Horváth Zoltán

Az oktatási tevékenység támogatása rejtjelezést, valamint adatrejtést megvalósító programcsomag alkalmazásával II.

Jelen cikkben a szerző előző cikkének folytatásaként [1] részleteiben mutat be néhány, a programcsomagban megvalósított algoritmust. Ezek alapján mélyebben megismerhetővé válik a programcsomag programjainak működése.
Kulcsszavak: szimmetrikus kódolás, aszimmetrikus kódolás, szteganográfia, programvédelem

Bevezetés

A programcsomag fejlesztése során számos probléma merült fel. Adott volt a fejlesztői környezet, amely behatárolta a lehetőségeket. Adottak voltak a kezdeti továbbá követelmények, amelyeket teljesíteni kellett. A lehetőségek és a követelmények tükrében néha csak kerülőutakat keresve sikerült megoldást találni. Részlépésenként, a megoldások sikerén felbuzdulva, magunk állítottunk új, szigorodó, előremutató követelményeket. Ez a cikk néhány ilyen problémamegoldásról szól.

Fejlesztői környezet

A programcsomag fejlesztése Windows operációs rendszer alatt, Delphi 6.0 fejlesztői környezetben történt. A programok alkalmazzák a Windows operációs rendszer erőforrásait, jellemző üzenet- és dialógusablakait.

A programcsomag alkalmazásának feltételei:

- Windows XP/7/8 operációs rendszer;
- definiált hálózati adapter (hálózati csatlakozás nem szükséges).

Kripto-számológép a természetes számok halmazán

A hagyományos, de még a tudományos kalkulátorok között sem található olyan eszköz, amely képes bemutatni az egyes rejtjelező algoritmusok működését. Ezen algoritmusok extrém méretű természetes számokkal végeznek műveleteket (alapműveletek, hatványozás), ahol követelmény a fix pontos számábrázolás. Található néhány művelet (pl. prímellenőrzés, kongruens számpár keresése), amely a rejtjelezést megvalósító algoritmusok szempontjából alapműveletnek számítanak. Továbbiakban ismertetem egy kialakított általunk kidolgozott számológép elvi alapjait, amelyek – a természetes számok halmazán – megoldást kínál az említett problémák megoldására.

Számábrázolás

A számábrázolás során nem elégedhetünk meg 10-12 jegyű természetes számok feldolgozásával. Követelmény, hogy szükség esetén több száz, esetleg több ezer számjegyet tartalmazó természetes számokkal is végezhessünk matematikai műveleteket.

A számítástechnikában alkalmazott *„nullsztring”* adatformátum ebben segítségünkre lehet. A *„nullsztring”* egy olyan változó, amely szövegek tárolását biztosítja. A tárolt szöveg hossza előre nem definiált. A tárolt szöveg végét a szöveg után található *#0* karakter jelzi. (Az ASCII-kódtáblázatban alkalmazott *nulla* a *#0 = Chr(0)* karakter. A továbbiakban a *#* előtagot alkalmazzuk kódjuk alapján az ASCII-karakterek jelölésére.) .Ebből adódóan a tárolni kívánt szöveg hosszának csak a memóriaméret szab határt. A szöveg egyes karaktereire tömbelemként hivatkozhatunk. Ha szövegként tárolunk nagy számokat, azokkal aritmetikai műveleteket is végezhetünk. Ez a *szöveg alapú aritmetika* kiindulópontja. A következő alapműveleteket dolgoztuk ki:

- összeadás, szorzás, hatványozás;
- kivonás, osztás, maradékképzés;
- prímellenőrzés, relatív prím ellenőrzése, kongruens számpár keresése.

Összeadás, szorzás, hatványozás

Legyen példaként a ’8’ és a ’6’ két operandus, amelyek két szövegként tárolt szám azonos helyi értéken található számjegyei (karakterei). Továbbá legyen $c = 1$ valamely, az alacsonyabb helyi értékről áthozott átvitel (a továbbiakban ***div*** az egészosztás, ***mod*** a maradékképzés jele).

Az összeadás algoritmusa ebben az esetben:

’8’ $\rightarrow a = \mathbf{8}$	karakter-számjegy konverzió;
’6’ $\rightarrow b = \mathbf{6}$	karakter-számjegy konverzió;
$s := a + b + c$	azaz $s = 8 + 6 + 1 = 15$;

$c := s$ ***div*** $10 = 1$	a keletkezett új átvitel;
$s := s$ ***mod*** $10 = 5$	az összeg;
$s \rightarrow$ '*s*', azaz '5'	számjegy-karakter konverzió, az adott helyi értéken az összeg karaktere.

A *szorzás* algoritmusa ebben az esetben:

'8' $\rightarrow a =$ **8**	karakter-számjegy konverzió;
'6' $\rightarrow b =$ **6**	karakter-számjegy konverzió;
$m := a \times b +$ c	azaz $m = 8 \times 6 + 1 = 49$;
$c := m$ ***div*** $10 = 4$	a keletkezett új átvitel;
$m := m$ ***mod*** $10 = 9$	a szorzat;
$m \rightarrow$ '*m*', azaz '9'	számjegy-karakter konverzió, az adott helyi értéken a szorzat karaktere.

Számjegyeken végzett műveletet vizsgálva induljunk ki a számjegyek aritmetikájából.

Ezek az eljárások az alapjai a karakteres formában tárolt számjegyek *összeadás*ának, illetve *szorzás*ának. A számok összeadása és szorzása a számjegyek összeadására és szorzására építhető fel. A szám önmagával történő szorzásának sorozatával a *hatványozás* is kivitelezhető.

Kivonás, osztás, maradékképzés

A bináris aritmetika a kettes komplemens képzésével, két szám összeadásával tulajdonképpen kivonást valósít meg. Ha a kivonandó kettes komplemensét összeadjuk a kisebbítendővel, elvégezzük a kivonást. Ezt a gondolatot adaptálva a decimális aritmetikára, lehetőség nyílik a *kilences komplemens* és a *tízes komplemens* alkalmazására.

Kilences komplemens képzése:

A forrásszöveg (szám) alapján létrehozunk egy ugyanolyan hosszú szöveget (számot), amelyben a számjegyek összege valamennyi helyi értéken 9. Ezt a számot nevezzük a *forrás kilences komplemens*ének.

Példaként állítsuk elő a := '34963' kilences komplemensét:

99999	mivel:	34963
–34963		+65036
65036		**99999**

Tehát például a 34963 kilences komplemense a 65036.

Tízes komplemens képzése:

A kilences komplemens legalacsonyabb helyi értéken található karaktert eggyel inkrementálva (egyet hozzáadva a karakterkód értékéhez) megkapjuk a forrásszöveg (szám) tízes komplemensét.

Tehát például a 34963 tízes komplemense 65036 + 1 = 65037.

Vagyis: ha egy számhoz (kisebbítendő) hozzáadjuk egy másik szám (kivonandó) tízes komplemensét, majd az összeg legmagasabb helyi értékét elhagyjuk, azzal *kivonást* végzünk.

Példaként legyen a feladat: 7834 – 3863.

3863 kilences komplemense: 6136, tízes komplemense: 6137.

1 (átv. kezdőértéke)

7834	helyett:	7834	vagy:	7834
–3863		+6137		+6136
3971		**13971**		**13971**

Ez a legegyszerűbben úgy oldható meg, ha az összeadás során kilences komplemenst alkalmazunk (gyorsan előállítható), és az összeadás során az átvitel kezdőértékének az 1-et választjuk (lásd a fenti művelet jobb oldali megoldását).

A kivonás az alapja az *egészosztás*nak és a *maradékképzés*nek.

Prímellenőrzés, relatív prím ellenőrzése, kongruens számpár keresése

Az eddigiekben bemutattam az alapműveletek végrehajtását szöveges aritmetika alkalmazásával. Ezekkel a szöveges aritmetikai alapműveletekkel megvalósítható műveletek a következők:

- *Prímellenőrzés:* annak ellenőrzése, hogy egy szám csak 1-el és önmagával osztható;
- *Relatív prím ellenőrzése:* annak ellenőrzése, hogy két számnak van-e 1-től különböző közös osztója;
- *Kongruens számpár keresése:* ha A és N relatív prím, az

$$A \times B = C \times N + 1$$

egyenlet megoldása B-re, ahol C futóparaméter. [5]

Szimmetrikus kódolás Polübiosz-tábla alkalmazásával [4]

A Polübiosz-tábla alkalmazása az egyik legegyszerűbb példája a szimmetrikus kódolásnak. Problémaként adódott az ábrázolható szimbólumok korlátozott mennyisége. Ennek megoldásaként a Polübiosz-tábla mintájára létrehoztunk egy 120 elemet tartalmazó kibővített mátrixot, amely minden nyomtatható karaktert mint szimbólumot tartalmaz. Ezután a szimbólum koordinátáinak meghatározása, valamint a szimbólumoknak a koordinátáik alapján történő azonosítása már egyszerű feladattá vált.

Aszimmetrikus kódolás-dekódolás az RSA-algoritmus alkalmazásával [2, 3, 5]

Az RSA kódoló-dekódoló program kialakításakor felhasználtuk a kripto-számológép fejlesztése során létrehozott algoritmusokat. Mivel ez a program a hangsúlyt a kulcsgene-

rálás helyett az egyszer kódolt, valamint a duplán kódolt információtovábbításra helyezi, megengedhetetlen az, hogy az alkalmazók kulcsgenerálásra fordítsák a gépidőt, elvéve azt az egyéb tapasztalatszerzés (hibrid kódolás, hitelesség, letagadhatatlanság) lehetőségétől. Ki kellett fejleszteni tehát egy gyors működésű kulcsgenerátort, amellyel másodpercek alatt lehet RSA-kulcsokat előállítani.

A megoldást ebben az esetben a következők jelentették:

- *Prímellenőrzés:* Prímszámnak ítélt számokra tippelés során valós idejű prímellenőrzés, és az ellenőrzés eredményének azonnali kijelzése. Felhasználói kezdeményezésre a bevitt számhoz legközelebbi nagyobb vagy kisebb prím megjelenítése. Nem léphetünk tovább, amíg a két prímszám nincs meg. Ha megvan, rendelkezésre áll az RSA-kulcs második szegmense, ami azonnal kijelzésre kerül.
- *Relatív prím ellenőrzése:* Tippelés a prímszámok dekrementált szorzatához képest relatív prímszámra. Ennek során szintén valós idejű ellenőrzés és az ellenőrzés eredményének azonnali kijelzése történik. Felhasználói kezdeményezésre a bevitt számhoz legközelebbi nagyobb vagy kisebb relatív prím megjelenítése. Nem léphetünk tovább, amíg nincs meg a relatív prím.
- *Kongruens számpár keresése:* Ha megvan a relatív prím, a program automatikusan azonnal kijelzi a prímszámok dekrementált szorzatának egész számú többszörösére vonatkozó kongruens számpárt. A relatív prím és kongruens számpárja, az RSA-kulcspár első szegmense, azonnal kijelzésre kerül.

Belátható, hogy ilyen körülmények között a kulcsgenerálás másodpercek kérdése.

Adatrejtés TrueColor (24 bites) nem tömörített BMP képfájl bittérképében

Az adatrejtés általános elve [2, 6, 7]:

A rejtés hatására létrejövő színtorzítás a bittérkép legalacsonyabb helyi értékű bitjei (LSB) módosításának hatására csekély, szabad szemmel homogénnak (egyszínűnek) látszó képterületeken is megfigyelhetetlen.

Tekintettel arra, hogy a sorokat leíró bájtok mennyiségét néggyel oszthatóvá tevő *#0* bájtok jól meghatározható helyen találhatók a bittérképben, ha ezek legalacsonyabb helyiértékű bitjeit (LSB) is módosítjuk, ez rögtön elárulhatja az adatrejtés tényét (kimutatható struktúraváltozás). A legegyszerűbb szabály ennek elkerülése érdekében, hogy a bittérkép *#0* bájtjait változatlanul hagyjuk. Ennek következménye lehet, hogy a töltelék *#0* bájtok mellett a bittérképben található pixelszínkomponens-intenzitást leíró egyéb *#0* bájtok miatt csökken a képfájl kapacitása, de természetes fényképeken ezek előfordulási gyakorisága csekély.

Tovább csökkenti a rejtés kimutathatóságát, ha a bittérképben a rejtéshez fel nem

használt RGB-komponensek legalacsonyabb helyi értékű bitjeit változatlanul hagyjuk, vagy véletlenszerűen beírt '0' vagy '1' bitekkel „zajosítjuk".

Rejtett adatstruktúra a bittérképben:
A program egy keretrendszert alkalmaz, amely közrezárja a beírt adatot:

('*A*' keret – rejtett adat – '*B*' keret)

A két keret felépítése azonos. Mindkettő négy bájt hosszon a rejteni kívánt szöveg hosszúságát rögzíti a bittérkép LSB-it alkalmazva. Ebből adódik (ha elég nagy a bittérkép), hogy egy képfájlba elméletileg maximum $2^{32}-n$, azaz mintegy *4,2 millió* karakteres szöveg rejthető. Mivel a hosszt kódoló keret négybájtos, ebből adódik, hogy a keretek 32-32 bájtot foglalnak le a bittérképből, a köztük található rejtett szöveg karakterenként további 8 bájtot használ fel.

A képfájl státuszának azonosítási folyamata:

- *„Tabula Rasa":* Ha a bittérkép valamennyi bájtjának LSB-je '0'. Az ilyen fájl elő van készítve adatrejtésre. A bittérkép bájtjainak beállítása, illetve ellenőrzése, ahol '*A*' a vizsgált bájt, a következő műveletekkel történik:
  - *Ellenőrzés:* Not('*A*' AND '00000001') = 1 (ez az állítás igaz?);
  - *Beállítás*: '*A*' := '*A*' AND '11111110' (ez legyen '*A*' értéke);
  - *Ha* '*A*' = #1, akkor '*A*' := #2 (karaktercsere).
- *„Zajos fájl", „Adathordozó":* A bittérkép első 32 bájtja ('*A*' keret) alapján meghatározzuk a feltételezett rejtett szöveg hosszát. Ha belefér a bittérképbe, átlapozva a fájlban a szöveghossz nyolcszorosának megfelelő mennyiségű (a *#0* bájtokat nem számolva) bájtot, az ezt követő 32 bájt elvileg újra a rejtett szöveg hosszát mutatja ('*B*' keret);
  - *Ha az 'A' keret túlmutat a bittérképen,* vagy a két keret nem egyenlő, akkor csak egy zajos (nyers) képfájlról beszélünk;
  - *Ha a két keret egyenlő,* akkor a képfájl adathordozó, amelyből a rejtett szöveg a bittérkép 33. bájtjától az LSB-k alapján nyolcbitenként bájtokká szervezve kinyerhető;
  - Ehhez alkalmazva az alábbi logikai műveleteket:
  - *LSB ellenőrzése:* '*A*' AND '00000001' = 1 (ez az állítás igaz?);
  - *LSB beállítása:* '*A*' := '*A*' OR '00000001' (ez legyen '*A*' értéke).

Bittérkép torzítása a legalacsonyabb helyi értékű bitek (LSB) alkalmazásával:
A következő példában az '*a*' karakter rejtését mutatom be. A rejtés a bittérkép **RGB** komponenseinek LSB-jét alkalmazza. Az '*a*' = *#97* karakter bináris kódja: 01100001.

	Nyers kép			Tabula Rasa			Adatrejtő kép		
…	**00001100**			**00001100**			**00001101**		
1	**01111010**	–	0	**01111010**	–	0	**01111010**	–	0
2	**00001100**	–	0	**00001100**	–	0	**00001101**	–	1
3	**11011100**	–	0	**11011100**	–	0	**11011101**	–	1
4	**01111010**	–	0	**01111010**	–	0	**01111010**	–	0
5	**00001100**	–	0	**00001100**	–	0	**00001110**	–	0
6	**11011011**	–	1	**11011010**	–	0	**11011010**	–	0
7	**01111010**	–	0	**01111010**	–	0	**01111010**	–	0
8	**00001100**	–	0	**00001100**	–	0	**00001101**	–	1
	11011010			**11011010**			**11011010**		

A táblázatban megfigyelhető az LSB-k nullába állítása (Tabula Rasa), valamint a ’01100001’ kód (*#97*, vagyis az *’a’* karakter) rejtése.

A programcsomag programjainak validálása

Létrejött egy program, amelynek rendeltetése a felhasználói számítógépre másolt rejtjelezést és adatrejtést megvalósító programok aktiválása, ezzel akadályozva az illetéktelen másolatok alkalmazását.

Mivel a létrehozott program a többi program futtathatóságának kulcsa, ezért fokozott felügyeletet igényel. A programról csak indokolt esetben készítsünk másolatot, illetve

semmilyen esetben se másoljuk át a felhasználó számítógépére,

hanem futtassuk külső meghajtóról.

A validálás alapja [8]

A számítógépek egyedi azonosítására a számítógéphez csatlakozó hálózati adapter fizikai címe (MAC-cím) ad lehetőséget.

Minden hálózati adapternek, eszköznek saját MAC-címe van. A címet (címtartományokat) a szabványügyi hivatal adja ki a gyártónak, és ezt a gyártó fizikailag „beégeti”, vagy szoftverrel beállítja az interfészben.

A címet 12 darab hexadecimális számjegy (pl. FE-32-D0-65-EA-54) formájában adják meg, amelyből az első hat hexadecimális számjegy kiosztását az IEEE (Institute of Electrical and Electronics Engineers) felügyeli, ezek a gyártót vagy az eladót azonosítják. A MAC-címnek ezt a részét egyedi szervezetazonosítónak (organizational unique identifier, OUI) nevezik. A fennmaradó hat hexadecimális számjegyet a gyártó adminisztrálja saját körben.

A több hálózati adapterrel rendelkező eszközöknek célszerűen több fizikai címe is lehet.

A program megállapítja a Windows operációs rendszer „Eszközkezelő"-ben, a „Hálózati kártyák" alatt felsorolt eszközök közül az *első* helyen található, engedélyezett eszköz MAC-címét (az adminisztrátor manuálisan is megadhat egy általa ismert címet). A program a validálás során ezt az értéket jegyzi be a validálandó programba a program azonosítása után.

A validálandó program felépítése

A validálandó program forráskódjában két fix hosszúságú mezőt alakítottunk ki, amelyek fordítás után is fellelhetők a programban. Ezek az *„azonosító mező"* és a *„MAC-cím mező"*.

Az *„azonosító mező"* a lefordított futtatható (*.exe) programfájlban, meghatározott pozícióban található. Tartalma a forráskódban rögzített, előre meghatározott karaktersorozat. Ez és a fájlhossz alapján azonosítható a validálandó program. Ebből következik, hogy a program átnevezhető, ez nem befolyásolja a validálás lehetőségét.

A forráskódban a *„MAC-cím mező"* tartalma is egy karaktersorozat, amelynek célja a futtatható (*.exe) programfájl adott fájlpozíciójában hely biztosítása a MAC-cím programfájlba történő bejegyzéséhez. A MAC-cím nem közvetlenül, hanem karakterenként egy XOR logikai művelet végrehajtását követően lesz bejegyezve a validálandó programba. Ennek oka, hogy közvetlen fájlkezelő programokat alkalmazva ne lehessen kideríteni a *„MAC-cím mező"* fájlbeli pozícióját, ne lehessen közvetlen fájlkezeléssel indirekt úton validálni.

A validálás folyamata

A validálás során végrehajtott lépések:

- MAC-cím meghatározása vagy bekérése
- Célfájl (*.exe futtatható programfájl) megnyitása olvasásra
- A célfájl hosszának meghatározása
- Ha a fájlhossz alapján a fájl nem tartozik a programcsomaghoz, kilépés
- Az *„azonosító mezőb* tartalmának ellenőrzése
- Ha az *„azonosító mező"* tartalma nem egyezik a várt karaktersorozattal, kilépés
- Célfájl átnevezése (forrásfájl lesz belőle)
- Eredeti fájlnéven új célfájl megnyitása írásra
- A forrásfájlból a célfájl írása a *„MAC-cím mezőig"*
- Az aktuális MAC-cím írása (logikai művelettel rejtve) a célfájlba
- A forrásfájlból – átugorva a *„MAC-cím mezőt"* – a hátralévő tartalom írása a célfájlba
- A forrásfájl törlése

A validált program működése

A validált program futtatáskor a következő lépéseket hajtja végre:

- Aktuális MAC-cím meghatározása
- Ha nincs aktív hálózati kártya, kilépés

- Az aktuális és a programba bejegyzett MAC-cím összehasonlítása
- Ha az aktuális és a programba bejegyzett MAC-cím nem egyezik, kilépés
- Funkcionális működés (a *„Reset"* funkció nem kezdeményez újabb azonosítást)

Összefoglalás

A fejlesztés kezdetekor még nem látszott előre, mennyi probléma merül fel. Nem volt előre meghatározható, hogy a programok milyen alkalmazói felület kialakítását igénylik, hogy kiérdemeljék a *felhasználóbarát* jelzőt. Ezért lépésről lépésre formálódott a programcsomag, amely összességében már megfelel az előre kitűzött követelményrendszernek.

Reményeim szerint cikkemnek sikerült bemutatnia a programfejlesztések során felmerülő váratlan problémák megoldásának izgalmát, illetve azt, hogy egy probléma megoldása milyen hatással lehet a végeredmény minőségére, lehetőségeinek bővítésére.

Irodalomjegyzék

[1] Horváth Zoltán: *Az oktatási tevékenység támogatása rejtjelezést, valamint adatrejtést megvalósító programcsomag alkalmazásával.* Kommunikáció 2014, Nemzeti Közszolgálati Egyetem, Budapest, 2014. november 12., 43–54. o. http://www.puskashirbaje.hu/index_html_files/Kommunikacio_2014-NSZTK.pdf (Letöltés: 2016. 03. 10.)

[2] Dr. Berta István Zsolt: *Nagy e-szignó könyv. Amit az elektronikus aláírásról tudni akartál, csak féltél megkérdezni.* Microsec Kft., Budapest, 2011.

[3] www.inf.elte.hu/karunkrol/digitkonyv/Jegyzetek2010/A_rejtjelezes_nehany_kerdese.pdf (Letöltés: 2015. 08. 27.)

[4] https://hu.wikipedia.org/wiki/Polübiosz-négyzet (Letöltés: 2014. 05. 24.)

[5] https://hu.wikipedia.org/wiki/RSA-eljárás (Letöltés: 2014. 05. 24.)

[6] https://hu.wikipedia.org/wiki/Szteganográfia (Letöltés: 2014. 05. 24.)

[7] https://hu.wikipedia.org/wiki/BMP (Letöltés: 2014. 05. 24.)

[8] https://hu.wikipedia.org/wiki/MAC-cím (Letöltés: 2014. 05.24.)

Using a software package to support the educational activities of encryption and steganography implementation II.

Horváth Zoltán

The present article is a continuation of a previous article, and shows in detail some of implemented algorithms in the program package. Using this basis, the functioning of the ability of programs, becomes more deeply understandable.

Keywords: symmetric encryption, asymmetric encryption, steganography, program protection