Eötvös Loránd Tudományegyetem

Informatikai Kar

A SecMS üzenetküldő fizetési rendszere

Témavezető: Nagy Dániel

ELTECrypt kutatócsoport

vezető szakértő

Készítette: Riskó Gergely

nappali tagozat

programtervező matematikus szak

Budapest, 2008.

Ez itt a témabejelentő helye.

Tartalomjegyzék

I. rész

Bevezetés

Dolgozatomban először rövid áttekintést adok a biztonságos üzenőrendszerekről. Olyan rendszereket vizsgálok, amelyben csak az üzenet küldője és címzettje ismerheti meg az üzenet tartalmát (konfidencialitás), illetve az is biztosított, hogy az üzenet címzettje biztos a feladó személyében (hitelesség) és abban, hogy az üzenetet az eredeti formájában, módosítások nélkül kapja meg (integritás).

Fontos, hogy a követelmények teljesülését maga a protokoll kriptográfiai eszközökkel biztosítsa, az üzenet továbbításában szerepet játszó harmadik felek (internet szolgáltatók, szerverek és azok rendszergazdái) ne sérthessék ezeket. Nagyon gyakori félmegoldás, hogy a levelezőkliensek és kiszolgálók közti csatornákat rejtjelezik, nem törődve a szerveren megvalósítható támadásokkal.

A már létező megoldások mellett bemutatok egy újat, amit az ELTECrypt kutatócsoport dolgozott ki és szakdolgozatként[36] egy korai verzióját már megvédtem. Ezzel új eredményeket értünk el: látni fogjuk, hogy a mi rendszerünk műveletigénye jóval kisebb, mint a versenytársaké, így pl. mobil környezetben is hatékonyan használható. Ezt demonstrálandó, a szakdolgozatom óta a kutatócsoportban a mobiltelefonokon (Java környezetben) futó kliensalkalmazást is elkészítettem és továbbfejlesztettük az üzenőrendszer szerverének szolgáltatásait is. Akár csak a szakdolgozatom tesztelésekor, ezeket az új komponenseket is bemutattuk egy szemináriumon, ahol a programmal először találkozó felhasználókat kértük meg, hogy próbálják meg használni a kezükbe adott mobiltelefonokon a

SecMS-t. A SecMS így önmagában is fontos eredmény, azonban igazi szerepét egy fizetőrendszerrel együtt tudja betölteni.

Ezért be fogom mutatni, hogy hogyan működik egy régóta ismert, David Chaum által kidolgozott, vak aláírásokon[7] alapuló fizetőrendszer[8]. Ez a javaslat 20 éves és azóta sem terjedt el. Ennek az okait vizsgálja Nagy[31] és egyben egy új javaslatot is ad a digitális készpénz megvalósítására, amit ePointnak hív. Az ePoint újdonsága a megbízható harmadik fél (kibocsátó) ellenőrizhetőségében, a tranzakciós díjaktól való mentességében rejlik. Megismétlem a Nagy által felhozott érveket és megvizsgálom az ePoint működését, megmutatva, hogy a Chaum javaslata óta kialakult új környezetben (sokkal olcsóbb és gyorsabb, mindenütt jelenlévő adathálózatok, kis kapacitású kliensek), az ePoint valóban életképesebb. Schuller [46] diplomamunkaként kidolgozott egy HTTP alapú protokollt az ePointok továbbítására, én ezt felhasználva megmutatom, hogy hogyan lehetne az ePoint rendszert integrálni a SecMS-sel.

A fizetőrendszer és a SecMS integrálásával a külön-külön is érdekes alkalmazásokból egy sokkal hasznosabbat kapunk, hiszen minden pénzügyi tranzakciónk közben szükség van a fenti biztonsági követelményeket teljesítő üzenetváltásra is. Fordított irányban is szükséges az integráció: a SecMS fizetőrendszer nélküli elterjedése esetén az emailhez hasonlóan nagy problémákat okozna a levélszemét kezelése. Az integráció után azonban a probléma egyszerűen orvosolható, egy olyan rendszert kell felépíteni, amiben minden levél mellé pénzt kell csatolni, így biztosítva a fogadó oldalon a figyelem felkeltését. Természetesen amennyiben a levél legitim, arra válaszolva a pénzösszeg visszajuttatható a feladónak.

Habár az azonnali felhasználása ezeknek az új technológiáknak a spam elleni védekezés, alkalmazási területe ennél sokkal szélesebb: zárásul leírom mikrofizetésekhez kapcsolódó, ma még megoldatlan problémák (pl. tartalom- és hírszolgáltatások, telekommunikációs szolgáltatások fizetésének) lehetséges megoldásait.

Egy korábbi fizikai megvalósítás

Bruce Schneier egyik internetes naplóbejegyzésében[43] megemlíti, hogy a vázolt megoldást 1933-ban már megvalósították. Igaz, akkor nem a kéretlen elektronikus levelek, hanem a jelentéktelen ügyekkel házalók ellen kívántak védekezni. A látogatónak be kellett helyeznie egy tízcentes érmét, ahhoz, hogy a csengő megszólaljon. Amennyiben a házigazda számára a vendég kívánatos volt, akkor ezt az érmét visszaadta.

DIME PUT IN SLOT RINGS DOORBELL

To SAVE a busy housewife from frequent annoyance by unwelcome callers, a doorbell that works only upon the insertion of a dime is soon to be marketed. The coin slides into an inside receptacle, where it closes an electric contact that permits the bell to be rung. If the caller proves to be a friend, the dime is returned as the guest enters; if the visitor is a stranger or one to whom entrance is refused, the money is retained. Dimes kept by the device provide a fund for charities.

Only after a coin has been deposited in the slot, as shown at the right, does the doorbell ring. The housewife can thus be reasonably sure the caller is one having legitimate business with her. This protects her against the intrusion of annoying or casual callers

THIS BELL WILL NOT RING WITHOUT INSERTING DIME

COIN WILL NOT BE REFUNDED TO CANVASSERS OR PEDDLERS

The interior of the coin-in-slot doorbell is shown above. The coin slides into receptacle inside the door where it closes electric contact and rings bell. Housewife can return the money

Schneier megjegyzi, hogy a levélszemét esetében két szempontból még kedvezőbb is a helyzet, mint a csöngő esetében. Először is, a kéretlen levelek száma a valós levelek számához képest sokkal nagyobb, mint az indokolatlanul becsöngetők száma a valódi látogatók számához képest. A mértéktartóbb becslések szerint[59] az összes levél 80-85 százaléka kéretlen, de a bátrabb becslések 95 százalékot említenek. Ezért a rendszert megérheti felépíteni és működtetni. Másrészt, ha az informatikai megoldást szabványosítják és minden elterjedt levelezőprogramban megvalósítják, akkor az mindenki számára elérhető. Ellentétben a fizikai megvalósítással, ahol elképzelhető, hogy valakinél nincs megfelelő pénzérme.

Aggodalmát fejti ki ugyanakkor amiatt, hogy ahogyan csöngetés helyett bekopoghatunk az ajtón vagy kiabálhatunk, az elektronikus megoldásban is meg fognak próbálni kiskapukat keresni. Sőt, azzal, hogy egy fizetőrendszer áll a szűrőrendszer mögött, nyilvánvaló, hogy a leveleket küldő és fogadó számítógépek feltöréséhez közvetlen anyagi érdek is fog fűződni. Hiszen ha rá tudunk venni jogosulatlanul egy kliensprogramot, hogy a saját címünkre minél több levelet továbbítson, akkor az azokhoz csatolt összegeket elvettük a gazdájától.

Schneier úgy gondolja, hogy emiatt, egy biztonsági szempontból a mai asztali számítógépeknél sokkal megbízhatóbb platform nélkül az ötletet el is kell vetni. Egyetértünk vele, pontosan ezért készítettük el a mobiltelefonokon futó implementációját a SecMS-nek és ezért tervezzük úgy, hogy a mindennapi felhasználók esetében a konkrét pénzügyi tranzakciók csak a mobiltelefon segítségével történhessenek majd. A mobiltelefonokra írható programok egymástól jól szeparáltak, biztosított a mobiltelefon és a programok egymástól való védelme, ugyanakkor megoldott a kommunikáció (bluetooth technológiával, infravörös kapcsolaton vagy akár interneten) az asztali- és szerverszámítógépekkel.

II. rész

Létező üzenetküldő- és fizetőrendszerek áttekintése

Ebben a részben először a biztonsági építőelemek (aszimmetrikus kriptográfia [29]; RSA rejtjelezés és digitális aláírás [37]; DSA digitális aláírás [32]; ElGamal rejtjelezés és digitális aláírás [20]; Diffie-Hellman kulcsmegegyezés [12]; RC4 folyamtitkosító [57]; különböző hash függvények) célját és működését mutatom be.

Ezek felhasználásával áttekintem a már kidolgozott üzenő- és fizetőrendszerek működését és specifikációik fontosabb részleteit.

1. fejezet

Biztonsági építőelemek

Biztonsággal foglalkozó protokolljainkat és programjainkat olyan építőkockákból szeretnénk felépíteni, amik önmagukban is biztonságosak. Azt várjuk ettől, hogy a kész rendszerben is kisebb eséllyel marad hiányosság. Azt, hogy mit érthetünk egy építőelem biztonságosságán, tárgyalja [21] és bevezeti (a közgazdaságtanból kölcsönözve) a Pareto-biztonságosság, illetve Pareto-teljesség fogalmát.

Akkor mond egy eljárást *Pareto-biztonságosnak*, ha a felhasználásakor tett feltételek teljesülése esetén, amellett, hogy minden felmerülő, praktikus támadásnak ellenáll, már nincsen olyan módosítás, ami a biztonságot jelentősen javítja egy területen, anélkül, hogy más területen érezhetően csökkentené.

Egy komponens *Pareto-teljes*, ha minden ésszerű feltételezés és biztonsági rendszerben való felhasználás mellett Pareto-biztonságos. Ez a fogalom abban több tehát, hogy nem szükséges pontosan specifikálnunk a támadóval szembeni feltételezéseinket, ezektől függetlenül használhatjuk a Pareto-teljes eljárásokat.

Ezek a definíciók elég szubjektívek és egy-egy új eredmény hatására korábban Pareto-teljesnek hitt eljárások válhatnak feltörhetővé. A diplomamunkámban bemutatott építőelemek többé-kevésbé Pareto-teljesnek tekinthetőek, de a Pareto-biztonságosságot az ismertetett felhasználások során mindig megköveteljük.

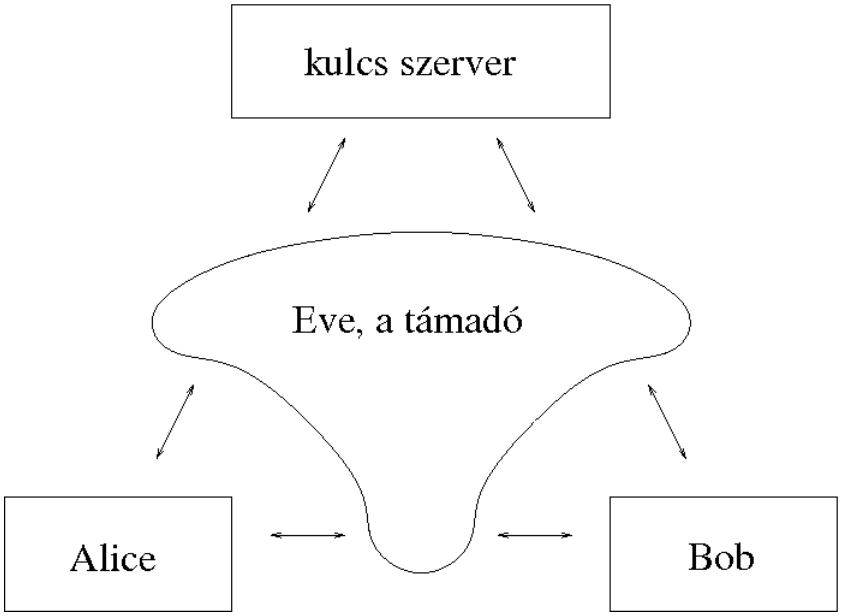
1.1. Aszimmetrikus kriptográfia

Ezekkel az eljárásokkal el lehet kerülni, hogy a rejtjelezést vagy hitelesítést használni kívánó feleknek előzetesen valamilyen biztonságos csatornán meg kelljen egyezniük közös titkokban, eljárásokban.

Ezekben az algoritmusokban az a közös, hogy minden résztvevőhöz tartozik egy nyilvános és egy titkos művelet. A rejtjelezést a nyilvános művelettel hajthatja végre bárki, amit megfejteni csak a titkos művelettel lehet; a digitális aláírást tetszőleges bájt sorozaton a titkos művelettel hajthatja végre a kulcs tulajdonosa, amit bárki leellenőrizhet a titkos kulcs nyilvános párjával. A digitális aláíráshoz és a rejtjelezéshez tartozó nyilvános és titkos művelet nem feltétlenül egyezik. Amennyiben igen (az RSA pl. ilyen), akkor nem tanácsos ugyanazt a kulcsot mindkét célra használni. Azt sejtjük minden ilyen eljárás esetében, hogy ahhoz, hogy a nyilvános műveletből előállítsuk a titkosat, valamilyen olyan problémát kell megoldanunk, aminek a megoldása nagyon költséges, így nem kivitelezhető. Ilyen problémák pl. a faktorizálás (az RSA épül erre), illetve egy csoport elemének egy generátorra nézve diszkrét logaritmusának a megállapítása (DSA).

Elfogadva azt a sejtést, hogy ezeknek a problémáknak a megoldásával nem fogják a protokollunkat támadni, meg kell még oldani a nyilvános műveletek (kulcsok) minél megbízhatóbb terjesztését. Különben egy résztvevőt megtámadhatnak az ún. man-in-the-middle támadással (1.1. ábra), aminek a lényege az, hogy elhitetik vele, hogy a partnerének más a nyilvános kulcsa, mint valójában. A kommunikációs csatornába beékelődve mindkét irányba újrakódolják az üzeneteket úgy, hogy a két fél számára ez a kezdeti kulcs letöltés után már nem észrevehető.

Ez ellen a támadás ellen két módon is lehet védekezni. Először: megegyeztek hash függvényekre épülő algoritmusokban, amik a nyilvános kulcsokhoz ujjlenyomatokat tudnak rendelni, amik már kellően rövidek ahhoz, hogy valamilyen megbízható csatornán (telefonon, faxon, postai levélben) közöljék egymással a kom-



1.1. ábra. Man-in-the-middle támadás

munikáló felek. Letöltve a nyilvános kulcsot előállítják az algoritmussal az ujjlenyomatot és mielőtt üzeneteket váltanának, leellenőrzik, hogy pontosan ugyanazt az értéket kapták, mint amit a megbízható csatornán megbeszéltek. Másodszor: a nyilvános kulcsokat digitális aláírással látják el mások, akikkel a megbízható kulcscsere már megtörtént. Az így keletkező kulcshitelesítést közzéteszik a kulccsal együtt és így a kulcs automatikusan ellenőrizhető, a felhasználó és megbízható csatorna igénybevétele nélkül.

Az, hogy kit hatalmazunk fel kulcshitelesítésre megszervezhető egy hierarchikus rendszerben (X.509) és ekkor arról kell csak gondoskodnunk, hogy a hierarchia fejének nyilvános kulcsa mindenkihez biztonságban eljusson. Így működik pl. a web biztonságát nyújtó HTTPS. Vagy felhatalmazhatunk mindenkit kulcshitelesítésre (OpenPGP[5]) és ekkor egy irányított gráffal lehet jellemezni a kulcsok közötti viszonyokat. Az utóbbi rendszerben egy sokkal pontosabban mérő, szinte folytonos skálán tudjuk osztályozni egy kulcs megbízhatóságát, attól füg-

gően, hogy a gráfban a mi kulcsunktól az ellenőrizendő kulcsig milyen utak vezetnek. Ez a jobban elterjedt megoldás az email esetében, annak ellenére, hogy az S/MIME nevű, X.509 alapú megoldás minden levelezőkliensben alapértelmezés szerint megtalálható. Azonban az OpenPGP kulcskezelése olcsó és nem kötelező, míg az X.509 használata esetén a tanúsítás kötelező és a hierarchikus szervezet fenntartása miatt drága is.

A nyilvános kulcsokra épülő rejtjelezés ún. kulcsmegegyezéssel is megoldható. Ha feltételezzük, hogy az üzenetet küldő és fogadó fél egyaránt rendelkezik regisztrált kulccsal, akkor lehetséges olyan művelet kidolgozása, ami ugyanazt a titkot állítja elő, ha az egyik fél titkos és a másik fél nyilvános kulcsára alkalmazzuk, mintha a másik fél titkos és az egyik fél nyilvános kulcsára alkalmaznánk. Így ezt a titkot rögtön használhatják egy szimmetrikusan rejtjelező algoritmus kulcsaként, a titok előállítása harmadik fél számára rendkívül drága.

A nyilvános kulcsú kriptográfia és a nem biztonságos csatorna fölötti kulcsmegegyezés alapötletét Merkle[29] vetette fel először, azonban a gyakorlatban használható megoldást nem adott. A kulcsmegegyezésre az első, azóta is használt megoldást Diffie és Hellman dolgozta ki[12], ezt Diffie–Hellman kulcsmegegyezésnek hívják. A nyilvános kulcsú kriptográfiát pedig Rivest, Shamir és Adelman valósította meg elsőként.

1.1.1. Vak digitális aláírás

A gyakorlatban, mivel az aszimmetrikus műveletek rendkívül lassúak, a digitális aláírás készítésekor az aláíró fél nem magát az üzenetet, csupán annak egy hash értékét írja alá. A különböző hash függvényekről még lesz szó a 1.8. szakaszban. Így az aláírás gyorsan elkészíthető, rövid és a digitális aláírásból (a tanúsított hash értékből) nem lehet következtetni az üzenet tartalmára.

Azonban ez még nem jelenti azt, hogy amennyiben az aláírt adat tulajdonosa

és az aláíró személy különbözik, akkor igaz lenne, hogy az aláíró semmit nem tud az üzenetről. Tudja a hash értékét ! Tehát ha az aláíró megjegyzi minden aláírt hash értékhez az aláírást kérvényező azonosítóját az aláírás pillanatában, akkor ha később találkozik egy általa aláírt üzenettel, össze tudja kapcsolni azt az aláírást kérő személlyel.

Az olyan digitális aláírási sémákat, ahol az aláíró entitás úgy tud aláírásokat kibocsátani, hogy az aláírt értékekről semmilyen információ nem jut a birtokába, *vak digitális aláírásoknak* nevezzük.

A vak digitális aláírás fizikai analógiája a szárazbélyegző. Szárazbélyegzővel (egy átlátszatlan boríték segítségével) úgy tudunk iratokat hitelesíteni, hogy az irat tartalmáról semmilyen információt nem kell felfednie a hitelesítést kérő személynek.

A vak aláírások felhasználására az első javaslat[7] az elektronikus készpénz. Vak aláírások segítségével úgy tud egy bank anoniman használható, digitális pénzt kibocsátani, hogy vakon aláírja az ügyfél által elkészített egységnyi értékű bankjegyet. Ezzel egyidőben levonja az egységnyi összeget a számlatulajdonos számlájáról. Az ügyfél a keletkezett digitális bankót odaadhatja bárkinek, akinek fizetni akar, aki a banknál azt beválthatja. Mivel a bankjegy alá van írva, a bank tudja, hogy fizettek érte. Egy adatbázissal pedig garantálható, hogy minden bankjegyet csak egyszer használjanak fel. A beváltó és fizető fél ugyanakkor az aláírás vak tulajdonsága miatt biztos lehet benne, hogy a bank nem tudja őket összekapcsolni, legalábbis forgalomelemzés nélkül.

Elektronikus szavazások esetén is megtartható a vak aláírások segítségével az anonimitás. A szavazásra jogosult authentikáció után aláíratja vakon a szavazatát a jegyzővel, aki kihúzza cserébe őt a szavazásra jogosultak listájáról. Ezután a szavazatát beküldheti (valamilyen anonim csatornán) a szavazatszámláló bizottságnak, ott le lehet majd ellenőrizni, hogy ez egy számlálandó szavazat (hiszen

alá van írva). Ugyanakkor a szavazat tartalmát nem lehet a szavazó személyéhez kötni. Természetesen ezzel az ötlettel csak a vak aláírások fontosságát akartam bemutatni, maga a rendszer a gyakorlatban használhatatlan, mivel sok csalási lehetőséget ad a bizottságok számára.

Azt, hogy az RSA aláírási séma hogy alakítható vak aláírási sémává, a 3.1. szakaszban bemutatom, a DSA digitális aláírás vakításáról pedig [6] ír, szintén diszkrét logaritmuson alapuló aláírási sémát vakít Brands [3]-ban és ezzel ad egy teljes digitális fizetőrendszert is.

1.2. RSA nyilvános kulcsú kriptográfia

A Merkle által kidolgozott ötlethez az első megvalósítást, Rivest, Shamir és Adelman írta le[37] 1977-ben. A szerzők nevének kezdőbetűiből adódik az RSA név, az eljárás megfelelően használva azóta is Pareto-teljes.

A kulcsgeneráláshoz keresni kell két vagy több nagy prímet ($p_1, p_2, \ldots, p_m$, ehhez eredetileg a Solovay-Strassen[49] valószínűségi prímtesztet javasolták a szerzők, mostanra a Miller-Rabin[33] teszt az elterjedtebb). A prímek szorzata a kulcs modulusa ($n := \prod_{i=1}^{m} p_i$), annyi bitesnek hívják a kulcsot, amilyen hosszú a modulus. Választani kell még egy olyan $1 < e < \varphi(n) := \prod_{i=1}^{n}(p_i - 1)$ értéket, ami relatív prím $\varphi(n)$-hez. Szokásos választás a modulustól függetlenül a $3, 5, 17, 257, 65537$. A kis e értékekkel hatékony a rejtjelezés, azonban sok tekintetben óvatosnak kell lenni. Pareto-teljes választásnak a 65537 tekinthető. A kiválasztott e érték multiplikatív inverzét $(\mathrm{mod}\ \varphi(n))$ az euklideszi algoritmussal meg kell keresni, ez lesz a d.

Egy tetszőleges $0 < m < n$ üzenet rejtjelezése az üzenet e-dik hatványának kiszámítása $(\mathrm{mod}\ n)$, míg a megfejtő inverz művelet a d-edik hatvány kiszámítása. A műveletek fordított szereposztása az RSA digitális aláírás. Az algoritmus

helyességét a kis Fermat-tétellel lehet bizonyítani, a bizonyítást az eredeti RSA cikk tartalmazza.

A szerzők eredetileg két prím használatát javasolták, amik egyező nagyságrendűek (de egymástól kellően távoliak). A több prím használatát később szabványosította az RSA cég[23], illetve Shamir 1995-ben írt arról, hogy különböző nagyságrendű prímekkel úgy kaphatunk nehezebben feltörhető (nagyobb modulussal rendelkező) rendszert, hogy a használat során a műveletigény nem nő[47].

Az RSA biztonságát számos cikk vizsgálja, a gyakorlati megvalósítás során megjegyzéseiket[53, 1, 22, 9] figyelembe kell venni, mind a kulcsgenerálás, mind a rejtjelezés, digitális aláírás során. Mivel a műveletek lassúak, a rejtjelezéskor egy szimmetrikus rejtjelező kulcsot, a digitális aláírás készítésekor az üzenet egy biztonságos hash érték használják. Formálisan összefoglalva:

Kulcsgenerálás:

$$p_1, p_2, \ldots, p_m \text{ nagy prímek}$$

$$n := \prod_{i=1}^{m} p_i, \quad \varphi(n) = \prod_{i=1}^{m} (p_i - 1)$$

e választása: $1 < e < \varphi(n), (e, n) = 1$ (pl. 65537)

d kiszámítása: $de \equiv 1 \pmod{\varphi(n)}$

Nyilvános: (e, n), titkos: d

Rejtjelezés, aláírás ellenőrzés: $m^e \bmod n$

Megfejtés, aláírás készítés: $m^d \bmod n$

Helyesség: $m^{d^e} = m^{de} \equiv m \pmod{n}$

1.3. Diffie-Hellman kulcsmegegyezés

A Diffie-Hellman kulcsmegegyezés[12] segítségével úgy tud két egymás számára ismeretlen fél megegyezni egy közös titokban, hogy soha nem kell ehhez titkos csatornát használniuk. A megegyezés eredményeként kapott értéket használhatják

aztán, mint szimmetrikus titkosítókulcsot. Az érték amiben megegyeznek, harmadik fél által csak irreálisan magas költséggel található ki.

A protokoll egy olyan G ciklikus csoportban játszódik, aminek g a generátoreleme és a rendje q. A csoportot úgy kell választani, hogy benne a diszkrét logaritmus probléma nehéz legyen. Adott $1 \leq x \leq q-1$ esetén az $y = g^x$ érték meghatározása négyzetre emelések és szorzások sorozatával könnyű [25]. Ugyanakkor az y érték ismeretében az x meghatározására nem ismert gyors algoritmus.

Alice és Bob, akik közös titkos kulcsban kívánnak megegyezni, mindketten közzéteszik a rögzített G (q és g által meghatározott) csoportban a saját g^a, illetve g^b értékeiket, viszont az a és b logaritmusokat titokban tartják. Ezzel már meg is egyeztek egy közös titokban, hiszen a $(g^a)^b = (g^b)^a = g^{ab}$ értéket csak az tudja a nyilvánosan rendelkezésre álló információkból kiszámolni, aki vagy a-t, vagy b-t birtokolja.

Amennyiben szükséges, hogy minden alkalommal új közös titok jöjjön létre kettőjük között, akkor alkalmanként megegyezhetnek egy r véletlen értékben és az átmeneti közös titkuk lehet a $h(g^{ab}||r)$ érték. A h egy tetszőleges hash függvényt jelöl. Az így létrejövő kulcsok egymástól teljesen különbözőek (különböző r értékek mellett) a hash függvény miatt. Ezeket a kulcsokat már használhatják pl. egy szimmetrikusan rejtjelezett csatorna felépítésére.

1.4. ElGamal nyilvános kulcsú kriptográfia

Az előző részben bemutatott Diffie-Hellman primitívből könnyen építhető az RSA-tól teljesen független aszimmetrikus kriptorendszer. Ezt Taher El Gamal ismertette 1984-ben[20].

Azt szeretnénk, hogy a rejtjelezéshez csak a fogadó fél nyilvános kulcsára (g^x) legyen szükség, a küldő fél akár olyan is lehessen, aki nem rendelkezik kulcsok-

kal. A megoldás az, hogy a küldő fél generál egy kulcsot (y) magának csak ehhez az egy üzenethez és ennek a kulcsnak a nyilvános részét (g^y), valamint a rejtjelezett üzenetet ($m \cdot (g^{xy})$) együtt küldi át a hálózaton.

A fogadó fél a kapott nyilvános kulcsból ki tudja számolni a g^{xy} értéket és így osztani tud vele. Amit kap az az eredeti m érték. Persze m-nek a G csoportból kell egy elemnek lennie, de minden üzenet átalakítható ilyen értékek sorozatává, illetve a gyakorlatban úgyis csak egy session kulcsot rejtjeleznek így, amivel aztán szimmetrikusan kódolnak. Az általában használt session kulcs pedig úgyis kisebb, mint a G elemszáma.

A digitális aláírása egy $0 \leq m \leq q-1$ értéknek az (r, s) pár, ahol $r := g^k$. $s := (k^{-1} \cdot (m - xr)) \bmod q$, k pedig egy véletlen érték $[0, q-1]$ között, úgy, hogy relatív prím $q-1$-hez, ugyanis ekkor teljesül a következő kongruencia:

$$m \equiv xr + ks = xr + kk^{-1} \cdot (m - xr) \equiv m \pmod{q}$$

Azaz az adott m értékhez és a sorsolt k-hoz a bizonyító fél előállította az s megoldást, amit csak x birtokában tehetett. Az, hogy a közölt r és s valóban jó, az ellenőrző fél a hatványozás homomorfiájának köszönhetően g^k és g^x ismeretében le tudja ellenőrizni:

$$g^m \stackrel{?}{=} g^{xr+ks} = (g^x)^r + (g^k)^s$$

Az ellenőrizendő egyenletben $r = g^k$ és s értékek ismertek, ők alkotják az aláírást, g^x pedig az aláíró nyilvános kulcsa, ami elérhető.

1.5. Schnorr csoport

Az 1.3. szakaszban említettük, hogy az itt leírtak mindegyikéhez szükség van egy olyan (q-ad rendű) csoportra, amiben a diszkrét logaritmus probléma nehéz.

A diszkrét logaritmus probléma nehézségét befolyásolja természetesen a csoport nagysága, illetve az is, hogy a $g^1, \ldots, g^q$ értékeket hogyan reprezentáljuk.

Claus-Peter Schnorr ajánlása[58, 45], hogy válasszunk egy q prímet, majd keressünk egy másik $p = qr + 1$ alakú prímet, ahol r tetszőleges, de mérete természetesen meghatározza p nagyságát. Ezután keressünk a $[2, p-1]$ intervallumban egy olyan h-t, amire:

$$g = h^r \not\equiv 1 \pmod{p}, \text{ és így}$$
$$g^q = (h^r)^q = h^{p-1} \equiv 1 \pmod{p} \text{ a kis Fermat-tétel miatt.}$$

Tehát a g által generált csoport nem triviális és rendje osztója q-nak. Mivel q prím, ezért a g által generált csoport q-ad rendű.

Az ilyen $g^1, \ldots, g^q \pmod{p}$ csoportokat hívják *Schnorr-csoportnak*.

Ennek a megoldásnak az előnye, hogy a q és a p mérete egymástól függetlenül választható. A q méretét úgy kell megválasztani, hogy az ún. meet-in-the-middle jellegű támadásoknak ellenálljon, a p mérete pedig az index kalkuluson alapuló támadások hatékonyságát befolyásolja[42].

1.6. DSA digitális aláírás

A DSA egy széleskörűen elterjedt, szabványosított, Schnorr csoportban működő, a diszkrét logaritmus problémára épülő digitális aláírás.

A DSA algoritmust és helyességét ismerteti a Wikipedia[56], tárgyalja [44] és részletesen specifikálták, mint szabványt DSS néven[32], implementálja az OpenPGP és számos egyéb megvalósítása is van.

Egy DSA titkos kulcs egy Schnorr csoport rendjéből (160 bites prím: q), egy 1024 bites p prímből, a csoport g generátor eleméből és egy $0 < x < q$ (azaz 160 bites) számból áll. Titokban csak az x-et kell tartani, ugyanakkor a másik három

érték nélkül az x használhatatlan, ezért szokták a (p, q, g, x) négyest hívni a titkos kulcsnak. A nyilvános kulcs a (p, q, g, y) négyes, ahol $y = g^x \bmod p$.

Egy üzenet aláírásához ki kell számolni az üzenet SHA-1 hash függvény szerinti értékét, ami szintén 160 bites, jelölje ezt m. Az aláírás az (r, s) pár, ahol $r = (g^k \bmod p) \bmod q$ és $s = (k^{-1}(m + xr)) \bmod q$, az ellenőrző fél az ElGamal aláírási sémához hasonlóan a hatványok terében győződik meg az aláírás hitelességéről, a következő egyenletet kell ellenőriznie m, r és s birtokában:

$$r \stackrel{?}{=} ((g^{u_1} y^{u_2}) \bmod p) \bmod q, \text{ ahol } \begin{array}{l} u_1 = ms^{-1} \bmod q \\ u_2 = rs^{-1} \bmod q \end{array}$$

A DSA aláíró kulcsokkal rendelkező felhasználóknak rejtjelezett üzenetet is lehet küldeni, az ElGamal sémánál látott módon. Sőt, amennyiben a felhasználók nem generálnak külön–külön p, q, g értékeket, hanem egy közös Schnorr csoportot használnak, akkor ezekkel a kulcsaikkal Diffie-Hellman kulcsmegegyezésben is részt tudnak venni, amivel sokkal olcsóbban tudnak egymásnak rejtjelezett üzeneteket küldeni. A közös csoport használata – természetesen felhasználónként különböző x értékekkel –, jelenlegi ismereteink szerint a biztonságot nem csökkenti.

1.7. Az RC4 folyamtitkosító

Az RC4[57] egy szinkron folyamtitkosító, ami azt jelenti, hogy az algoritmusa a kulcstól függő álvéletlen sorozatot állít elő, amit a bináris „kizáró vagy” (XOR, összeadás) művelettel kombinálnak a nyílt szöveggel[60]. A rejtjelezett üzenet átvitele után a kulcs ismeretében előállítható az álvéletlen sorozat és így az összeadás újbóli alkalmazásával a nyílt szöveg.

Az algoritmust 1987-ben tervezte az RSA feltalálásában is résztvevő Rivest és üzleti titokként kezelték egészen 1994 szeptemberéig, amikor egy anonim feladó közzétette a Cypherpunks levelezőlistán. A publikálást követően hamar megjelen-

tek a programkód másolatai az internet más oldalain, így az algoritmus már nem tekinthető titoknak. Ugyanakkor az RC4 algoritmus nem hivatalos megvalósításai (ha a kimenet meg is egyezik minden esetben a hivatalos megvalósításával) nem használhatják a nevet, mivel az védjegy. Ezért terjedt el az ARC4, illetve az ARCFOUR név, amiből az első betű az állítólagosságra (alleged) utal.

Az algoritmus más blokk-, illetve folyamtitkosítókhoz hasonlítva egyszerű és nagyon gyors, a programkód 15-20 sor; a rövid, konstans idejű inicializációtól eltekintve futás ideje az előállított bájtok számával arányos.

A közzététel óta eltelt 14 év alatt az RC4 biztonságát számos szakértő vizsgálta, a legjelentősebb támadást [14] és [24] jelenti. Egy összeadással működő szinkron folyamtitkosító kulcsát sosem szabad újra felhasználni, hiszen két ilyen lehallgatott eredményt összeadva megkapjuk a két nyílt szöveg összegét. Ezért pl. a vezeték nélküli hálózatokat védő WEP esetében úgy döntöttek, hogy a rejtjelező berendezések között megosztott titokhoz minden adatcsomag esetében hozzáfűznek egy számot és az így kapott bájt sorozat lesz a kulcs. Az egyik említett cikk pont azt taglalja, hogy az RC4 hibái miatt ez a megoldás támadható, a nagyban hasonló kulcsok esetén kellő számú álvéletlen sorozatból magára a kulcsra is lehet következtetni. Az álvéletlen sorozatok megismerését a lehallgatást követően könnyíti, hogy tudjuk: a rejtjelezett tartalom protokollja az IP.

Jelenleg azonban nem ismert támadás akkor, ha a konkatenált kulcsnak egy hash értékét használjuk. A hash függvény közbeiktatása megoldja, hogy a nagyban hasonló kulcsokból is teljesen különböző legyen és így ez a támadás használhatatlan. Továbbá ezzel a „biztonsági övvel”, ha sikerül is egy egyedi esetben egy lehallgatott üzenetből a támadónak következtetnie a használt rejtjelező kulcsra, utána még az egyirányú függvényt is meg kellene fordítania, hogy a felhasználó által használt titkos kulcsot megkapja. Szintén az említett cikkekben leírt eredmények miatt el kell dobni az RC4 folyam első 256 bájtját, mivel ott az álvéletlen

kitétel nem teljesül. Az így használt RC4 olcsón nem támadható, azonban hatékonyabb, mint a más rendelkezésre álló alternatívák, pl. az AES.

1.8. Kriptográfiai hash függvények

Ezek a leképezések tetszőleges bájt sorozatokhoz rendelnek fix bithosszú kimenetet. Ezeket a függvényeket régóta vizsgálják, hiszen a kriptográfián kívül is hasznosak, könnyen meg lehet velük állapítani egy adat integritását pl. hálózati átvitel után vagy kétes adattárolóról visszaolvasáskor. A (vissza)olvasott adatra újra kiszámítjuk a hash értéket és ha az nem egyezik a letárolt vagy szintén hálózaton kapott értékkel, akkor tudjuk, hogy hiba keletkezett. Az algoritmusokat direkt úgy választják meg, hogy az adatban kis változás (hibás átvitel) nagyban eltérő hash értéket produkáljon. Egy ilyen ellenőrzőösszeget, a CRC-t[55] használja a zip és az arj tömörítő program is.

A kriptográfiai alkalmazásoknál is ugyanez az alapötlet, de plusz kikötéseket teszünk[4, 54] a függvényre:

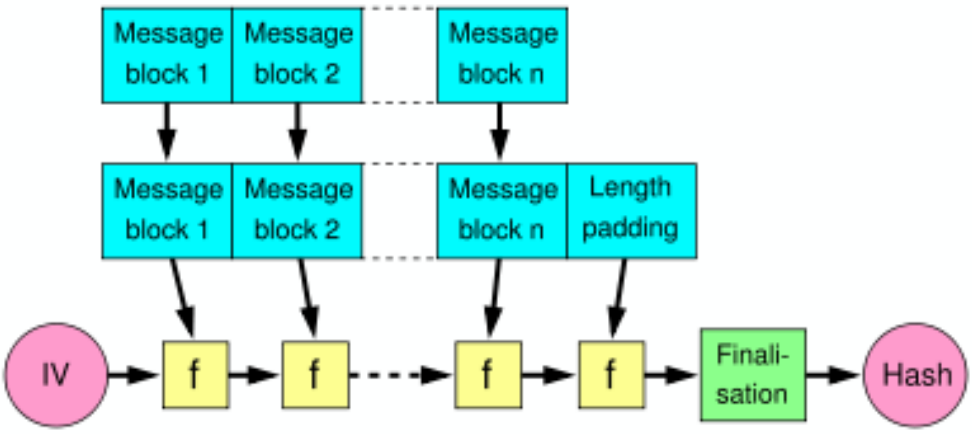
- legyen *őskép ellenálló*, azaz adott függvényértékhez költséges legyen olyan bájt sorozatot találni, aminek a hash értéke az adott függvényérték;
- legyen *második őskép ellenálló*, azaz adott bájt sorozathoz nehéz legyen egy másik bájt sorozatot felmutatni, aminek azonos a hash értéke;
- legyen *ütközés ellenálló*, azaz nehéz legyen két bájt sorozatot felmutatni, amiknek azonos a hash értékük.

Az ilyen hashek rendkívül hasznosak és elterjedtek a kriptográfiában, pl. a digitális aláírás hatékony megvalósítása is velük lehetséges. Ugyanis az antiszimmetrikus kriptográfiát megvalósító algoritmusok lassúak, azonban a nagy dokumentumok aláírása ennek ellenére megvalósítható: nem magát a dokumentumot,

hanem a hash értékét kell csak aláírni, ami rövid. A fenti feltételek biztosítják, hogy ezzel ugyanolyan szintű biztonságot érünk el, mintha a dokumentumot írtuk volna alá.

Sajnos azonban nincs olyan függvény, ami a gyakorlatban alkalmazható és bebizonyították volna róla maradéktalanul ezeket a tulajdonságokat. „Csupán” olyanok vannak, amik kiforrottak, régóta használtak és analizáltak, így feltételezhető róluk, hogy az ismert hibáiknál sokkal durvább támadás nem lesz lehetséges. Elővigyázatosságból, az ismertetésre kerülő ePoint és SecMS rendszerek protokolljai biztonságosak maradnak akkor is, ha a harmadik, legerősebb feltétel már nem teljesül a használt hashre.

A ma használt hash függvények a Merkle-Damgard konstrukcióra épülnek, ami a tetszőleges hosszú bájt sorozat feldolgozását visszavezeti két fix hosszú bájt sorozaton értelmezett egyirányú függvény előállításának a problémájára. A konstrukcióval nyert hash függvény erőssége megegyezik a felhasznált egyirányú függvény biztonsági tulajdonságaival.



1.2. ábra. A Merkle-Damgard konstrukció

A konstrukcióból következik, hogyha sikerül ütközést találni az f egyirányú

függvényben, akkor végtelen sok ütköző pár létezik a hash függvényre nézve, sőt ezek szisztematikusan generálhatók, csak teljesen azonos blokkokat kell az ütköző blokk mögé, illetve elé tenni. Ezzel a módszerrel bonyolultabb dokumentumleíró nyelvek vagy formátumok esetén (mint pl. a PostScript, PDF, XLS, DOC), illetve számítógépes programoknál tetszőleges két teljesen máshogy működő, mást jelentő példányt állíthatunk elő azonos hash értékkel. Így feltörve az adott hashre épülő digitális aláírást. Az MD5 ilyen támadását részletesen ismerteti [27], építve arra az eredményre hogy az MD5 már nem ütközés mentes[51, 50]. Hasonló eredményekre kell számítani a többi elterjedt hash-sel, pl. a SHA-1-gyel kapcsolatban is.

„Biztonsági övként" megpróbálunk mindig minél egyszerűbb formátumokat használni, amiben a csalás ténye már ránézésre kibukik. A hivatkozott cikkben szereplő támadás például nem lenne lehetséges, ha Caesar ASCII kódolt szövegfájlokat használna.

Két elterjedt, biztonságos hash függvény:

– SHA-1: 20 bájtos (160 bites) értéket állít elő, többek közt az OpenPGP szabvány használja integritás védelemre, illetve a kulcsazonosítók és a kulcsok ujjlenyomatát is ezzel a függvénnyel számítja. 2005-ben ismertté vált egy módszer, amivel ütköző sorozatok előállításának bonyolultsága (a születésnap paradoxonból következő) 2^{80}-ról 2^{63}-ra csökkent. [52]

– RIPEMD-128: 16 bájtos (128 bites) értéket előállító hash függvény, a SecMS jelenlegi protokolljában ezt használjuk minden olyan helyen, ahol az OpenPGP-vel való kompatibilitás nem teszi szükségessé a SHA-1 használatát. Jelentősen gyorsabb a SHA-1 függvénynél (vagy a szintén 160 bites RIPEMD-160-nál); ez a sebességkülönbség fontos lehet, amikor a kliens-számítógép egy mobiltelefon. 2004 augusztusában találtak egy ütközést az eredeti RIPEMD függvényben. [51]

1.9. SSL/TLS

Az SSL, illetve annak továbbfejlesztett verziója, a TLS[11] egy olyan protokoll, ami tetszőleges TCP kapcsolat rejtjelezését és integritásának biztosítását szabványosítja. A hitelességről is gondoskodik, a kliensprogramok authentikálhatják magukat a szerverek felé, a szerverek is a kliensek felé.

A protokoll rendkívül rugalmas, mind RSA, mind DSA kulcsokat támogat, kriptográfiai hash függvényekből is választhat a felhasználó. Amikor kiderült az említett MD5 elleni támadás, egyszerűen, a szoftverek lecserélése nélkül lehetett olyan kulcsokra váltani, amik a SHA-1 hash függvényt használják.

Ami a legnagyobb problémája ennek a protokollnak, hogy jelenleg kulcskezelésre csak az X.509 szabványt támogatja, ami ahogy a 1.1. szakaszban láttuk, hierarchikus. Minden programba (böngészőbe, email kliensbe) beépítenek 10-es nagyságrendű, ún. tanúsítókhoz (certificate authority) tartozó nyilvános kulcsokat. A böngészők a tanúsítók által kiállított tanúsítványokat fogadják el hitelesnek, csak olyan szerverekkel állnak a felhasználó figyelmeztetése nélkül biztonságos kapcsolaton szóba, amelyek kulcsát aláírta valamelyikük.

Azonban az ilyen aláírás beszerzése rendkívül költséges, a gyakorlatban csak nagyobb cégek (bankok, internetszolgáltatók) engedik meg maguknak, hogy kifizessék a tanúsítás árát. További probléma, hogy a tanúsítás folyamata sokszor nem is személyes ellenőrzésre alapszik (hiszen az még jobban megnövelné a tanúsítás költségét), hanem egyszerűen olyan rendszerek megfelelően való működését ellenőrzi csak, amik könnyen támadhatók és hamisíthatók (email, DNS). Ráadásul nem is minden böngészőben ugyanazok a tanúsítócégek találhatóak meg beépítve, így az esetleges többszörös tanúsítás tovább növelheti a felhasználók költségeit. Mindezek eredménye az, hogy az általános felhasználási területen a rendszerek tulajdonosai nem foglalkoznak a kulcsok hitelesítésével, hanem a felhasználók megtanulták figyelmen kívül hagyni az erre a tényre utaló figyelmeztetéseket. Sőt,

amennyiben egyáltalán nem gondoskodnak a kapcsolat rejtjelezéséről a rendszergazdák, akkor semmilyen figyelmeztetést nem kap a felhasználó.

Összességében az X.509 modelljén alapuló kulcskezelés a nem hierarchikusan rendeződő interneten a man-in-the-middle támadás ellen, ami miatt készítették, lényegében nem véd. Sőt, azzal, hogy a felhasználókat a gondolkodás és mérlegelés nélküli „folytatás” gomb választására szoktatja, a későbbi esetleg jobb modellek hasznosságát is rontja.

Szerencsére már elkészült az SSL/TLS-t OpenPGP kulcsokkal kiegészítő szabvány[28]. Amely szoftverek ezt implementálják, azok számára lehetővé válik, hogy a SSL/TLS-hez használt szerver-, illetve kliens kulcsokat ne hitelesítő szervezetek, hanem maguk a felhasználók írják alá. Mint azt az OpenPGP tárgyalásánál látni fogjuk, annak az esélye, hogy egy ilyen kulcshoz találunk megbízható „utat” sokkal nagyobb, mint annak, hogy az X.509-ben terjesztett kulcsban megbízhatnánk.

2. fejezet

Biztonságos üzenőrendszerek

Minden kliens–szerver architektúrájú rendszer védhető azzal, hogy rejtjelezzük a szerver és kliensek közti csatornákat SSL/TLS protokollal.

Már ez is segítség, egyre több SMTP és POP3/IMAP szerver képes erre, de ahogy a bevezetésben említettük, a mi céljainkra ez nem elegendő. Ugyanis ettől még a rendszerek gazdái ugyanúgy hozzáférhetnek jogosulatlanul az adatokhoz, illetve megbízható rendszergazdák esetén is egyre nehezebb lesz ezeknek a rendszereknek a védelme. Minél több felhasználóval rendelkezik egy szerver, annál nagyobb előnyt szerez, aki feltöri, így a betöréssel járó költségeket folyamatosan emelni kell.

Ha ezekről a rendszerekről nem tesszük fel, hogy ők megbízható harmadik felek, azzal jelentősen csökkentjük az üzemeltetésük költségét és az egyik legsűrűbben kihasznált veszélyforrást is kiküszöböljük.

2.1. Email

Az email az egyik legelterjedtebb és talán legbonyolultabb üzenetküldő rendszer. Sok különböző protokoll integrációjával valósul meg a üzenetek továbbítása és le-

töltése. A felhasználók mindenféle számítástechnikai környezetből el tudják érni, létezik csak szöveges alapú levelezőprogram, lehetőség van a levelek szerveren tárolására vagy letöltésére is. Elterjedtségének köszönhetően bárkivel felvehetjük a kapcsolatot emailben.

Az emaillel foglalkozó eredeti szabványoknak egyáltalán nem volt célkitűzése egyik biztonsági követelményünk (I. rész) megvalósítása sem. Ha valaki el is tekint a konfidencialitástól és integritástól; a hitelesség hiánya számára is zavaró. Nem lehet megállapítani, hogy egy levelet kitől kaptunk, ezért automatikus szűrésre az esetek nagy részében nincs egyszerű és biztos mód. Így aztán az ún. spam levelek (szemétlevelek) feladása és kézbesítése nem akadályozható meg, azok visszaszorítása rengeteg gépi és emberi erőforrást emészt fel és az elért hatásfok messze nem kielégítő.

Születtek megoldásjavaslatok az említett követelmények teljesítésére, de ezek elterjesztése nem vagy csak részben sikerült. Majdnem minden ilyen megoldás arra támaszkodik, hogy a felhasználót teljes mértékben érdekelt félnek tekinti és rá próbálja kényszeríteni, hogy a biztonság elérésének érdekében órákig vesződjön a különböző problémák szakértőszintű beállításával. Két ilyen létező megoldást is bemutatok a fejezet további részében.

2.1.1. S/MIME

Az email formátumát először dokumentáló szabvány[10], mind a levél fejlécében szereplő értékeket, mind a levél törzsét, mint 7 bites értéket határozta meg. Így abban a formátumban átalakítások nélkül nem lehetséges nemzetközi írásjeleket továbbítani, illetve csatolt dokumentumokat sem.

A dokumentumok csatolását nem csak az akadályozza, hogy sokszor, amit csatolnánk az bináris, hanem az is, hogy a levél törzse a szabvány szerint egyszerűen egy hosszú szöveg. Tehát, ha a csatolandó dokumentum még 7 bites is, akkor is a

fogadó félnek valahogy észre kell vennie, hogy a levél mely része a csatolt dokumentum, mennyi csatolt dokumentum van, azok meddig tartanak, milyen típusú adatot tartalmaznak, mik a fájl neveik, stb.

Ezeket a problémákat hivatott megoldani az RFC szabványok MIME családja [16, 17, 30, 18, 15], amelyekkel egy adatfolyam feloszthető sok részre, megadható, hogy mely részek egymás különböző alternatívái (pl. egy email text/plain és text/html formátuma), melyek egymással párhuzamos médiák (pl. hang és kép). Az egyes részekhez különböző jellemzők is kapcsolhatók, pl. nem ASCII adathoz a kódolás módja. Így a levél törzsében használható tetszőleges karakterkészlet, illetve tetszőleges nem szöveges média is. Ezt az eredményt később más szabványok készítésekor is felhasználták. Például ugyanez a probléma áll fenn akkor, ha egy böngészőnek kell elküldenie egyetlen kérésként a felhasználó több fájlját egy távoli szerver számára. Ekkor is egy csatorna (a HTTP kérést tartalmazó törzs) áll csak rendelkezésre, amit több részre kell osztani.

A levél fejlécrészében (például a tárgy mezőben) használható kiegészítéseket is ennek a szabványcsaládnak egy tagja specifikálja. Ennek felhasználásával lehetséges pl. ékezeteket írni egy email tárgyába vagy feladójába.

Az MIME tehát azt teszi lehetővé, hogy egy hierarchikusan rendezett dokumentumcsokrot ábrázoljunk egyetlen karakterláncként. Az S/MIME[34, 35] ezt azzal egészíti ki, hogy a hierarchiának megfelelő fa bizonyos részeit digitálisan aláírhatjuk, illetve rejtjelezhetjük. Minden fontosabb algoritmust támogat a szabvány, az implementációk választhatnak különböző hash függvények, DSA és RSA kulcsok közül. Ugyanakkor egy kulcs megbízhatóságát csak hierarchikusan, tanúsítókon keresztül ellenőrizhetjük, mivel az S/MIME kulcskezelése X.509 alapú. Ez a hibája a szabványnak, ami miatt nagyon kevesen használják. A tanúsítás (konkrét anyagi) költsége messze meghaladja azt az értéket, amit egy hétköznapi felhasználó erre a célra szánna.

2.1.2. OpenPGP

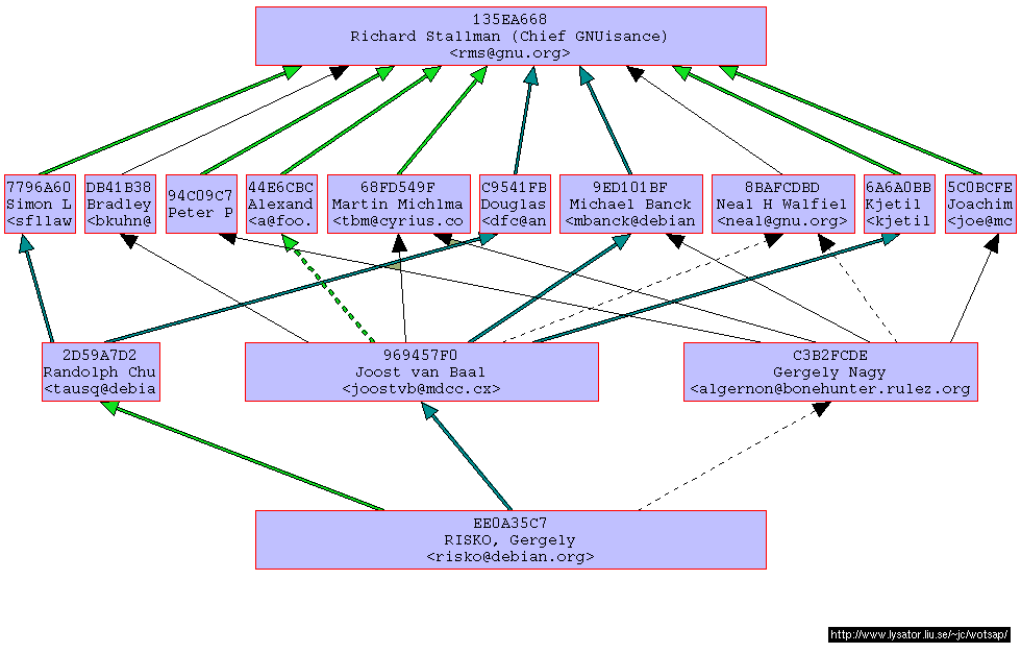
Az OpenPGP egy jelenleg is aktívan fejlesztett és karbantartott, kriptográfiai szabálygyűjtemény, melyhez már számos implementáció készült. Specifikálja DSA és RSA kulcsokhoz a (nem hierarchikus) kulcskezelés pontos módját, üzenetek digitális aláírását, rejtjelezését. Moduláris felépítésű, rendkívül sok területen használható, pl. specifikál mind szimmetrikus, mind antiszimmetrikus rejtjelezést, többféle blokktitkosítóval. A létrehozott üzenetek öntartalmazóak, nem szükséges semmilyen más kommunikáció vagy metaadat ahhoz, hogy az OpenPGP ismeretében később értelmezhetőek legyenek. A szabványt követő implementációk így probléma mentesen fel tudják használni az egymás által létrehozott üzeneteket. Sőt a használt titkos- és nyilvános kulcsokat is, így két különböző termék közötti váltás sem okoz kompatibilitási problémát.

Az OpenPGP-nek a 3-as verziója volt az első széleskörűen elterjedt és használt adatformátuma, azonban ez sok hiányossággal rendelkezett: csak RSA kulcsokat kezelt, az aláírások nem voltak kellően flexibilisek. Kiderültek azóta biztonsági problémák is a használt MD5-tel, illetve a kulcsazonosítók és ujjlenyomatok generálásának módjával kapcsolatban. Az aktuális OpenPGP szabvány, az RFC 4880 tiltja az új implementációk számára a 3-as verziójú kulcsok létrehozását, a 4-es verzió használatát javasolja.

A nem szimmetrikus kulcskezelés a legnagyobb különbség az OpenPGP és a konkurens technológiák között, ezzel biztosítják a DSA és RSA kulcsok megbízható közzétételét. Abban, hogy egy nyilvános kulcs valóban a tulajdonosához tartozik, annál biztosabbak lehetünk, minél több olyan személy állította ezt, akiben már megbízunk. Ezeket a kulcshitelesítő üzeneteket az interneten több ún. kulcsszerver gyűjti és folyamatosan szinkronizálja. A készült statisztikák[1] szerint az én kulcsom 15 ilyen aláírással rendelkezik és így átlagosan 4-5 lépéssel meg tudok

[1] http://pgp.cs.uu.nl/mk_path.cgi?STAT=ee0a35c7

győződni valaki kulcsának a megbízhatóságáról (és fordítva). Ez az infrastruktúra már kiforrott és sokak által használt, így a valóban előforduló esetek többségében sokkal kevesebb lépés is elegendő, pl. a szabadszoftver-mozgalom alapítója, Richard Matthew Stallman csupán egy lépéssel megbizonyosodhat egy általam adott aláírás esetén az ellenőrzéshez használt kulcsom hitelességéről. Sőt, két közbülső ember közül is választhat. Fordított irányban két lépésre van szükség, azonban a sok követhető útnak köszönhetően a biztonság garantálható:



2.1. ábra. Riskó Gergely — Richard M. Stallman

A PGP első verziója korábbi a MIME megjelenésénél, így a szükségszerűen bináris rejtjelezett adat 7 bites csatornán való közlésére saját megoldása van. A MIME elterjedése után azonban specifikálták a PGP/MIME-ot[13], amivel az S/MIME-hoz hasonlóan a MIME fa bármely része rejtjelezhető és/vagy aláírható és az így készült rész a MIME fába visszailleszthető.

2.2. SMS

A mobiltelefonok között igen sokat használt SMS üzenetek (a valóban forgalmazott adatmennyiséghez képest) nagyon költségesek, biztonságosnak viszont nem mondhatók. A mobil környezetben való használhatóság fontosságát viszont jól mutatja elterjedtségük, ami ennek köszönhető.

Az SMS rendszerben feladott üzenetek rejtjelezése csak a mobil entitások és az adó–vevő tornyok között megoldott, a szolgáltatók belső, illetve a szolgáltatók közötti hálózatokon nincs előírva semmilyen titkosítás.

Az integritás, illetve a hitelesség nem biztosított, pl. a számhordozás után, annak megtörténtéről olyan SMS üzenetet kaptam a szolgáltatótól, aminek feladó rovatában a szolgáltató neve szerepelt, ami olyannyira hamis feladó, hogy nem is értelmezhető az adott kontextusban — telefonszámként.

Habár az SMS ad lehetőséget visszaigazolás kérésére, annak kézhezvétele csupán annyit jelent, hogy az adott pillanatban a címzett GSM készülék a hálózathoz kapcsolódva volt és számára az üzenetet az adótorony lesugározta. Arra vonatkozóan nincs információ, hogy az adott üzenetet tudta is fogadni, illetve, hogy azt változatlan formában kapta meg és elolvasta.

2.3. Instant üzenetváltás, Jabber

A Jabber az XMPP[39, 40, 41] protokollon alapuló azonnali üzenetküldő megoldás. Széles felhasználói táborral rendelkezik, számos program készült, amellyel ez a hálózat elérhető. A kliensprogramok legtöbbje támogatja az integritás, hitelesség és konfidencialitás biztosítását, *feltételezve*, hogy megbízunk a szerverprogramban. Ezt egyszerűen úgy érik el, hogy minden kapcsolatot, ami a szerverek, illetve a kliensek és a szerverek között születik titkosítanak SSL/TLS protokollal.

Mivel az XMPP könnyen kiterjeszthető, születtek megoldások, amelyekkel

biztonsági követelményeink mindegyike megvalósítható, anélkül, hogy megbíznánk harmadik félben, ilyen például a [38]-ban ismertetett kiterjesztés.

Itt a követelményeket egyszerűen az üzenetek S/MIME titkosításával és aláírásával oldják meg. Ennek következménye, hogy egy ilyen rendszerben folytatott minden beszélgetésünk megfejtés után harmadik fél számára bizonyíthatóan hiteles. A legtöbb privát felhasználási módot ez a következmény ki is zárja.

Született egy másik, nem szabványos megoldás is, mivel felismerték ezt a problémát, ez az „Off-the-Record Messaging”[2] nevet viseli. A használt kriptográfiai elemek feltételezik — lévén, hogy azonnali üzenetküldésről van szó —, hogy a kommunikáló felek ugyanazon időpontban elérhetőek és a hálózatra csatlakoztak. Ezzel a plusz feltételezéssel azonban megtudnak valósítani egy érdekes új követelményt, nevezetesen: a váltott üzenetek lehallgatásával nyert archív anyag nem fejthető meg azután sem, hogy a támadó valamelyik fél titkos kulcsának birtokába kerül.

Mindkét kiterjesztésről elmondható, hogy mivel intenzíven használ hatványozást, a mai mobiltelefonok többségében implementációja nem megvalósítható, úgy, hogy az a felhasználónak élvezhető sebességet nyújtson.

2.4. SecMS

A SecMS az ELTECRYPT fejlesztés alatt álló üzenőrendszere, ami az OpenPGP 4-es verziójú csomagjaira és DSA kulcsformátumára épít, azonban az OpenPGP-t olyan új RC4 alapú kriptográfiával egészíti ki, aminek segítségével a gyors, de biztonságos működés minimalista környezetekben, pl. (ahogy a névválasztás is sugallja) mobiltelefonon is megvalósítható.

A SecMS az előző részben taglalt XMPP alapú megoldáshoz hasonlóan nem kötelezi a feleket, hogy harmadik fél számára is bizonyítóerejű üzeneteket váltsa-

nak, ugyanakkor lehetővé teszi ilyenek készítését (pl. pénzügyi tranzakciók esetén későbbi viták rendezéséhez).

A SecMS nem követeli meg, hogy a kommunikáló felek egyszerre online legyenek, az üzeneteket (természetesen rejtjelezett formában) szerverek továbbítják. Kutatócsoportunknak a számítógépes, illetve a mobil platformon működő kliensen túl egy referencia szervere is készen van.

A SecMS protokolljait a diplomamunka III. részében részletesen bemutatom.

3. fejezet

Biztonságos fizetőrendszerek

Bár nagy lépés a kéretlen levelek megakadályozásában, ha azonosíthatóak a feladók, nem gondoljuk, hogy csupán ezzel a segítséggel a probléma teljesen megoldható. Az általunk javasolt SecMS rendszer egy fizetési rendszerrel fog rendelkezni, aminek egyik célja pontosan az lesz, hogy ha kívánjuk, leveleink mellé kevéske pénzt csatolhassunk, így biztosítva az elolvasásukat. Természetesen amennyiben a címzett korrekt és azt tapasztalja, hogy levelünk őt valóban érdekelte, nem pazarolta az idejét, akkor válaszában a küldött pénzt számunkra visszajuttathatja.[19, Slicing Spam]

Egy a készpénzhez hasonló, de elektronikus fizetőeszköz persze számos más felhasználási területtel is rendelkezne. Ezekből mutat be pár példát a dolgozat utolsó része.

Ahhoz, hogy a készpénz fényében értékelni tudjunk fizetőrendszereket, először át kell tekintenünk a készpénz azon tulajdonságait, aminek elterjedtségét köszönheti. Ezeket Nagy összegyűjtötte az ePointot ismertető cikkében. Az alábbiakat említette[31]:

– A készpénzzel végrehajtott tranzakciók *anonimak*, nem követhetőek harmadik fél számára. A kibocsátó számára sem.

– A tranzakciók *véglegesek*, a vevő vagy bárki más nem tudja önkényesen érvényteleníteni egyiket sem.

– A fizetések *peer–to–peer* jellegűek, nincs különbség vevő és eladó fél között, bárki fizethet bárkinek. Ahhoz, hogy készpénzt fogadjunk el, nem szükséges szerződést kötnünk az ügylettől független harmadik felekkel.

– A készpénzt használók részt tudnak venni ún. *naiv tranzakciókban*. Ez azt jelenti, hogy aki nincs tisztában az összes biztonsági intézkedéssel (vízjel, fémszál, stb.) vagy nincsenek meg az eszközei ezek vizsgálatára, az is el fogadhatja a bankjegyeket.

– A kibocsátó által kibocsátott pénz mennyiség nyilvános információ, a *kibocsátó auditálható*.

A tranzakciók véglegessége fontos költségcsökkentő tényező, ugyanis így nem szükséges a termékek árába beépíteni a visszafordítás elleni biztosítást.

A peer–to–peer tulajdonság lehetővé teszi, hogy bárki, kis szereplők is a piacra lépjenek, így szélesíti a fizetőeszközzel elérhető szolgáltatások körét és a versenyt.

Amennyiben egy fizetőeszközzel lehetséges naivan kereskedni, az jelentősen megnöveli azon helyzetek számát, amikor a fizetőeszköz felhasználható. Valamint a felhasználók biztonságérzetét is növeli, ha valamilyen hiba (áramszünet, készülékproblémák) esetén is tudnak a megbízható üzletfeleikkel kereskedni és az ellenőrzéseket későbbre halaszthatják.

A kibocsátó ellenőrizhetősége létfontosságú, Nagy véleménye szerint azért nem terjedt el eddig egy digitális fizetőrendszer sem, mert ezt a követelményt eddig figyelmen kívül hagyták. Ha a kibocsátó tetszőleges mértékben észrevétlenül tud pénzt kibocsátani, az szükségszerűen a fizetési rendszer hiperinflálódásához vezet.

3.1. Chaum vak RSA aláírásokra épülő rendszere

Chaum ötlete[7, 8] az, hogy az RSA digitális aláírás vakításával megvalósítható a 1.1.1. alszakaszban leírt módon az elektronikus készpénz.

Tehát azt kell csak kitalálni, hogy hogyan tud egy tanúsító fél olyan bankjegyeket digitálisan aláírni, amelyeket sosem látott.

Tegyük fel, hogy a bank rendelkezik az (e, n) nyilvános kulcshoz rendelt (d, n) titkos RSA aláírókulccsal. Alice, a készülő bankjegy tulajdonosa előkészíti azt, sorsol egy kellően nagy véletlen számot (x) és kiszámolja annak egy hash függvény által felvett értékét ($f(x)$). Kell még sorsolnia egy r véletlen számot, majd a bank számára megküldi az $r^e \cdot f(x)$ értéket. A bank a kapott számot felhatványozza a titkos d-edik hatványra és az így kijövő $(r^e \cdot f(x))^d = r^{ed} \cdot f(x)^d \equiv$ $\equiv r \cdot f(x)^d \pmod{n}$ számot visszaküldi Alicenak. Alice a moduláris inverzét kiszámolva r-nek meg tudja alkotni az $(x, f(x))$ párt, ami pontosan egy egységnyi értéket képvisel.

Alice amikor Bobnak fizet, akkor ennek a párnak az átnyújtása után Bob a banknál a párt egyetlen egyszer készpénzre válthatja. Ha a beváltás sikeres, akkor Alice megkapja a vásárolt terméket Bobtól. A beváltáskor a bank nem tudja, hogy Alice és Bob üzletelnek egymással, ugyanis a Bob által adott pár második tagja már nem tartalmazza az r tényezőt. Így a fizetés anonim.

A dupla költések ellen a bank nyilvántartása segít, kétszer semmilyen értéket nem lehet beváltani. Egy, az ún. cut–and–choose módszerre épülő javaslattal a rendszer módosítható úgy, hogy az anonimitás és a dupla költés elleni védekezés az internet állandó használata nélkül, offline módon is biztosítható legyen[8]. Illetve az is megoldható, hogy ne minden bankjegy ugyanannyit érjen. Az ötlet lényege az, hogy a felhasználónak előírt formájú, nem teljesen véletlen (mint az x esetében) bankjegyeket kell készítenie. Azonban a bank továbbra is vakon ír alá, így nem tudhatja, hogy betartja-e az előírásokat a felhasználó. A két követelmény

úgy egyeztethető össze, hogy a felhasználó előállít k darab megfelelő bankjegyet, mindegyikkel kapcsolatban elkötelezi magát (pl. a hash érték közlésével), majd a bank $k-1$ darab tetszőlegesen választott felfedését megköveteli és ha a $k-$ -1 darab esetében minden helyes, akkor a maradék 1-et hitelesíti. Habár az ötlet szép, a gyakorlatban jelentősen megnöveli a kommunikációs költségeket, ezért a gyakorlatban $k=2$ értékkel valósították meg. Mivel a rendszerben a felhasználók névhez és bankokhoz kötöttek, ez a biztonság is elegendő, csalás kísérlete esetén a felhasználó ellen el lehet járni.

Az így kiterjesztett rendszer habár még mindig anonim, a többi követelményünket nem vagy csak részben teljesíti. A fizető fél közreműködése esetén a tranzakció könnyen visszafordítható. A bankjegy elfogadó helyeknek kódokat kell kiosztani az offline fizetés biztosításához, azaz a rendszer nem peer–to–peer, szerződést kell kötni a kibocsátóval, hogy elfogadó helyet indíthassunk. Naiv tranzakciók lebonyolítása sem lehetséges Chaum rendszerében. A legnagyobb probléma azonban az utolsó követelmény hiánya, látható, hogy az aláíró kulccsal rendelkező személy annyi aláírást készíthet, amennyit csak akar és a vak tulajdonság remekül biztosítja, hogy ha esetleg kiderül, hogy túlbocsátotta magát, akkor sem lehet megmondani, hogy melyik bankjegy jogos és melyik jogtalan. Igazából azonban kis túlbocsátás ki sem derül, tekintve, hogy csak a beváltott bankjegyekről kell adatbázist vezetnie. Ezért ösztönözve van a túlbocsátásra.

3.2. ePoint

Az ePoint[31] alapötlete egész más, nem szükségesek vak aláírások. Arra van szükség, hogy a kibocsátó egy nyilvánosan elérhető ígérvény adatbázist üzemeltessen. Ebben a tranzakciók sorban kerülnek bejegyzésre, mindegyiknek van egy sorszáma és minden ígérvényt digitálisan aláír a kibocsátó.

Egy S_i ígérvény a következő adatoknak a rekordja:

- i: az egyedi sorszám;
- I: a kibocsátó neve, azonosítója;
- V_i: az érték (valamilyen meghatározott egység számszorosa), aminek kifizetését a kibocsátó vállalja, a
- C_i kriptográfiai kihívás megoldójának számára,
- N_i: indoklás (az ígérvény kibocsátását kérvényező üzenet).

Aki a C_i kihíváshoz tartozó R_i megoldást meg tudja adni (amihez szükséges a D_i titok), az rendelkezik a V_i mennyiségű ePointtal.

Alice úgy tud pénzt adni Bob számára, hogy a kibocsátónak megmondja egy meglévő S_i-hez a kihívás (C_i) megfejtését (R_i), és egy új C_j ($j > i$) kihívást, amit Bobtól kapott. Az utasítás hatására a kibocsátó létrehoz egy új S_j-t, aminek az adatai megegyeznek a régi S_i-vel, csupán a kihívás más, illetve az indoklás részben feltünteti a kibocsátó az R_i-t.

Mivel a nyilvántartás publikus, mindenki látja, hogy a D_i titok (és a segítségével előállítható R_i megfejtés) már nem képvisel értéket, így dupla költés nem lehetséges. Ugyanakkor az új rekordhoz már csak Bob tudja a D_j titkot, így a pénz átruházása valóban megtörtént. Természetesen szükség van egyéb üzenettípusokra is, ezeket az 5. fejezetben bemutatom.

Tehát az ePoint rendszerben a pénztárca programban szereplő titkok felelnek meg a pénztárcánkban lévő bankjegyeknek. A bankjegyek azon tulajdonságát, hogy nem másolhatók, pedig a nyilvánosságra hozással lehet elektronikusan utánozni. Nyilvánosságra hozni egy adatot csak egyszer lehet és a nyilvánosságra került adatot már nem lehet visszavonni.

A felvázolt rendszer így anonim és a tranzakciók visszafordíthatatlanok. Minden résztvevő lehet eladó és vevő is (az 5. fejezetben látni fogjuk, hogy a szerver nem is tudja, hogy éppen kivel áll szemben), a rendszer peer–to–peer. Naiv tranzakciók bonyolítására lehetőség van, Alice egy bankjegyének a D_i titkát egyszerűen odaadhatja Bobnak, aki akkor dolgoztatja fel a cserét a szerverrel, amikor kívánja. A nyilvános adatbázis végigjárásával bárki, aki internethozzáféréssel rendelkezik ellenőrizheti, hogy a kibocsátó minden kérést rendben hajt végre és meggyőződhet a kibocsátott pénz mennyiségéről, így az auditálás megoldott.

III. rész

A SecMS és az ePoint részletes bemutatása

A SecMS az OpenPGP 4-es verziójú csomagjaira épül, bemutatom az OpenPGP 4-es verziójának szükséges részeit, majd azokat a kiegészítéseket, amikkel lehetővé tettük az OpenPGP számítástakarékos, illetve kliens–szerver környezetben való használatát.

Az ePoint specifikációját is részletesebben bemutatom, mint az előző részben, megnézzük a konkrét tranzakciók lebonyolításának módját és a technikai, műszaki megvalósítás pár részletét.

A pontos, apróságokat is leíró specifikációt [26], illetve [46] tartalmazza.

A rész utolsó előtti fejezetében specifikálom a SecMS ePointra épülő levélszemét kezelő protokollját, illetve az ePoint SecMS-re épülő számlázási és fizetési protokollját.

Az utolsó fejezetben olyan problémákat vizsgálok, amik jelenleg a mikrofizetések megoldatlansága miatt léteznek és így az ePointtal megszüntethetőek.

4. fejezet

SecMS

A SecMS üzenetek az emailhez hasonlóan szervereken keresztül jutnak el a feladótól a címzetthez. Lényegi különbség a már leírt biztonsági követelményeken túl, hogy a protokollt a kezdetektől fogva úgy terveztük, hogy lehetőség legyen a kliens–szerver paradigmán túlmutató peer–to–peer üzenetküldésre is, pl. bluetooth vagy SMS használatával, esetleg valamilyen instant üzenőrendszerbe (XMPP, MSN, stb.) ágyazva.

Az OpenPGP[5] szabványra építve pontosan specifikáltuk az üzenetek rejtjelezését, integritás-védelmét, címzését, ezért válik lehetségessé a fent leírt csatornafüggetlenség, hiszen tetszőleges csatornán átvihetők a specifikált bájt sorozatok. Az sem okoz problémát, ha a csatorna nem 8 bites, ugyanis az OpenPGP-ben már erre is gondoltak, az ún. armoring technológiával megoldható tetszőleges OpenPGP csomag 7 bites sorozatként való ábrázolása.

Természetesen a mindennapi felhasználás során túlnyomóan a szerveren keresztül váltanak üzeneteket a felhasználók, hiszen ez a módszer érhető el a legolcsóbban a lehető legtöbb platformról. Ezért az OpenPGP specifikációját nem csak a SecMS üzenetek kriptográfiájával kellett kibővítenünk, hanem lehetővé kellett tennünk az OpenPGP-ben kliens–szerver parancsok leírását is. Hasonlóan ahhoz,

ahogy az email működéséhez is szükséges volt a pontos fejléc- és formátumleírásokon túl az SMTP, POP3 (vagy IMAP) protokollok specifikálása.

4.1. OpenPGP

Az OpenPGP specifikációja nagyon bonyolult. Implementációt csak az eredeti specifikáció alapján[5] szabad készíteni, semmiképp nem az itt kiemelt részletek szerint. A szabvány vizsgálata közben nagy segítségére lehet a fejlesztőnek a `pgpdump` nevű segédprogram, amivel vizsgálhatja egy bájt sorozatnak a szabvány szerinti felépítését.

4.1.1. Alaptípusok ábrázolása az OpenPGP-ben

- *előjel nélküli számok*: a szükséges mérettől függően 2 vagy 4 bájton, mindig big-endian formátumban.
- *előjel nélküli, nagy számok (MPI-k)*: a kriptográfiában sokszor szükséges különösen nagy természetes számok kezelése, ezek ábrázolása egy olyan bájt sorozatként történik, melynek első két bájtja mondja meg (előjel nélküli számként) az ábrázolt szám bitjeinek számát, majd maga a szám következik big-endian formátumban. Az ábrázolás mindig egész számú bájton történik, a fölösleges bitek értéke nulla kell, hogy legyen és azokat a bal oldalon kell szerepeltetni. Az említett bithosszleírás ettől függetlenül, csak az értékes bitek számát jelzi.
- *kulcsazonosítók*: 64 bites számérték, ami egy nyilvános kulcsot azonosít (nem feltétlenül egyértelműen), ezt az értéket a nyilvános kulcsból és a létrehozási idejéből egy hash függvény segítségével számolják.
- *szövegek*: Unicode karakterláncok az UTF-8 szabvány szerinti kódolással.

– *másodperc felbontású dátum*: 4 bájtos számérték, ami az 1970. január 1-e éjfél óta eltelt másodpercek számával ábrázolja az időpontot. Ez ugyanaz az ábrázolás, mint a (32-bites) Unix alapú rendszerek esetében, azzal a különbséggel, hogy az OpenPGP szabvány előjel nélküli 4 bájtos egészt rögzít, nem pedig előjeleset, így későbbi időpontok ábrázolása is lehetséges.[1]

– *adathossz leírás*: 1, 2 vagy 5 bájtos számérték, olyan kóddal, hogy a kisebb értékekhez rövidebb kód tartozik, egy hossz (pozitív egész) reprezentálása:

  • 0 és 191 között 1 bájtként,

  • 192 és 8383 között 2 bájtként, az 1. bájt kötelezően nagyobb, mint 191, a következő képlettel: $256(\texttt{bájt1} - 192) + \texttt{bájt2} + 192$,

  • 8384 és 4294967295 között 5 bájtként, az 1. bájt kötelezően 255, a következő képlettel: $256(256(256(\texttt{bájt2}) + \texttt{bájt3}) + \texttt{bájt4}) + + \texttt{bájt5}$ történik.

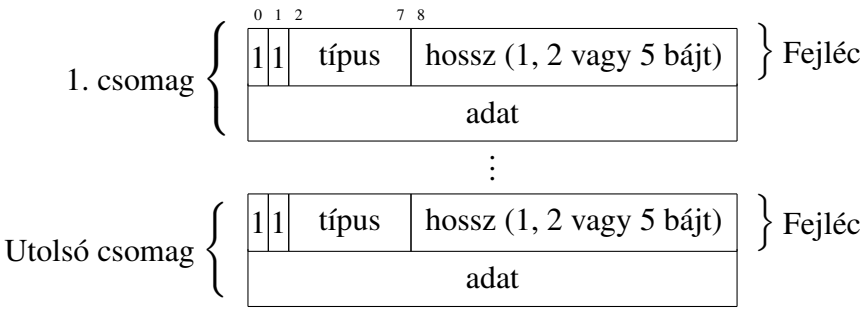
  • Lehetőség van hosszabb adat esetén az adat és hossz szinkronizált, folyamatos közlésére, azonban erre a SecMS esetében sosincs szükség.

4.1.2. Az OpenPGP csomagok szerkezete

Minden OpenPGP csomag a fejlécében leírja a típusát 6 biten, valamint az adattartalom (csomagtörzs) hosszát. Mivel a csomag törzse előtt ismert a hossza, ezért több csomag egyszerű egymásután írással összefűzhető, egy érdektelen csomag a megadott hossz szerint átugorható.

A diplomamunkában felhasznált OpenPGP csomagtípusok:

[1] Az előjeles egészek használatával 2038. januárjában, míg az előjel nélküliek használatával 2106. februárjában lesznek problémák.



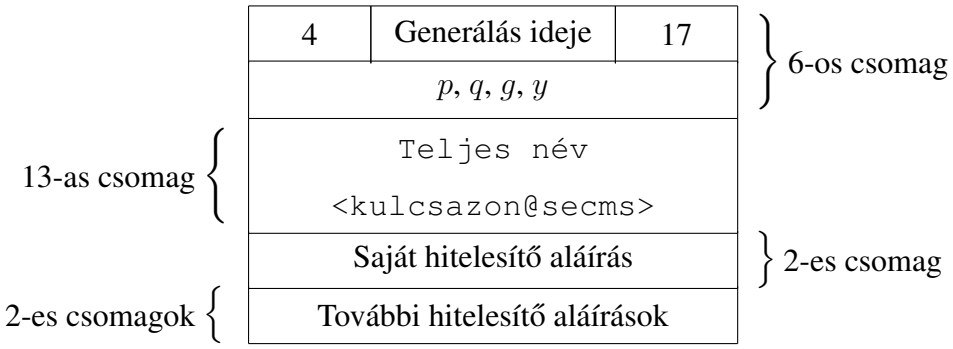
4.1. ábra. OpenPGP csomagformátum

1	Titkos–nyilvános kulcsú rejtjelezés inicializációja
2	Titkos–nyilvános kulcsú digitális aláírás
6	Nyilvános kulcs
11	Egyszerű, nyílt szöveg
13	Felhasználói azonosító
18	Szimmetrikusan rejtjelezett, integritás védett adatcsomag
19	Integritás védelem a 18-as csomaghoz
60	SecMS utasítások
61	Új felhasználó regisztrációja a SecMS-be
62	Jelenlegi felhasználó „bejelentkezése” a SecMS-be
63	SecMS szerver nyilvános kulcsának letöltése

Bemutatom, hogyan lehet ezekkel a csomagtípusokkal a szükséges adatokat, illetve (a hálózaton átküldendő) üzenetváltásokat reprezentálni. Az ábrákban a csomagok adat tartalmát szemléltetem, természetesen minden csomagot megelőz a fejléce.

4.1.3. DSA nyilvános kulcsok ábrázolása

A SecMS-ben minden felhasználó rendelkezik egy DSA kulccsal, amit titokban tart és segítségével tetszőleges bájt sorozatot hitelesíteni tud.



4.2. ábra. DSA kulcs leírása OpenPGP csomagokkal

A SecMS-ben kötelezően használt p, q, g értékeket [26] rögzíti, a Diffie-Hellman kulcsmegegyezés, amit felhasználunk, csak egyező értékek mellett működik.

Bármilyen nyilvános kulcs, így a DSA kulcsok leírása is a 6-os azonosítójú csomaggal történik. Ebben a csomagban kell leírni a p, q, g, y értékeket. Valamint itt meg kell adni a generálás idejét másodperc felbontású dátumként. A 4-es érték a kulcs verziószáma, minden új szoftvernek 4-es verziójú kulcsokat ajánlott létrehoznia, a 17-es érték pedig azt jelzi, hogy ez egy DSA kulcs.

Pazarlásnak tűnhet az, hogy a sok különböző felhasználó mindegyikéhez közöljük a p, q, g értékeket, pedig azok változatlanok és nyilvános szabványban, a SecMS részeként specifikáltak lesznek. Dönthetnénk úgy is, hogy bevezetünk egy új kulcstípust, amely esetében a p, q, g értékek rögzítettek és csak az y változhat. A 17-es érték helyett egy új értéket választanánk a 100-110-es tartományból, amely a kísérleti, illetve privát felhasználásra van fenntartva. Azonban ezzel a spórolással megakadályoznánk, hogy meglévő OpenPGP implementációk megértsék a felhasználóink kulcsait (és számukra például titkosított üzeneteket küldjünk, vagy az általunk kiállított digitális aláírásokat ők ellenőrizzék). A legtöbb jövőbeni SecMS felhasználónak jelenleg nincs PGP kulcsa, nem használ OpenPGP technológiát, a kompatibilitás megtartásával biztosítjuk, hogy a SecMS miatt létrehozott kulcsát

később használhassa más célokra is.

A 6-os csomag után a 13-as csomag tartalmazza a felhasználó nevét és email címét, majd ennek a névnek és címnek a kulcshoz való hozzárendelését hitelesítő aláírások következnek. Ezek között szerepelnie kell egy saját hitelesítő aláírásnak (ún. self signature-nek), az ezzel nem rendelkező kulcsokat az implementációk elutasítják.

4.1.4. Aláírások ábrázolása

Az OpenPGP 4-es verziójától kezdve a digitális aláírások formátuma megengedi, hogy azokhoz előre meghatározott típusú metaadatokat (pl. készítés időpontja, lejárat időpontja, aláírás visszavonásának az oka), illetve saját metaadatokat csatoljunk (név-érték párokként). Az aláírás ilyen metaadatokat kétféleképpen tartalmazhat. Amennyiben az aláíráshoz hashelt, akkor az aláírás sikeres ellenőrzése a metaadat hitelességét is jelenti, amennyiben ahhoz nem hashelt, akkor nem. Hashelt metaadat például az aláírás létrehozási ideje, míg nem hashelt az aláíró személy kulcsazonosítója. Minden metaadat egy külön alcsomagban foglal helyet. Az alcsomagok a csomagokhoz hasonlóan leírják a saját méretüket, így azokból több egyszerűen egymásután írható. Az alcsomagok pontos formátumát a 4.3. ábra mutatja. Típusukat egy bájt határozza meg, az általunk felhasznált típusok:

2	az aláírás készítésének ideje másodperc felbontású dátumként (self signature esetén egyezik a kulcs készítésének idejével)
3	az aláírás lejárata, a létrehozástól számított másodpercekként 4 bájton, (ha ez a csomag hiányzik vagy az érték nulla, akkor sosem jár le)
16	az aláírás kibocsátójának 8 bájtos kulcsazonosítója
20	tetszőleges név–érték pár (notation data) (ld. 4.4. ábra)

A DSA digitális aláírást az aláírandó üzenet és a hashelt metaadatok SHA-1 értékére számoljuk. Amennyiben a 2.1.2. alszakaszban ismertetett kulcskezelő

Hátralévő hossz (1, 2 vagy 5 bájt)	típus	} Fejléc
adat		

4.3. ábra. Aláírási alcsomagok formátuma

aláírásról van szó, akkor az üzenet az aláírandó kulcs és felhasználói név. Természetesen ennél sokkal precízebben specifikálni kell a hash függvény argumentumának a felépítését, hiszen a digitális aláírás ellenőrzése csak akkor lehet sikeres, ha ez a specifikáció félreérthetetlen. A szabvány pontosan specifikálja is a hash függvény bemenetét, a specifikációt később a 4.6. ábra segítségével összefoglaljuk.

A digitális aláírást és a hozzá tartozó metaadatokat egy 2-es azonosítójú csomag tartalmazza (ld. 4.5. ábra), szerepel benne az aláírás verziója (4), az aláírás típusa (t, 0 bináris üzenet, 1 CRLF sörvégű szöveges üzenet[2], illetve 16 és 19 közötti érték kulcs-felhasználónév hozzárendelés esetén a megbízhatóság fokától függően), a használt digitális aláírás algoritmusának azonosítója (17 DSA esetén), a használt hash függvény azonosítója (2 SHA-1 esetén), a hashelt, majd a nem hashelt metaadatok hossza és tartalma, a kiszámolt hash érték első két bájtja, majd a hash érték digitális aláírása (DSA esetén r, s) előjel nélküli nagy számok

[2] Azaz olyan szöveges üzenet, amiben a sorvégeket az ASCII táblázat 10-es és 13-as karakterének egymásutánja jelzi.

Jellemzők*	0	0	0
Név hossza 2 bájton		Érték hossza 2 bájton	
Név**		Érték	

* 1 bájt: 128, ha az érték UTF-8 kódolású szöveg, egyébként 0.

** `valami@valahol.com` alakú UTF-8 kódolású szöveg az ütközések elkerülésére.

4.4. ábra. Név–érték pár ábrázolása a 20-as alcsomagban

4	t	17	2	
Hashelt csomagok hossza (2 bájt)				(α)
Hashelt csomagok				
Nem hashelt csomagok hossza (2 bájt)				
Nem hashelt csomagok				
Hash érték első két bájtja				
r, s; mindkettő MPI-ként				

4.5. ábra. DSA digitális aláírás az OpenPGP-ben (2-es csomag)

sorozataként. A kétféle alcsomagok előtti méret megadása valóban szükséges, hiszen maguk a csomagok saját hosszukat leírják, de ebből nem lehet következtetni az összes csomag hosszára.

Feleslegesnek tűnik a hash érték első két bájtjának közlése előre, hiszen a digitális aláírás úgyis pont az egész értéket (nem csak az első két bájtját) hitelesíti majd. Azért szól így a szabvány, mert az aszimmetrikus kriptográfiai műveletek (és így egy nyilvános kulcsú aláírás ellenőrzése is) drágák. Hálózati hiba, sérült adathordozó vagy bármilyen más adathiba esetén ez a két bájt már nagyon nagy valószínűséggel eltér és így az aláírás olcsón visszautasítható. Természetesen ennek az értéknek a helyes volta semmilyen biztonságot nem nyújt, hiszen ha egy támadó meg tudja változtatni az aláírt adatot, akkor nyilván ezt a hash értéket is újraszámolhatja és felülírhatja.

4.1.5. Rejtjelezett üzenet ábrázolása

Mind szimmetrikusan rejtjelezett, mind (akár több) nyilvános kulcs számára kódolt üzenetek ábrázolását támogatja az OpenPGP. A rejtjelzendő adathoz (ami tipikusan egy vagy több másik OpenPGP csomag, pl. tömörített vagy nyílt szöveg) először hozzá kell fűzni az integritás védelmet szolgáló 19-es csomagot, majd az

egész struktúrát kell kódolni, így keletkezik a 18-as csomag. A kódolás mikéntjét címzettenként egy külön csomaggal kell jelezni a 18-as előtt. Nyilvános kulcsú kriptográfia esetén erre az 1-es csomag való, szimmetrikus esetben a 3-as. Ezek a csomagok tartalmaznak minden szükséges adatot (algoritmus azonosítók, kódolt session kulcs), ami a címzett titkos kulcsán, illetve szimmetrikus esetben a jelszón kívül szükséges az üzenet megfejtéséhez.

Az OpenPGP szabványba a mi általunk használni kívánt RC4 kódolás jelenleg nincs beillesztve, a SecMS-ben nem használunk semmilyen jelenleg definiált rejtjelezést az OpenPGP-ből, ezért azoknak az egyébként is bonyolult részleteit nem ismertetem. Bemutatom viszont, hogy hogyan integrálta kutatócsoportunk az RC4-et a szabványba. Az itt bemutatott specifikációt (vagy ahhoz nagyon hasonló megoldást) fogunk javasolni az OpenPGP következő verziójába való elfogadásra a megfelelő fórumokon. Erre szükség van, ugyanis az OpenPGP jelenleg nem kínál alternatívát olyan kis számításkapacitású környezeteknek, mint a mobiltelefonok, annak ellenére, hogy a készülékek gyorsan terjednek, fejlődnek és a programozhatóságuk javul.

Üzenet*		
(α) a 4.5. ábrából		
4	255	(α) hossza, 4 bájton

* Szöveges üzenet esetén ($t = 1$) CRLF sorvéggel hashelendő az üzenet. Kulcshitelesítéskor ($t \in [16,19]$) üzenetként a 4.7. ábrán szemléltetett bájtok hashelendők.

4.6. ábra. A SHA-1 hash függvény bemenete aláíráskor

153	6-os csomag hossza 2 bájton	6-os csomag törzse
180	13-as csomag hossza 4 bájton	13-as csomag törzse

4.7. ábra. Kulcs és felhasználói név kanonikus ábrázolása

4.2. SecMS kiegészítések az OpenPGP-hez

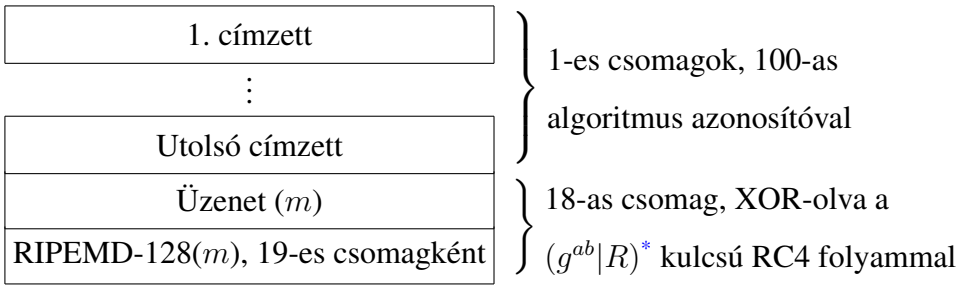
A SecMS megvalósításához két szempontból kellett kiegészíteni az OpenPGP-t. Az üzenetek rejtjelezésére és integritás védelmére olcsóbb, minimalista eszközöket (RC4, RIPEMD-128) illesztettünk az OpenPGP-be, másrészt a kliens–szerver paradigma megvalósításához specifikáltunk egy kiegészítést.

Ezenkívül a SecMS mostani megvalósításában lehetséges a szerverre kulcsokat feltölteni (új felhasználó regisztrálásához), illetve a szervertől le lehet kérdezni a saját nyilvános kulcsát. Ezeket a kiegészítéseket azonban nem ismertetem, mivel ezek átmenetiek, a későbbi verziók nem fogják támogatni. A szerver kulcsát ugyanis sokkal egyszerűbb és biztonságosabb a kliensprogramba ágyazni és így annak telepítése után már rendelkezésre áll, az új kulcsok szerverre való feltöltésére pedig már van kulcskezelő protokoll[48], amit az interoperabilitás érdekében amúgy is támogatnunk kell.

4.2.1. Üzenetek rejtjelezése és integritás védelme

Az OpenPGP rejtjelezésére építve, minden titkosított üzenetet 3 csomag együttese reprezentál (ld. 4.8. ábra), a 18-as tartalmazza a rejtjelezett levelet, a beleágyazott 19-es az integritás védelmet biztosító hash értéket, a 18-ast megelőző egy vagy több 1-es (a címzettek számától függően) a rejtjelezés inicializáló adatait.

Az alapján lehet felismerni a SecMS speciális, RC4-et használó titkosított csomagjait, hogy az 1-es csomagokban a privát felhasználásra fenntartott 100-as érték szerepel, mint algoritmusazonosító. Új levél küldése esetén az 1-es csomag köz-



* Több címzett esetén a $(g^{ab}|R)$ értékkel csak egy sorsolt session kulcs van titkosítva az 1-es csomagokban és itt az a véletlen érték a rejtjelező kulcs.

4.8. ábra. Üzenetek rejtjelezése

li az üzenet címzettjét is, a szerver a címzett levelesládájába lementi a titkosított üzenetet a feladó kulcsazonosítójával, illetve a feladó–szerver kommunikációjából kiszámítható R értékkel együtt.

A 18-as csomag rejtjelezett részének kódolása RC4-el történik, a RIPEMD-128$(g^{ab}|R)$ kulccsal (ahol a és b a kommunikáló felek titkos kulcsait jelölik, ld. 1.3. szakasz), az integritás védelmet a szabványtól eltérően (hatékonysági megfontolásokból) a 19-es csomagban nem a SHA1, hanem szintén a RIPEMD-128 hash függvény biztosítja. Fontos megjegyezni, hogy ez a módszer valóban rendkívül takarékos. A g^{ab} érték minden olyan üzenetben azonos, ahol ugyanaz a két személy kommunikál, így ez az érték a kliensprogram által megjegyezhető, nem szükséges mindig újra és újra kiszámolni. A hálózati forgalommal is takarékos ez a módszer, a feladó kulcsazonosítóján és az R értéken kívül semmi más nem szükséges a címzettnél az üzenet megfejtéséhez. Ugyanakkor a kódoló kulcs minden üzenethez teljesen más, az R (lényegében) véletlen érték és a RIPEMD függvény miatt. Az RC4 folyam első 256 bájtját nem használjuk. Mindezen megfontolásokkal elkerüljük, hogy protokollunk az 1.7. szakaszban ismertetett biztonsági hiányosságok bármelyikével rendelkezzen.

4.2.2. Kliens–szerver üzenetváltás

Rejtjelezve, authentikálva és az integritást védve kell, hogy a kliensprogramok és a SecMS szerver üzeneteket váltson. Ezt a három követelményt könnyedén meg lehet oldani az előbb ismertetett módon, csak most arról van szó, hogy az egyik szereplő (a) a kliensprogram, a másik (i) pedig a szerver. Így a Diffie-Hellman kulcsmegegyezésben RIPEMD-128($g^{ai}|R$) titkok alakulnak ki, amiket a szerver is ki tud számolni, ugyanis az R értéket kapcsolatonként szekvenciálisan növeli a kliensprogram (a kezdőértékben együtt egyeznek meg), a g^a nyilvános kulcshoz tartozó azonosítót ($K_{64}(a)$) pedig a levél címzettjének a helyén megadja a kapcsolatkezdeményező kliens.

A rejtjelezett adattartalom kliens–szerver üzenet esetén nem levél, hanem levélkezelő parancsok[3] (postaláda listázása, letöltése, új üzenet küldése, levelek törlése, mappák közti mozgatása, stb.), amiknek a formátuma is specifikált, nem ismertetett kriptográfiai (vagy egyéb érdekes) problémát azonban ez a specifikáció nem tartalmaz, csupán olyan jellegű kérés–válasz párok formalizálása, mint „14-es számú üzenet letöltése”–„14-es üzenet”.

4.3. Az üzenetek belső formátuma

Nem esett még szó arról, hogy a mindenhol hivatkozott m karaktersorozat formátuma milyen. Ennek specifikálása is fontos, hiszen különben a kliensprogramok nem tudnak együtt működni.

A SecMS-ben az üzenetek nyílt tartalma egyszerű szöveges üzenet esetén UTF-8[61] kódolású Unicode szöveg, bonyolultabb esetben egy tetszőleges MIME üzenet. Az üzenettől elválasztható, bizonyító erejű aláírások készítése így

[3] Fontos, hogy több parancs is küldhető egy csomagban, ugyanis így a forgalomanalizálás ellen is valamekkora védelmet nyújthat egy ügyes megvalósítás.

SecMS-ben is az S/MIME előírásai szerint működik a felhasználó DSA kulcsával. Habár ez drágább, mint a hamarosan bemutatott rejtjelezés és integritásvédelem, cserébe jóval ritkábban is van rá szükség. Akár MIME, akár szöveges üzenetről van szó, az üzenet törzsét megelőzhetik fejlécek, a [10]-ben látott módon:

```
Fejléc1: tartalom1
Fejléc2: tartalom2

Törzs (MIME vagy UTF-8)
```

A mobil környezetben való takarékosság és egyszerűség érdekében a sorvégeket a SecMS üzeneteiben a 10-es ASCII kódú karakter egyedül jelzi, a fejlécekben tetszőleges UTF-8 karakterlánc megengedett.

5. fejezet

ePoint

A 3.2. szakaszban bemutattam nagy vonalakban az ePoint működését. Most áttekintem, hogy pontosan milyen tranzakciók is vonatkoznak az S_i ígérvényekre, illetve a lehetséges kriptográfiai kihívások fajtáira, majd a kereskedés módjára és a technikai megvalósítás részleteire is kitérek.

5.1. Tranzakciótípusok

Az N_i tranzakciós kérést sikeres feldolgozás esetén a szerver mindig elmenti a létrejövő nyilvános S_i ígérvény részeként, mint indoklást. A kérés a következők valamelyike lehet:

- $\mathcal{E}$: kibocsátás kérvényezése. Ez az üzenettípus kötelezően tartalmazza az érték és kihívás digitális aláírását valamely kibocsátásra jogosult személytől;
- $\mathcal{X}$: csere. Az ilyen kérvény tartalmaz egy korábban már nyilvánosságra hozott S_j-hez ($j < i$) tartozó kihíváshoz (C_j) egy megfejtést és egy új kihívást (C_i);
- $\mathcal{M}$: két kibocsátott ePoint összevonása. Ennek a kérésnek tartalmaznia kell

egy érvényes kihíváshoz egy választ (R_j, illetve egy érvényes ígérvény azonosítóját (k), úgy, hogy $k, j < i$. A kérés feldolgozása után a j-hez tartozó V_j értékkel az S_k-hoz tartozó V_k-t a kibocsátó megnöveli és az így létrejött ígérvény lesz az S_i. S_i nyilvánosságra hozása után sem az S_k, sem az S_j nem érvényes többé;

– $\mathcal{S}$: ePoint felváltása. Az N_i üzenet ekkor egy régi j indexű kihívásra a választ tartalmazza, valamint két új kihívást és az új S_i-hez tartozó $V_i < V_j$ értéket. A szerver két új ígérvényt bocsát ki, S_i-t V_i értékkel és az első új kihívással, illetve S_{i+1}-et a $V_j - V_i$ értékkel és a második új kihívással.

– $\mathcal{I}$: visszavonási kérelem. Ez az üzenettípus csak egy választ tartalmaz valamilyen korábbi S_j ígérvényben lévő C_j kihívásra. A tranzakció jelentése az, hogy az S_j-ben ígért tartozás a fizetési rendszeren kívül (például készpénzzel) teljesítésre került.

Természetesen az összes üzenettípus esetében a szervernek ellenőriznie kell pár biztonsági feltételt, ilyen pl. az új kihívások egyedisége, hogy azokat korábban nem használták fel.

Az így bonyolított tranzakciók a felsorolt adataikkal teljes egészében nyilvánosak. Ebből következik a kibocsátó ellenőrizhetősége. Ugyanis a sorszámokat szigorúan monoton növekvő sorrendben kell kiadnia és ellenőrzésképp bármikor végezhetünk egy tranzakciót az aktuális sorszám ellenőrzésére. Továbbá az összes tranzakció indoklás része hivatkozik korábbi ígérvény(ek)re vagy a kibocsátás tényére, ezért régi tranzakció, illetve kibocsátás nem tagadható le. A jelenleg forgalomban lévő pénz mennyisége (a kibocsátó fennálló összes tartozása) egyszerűen számolható: »»»»> .r1627

$$V = \sum_{i:N_i \in \mathcal{E}} V_i - \sum_{i:N_i \in \mathcal{I}} V_{N_i.j}$$

5.2. Kriptográfiai kihívások

Az elmondottak működéséhez szükséges, hogy nagy mennyiségben tudjunk olyan matematikai feladatokat generálni, amelyeknek generálása és a megoldás ellenőrzése gyors, de csak a feladatból a megoldás előállítása reménytelenül lassú.

Minden ilyen kihívás egy olyan C feladat, amihez tartozó R válasz bizonyítja, hogy a válaszadó birtokában van a D titoknak, ami alapján a C készült.

Több lehetséges kihívást is támogat a rendszer, hogy a felhasználó a neki megfelelő (a védett érték és a számítási környezet szerinti) biztonságú megoldást választhassa.

5.2.1. Hash értékhez tartozó őskép

Kihívás: $C = h(D)$.

Titok: D, véletlen bájt sorozat, a C ősképe h szerint.

Megfejtésként adott válasz: $R = D$, elfogadható, ha $h(R) = C$.

A h valamilyen hash függvény, pl. a SHA-1.

Az előnye ennek az egyszerű kihívásnak, hogy nagyon gyorsan generálható, viszonylag kicsi titkokat eredményez (kb. akkorákat, mint a SHA-1 képe, azaz 160 bit), amik akár szűk csatornákon is átvihetők (beszéd, billentyűzet, vonalkód). Így az ilyen jellegű kihívásokkal könnyen megvalósíthatóak a naiv tranzakciók. Mivel a hash értéke a titoknak nagyon gyorsan kiszámolható, magát az ígérvényt nem is muszáj feltétlenül a titokkal együtt tárolnunk és átadnunk, az mindig lekérdezhető a nyilvános adatbázisból.

Hátrány, hogy a válasz pontosan egyezik a titokkal, így nem megbízható csatornán nem használható, illetve a kibocsátó csalási kísérlete ellen sem véd.

5.2.2. Nyilvános kulcshoz tartozó digitális aláírás

Kihívás: $C = K$, egy aszimmetrikus nyilvános kulcs.

Titok: $D = K'$, a K-hoz tartozó titkos kulcs.

Megfejtésként adott válasz: $R = \sigma_K(N')$, a végrehajtandó N' tranzakciós üzenet K-val aláírt példánya.

Bármilyen digitális aláírási séma megfelel (pl. a DSA vagy az RSA), a kulcsot minden kihíváshoz újonnan kell egy elegendően nagy halmazból véletlenszerűen választani, hogy egy támadó azt ne tudja kitalálni.

A nyilvánvaló hátránya ennek a módszernek a rendkívül magas számítási és tárolási költsége. Egy nyilvános–titkos kulcspár generálása még számítógépen is lassú, a létrejövő kulcs mérete is jóval nagyobb, mint egy hash függvény ősképe.

Előnye ezeknek a kihívástípusnak, hogy a válasz akár nem megbízható csatornán is továbbítható, valamint a kibocsátó ellen is véd, mivel a D titok később (még a tranzakció feldolgozása után) sem kerül felfedésre, így az aláírt N' üzenet módosítása senki számára nem lehetséges.

5.2.3. Adott hashű nyilvános kulcshoz tartozó aláírás

Kihívás: $C = h(K)$, egy aszimmetrikus nyilvános kulcs hash értéke.

Titok: $D = (K, K')$, a titkos–nyilvános kulcspár.

Megfejtésként adott válasz: $R = (\sigma_K(N'), K)$, a végrehajtandó N' tranzakciós üzenet K-val aláírt példánya és a K kulcs.

Ez a kihívás az előzőnek a módosítása annyival, hogy a szerveren tárolandó kihívás pontosan ugyanúgy néz ki, mint az első esetben. Ez azzal az előnnyel jár, hogy a nyilvános kulcs nem áll rendelkezésére az esetleges támadónak, így elég kisebb nyilvános kulcsokat is használni. Egyáltalán, az esetleges támadó (illetve a kibocsátó) azt sem tudhatja egészen a felhasználás pillanatáig, hogy az ilyen

kihívású ePoint valóban ilyen, és nem egy egyszerű véletlen karakterlánc hash értéke, mint az első esetben.

5.2.4. Rejtjelezett nyilvános kulcshoz tartozó aláírás

Kihívás: $C = (K, \rho_D(K'))$, egy aszimmetrikus nyilvános kulcs és a titkos kulcs D-vel rejtjelezve.
Titok: D, egy véletlenül választott rejtjelező kulcs.
Megfejtésként adott válasz: $R = \sigma_K(N')$, a végrehajtandó N' tranzakciós üzenet K-val aláírt példánya.

Ez egy másik módosítása a nyilvános kulcsos kihívásnak. Azt a problémát oldja meg, hogy a pénz birtokosának ne kelljen nagy titkokat tárolnia és küldenie. Ugyanis a titkos kulcsot maga a szerver tárolja, de rejtjelezett formában. Csak az fér hozzá a titkos kulcshoz és így az ePointhoz, aki a D szimmetrikus kulccsal tisztában van. A D már elegendő, ha kb. 100 bites. Ugyanakkor mivel a D-ből nem lehet következtetni az ePoint indexére, annak a tárolása is szükséges. A két érték együtt kb. akkora, mint az első kihívástípus esetében, így ez a megoldás is használható az ott felsorolt csatornákkal akár naivan is.

5.3. A kereskedés módjai

Az ePoint rendszerben nem csak a megfelelő kihívás megválasztása során vehetik figyelembe a felhasználók az aktuális biztonsági meggondolásaikat, illetve környezetüket.

Ugyanis a kihívás megválasztásakor csak az ePoint értékét tudják, azonban egy tranzakció során legalább ilyen fontos a használt csatorna, a másik fél személye, korábbi ismeretségük, az üzlet tárgya. Ezektől is függővé tehetik, hogy miként is kereskednek.

A két módszer közül az első a teljes bizalomra épít, hasonlatos egy üdítő automatához, ahol egyik fél sem tud reklamálni később, a második pedig a bolti vásárláshoz, ahol bizonyító erejű számla készül.

5.3.1. Előre fizetés

Tegyük fel, hogy Alice szeretne Bobnak 10 ePointot adni. Feltesszük azt is, hogy van köztük valami olyan biztonságú csatorna, amit az adott érték mellett elfogadhatónak tartanak.

Alice keres a pénztárcájában egy vagy több olyan titkot, ami(k) legalább 10 ePointot ér(nek). Amennyiben csak csupa kisebbet talál, azokat össze tudja vonni a $\mathcal{M}$ használatával egy nagyobbá. Ezután, vagy ha csak nagyobb értékűt talál, akkor az $\mathcal{S}$ jelű üzenettípussal ki tud vágni pontosan 10 ePointot.

A létrejött 10 ePoint értékű titkot (D) elküldi Bob részére, aki az $\mathcal{X}$ üzenettípussal becseréli egy saját maga által generált kihívású ePointra.

A tranzakció anoniman zajlott le, Alice és Bob csak egymással kommunikált, a kibocsátónak egyiküknek sem kellett megmondania a másik kilétét.

5.3.2. Számlás fizetés

A szereplők és az összeg legyen változatlan, azonban most feltesszük, hogy Alice nem szeretné kitenni magát annak a veszélynek, hogy Bob az ePointok beváltása után letagadja az üzletet és ne teljesítse a vállalását.

Ekkor először egyeztetik az árat és a terméket Bobbal és Bob erről kiállít egy digitálisan aláírt számlát, amibe beleír egy általa választott kihívást is (természetesen a kihívásnak csak a nyilvános részét, C-t).

Alice a számla és a rajta lévő digitális aláírás ellenőrzése után bevált egy 10 ePointost az $\mathcal{X}$ típusú üzenettel olyanra, aminek a kihívása a Bob által adott C.

Ebben a pillanatban Alice kezében nyilvánosan ellenőrizhető, akár bíróság által feldolgozható bizonyíték van arra, hogy Bobbal megegyezett és a saját vállalását, a fizetést teljesítette. Természetesen ezt a bizonyítékot harmadik fél előtt csak akkor kell felfednie, ha vita kerekedik. Az esetek túlnyomó többségében az üzlet rendben lezajlik és a tranzakció anonim marad.

Fontos még észrevenni, hogy a kibocsátó által üzemeltetett ePoint szerver számára a két féle fizetés nem különböztethető meg, ő mindkét esetben egy $\mathcal{X}$ kéréssel találkozott, azonban egyik esetben a fizető fél küldte a kérést, másik esetben az eladó fél.

5.4. Technikai megvalósítás

Schuller elkészítette diplomamunkaként[46] egy egyszerű, HTTP(S) alapú megvalósítását a kibocsátó szerverein futó szoftvernek. A diplomamunkájában pontosan specifikálja, hogy ezzel a szerverrel miként kell a fent leírt tranzakciókat HTTP(S) felületen végrehajtatni, miként lehet a nyilvános adatbázist lekérdezni.

A HTTP(S) alapú megvalósítás egyik nagy előnye, hogy egy egyszerű böngészővel meg lehet nézni egy-egy adott sorszámú ePointot.

A SecMS-sel való integrálás során a SecMS kliensről feltételezzük, hogy rendelkezik egy ePoint pénztárcával és abba a Schuller által ismertetett módon pénzt tud elhelyezni a szerver segítségével, illetve abból pénzt tud költeni.

6. fejezet

A SecMS és az ePoint integrációja

A két részletesen bemutatott rendszer integrációjától két problémára várunk megoldást: egyrészt szeretnénk, ha az említett ePoint kereskedési módok megvalósulhatnának a SecMS használatával, másrészt szeretnénk, ha az ePoint előre fizetési rendszerével meg lehetne védeni a SecMS üzenőrendszert még elterjedése közben a spam ellen, örökre kiküszöbölve ennek a problémának a jövőbeni felmerülését.

6.1. ePoint előre fizetése a SecMS-ben

Ezt a feladatot a lehető legegyszerűbben szeretnénk megoldani, mivel minden egyes SecMS üzenetnek tartalmaznia kell majd egy minimális előre fizetést, ami a levél fogadó oldali elolvasásának a ráfordításait (idő, fáradság) fedezi.

A 4.3. szakaszban látott módon lehetséges tetszőlegesen sok fejlécet csatolni egy üzenethez. Minden egyes átküldendő ePoint bankjegyet egy ilyen fejlécben küldünk el, több is lehet egy levélben. Schuller munkájában csak a hash függvény ősképére vonatkozó kriptográfiai kihívás van megvalósítva, így ebben a specifikációban is elegendő lehetőséget biztosítani az ősképátküldésére.

Tehát minden egyes átutalandó ePointhoz a fejléc formája legyen a következő:

`EPoint-Forward-Payment: <B64(RAND)>`

Itt a `RAND` egy véletlen érték, a korábban D-nek hívott titoknak felel meg, `B64` pedig a Base64 függvény.

A fejlécek a rejtjelezett üzenet részei, tehát a fizetéseket senki más nem tudja beváltani, csak a címzett.

6.2. Számlák formátuma, csatolása üzenetekhez

Számlák küldésére már jóval ritkábban van szükség, így azt egy új MIME típusként specifikálom. Ezzel az az előny is jár, hogy az itt leírt specifikáció később tetszőleges más MIME környezetben (web, email) is használható lesz számlák továbbításához.

A számla kötelezően a `vnd.epointsystem.invoice` MIME típussal rendelkezik. A `vnd` névtér használata természetesen átmeneti megoldás, a rendszer széles elterjedése esetén kutatócsoportunknak célja elkészíteni és beadni a szükséges specifikációkat ahhoz, hogy szabványos `text/plain+epinvoice` jellegű típussal rendelkezhessünk.

A számla csak akkor érvényes, ha hozzá egy S/MIME aláírás is tartozik. A számla kibocsátója az aláírókulcs tulajdonosa, így azt a számlában külön nem kell specifikálni.

Minden számlának van egy kötelező sorszáma, ami a kibocsátóra vonatkozóan egyedi és folyamatosan, egyesével növekvő kell, hogy legyen. A számla tartalmazza a kibocsátó által generált új kihívást is, amire a fizetés elvégezhető. Amennyiben valakinek sok számlát kell kiállítania, hasznos lehet, ha a kihívásokhoz tartozó titkokat nem véletlenszerűen választja, hanem valamilyen titokból, a sorszámból és a dátumból generálja hash függvénnyel. Ezzel elérhető, hogy a még nem fizetett számlák után nem kell titkokat tárolni. Természetesen a beérkezett fi-

zetések titkait azonnal teljesen véletlenszerűen generált titkokra kell cserélni a $\mathcal{X}$ üzenettípussal, hogy ne egy jelszó védjen folyamatosan növekvő értéket.

A számla egy egyszerű szövegfájl, melyben minden sor végét az ASCII tábla 10-es karaktere jelzi. A számla végét egyetlen olyan sor mutatja, amiben csak a ~ karakter szerepel (ASCII 126). A számlában szereplő kötelező mezők:

```
Serial: <sorszám>
Challenge: <B64(MD)>
Value: <érték ePoint egységben>
Due: YYYY-MM-DD HH:MM:SS
Expiry: YYYY-MM-DD HH:MM:SS
Subject:
.
.
.
~
```

Itt a `MD` (a korábban C-nek hívott érték) egy `RAND`-hoz tartozó hash érték.

A számla `Due` mezője az esedékességet, a `Expiry` a lejáratot határozza meg. A `Subject` mezőben írható le akár több sorban is a termék, amelyről a számla szól.

Valamely implementáció a saját mezőit (ami például utal a számlázószoftver készítőjére, elérhetőségeire) a `Subject` elé szúrhatja be, tetszőleges név–érték párokat rendelhetnek össze ezek a privát mezők is, azonban mindegyik nevének az `X-` prefixszel kell kezdődnie. Semelyik implementáció nem építhet ilyen mezők meglétére, minden olyan számla kezelése kötelező, ami a fent említett állandó mezők mindegyikét tartalmazza.

6.3. Levélszemét elleni védekezés

Beérkezett levél további feldolgozását, listázását csak akkor tegye bármilyen SecMS implementáció, ha legalább annyi előre fizetett ePoint beváltása sikerült a levél fejléceiből, amennyi a küldőhöz a felhasználó által beállított díj. Olyan küldő esetén, amelyhez nincs díj beállítva az alapértelmezett díjnak megfelelő ePointok beváltása szükséges.

Az üzenet megmutatásakor a felhasználó számára jelezni kell, ha a beállított díjnál többet küldött a feladó.

6.3.1. Az alapértelmezett díj beállítása

Ez az érték szabályozza, hogy egy adott kulcsazonosítójú SecMS felhasználó mennyi ePoint csatolását követeli meg egy levél elolvasásához.

Ez az alapértelmezett díj a SecMS szerveren bárki számára lekérdezhető, az adott felhasználó számára pedig írható is.

A SecMS rendszer üzemeltetője határozza meg és állítja be azt az alapértelmezett értéket, ami az újonnan létrehozott felhasználókra vonatkozik.

6.3.2. Kivételek kezelése

Amennyiben sokat levelezünk valakivel, kényelmesebb lehet az állandó ePoint küldés helyett beállítani, hogy tőle nem vagy csak kevesebb ePointot kérünk levelekért. Akkor is szükséges lehet ez, ha feliratkozunk pl. egy levelezőlistára, hiszen a SecMS-en belül az ilyen szolgáltatások nyújtói nyilván nem fognak még fizetni is azért, hogy azokat igénybe vegyük.

Amikor a felhasználó beállít egy valaki másra vonatkozó kivételt a saját SecMS felületén, akkor ezt a partnerével az implementációnak egy külön speciális üzenetben kell közölnie. A kivételt közlő üzenet egyetlen fejlécet tartalmaz:

`EPoint-Required-Payment: v, YYYY-MM-DD HH:MM:SS`

A lejárati dátummal az implementációnak óvatosan kell bánnia. A jövőben távol lévő dátumot csak olyan környezetek állítsanak be, ahol garantálható, hogy az üzenet elküldéséről nem feledkeznek el. Pl. elveszthető mobiltelefon esetén, ha a felhasználónak van mentése a SecMS titkos kulcsából, de a leveleiből, illetve a beállításokból már nincs, akkor az új telefon megvásárlása és beállítása után a korábban engedélyezett baráti körétől a leveleket nem fogja megkapni, ha túl hosszú lejárati időt állít be. Hiszen azok a telefonok továbbra is csökkentett ePoint értékkel (vagy $v = 0$ esetén ePoint nélkül) fogják küldeni a SecMS üzeneteiket, ami az új telefonon már kéretlen levélnek számít.

Éppen ezért kötelező az implementációnak titkos kulcs mentésből való visszaállítása után a szerveren a küldött leveleket végig néznie, ilyen speciális kivétel közlő üzeneteket keresve és azokat feldolgozni. A lejárati dátum egy „biztonsági öv” arra az esetre, amikor ez az eljárás valamilyen hiba miatt nem talál meg beállításokat.

7. fejezet

Az ePoint egyéb felhasználási területei

Az itt leírt problémák megoldása nem csak azért fontos, mert önmagukban is érdekesek és megoldatlanságuk jelentős hátránnyal jár napjainkban, hanem azért is, mert mindegyik égető annyira, hogy ePoint alapú megoldásukkal az ePoint elterjedése segíthető.

7.1. Telekommunikációs szolgáltatások

Számos IP alapú új kommunikációs megoldás versenyzik. Ezek többnyire jóval fejlettebbek, mint a hagyományos vonalas-, illetve mobiltelefonok. Például többségükben megoldott a videótelefonálás, szöveges üzenetek gyors váltása. Ezek használata a rendszeren belül jellemzően díjmentes, mivel ilyen felhasználáskor az amúgy már kifizetett internetszolgáltatáson kívül alig használnak más erőforrást. Ilyen telefonszolgáltatás pl. a Skype.

Ugyanakkor természetes igény, hogy ezekből a rendszerekből is el kívánunk érni hagyományos telefonszámokat. Ez nem valósulhat meg díjfizetés nélkül, hi-

szen a felhasznált csatornák (frekvenciák, telefonállomások) fenntartása és létesítése költségekkel jár.

Ezek a költségek nagyon aprók, nem mérhetőek egy bankkártyás fizetés vagy átutalás költségeihez. Ezért jelenleg csak úgy érhetőek el a modern IP alapú megoldásokból a régi hálózatok, ha a felhasználók jelentős mértékű (több száz órányi beszélgetésre elegendő) befizetést eszközölnek egyszerre. Ezzel elkötelezik magukat hosszú távra egy szolgáltató mellett.

Az ePointtal az ilyen szolgáltatások úgy biztosíthatóak, hogy csak a telefonálás pillanatában kell kifizetni a díját. Ezzel a szereplők közti verseny jelentősen nőhet és az IP alapú megoldások piaca jelentősen kiszélesedhet, hiszen olyan felhasználók is kipróbálhatják, akiknek eddig nem volt bizalmuk a több száz órás költség előre utalásához.

Ugyanez a probléma tapasztalható az internetet lokálisan (egy-egy háztömbben vagy utcában), nagyon olcsón szolgáltatni tudó, jellemzően vezeték nélküli megoldásokat alkalmazó kis szolgáltatók esetén. Őket az ePoint a számlázás jelentős adminisztrációs költségeitől is megszabadíthatja.

7.2. Tartalomszolgáltatás

Jelenleg az elektronikus tartalomszolgáltatás nagy részét reklámok bevételéből fedezik. Ez több szempontból is szuboptimális, jelentősen rontja a web használatának élményét, a reklámok általában technikailag a legszörnyűbb, a hordozó weblaptól elütő megoldásokat használnak. A tartalomszolgáltatóknak pedig jóval kisebb bevételt biztosítanak, mint amennyiért a tartalmat a piacon eladhatnák a felhasználóik számára.

Ráadásul a felhasználók egy nem elhanyagolható (és egyre növekvő) része megpróbálja elkerülni a reklámokat. Különböző szoftverek segítik őket ebben, a

reklámozók pedig válaszul ezen szoftverek hibáit keresik. Egyre sűrűbben jelenik meg olyan nyilvános felhívás is, ami azokat a programozókat tünteti fel rossz színben, akik ebben a versenyben kényszerűségből (és/vagy megbízásra) részt vesznek.

Természetes folyamat ez, hiszen az internet egy interaktív médium. Minden résztvevő definíció szerint saját maga döntheti el, hogy melyik tartalomra és annak melyik részére kíváncsi és melyikre nem. Hasonlóan ahhoz, ahogy egy újságban a reklám átlapozható. Éppen ezért a nívós folyóiratok előállítása reklámbevételekből nem fedezhető, azokért az olvasóknak fizetnie kell.

Az ePoint által szolgáltatott mikrofizetésekkel könnyen kidolgozható a webhez egy olyan kiterjesztés, amivel a fizetést igénylő weblapok biztonságosan és egyszerűen megvalósíthatóak és használhatóak lesznek.

7.3. Adományok, támogatások kezelése

Sok interneten elérhető szolgáltatásért jelenleg nem kötelező (sőt sok esetben nem is lehetséges) fizetni. Ilyenek például a levelezőlistákon és más fórumokon elérhető segítségek, a szabadszoftverekkel kapcsolatos egy felhasználóra jutó fejlesztések.

Általában a szolgáltatás nyújtója reklámmal nem kívánja rondítani a terméket, így (néha elég sikeresen) az adományokra hagyatkozik. Mivel ezeknek a mértéke kicsi, ezek célba juttatásában is jelentős szerepe lehet az ePointnak.

IV. rész

Elért eredmények

Diplomamunkámban

- áttekintettem ismert üzenetküldő- és fizetőrendszerek biztonsági megoldásait;
- ismertettem az ehhez szükséges kriptográfiai hátteret;
- részletesen bemutattam mindkét problémára egy-egy újszerű, a kutatócsoportunkban kidolgozott megoldást,
- amelyek integrálásával megoldási javaslatot adtam a levélszemét problémájának újszerű, gazdasági alapokon nyugvó megoldására;
- végül vázlatosan felsoroltam három érdekes probléma ePoint alapú megoldását.

Köszönetnyilvánítás

Köszönetet mondok témavezetőmnek, Nagy Dánielnek, aki bármikor rendelkezésemre állt a diplomamunka készítése során, bizalommal fordulhattam hozzá kérdésekkel.

Köszönettel tartozom még Sziklai Péternek, az ELTECrypt kutatócsoport vezetőjének, Bárász Mihálynak és Ligeti Péternek, a kutatócsoport két kutatójának. Rengeteget segítettek ők technikai és adminisztrációs problémák megoldásában, illetve az általuk tartott szemináriumok jelentősen szélesítették látókörömet a témában.

Irodalomjegyzék

[1] Dan Boneh: Twenty years of attacks on the RSA cryptosystem. 46. vol. (1999) 2. sz., *Notices of the American Mathematical Society (AMS)*, 203–213. p.

[2] Nikita Borisov – Ian Goldberg – Eric Brewer: Off-the-record communication, or, why not to use PGP. In *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society* (konferenciaanyag). New York, NY, USA, 2004, ACM, 77–84. p. ISBN 1-58113-968-3.

[3] Stefan A. Brands: An efficient off-line electronic cash system based on the representation problem. In *246*. ISSN 0169-118X, 1993. 31, Centrum voor Wiskunde en Informatica (CWI), 77. p.

[4] Buttyán Levente – Vajda István: *Kriptográfia és alkalmazásai*. Budapest, 2005, Typotex. ISBN 9639548138.

[5] J. Callas – L. Donnerhacke – H. Finney – D. Shaw – R. Thayer: *OpenPGP Message Format (RFC 4880)*. 2007. november, IETF. http://www.ietf.org/rfc/rfc4880.txt. (2008. február 6.).

[6] Jan L. Camenisch – Jean-Marc Piveteau – Markus A. Stadler: Blind signatures based on the discrete logarithm problem. 950. vol. (1995), *Lecture Notes in Computer Science*, 428–432. p.

[7] D. Chaum: Blind signatures for untraceable payments. In *CRYPTO '82* (konferenciaanyag). New York, 1983, Plenum Press, 199–203. p.

[8] D. Chaum – A. Fiat – M. Naor: Untraceable electronic cash (extended abstract). In *CRYPTO '88* (konferenciaanyag). 1989, 319–327. p.

[9] Don Coppersmith: Small solutions to polynomial equations, and low exponent rsa vulnerabilities. 10. vol. (1997) 4. sz., *J. Cryptology*, 233–260. p.

[10] D. Crocker: *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. 1982. augusztus, IETF. http://www.ietf.org/rfc/rfc822.txt. (2008. március 13.).

[11] T. Dierks – C. Allen: *The TLS Protocol Version 1.0*. 1999. január, IETF. http://www.ietf.org/rfc/rfc2246.txt. (2008. április 24.).

[12] Whitfield Diffie – Martin E. Hellman: New directions in cryptography. IT-22. vol. (1976) 6. sz., *IEEE Transactions on Information Theory*, 644–654. p.

[13] M. Elkins – D. Del Torto – R. Levien – T. Roessler: *MIME Security with OpenPGP*. 2001. augusztus, IETF. http://www.ietf.org/rfc/rfc3156.txt. (2008. május 18.).

[14] Scott Fluhrer – Itsik Mantin – Adi Shamir: Weaknesses in the key scheduling algorithm of RC4. 2259. vol. (2001), *Lecture Notes in Computer Science*, 1–24. p.

[15] N. Freed – N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*. 1996. november, IETF. http://www.ietf.org/rfc/rfc2049.txt. (2008. április 16.).

[16] N. Freed – N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies (RFC 2045)*. 1996. november, IETF. http://www.ietf.org/rfc/rfc2045.txt. (2008. február 18.).

[17] N. Freed – N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. 1996. november, IETF.
http://www.ietf.org/rfc/rfc2046.txt. (2008. április 16.).

[18] N. Freed – J. Klensin – J. Postel: *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. 1996. november, IETF.
http://www.ietf.org/rfc/rfc2048.txt. (2008. április 16.).

[19] David D. Friedman: Future imperfect.
http://www.daviddfriedman.com/Future_Imperfect.html.
(2008. május 20).

[20] Taher El Gamal: A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology* (konferenciaanyag). New York, NY, USA, 1985, Springer-Verlag New York, Inc., 10–18. p. ISBN 0-387-15658-5.

[21] Ian Grigg: Pareto-secure: A definition of security using the theory of Pareto Efficiency.
http://iang.org/papers/pareto-secure.html. (2008. április 27.).

[22] Johan Hastad: On using rsa with low exponent in a public key network. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85* (konferenciaanyag). New York, NY, USA, 1986, Springer-Verlag New York, Inc., 403–408. p. ISBN 0-387-16463-4.

[23] J. Jonsson – B. Kaliski: *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 (RFC 3447)*. 2003. február, IETF.
http://www.ietf.org/rfc/rfc3447.txt. (2008. február 8.).

[24] Andreas Klein: Attacks on the RC4 stream cipher. 2008., *Designs, Codes and Cryptography*.

[25] D. Knuth: *The Art of Computer Programming, Vol. 2, Semi-Numerical Algorithms*. 1969, Addison-Wesley, 398–422. p.

[26] Ligeti Péter – Németh Zsolt Balázs – Riskó Gergely: SecMS protocol description. http://www.sourceforge.net/projects/secms. (2008. február 11.).

[27] Stefan Lucks: Attacking hash functions by poisoned messages. http://www.cits.rub.de/MD5Collisions/. (2008. január 17.).

[28] N. Mavrogiannopoulos: *Using OpenPGP Keys for Transport Layer Security (TLS) Authentication*. 2007. november, IETF. http://www.ietf.org/rfc/rfc5081.txt. (2008. május 2.).

[29] Ralph C. Merkle: Secure communications over insecure channels. 21. vol. (1978) 4. sz., *Commun. ACM*, 294–299. p.

[30] K. Moore: *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. 1996. november, IETF. http://www.ietf.org/rfc/rfc2047.txt. (2008. április 16.).

[31] Nagy Dániel: On digital cash-like payment systems. In *ICETE* (konferenciaanyag). 2005, 66–73. p.

[32] National Institute of Standards and Technology: *FIPS PUB 186: Digital Signature Standard*. 1994, U.S., Department of Commerce.

[33] M. O. Rabin: A probabilistic algorithm for testing primality. 12. vol. (1980), *Journal of Number Theory*, 128–138. p.

[34] B. Ramsdell: *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling*. 2004. július, IETF. http://www.ietf.org/rfc/rfc3850.txt. (2008. április 16.).

[35] B. Ramsdell: *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*. 2004. július, IETF. http://www.ietf.org/rfc/rfc3851.txt. (2008. április 16.).

[36] Riskó Gergely: SecMS üzenetküldő kliensprogram. szakdolgozat (Eötvös Loránd Tudományegyetem). Budapest, 2007.

[37] R. L. Rivest – A. Shamir – L. M. Adelman: A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS. MIT/LCS/TM-82. Jelentés, 1977, MIT.

[38] P. Saint-Andre: *End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)*. 2004. október, IETF. http://www.ietf.org/rfc/rfc3923.txt. (2008. május 18.).

[39] P. Saint-Andre: *Extensible Messaging and Presence Protocol (XMPP): Core*. 2004. október, IETF. http://www.ietf.org/rfc/rfc3920.txt. (2008. május 18.).

[40] P. Saint-Andre: *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. 2004. október, IETF. http://www.ietf.org/rfc/rfc3921.txt. (2008. május 18.).

[41] P. Saint-Andre: *Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)*. 2004. október, IETF. http://www.ietf.org/rfc/rfc3922.txt. (2008. május 18.).

[42] Oliver Schirokauer – Damian Weber – Thomas F. Denny: Discrete logarithms: The effectiveness of the index calculus method. In *ANTS* (konferenciaanyag). 1996, 337–361. p.

[43] Bruce Schneier: 1933 anti-spam doorbell. http://www.schneier.com/blog/archives/2007/05/1933_antispam_d.html. internetes naplóbejegyzés, 2007. május 10.

[44] Bruce Schneier: *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, NY, USA, 1993, John Wiley & Sons, Inc. ISBN 0471597562.

[45] Claus P. Schnorr: Efficient identification and signatures for smart cards. In *CRYPTO '89: Proceedings on Advances in cryptology* (konferenciaanyag). New York, NY, USA, 1989, Springer-Verlag New York, Inc., 239–252. p. ISBN 0-387-97317-6.

[46] Janis Schuller: Designing and implementing a system for digital cash. diplomamunka (Universität Bremen). Bremen, 2007.

[47] A. Shamir: RSA for paranoids. 1. vol. (1995), *CryptoBytes*, 1–4. p.

[48] D. Shaw – Jabberwocky Tech: *The OpenPGP HTTP Keyserver Protocol (HKP)*. 2003. március.

[49] R. Solovay – V. Strassen: A fast monte-carlo test for primality. 6. vol. (1977) 1. sz., *SIAM Journal on Computing*, 84–85. p.

[50] Marc Stevens: Fast collision attack on MD5. 2006., *Cryptology ePrint Archive, Report 2006/104*.

[51] Xiaoyun Wang – Dengguo Feng – Xuejia Lai – Hongbo Yu: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. 2004., *Cryptology ePrint Archive, Report 2004/199*.

[52] Xiaoyun Wang – Yiqun L. Yin – Hongbo Yu: Finding collisions in the full SHA-1. 3621. vol. (2005. November), *Lecture Notes in Computer Science*, 17–36. p.

[53] Michael J. Wiener: Cryptanalysis of short rsa secret exponents. In *EUROCRYPT '89: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology* (konferenciaanyag). New York, NY, USA, 1990, Springer-Verlag New York, Inc., 372. p. ISBN 3-540-53433-4.

[54] Wikipedia: Cryptographic hash function.
http://en.wikipedia.org/wiki/Cryptographic_hash. (2008. január 26.).

[55] Wikipedia: Cyclic Redundancy Check.
http://en.wikipedia.org/wiki/Cyclic_Redundancy_Check.
(2008. március 16.).

[56] Wikipedia: Digital Signature Algorithm.
http://en.wikipedia.org/wiki/Digital_Signature_
Algorithm. (2008. január 4.).

[57] Wikipedia: RC4.
http://en.wikipedia.org/wiki/RC4. (2008. március 6.).

[58] Wikipedia: Schnorr group.
http://en.wikipedia.org/wiki/Schnorr_group. (2008. február 18.).

[59] Wikipedia: Spam (electronic).
http://en.wikipedia.org/wiki/Spam_(electronic). (2008. január 8.).

[60] Wikipedia: Stream cipher.
http://en.wikipedia.org/wiki/Stream_cipher. (2008. február 28.).

[61] F. Yergeau: *UTF-8, a transformation format of ISO 10646 (RFC 3629)*. 2003. november, IETF.
http://www.ietf.org/rfc/rfc3629.txt. (2008. február 16.).