Budapesti Műszaki Főiskola
Kandó Kálmán Villamosmérnöki Főiskolai Kar
Műszertechnikai és Automatizálási Intézet

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

A nyílt kulcsú titkosítás és a digitális aláírás

Szerző: **Fuszenecker Róbert**
villamosmérnöki szak, II. évfolyam

Konzulens: **Dr. Tuzson Tibor**
főiskolai docens

Budapest, 2006

Tartalomjegyzék

Terjesztés

Váljon ez a dokumentum minden érző lény javára!

Bevezetés

Ezen dolgozat megírására azért vállalkoztam, hogy megismertessem az Olvasót a nyílt kulcsú titkosítással – és általában mindazzal, ami ahhoz szükséges, hogy az Olvasó belássa, a titkosítás nem feltétlenül egy elvont, csak matematikusok által érthető tudomány. A témaválasztást az is indokolta, hogy a mai, modern világunkban egyre nagyobb hangsúlyt kap az interneten történő vásárlás (csúnya szóval: online kereskedelem) és az internetes ügyintézés, ezek pedig elképzelhetetlenek a nyílt kulcsú titkosítás és a digitális aláírás nélkül.

A dolgozat első részében megismerkedünk a nyílt kulcsú titkosításhoz vezető rögös úttal: belátjuk, hogy a „hagyományos” titkosító eljárások nem használhatók fel a megfelelő kiegészítés nélkül. Sajnos a dolgozat terjedelme nem teszi lehetővé, hogy minden nyílt kulcsú titkosító algoritmussal megismerkedjünk, de törekedni fogok arra, hogy a ma használt módszerek közül a két legfontosabbat – a Diffie-Hellman kulcscsere algoritmust és az RSA algoritmust kellő mélységben megismertessem az Olvasóval. Ehhez persze szükséges lesz bizonyos matematikai háttér megléte, de megnyugtatásként elmondhatom, hogy a hatványozásnál bonyolultabb művelettel sehol sem fogunk találkozni. Talán éppen az algoritmusok ilyen egyszerű és elegáns volta fogja legjobban megragadni a Tisztelt Olvasó fantáziáját.

És hogy mire lehet mindezt használni? – tehetnénk fel a kérdést. Erre ad választ a dolgozat utolsó része, ahol néhány, a munkám során készített képernyőképen keresztül mutatom be, hogy a gyakorlatban hol használjuk (akár tudatosan, akár anélkül, hogy észrevennénk) a nyílt kulcsú titkosítást.

Kívánom, hogy legalább akkora örömöt találjon a Tisztelt Olvasó ezen dolgozat olvasásában, mint amekkora örömet nekem okozott a megírása.

A szerző

1. A kommunikáció alapjai

Ebben a fejezetben a kommunikáció, az információáramlás legegyszerűbb modelljeivel fogunk megismerkedni, felvetjük továbbá a titkosítás szükségességét akkor, ha olyan adatokat kívánunk célba juttatni, melyek kizárólag csak az arra illetékes feleknek állhatnak rendelkezésére.

1.1. A kommunikáció modellje

Napjainkban, hála az telefonvonalaknak és az internetnek, egyre nagyobb szerepet kap a kommunikáció. Szinte el sem tudnánk képzelni, hogy úgy teljen el egy napunk, hogy közben nem vesszük igénybe a modern civilizáció áldásos (vagy éppenséggel átkozott) távközlő eszközeit. Persze nem csak akkor kommunikálunk, amikor ezen segédeszközöket használjuk, hanem minden olyan alkalommal kommunikációról beszélünk, mikor információt juttatunk el valakihez, valamihez.

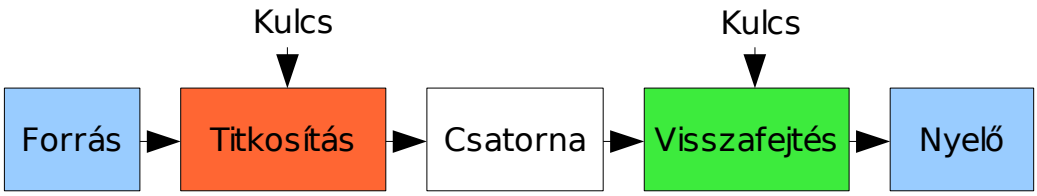
Ahhoz, hogy a kommunikáció létrejöhessen, jelen kell lennie valakinek (vagy valaminek), ami az információ forrása, egy másik valaminek, ami az információáramlás célja, és szükséges egy közvetítő közeg, ami eljuttatja az információt a forrástól a célig.

Azt a résztvevőt, akitől az információáramlás kiindul **forrás**nak nevezzük, míg az információáramlás célja a **nyelő**. A kettő között foglal helyet a **csatorna**, amely lehetővé teszi, hogy a kommunikáció létrejöhessen. Egy előadás során az előadó tekintendő a forrásnak, a hallgatóság a nyelőnek, és a levegő (vagy éppen a levegő-mikrofon-erősítő-hangszóró-levegő lánc) szolgál csatornaként. Természetesen állhatna itt egy másik példa is: a modern informatika korában a forrás és a nyelő egy-egy számítógép, és a csatorna pedig az azokat összekötő a hálózat.

1.2. Kommunikáció titkosított csatornán

Sajnos a rendelkezésre álló csatornáink gyakran olyanok, hogy azokon egyszerre sokan kommunikálnak, vagyis a csatornát több fél megosztottan éri el. Ha mégsem így lenne, akkor is fennáll az veszélye, hogy az általunk biztonságosnak hitt csatornát „megbontják”, és az illetéktelenek számunkra kényes információkhoz juthatnak. Ennek megelőzésére alkalmazzuk a titkosítást.

Ebben az esetben egy kicsivel bonyolultabb modellt kell használnunk, de ez mindössze két kiegészítő lépést jelent:



Még mielőtt az információ belépne a csatornába (ahol tulajdonképpen bárki megszerezheti) titkosítjuk, és ha az információ elhagyja a csatornát, visszafejtjük azt. Ebből a forrás és a nyelő – elvileg – nem vesz észre semmit, hiszen ők a csatorna részének tekintik mind a titkosító, mint a visszafejtő „berendezést”.

Nem lenne biztonságos, ha a titkosítás során mindig pontosan ugyanúgy módosítanánk az információfolyamot, ezért alkalmazunk egy paramétert, a **kulcs**ot, ami némiképpen módosítja a titkosítás menetét, ezzel változatosságot visz a rendszerbe, így nagymértékben csökken annak a valószínűsége, hogy az ellopott titkos információból valaki visszaállítsa az eredeti információt.

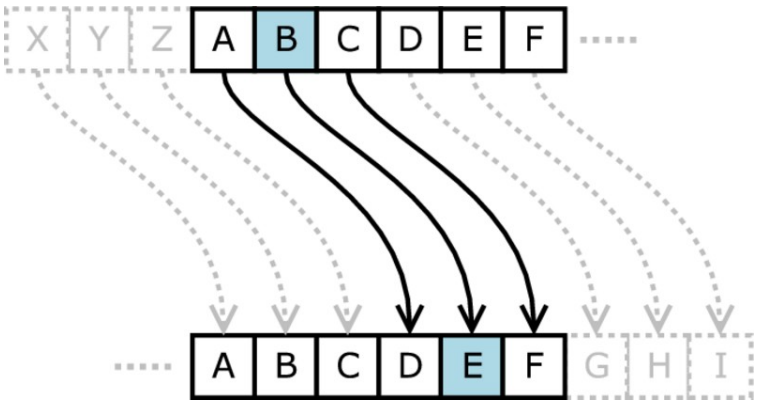
2. A szimmetrikus kulcsú titkosítás

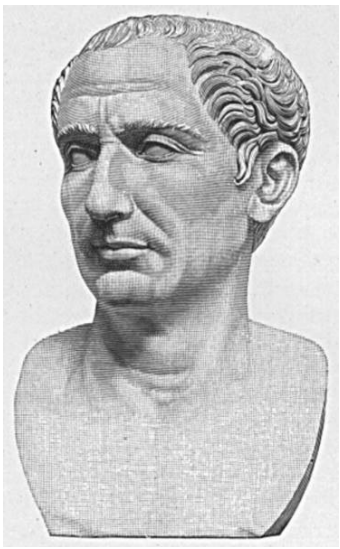
Természetesen a visszafejtő félnek ismernie kell a titkosítás során használt kulcsot, hiszen (elvileg) csak a kulcs ismeretében alakítható vissza az eredeti információ (a nyelő dolgát nem szándékozunk megnehezíteni azzal, hogy nem adjuk meg a kulcs értékét).

Ha a titkosításhoz használt kulcs és a visszafejtéshez használ kulcs ugyanaz, akkor **szimmetrikus kulcsú titkosítás**ról beszélünk. Ekkor olyan titkosító eljárást, algoritmust használunk, melynek létezik inverz művelete.

Ha ugyanannak a kulcsnak a segítségével „visszafelé” végezzük el a titkosító eljárást, akkor visszafejtésről beszélünk.

Ennek egyik, már klasszikussá vált példája a Caius Iulius Caesar által kétezer éve kidolgozott módszer.





Caius Iulius Caesar
Kr. e. 100. – Kr. e. 44.

Módszerének az a lényege, hogy a forrástól származó szöveget betűire bontjuk (ebben az esetben a latin ABC betűire), és minden betűhöz hozzárendelünk egy számot (A=1, B=2, stb.). A titkosított szöveget úgy kapjuk, hogy minden betű kódjához hozzáadunk 3-at. A „B” betű kódja kettő. 2+3 = 5, ami éppen az „E” betű kódja. Így lesz a képen látható módon „B” betűből „E” betű.

Visszafejtéskor természetesen az összeadás inverz műveletét (a kivonást) végezzük el (ugyancsak a 3 értékkel, hiszen most ez a *kulcs*). 5 – 3 = 2. Így kapunk a titkosított „E” betűből „B”-t.

Hogy miért 3-at választott a római hadvezér, és miért nem 4-et vagy 5-öt? Választhatott volna 5-öt is, vagy bármilyen más számot, de abban a pillanatban ő úgy döntött, hogy a kulcs értéke legyen 3.

Nem lenne helyes azt állítani, hogy a Caesar-módszer jelenti a mai kriptográfia (a titkosítás tudományának) legbonyolultabb és legbiztonságosabb algoritmusát. Ennek a módszernek az a legnagyobb hátránya, hogy nagyon egyszerű megtörni: megfigyelhetjük, hogy egy adott nyelvben melyek a leggyakrabban előforduló betűk. A magyar nyelv esetében ez minden bizonnyal az „e” betű. Ezek után már nem lenne más dolga a törést megkísérlőnek, mint hogy megállapítsa, a titkosított szövegben melyik betű fordul elő a leggyakrabban. A két betű kódjának különbsége éppen a kulcsot adja.

Hogy elkerülhető legyen a titkosított információk illetéktelenek által történő visszafejtése, a matematikusok (kriptográfusok) rengeteg bonyolult módszert dolgoztak ki az évszázadok során. Ezeket persze nem áll módomban részletesen ismertetni, mindenesetre elmondhatjuk, hogy ezek száma igen nagy, vannak gyorsabban elvégezhetők, és lassabban elvégezhetők, könnyen törhetők és nehezen törhetők, stb. Azt, hogy ezek közül melyiket használjuk attól függ, hogy az adott alkalmazásban mire van szükségünk. Caesar esetében elegendő volt egy ilyen egyszerű módszer alkalmazása is, hiszen ellenségei így sem tudták visszafejteni az általa kiadott parancsokat, és el is veszítették a csatát.

Ma már persze rendelkezünk olyan számítógépekkel, melyek sokkal bonyolultabb módon titkosított adatokat is meg tudnak törni.

A teljesség igénye nélkül nézzünk meg néhány szimmetrikus titkosító algoritmust, melyeket igen gyakran használunk a mindennapi életben:

DES – ezt a titkosító algoritmust a 70-es évek elején fejlesztették ki az IBM munkatársai az Amerikai Egyesült Államosban. A titkosítandó üzenetet 64 bites részekre, szeletekre bontotta, és azokat külön-külön titkosította egy 56 bit méretű kulcs segítségével. Jogos a múlt idő használata, hiszen ma már ez az algoritmus nem használatos, éppen azért, mert a kis kulcsméret miatt könnyen törhető.

3DES – Az előbbi algoritmus hibáit azzal próbálja megszüntetni, hogy háromszor végzi el a titkosítást, így a törés is bonyolultabbá válik. Sajnos mindkét algoritmus mára már csak történelmi örökség, gyakorlati jelentősége nincsen.

AES – Az Egyesült Államok kormánya 1997 januárjában kiírt egy versenyt, amelyben a legjobb titkosító algoritmust keresték országuk (és a világ) számára. Rengeteg jelentkező közül a belga fejlesztésű Rijndael lett a győztes, amit azóta szabványosítottak. A Rijndaelt ma AES (Advanced Encryption Standard – fejlett titkosító szabvány) néven ismerünk. Ez az algoritmus 128 bites blokkokat titkosít és a kulcs mérete 128, 192 vagy 256 bit hosszúságú lehet.

CAST – ez egy kanadai mérnökök által kifejlesztett titkosító algoritmus, rendkívül biztonságosnak bizonyult.

Twofish – a twofish egyike annak az 5 algoritmusnak, melyek bekerültek az előbb ismertetett verseny utolsó körébe. A twofish egyik legfontosabb tulajdonsága (a biztonsága mellett), hogy mindenféle megkötés nélkül használható akár kereskedelmi célokra is.

XTEA – ez a rövidítés az eXtended Tiny Encryption Algorithm kifejezésből származik, ami magyarul annyit tesz, hogy „kiterjesztett kicsi titkosító algoritmus”. A neve pontosan fedi legjellemzőbb tulajdonságát: kicsi és biztonságos. A megvalósításához szükséges C nyelvű kód alig 10-20 soros:

```
void encipher(unsigned int num_rounds, unsigned long* v, unsigned long* k) {
   unsigned long v0=v[0], v1=v[1], i;
   unsigned long sum=0, delta=0x9E3779B9;
   for(i=0; i<num_rounds; i++) {
      v0 += ((v1 << 4 ^ v1 >> 5) + v1) ^ (sum + k[sum & 3]);
      sum += delta;
      v1 += ((v0 << 4 ^ v0 >> 5) + v0) ^ (sum + k[sum>>11 & 3]);
   }
   v[0]=v0; v[1]=v1;
}

void decipher(unsigned int num_rounds, unsigned long* v, unsigned long* k) {
   unsigned long v0=v[0], v1=v[1], i;
   unsigned long delta=0x9E3779B9, sum=delta*num_rounds;
   for(i=0; i<num_rounds; i++) {
      v1 -= ((v0 << 4 ^ v0 >> 5) + v0) ^ (sum + k[sum>>11 & 3]);
      sum -= delta;
      v0 -= ((v1 << 4 ^ v1 >> 5) + v1) ^ (sum + k[sum & 3]);
   }
   v[0]=v0; v[1]=v1;
}
```

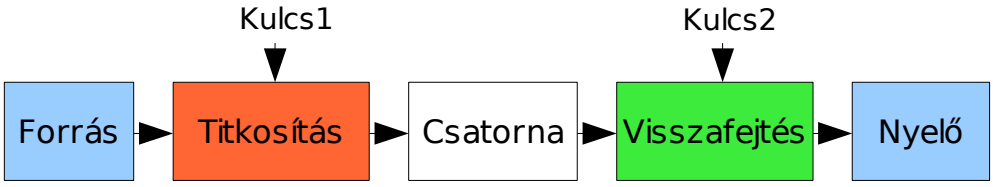
Sajnos, a szimmetrikus kulcsú titkosítás nem használható minden körülmények között. Ahhoz, hogy az információ visszafejthető legyen, a visszafejtő oldalon ugyanazt az algoritmust kell alkalmazni. Ez nem túl szigorú kikötés, és általában minden gond nélkül megoldható.

Nagyobb problémát okoz viszont a kulcs biztonságos eljuttatása a visszafejtés helyére. Gondoljuk csak meg! A csatornát nem használhatjuk a kulcs átvitelére, hiszen a csatornát éppen azok „figyelik”, akik elől el akarjuk rejteni az átvitt információinkat.

Gondolhatnánk azt is, hogy elegendő egyszer megegyezni egy kulcsban, és az idők végezetéig azt használni. Sajnos ez nem járható út, hiszen a titkosító algoritmusok is fejlődnek, gyakran más-más méretű kulcsot igényelnek, így a kulcsok akkor is elavulnak, ha közben senki sem töri meg azokat. A helyzet ettől bonyolultabb, hiszen a kulcsokat a biztonság növelése érdekében is célszerű időnként cserélni, mert ha ma kicseréljük a kulcsot, és a „sötét oldal” éppen holnap végezne a régi kulcsunk törésével, akkor kezdheti elölről az egész törési folyamatot.

3. A nyílt kulcsú titkosítás

Az előbb felmerült az a probléma, hogy hogyan juttathatnánk el a kulcsot a visszafejtés helyére úgy hogy a „lehallgatók” semmiképpen se szerezhessék meg azt. Erre nyújt megoldást a nyílt kulcsú titkosítás.



A nyílt kulcsú titkosítás egyik legérdekesebb tulajdonsága, hogy a titkosításhoz és a visszafejtéshez más-más kulcsot használ. Ennek igen nagy a jelentősége, hiszen a *forrás oldalára* elküldhetjük a titkosításhoz szükséges kulcsot (**nyilvános kulcs**), és mindezt egy nem biztonságos csatornán keresztül is megtehetjük. A nyilvános kulcs segítségével nem lehet visszafejteni a titkosított információfolyamot, mert a visszafejtéshez szükséges kulcs (**titkos kulcs**) csak a nyelő oldalán található meg, így kizárólag a nyelő képes visszaalakítani a titkosított adatokat.

Ebben a fejezetben megismerjük a két legfontosabb nyílt kulcsú algoritmust, ezek közül az első kifejezetten kulcscsere algoritmus, vagyis arra való, hogy a két kommunikáló fél úgy egyezzen meg egy kulcsban, hogy rajtuk kívül más sehogyan se tudja azt megszerezni (kiszámítani). A második algoritmus ettől érdekesebb, mert ezzel nem csak titkosítani lehet, hanem digitális aláírást is lehet csatolni az átvitt információhoz (erről később lesz még szó).

3.1. Matematikai háttér

Ahhoz, hogy megértsük a nyílt kulcsú titkosító módszereket, meg kell ismerkednünk a felhasznált matematikai összefüggésekkel. A következő részben jóval részletesebben foglalkozunk a matematikai műveletekkel, mint amennyire ezt a titkosító algoritmus ismerete megkövetelné, mégis úgy gondolom, hogy akkor teljes a kép, ha az elejétől a végéig mindent egy helyen találunk.

Az ismertetésre kerülő algoritmusok közös jellemzője, hogy úgynevezett **véges matematikai mezők**ön dolgoznak. Ez bonyolultnak hangzik elsőre, de ne ijedjünk meg ettől a szakmai műszótól. Egész egyszerűen arról van szó, hogy kiválasztunk egy számot (**p**), és kijelentjük, hogy ennél nagyobbat nem alkalmazunk a műveletek paramétereként, és „p”-nél nagyobb számot nem is kaphatunk eredményként sem. Ez utóbbiról külön kell gondoskodnunk a műveletek elvégzése során.

Ha „p” értékét prímszámnak választjuk (a prímszám olyan pozitív egész szám, ami csak önmagával és 1-gyel osztható maradék nélkül), akkor az általunk eddig is ismert matematikai összefüggésekhez hasonlóakkal dolgozhatunk.

Meg kell ismerkednünk az **maradékképzés** nevű művelettel is. Ez azért szükséges, mert minden elvégzett művelet után maradékképzést kell végeznünk, hogy semmiképpen se álljon elő a már kiválasztott „p” értéknél nagyobb szám. A maradékképzés azt jelenti, hogy elvégezzük az osztást „p”-vel, de kivételesen nem a hányadost tekintjük végeredménynek, hanem a maradékot. Például: ha p = 3, akkor 7 : 3 = 2, maradék: 1. Mivel minket csak a maradék érdekel, ezért az „értékes” információ ebben az esetben az *egy*. Mivel nem szeretnénk mindig ilyen bonyolultan leírni a műveletet, vezessünk be egy új „műveleti jelet”, a maradékképzés operátorát: ez legyen a „**mod**”.

Ennek alapján az előbbi maradékképzést így is írhatjuk: 7 mod 3 = 1.

Nézzük most a legfontosabb műveletekre vonatkozó azonosságokat egy véges matematikai mezőn. A „p” jelentse továbbra is a mezőt generáló prímszám értékét!

Összeadás és szorzás

Mindkét műveletet úgy végezzük el, mintha nem véges matematikai mezőkön dolgoznánk, csak a végén maradékképzést alkalmazunk. Például:

C = (a + b) mod p
*D = (a * b) mod p*

Példa:

Legyen p = 7
C = (4 + 2) mod 7 = 6 mod 7 = 6
*D = (4 * 2) mod 7 = 8 mod 7 = 1*

Előjelváltás

A kivonást egy érdekes művelet, mert definíció szerint egy szám kivonása megegyezik egy ellenkező előjelű szám hozzáadásával. Az „ellenkező előjelű szám” azt a számot jelenti, amit az eredeti számhoz hozzáadva nullát kapunk. Lássuk, hogyan történik az előjelváltás véges matematikai mezők esetén:

$$-a \bmod p = p - (a \bmod p)$$

Példa:

Legyen: $p = 7$ és $a = 4$
$-4 \bmod 7 = 7 - 4 = 3$

Ellenőrzés:

$(a + [-a]) \bmod p = 0$
$(4 + 3) \bmod 7 = 7 \bmod 7 = 0$

Multiplikatív inverz képzés ($^1/_a$ vagy a^{-1})

Ha a kivonásra azt mondtuk, hogy az egy ellentétes előjelű szám hozzáadása, akkor az osztásra azt kell mondanunk, hogy ez a szám multiplikatív inverzével való szorzás. Egy „a” szám multiplikatív inverze az a szám, amit „a”-val megszorozva 1-et kapunk. Vagyis:

$$(a^{-1} * a) \bmod p = 1$$

A multiplikatív inverz képzéséhez szükségünk lesz Fermat kis tételére (ezt az Olvasó megtalálja a mellékletek között), ami formailag így néz ki:

$$a^p \bmod p = a$$

Vagyis ha veszünk egy egész számot („a”), és a „p”-edik hatványra emeljük, utána pedig elvégezzük a maradékképzést, akkor végeredményként „a”-t kapunk. Kis matematikai ügyeskedéssel megkaphatjuk a multiplikatív inverz értékét. Osszuk el mindkét oldalt „a”-val:

$$a^{p-1} \bmod p = 1$$

Ha még egyszer elosztjuk mindkét oldalt „a”-val, akkor:

$$a^{p-2} \bmod p = a^{-1}$$

Nem kell mást tennünk „a” multiplikatív inverzének kiszámításához, mint „a”-t a „p-2”-edik hatványra emelni, és „p”-vel maradékképzést végezni.

Példa:

$p = 7$
$a = 2$
$a^{-1} = a^{p-2} \bmod p = 2^{7-2} \bmod 7 = 2^5 \bmod 7 = 32 \bmod 7 = 4$

Ellenőrzés:

$$a * a^{-1} = (2 * 4) \bmod 7 = 8 \bmod 7 = 1$$

Ezzel a véges matematikai mezőkön értelmezett alapműveleteket át is néztük, elérkeztünk hát a dolgozat valódi témájához.

3.2. A Diffie-Hellman kulcscsere algoritmus

Ez a fejezet arról szól, hogy miként tud két kommunikáló fél úgy megegyezni egy közös titkosító kulcsban, hogy a csatornát lehallgató személyek, robotok, számítógépek semmiképpen se tudják azt megszerezni.

Az algoritmust két amerikai mérnök dolgozta ki 1976-ben:

Whitfield Diffie

Martin Hellman

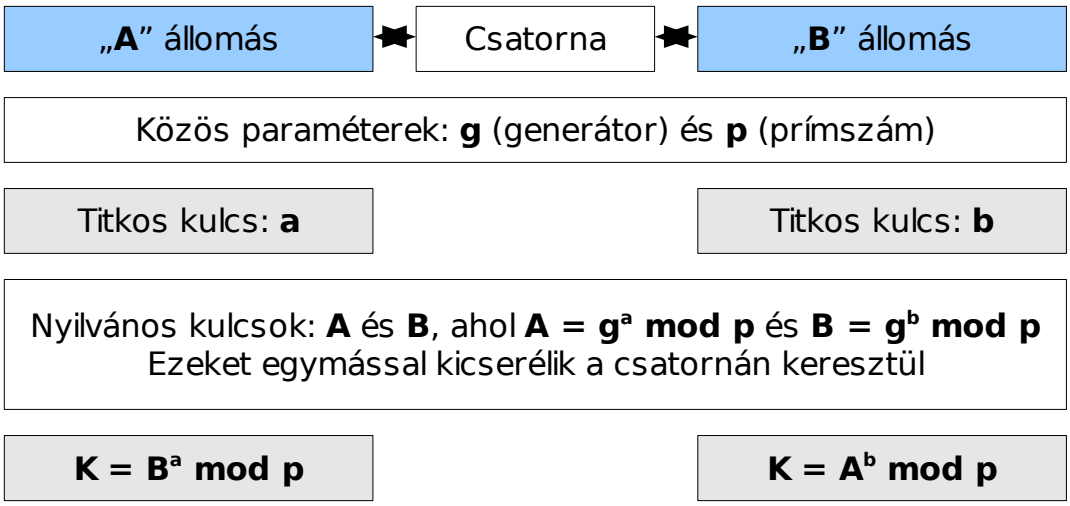
Az algoritmus legérdekesebb tulajdonsága, hogy a kulcsot nem küldjük keresztül a csatornán, legfeljebb a számítás részeredményeit. Ha a számítás olyan bonyolult, hogy egy lehallgató fél számára a részeredmények ismerete kevés a titkosító kulcs kiszámításához, akkor az algoritmus biztonságos.

Az algoritmus úgy működik, hogy mindkét fél azonos műveleteket végez el azonos számokon (a műveletek sorrendje nem feltétlenül azonos, de arról gondoskodni kell, hogy a műveletek felcserélhetők legyenek: kommutatív műveletek szükségesek), ekkor az eredmény is azonos lesz. Ilyen művelet például a hatvány hatványozása (mod p), ami tulajdonképpen a kitevők összeszorzása.

A kommunikáció elején a két fél megegyezik egy „g” számban, és „p” értékében. A „g” számot generátornak nevezzük, értéke általában 2 vagy 5. Az *OpenSSL* alapértelmezés szerint 2-t használ, az *ssh* a 2-t és az 5-öt felváltva alkalmazza. Természetesen más számokat is választhatnának, de az algoritmus így is megfelelően biztonságos lesz.

„p” értéke egy prímszám, ami nagyon nagy (1024-4096 bit hosszú), hogy minél nehezebb dolga legyen annak, aki a titkosított adatok megszerezésére adja a fejét.

A „g” és a „p” nyilvános paraméterek, ezeket nyugodtan átküldhetik a csatornán.



Miután „A” és „B” oldal megegyezett a közös paraméterek értékében, mindkét fél „kitalál” egy-egy véletlenszámot. Ezek értékét gondosan elrejtik, mert ezek lesznek a titkos kulcsok:

„A” oldal titkos kulcsa legyen „a”, „B” oldal titkos kulcsa legyen „b”!

Ezután kiszámítják a nyilvános kulcsok értékét:

$$A = g^a \bmod p \qquad \text{és}$$
$$B = g^b \bmod p$$

A kapott értékeket kicserélik a csatornán keresztül. Mivel „A” értékéből *szinte lehetetlen* „a” értékét kiszámolni, ezért az „A” állomás biztonságban tudhatja titkos kulcsát. Ugyanez „B” állomásra is igaz. Nincs más dolga a két állomásnak, mit a másikuk nyilvános kulcsát a titkos kulcsuk alapján hatványozni, és elvégezni a maradékképzést.

Nézzük, hogy mit kell „A” állomásnak elvégeznie, hogy megkapja a szimmetrikus titkosításhoz szükséges **titkosító kulcs (K)** értékét! Mivel ismeri a „p” és a „B” állomás nyilvános kulcsának értékét, ezért a a titkosító kulcsot így tudja kiszámítani:

$$K_A = B^a \bmod p$$

Ugyanezt „B” is megteszi, az „A” állomás nyilvános kulcsával, és a saját titkos kulcsával:

$$K_B = A^b \bmod p$$

Ha mindketten jól számoltak, akkor ugyanazt az eredményt kapták. Lássuk, hogy miért szükségszerű ez! Az „A” fél számításait így foglalhatnánk össze:

$$K_A = B^a \bmod p$$

Az „A” fél a „B” oldal nyilvános kulcsát emelte az „a”-dik hatványra. Ha behelyettesítjük „B” helyére azt a képletet, ahogyan „B”-t kiszámítottuk, és elvégezzük a hatvány hatványozását (a kitevők összeszorozását), akkor azt kapjuk, hogy

$$K_A = (g^b \bmod p)^a \bmod p = g^{ba} \bmod p$$

Ha ugyanezt „B” is megcsinálja, erre az eredményre jut:

$$K_B = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p$$

Most már csak azt kell belátnunk, hogy a két titkosító kulcs (K_A és K_B) értéke azonos: mivel a szorzás kommutatív művelet, vagyis a két tényező felcserélhető (ab = ba), ezért

$$K = K_A = g^{ba} \bmod p = K_B = g^{ab} \bmod p$$

Nem minden művelet kommutatív: például a vektoriális szorzás vagy a mátrixszorzás nem az. Ezért kell előírnunk, hogy a műveletnek mindenképpen kommutatívnak kell lennie.

Nézzünk egy egyszerű példát a Diffie-Hellman kulcscsere algoritmusra:

Legyen:

$$g = 2$$
$$p = 257$$

Legyen továbbá „a” értéke (amit csak „A” oldal ismer) 19,
és „b” értéke (amit csak „B” oldal ismer) 15.

Az „A” állomás nyilvános kulcsa:

$$A = g^a \bmod p = 2^{19} \bmod 257 = 524288 \bmod 257 = 8$$

És a „B” állomás nyilvános kulcsa:

$$B = g^b \bmod p = 2^{15} \bmod 257 = 32768 \bmod 257 = 129$$

Az „A” oldal által kiszámított titkosító kulcs értéke:

$$K_A = B^a \bmod p = 129^{19} \bmod 257 =$$
$$1262421829610131696467448865211057404556 9 \bmod 257 = 225$$

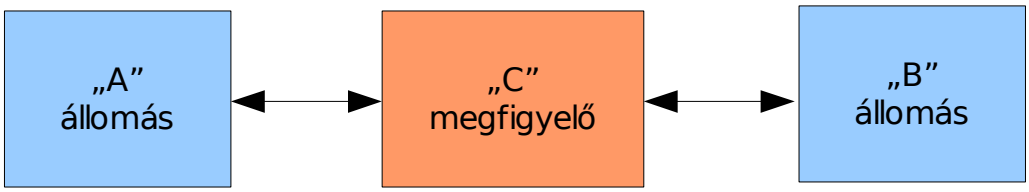
És „B” oldal által kiszámított titkosító kulcs értéke:

$$K_B = A^b \bmod p = 8^{15} \bmod 257 = 35184372088832 \bmod 257 = 225$$

Látható, hogy mindketten ugyanazt az értéket kapták. Ezt a kulcsot most már arra tudják használni, hogy egy kiválasztott szimmetrikus kulcsú algoritmussal titkosítsák a köztük folyó kommunikációt.

3.3. A „középen lévő ember”

Sajnos a Diffie-Hellman kulcscsere algoritmusnak van egy súlyos hiányossága. Mivel a kommunikáció során alkalmazott csatorna nem tekinthető biztonságosnak, ezért nem tudhatjuk, hogy valóban azzal folytatjuk-e a kommunikációt, akivel szeretnénk. Ha pizzát szeretnék rendelni, és felhívom azt a telefonszámot, amit egy (nem megbízható) telefonkönyvből másoltam ki, akkor nem lehetek biztos benne, hogy valóban egy pizzériával beszélek, még akkor sem, ha a telefonvonal másik végén található hölgy mégoly határozottan állítja is. Ez már súrolja a paranoia határát, de ne feledjük el, hogy senki sem szeretné, ha mondjuk a bankkártyaszáma illetéktelenek kezébe kerülne.



Ez az ábra azt a helyzetet mutatja, mikor „A” állomás azt hiszi, hogy „B”-vel kommunikál, vele egyeztette a nyilvános paramétereket, vele cserélt nyilvános kulcsot, közben pedig egy közbülső emberrel kommunikál. A „C” állomás, a lehallgató, az illetéktelen, a gonosz azt állítja „B”-nek, hogy ő valójában „A”, és ennek rendje és módja szerint elvégzi „B”-vel a kulcscserét. Ekkor „C” az „A” és „B” állomás közötti teljes kommunikációt megfigyelheti, bizalmas adatokhoz (jelszavakhoz, személyi adatokhoz, bankkártyaszámokhoz) férhet hozzá.

Mivel „C” állomásnak logikusan a két kommunikálni szándékozó fél között kell helyet foglalnia, ezért ezt a támadási módot a „középen lévő ember” problémájának (angolul: man in the middle) nevezzük. Ez a Diffie-Hellman algoritmus legnagyobb hibája. Természetesen a matematikusok és a titkosítási szakemberek megtaláltak megoldást: a probléma ugyanis csak addig áll fenn, amíg nem tudjuk leellenőrizni, hogy valóban azzal kommunikálunk, akivel szeretnénk. Az ellenőrzésre és az azonosításra a digitális aláírás szolgál (a gyakorlatban az *ssh* ezt a módszert használja).

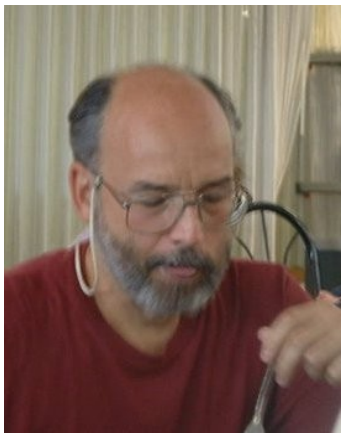
3.4. Az RSA algoritmusa

Ezt az algoritmust az Amerikában található MIT intézet 3 mérnöke dolgozta ki 1977-ben.

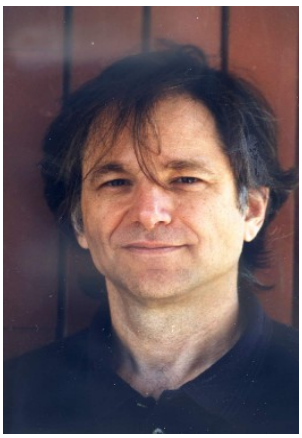
Pontosan nem tudjuk, hogy maga az algoritmus mióta létezik, mert 1973-ban brit matematikusok leírtak egy ekvivalens rendszert, de nem hozták nyilvánosságra. Az amerikai mérnökök 1977-ben publikálták saját titkosító algoritmusukat, így vált végül közkinccsé ez a nagyszerű nyílt kulcsú titkosító rendszer.

De ki volt ez a 3 tudós? Kinek köszönhetjük mindezt?

Ron Rivest



Adi Shamir



Leonard Adleman

Az algoritmus neve hármuk vezetéknevének kezdőbetűiből származik, azonban Adleman professzor nem hitt az algoritmus sikerében, ezért azt kérte, hogy legyen ő az utolsó a sorban.

Az általuk kidolgozott algoritmus Fermat kis tételén alapszik (amit a Tisztelt Olvasó megtalál a mellékletek között). A tétel egyszer már előkerült ebben a dolgozatban, de azért elevenítsük fel még egyszer:

$$a^p \bmod p = a$$

Vagyis ha választunk egy „p” prímszámot, és egy tetszőleges „a” számot („a” és „p” relatív prímek, vagyis 1-en kívül más egész számmal nem oszthatók maradék nélkül), akkor ha „a”-t a „p”-edik hatványra emeljük, és elvégezzük a maradékképzést, éppen „a”-t kapunk.

Most az Olvasó az RSA algoritmushoz tartozó levezetést fogja megismerni. Ha azt gondolná, hogy ez csak bonyolult matematikai formulákon keresztül lehetséges, akkor kellemesen fog csalódni.

Először osszuk el mindkét oldalt „a”-val („a” értéke ugyebár nem lehet nulla, de ha mégis az lenne, akkor a titkosításnak nem lenne értelme, hiszen nulla a „p”-ediken is nulla)! Ekkor arra jutunk, hogy

$$a^{p-1} \bmod p = 1$$

Ha tréfálni szeretnék, azt mondanám, hogy ezzel a bonyolult formulával bármikor kiszámíthatjuk az „1' pontos értékét. Természetesen nekünk nem ez a célunk. Végezzünk egy formai módosítást: cseréljük ki a „p” betűt „q” betűre! Ekkor „q” is egy prímszám lesz, csakúgy, mint „p”.

$$a^{q-1} \bmod q = 1$$

Mindkét egyenlet jobb oldala 1. Emeljük ez első egyenletet a (q-1)-edik hatványra, a másodikat pedig a (p-1)-edik hatványra! Azt kapjuk, hogy

$$a^{(p-1)(q-1)} \bmod p = 1^{(q-1)} = 1 \quad \text{és}$$
$$a^{(p-1)(q-1)} \bmod q = 1^{(p-1)} = 1$$

Mivel mindkét egyenlet jobb oldala 1, ezért (a kínai maradék-tétel értelmében, ami szintén megtalálható a mellékletben) a két egyenlet együtt is igaz (itt most a maradékképzés művelete az érdekes):

$$a^{(p-1)(q-1)} \bmod pq = 1$$

Ahhoz, hogy később egyszerűbb dolgunk legyen, emeljük az egyenlet mindkét oldalát a „k”-adik hatványra („k” egy tetszőleges egész szám)!

$$a^{k(p-1)(q-1)} \bmod pq = 1^k = 1$$

Szorozzuk most meg mindkét oldalt „a”-val!

$$a^{k(p-1)(q-1)+1} \bmod pq = a$$

Ezzel végeztünk is. De hogyan lehet ezt titkosításra használni? A lényeg az, hogy egy bonyolultnak tűnő matematikai kifejezéssel „a”-ból kiszámítottuk „a” értékét. És ez a dolog lényege! Ha a k(p-1)(q-1)+1 értékét fel tudjuk bontani két szám szorzatára,

$$k(p-1)(q-1)+1 = ed$$

akkor azt kapjuk, hogy

$$a^{ed} \bmod pq = a^{k(p-1)(q-1)+1} \bmod pq = a$$

Ha a két hatványozást egyszerre végezzük el, akkor sehova sem jutunk. Ezért a hatványozásokat külön-külön végezzük: az egyiket titkosításkor, a másikat visszafejtéskor.

Mivel „e” és „pq” értékét ki kell adnunk, ezért ezek lesznek a nyilvános kulcs összetevői, míg „d” értékét megőrizzük, ez lesz a titkos kulcs.

3.4.1. Titkosítás és visszafejtés

A titkosítást így kell elvégezni: legyen „m” a titkosítandó üzenet (a szimmetrikus algoritmusokhoz használható titkosító kulcs). Ekkor a titkosított üzenet:

$$M = m^e \bmod pq$$

A visszafejtés is hasonlóan egyszerű:

$$m = M^d \bmod pq$$

De hogyan bontsuk fel a k(p-1)(q-1)+1 értéket két szám szorzatára, ha „k”-t nem ismerjük? Pont ez a lényeg! A (p-1)(q-1)+1 kifejezés nem mindig bontható fel

célszerűen (pl. e = 1 esetén nem beszélhetünk titkosításról), ezért beleviszünk egy szabadsági fokot az egyenletbe. „k” értékét mi határozhatjuk meg úgy, hogy „e” és „d” megfelelő értéket vehessen fel.

Példa

Legyen:

$p = 23$
$q = 43$

Ekkor:

$pq = 989$

Kezdetben legyen k = 1, ekkor

$k(p-1)(q-1)+1 = 1*(23-1)(43-1)+1 = 22*42+1 = 925$

Szerencsére a 925 felbontható két szám szorzatára, ez a két szám pedig az e = 25 és d = 37. Ha nem sikerült volna a felbontás, akkor k = 2-vel, majd k = 3-mal, stb. kellene próbálkoznunk.

A nyilvános kulcs ezek szerint: [e, pq] = [25, 989], míg a titkos kulcs: [d, pq] = [37, 989].

Ha valaki a például az m = 104-es számot szeretné elküldeni nekünk, csak annyit kell tennie, hogy a nyilvános kulcsunk ismeretében kiszámítja a következő értéket:

$M = m^e \bmod pq = 104^{25} \bmod 989$
$M = 26658363314874199930406400339519781528339147548 2624 \bmod 989$
$M = 417$

Az M = 417 érték a titkosított üzenet, amit már bátran átküldhet nekünk a nem megbízható csatornán. Mi a visszafejtéshez az „M” értékét „d”-edik hatványra emeljük, és elvégezzük a maradékképzést:

$m = M^d \bmod pq = 417^{37} \bmod 989$
$m = 8\,811\,179\,202\,078\,532\,855\,939\,555\,632\,397\,829\,696\,546\,060\,991\,941\,269\,813\,042\,461\,998\,861\,137\,523\,203\,132\,627\,473\,086\,977\,926\,177 \bmod 989$
$m = 104$

Bonyolult számítás, de a mai számítógépeknek ez nem jelent kihívást. Végeredményül 104-et kaptunk, azt az értéket, amit küldeni szándékoztak nekünk.

3.4.2. A digitális aláírás készítése és ellenőrzése

Mivel „e” és „d” formailag ugyanúgy néz ki, ezért mi dönthetjük el, hogy a kettő közül melyiket tesszük közzé (nyilvános kulcsként), és melyiket tartjuk meg (titkos kulcsként).

Végezhető olyan titkosítás is, mikor a titkosításhoz a titkos kulcsunkat használjuk, a visszafejtéshez pedig a nyilvános kulcsunk szükséges. Ennek látszólag semmi értelme,

hiszen miért titkosítanánk valamit, amit bárki visszafejthet. A dolog lényege éppen ebben van: ha valaki találkozik egy dokumentummal, és azt mondják neki, hogy az tőlünk származik, akkor jogosan jelenik meg benne a kétség szikrája.

De ha a dokumentum hitelességéről meg tud győződni, (például a nyilvános kulcsunk segítségével), akkor elfogadhatja a dokumentumot tőlünk származónak. Ezt úgy oldjuk meg, hogy előállítunk egy nagyon nagy számot (128-160 bit hosszút), ami csakis a mi dokumentumunkra jellemző. Erre vannak már nagyon jó kivonatkészítő algoritmusok (MD5, SHA1, SHA256), és ezt a „számot" titkosítjuk a *titkos* kulcsunkkal. Ezzel a művelettel **aláírtuk** a dokumentumot. Az aláírást csatoljuk a dokumentumhoz, ezzel mindent megtettünk, amit csak tehettünk.

Ha valaki kétségbe vonja a dokumentum hitelességét, kiszámítja a dokumentumra jellemző számot (ugyanazzal az algoritmussal, amit mi alkalmaztunk), majd a nyilvános kulcsunk felhasználásával „visszafejti" az aláírást. Ha a két érték egyezik, akkor a dokumentum valóban tőlünk származik, és senki sem módosította az átvitel során. Ez a művelet az **aláírás ellenőrzése**.

Ezzel a módszerrel írja alá az *ssh* a Diffie-Hellman nyilvános kulcsokat, elkerülve azt, hogy egy „középen levő ember" illetéktelenül hozzáférjen az információfolyamhoz.

Példa:

Legyen a dokumentumunk a következő üzenet:

„Az erőszak a gyengék végső menedéke"
Isaac Asimov: Alapítvány

Tegyük fel, hogy erre jellemző egyedi szám (kivonat, hash) s = 67. Nyilván nem annyi, de a példa kedvéért fogadjuk el ezt az egyszerűsítést. Ekkor dokumentumhoz tartozó digitális aláírás:

$$S = s^d \bmod pq = 67^{37} \bmod 989$$
$$S = 36\,708\,589\,715\,158\,079\,306\,312\,158\,856\,962\,101\,838\,057\,111\,073$$
$$092\,849\,079\,204\,507\,042\,227 \bmod 989$$
$$S = 53$$

Vagyis a digitális aláírás ebben az esetben 53. Nem hangsúlyoztam eléggé, de most pótlom a mulasztásomat: mivel a kivonat (elvileg) egyedi, ezért a digitális aláírás is az. Szinte lehetetlen, hogy két dokumentumhoz ugyanaz a kivonat (és ugyanaz a digitális aláírás) tartozzon, ezért alkalmas az aláírás az eredetiség igazolására.

Most nézzük meg, hogyan lehet az aláírást ellenőrizni. A bizalmatlan fél megkapta Isaac Asimov bölcs mondását, és szeretne meggyőződni arról, hogy az valóban tőlünk származik-e. Ekkor kiszámítja a kivonatot, és ő is 67-et kap. Ha elvégzi a digitális aláírás visszafejtését, erre az eredményre jut:

$$s = S^e \bmod pq = 53^{25} \bmod 989$$
$$s = 1279077148361051944334279126645199622946069 3 \bmod 989$$
$$s = 67$$

Azt találja, hogy az általunk kibocsátott kivonat és az általa kiszámított kivonat ugyanaz, 67, vagyis tényleg tőlünk származik a dokumentum (a bölcsesség természetesen nem).

3.5. A Diffie-Hellman és az RSA törése

Ebben a részben azt nézzük meg, hogy miért tekinthető mind az Diffie-Hellman kulcscsere algoritmus, mint az RSA algoritmus biztonságosnak. Mivel két, nagyon eltérő módszerről van szó, külön-külön kell megvizsgálnunk őket.

Lássuk először a Diffie-Hellman kulcscsere algoritmus törését!

A törést megkísérlő fél akkor tudja kiszámítani a közös titkosító kulcs értékét, ha megszerzi valamelyik kommunikáló fél titkos kulcsát. A másik fél nyilvános kulcsának és az egyik fél titkos kulcsának ismeretében könnyen meghatározhatja a titkosító kulcsot (csak egy hatványozást kell elvégeznie). A nyilvános kulcsokból rendkívül nehéz kiszámolni a titkos kulcsot. Tegyük fel, hogy a törő fél ismeri „A" állomás nyilvános kulcsát, és ebből igyekszik meghatározni a titkos kulcsok egyikét:

$A = g^a \bmod p$ ebből
$g^a = px + A$

Példa:

$4 = 2^a \bmod 7$
$2^a = 7x + 4$

De mennyi „x" értéke és mennyi „a" értéke? Egy kétismeretlenes egyenletet kaptunk, amihez csak egy egyenletünk van. Elvileg ezt nem tudjuk megoldani.

Ebben az egyszerű esetben „ráérezhetünk", hogy $4*7+4 = 32 = 2^5$, vagyis „a" értéke 5, de több ezer bites számok esetén erre esélyünk sincsen. Ezért bátran kijelenthetjük, hogy kellően nagy „p" esetén nem lehet „a"-t kiszámítani (rövid idő alatt). És milyen hosszú legyen „p"? A www.ha5kfu.hu nevű számítógép /etc/ssh/moduli fájljában 1024 bites prímszámok találhatók.

Mivel egész számokon végezzük a műveleteket, és logaritmus függvényt is kell alkalmaznunk, ezt a problémát **diszkrét logaritmus problémá**nak (DLP) nevezzük.

Az RSA törése sem egyszerű feladat. A titkosított üzenetből éppoly értelmetlen lenne a visszafejtő kitevő (titkos kulcs) kiszámítása, mint a Diffie-Hellman algoritmus esetén a hatványkitevő meghatározása, mert ez is kétismeretlenes egyenlethez vezetne. Ha a törést megkísérlő személy végezne egy próbatitkosítást, erre az eredményre jutna:

$M = m^e \bmod pq$ és
$m = M^d \bmod pq$ ebből:
$M^d = kpq + m$

A visszafejtő kitevő („d") meghatározása azért nehéz, mert ismeri ugyan „m", „e" és „M" értékét, de „d" és „k" ismeretlen, csakúgy, mint az előző esetben.

Ezért az RSA törésére más megoldások születtek a történelem folyamán: ha meg lehetne határozni „p" és „q" értékét a „pq" szorzatból (ami ugye a nyilvános kulcs része), akkor könnyűszerrel ki lehetne számolni a visszafejtő kulcs értékét. Hogy ezt elkerüljük, „p" és „q" értékét olyan nagyra választjuk, hogy a prímszámokra bontás lehetőleg évekig-évtizedekig-évszázadokig tartson. Ez az **egész felbontásának problémája** (integer factorization problem – IFP). A „p" és „q" tipikusan 1024-2048 bit hosszú, a szorzatuk pedig 2048-4096 bites.

A teljesség kedvéért álljon itt egy prímtényezőkre bontó algoritmus: a függvény bemenő paramétere a „pq" szorzat, visszatérési értéke pedig a két prím értéke.

```
NUM[2] factorize(NUM pq) {
   f = sqrt(pq);

   while (pq % f)
      f -= 2;

   p = f;
   q = pq / f;

  return (p, q);
}
```

Az algoritmus meghatározza „pq” négyzetgyökét, és megpróbálja „f”-fel elosztani „pq”-t. Ha sikerül, vagyis a maradékképzés 0 értéket ad vissza, akkor f értéke éppen „p”, és „q” már könnyen kiszámítható. Ha a maradékképzés nem nulla értéket ad vissza, akkor nem volt sikeres az osztás, csökkenti „f” értékét 2-vel (mivel csak páratlan számokkal kíséreli meg az osztást), és újra képezi a maradékot. Arról természetesen gondoskodni kell, hogy „f” értéke páratlan legyen.

3.6. Hogyan találhatunk nagy prímszámokat

A történelem során rengeteg prímszámkereső algoritmus látott napvilágot, Terjedelmi okokból azonban csak hárommal tudunk részletesen megismerkedni.

A legelső, és legkevésbé hatékony **a nyers erő módszere:** definíció szerint a prímszám olyan pozitív egész szám, ami csak saját magával és az 1-gyel osztható maradék nélkül.

```
bool is_prime(NUM p) {
   NUM q;

   for (q = 2; q <= p-1; q++)
      if (p mod q == 0)
         return FALSE;

   return TRUE;
}
```

Ez az algoritmus 2-től p-1-ig próbál olyan „q”-t találni, amivel „p” osztható. Ha nem sikerül ilyet találni, akkor a „p” számot prímszámnak tekinti.

Ettől jobb eredményt nyújt a következő módszer: **a prímszámjelöltek keresése**. Egy tétel szerint minden prímszám felírható 6k + 1 vagy 6k – 1 alakban. Ezt könnyű belátni:

a 6k + 0 alakú számok oszthatók 6-tal, tehát nem prímszámok
a 6k + 2 alakú számok oszthatók 2-vel, tehát nem prímszámok
a 6k + 4 alakú számok oszthatók 2-vel, tehát nem prímszámok
a 6k + 3 alakú számok oszthatók 3-mal, tehát nem prímszámok

Maradnak tehát a 6k + 1 és a 6k – 1 alakú számok. Vagyis ezeken a számokon már érdemes lehet a nyers erő módszerét alkalmazni.

A legelegánsabb prímszámkereső módszer Fermat kis tételén alakul. Ez a **Fermat-féle statisztikus módszer**. Már harmadszor találkozunk ezzel az érdekes matematikai igazsággal:

$$a^p \bmod p = a$$

Ez a tétel akkor igaz, ha „p” egy prímszám, valamit „a” és „p” relatív prímek. Ha néhány (millió) „a”-ra fennáll az előbbi összefüggés, akkor azt mondhatjuk, hogy „p” valószínűleg prímszám.

3.7. Hogyan számoljunk nagy számokkal?

Jogosan merülhet fel az Olvasóban a kérdés: hogyan dolgozzak olyan nagy számokkal, melyeket fel sem tudok olvasni? Szerencsére a véges matematikai mezők felett végzett hatványozás (akármekkora legyen is a kitevő vagy az eredmény) nem okoz komoly problémát. Nézzünk egy példát! Számítsuk ki az alábbi kifejezés értékét!

$$5^{1030} \bmod 257$$

Tudjuk, hogy ez a kifejezés azonos a

$$5^{1024 + 4 + 2} \bmod 257$$

kifejezéssel. Nem tettünk mást, mint 2 egész kitevőjű hatványaira bontottuk a kitevőt ($1024 = 2^{10}$, $4 = 2^2$, $2 = 2^1$). Bontsuk tényezőkre a hatványozást:

$$(5^{1024} * 5^4 * 5^2) \bmod 257$$

Ezután számítsuk ki az egyes tényezők értékét. Tudjuk, hogy:

$5^1 \bmod 257 = 5 \bmod 257 = 5$
$5^2 \bmod 257 = 5^{1 + 1} \bmod 257 = 5 * 5 \bmod 257 = 25$
$5^4 \bmod 257 = 5^{2 + 2} \bmod 257 = (5^2 * 5^{2)} \bmod 257 = 25 * 25 \bmod 257 = 111$
$5^8 \bmod 257 = 5^{4 + 4} \bmod 257 = (5^4 * 5^4) \bmod 257 = 111 * 111 \bmod 257 = 242$
$5^{16} \bmod 257 = 5^{8 + 8} \bmod 257 = (5^8 * 5^8) \bmod 257 = 242 * 242 \bmod 257 = 225$
$5^{32} \bmod 257 = 5^{16 + 16} \bmod 257 = (5^{16} * 5^{16}) \bmod 257 = 225 * 225 \bmod 257 = 253$
$5^{64} \bmod 257 = 5^{32 + 32} \bmod 257 = (5^{32} * 5^{32}) \bmod 257 = 253 * 253 \bmod 257 = 16$
$5^{128} \bmod 257 = 5^{64 + 64} \bmod 257 = (5^{64} * 5^{64}) \bmod 257 = 16 * 16 \bmod 257 = 256$
$5^{256} \bmod 257 = 5^{128 + 128} \bmod 257 = (5^{128} * 5^{128}) \bmod 257 = 256 * 256 \bmod 257 = 1$
$5^{512} \bmod 257 = 5^{256 + 256} \bmod 257 = (5^{256} * 5^{256}) \bmod 257 = 1 * 1 \bmod 257 = 1$
$5^{1024} \bmod 257 = 5^{512 + 512} \bmod 257 = (5^{512} * 5^{512}) \bmod 257 = 1 * 1 \bmod 257 = 1$

Ebből:

$$(5^{1024} * 5^4 * 5^2) \bmod 257 = (1 * 111 * 25) \bmod 257 = 2775 \bmod 257 = 205$$

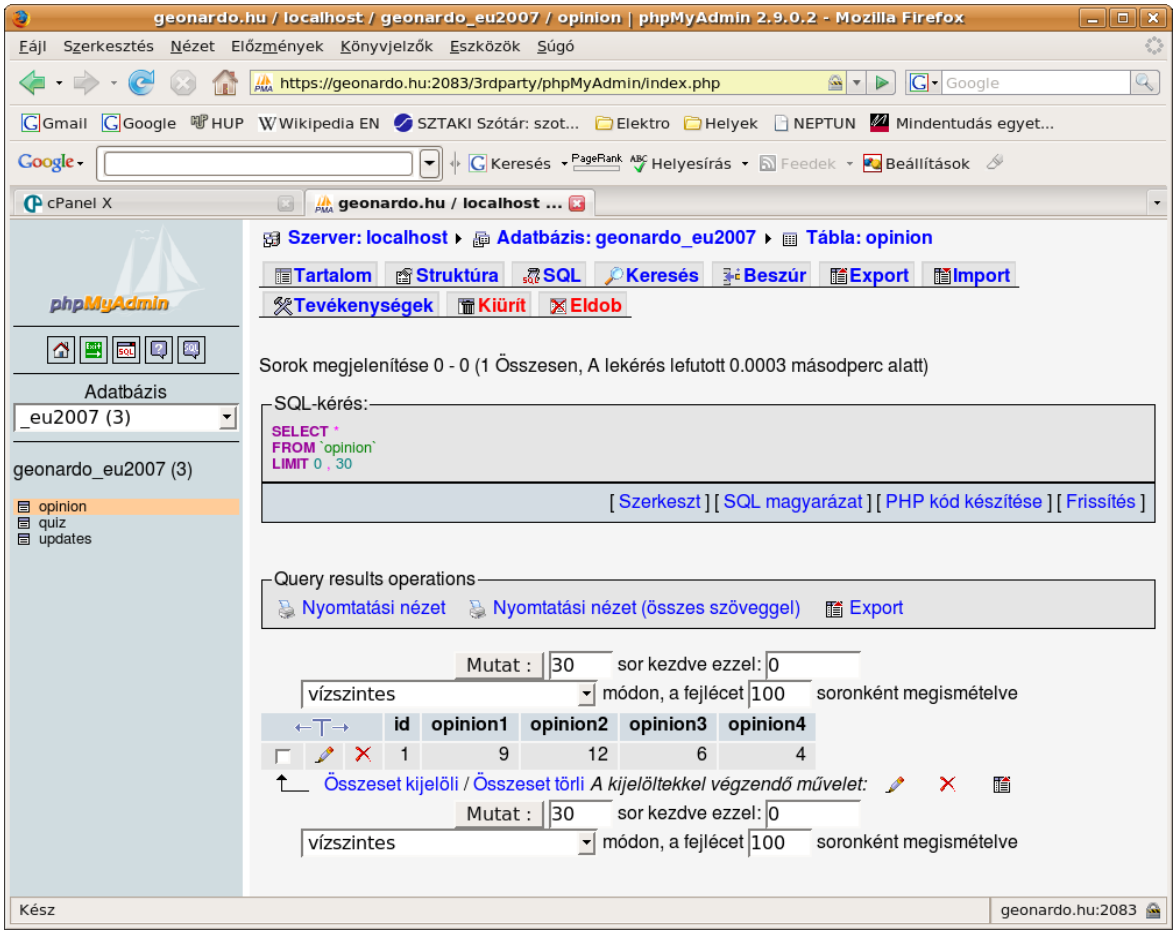
Ezzel a módszerrel dolgoznak a számítógépek is, hiszen az előbbi hatványozást mi sem 1030 lépésben végeztük el, hanem csak 13-ban. Ez nagymértékben meggyorsítja a számításokat.

4. Gyakorlati megvalósítások

Ebben a részben néhány példát látunk arra, hogy munkánk, tanulmányaink során mikor találkozhatunk titkosítással. A leggyakoribb eset az, mikor egy webszerver és a böngészőnk között épül fel egy titkosított csatorna: vásárlás, banki ügyintézés, távoli adminisztráció esetén megengedhetetlen, hogy valaki hozzáférjen személyes vagy céges adatainkhoz.

A titkosított kapcsolat létrejöttekor a böngészőnk és a webszerver valamelyik nyílt kulcsú titkosító algoritmus (tipikusan az RSA) segítségével generál egy megfelelő méretű kulcsot. A tényleges kommunikációt szimmetrikus algoritmussal titkosítják, mert ezek sokkal gyorsabbak, mint a nyílt kulcsú társaik.

A következő képernyőképen éppen azt láthatjuk, mikor a felhasználó egy távoli kiszolgálón található adatbázist adminisztrál.



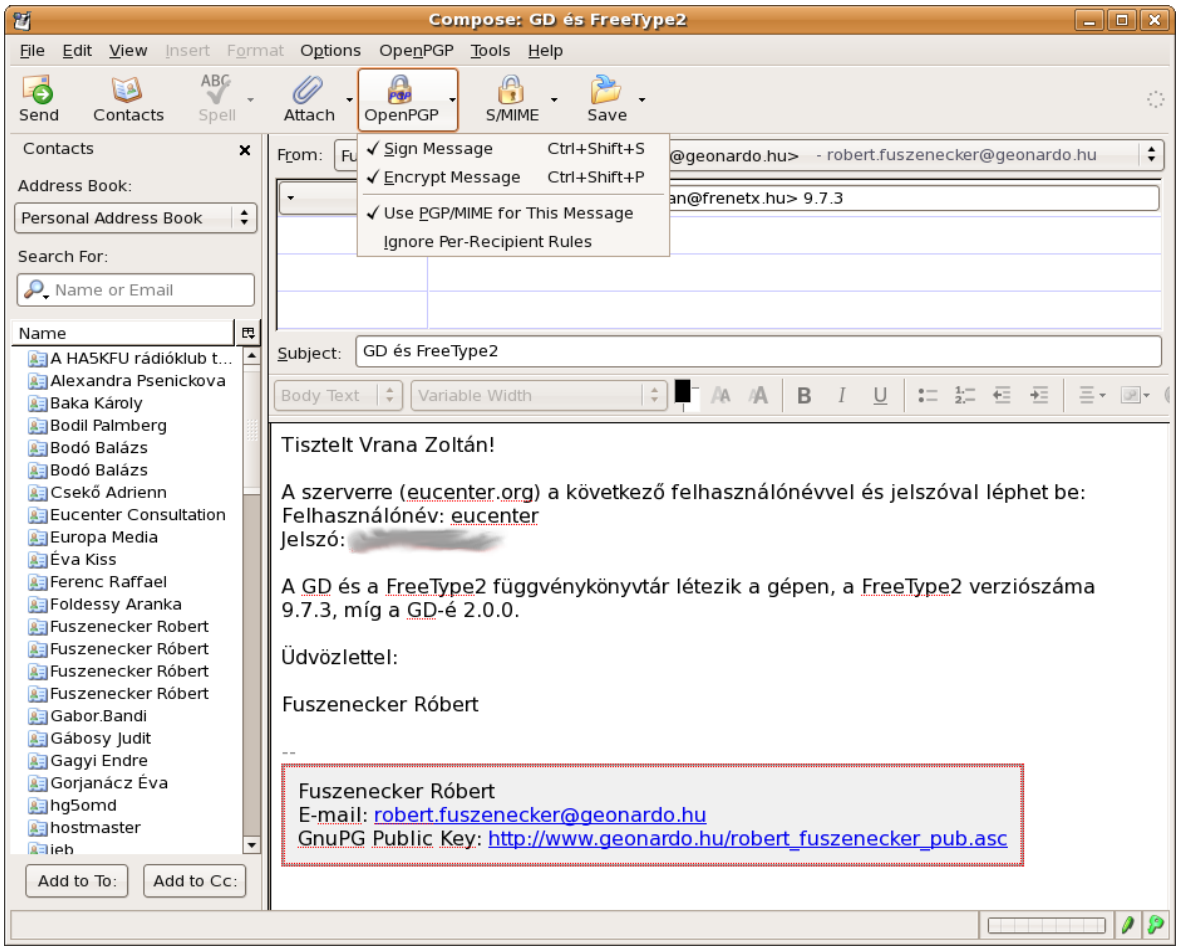
Figyeljük meg a képernyő jobb alsó sarkában látható lakatot, ami azt jelzi, hogy a böngésző és a szerver közötti információátvitel titkosított. A lakatra kattintva megtudhatjuk a titkosítás részleteit.

Levélben is küldhetünk olyan adatokat, melyeket nem lenne szerencsés a levél címzettjén kívül mással is „megosztani”. A levelező rendszerek önmagukban nem teszik lehetővé a feladó egyértelmű azonosítását, de ha a levelezőprogramunk rendelkezik digitális aláíró funkcióval, akkor mintegy kiegészítésként hozzáfűzhetjük aláírásunkat a levélhez. Az aláírást nem támogató, alacsonyabb rendű

levelezőprogramok mindössze egy csatolt állományt látnak, amivel semmit sem tudnak kezdeni (azt sem ismerik fel, hogy az egy aláírás).

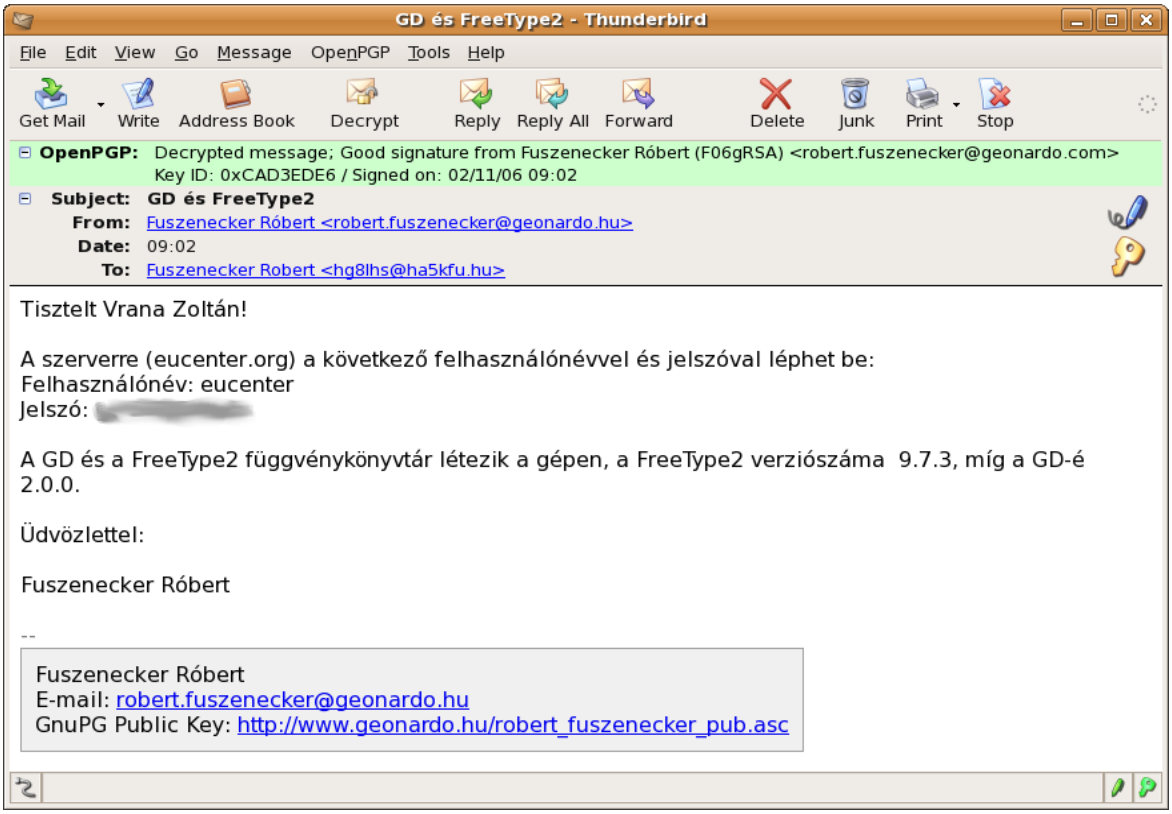
A képernyőképen az látható, hogy levelezőprogram OpenPGP menüjének aláírás (*Sign Message*) funkciója mellett a titkosítás (*Encrypt Message*) is be van kapcsolva, így a levél szövegét csak a címzett tudja olvasni. Természetesen a titkosítást és az aláírást külön-külön is alkalmazhatjuk, célszerű az aláírást minden levélhez csatolni, a titkosítás engedélyezéséről pedig ráérünk akkor gondoskodni, amikor olyan adatokat közlünk a címzettel, ami kizárólag rá tartozik.

Ebben az esetben RSA titkosítást és aláírást alkalmaz a program; az aktuális algoritmus mindig a kulcs típusától függ.



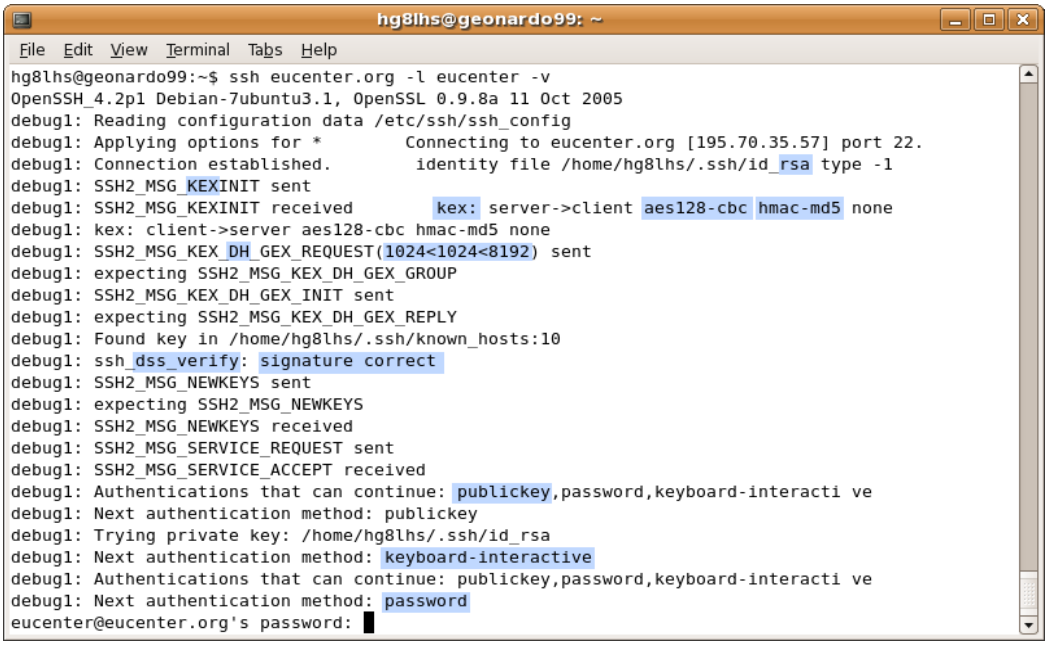
A következő képernyőkép azt mutatja, hogy mit láthat a címzett, ha a levelezőprogramja képes értelmezni a titkosított és aláírt leveleket. A levél fejlécében több érdekes dologra is figyelmesek lehetünk: először is egy zöld színű sávban (remélhetőleg a Kedves Olvasó a dolgozat egy színes példányát tartja a kezében) megjelenik a feladó neve, címe, és ami a legfontosabb: a helyes digitális aláírást jelző azonosító szöveg.

A másik dolog, amit még észrevehetünk, hogy a jobb oldalon megjelenik két ikon. A felső arról tanúskodik, hogy a feladó aláírta a levelet, míg az alsó ikon a titkosítás alkalmazására utal. A levél feladója arról is gondoskodott, hogy (az ehhez az email-címéhez tartozó) nyilvános kulcsa bárki számára elérhető legyen.



Eddig arról beszéltünk, hogy mikor használunk RSA titkosítást és aláírást. Lássunk most egy példát a Diffie-Hellman kulcscsere algoritmusra is. Ezt a nyílt kulcsú titkosító módszert jellemzően akkor használjuk, amikor egy távoli számítógép parancssoros környezetét szeretnénk elérni. Ezt az *ssh* nevű segédprogrammal tehetjük meg a legegyszerűbben.

A következő képernyőkép egy ilyen belépést örökít meg. Az kulcscsere szempontjából érdekes adatok a kék téglalapokban láthatók.



Első lépésként a mindkét kommunikáló fél előkészíti a kulcscserét, erre utal a kép felső részén olvasható KEXINIT üzenet. A KEX a *key exchange* (kulcscsere) szavak rövidítése. Ezután megegyeznek a tényleges titkosítást végző szimmetrikus algoritmusban. Jelen példában az AES-re esett a választás, a 128 azt jelenti, hogy 128 bit hosszúságú titkosító kulcsot fognak használni.

Az aláírás elkészítéséhez az MD5 kivonatkészítő függvényt alkalmazzák.

A következő lépésben a kliens közli a szerverrel, hogy a Diffie-Hellman nyilvános kulcs legalább 1024, de legfeljebb 8192 bit hosszúságú lehet. Ezután megtörténik a kulcscsere, majd mindkét fél ellenőrzi a Diffie-Hellman kulcsok digitális aláírását. A képen jól látható, hogy a kliens program megfelelőnek találta az aláírást, és elfogadta a kulcsot. A „középen lévő ember” támadása ily módon kizárható.

Legutolsó lépésként megtörténik a felhasználó azonosítása, és a jelszó (password) beolvasása a billentyűzetről. Ha a szerver sikeresen azonosítja felhasználót, akkor megjelenik a parancsok begépelésére szolgáló parancssor. Ezt az állapotot a képernyőkép nem sajnos mutatja.

Összefoglalás

Ha az Olvasó elejétől a végéig elolvasta és megértette ezt a dolgozatot, akkor bizonyára egyetért velem abban, hogy a nyílt kulcsú titkosítás, és általában véve a kriptográfia létrejötte majdnem akkora hatással volt az emberi civilizáció fejlődésére, mint a kerék feltalálása. A titkosítás jelenléte nem annyira szembetűnő mindennapi életünkben, mint a keréké, vagy a villamos áramé, hiszen egy algoritmus kevésbé kézzel fogható, mint egy fizikailag megjelenő, érzékelhető, megtapasztalható jelenség. Ennek ellenére óriási hatást fejt ki már most is, és az oly gyakran emlegetett, küszöbön álló információs társadalom elképzelhetetlen a biztonságos kommunikáció nélkül.

Tudjuk már, hogy miért nem használhatjuk a kommunikációs csatornákat fenntartás nélkül: láttuk, hogy egy olyan csatorna, amihez a kommunikáló feleken kívül bárki hozzáférhet, nem tekinthető biztonságosnak, ezért nem mellőzhetjük az adataink elrejtését végző titkosító algoritmusok használatát. Felvetettük azt, hogy miért nem elegendő olyan módszerek alkalmazása, mikor a csatorna mindkét oldalán ugyanazt a kulcsot használjuk – ezt neveztük szimmetrikus titkosításnak –, és megválaszoltuk azt a kérdést is, hogy mi ennek a problémának a legkézenfekvőbb megoldása.

A nyílt kulcsú titkosító módszerek közül megismerkedtünk a legrégibb, és ma is nagy jelentőséggel bíró algoritmusokkal: elsőként a Diffie-Hellman kulcscserét vizsgáltuk meg, majd RSA titkosítással és aláírással folytattuk a sort.

Bebizonyítottuk azt is, hogy mind a Diffie-Hellman kulcscsere, mind az RSA titkosítás kellően biztonságos addig, amíg a matematikusok nem találnak megfelelő megoldást a diszkrét logaritmus kiszámítására és az egész számok felbontásának problémájára. Nem állíthatjuk biztosan, hogy ez az idő sosem jön el – talán már valamelyik katonai szervezet már rendelkezik is a megfelelő algoritmussal –, de a mai napig még senki sem bizonyította, hogy képes lenne akármelyik „jól beállított” nyílt kulcsú rendszert megtörni.

Ha az Tisztelt Olvasó kedvet kapott ahhoz, hogy a gyakorlatban is próbára tegye egyik vagy másik titkosító módszert, bátran merítsen ötletet a dolgozat negyedik fejezetéből. A megfelelő programok kiválasztásában segítséget nyújthat a felhasznált szoftvereket felsoroló részt, ami a dokumentum utolsó néhány lapjának egyikén kapott helyet.

Ez a dolgozat nem referenciamű, mégis igyekeztem közérthetően, pontosan, és a lényeghez ragaszkodva átadni azt a tudást, amit az elmúlt fél évben saját erőfeszítésem révén szereztem.

Végezetül csak remélni tudom, hogy az Olvasó hasznosnak találta ezen írás elolvasását, és a benne felvetett gondolatok megértését.

Mellékletek

Fermat kis tétele

Tétel:

Ha „p” prímszám, és nem osztója egy „a” egésznek, akkor a^{p-1}-1 osztható „p”-vel.

Bizonyítás:

A bizonyítás abból fog állni, hogy megvizsgáljuk sorra az a, 2a, ..., (p–1)a számok maradékát „p”-vel való osztásnál:

Legyen

$$ka = pq_k + r_k, \qquad \text{ahol } 0 <= r_k < p \qquad \text{és } (k = 1, 2, ..., p–1).$$

Egyik „r_k” sem lehet nulla, mert „k” nem osztható „p”-vel, és így relatív prím hozzá, „a”-ra feltétel szerint ugyanaz áll, s így „ka” is relatív prím „p”-hez.

Az összes maradékok különböznek egymástól. Legyen ugyanis $k_1 > k_2$ és vonjuk ki a „k_1”-edik egyenletből a „k_2”-ediket:

$$(k_1 – k_2)a = p(q_{k1} – q_{k2}) + (r_{k1} – r_{k2}).$$

Itt a (k_1–k_2) szám „p”-nél kisebb pozitív egész, és így a bal oldal az előbbiekhez hasonló okoskodás szerint nem osztható „p”-vel, tehát r_{k1}–r_{k2} sem lehet 0.

Az r_1, r_2, ..., r_{p-1} számok tehát az 1, 2, ..., p–1 számok közül kerülnek ki, és nincs köztük két egyenlő. Ebből viszont az következik, hogy valamilyen sorrendben az 1, 2, ..., p–1 számok mindegyike előfordul közöttük. Szorozzuk össze most az egyenlőségeinket és vizsgáljuk meg a jobb oldalt. Itt, ha tagonként beszorzunk, az $r_1 r_2 ... r_{p-1}$ tagon kívül csupa „p”-vel osztható tag keletkezik, e tag pedig utolsó megállapításunk szerint (p–1)!-sal egyenlő. Ilyen alakú egyenlőséget kapunk tehát:

$$(p–1)!\ a^{p-1} = pQ + (p–1)!$$

ahol „Q” valamilyen egész szám. Ezt úgy is mondhatjuk, hogy

$$(p–1)!\ (a^{p-1}–1)$$

osztható „p”-vel. (p–1)! azonban relatív prím „p”-hez, mert mindegyik tényezője relatív prím hozzá. Így a^{p-1}–1 osztható „p”-vel, és ezt akartuk bizonyítani.

A kínai maradék-tétel

Tétel:

Ha az m_1 ,m_2, ..., m_k számok páronként relatív prímek és a_1, a_2, ..., a_k tetszőleges egészek, akkor az

$$x = a_1 \ mod(m_1)$$

$x = a_2 \ mod(m_2)$
.....
.....
$x = a_k \ mod(m_k)$

szimultán kongruencia rendszernek pontosan egy megoldása van moduló „N” (ahol $N = m_1, m_2, ..., m_k$).

Bizonyítás:

Legyen $M_i = \prod_{j \neq i} m_j$, ekkor igaz a következő:

$$\left((M_i, m_i) = 1\right), \check{Z}s \ \ \left(\exists\left(M_i^{-1}\right) \in Z\ \right), melyre\left(M_i M_i^{-1} \equiv 1 \bmod(m_i)\right)$$

Legyen $x_0 = \sum_{i=1}^{i=k} a_i M_i M_i^{-1}$, behelyettesítve a „j.” kongruenciába valóban $x_0 = a_j \bmod(m_j)$-et kapunk, mivel a $\sum_{i=1}^{i=k} a_i M_i M_i^{-1}$ összeg „j.” tagja pontosan „a_j” és az összes többi tag 0 lesz $\bmod(m_j)$. A megoldás mod(N)-re vonatkozó egyértelműségét indirekt bizonyítsuk. Tételezzük fel, hogy van egy másik „x_0*” megoldás, mely nem kongruens mod(N) „x_0”-lal. Az

$x_0 = a_j \ mod(m_j)$ és az
$x_0^* = a_j \ mod(m_j)$

kongruenciákat kivonva egymásból $x_0^* - x_0 = 0 \bmod(m_j)$-ből $(m_j)|(x_0^* - x_0)$ adódik bármely j = 0, 1, ..., k-ra, s mivel az „m_j” páronként relatív prímek voltak $N|(x_0^* - x_0)$, ami ellentmond feltevésünknek.

Felhasznált szoftverek

- Ubuntu Linux 6.06 (Linux kernel 2.6.15)
- OpenOffice.org 2.0.4
- The GIMP 2.2.11
- Mozilla Firefox 1.5.0.7
- Mozilla Thunderbird 1.5.0.2
- Python 2.4.3
- OpenSSH 4.2p1

Felhasznált irodalom

- Wikipedia, The Free Encyclopedia: www.wikipedia.org
- Erdős Pál - Surányi János: Válogatott fejezetek a számelméletből, Tankönyvkiadó Vállalat, Budapest, 1960