változatának vizsgálatát. Elért eredményeiket és egy katonai Enigma-másolatot 1939 júliusában átadták az angol félnek [26]. A kódfejtő munkák a Londontól 40 mérföldre található Bletchley Park-ban folytatódtak az ULTRA projekt fedőnév alatt, ahol Alan Turing matematikus irányításával – és közel 10 ezer munkatárs támogatásával – sikerült egy Bombe nevű elektromechanikus számítógépet építeni, amellyel már lehetővé vált az elfogott német üzenetek többségének visszafejtése. 1943-ban havonta 80 000 üzenetet törtek fel, ez átlagban kb. napi 2600 üzenet

visszafejtését jelenti. Az ULTRA projekt pontos hatásáról a mai napig viták folynak, de az általánosan elfogadott nézet szerint a németek titkosításának feltörése két évvel rövidítette meg a háborút [27]. Érdekesség, hogy az angolok egészen az 1970-es évekig olyan mértékben titkolták az Enigma feltörésének tényét, hogy a német fél nem is sejtette, hogy ellenfelük képes volt visszafejteni haderejük kommunikációját [28].

A 2001. szeptember 11-i terrortámadás (ismert nevén a 9/11) a XXI. század kezdetén az Amerikai Egyesült Államok ellen irányult. Az al-Kaida terroristái eltérítettek négy nagy méretű Boeing utasszállító gépet, majd kettőt ezek közül a New York-beli World Trade Center ikertornyainak vezettek, melyek kigyulladtak, majd összeomlottak. A harmadik gép a Pentagon épületébe csapódott. A negyedik gép, melynek célpontja valószínűleg a Fehér Ház lett volna, az utasok közbelépése után Shanksville mellett a földbe csapódott. A merénylet közel háromezer áldozatot követelt [29]. Ebben az esetben az amerikai hírszerzés nem tudta időben megszerezni a terroristák merényletre vonatkozó terveit. A későbbi nyomozások során felmerült, hogy az al-Kaida emberei a támadások tervezése során az egymás közti kommunikációjukban rejtett üzenetekkel érintkeztek. Ezek felfedésére az események tükrében a bűnüldöző szervek nem voltak megfelelően felkészülve [30].

Mint ahogyan az a kiragadott példákból is látható, a kriptográfia és a kriptoanalízis, illetve a szteganográfia és a szteganalízis komoly történelmi eseményeket befolyásoló erővel bírt, és bír a mai napig is. A régmúlthoz képest lényeges változást jelent az, hogy a XX. század közepén megjelentek a számítógépek, így a titkosítást ma már nem csak kézi úton vagy elektromechanikus berendezésekkel lehet elvégezni, hanem nagy számítási teljesítményt biztosító gépekkel is. Így a megelőző évszázadok lehetőségeihez képest sokkal bonyolultabb titkosítási és visszafejtési módszereket lehet bevetni az információ titkosságának megőrzésére, illetve az információhoz történő jogosulatlan hozzáférésre.

3. A felhasznált technológiák bemutatása

Jelen fejezetben a szakdolgozat készítése során felhasznált környezetet és fejlesztői eszközöket ismertetem.

3.1. A .NET Framework

A .NET Framework (keretrendszer) a Microsoft által készített környezet, egy olyan technológia, amely lehetővé teszi, hogy a hozzá kapcsolódó nyelveken (mint pl. a későbbiekben ismertetett C#) fejlesztett alkalmazásokat és webszolgáltatásokat futtassunk Windows operációs rendszereken. A .NET részét képezi a Common Language Runtime (CLR), amely a programok futási környezetét biztosítja, illetve egy nagy méretű osztálykönyvtárat is találhatunk benne, amely rengeteg jól használható osztályt biztosít a programfejlesztők számára. Számos olyan alaposan letesztelt és megbízhatóan működő szolgáltatást találunk benne készen, amelyeket más fejlesztői eszközök használata esetén magunknak kellett kifejleszteni, vagy csak hosszas keresgélés után találhattunk egy más programozók által készített, gyakran nem igazán megbízható és elavult, nem követett verziót.

A .NET Framework első kiadása 2002-ben jelent meg. Azóta a Microsoft több verziót adott ki a keretrendszerből, jelen pillanatban (2015) a 4.5.2-es az utolsó változat. A szakdolgozat készítése során a .NET Framework 4.5-öt használtam.

3.2. A C# programozási nyelv

A C# a Microsoft által létrehozott modern, általános célú programozási nyelv, amely szorosan összefügg a szintén a Microsofthoz kapcsolódó .NET keretrendszerrel. A nyelv béta változata 2000 júniusában jelent meg, az első hivatalos kiadás pedig 2002-ben látott napvilágot. A C# egy C-szerű nyelvnek tekinthető, gyökerei egészen a C, illetve a C++ programozási nyelvig nyúlnak vissza, de erősen érezhető rajta a Java hatása, illetve a Delphiben alkalmazott elemekkel is találkozhatunk. Ez utóbbi nem véletlen, ugyanis a nyelv egyik fő fejlesztője Anders Hejlsberg, aki a Borland cégen belül részt vett a Turbo

Pascal és a Delphi megalkotásában. A C# folyamatos fejlődésen megy keresztül, rendszeresen kerülnek bele új lehetőségek, és jelen pillanatban (2015) az 5.0-s verziónál tart.

A C# nyelv specifikációját a Microsoft benyújtotta szabványosítás céljából az ECMA International nevű szervezethez. A C#-hoz és a .NET keretrendszerhez elsősorban a Microsoft nevét és a Windows operációs rendszert szokás kapcsolni, de – részben a szabványosításnak köszönhetően – más operációs rendszereken, pl. Linuxon is léteznek hasonló megoldások, mint például a Mono Project. Az említett megoldások hátránya az, hogy a fejlesztés üteme elmarad a Microsoft által diktált tempótól, így a .NET konkurensei a Microsoft-féle verziónak csak egy szűkebb részhalmazát képezik. Miután a Microsoft 2014-ben megnyitotta a .NET keretrendszer forrásának egy részét [31], a Mono is építkezhet a Microsoft-féle .NET kódokra [32].

Megjegyzendő, hogy a C#, mint nyelv sem egyedüli a .NET fejlesztéshez használható nyelvek között, ugyanis számos egyéb, a .NET keretrendszerhez illeszkedő megvalósítás is létezik. Erre példa a szintén a Microsoft által készített Visual Basic, de az Embarcadero-féle Delphinek, vagy az eredetileg az 1950-es évek végén megjelent COBOL-nak is létezik .NET-hez használható verziója.

A fejlesztéshez azért választottam a C# nyelvet, mert több éves tapasztalattal rendelkezem a használatában, és kellően rugalmas ahhoz, hogy a .NET keretrendszer által biztosított technológiákat felhasználva létrehozzam a szakdolgozat témájául választott grafikus felületű programot.

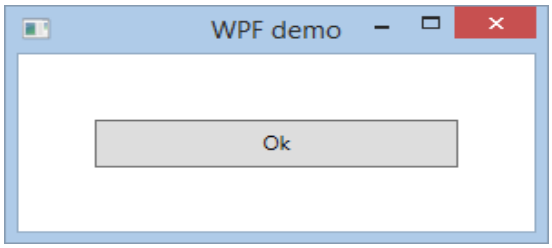
3.3. Windows Presentation Foundation, WPF

A Windows Presentation Foundation (WPF) a .NET keretrendszer része, egy grafikus alrendszer és alkalmazásprogramozási felület (API), amelynek segítségével Windows alapú asztali alkalmazások szolgáltatásgazdag grafikus felhasználói felületét (Graphical User Interface, GUI) készíthetjük el. A WPF a felület megjelenítéséhez a DirectX-et használja, így az alkalmazásaink kihasználhatják a videokártyák által biztosított hardveres gyorsítást.

A WPF előnye a régebben használt grafikus felületekhez, pl. a Windows Formshoz képest, hogy sok olyan funkció, amely a WPF-ben viszonylag könnyen megvalósítható, a megelőző rendszerekben csak nehezen volt elérhető. A WPF segítségével könnyen építhetünk rugalmas felhasználói felületeket, és sok egyéb grafikus lehetőség (átméretezhető grafikák, animációk, a webfejlesztésből ismert CSS-hez hasonló elven használható stílusok stb.) is elérhető a használatával. A keretrendszer ezeket túl számos kész komponenst – mint például nyomógombok, rádiógombok, szerkesztőmezők, panelek stb. – bocsát a fejlesztők rendelkezésére. A felhasználói felület leírása a WPF használata esetén deklaratív módon, egy XML-alapú nyelvvel, a XAML-lel (eXtensible Application Markup Language) történik. A következő keretben olvasható egy rövid XAML nyelvű kódpélda, amely jól illusztrálja a leírónyelv szerkezetét. A leírásban meghatározzuk az ablak (Window) feliratát és méreteit, illetve létrehozunk egy Ok feliratú nyomógombot (Button), amelynek van egy eseménye, amely a gomb lenyomása esetén következik be.

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WPF demo" Height="150" Width="300">
    <Grid>
        <Button Name="buttonOk" Content="Ok" Width="200" Height="30"
                Click="buttonOk_Click" />
    </Grid>
</Window>
```

A kód eredményeképpen a következő ablak jelenik meg:



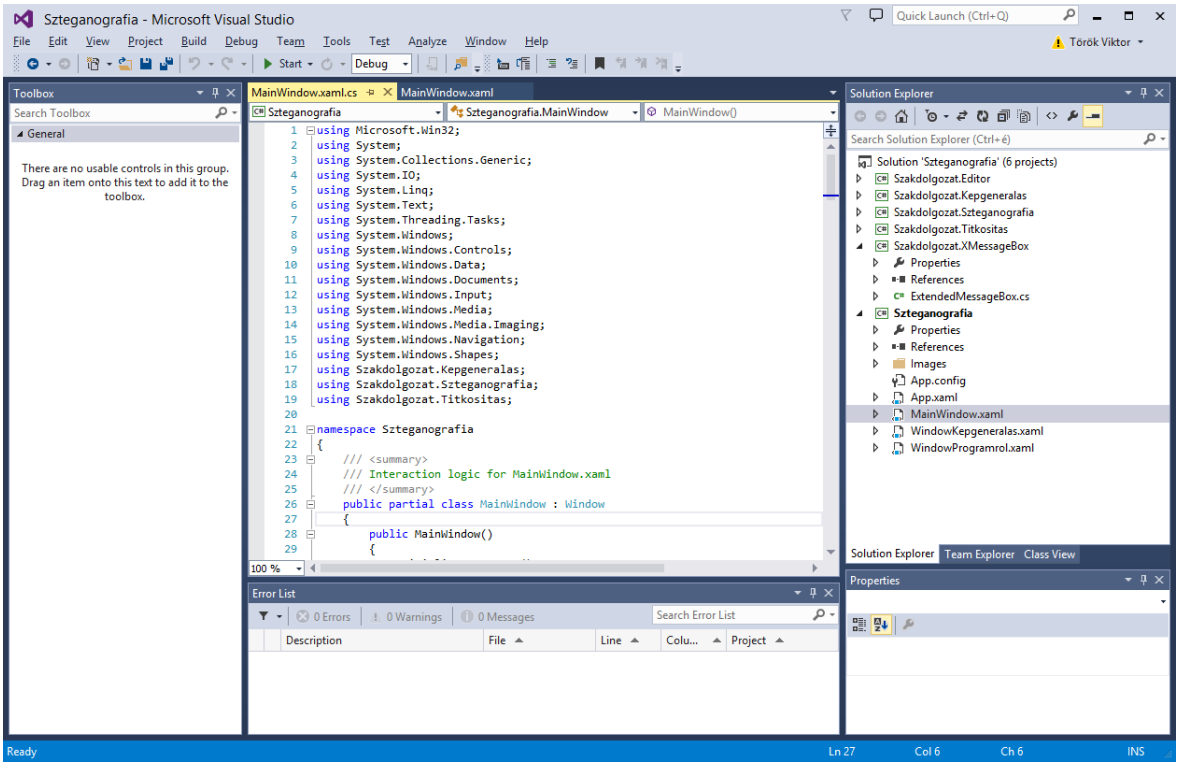
2. ábra. WPF Demo ablak

3.4. Microsoft Visual Studio Professional 2013

A Microsoft Visual Studio egy integrált fejlesztői környezet (Integrated Development

Environment, IDE), mely az évek során több különböző verziószámú kiadást ért meg. A legújabb stabil állapotú verzió jelenleg a Microsoft Visual Studio 2013. A fejlesztői környezetnek több elérhető változata létezik, melyek között ingyenes és fizetős verziókat is találhatunk. A fejlesztés során a DreamSpark program keretében főiskolai hallgatók számára ingyenesen letölthető Microsoft Visual Studio Professional 2013 változatot használtam, mely támogatja az előzőekben ismertetett C# nyelvű, WPF technológiát használó programok készítését.

A Visual Studio előnye – mint ahogyan az általában az integrált fejlesztői környezetekről általában elmondható –, hogy egy keretbe foglalja a forráskód szerkesztését, a program fordítását, összeszerkesztését és futtatását, és szükség szerint egyéb funkciókat (nyomkövetés, tesztelés) is elérhetővé tesz. A szerkesztő szintaxiskiemelést, automatikus kódkiegészítést (IntelliSense) és kódformázási lehetőségeket biztosít, amely jelentős mértékben megkönnyíti a programozók munkáját. Ezek az előnyök elmondhatók a Visual Studio ingyenes verzióiról is, míg a fizetős verziók további hatékonyságnövelő eszközöket is tartalmaznak.



3. ábra. A Microsoft Visual Studio Professional 2013

3.5. A fejlesztés során használt egyéb programok

Grafikus felületű programok fejlesztése során gyakran van szükség különböző grafikai elemek szerkesztésére. A szakdolgozat készítéséhez a Corel Paint Shop Pro X7 programot használtam, amely egy olcsó, ám mégis sokoldalú, számos képfeldolgozási lehetőséget biztosító szoftver.

Az UML diagramok elkészítéséhez a Visual Paradigm program Community Edition változatát használtam.

A szakdolgozat szövegének szerkesztését az Apache OpenOffice 4.0.1 programcsomag OpenOffice Writer programjával végeztem el.

4. Az LSB algoritmus ismertetése

4.1. Az adatrejtéshez használt képek szerkezete

A szakdolgozatban elkészített program az RGB színteret alkalmazó képeket képes felhasználni az adatrejtési folyamat során.

A fénysugárzó testek elsődleges és másodlagos fényforrások lehetnek. Az első csoportba azok tartoznak, amelyek önállóan képesek fény kibocsátására, mint pl. a Nap vagy a számítógépes monitorok. A másodlagos fényforrások azok a testek, amelyek önállóan nem bocsátanak ki fényt, csak visszaverik a rájuk eső fényt [33].

A színkeverésnek két alapvető módját ismerjük, attól függően, hogy elsődleges vagy másodlagos fényforrást szeretnénk modellezni. Az additív vagy összeadó színkeverésnél a vörös (Red), a zöld (Green) és a kék (Blue) alapszínekből vett meghatározott mennyiségeket adunk össze, és így kapjuk a különböző színárnyalatokat. Ezzel a módszerrel az elsődleges fényforrások színeit tudjuk előállítani. A szubsztraktív vagy kivonó színkeverés esetén a ciánkék, bíborvörös és sárga színekből állítjuk elő a kívánt színeket. Ezzel a másodlagos fényforrásokat lehet modellezni [34].

Az RGB színtér a vörös, a zöld és a kék alapszínekből kikeverhető színeket tartalmazza, és az additív színkeverés modellezésére használjuk. A számítógépes grafikában gyakran használjuk a háromcsatornás színkódolást, amikor a három RGB alapszín intenzitását 3 x 8, azaz összesen 24 biten kódoljuk. Így egy-egy összetevő ábrázolása 1-1 byte-on történik, tehát az R, G, B komponensek értéke 0 és 255 közé eshet. Az RGB színtér mellett használjuk még az RGBA színteret is, ahol az A az alfa csatornát jelenti, amely az átlátszóságot kódolja [35].

4.2. Képtömörítés

Ha egy nagy méretű képet az előző pontban leírt módszerrel 3 byte / pixel vagy 4 byte / pixel helyigénnyel tárolunk, akkor eredményül igen nagy méretű állományokat kapunk. Ilyen képformátum lehet pl. a Microsoft által kifejlesztett BMP (Bitmap), amelynek létezik

olyan verziója, amely egyáltalán nem tömöríti a képet [36].

Ennek a problémának a megoldására dolgozták ki a különböző képtömörítési eljárásokat, melyek egyrészt csökkentik az igénybe vett tárolóhely méretét, másrészt pedig az adatátviteléhez szükséges időt. Ezeknek a módszereknek a vizsgálata lényeges az LSB algoritmus szempontjából.

A képtömörítési eljárás lehet veszteségmentes, amely a kép összes információját megőrzi, azaz a tömörített adathalmazból teljes pontossággal visszaállítható az eredeti kép. Ilyen képformátum például a PNG (Portable Network Graphics), amely veszteségmentességen túl képes RGB képek tárolására is [37]. Az LSB algoritmus a PNG-hez hasonló veszteségmentes módon tömörített képek esetén alkalmazható.

Használhatunk még veszteséges képtömörítési eljárásokat is, amelyek a veszteségmentes eljárásokkal ellentétben nem teszik lehetővé a tömörített adatból az eredeti kép pixelre pontos rekonstrukcióját, ám egy elfogadható rekonstrukciót igen [38]. Ezek a módszerek az emberi érzékelés szempontjából nem jelentenek problémát, ugyanis az állomány méretének a teljes mérethez képest történő komoly csökkenése mellett sem lép fel általában szemmel érzékelhető minőségromlás, ám a szteganográfiai algoritmusok egy részének eredményét az ilyen tömörítési eljárások tönkreteszik. Ilyen képformátum például a JPEG (Joint Photographic Experts Group) [39]. A JPEG-gel tömörített képek esetén is lehet használni szteganográfiai módszereket, mint pl. a JSteg [40], de az LSB algoritmus veszteséges tömörítés esetén nem alkalmazható.

4.3. Az adatok fejléce

Egy adott képben található titkosított és elrejtett információk visszafejtéséhez a következő adatokat célszerű eltárolni:

- a titkosított adatok hossza byte-ban számolva
- az esetlegesen használt titkosító algoritmus típusa

4.4. Az LSB algoritmus

Az LSB-algoritmus megnevezésében az LSB a Least Significant Bit kifejezés rövidítése. Ez egy kettes számrendszerben ábrázolt szám esetén a jobb szélen, a nulladik pozíción található bitet jelenti. A bal szélen, a hetedik pozíción elhelyezkedő bitet MSB-nek, vagy Most Significant Bit-nek szokták nevezni.

Bitpozíció	7.	6.	5.	4.	3.	2.	1.	0.
Helyiérték	128	64	32	16	8	4	2	1
Az 1 szám kettes számrendszerben	0	0	0	0	0	0	0	1
Rövidítés	MSB							LSB

Legyen adott egy bájt, amelynek a tartalmát el szeretnénk rejteni egy bájtokból álló tömbben. Ekkor a bájt minden egyes bitjét a hordozó tömb egy-egy bájtjának LSB bitjében lehet elrejteni, azaz 1 bájt tartalmának elrejtéséhez a hordozó tömb 8 bájtjának felhasználására lesz szükség. Legyen például az elrejtendő adat a következő (kettes számrendszerben ábrázolva):

1	0	0	0	1	1	1	1

A rendelkezésünkre álló 8 elemű tömb álljon a következő táblázatban megadott számokból, ahol az első oszlop a tömb elemének az indexét tartalmazza, a többi oszlop pedig a tömbben található számok egyes bitjeit:

Az elem indexe								**LSB**
0	0	0	0	0	0	0	0	**1**
1	0	0	0	0	0	0	0	**0**
2	0	0	0	0	0	0	0	**0**
3	0	0	0	0	0	0	0	**0**
4	1	1	1	1	1	1	1	**1**
5	1	1	1	1	1	1	1	**1**
6	1	1	1	1	1	1	1	**1**
7	1	1	1	1	1	1	1	**1**

Az LSB-algoritmus futása során végighalad az elrejtendő bájt bitjein jobbról balra haladva, és azokat beírja a tömb elemeinek LSB pozíciójába. A sikeres futás után a tömb a következő adatokat fogja tartalmazni:

								LSB
0	0	0	0	0	0	0	0	**1**
1	0	0	0	0	0	0	0	**1**
2	0	0	0	0	0	0	0	**1**
3	0	0	0	0	0	0	0	**1**
4	1	1	1	1	1	1	1	**0**
5	1	1	1	1	1	1	1	**0**
6	1	1	1	1	1	1	1	**0**
7	1	1	1	1	1	1	1	**1**

Mint ahogyan az a példából is látható, a tömb elemei nem feltétlenül módosulnak, ugyanis ha az elrejtendő bit és a tömbben található szám LSB-je megegyezik, akkor az eredeti adat változatlan marad.

Ez a módszer jól használható az RGB színeket tartalmazó képeknél, vagy a WAV formátumú hangállományoknál, mivel itt az egyes bájtok kis mértékű módosítása az emberi látás és hallás számára észrevehetetlen változásokat eredményez. Nem alkalmazható ellenben az LSB módszer például az ASCII kódtábla alapján kódolt szövegfájloknál, mivel ha egy adott karakter kódja csak egyetlen biten is megváltozik, az az olvasható szöveg könnyen észrevehető módosulásával jár együtt.

Az LSB-módszer továbbfejlesztett változatában nem csak a jobb oldali bitet használjuk fel az adatrejtésre, hanem – a bájtban jobbról elindulva – kettőt vagy többet. Ezzel a módszerrel viszont óvatosnak kell lenni, mivel a több bit módosítása az egybites esethez képest nagyobb torzulásokkal jár együtt a hordozó képben vagy hangállományban, ami előbb-utóbb a külső szemlélő számára is érzékelhető lesz.

5. A szteganográfiai program terve

A szteganográfiai programnak alapvetően két típusú felhasználója van:

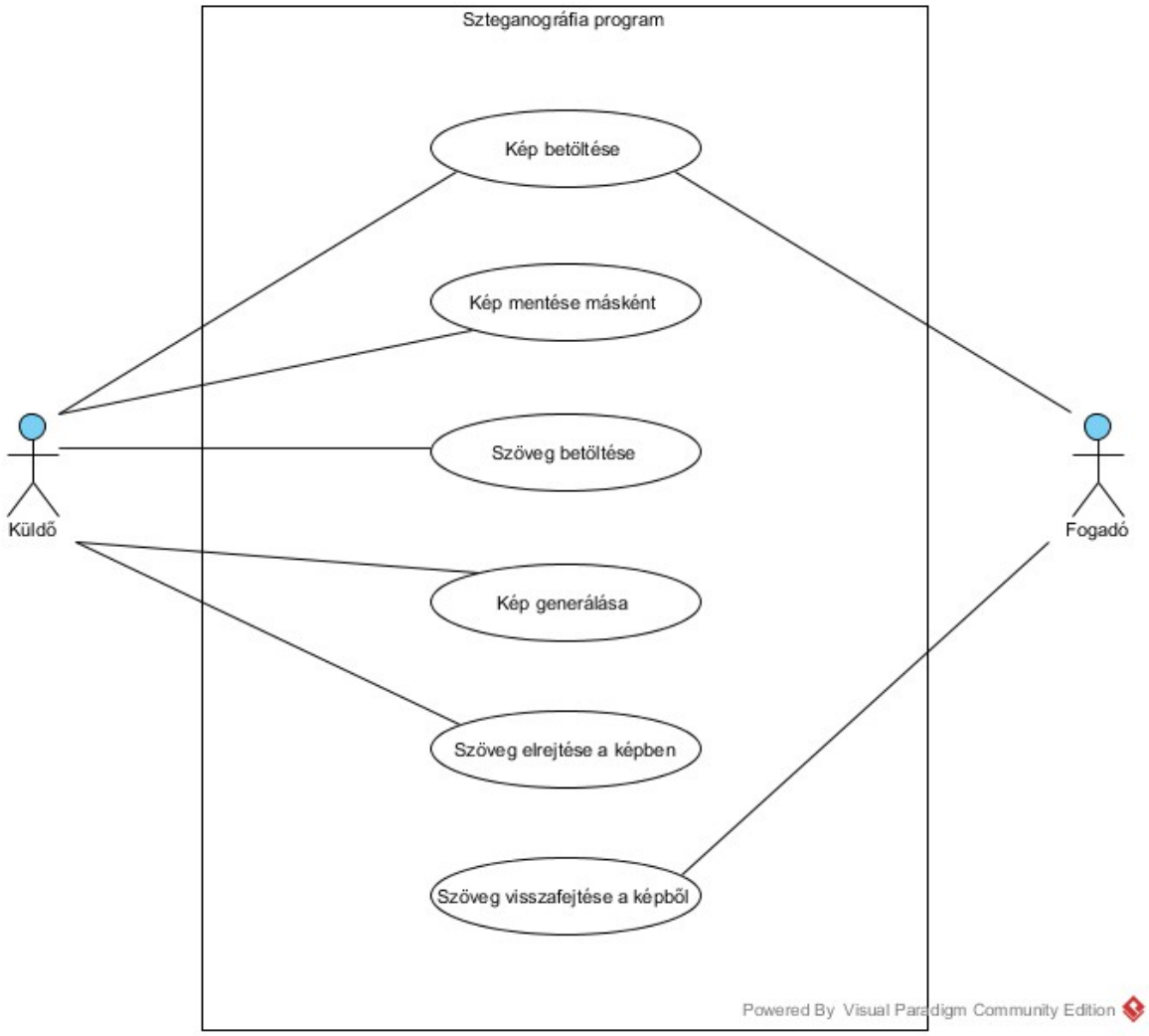
- a küldő fél, aki az üzenetet megfogalmazza, titkosítja, elrejti egy képben, majd elküldi a fogadó félnek
- a fogadó fél, aki a képben megkapott üzenetet visszafejti

A programnak a következő funkciókat kell biztosítania a küldő fél számára:

- Kép betöltése: egy olyan kép betöltése, amelyben nincsenek rejtett információk. A kép formátuma lehet JPG, PNG, BMP és TIF.
- Kép mentése másként: a titkosított és rejtett információkat tartalmazó kép lemezre történő mentése. A mentett kép nem lehet olyan formátumú, amely veszteséges tömörítést használ. A program a BMP formátumot képes kezelni.
- Szöveg betöltése: egy szövegfájl betöltése a lemezről, amelyet titkosítani és rejteni szeretnénk. A betöltött szövegnek szerkeszthetőnek kell lennie, azaz a felhasználónak legyen lehetősége arra, hogy szöveget írjon be egy beviteli mezőbe, illetve egy betöltött szöveget módosítson.
- Kép generálása: a tesztelés során hasznos lehet, ha a program képes tesztképeket generálni (pl. egyszínű kép, zajos kép, Mandelbrot halmaz stb.).
- Szöveg elrejtése a képben: a beviteli mezőben található szöveg titkosítása és rejtése.

A programnak a következő funkciókat kell biztosítania a fogadó fél számára:

- Kép betöltése: egy olyan kép betöltése, amelyben valószínűsíthetően rejtett információk találhatók. A program visszafejtési funkciója a BMP formátumot képes kezelni.
- Szöveg visszafejtése a képből: a képben található információk visszafejtése.



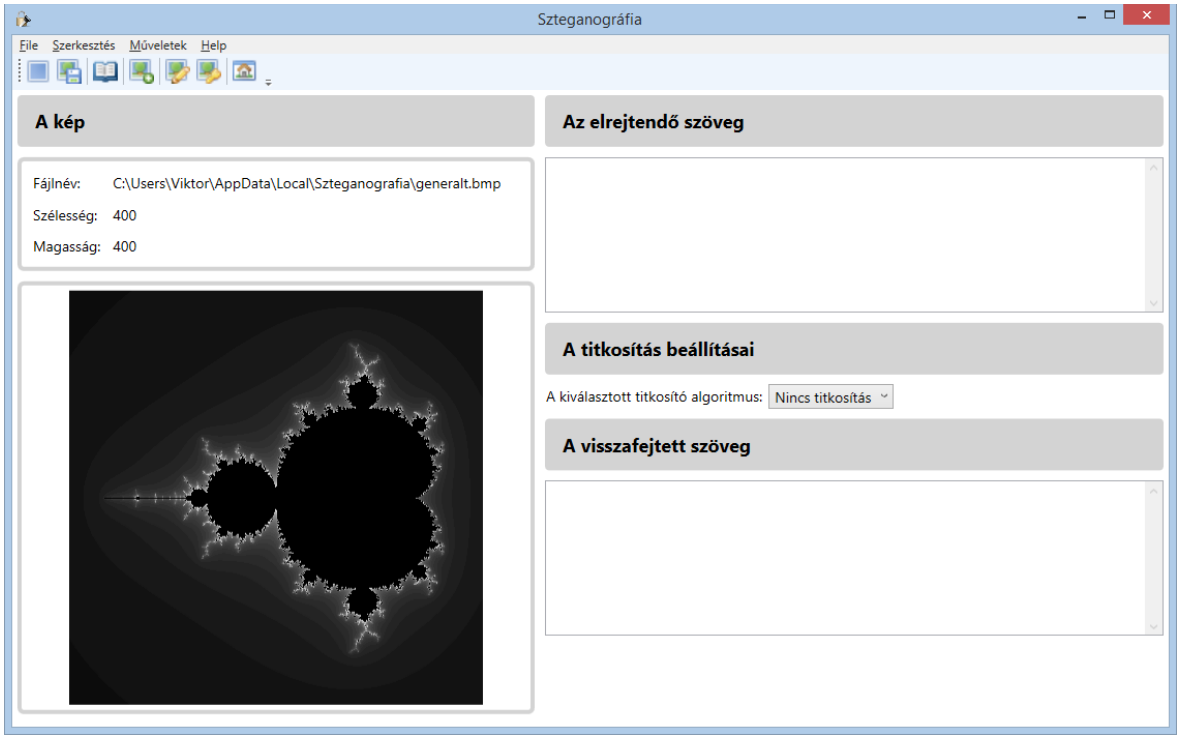
4. ábra. Use Case Diagram

6. A szteganográfiai program használata

6.1. A program általános felépítése

A program használatához a Szteganografia.exe nevű futtatható állományt kell elindítani.

A program sikeres indítása után az alábbi képernyő jelenik meg:



5. ábra. A program kezdőképernyője

A képernyő több részre osztható:

- Felső menüsor: az egyes menükből a program bármely funkcióját el tudjuk érni.
- Felső toolbar: a toolbar ikonjai segítségével néhány – a gyakrabban használtak közül kiválasztott – funkciót érhetünk el. Az egyes gombok funkcióinak azonosítását tooltip-ek segítik, amelyek akkor jelennek meg, ha az egérkurzort a gomb felé visszük.

- A kép: a lemezről betöltött kép és a kép alapvető adatai (fájlnév, pixelben mért szélesség és magasság) a képernyő bal oldalán jelennek meg. Az oszlopszélességet meghaladó méretű kép a megfelelő megjelenítés érdekében átméretezésre kerül.
- Az elrejtendő szöveg: ebben a szerkeszthető szövegdobozban jelenik meg az elrejtendő szöveg. A szöveget betölthetjük lemezről, illetve kézzel is tudjuk módosítani.
- A titkosítás beállításai: az itt látható lenyíló lista segítségével határozhatjuk meg, hogy milyen titkosítási algoritmust szeretnénk használni az adatrejtés előtt.
- A visszafejtett szöveg: ebben a csak olvasható szövegdobozban jelenik meg a betöltött képből visszafejtett szöveg.

6.2. A File menü

A File menü a következő funkciókat tartalmazza:

- Kép betöltése: a menüpont segítségével képeket tölthetünk be a lemezről. A program a JPG, PNG, BMP és a TIF formátumú képeket kezeli. Ha elrejtett szöveget szeretnénk keresni, akkor csak a BMP formátumú képek jöhetnek szóba.
- Kép mentése másként: miután elrejtettünk egy szöveget egy adott képben, az eredményt lementhetjük egy általunk meghatározott nevű, BMP formátumú állományba.
- Szöveg betöltése: egy TXT kiterjesztésű szövegfájlt olvashatunk be a lemezről.
- Kilépés a programból: a menüpont meghívásával kiléphetünk a programból.

6.3. A Szerkesztés menü

A Szerkesztés menü a szövegszerkesztőkben megszokott funkciókat tartalmazza: visszavonás, ismét, kivágás, másolás és beillesztés. Ezeket a műveleteket a programnak az elrejtendő szöveget tartalmazó szerkeszthető szövegdobozában használhatjuk.

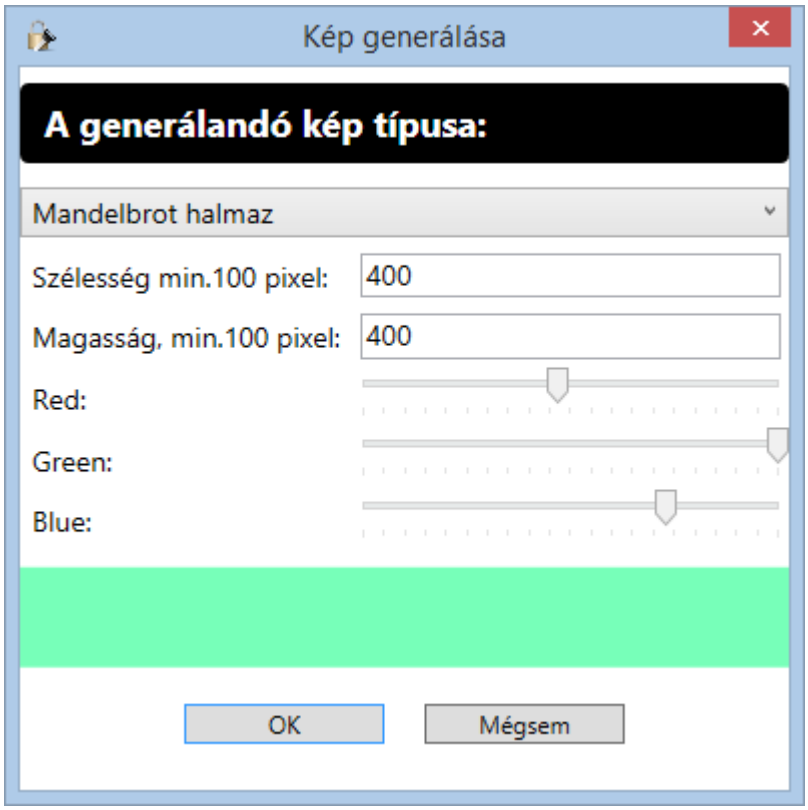
6.4. A Műveletek menü

A Műveletek menüből elérhető funkciók a program legfontosabb részét képezik, ezért ezeket részletesen ismertetem.

6.4.1. Kép generálása

Ha esetleg nem áll rendelkezésünkre olyan kép, amelyben elrejthetnénk a szöveget, akkor ennek a menüpontnak a segítségével létrehozhatunk néhány speciális típusba sorolható képet.

A menüpont elindítása után a következő dialógus ablak jelenik meg:



6. ábra. A Kép generálása dialógus ablak

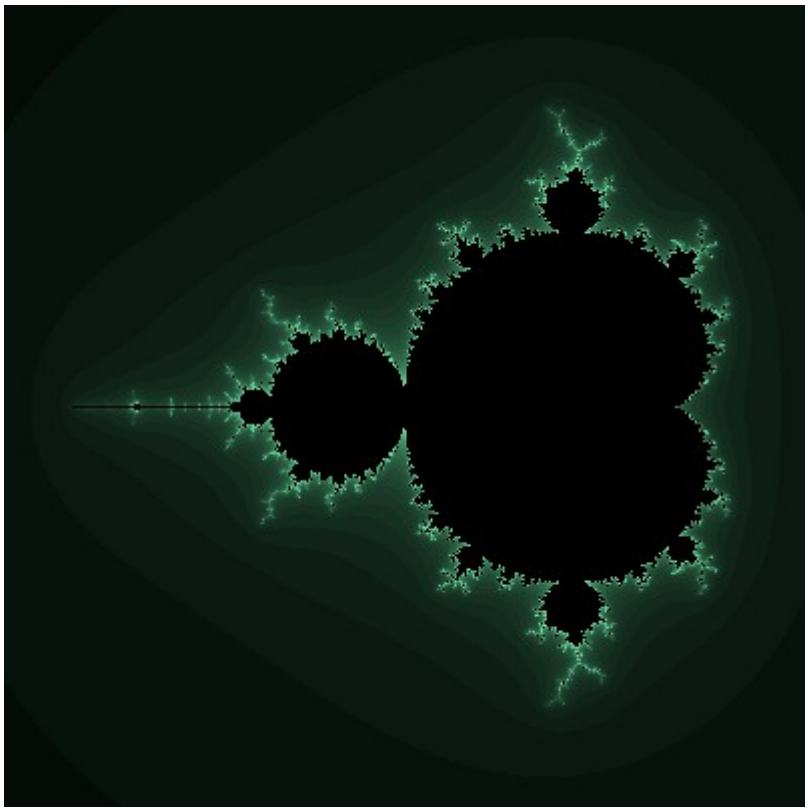
A generálandó kép típusát egy lenyíló lista segítségével határozhatjuk meg.

A választási lehetőségek a következők:

- Mandelbrot halmaz
- Véletlenszerű Julia halmaz
- Egyszínű kép
- Véletlenszerű kép

A Szélesség és a Magasság adatok megadásával a kép méreteit határozhatjuk meg pixelben számítva. A Red, Green és Blue csúszkákkal a generált kép színét befolyásolhatjuk.

Az OK gomb lenyomására megjelenik a generált kép.



7. ábra. Mandelbrot halmaz

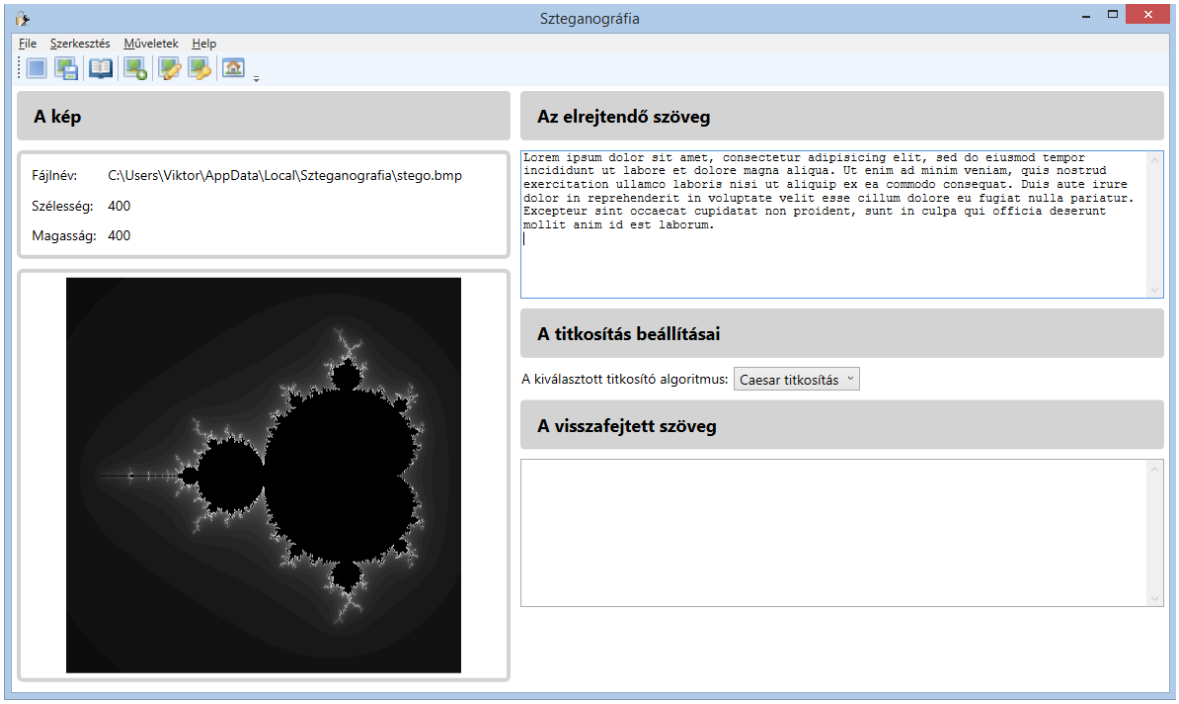
6.4.2. A szöveg elrejtése a képben

A funkció használatával az elrejtendő szöveg mezőben megadott szöveget rejthetjük el az előzőleg lemezről betöltött vagy az általunk generált képben.

A titkosítás beállításai szekcióban található lenyíló lista segítségével ha szeretnénk, kiválaszthatunk egy kriptográfiai algoritmust, amellyel az adatrejtésen túl még titkosíthatjuk is a megadott szöveget.

A választható kriptográfiai algoritmusok az alábbiak:

- Nincs titkosítás
- Caesar titkosítás
- Titkosítás 10 pozícióval történő eltolással
- Vigenère titkosítás egy konstans kulccsal



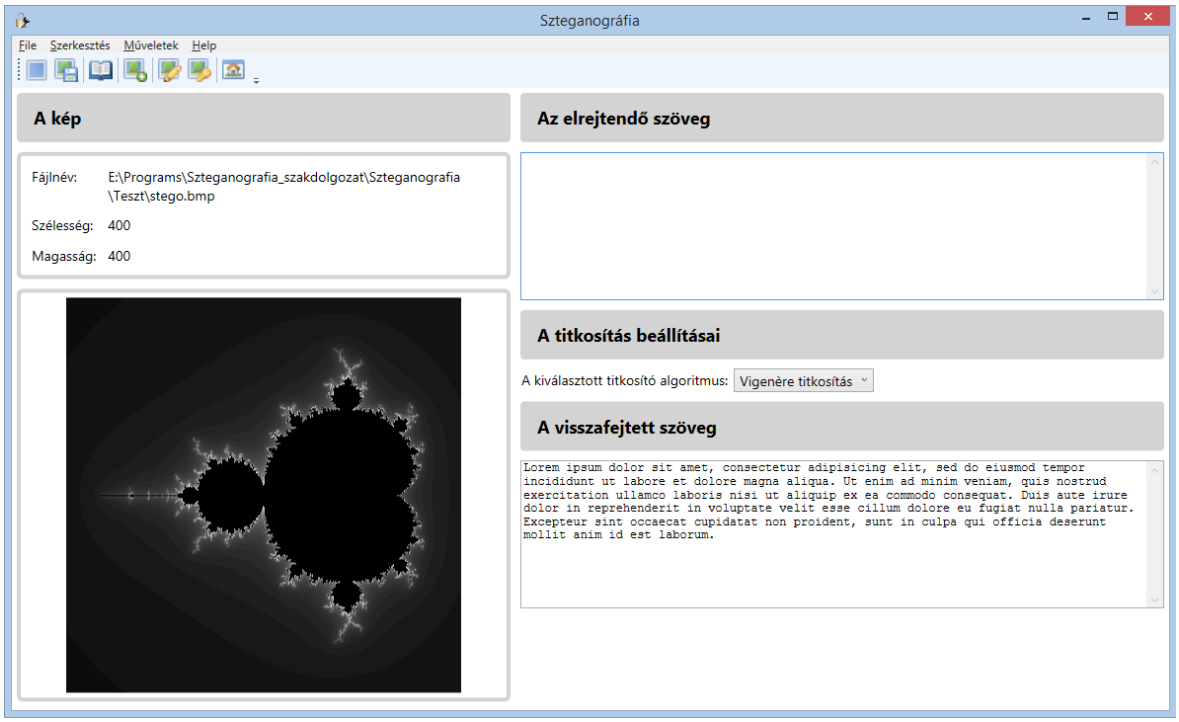
8. ábra. Sikeres adatrejtés

A program az elkészült képet ideiglenesen a felhasználó könyvtárában tárolja stego.bmp

néven. Ezt a képet a Kép mentése másként funkcióval menthetjük el az általunk kívánt helyre.

6.4.3. Szöveg visszafejtése a képből

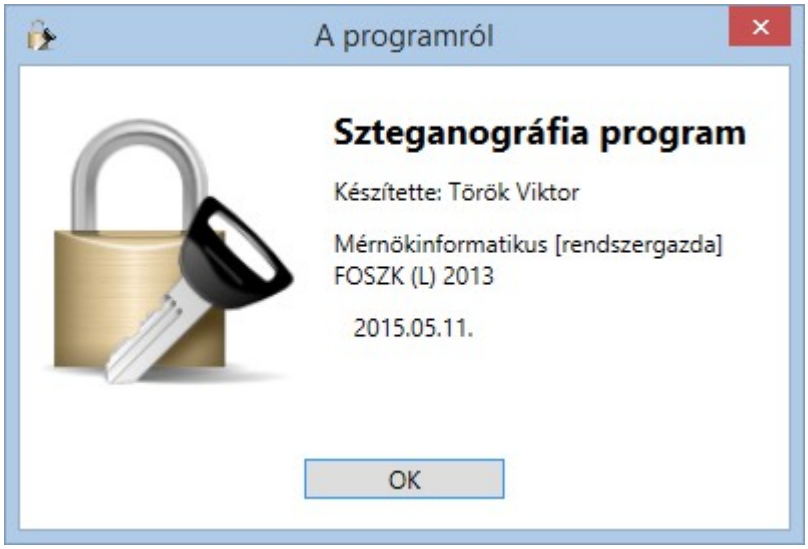
A funkcióval megkísérelhetünk visszafejteni egy képben elrejtett szöveget. A visszafejtés csak BMP formátumú képek esetén működik. A visszafejtéshez először be kell töltenünk a vizsgálni kívánt állományt, majd elindítani a funkciót. Sikeres művelet esetén a szöveg megjelenik az alsó, nem szerkeszthető szövegdobozban, probléma esetén pedig hibajelzést kapunk.



9. ábra. Sikeres visszafejtés

6.5. Help

A Help menüpont alatt egy újabb funkció található, amely a program névjegyét jeleníti meg.

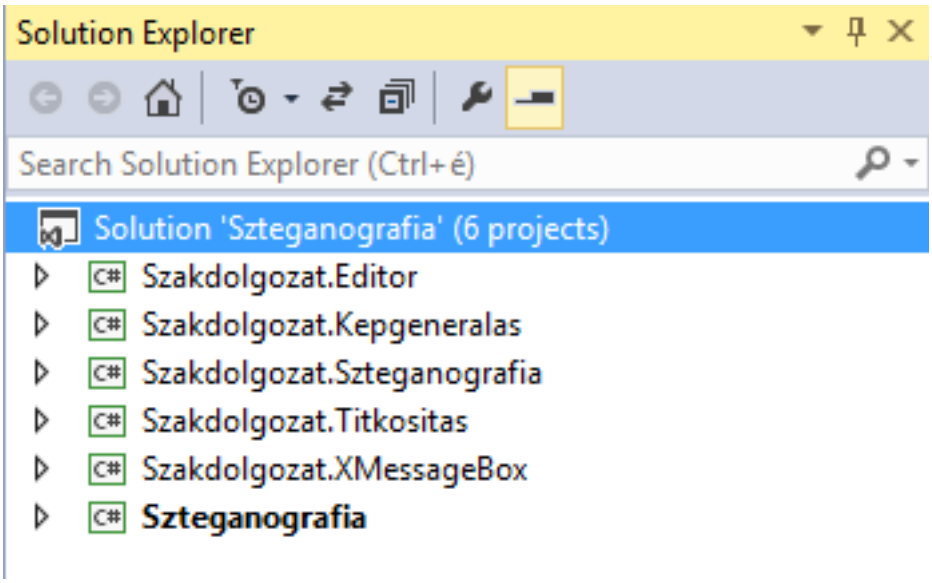


10. ábra. A program névjegye

7. A megvalósítás részletei

7.1. A projekt részletei

Az elkészült program 6 db projektből áll össze. Ezek a következők:



11. ábra. A program felépítése

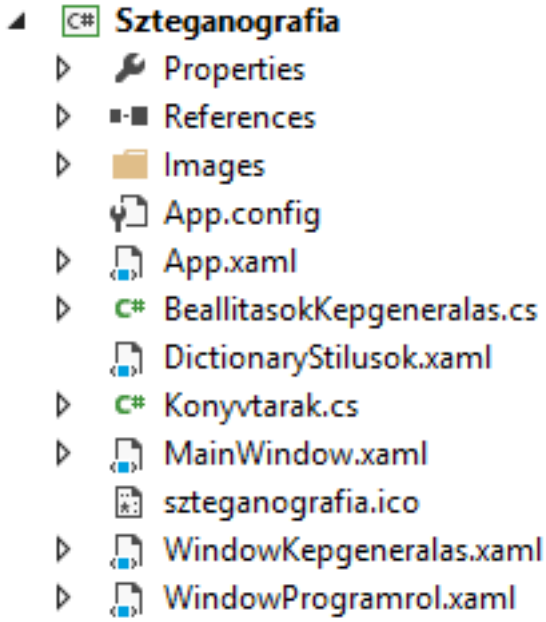
- *Szteganografia*: a főprogramot tartalmazó projekt.
- *Szakdolgozat.Editor*: a projekt egy olyan speciális szerkesztőmezőt tartalmaz, amely csak egész számokat fogad el. Ezt a képgeneráló algoritmusok paraméterezésénél használjuk.
- *Szakdolgozat.XMessageBox*: a projekt egy olyan statikus osztályt tartalmaz, amely a *MessageBox.Show* függvény, azaz a különböző üzenetablakok (információ, figyelmeztetés, hibajelzés) kényelmesebb használatát teszi lehetővé.
- *Szakdolgozat.Kepgeneralas*: a projektben a képgeneráló algoritmusok megvalósítása található.
- *Szakdolgozat.Szteganografia*: a projektben az LSB szteganográfiai algoritmus

megvalósítása található.

- *Szakdolgozat.Titkositas*: a projektben néhány klasszikus kriptográfiai algoritmus megvalósítása található.

7.2. A főprogram

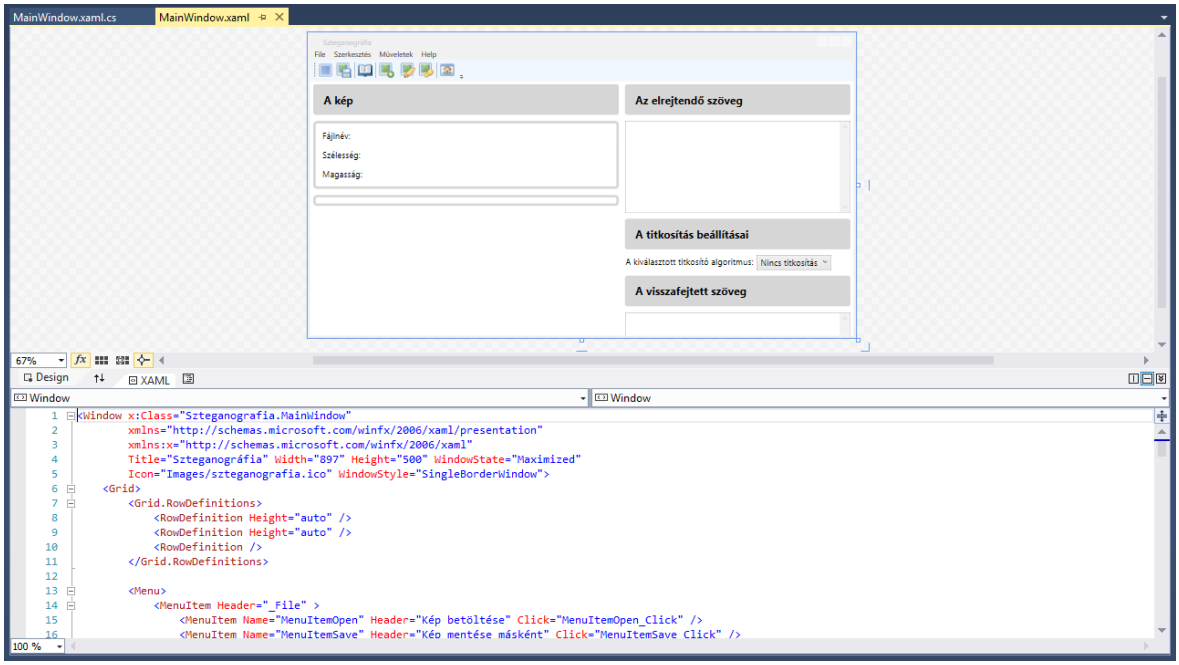
A főprogram a következő ábrán látható állományokból épül fel:



12. ábra. A Szteganografia projekt

A program három ablakot használ:

- *MainWindow.xaml*: a fő képernyő
- *WindowKepgeneralas.xaml*: dialógus ablak, amelyben a képgeneráló funkció beállításait lehet megadni
- *WindowProgramrol.xaml*: a program névjegyét megjelenítő dialógus ablak



13. ábra. A program fő képernyője a Visual Studioban

A WPF lehetőséget ad arra, hogy egy program megjelenését – bizonyos mértékig a webfejlesztésben használt CSS stíluslapokhoz hasonlóan – stílusok segítségével egységes módon határozzuk meg. A WPF stílusai lehetővé teszik, hogy a *FrameworkElement* ősosztály minden leszármazottjánál megadjunk bizonyos általánosan használt tulajdonságokat [41]. A Szteganográfia program is kihasználja a stílusok biztosította lehetőségeket.

A stílusbeállításokat a *Szteganografia* projekten belül a *DictionaryStilusok.xaml* állomány tartalmazza. Egy *TextBlock* szövegmezőre vonatkozó konkrét beállítást láthatunk a következő kódrészletben:

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

<Style TargetType="{x:Type TextBlock}" x:Key="TextBlockCimke">
        <Setter Property="FontSize" Value="14" />
        <Setter Property="Margin" Value="4 4" />
        <Setter Property="Padding" Value="2" />
    </Style>
</ResourceDictionary>
```

Az így megadott stílusra a *MainWindow.xaml* állományban a következő módon kell hivatkozni:

```
<TextBlock Text="Fájlnév:" Grid.Row="0" Grid.Column="0"
 Style="{StaticResource TextBlockCimke}" />
<TextBlock Name="TextBlockFilenev" TextWrapping="Wrap" Grid.Row="0" Grid.Column="1"
Style="{StaticResource TextBlockAdat}" />
```

A stílusok alkalmazásának több előnyét is tapasztaltam a program készítése során:

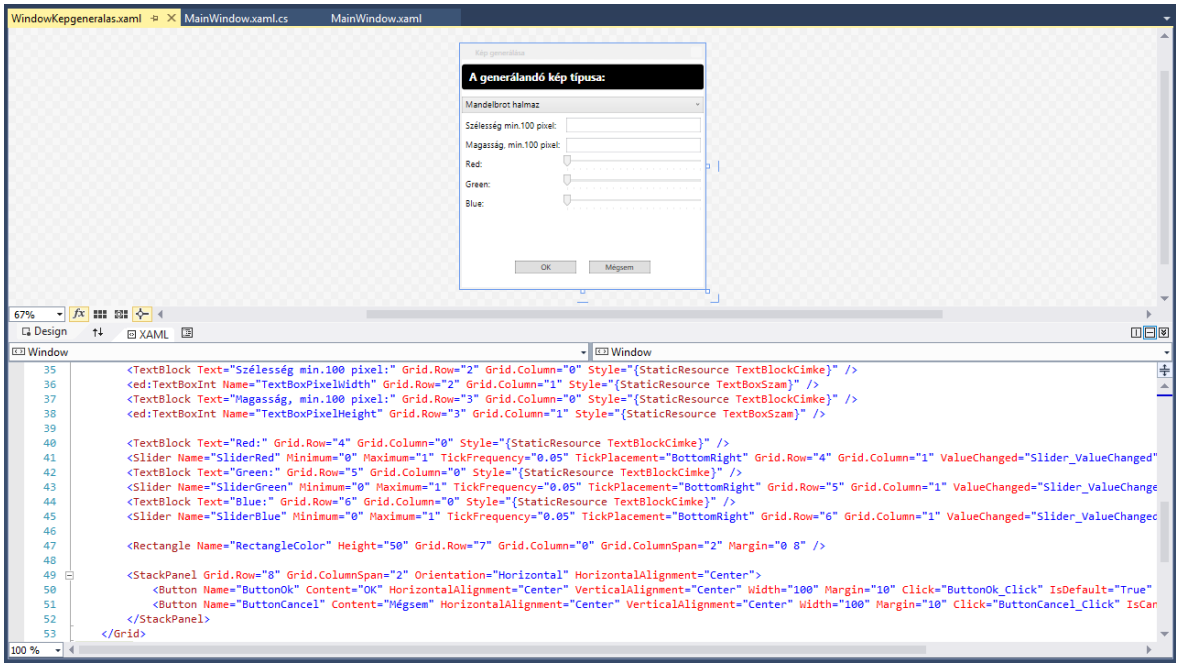
- egy-egy megadott típusba sorolt elem megjelenését a teljes programra vonatkozóan egységesen lehet szabályozni
- ha meg szeretnénk változtatni egy tulajdonságot, például a címke betűméretét, akkor azt elegendő egy helyen megtenni, és az összes ilyen stílusú címke megjelenése megváltozik
- a grafikus felületet leíró XAML kód rövidebb és átláthatóbb lesz

7.3. A képgenerálás beállításai

A képgenerálás beállításait egy dialógus ablakban adhatjuk meg. Az ablak XAML kódját a *WindowKepgeneralas.xaml* állomány tartalmazza. Ebben az esetben is felhasználtam a projektre vonatkozó közös stílusbeállításokat.

Azért, hogy a dialógus ablak beállításait meg tudjuk jegyezni az ablak két meghívása között, a program a felhasználó könyvtárában létrehoz egy *WindowKepgeneralas.xml* nevű XML-fájlt, amely a lényeges beállítások (a képtípus, a kép méretei, a csúszkák aktuális beállítása) értékét tartalmazza.

```
<?xml version="1.0" encoding="utf-8"?>
<Beallitasok>
  <ComboBoxKepTipus>0</ComboBoxKepTipus>
  <TextBoxPixelWidth>400</TextBoxPixelWidth>
  <TextBoxPixelHeight>400</TextBoxPixelHeight>
  <SliderRed>1</SliderRed>
  <SliderGreen>1</SliderGreen>
  <SliderBlue>1</SliderBlue>
</Beallitasok>
```



14. ábra. A WindowKepgeneralas.xaml a Visual Studioban

A generálandó kép méreteinek megadására használjuk az ablakon a *TextBoxInt* nevű új osztályt, amely a *TextBox* osztályból került leszármaztatásra, és csak egész számokat enged beírni a felhasználónak. A megvalósítás egy érdekes része látható a következő kódrészletben:

```
public class TextBoxInt : TextBox
    {
        public TextBoxInt()
        {
            this.KeyDown += TextBoxInt_KeyDown;
        }

        void TextBoxInt_KeyDown(object sender, KeyEventArgs e)
        {
            bool shiftDown =
              (Keyboard.Modifiers & ModifierKeys.Shift) == ModifierKeys.Shift;

            if ((e.Key >= Key.D0 && e.Key <= Key.D9 && !shiftDown) ||
                (e.Key >= Key.NumPad0 && e.Key <= Key.NumPad9) ||
                e.Key == Key.Tab || e.Key == Key.Enter || e.Key == Key.Escape)
            {
                e.Handled = false;
            }
            else
            {
                e.Handled = true;
```

```
            }
        }
    }
```

A *TextBoxInt* osztályból létrehozott editorok a billentyűk lenyomása esetén megvizsgálják azok kódját, és csak a következő kódokat engedik meg beírni:

- a számok a billentyűzet felső részén (ha nincs lenyomva a Shift billentyű)
- a számok a numerikus billentyűzeten
- a Tab, az Enter és az Escape billentyűk valamelyike

7.4. A képgeneráló algoritmusok

A képgeneráló algoritmusok a *Szakdolgozat.Kepgeneralas* nevű projektben találhatók.

Az egyes algoritmusok absztrakt (azaz nem példányosítható), *AbstractKepgeneralo* nevű ősosztálya a következő módon épül fel:

```
public abstract class AbstractKepgeneralo
    {
        public WriteableBitmap Bitmap { get; protected set; }

        public double Red { get; set; }
        public double Green { get; set; }
        public double Blue { get; set; }

        public AbstractKepgeneralo(int pixelWidth, int pixelHeight)
        {
            Bitmap = new WriteableBitmap(pixelWidth, pixelHeight, 96, 96,
                     PixelFormats.Rgb24, null);

            Red = 1.0;
            Green = 1.0;
            Blue = 1.0;
        }

        public abstract void Generalas();
    }
```

A képgenerálás eredménye a *Bitmap* nevű, *WriteableBitmap* típusú tulajdonságban jelenik meg. A *WriteableBitmap* osztály a WPF egyik rugalmasan használható bitmap-kezelő

osztálya, amellyel lehetővé válik a képek képpontonkénti manipulálása. A képpontok kezelését az általam implementált módszerek közük a legegyszerűbb, az egyszínű képet előállító *KepgeneraloEgyszinu* osztály *Generalas* függvénye a következő módon valósítja meg:

```
public override void Generalas()
        {
            byte[] pixels = new byte[Bitmap.BackBufferStride * Bitmap.PixelHeight];

            int pos = 0;

            for (int i = 0; i < Bitmap.PixelWidth; i++)
             {
                 for (int j = 0; j < Bitmap.PixelHeight; j++)
                  {
                     pixels[pos] = (byte)(Red * 255);
                     pixels[pos + 1] = (byte)(Green * 255);
                     pixels[pos + 2] = (byte)(Blue * 255);

                     pos += 3;
                  }
             }

            Bitmap.WritePixels(
              new Int32Rect(0, 0, Bitmap.PixelWidth, Bitmap.PixelHeight), pixels,
                           Bitmap.BackBufferStride, 0);
        }
```

A kódrészletből látható, hogy az Rgb24 formátumú kép képpontjait egy tömb feltöltése után a bitmap *WritePixels* metódusával írhatjuk vissza az eredeti képbe.

Az egyszínű kép előállításán túl megvalósításra került két érdekes és széles körben ismert képtípus generálása is, mégpedig a Mandelbrot és a Julia halmazok előállítása [42]. A két algoritmus komplex számokkal végzett iterációk vizsgálatán alapszik, amelyhez a .NET Framework a *Complex* nevű komplex szám típus biztosításával nyújt segítséget.

7.5. A titkosító algoritmusok

A titkosító algoritmusok a *Szakdolgozat.Titkositas* nevű projektben találhatók.

A megvalósított algoritmusoknak az *ITitkosito* nevű interfészt kell implementálniuk. Ennek érdekében egyrészt létre kell hozni egy *Titkositas* nevű függvényt, amely paraméterül egy bájtokból álló tömböt kap, és a visszatérési értéke is egy hasonló tömb. A paraméterben a

titkosítandó adatok találhatók, a visszatérési érték pedig a titkosított adatokat tartalmazza. Másrészt pedig meg kell írni a *Visszafejtes* nevű függvényt is, amely a *Titkositas* függvény inverze, azaz egy titkosított bájt-tömböt fejt vissza.

```
public interface ITitkosito
    {
        byte[] Titkositas(byte[] titkositando);
        byte[] Visszafejtes(byte[] visszafejtendo);
    }
```

Az egyik egyszerű titkosítási algoritmus a Caesar-módszer módosított változata, amely egy megadott számmal növeli vagy csökkenti a tömb elemeit [43]. Ebben az esetben a megvalósított függvények a következő módon épülnek fel:

```
public byte[] Titkositas(byte[] titkositando)
        {
            byte[] titkositott = new byte[titkositando.Length];

            for (int i = 0; i < titkositando.Length; i++)
            {
                titkositott[i] = (byte)(titkositando[i] + Eltolas);
            }

            return titkositott;
        }

public byte[] Visszafejtes(byte[] visszafejtendo)
        {
            byte[] visszafejtett = new byte[visszafejtendo.Length];

            for (int i = 0; i < visszafejtendo.Length; i++)
            {
                visszafejtett[i] = (byte)(visszafejtendo[i] - Eltolas);
            }

            return visszafejtett;
        }
```

A C# char típusának mérete két bájt, ezért ha egy stringet szeretnénk titkosítani, akkor a fenti algoritmusoknak történő átadás előtt a szöveget át kell konvertálni bájtokból álló tömbbé. Ezt a feladatot az *Encoding.UTF8.GetBytes* függvénnyel, a visszaalakítást pedig az *Encoding.UTF8.GetString* függvénnyel lehet elvégezni.

7.6. A titkosító algoritmusok létrehozása Egyszerű Gyárral

A titkosító algoritmusok példányosításának egyszerűsítésére az Egyszerű Gyár (Simple Factory) tervezési mintát használtam fel [44]. A *GetTitkosito* nevű statikus függvény a paraméterként kapott típus alapján visszaad egy újonnan példányosított titkosító objektumot. Ez lehet Caesar-titkosítás, ennek egy módosított változata, amely 10-zel való eltolással dolgozik, vagy pedig egy konstans kulccsal működő Vigenère-algoritmus.

```
public static class FactoryTitkosito
    {
        public static ITitkosito GetTitkosito(TitkositoAlgoritmus tipus)
        {
            ITitkosito titkosito = null;

            switch (tipus)
            {
                case TitkositoAlgoritmus.Caesar:
                    titkosito = new TitkositoCaesar();
                    break;

                case TitkositoAlgoritmus.Eltolas10:
                    titkosito = new TitkositoEltolas(10);
                    break;

                case TitkositoAlgoritmus.Vigenere:
                    titkosito = new TitkositoVigenere(new byte[] { 1, 2, 3, 4 });
                    break;
            }

            return titkosito;
        }
    }
```

7.7. Az LSB algoritmus

Az LSB-algoritmus megvalósítása a *Szakdolgozat.Szteganografia* nevű projektben található.

Az adatrejtő algoritmusok általános megfogalmazása az *AbstractStego* nevű absztrakt osztályban került leírásra.

```
public abstract class AbstractSztego
    {
        public TitkositoAlgoritmus Titkosito { get; set; }
```

```
        public abstract void Elrejtes(byte[] elrejtendo, byte[] adat);

        public void Elrejtes(string elrejtendoSzoveg, byte[] adat)
        {
            Elrejtes(Encoding.UTF8.GetBytes(elrejtendoSzoveg), adat);
        }

        public abstract string Visszafejtes(byte[] adat);
    }
```

Ennek alapján az *AbstractStego* osztály leszármazottainak meg kell valósítaniuk az elrejtés és a visszafejtés műveletét. Ezek a függvények – a titkosító algoritmusokhoz hasonló módon – bájtokból álló tömböt kapnak paraméterül. A *Titkosito* nevű tulajdonságban egy titkosító algoritmust lehet megadni, és ha ennek értéke nem null, akkor a programnak az adatrejtés előtt meg kell hívnia a titkosító függvényt az elrejtendő adatokon, illetve a visszafejtés esetén is figyelembe kell ezt vennie.

Az LSB algoritmus rejtő része a következő feladatokat kell, hogy elvégezze az itt megadott sorrendben:

- az elrejtendő szöveg hosszának kódolása
- a titkosító algoritmus azonosítójának kódolása
- az elrejtendő szöveg kódolása

A visszafejtő rész pedig a következő feladatokat végzi el:

- az elrejtett szöveg hosszának dekódolása
- a titkosító algoritmus azonosítójának dekódolása
- az elrejtett szöveg dekódolása

Az LSB-algoritmus implementálása során szükség volt a C# nyelv által biztosított bitműveletek (és, vagy, >>) használatára. Az elrejtendő szöveg kódolását tartalmazó programrészlet jól illusztrálja a módszert:

```
for (int i = 0; i < elrejtendo.Length; i++)
            {
                byte a = elrejtendo[i];

                for (int j = 0; j < 8; j++)
                 {
                    if ((a & 1) == 0)
                     {
                         adat[pos++] &= 254;
                     }
                    else
                     {
                         adat[pos++] |= 1;
                     }

                     a >>= 1;
                 }
            }
```

8. Továbbfejlesztési lehetőségek

A programmal kapcsolatban számos továbbfejlesztési lehetőséget lehet elképzelni.

Az egyik fejlesztési irány a szteganográfia területét érinti. A szakdolgozat csak a BMP formátumú képekben történő adatrejtést tárgyalta, az LSB-algoritmust viszont nem csak képek, hanem hangfájlok esetén is alkalmazhatjuk [45]. A Szteganográfia programot így fel lehetne készíteni például a WAV formátum kezelésére is.

A másik fejlesztési irány az LSB-algoritmus futtatása előtt használt kriptográfiai módszerekben rejlő lehetőségek kihasználásában gyökerezik. A .NET Framework számos kriptográfiai algoritmusra kínál kész megvalósítást, melyek a *System.Security.Cryptography* névtérben kaptak helyet [46]. Ezek között található pl. a DES, a Triple DES és az AES algoritmus. Ezen összetett algoritmusok implementálásával így nem is kellene külön foglalkozni, hanem rögtön alkalmazásra kerülhetnének a kész .NET-beli osztályok.

A titkosítás összetettebbé tételére kínál lehetőséget a produkciós titkosítók alkalmazása, amelyek kettő vagy több eltérő elvű művelet kombinálásával szolgáltatják eredményüket [47]. A produkciós titkosítók megfelelő módon történő használata nagyobb biztonságot eredményez, mint a bennük szereplő műveletek külön-külön. Egy ilyen módszer implementálása, a kriptográfiai műveletek egymás után fűzésének lehetősége tovább növelné az elrejtett adatok biztonságát.

9. Összegzés

Szakdolgozatomban az LSB-módszert alkalmazó szteganográfiai algoritmus vizsgálatát és egy olyan könnyen és egyszerűen kezelhető program implementálását tűztem ki célomul C# nyelven, a Windows Presentation Foundation (WPF) alkalmazásával, amely képes titkosítani és egy képben elrejteni a felhasználó által megadott szöveget. A program fejlesztése sikeresen megtörtént, és számos új tapasztalatra tettem szert mind a programozási nyelvvel, mind a keretrendszerrel kapcsolatban.

A fejlesztés során különösen érdekes volt a WPF használatában újabb tapasztalatokat szerezni, ugyanis a program készítése során több olyan technikai jellegű problémával is találkoztam, melyekre csak hosszas internetes keresés után találtam elfogadható megoldást.

Szándékomban áll az elkészült program továbbfejlesztése, amely tartalmazza a további hordozók (pl. hangállományok) kezelését és a .NET Framework által biztosított kriptográfiai algoritmusok felhasználásának lehetőségét.

10. Irodalomjegyzék

Hivatkozott irodalom

[1] Göcs László: Informatikai biztonság alapjai, 2. konzultáció, 2014-15. 1.félév, 44. dia

[2] Göcs László: Informatikai biztonság alapjai, 2. konzultáció, 2014-15. 1.félév, 59. dia

[3] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 4. oldal

[4] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 8. oldal

[5] Oxford Számítástechnikai értelmező szótár, Novotrade Kiadó, 1989, 105. oldal

[6] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 8. oldal

[7] Oxford Számítástechnikai értelmező szótár, Novotrade Kiadó, 1989, 105. oldal

[8] Göcs László: Informatikai biztonság alapjai, 2. konzultáció, 2014-15. 1.félév, 43. dia

[9] Unicsovics György: Szteganográfia elemeinek implementálási lehetőségei a védelmi szektorban, Budapest, 2007, 20. oldal

[10] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 237. oldal

[11] Unicsovics György: Szteganográfia elemeinek implementálási lehetőségei a védelmi szektorban, Budapest, 2007, 60. oldal

[12] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 227. oldal

[13] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 228. oldal

[14] Az arcképek forrása: http://hu.wikipedia.org/wiki/Karl_D%C3%B6nitz
http://www.independent.co.uk/news/people/news/alan-turing-gets-royal-pardon-for-gross-indecency--61-years-after-he-poisoned-himself-9023116.html
http://www.maritimequest.com/warship_directory/germany/u_boats/ww2/pages/001_099/u_47_page_2.htm

[15] https://hu.wikipedia.org/wiki/H%C3%A9rodotosz

[16] Hérodotosz: A görög-perzsa háború, Ötödik könyv, 35.

[17] Hérodotosz: A görög-perzsa háború, Hetedik könyv, 239.

[18] Gaius Suetonius Tranquillus: A caesarok élete, Európa Könyvkiadó, 1984, 29. oldal

[19] Gaius Suetonius Tranquillus: A caesarok élete, Európa Könyvkiadó, 1984, 87. oldal

[20] http://hu.wikipedia.org/wiki/Tannenbergi_csata_%281914%29

[21] John Keegan: Az első világháború, Európa Könyvkiadó, Budapest, 2010, 301-302.

oldal

[22] http://hu.wikipedia.org/wiki/Midwayi_csata

[23] Walter Lord: Hihetetlen győzelem, Zrínyi Katonai Kiadó, Debrecen, 1980, 25. oldal

[24] Walter Lord: Hihetetlen győzelem, Zrínyi Katonai Kiadó, Debrecen, 1980, 26-32. oldal

[25] Mérai László, Kékesi Júlia: Lengyelek az Enigma ellen, ELTE TTK, 3. oldal

[26] Mérai László, Kékesi Júlia: Lengyelek az Enigma ellen, ELTE TTK, 18. oldal

[27] http://hu.wikipedia.org/wiki/Enigma_%28g%C3%A9p%29

[28] T.Dénes Tamás: Titkos-számítógép-történet, Aranykönyv Kiadó Bt., Budapest, 2003, 56. oldal

[29] http://hu.wikipedia.org/wiki/2001._szeptember_11-ei_terrort%C3%A1mad%C3%A1sok

[30] Unicsovics György: Szteganográfia elemeinek implementálási lehetőségei a védelmi szektorban, Budapest, 2007, 5-6. oldal

[31] http://prog.hu/hirek/3734/Megnyitja+es+Linuxra+valamint+Mac-re+is+portolni+fogja+a+dot-net-et+a+Microsoft.html

[32] http://prog.hu/hirek/3898/Mar+toltheto+a+reszben+Microsoft-kodokra+epulo+Mono+4+0.html

[33] Budai Attila: A számítógépes grafika, LSI Oktatóközpont, Budapest, 1999, 148. oldal

[34] Budai Attila: A számítógépes grafika, LSI Oktatóközpont, Budapest, 1999, 149. oldal

[35] http://en.wikipedia.org/wiki/RGBA_color_space

[36] http://hu.wikipedia.org/wiki/BMP

[37] http://hu.wikipedia.org/wiki/PNG

[38] http://hu.wikipedia.org/wiki/Vesztes%C3%A9ges_t%C3%B6m%C3%B6r%C3%ADt%C3%A9s

[39] http://hu.wikipedia.org/wiki/JPEG

[40] Peter Wayner: Disappering Cryptography, Second Edition, Morgan Kaufmann Publishers, 2002, 177. oldal

[41] Cristopher Bennage – Rob Eisenberg: Tanuljuk meg a WPF használatát 24 óra alatt, Kiskapu Kft., 2009, 223. oldal

[42] Tariq Rashid: Make your own Mandelbrot, 2014

[43] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 18. oldal

[44] Tony Bevis: C# Design Pattern Essentials, AbilityFirst Limited, 2012, 185. oldal

[45] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 252. oldal

[46] Stephen Haunts: Cryptography in .NET Succintly, Syncfusion Inc., 2015, 10. oldal

[47] Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004, 30. oldal

Felhasznált irodalom

[1] Reiter István: C# programozás lépésről lépésre, Jedlik Oktatási Stúdió, Budapest, 2012

[2] Matthew MacDonald: Pro WPF 4.5 in C#, Apress, 2012

Tartalomjegyzék