SecMS üzenetküldő kliensprogram

Szakdolgozat

Készítette: Riskó Gergely
Programtervező matematikus szak, nappali tagozat

Témavezető: Nagy Dániel, ELTECRYPT kutatócsoport

Eötvös Loránd Tudományegyetem, Informatikai Kar, 2007.

Tartalomjegyzék

1. fejezet

A SecMS bemutatása

A SecMS használatával lehetőség nyílik, mint annyi más informatikai rendszerrel, üzenetek váltására. A lényegi különbség az eddig elkészített üzenetküldő rendszerek és a SecMS között, hogy a SecMS esetében az ELTECRYPT kutatócsoport előre meghatározta a — feltételezett felhasználási területnek megfelelő — biztonsági követelményeket, amiket a rendszer algoritmusainak kidolgozása során kriptográfiai eszközök felhasználásával el is ért.

A szakdolgozat a már kidolgozott matematikai algoritmus „feldolgozásáról" és megvalósításáról szól. Azért van szó feldolgozásról is, mert a cikk, amiből a szakdolgozat építkezik — mint semmilyen ilyen jellegű cikk — nem tárgyalta a számítástechnikai megvalósítás részleteit, meglévő (és felhasználható) megoldások ajánlását, értékelését.

Ebben a fejezetben ismertetésre kerül a rendszer felépítése és az általa biztosított funkcionalitás, a biztonsági követelmények és az azokat megvalósító kriptográfiai eljárások. Ez utóbbi rész megértése nem feltétlenül szükséges a program használatához, ugyanakkor a felhasználó olyan ismereteket szerezhet, ha feldolgozza, amelyek segítenek abban, hogy miközben a SecMS-t használja, biztonságban érezze magát és a saját biztonságáról gondoskodni tudjon. Ezért nem lett volna helyes ezt teljes egészében a fejlesztői dokumentációt tartalmazó fejezetben leírni. Természetesen az algoritmusok megértése a programkód megértéséhez és módosításához elengedhetetlen. A fejezet zárásaként összehasonlítjuk a SecMS funkcióit és biztonsági követelményeit más — elterjedt — üzenetküldő megoldásokéval.

1.1. Komponensek

A levelek feladásához és fogadásához szükség van a számítógépen futó programon túl egy központi számítógépre (szerverre) is, ami az üzeneteket tárol-

ja. Mint később látni fogjuk, ebben a szerverben (más rendszerektől eltérő módon) nem kell nagyon erősen bíznunk, az ő csalási lehetőségei nem igazán fenyegetőek. A demonstrációk kedvéért a kutatócsoport egy referencia szerverimplementációt is csinált, de a jövőben a SecMS elterjedése esetén harmadik fél által készített szerver és kliens implementációk megjelenése is várható, ezek kompatibilitását a közös hálózati protokoll biztosítja.

Természetesen szükséges, hogy a kliensszámítógép és a szerverszámítógép között legyen hálózati kapcsolat. A szerver használatához a kliensnek el kell tudnia érni a szerver TCP portját (alapértelmezés szerint: 8080). A kliens és a szerver közötti minden párbeszéd a hálózat szempontjából HTTP (web) forgalomnak tűnik, ilyen módon van csomagolva. Ez pazarlásnak tűnhet, azonban így az elérhető hálózati kapcsolatok köre (a proxyszámítógépeknek köszönhetően) sokkal nagyobb.

A kliens nem csak számítógép lehet, hanem pl. programozható mobiltelefon is. Az ELTECRYPT kutatócsoport későbbi munkájának fontos eleme is lesz, hogy ezzel a rendszerrel számítógép és mobiltelefon felhasználók problémamentesen és platformfüggetlenül kommunikálhatnak.

A szakdolgozatomban megvalósított kliens implementációjakor ügyeltem arra, hogy az elkészülő program egyaránt futtatható legyen Java alkalmazásként és böngészőbe ágyazódó Java Appletként is. Az utóbbi lehetőség felhasználásával olyan felhasználók is tudják használni a rendszert, akik programok telepítésével nem kívánnak foglalkozni vagy arra nem jogosultak; a Java Appletet bárki elérheti egy korszerű böngészővel.

1.2. Funkciók a jelenlegi változatban

1.2.1. A szerver nyilvános kulcsának letöltése

A program a bejelentkezés vagy regisztráció elején — tehát a rendszer minden egyes indításakor — automatikusan letölti és feldolgozza a szerver nyilvános kulcsát, ami szükséges a további kommunikációhoz.

1.2.2. Regisztráció

Amennyiben új felhasználó kívánja igénybe venni a SecMS rendszer szolgáltatásait, szüksége van egy (bizonyos szabályoknak megfelelő) titkos–nyilvános kulcspárra. A regisztráció folyamata ezen kulcspár nyilvános részének feltöltése a szerverre, amelynek során egy tetszőleges ún. megjelenítési név is megadható, hogy mások könnyebben összeköthessék ezt a kulcsot a tulajdonos személyével.

A kulcspárt az ismertetett SecMS kliens egy a felhasználó által adott felhasználónév–jelszó párból állítja elő. Az előállítás algoritmusa determinisztikus és dokumentált, tehát ugyanaz a név és jelszó hordozható különböző számítógépek, illetve implementációk között. Cserébe a felhasználónév és jelszó illetéktelen kezekbe kerülése vagy elvesztése esetén nincs lehetőség azok megváltoztatására, hanem a kulcs visszavonása szükséges. További hátrány, hogy fel kell hívni a felhasználók figyelmét (és a felhasználók tudomásul kell, hogy vegyék), hogy más rendszerektől eltérően, már a regisztráció pillanatában kellően erős jelszót kell választaniuk, mert annak későbbi megváltoztatása nem lehetséges, más kulcs használatára való áttérés pedig legalábbis nehézkes.

A nyilvános kulcsból a jelenleg elérhető számítási kapacitásokkal nincs lehetőség a titkos kulcs előállítására, illetve mindkét kulcs ismerete sem elegendő a kiindulási felhasználónév és jelszó meghatározásához. Ezért a rendszerben azonos felhasználónevű felhasználók előfordulhatnak, azonban az, hogy vannak-e ilyenek sosem válik ismertté (a rendszer üzemeltetője számára sem): a felhasználókat a kulcsuk nyilvános része, illetve annak lenyomata és a saját maguk által megadott megjelenítési név azonosítja.

Természetesen a szerver sem tudja eldönteni, hogy egy kliens nyilvános kulcsát felhasználónév–jelszó párból (és ha igen, milyenből) generálták-e vagy valamilyen más algoritmussal jött létre. Ezért teljesen legitim a SecMS rendszer egy későbbi, pl. mobiltelefonokon használt kliensprogramjában a kulcsot máshogy létrehozni (mondjuk a felhasználó által generált véletlen események alapján) és tárolni (pl. PIN kóddal védve). Azonban a fokozott biztonságnak (nevezetesen annak, hogy a kulcs a telefonból nem távolítható el) ára van: a felhasználó telefoncsere során vagy nem tudja megtartani a kulcsát vagy külön eljárásokra, esetleg szakember segítségére szorul.

Ha a szerveren már regisztrálva van a felhasználó, akkor a kliens hibaüzenetet jelenít meg.

1.2.3. Bejelentkezés

A „bejelentkezés” funkció lényegében nem lenne szükséges a SecMS-ben, mivel a megfelelő kulcs használatával a szerverrel való kommunikáció azonnal elkezdhető. Azért vezettük be mégis ezt a fogalmat, hogy az esetlegesen megadott rossz felhasználónév–jelszó pár ne gyűrűzzön tovább és ne akkor okozzon hibaüzenetet, amikor a felhasználó már megírt pl. egy 5 oldalas levelet. Továbbá a Java Appletként megjelenített webfelületen elérhető kliensprogram így egy szokásos üzenetküldő rendszer benyomását kelti, ezek mind a bejelentkezőképernyőn figyelmeztetnek hibás felhasználónév–jelszó pár megadása esetén.

Az érdekesség kedvéért jegyzem meg, hogy nem lenne szokatlan az informatika világában, ha a hálózattal (és tranzakciókkal) való takarékoskodás végett ezt a kényelmet nem nyújtanánk a felhasználónak, pl. a Magyarországon is elterjedt bankautomaták szinte mindegyike a megadott PIN kódot csak az első tranzakció feldolgozása közben ellenőrzi, a felhasználó csak jóval a PIN kód megadása után (pl. amikor már kétszer megadta a feltöltendő mobiltelefon számát és a feltöltés összegét vagy begépelte a felveendő készpénz összegét) tudja meg, hogy a párbeszéd legelején megadott kód hibás volt.

A mi esetünkben és a rendszer készítésekor már nem volt indokolt az ilyen mértékű takarékoskodás, így tehát a kliensprogram minden belépéskor ellenőrízteti a szerverrel a kulcsot és a regisztráció ellentéteként pont akkor ad hibaüzenetet, ha a kulcs nem létezik a szerver adatbázisában.

1.2.4. Üzenetek listázása és olvasása

A kliensprogram belépéskor letölti a szerverről az összes, a postafiókjában lévő üzenetet és ezeket a felhasználónak listában megjeleníti, amely listában a feladóra vagy a dátumra kattintva láthatóvá válik a megfejtett üzenet. Az üzenetre erről a képernyőről válaszolni is lehet.

A program az üzenetek listáját csak kérésre (vagy újra belépés esetén) bővíti a szerverre újonnan érkezettekkel.

1.2.5. Új üzenet írása

Tetszőleges felhasználónak, akitől már kaptunk üzenetet vagy akinek tudjuk a kulcsazonosítóját, írhatunk új levelet.

1.2.6. Nyomkövetés — kulcsok kezelése

A rendszer könnyebb demonstrációjáért és a nyomkövetés egyszerűsítésére a program egy menüpontjában lehetőség van az összes kezelt partnerkulcs listázására, illetve új kulcsok letöltésére, a nyilvános kulcslenyomatuk alapján.

1.3. Funkciók a jövőben

Számos funkció nem — sőt pár alapfunkció sem — került megvalósításra ebben a demonstrációs verzióban. Ezeket áttekintjük, hogy egyértelmű legyen, hogy ezek megvalósítása nem ütközik a SecMS működési elveivel, csupán időráfordítás kérdése a kidolgozásuk.

1.3.1. Üzenetek továbbítása

Szokásos funkció az üzenőrendszerekben, hogy lehetőség van levelek továbbítására, a jelenlegi kliensverzióban erre nincs külön felhasználói felület, a funkció a válaszadás gombbal, majd a címzett megváltoztatásával helyettesíthető.

A továbbítás funkció implementációja a későbbi verziókban azért is fontos, mert amikor már lehetőség lesz olyan üzenet küldésére, ami a hitelességet és integritást (digitális aláírással) harmadik fél számára is bizonyítja, akkor a kliens programnak ügyelnie kell arra, hogy a továbbítás után az aláírás ellenőrízhető maradjon.

1.3.2. Üzenetküldés több címzett számára

Az ELTECRYPT kutatócsoport kidolgozott egy külön üzenettípust arra az esetre, ha ugyanazt az üzenetet (pl. meghívó egy megbeszélésre) több, mint egy címzettnek kívánjuk egyszerre elküldeni. Ez az új üzenettípus hatékonyabb, mintha az összes címzettet külön-külön üzenettel értesítjük, kevesebb sávszélességre van szükség.

1.3.3. Üzenetek osztályozása

A következő verziókban lehetőség lesz a szerveren tárolt üzeneteink osztályozására is; az, hogy ez milyen biztonsági követelmények mellett valósítható meg (pl. ha a szerver sem látja a tulajdonságokat, akkor nem biztos, hogy tud indexelni) még kidolgozás alatt áll.

1.3.4. Üzenetek törlése

Amikor az üzenetek osztályozása funkció elkészül, természetesen egyik osztály lehet a törölt üzenetek osztálya, amiből való törlés után az adott felhasználó már valóban nem tudja majd elérni a szóban forgó üzenetet. Ha mind a címzett(ek), mind a feladó kitöröl egy levelet véglegesen, a rejtjelezett üzenet a szerverről is törölhető lesz.

1.3.5. Küldött üzenetek tárolása

A küldött üzenetek osztályának (a felhasználó szempontjából: mappájának) létrehozása szintén az osztályozáshoz kötődik.

1.3.6. Kézbesítési, olvasási visszaigazolások kezelése

Tekintve, hogy a SecMS majd fontos szerepet fog játszani elektronikus fizetések lebonyolításában, kiemelkedő fontosságú, hogy az üzenetek átvétele, illetve elolvasása a feladó számára követhető és ellenőrízhető legyen. A későbbi verziókban többféle visszaigazolás kérésére is lesz lehetőség (ld. később a biztonsági követelményeknél).

1.3.7. Különböző szerverek használata

Mivel lehetőséget szeretnénk biztosítani sokféle SecMS szerver- és kliensprogram létrehozására, szükséges megoldani, hogy a különböző kiszolgálószámítógépeken regisztrált felhasználók is küldhessenek üzeneteket egymásnak. Az az információ, hogy egy felhasználó melyik szervert használja a levelei fogadásához, megadható (és változtatható is) a nyilvános kulcsában ([OpenPGP, Section 5.2.3.16. Key server preferences]).

1.4. Biztonsági követelmények

Most ismertetjük azokat a célkitűzéseket és jelentésüket, amiket előre meghatároztunk a SecMS elkészítésekor. Ez a rész különösen fontos, hiszen ez adja a SecMS egyedi jellegét.

1.4.1. Integritás

Az üzenetek fogadásakor a címzett biztos lehet benne, hogy az üzenetet változatlan formában kapja meg. Erről saját maga meg tud győződni, harmadik félben nem szükséges megbíznia. Későbbi verziókban a feladó majd eldöntheti, hogy olyan üzenetet ad-e fel, ami azon túl, hogy a címzett számára garantálja az integritást, még lehetővé teszi, hogy a címzett ezt általa kiválasztott személyek számára be is bizonyítsa. A hétköznapi életben például ezzel az erővel bír egy postai átutalási megbízás a posta által lebélyegzett feladóvevénye, de nem bír ezzel az erővel egy internetes bankon adott átutalási megbízás kinyomtatott képe: hiszen azt a HTML kódot lementve és átírva tetszőleges megbízás nyomtatható. Jelenleg a lakossági körben nincs olyan szélesen elterjedt banki szolgáltatás, amivel megnyugtatóan igazolhatnák magánemberek az interneten feladott megbízásaikat (pl. lakástulajdonosnak a bérlője valamely közüzemi átutalást).

1.4.2. Hitelesség

A címzett biztos lehet a feladóban. Természetesen, mint mindegyik másik biztonsági követelmény, ez is csak megfelelő felhasználás esetén érvényes, a felhasználónak tisztában kell lennie a szükséges lépésekkel, azonban más üzenőrendszerekkel ellentétben, megfelelő jártassággal az üzenet hitelességét saját maga ellenőrízni tudja.

1.4.3. Konfidencialitás

A SecMS a rábízott leveleket a feladó számítógépén rejtjelezi és csak a címzett számítógépén fejti meg. A felhasznált rejtjelezés kizárja, hogy az üzenetet bárki más elolvashassa (pl. a szerver üzemeltetője, vagy az internetszolgáltató). Azon túl, hogy az üzeneteket csak a címzett és a feladó tudja megfejteni, a szerverrel való kommunikáció is rejtjelezve zajlik, így más műveletek (elküldött levelek újraolvasása, kulcsletöltés, visszaigazolások) is rejtve maradnak a passzív lehallgatással próbálkozó támadó elől.

1.4.4. A visszaigazolások megbízhatósága

A SecMS későbbi verzióiban lehetőség lesz különböző visszaigazolások kérésére. Ez különösen fontos, mivel pénzforgalmi megbízások kezelésére is fel szeretnénk majd használni.

A megvalósítandó visszaigazolástípusok:

- átvételkor: a szerver igazolja, hogy feladtunk egy bizonyos ellenőrzőösszeggel rendelkező levelet egy megadott címzett számára. Az üzenet felfedésével és a kapott hitelesített (ellenőrzőösszeg, feladó, címzett, dátum) négyes bemutatásával bizonyíthatjuk később a feladás tényén túl a feladott üzenet pontos tartalmát is.

- letöltéskor: lehetőség lesz arra, hogyha kértünk letöltéskori visszaigazolást, akkor a címzettnek ezt kötelező legyen elküldeni, a saját kulcsával hitelesítve (digitálisan aláírva), a szerver nem fogja engedni, hogy más műveletet végezzen (a levél ismételt letöltésén kívül), amíg a visszaigazolásról nem gondoskodott.

- olvasás után: ez a típus csupán informatív (hiszen a számítógép nem tudja ellenőrízni, hogy valamit valóban elolvastunk és megértettünke); a jelenlegi verzióban is elérhető pl. úgy, hogy ha tudatni akarjuk, hogy egy üzenetet megértettünk, akkor válaszolunk rá. Az automatizáció csak azt teszi majd lehetővé, hogy ezek az információk a vissza-

igazolást kérő személynél szebben jelenjenek meg, a visszaigazolást adó személy pedig könnyebben előállíthassa.

1.5. Kriptográfiai elemek

Az ismertetésre kerülő kriptográfiai elemek közül a folyamtitkosító és a hash-függvény konkrét megvalósítása kis programozói munkával másra kicserélhető mind a kliens-, mind a szerverprogramban. A Diffie-Hellman kulcsmegegyezés is helyettesíthető más primitívvel, azonban ez a felhasználóknál a szoftver frissítésén túl a DSA kulcsaik cseréjét is igényelheti, ami nagy számú felhasználónál már szinte megvalósíthatatlan.

1.5.1. A Diffie-Hellman kulcsmegegyezés

Az egész rendszerre vonatkozóan adott[1] egy Schnorr csoport rendje (160 bites prím: q), az 1024 bites p prím és az ezek alapján számolt g generátor elem, melyek megfelelnek a [FIPS 186-2] szabványban leírt kritériumoknak. A SecMS minden felhasználója rendelkezik egy 160 bites $0 < x < q$ titkos kulccsal (értsd: természetes számmal). A felhasználó nyilvános kulcsa a titkos kulcsából könnyen kiszámolható: $y = g^x \bmod p$. A másik két említett értékkel együtt (p-vel és q-val) a (p, q, g, y) rendezett négyes hozható létre, ami egy [OpenPGP] [DSA] kulcs, tehát a felhasználók a SecMS kulcsukat az OpenPGP rendszerben is használhatják digitális aláírások létrehozására. Ezt a lehetőséget kihasználva fogunk a későbbiekben harmadik fél számára is bizonyítható integritású és hitelű üzeneteket létrehozni.

Most tegyük fel, hogy Alice és Bob már a leírt módon létrehozták kulcsaikat, amik rendre: $A = g^a \bmod p$, illetve $B = g^b \bmod p$ és a szerverre fel is töltötték a nyilvános részt (tehát A-t és B-t). Ekkor mindketten könnyedén ki tudják számolni a titkos kulcsukból és a szerveren elérhető információkból az $A^b = (g^a)^b = (g^b)^a = B^a \pmod p$ értéket (a középső egyenlőséget a hatványozás kommutativitása biztosítja), ami azonban a nyilvánosan elérhető g^a és g^b számokból más számára csak túlságosan költségesen számolható ki. Ezért ez az érték közös titoknak tekinthető Alice és Bob között és az RC4 folyamtitkosító kulcsát ennek segítségével fogjuk kapni.

A szerver is rendelkezik DSA kulccsal és a vele való kommunikáció a leírt módon minden esetben titkosításra kerül, kivéve az első párbeszédet, amikor a szerverkulcshoz tartozó nyilvános értéket tölti le a kliensprogram.

[1] Az adott értékeket és generálásuk módját a protokoll dokumentáció tartalmazza.

1.5.2. Az RC4 folyamtitkosító

Az RC4[2] algoritmus lényege, hogy egy bizonyos kulccsal való inicializálás után pszeudóvéletlen bájtsorozatot generál. A bájtok létrehozásának költsége szinte csak a bájtok számával arányos.

Az így generált bájtsorozatot és a titkosítandó szöveget bájtonként a „kizáró vagy" (továbbiakban: XOR vagy összeadás) művelettel kombináljuk. A kapott rejtjelezett üzenet csak a kulcs ismeretével fejthető meg: újra elindítva az RC4 folyamot az előálló bájtokat kell a rejtjelezett üzenethez hozzáadni. Mivel a XOR művelet kétszeri egymásutáni alkalmazása azonos operandussal az identikus leképezést adja, amit kapunk az az eredeti üzenet.

Az RC4 folyamtitkosító széles körben elterjedt, egyszerűsége és gyorsasága miatt kedvelik. Azonban vigyázni kell használatakor, bizonyos szabályokat mindenképp be kell tartani, különben olyan könnyedén feltörhető rendszereket kapunk, mint amilyen (a szintén széles körben elterjedt) WEP.

Sosem szabad két különböző üzenetet azonos kulccsal rejtjelezni, hiszen ekkor a két rejtjelezett üzenetet összeadva megkapható az eredeti üzenetek összege, amiből már nagyon erős következtetéseket lehet levonni az üzenetekre. A SecMS ezt a hibát úgy kerüli el, hogy a Diffie-Hellman kulcsmegegyezés eredményeként kapott közös kulcshoz még egy véletlen értéket (esetenként egyesével növekvő értéket) is sorsol és ezt a két számot a következőkben ismertetendő hashfüggvénnyel alakítja kulccsá. Az így kapott értékek és a feladó–címzett pár között nincs kimutatható összefüggés, mivel a hashfüggvénytől elvárjuk, hogy jól szórjon.

Meg kell semmisíteni az inicializált RC4 folyam első 256 bájtját, mert bizonyított, hogy azon az intervallumon a pszeudóvéletlen kitétel nem teljesül, azaz az indokoltnál nagyobb összefüggés van a kulcs és a folyamként előálló bájtok között.[FluManSha]

1.5.3. Hashfüggvények

A hashfüggvények olyan egyirányú függvények, amik tetszőleges hosszú adatfolyamra alkalmazhatóak. Csak olyan függvényeket tekintünk kriptográfiai szempontból még alkalmazhatónak, amelyekkel kapcsolatban nem ismert (kivárható időn belül lefutó) algoritmus sem a megfordításukra, sem ütközés generálására.

A két használt hashfüggvény:

- SHA-1: 20 bájtos (160 bites) értéket előállító hashfüggvény, az OpenPGP szabvány előírja bizonyos szituációkban a használatát (pl. kulcs-

[2]Egyéb ismert nevek: ARC4 (**A**lleged **R**ivest **C**ipher 4), ARCFOUR

azonosító generálása nyilvános kulcs alapján). 2005-ban ismertté vált egy módszer, amivel ütköző sorozatok előállításának bonyolultsága 2^{80}-ról 2^{63}-ra csökkent. [WYY SHA1]

- RIPEMD-128: 16 bájtos (128 bites) értéket előállító hashfüggvény, a SecMS jelenlegi protokolljában ezt használjuk minden olyan helyen, ahol kompatibilitási okok nem indokolják 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 kliensszámítógép egy mobiltelefon. 2004 augusztusában publikáltak egy ütközést az eredeti RIPEMD hashfüggvényt illetően. [WFLY RIPE]

1.6. Összehasonlítás

1.6.1. Email

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

A felsorolt előnyökkel számos hátrány is együtt jár. Elterjedtsége leginkább annak köszönhető, hogy régóta létezik. Azonban az, hogy ilyen régi, azzal jár, hogy az eredeti szabványoknak egyáltalán nem volt célkitűzése egyik biztonsági követelményünk 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. Az utolsó biztonsági követelmény, a „visszaigazolások megbízhatósága” ráadásul nem is teljesíthető a szerverüzemeltetők közreműködése nélkül.

Bár nagy lépés a kéretlen levelek megakadályozásában, ha azonosíthatóak a feladók, nem gondoljuk, hogy csupán ezzel a segítséggel a probléma telje-

sen megoldható. A SecMS rendszer egy jól átgondolt 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 visszajuttatja.[DDFriFuIm, Slicing Spam] A szóba jövő fizetési rendszerek vizsgálatát a diplomamunkám témájának tűztem ki.

1.6.2. SMS

A mobiltelefonok között igen sokat használt SMS üzenetek (a valóban forgalmazott adatmennyiséghez képest) igen költségesek, biztonságosnak viszont nem mondhatók. Annak a fontosságát, hogy a SecMS használható legyen mobilkörnyezetben, ugyanakkor jól indokolja elterjedtségük, ami pontosan ennek a ténynek köszönhető.

Az SMS rendszerben feladott üzenetek rejtjelezése csak a mobilentitások 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, az érdekesség kedvéért megemlítem, hogy 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. Arra vonatkozóan nincs információ, hogy az adott üzenetet tudta is fogadni, illetve, hogy azt változatlan formában kapta meg (lévén az integritás nem garantált).

1.6.3. Jabber

A Jabber az [XMPP] protokollon alapuló azonnali üzenetküldő megoldás. Széles felhasználói táborral rendelkezik, számos program készült, amellyel ez a hálózat elérhető. Ezen 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 a [TLSv1] protokollal.

Mivel az XMPP könnyen kiterjeszthető, születtek megoldások, amelyekkel biztonsági követelményeink mindegyike megvalósítható, anélkül, hogy megbíznánk harmadik félben, ilyen például az [XMPP E2E].

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. Megjegyezzük, hogy a SecMS rendszerben, miután lehetőség is lesz aláírt üzenetek küldésére, a legtöbb üzenet hitelessége továbbra is csak a küldő, illetve a feladó számára lesz egyértelmű.

Született egy másik, nem szabványos megoldás is, mivel felismerték ezt a problémát, ez az „Off-the-Record Messaging" [OTR] nevet viseli. A használt kriptográfiai elemek nagyon hasonlóak a SecMS-ben felhasználtakkal, azonban itt 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. A SecMS ezt a követelményt nem teljesíti, de cserébe nem kívánja meg, hogy mindkét fél mindig egyszerre online legyen.

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

2. fejezet

Felhasználói dokumentáció

2.1. A SecMS indítása

Tekintve, hogy a kliensprogram mind alkalmazásként, tetszőleges szerverhez csatlakozva, mind Appletként, a weblap üzemeltetőjének szerveréhez csatlakozva is használható, leírjuk a kétféle működés eléréséhez szükséges indítási módot, majd a további opciókat már csak egyszer tárgyaljuk.

A program két nyelven is használható, a magyar és az angol nyelv között a nyitó képernyőn van lehetőség választásra. Demonstrációként az alábbi képernyőképeken egyszer angolul, egyszer magyarul „fényképeztem” le a programot, de a további tárgyalásban, ha valahol képernyőképre van szükség, annak nyelve (hasonlóan a szakdolgozatom nyelvéhez) magyar lesz.

A kliens minden funkciója elérhető kényelmesen a billentyűzetről is, ehhez a gyorsbillentyűket a megszokott módon aláhúzással jelöli a program. A gyorsbillentyűk nyelvenként különbözőek lehetnek. Minden funkció a jelölt gyorsbillentyűjével és az ALT gomb együttes megnyomásával érhető el, a vezérlők közötti fókuszváltásra a TAB, illetve (mivel levélküldésnél a levéltörzsben szerepelhet tabulátor) a Ctrl+TAB gombok használhatók, fókuszváltás visszafele is lehetséges, ha még a Shift gombot is nyomva tartjuk. A fókusszal rendelkező nyomógomb a Space gombbal aktiválható.

2.1.1. Alkalmazásként

Ahhoz, hogy a SecMS kliensprogramot alkalmazásként, közvetlenül a számítógépünkről futtathassuk, rendelkeznünk kell a Java programozási nyelv Sun által készített futtatókörnyezetének legalább 1.5-ös verziójával, amely ingyenesen letölthető[1]. Továbbá be kell szereznünk a SecMS kliensprogram Java

[1] http://java.sun.com/javase/downloads/index_jdk5.jsp

tömörítvényét (JAR)[2].

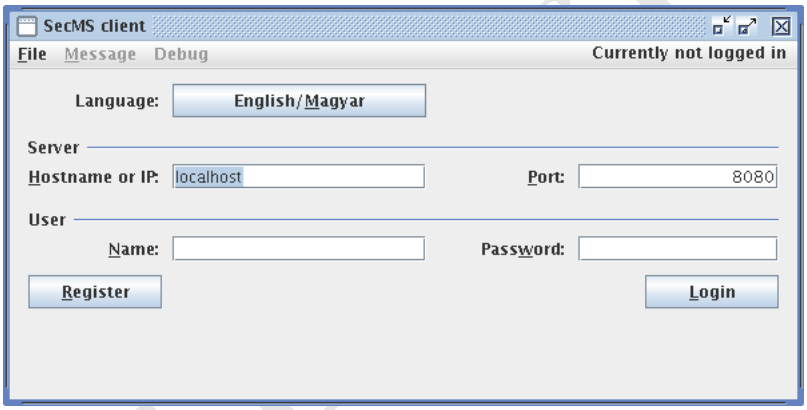
Ezekután, feltéve, hogy a `java` parancs elérhető és a `secms-client.jar` állomány az aktuális könyvtárban van, adjuk ki a

```
java -jar secms-client.jar
```

parancsot!

Az indulóképernyőhöz bármikor visszatérhetünk majd a későbbiekben is a fájl menü kijelentkezés pontját választva, illetve szintén bármikor kiléphetünk a programból, ha a fájl menü kilépés parancsát választjuk, vagy az éppen használt ablakkezelőnk segítségével bezárjuk a kliensprogram ablakát.

Amennyiben a képernyőnkön túl sok helyet foglal a kliensprogram, vagy éppen ellenkezőleg, a képernyőnkön jóval több helyet is rá tudnánk szánni, hogy kényelmesebben dolgozhassunk, nyugodtan átméretezhetjük az ablakot, a program kifejlesztése közben figyeltem arra, hogy ekkor a felhasználói felület elemei logikusan és konzisztensen változtassák méretüket.



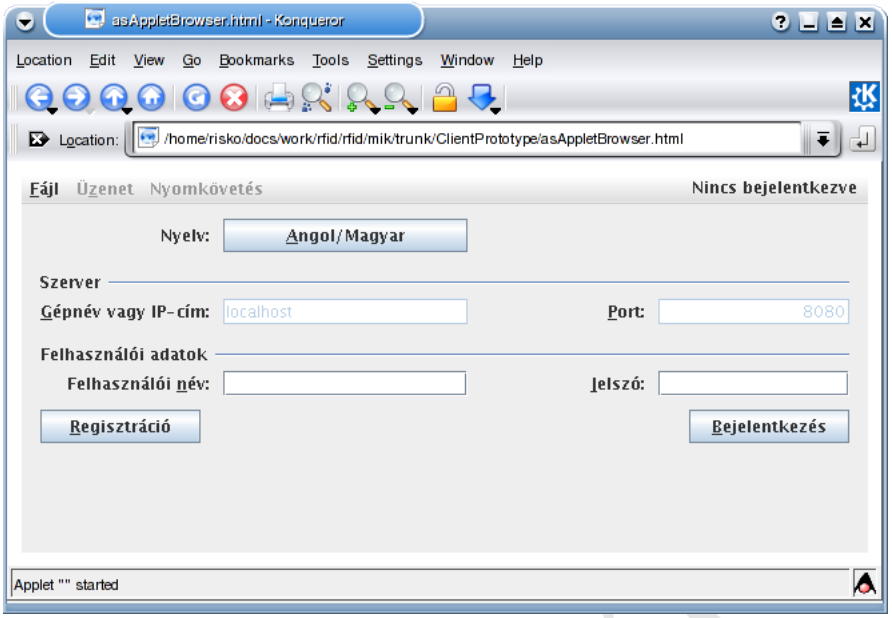
2.1. ábra. Alkalmazásként elindítva

2.1.2. Appletként

Amennyiben csak egy böngésző program áll a rendelkezésünkre, a SecMS rendszerünk szolgáltatója elérhetővé teheti a kliensalkalmazást a böngészőbe épülő programként is, ami így bármiféle telepítés nélkül futtatható.

A bejelentkezőképernyőhöz az alkalmazásként való indításhoz hasonlóan térhetünk vissza (fájl menü, kijelentkezés), azonban a kilépés menüpont most hiányzik, böngésző esetén ezt egészen egyszerűen az aktuális böngészési felület (ablak/fül) bezárása vagy a címsorba más címet írva a lap elhagyása jelenti.

[2] http://sourceforge.net/projects/secms



2.2. ábra. Appletként elindítva

2.2. Bejelentkezés

A bejelentkezés képernyőn hozzáférhetünk meglévő felhasználói nevünkkel a SecMS funkcióihoz vagy ha még nem rendelkezünk jogosultsággal, a regisztráció funkció segítségével létrehozhatunk egy újat.

A szerverszámítógép nevét és portját a szolgáltatónk tudja rendelkezésre bocsátani, illetve amennyiben böngészőt használunk a rendszer elérésére, akkor ezek az értékek a programba vannak táplálva és megváltoztatásuk nem lehetséges, de nem is szükséges.

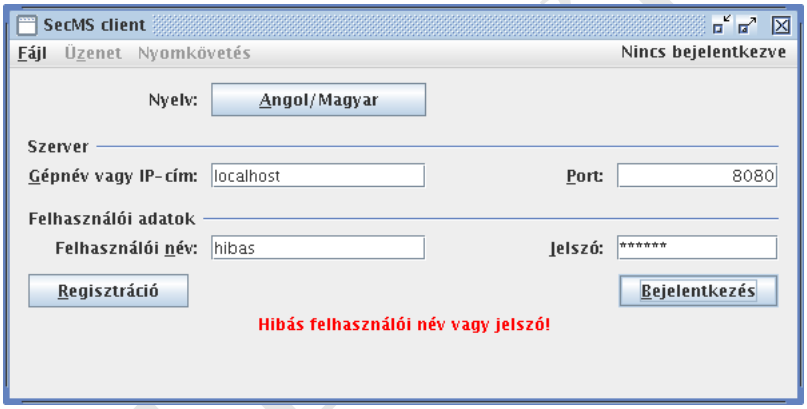
Bár a bevezetőben jeleztük, itt újra hangsúlyozzuk, hogy a jelszó utólagos megváltoztatására nincs lehetőség. Már a rendszer kipróbálásakor válasszunk kellően erős jelszót, ha elképzelhető, hogy a SecMS-t hosszútávon használni fogjuk!

Szintén fontos megérteni, hogy a felhasználói név és a jelszó nincs annyira erős módon megkülönböztetve, mint más rendszerekben, a felhasználói nevünk és a jelszavunk együtt játszik szerepet a titok kialakításában, a felhasználói nevünket, akár csak a jelszavunkat sosem kérik a rendszer üzemeltetői, illetve a felhasználói név megadása nem lehet szükséges senki számára. Nem szükséges, hogy ismerjék a felhasználói nevünket ahhoz, hogy üzenetet küldjenek nekünk, sőt, ha csak a felhasználói nevünket ismerik, azáltal nem tudnak azonosítani minket és így üzenetet sem küldhetnek. Partnereinknek megadni majd a későbbiekben ismeretetésre kerülő kulcsazonosítót (illetve

megjelenített nevet) kell!

A regisztráció során megadható megjelenített név szerepe, hogy ezt a nevet használhatják kliensprogramok egy adott felhasználó kulcsazonosítójának „becézésére”. Azonban mivel más felhasználók programjainak működésébe nem szólhatunk bele, könnyen lehet, hogy az általunk megadott nevet ő felülbírálja és a képernyőjén más (számára könnyebben kezelhető) névvel fog azonosítani minket. Ez az adat nem tekinthető hitelesnek, a jelenlegi verzióban a szerver bárkinek a megjelenített nevét megváltoztathatja. A hiányosságot könnyen orvosolja majd, amikor áttérünk az OpenPGP szabványnak megfelelő kulcskezelésre, mivel ott a felhasználói kulcshoz tartozó adatokat a kulcs által létrehozott digitális aláírás (ún. self-signature) hitelesíti.

Amennyiben a bejelentkezés funkciót választjuk, de eltévesztjük a felhasználói nevünket vagy jelszavunkat, erre figyelmeztet a program, illetve hasonlóan figyelmeztet, ha már létező felhasználói nevet és jelszót probálunk meg újból regisztrálni. Ilyenkor lehetőségünk van újból próbálkozni.



2.3. ábra. Hibás bejelentkezés

Sikeres regisztráció után rögtön lehet új levelet írni, míg bejelentkezés után meglévő (régi és új) üzeneteinket mutatja a program. A két funkció között bármikor válthatunk az üzenet menü segítségével.

Mindvégig, amíg be vagyunk jelentkezve a programpanel jobb felső részében látható a kulcsazonosítónk, amely megadása szükséges, hogy mások felvehessék velünk a kapcsolatot. A kulcsazonosító egy 64 bites érték, ami 16 darab hexadecimális számjegyként jelenik meg, két csoportra választva egy kötőjellel a 8. jegy után. Természetesen igen nagy véletlen kell hogy előforduljon már ahhoz is, hogy két különböző felhasználónak megegyezzen az utolsó 32 bitet képviselő 8 számjegye, ezért kulcsok letöltése ezen 8 számjegy segítségével is kezdeményezhető és a további számjegyek megadása (kettesével) csak akkor szükséges, ha ütközés fordul elő.

Ehhez kapcsolódóan egy újabb biztonsági meggondolásra fel kell hívni a figyelmet: jelenlegi ismereteink szerint már rendelkezésre állhat akkora számítási kapacitás, amivel létre lehet hozni belátható időn belül egy adott nyilvános kulcshoz azonos 64 bites azonosítóval rendelkező másik kulcspárt. Ezért a biztonsági követelményeink csak úgy garantálhatóak, ha a felhasználó egy ujjlenyomatnak nevezett 160 bites értéket is ellenőríz üzenet küldésekor, illetve fogadásakor. Erre mégegyszer felhívjuk majd a figyelmet az említett funkciók tárgyalásakor.

2.3. Új üzenet küldése

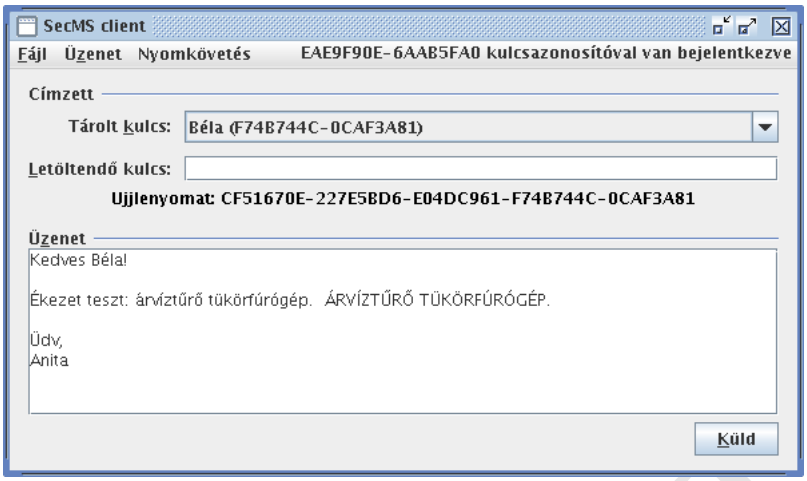
Új üzenet küldéséhez válasszuk ki az üzenet menü új menüpontját, majd a címzett képernyőrészen nézzük meg, hogy a címzett szerepel-e a legördülő listában. Ez akkor fordulhat elő, ha küldtünk neki üzenetet vagy kaptunk tőle üzenetet a program indítása óta. Ha szerepel, akkor innen kiválaszthatjuk, míg ha nem szerepel a listában, akkor a nem tárolt kulcs letöltése funkciónál megadhatjuk a kívánt partner kulcsazonosítóját, ami ekkor letöltésre kerül és amennyiben ez a kulcsazonosító valóban létezik és egyértelmű, akkor a legördülő listában automatikusan ki is választja a program.

Ezekután ellenőrizzük a képernyőn az üzenet felett látható ujjlenyomatot, ami meg kell, hogy egyezzen a címzettel korábban egyeztetett ujjlenyomattal.

A szöveges beviteli mezőben írjuk meg a levelet és kattintsunk a küld gombra. Amennyiben az üzenetet a szerver feldolgozta, a nyugtázó üzenet zöld színnel megjelenik.

Felhívjuk a figyelmet, hogy a jelenlegi verzióban az elküldött levelek nem kerülnek automatikusan elmentésre, így amennyiben fontos, még az üzenet elküldése előtt másoljuk ki és mentsük el saját magunk a küldendő üzenetet a számítógépünkre.

Amennyiben az új üzenet küldésére szolgáló képernyőt úgy próbáljuk elhagyni, másik menüpont választásával, hogy már megírtunk valamennyit a levél tartalmából, akkor figyelmeztet minket a program, hogy ez az éppen írt levél elvesztésével jár. Ilyenkor még lehetőségünk van a mégse gomb választásával a művelet visszavonására. A kliensprogram tehát nem biztosít lehetőséget párhuzamosan új levél írására és a régiek böngészésére, viszont működése olyan egyszerű, hogy nem okoz semmiféle problémát, ha a számítógépen kettő klienst is elindítunk (akár böngészővel, akár alkalmazásként). Az egyik progarammal olvashatjuk ekkor a levelinket, amíg a másikban az újat fogalmazzuk.



2.4. ábra. Üzenet elküldés előtt

2.4. Üzenetek olvasása és megválaszolása

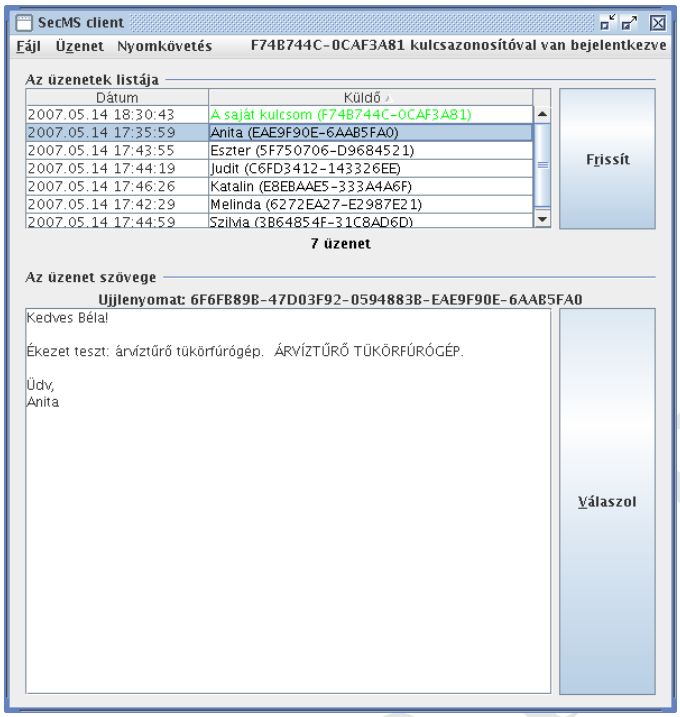
Bejelentkezés után rögtön a kapott üzeneteink listáját látjuk, de később is visszatérhetünk ide az üzenet menü lista menüpontjával. A képernyő felső részén látható táblázatban minden üzenetnek szerepel a feladója és a feladás dátuma, s ezek alapján rendezhető is (akár növekvően, akár csökkenően) a fejlécre kattintással. A táblázat alatt az összes üzenetünk számát írja ki a program.

Ahhoz, hogy egy üzenetet elolvassunk, ki kell jelölni a listában, ami után a képernyő alsó részén láthatóvá válik és amennyiben a hossza indokolja görgető sávval lapozhatunk is.

Az üzenet hitelességében csak akkor lehetünk biztosak, ha az üzenet fölött megjelenő ujjlenyomtatot ellenőríztük. Az ujjlenyomat ellenőrzését két speciális esetben helyettesíti a kliensprogram. Bármikor, amikor a saját kulcsunkról van szó egy listában, akkor a megjelenített név „A saját kulcsom”, és a név színe ekkor zöld, kivéve ha a lista kijelölt eleme ez, ekkor sárga; ugyanis a zöld a kék kijelölés háttéren nem olvasható. Piros színnel jeleníti meg „A szerver kulcsa” üzenettel a szerver kulcsát a kliensprogram a nyomkövetés menü tárolt kulcsok menüpontjában (ld. később).

Az üzenetek táblázatát a program bejelentkezéskor tölti le a kiszolgálóról, az automatikusan nem frissül. Ha azt sejtjük, hogy olyan új levelünk érkezhetett, ami nem szerepel még a táblázatban, akkor nyomjuk meg a frissít gombot!

Ha az olvasott üzenetre válaszolni szeretnénk, nyomjuk meg a válaszol gombot, ennek hatása azonos azzal, mintha új üzenetet küldenénk és a part-



2.5. ábra. Üzenetek olvasása

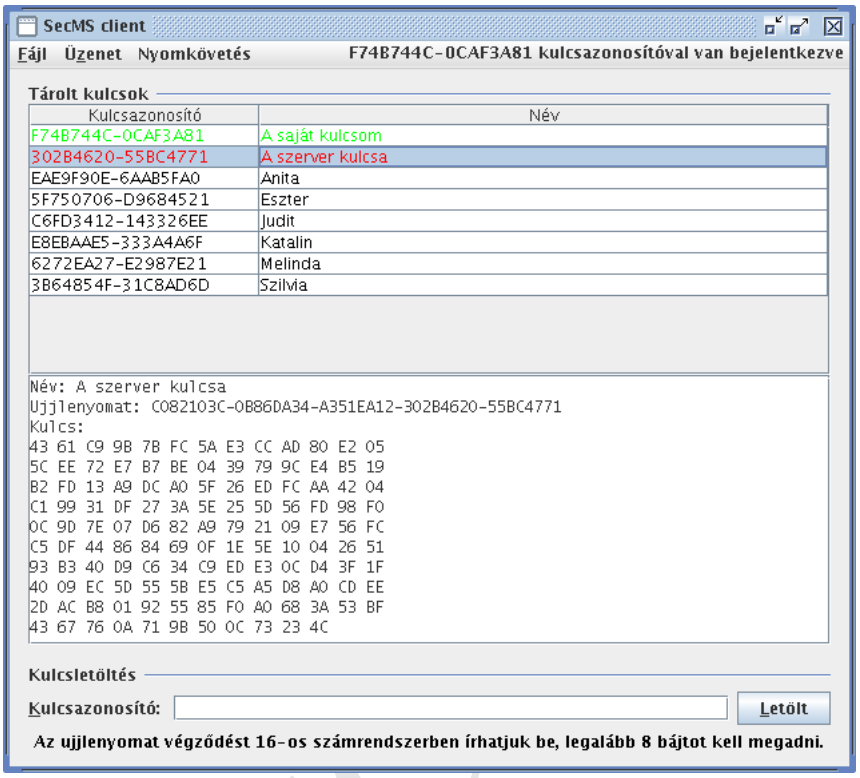
nerünket választanánk címzettnek, azonban további kényelmi szolgáltatás, hogy a kapott üzenet sorait az email szokásainak megfelelően megjelöli a program, így a válaszainkat úgy rendezhetjük, hogy aki a választ kapja emlékezzen rá, hogy mire is válaszoltunk.

2.5. Tárolt kulcsok adatai

A program memóriájában tárolt kulcsokat megtekinthetjük a nyomkövetés menü tárolt kulcsok menüpontja alatt. Miután kiválasztunk egy kulcsot, a szöveges ablakban megjelenik a kulcshoz rendelt megjelenítendő név, a kulcs ujjlenyomata és a teljes nyilvános kulcs. Amennyiben a saját kulcsunkat választjuk, akkor a DSA kulcs titkos értéke is látható.

Ez a képernyő leginkább a hibák felderítésében és kijavításában nyújt segítséget a program fejlesztése során, de a felhasználónak is segítséget nyújt, mivel itt lehetőség van a különböző értékek kijelölésére és így más programba másolására. Ebben az ablakban nyílik csak lehetőség a szerver ujjlenyomatának ellenőrzésére is. Bár megjegyezzük, hogy a SecMS felépítése miatt a biztonsági követelmények alapvetően nem sérthetőek meg a szerver, illetve az internetkapcsolat aktív vagy passzív lehallgatásával. Ugyanakkor sok ké-

nyelmetlenségtől megkíméljük magunkat, ha biztosak vagyunk benne, hogy megfelelő kiszolgálógéppel állunk kapcsolatban.



2.6. ábra. A szerver kulcsa

2.6. Üzenetváltás a biztonságra ügyelve

Példaként feltesszük, hogy egy SecMS rendszerbe már regisztrált két felhasználó, nevezetesen Anita, kinek a jelszava „küldő” (természetesen ékezetek nélkül[3]), illetve Béla, kinek a jelszava „fogadó” (szintén ékezet nélkül). Béla és Anita azért választott ilyen könnyen kitalálható jelszavakat, mert csupán a rendszer tesztelése volt a feladatuk.[4]

A kezdeti regisztráció után mindketten ellenőrízték a tárolt kulcsok képernyőn, hogy a szerver kulcsának ujjlenyomata megegyezik a szolgáltató által

[3]Bár az ékezetek kezelése nem okoz problémát a kliensprogram egyik részén sem, de Anita és Béla jól tudja, hogyha egy konkrét környezetben még be is tudja írni megfelelően a magyar ékezeteket, nem biztos, hogy erre módja lesz minden számítógép előtt.

[4]Egy másik, általuk élesben használt SecMS rendszerben kis- és nagybetűket, valamint számokat tartalmazó jelszavuk van mindkettejüknek.

előre ígérttel, így tudják, hogy az üzeneteiket annak a szervernek küldik, akinek a kezelőjével megállapodtak.

Regisztráció után azt tapasztalta Béla, hogy az ő kulcsazonosítója, amit a jobb felső sarokban lát `F74B744C-0CAF3A81`, ezt az értéket telefonon egyeztette Anitával, aki cserébe elárulta, hogy az általa regisztrált felhasználónévhez és jelszóhoz a `EAE9F90E-6AAB5FA0` kulcsazonosító tartozik. Egymás kulcsainak 160 bites ujjlenyomatait is kicserélték.

Ezekután csupán arra kell figyelniük, hogy üzenetek küldésekor ellenőrizzék a megjelenő ujjlenyomatot, miután kiválasztották a címzettet, illetve hogy olvasott üzenet hitelességében vagy integritásában ne bízzanak egészen addig, amíg az üzenet felett megjelenő ujjlenyomatot nem hasonlították össze az általuk feljegyzett, a feladóhoz tartozó ujjlenyomattal.

Természetesen egy olyan kliensprogram esetében, amit saját eszközről használ a felhasználó lehetőség van ezeknek az ellenőrzéseknek a számát jócskán csökkenteni a már ellenőrzött értékek tárolásával. A mobiltelefonon futó alkalmazás többek közt ebben is kényelmesebb lesz.

3. fejezet

Fejlesztői dokumentáció

A szakdolgozatban bemutatott klienst a Java programozási nyelven készítettem el. Ez egy széleskörben elterjedt, imperatív, objektum–orientált programozási nyelv, az elkészített programok bináris kódja (bájtkódja) változtatás nélkül futtatható különböző típusú számítógépeken, ezáltal az elkészített programok rendkívül könnyen hordozhatók. Meg kell jegyezni, hogy ez a hordozhatóság természetesen azokra az architektúrákra korlátozódik, amikre készült a bináris kódhoz értelmező (ún. virtuális gép); ezek száma azonban jóval kisebb, mint pl. a C vagy Ada fordítóval rendelkező architektúrák száma.

Ugyanakkor hatalmas előny, hogy a grafikus felhasználói felület fejlesztését lehetővé tévő Swing csomag minden platformon azonos kinézetet ad. Sajnos a Swing az alapnyelvnél, a Javánál jóval kevesebb architektúrán támogatott.

A böngészőbe épülés lehetősége miatt is célszerű volt ezt a programozási nyelvet választani. Fontos szempont volt továbbá az is, hogy a mai mobiltelefonok egyetlen közös programozási nyelve a Java, a jövőben pedig a szakdolgozatomból kiindulva szeretnénk telefonon futó SecMS alkalmazást készíteni.

A SecMS rendszer készítése során az ELTECRYPT kutatócsoportban a Subversion verziókezelő–rendszert használtuk. Ezzel lehetővé vált, hogy a programfejlesztés során nyomonkövethessük a változtatásokat, felfedezett hibák esetén egészen pontosan rögzítsük, hogy a hiba melyik „verzióban” volt. Ennek használatát nem részletezem, mivel jelenleg a mi forrásfánk egyik része sem érhető el nyilvánosan.

Először bemutatom a program lefordításához szükséges parancsokat és felhasznált eszközöket, majd ismeretem az általam használt és a projekt részeként beállított fejlesztői környezetet, bemutatom a felhasznált (a Java részét nem képező) fejlesztői programcsomagokat és könyvtárakat, leírom az elkészített osztályok szerepét, működését és csomaghierarchiába szervezésük módját. A fejlesztői dokumentációt a végzett tesztelés leírása zárja.

3.1. Fordítás, Apache Ant

Amennyiben a programot módosítjuk, a futtatás előtt szükséges a bináris kódot újrafordítani, ez már egyszerűbb programok esetében is bonyolult lehet, mivel figyelni kell a különböző függőségekre, be kell szerezni a felhasznált programkönyvtárak egy másolatát, végül elő kell állítani azt a tömörítvényt, ami már csak egy fájl és megfelelően van formázva ahhoz, hogy a böngészők appletként futtatni tudják.

Természetesen ezek a feladatok mind nagyon jól automatizálhatóak, ezért már nagyon régóta léteznek a folyamat leírására szánt programnyelvek.

Unix alapú környezetben nem ritka, hogy egy shell scriptet — azaz a parancsértelmező által futtatható programot — mellékelnek, ami tartalmazza a szükséges utasításokat a program lefordításához. Ez elég pazarló, mivel minden kis módosítás esetén minden parancsot végrehajt, illetve ha ennél komolyabb logika van benne leprogramozva, akkor eléggé bonyolult és nehezen módosítható.

Szerencsére a módosított és lefordított fájlok módosítási idejének figyelése a konkrét forráskódtól függetlenül megvalósítható, ezért létrehozták az ún. `make` segédeszközt, aminek egy elterjedt implementációja a GNU Make[1]. Megfelelően leírva a függőségeket a `make` kiszámítja és végrehajtja a szükséges feladatok körét, amiket shell script utasításokkal kell megadni.

Azért nem választhattuk egyik említett megoldást sem, mert a shell scriptekben biztonsággal felhasználható parancsok köre elég szűk, ui. ezek minden Unix alapú gépen megtalálhatóak kell, hogy legyenek. Nincs írott vagy íratlan megegyezés pl. azt illetően, hogy pl. letöltések kezdeményezésére milyen parancs javasolt. Ha elegendőek lennének számunkra a biztosan megengedett parancsok, akkor sem működne ez a megoldás nem Unix alapú operációs rendszereken, hiába támogatja őket a Java.

Egy bevált, a Java világában (és azon kívül) sokak által használt XML alkalmazás, ami programfordítások leírására és végrehajtására való, az Apache Ant[2]. Az Ant Java nyelven készült, platformfüggetlen, minden igényünket kielégítő megoldás. Egyedül a `po/` könyvtáron belül található, a felhasználói felület magyar fordítását tartalmazó fájlok binárissá alakítása túl rendszer- és telepítésfüggő ahhoz, hogy az Ant eszközt használjuk, ezért ott visszanyúlunk a `make` parancshoz; valamint a megfelelő rendszerrel nem rendelkező fejlesztők kedvéért az egész `po/` alkönyvtárat és nem csak a forrást terjesztjük.

Ahhoz, hogy az Antot használjuk, el kell készíteni a programforrás főkönyvtárában egy `build.xml` nevű XML dokumentumot, ami a tevékenysé-

[1] http://www.gnu.org/software/make/
[2] http://ant.apache.org/

gek leírását, egymástól való függéseit tartalmazza (pl. fordítás nélkül nem lehet elkészíteni a kész tömörítvényt). Ezen leírások és a fájlokhoz tartozó időbélyegek alapján az Ant különböző moduljai kiszámítják és elvégzik a szükséges tevékenységeket, a fordítást indító programozónak csupán egy magasszintű célt kell megadnia.

A SecMS kliens fordításának lehetséges céljai (zárójelben az egyes cél előtt kötelezően és automatikusan végrehajtandó egyéb célok nevei szerepelnek):

- *download*: letölti az Internetről a fordításhoz és futtatáshoz szükséges programcsomagok rögzített, tesztelt verzióját, azok nyilvános archívumából;
- *createlib (download, libclean)*: a letöltött programcsomagokat „feldolgozza”: általában minden programkönyvtárat egy tömörítvényként publikálnak, ami tartalmaz a futtatható és felhasználható bináris bájtkódon kívül dokumentációt, fordítási segédleteket, példaprogramokat, stb. Ezért szükséges a lényegi bájtkódot előállítani a terjesztett formátumból és ezt a `lib/` könyvtár alatt elhelyezni, ahol futtatáskor a Java virtuális gép meg tudja találni.
- *depend*: letörli azokat a kliensprogramhoz tartozó bájtkódfájlokat, amiket szükséges újrafordítani, mert a hozzájuk rendelt forrásfájl megváltozott. Az olyan bájtkódfájlokat is letörli, amik olyan bájtkódfájlt használnak futás közben, amiket az előző szabály miatt le kellett törölni, és így tovább.
- *compile (depend)*: előállítja azokat a bájtkódfájlokat, amik nem léteznek, pedig van azonos nevű Java forrásfájl.
- *jar (compile)*: kiszámítja a hibátlan futáshoz szükséges bájtkódfájlok legszűkebb halmazát és ezekből elkészíti a futtatható és terjeszthető bájtkódarchívumot.
- *clean*: letörli a fordításkor (a `compile` cél által) előállított `build/` eredménykönyvtárat.
- *libclean*: letörli a `createlib` által létrehozott `lib/` nevű könyvtárat.
- *distclean (clean, libclean)*: letörli a `download` által létrehozott `dl/` alkönyvtárat, a parancs futtatása után a SecMS forráskönyvtára csak nélkülözhetetlen, nem automatikusan generált fájlokat tartalmaz. Tehát egy ilyen archívum publikálásra kész, a cél neve is erre emlékeztet.
- *run (compile)*: alkalmazásként futtatja a SecMS kliensprogramot.

- *javadoc*: előállítja a `javadoc/` alkönyvtár alatt a program forráskódjának angol nyelvű dokumentációját a kódban elhelyezett (angol) megjegyzések alapján.

Lényegében egyszerű fejlesztési feladatok közben csak a `run` célt szükséges használni, ugyanis ez rögtön fordítja és futtatja is az alkalmazást és így rögtön láthatjuk a módosítás hatását (vagy a fordítási hibaüzeneteket, természetesen az Ant okos és fordítási hiba esetén nem futtatja a régi kódokat).

A `createlib` célra lehet még szükség, de csak egyetlen egyszer, a forrás kibontása és használatba vétele után, ez letölti és rögtön megfelelően konfigurálja is a felhasznált programkönyvtárakat a `lib/` alkönyvtárban.

3.2. Fejlesztői környezet: GNU Emacs, JDEE

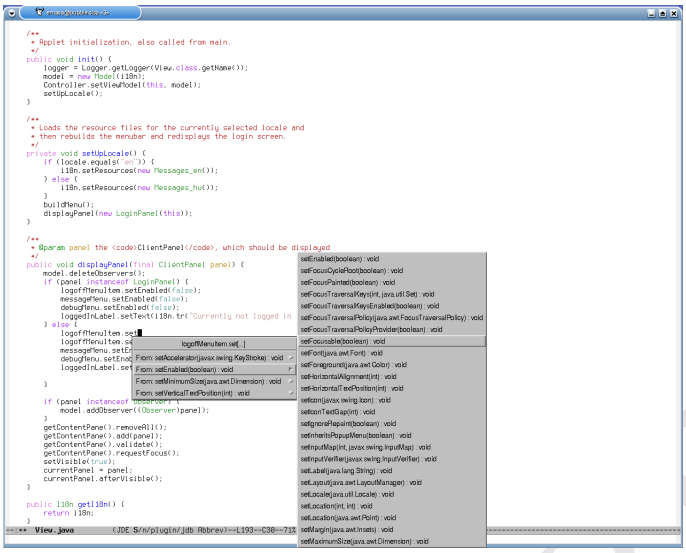
A forrásszöveg szerkesztésére az egyik legrégebben aktívan fejlesztett editort, a GNU Emacset[3] használtam. Az Emacs egy teljesen testreszabható, bővíthető, nagy felhasználói táborral rendelkező program, ennek megfelelően nagyon sok kiegészítés létezik hozzá.

Ilyen kiegészítés a JDEE[4] (Java Development Environment for Emacs) is, amivel a java programkódok létrehozása és módosítása nagyon kényelmes. Telepítése után olyan szolgáltatásokat nyújt, mint a forráskód szintaktikájának színekkel való kiemelése, a forráskód szemantikai értelmezésével kiegészítések felajánlása, egy adott környezetben releváns dokumentáció böngészése, interaktív hibakeresés, stb.

Mindezen szolgáltatások talán könnyebben elérhetőek egy modern grafikus fejlesztői környezet feltelepítésével, azonban ezekre jellemző, hogy a tanulási folyamat rövidségéért cserébe az nem is folytatható, a munkánk sokkal jobban nem gyorsul fel az eszköz egyre jobb megismerésével. Ezek az eszközök sokszor nem is futnak több platformon, reakcióidejük még modern számítógépeken is nagyságrendekkel elmaradnak az Emacs mögött, továbbá nem stabilak.

A projektre vonatkozó beállításokat a forráskönyvtár `prj.el` nevű emacs lisp fájla tartalmazza.

[3] http://www.gnu.org/software/emacs/
[4] http://jdee.sunsite.dk/



3.1. ábra. GNU Emacs, JDEE

3.3. Felhasznált programcsomagok

3.3.1. Log4j

Az Apache alapítvány keretein belül készült Log4j[5] projekt egy olyan utasításkészletet ad a programozók kezébe, amivel egyszerűen tudják kiegészíteni naplózással a Java alkalmazásaikat.

Tapasztalatom szerint a készülő (illetve elkészült) programban lévő szemantikai hibák sokkal könnyebben érhetőek meg és javíthatók naplózás segítségével, mint nyomkövetéssel. A nyomkövetés habár nagyon hasznos segédeszköz és sokszor nélkülözhetetlen, kicsit is bonyolultabb környezetben (pl. a felhasználó távoli gépén, mobiltelefonon) nehezen kivitelezhető. Ahhoz, hogy nyomkövetéssel felderíthető legyen egy hiba, az is kell, hogy biztosak legyünk a hiba reprodukálásának módjában és ne arról legyen szó, hogy mondjuk naponta egyszer jelentkezik.

Az összetettebb, nehezebben kezelhető hibák esetleg megelőzhetőek vagy megértésük jelentősen könnyíthető naplózással, illetve kikötések ellenőrzésével (assertek). Mindkét funkció megtalálható már a Java nyelv 1.5-ös verziójában, azonban a naplózásra a szakmában elfogadottabb és régebb óta rendelkezésre álló megoldás a Log4j, tulajdonképpen a Sun megvalósítása majdnem egyezik az Apache korábbi megvalósításával.

Használatához minden olyan osztályban, ahol naplózást szeretnénk hasz-

[5] http://logging.apache.org/log4j/docs/

nálni, létre kell hozni egy statikus `Logger` objektumot az alábbi utasítással:

```
private static Logger logger =
        Logger.getLogger(Osztálynév.class.getName());
```

Bármelyik metódusban meg lehet hívni ezután a `logger.fatal(...)`, `logger.error(...)`, `logger.warn(...)`, `logger.info(...)` utasításokat, amivel egy-egy naplóbejegyzést helyezünk el a naplóban. Pontosan fogalmazva, azt a Log4j programcsomag dönti el a programkódtól függetlenül szerkeszthető konfigurációs fájl alapján, hogy milyen részletességgel és milyen médiára naplózzon. Különféle moduljaival akár képernyőre, háttértárolóra vagy hálózatra is naplózhatunk.

A forráskönyvtár főkönyvtárában lévő `log4j.properties` fájl az említett konfigurációs fájl abban az esetben, ha a SecMS kliensprogram appletként fut, és a `log4j.properties.application` abban az esetben, ha alkalmazásként. A jelenlegi beállításokkal minden fontosságú üzenet megjelenik, applet esetén a java konzolon, alkalmazás esetén a képernyőn, illetve a `logs/` alkönyvtár `client.log` nevű fájljában.

3.3.2. JGoodies Forms

Nem könnyű feladat olyan felhasználói felületeket készíteni, amik amellett, hogy kényelmesen, gyorsan kezelhetőek, átméretezhetőek is.

Sok ma használt felület nem is biztosítja az átméretezhetőséget. Szélsőséges esetben ez a funkcionalitás teljes elvesztéséhez is vezethet: a legnagyobb magyarországi bank internetes felülete jó ideig appletként volt elérhető. Méghozzá olyan appletként, ahol képpontról képpontra meghatározták, hogy mi mekkora, egy konkrét számítástechnikai platform egy konkrét beállításai és betűkészletei mellett. Amikor valaki elindította ezt az elvileg hordozható alkalmazást a bank weblapjáról, úgy, hogy nem pont azt a környezetet használta, mint amit a fejlesztők, akkor a kiszámított méretek nem működtek, ezért például nem volt kiválasztható a menüsorban a jelszóváltoztatás. Mivel a szerződéskötés után kötelező volt jelszót változtatni, a megfelelő platformmal nem rendelkező felhasználók gyakorlatilag nem tudták használatba venni a szolgáltatást.

A legtöbb grafikus fejlesztőeszköz egyenesen ösztönzi a programozót arra, hogy ilyen kényelmetlen (vagy szélsőséges esetekben hibás) programot készítsen, azzal, hogy a felhasználói felület tervezésére csak olyan eszközöket biztosít, amiknél a képpontról képpontra kell megadni a felületi elemek pontos helyét és méretét. Szerencsére a jobbakban már lehetőség van az átméretezhetőség módjainak megadására (akár grafikus, akár szöveges úton).

Az viszont tényleg minden grafikus fejlesztőeszközről elmondható, hogy az általa létrehozott grafikus felület más segédeszközökkel nem vagy csak nagyon körülményesen módosítható, ráadásul az ember által írt és programlogikát tartalmazó forráskódba sok-sok érthetetlen, gépi generálású programszöveget raknak, ez az átláthatóságot csökkenti, ha más eszközzel vizsgáljuk a programot, ami nem rejti el ezeket a „fölösleges” sorokat a szemünk elől.

Az említett problémák miatt a szakdolgozatomban egy olyan a JGoodies[6] által fejlesztett Forms[7] névre hallgató felületelem elhelyező megoldást (layout managert) használok, amivel viszonylag rövid tanulási idő után praktikusan, szépen (a különböző belső konzisztenciákra figyelve), a fejlesztői környezettől függetlenül megoldható a felhasználói ablakok megtervezése az átméretezhetőséget megtartva.

A Forms használatával úgy oldható meg a feladat, hogy a megjelenítendő felület különböző elemeit egy táblázat rácsán képzeljük el és ezen táblázat különböző tulajdonságait (az egyes oszlopok és sorok szélességét és magasságát, közeik méreteit, átméretezés esetén a növekedési és csökkenési arányokat) adjuk meg egy speciálisan az erre a célra készített leírónyelvvel. Ezután már csak létre kell hozni a felületi elemeket és elhelyezkedésük sorrendjében hozzáadni a táblázathoz. Ilyenkor van lehetőség bizonyos elemhez megadni azt is, ha az az elem több oszlopot vagy sort kell, hogy átöleljen (pl. egy hosszú nyomógomb a képernyőablak alján).

Magáról a nyelvről, valamint az API felhasználási módjáról részletesen tájékoztat a Forms weblapja.

3.3.3. Gettext Commons

A SecMS projekt legnagyobb részének nyelve angol. Angolul dokumentáltuk a protokoll leírást, angolok a szerver és a kliens naplóüzenetei, a programban használatos azonosítók angol nyelven emlékeztetnek a megfelelő funkcióra, azonban természetesen a kliens felhasználói felülete több nyelvű kell, hogy legyen. Példaként második nyelvként a magyar nyelv támogatását oldottuk meg, ugyanis ez a szakdolgozat nyelve is.

A már említett Emacset fejlesztő GNU csapat 1998-ban kezdett fejleszteni a C nyelvhez egy olyan segédeszközt, amivel a kódsorokban kijelölve a fordítandó üzenetek körét a fordítást már nyelvenként más és más fájlban (és így más és más személy által) lehet elvégezni, az elkészült fordítás a programkód teljes újrafordítása nélkül tesztelhető, a fordítások külön terjeszthetőek és tárolhatóak. Ez a rendszer a Gettext[8] nevet viseli, azóta számos nyelvhez

[6] http://www.jgoodies.com/
[7] http://www.jgoodies.com/freeware/forms/
[8] http://www.gnu.org/software/gettext/

elkészült az adoptációja.

A Java esetében (a virtuális gép megkötései miatt) kicsit másképpen oldották meg az illesztést — a Gettext Commons nevű programkönyvtárral —, a programozónak létre kell hoznia egy megfelelően felparaméterezett példányt az `I18n` osztályból és ennek a példánynak a `tr`, illetve (paraméterekkel rendelkező szövegrészek esetén) `trc` utasításaival kérheti le az éppen beállított nyelvhez tartozó sorokat.

A Gettext Commons[9] SecMS kliensbeli felhasználásához kapcsolódó infrastruktúra a `po/` alkönyvtáron belül van, itt található a fordítást bináris java osztállyá alakító `make` programot vezérlő `Makefile`, illetve maga a magyar fordítás is (`hu.po`).

3.3.4. SecMS Commons

A protokoll által használt kriptográfiai elemekre mind a szerver, mind a kliens implementációjában szükség van, ezért ezek megvalósítását a referenciaszerver írójával közösen egy Secms Commons nevű csomagban foglaltuk össze. [SECMS]

A programkönyvtár utasításkészletének az alapvető működési ötlete, hogy minden protokollbeli üzenet ábrázolható egy fával, ahol a levelek hordozzák a különböző tényleges információkat (parancsok a szervernek, felhasználók egymásnak küldött üzenetei), a fa belső szerkezete pedig a használt rejtjelezésnek megfelelően épül fel.

Amennyiben üzentet kívánunk előállítani, akkor a programnyelv utasításaival létrehozzuk az üzenetnek megfelelő fát, majd a gyökérelem megfelelő metódusa elő tudja állítani a rejtjelezett bájtfolyamot.

Ellenkezőleg, amennyiben egy kapott üzenetet szeretnénk a programcsomag használatával megfejteni, akkor meghívva egy megfelelő osztályszíntű metódust, a rejtjelezett üzenetből felépül a hozzá tartozó fa, mindaddig kibontva, amíg a megadott privát kulccsal lehetséges. Ezekután a fa hagyományos java utasításokkal már elemezhető és bejárható.

Az osztályok elnevezése analóg a protokoll leírásban használt fejezetcímekkel, így annak segítségével a programkönyvtár rögtön használatba vehető.

3.4. Csomaghierarchia (osztályok)

A SecMS kliensprogram minden osztálya a `mik.client` csomagon belül helyezkedik el, így a forrásfájlok mindegyike valahol az `src/mik/client` al-

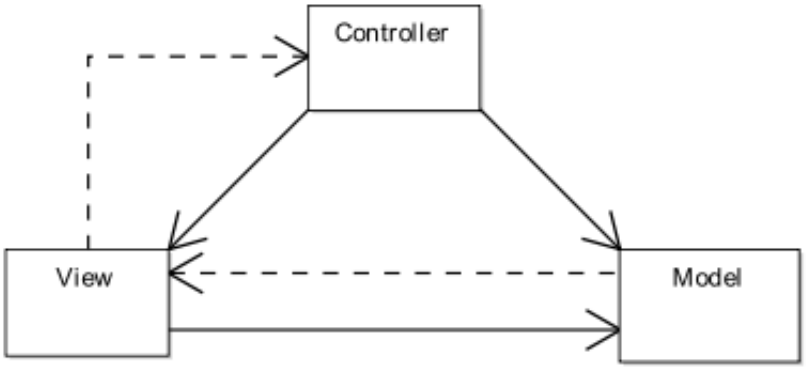
[9] http://xnap-commons.sourceforge.net/gettext-commons/

könyvtár alatt van. A MIK[10] annak a projektnek a rövidítése, amely keretein belül a SecMS rendszert az ELTECRYPT készíti.

Ezen a névtéren belül az MVC (model–view–controller, modell–nézet–vezérlő) tervmintát használom, melynek lényege, hogy szétválasztja a feldolgozandó információk, adatok tárolását (modell) azok prezentációjától (nézet) és a rajtuk végrehajtható műveletektől (vezérlő).

Például a kliens által tárolt kulcsokat és üzeneteket a modell valamelyik alosztálya tárolja, míg a felhasználó által látott komponenseket mindig valamelyik olyan osztály valósítja meg, amelyik a nézet csomagba tartozik. A vezérlőbe tartoznak a felhasználó által generált események által előidézett műveletsorok, például a regisztrációkor a szerverrel való kommunikáció, majd sikeres regisztráció után a megfelelő panel kiválasztása és megjelenítése.

A tervmintát a könnyű újrafelhasználás érdekében úgy kell használni, hogy közvetlenül csak a vezérlő modul érheti el a modell és a nézet osztályait, illetve a nézet a modell osztályait. A nézet a vezérlőt, illetve a modell a nézetet csak korlátozottan (pl. figyelőkön keresztül) kezelheti.



3.2. ábra. Az MVC tervminta

Ebben a részben leírom a három modul összes osztályának alkotóelemeit, azok működését, a fontosabb műveleteket (metódusokat). A kód részletes megértését a benne elhelyezett megjegyzések segítik, amik betartják a Java formai megkötéseit, így belőlük automatikus HTML dokumentáció generálható (ld. fordítás, javadoc cél).

[10]Mobil Innovációs Központ (http://www.mik.bme.hu/)

3.4.1. Vezérlő (Controller)

`mik.client.controller.Controller`

A vezérlő csomag minden osztályának őse. Osztályszíntű (statikus) hivatkozást tartalmaz a `Model` és `View` osztályok egy-egy példányára, hiszen ezek működését hangolja össze, illetve a (hiba)üzenetek többnyelvűsítéséért az `I18n` osztály példányára is.

A vezérlő osztályok nagy része leszármazottja az `AbstractAction` Swing osztálynak is, hogy nyomógombokhoz és gyorsbillentyűkhöz egyszerre hozzárendelhető legyen. Mivel a Java nyelvben nincs többszörös öröklődés, így a `Controller` osztály őse ez kell, hogy legyen. Nem ró ez ránk nagy terhet abban a pár ritka esetben, amikor az `AbstractAction` leszármazottjaként átdefiniálandó `actionPerformed` metódusra igazából nincs szükségünk, ekkor ezt átdefiniáljuk, de üresen hagyjuk.

Mivel minden vezérlő osztály ennek leszármazottja, ott egyszerűen már a `view`, `model` és `i18n` változónevekkel lehet hivatkozni az említett objektumokra, így a vezérlő modul osztályokra bontható anélkül, hogy e három mindenki által igényelt referencia terjesztésével állandóan vesződni kellene.

public static **void** setViewModel(View `_view`, Model `_model`)

A `view` és `model` osztályváltozó beállítását végző statikus metódus. A `view` alapján a `i18n` referenciát is beállítja.

`mik.client.controller.···Action`

Minden `Action` osztály felépítése azonos: a konstruktorban megkapja a nézetpanel azon komponenseit, amiknek tartalmától a működése függ, míg az `actionPerformed` metódusban (ami az `AbstractAction` ősben absztrakt) megvalósítja a kívánt műveletet. A továbbiakban ezen `Action` osztályokról csak annyit sorolunk fel ezért, hogy mi a feladatuk:

- `GetKeyAction`: adott azonosítójú nyilvános kulcs letöltése,
- `GetKeyComboBoxAction`: adott azonosítójú nyilvános kulcs letöltése, majd egy adott legördülődobozban az újonnan beszerzett kulcs kiválasztása,
- `LoginAction`: adott felhasználói névvel és jelszóval a bejelentkezés ellenőrzése és az üzeneteket listázó nézetpanel aktivizálása,
- `RealRegisterAction`: adott felhasználói névvel és jelszóval a regisztráció végrehajtása és az üzenetküldést lehetővé tévő nézetpanel aktivizálása,

- `RefreshAction`: az üzenetlista frissítése,
- `RegisterAction`: a bejelentkezési nézetpanel átalakítása oly módon, hogy az a regisztrációhoz szükséges plusz adatok beírására lehetőséget adjon,
- `ReplyAction`: az üzenetküldő nézetpanel létrehozása a kiválasztott üzenet küldőjével, mint címzettel és a kiválasztott üzenet idézésével, mint tartalommal,
- `SendAction`: üzenet elküldése egy adott legördülődobozban kiválasztott felhasználónak az aktuális dátummal és egy szövegmezőben adott üzenettel.

`mik.client.controller.KeyListSelectionListener`

Mivel implementálja a Swing `ListSelectionListener` interfészét, a nézet `KeyPanel` konstruktora a panelen szereplő kulcsokat kiválaszhatóvá tevő táblázathoz tudja rendelni, mint a kiválasztás megváltozásának eseményére való reakciót.

public **void** valueChanged(ListSelectionEvent `listSelectionEvent`)

Az aktuális nyelvnek megfelelően megformázza és kiírja az adott szövegmezőben a kiválasztott nyilvános vagy titkos kulcs adatait.

`mik.client.controller.MessageListSelectionListener`

Az előző osztályhoz hasonló `ListSelectionListener` interfész megvalósítás, azonban ezt az üzenetek megjelenítéséért felelős panel használja.

public **void** valueChanged(ListSelectionEvent `listSelectionEvent`)

Megjeleníthető, megfejtett üzenet kiválasztásakor láthatóvá teszi az üzenet tartalmát és a feladó ujjlenyomtát. Valamint engedélyezi a válasz gomb megnyomását, ami egyébként természetesen tiltott.

`mik.client.controller.MainMenuListener`

A menürendszer gombjait kezelő osztály, az `actionPerformed` metódust hívja minden menüelem, de különböző `actionEvent` paraméterrel, ez alapján dönthető el, hogy melyik panel jelenjen meg. Az új panel megjelenítését (vagy a programból való kilépést) csak akkor hajtja végre az említett metódus, ha az aktuálisan megjelenített panel `beforeDestroy` metódusa igaz értékkel tér vissza.

3.4.2. Modell (Model)

`mik.client.model.Message`

A `Message` osztály egy üzenetet reprezentál, konstruktora (a szervertől kapott üzenetlistából kinyerhető) üzenetazonosító alapján letölti és megfejti az üzenetet. Amennyiben ez sikeres, akkor a `boolean valid()` metódus igaz értéket értéket ad vissza, különben hamisat, az előbbi esetben a többi getter metódus (`getDate()`, `getSender()`, `getText()`) használható az üzenet részleteinek kinyerésére.

`mik.client.model.Messages`

A felhasználó összes érkezett üzenetét reprezentáló osztály, a konstruktorban automatikusan letölti a rendelkezésre álló üzeneteket, de ez a lista később az `update` metódus használatával újratölthető. A lista ismeretében létrehozza az összes üzenethez a hozzátartozó `Message` példányt.

Kiterjesztése az `AbstractTableModel` Swing osztálynak, így segítségével egy megjelenített táblázat feltölthető. Ezt a funkcionalitást a `getRowCount`, `getColumnCount`, `getColumnName`, `getValueAt` metódusok biztosítják, a szokásos tábla modell interfésznek megfelelően.

public **void** update(TimeJLabel `errorLabel`)

Frissíti az üzenetlistát, az esetlegesen előforduló hibákat az `errorLabel` címkén keresztül jelzi. Mivel az üzenetek törlése jelenleg nem lehetséges, az üzenetlistát csak bővíteni tudja ez a metódus. Ha majd már lesz üzenettörlés a protokollban, akkor ezt a részt ennek figyelembevételével át kell írni.

public **Object** getValueAt(int `row`, int `column`)

A metódus hívható -1-el, mint oszlop (`column`) számmal, aminek eredményeként a megfelelő sorhoz (`row`) tartozó teljes `Message` objektumot visszaadja. Ekkor természetesen a felhasználó programkódnak a visszaadott `Object` hivatkozást explicite `Message` hivatkozásként kell értelmeznie. Erre a „trükkre” akkor van szükség, amikor a felhasználói táblázat a `TableSorter` segítségével rendezhető és így a táblázat által visszaadott `getSelectedRow` érték függ az aktuális rendezéstől, míg a `TableSorter` modellen keresztül kapott érték nem. A Java 1.6-ban a táblázatok már külön `TableSorter` nélkül is rendezhetőek és ott ez probléma elegánsan, egy `Message getMessage(int row)` metódus bevezetésével megoldható lesz.

`mik.client.model.Model`

Tartalmazza a kliens titkos és nyilvános, a szerver és egyéb partnerek tárolt nyilvános kulcsát, a szerverről már letöltött üzeneteket, a szerver felé hasz-

nálható kommunikációs csatornát és beállításokat. Metódusaival ezeken az adatokon operálva valósítja meg a szükséges funkcionalitást.

Az `Observable` osztályból öröklődik, így az esetleges változásról értesíteni tudja a képernyőn éppen megjelenített nézet panelt.

Implementálja a SecMS Commons `KeyLoader` interfészét, méghozzá úgy, hogy a kért partner nyilvános kulcsát a szerverről tölti le.

`mik.client.model.NamedKey`

Egy nyilvános kulcsot és a vele megjelenítendő nevet összerendelő osztály. Rendelkezik `equals` metódussal, így két `NamedKey` összehasonlítható, és három különböző megjelenítési módot eredményező karakterlánc-előállító metódussal:

public **String** keyIDToString()
A kulcs azonosítóját `XXXXXXXX-XXXXXXXX` formában állítja elő.

public **String** fingerprintToString()
`XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX` formában állítja elő a kulcs ujjlenyomatát.

public **String** toString()
`Megjelenített név` `(XXXXXXXX-XXXXXXXX)` formában állítja elő a kulcs ujjlenyomatát és a kulcshoz rendelt nevet.

`mik.client.model.ProtocolException`

A modell kommunikáló metódusai által használt kivétel osztály, egy informatív üzenettel dobják abban az esetben, ha a szerver nem várt, hibás módon reagál egy kérésre.

`mik.client.model.Talker`

A szerverrel való kommunikációt, a rejtjelezést és megfejtést vezérlő osztály.

public **«constructor»** Talker(String `serverHost`, int `serverPort`, BuildContext `bc`, ParseContext `pc`)
Egy-egy megfelelően inicializált `BuildContext` és `ParseContext` példányt használva kommunikál a `serverHost` és `serverPort` által meghatározott szerverrel.

Szinkron üzenetváltást támogat, mindig megvárja a szerver által adott rejtjelezett választ, amit meg is fejt az adott `ParseContext` segítségével. A rejtjelezendő és elküldendő üzenetfát vagy amennyiben csak alcsomagokat

(ld. SecMS protokoll leírás[SECMS]) küldünk, annak gyökerét két különböző túlterhelt `syncMessage` nevű metódus dolgozza fel:

public **Packet** syncMessage(Packet `request`)

public **SubpacketContainerPacket**
syncMessage(SubpacketContainerPacket `spRequest`)

3.4.3. Nézet (View)

`mik.client.view.ClientPanel`

A főablakban mindig egy panel jelenik meg, amin valamilyen műveletet elvégezhet a felhasználó. Minden ilyen panelnek közös őse a `ClientPanel` osztály, ami leszármazottja a Swing által szolgáltatott `JPanel` megjeleníthető panelnek.

Mivel magát a ClientPanelt muszáj kibővíteni, hogy valódi viselkedése lehessen, absztrakt, így nem példányosítható.

protected **«constructor»** ClientPanel(final View `parent`)

A konstruktor a kapott View referencián keresztül protected változókba elhelyezi a `view`, `i18n`, `model` referenciákat, amiket így a leszármazott osztályok egyszerűen elérhetnek.

public **boolean** beforeDestroy()

Átdefiniálásával a leszármazott elérheti, hogy valamilyen műveletsort végre hajthasson, mielőtt egy másik panelre cserélné a program. A metódus visszatérési értéke jelzi, hogy hozzájárulását adja-e a panel a lecseréléshez. Ezt a lehetőséget jelenleg akkor használjuk, amikor a felhasználó egy félkész levél írása közben másik panelre kíván váltani vagy ki szeretne lépni a programból, ekkor figyelmeztetjük és amennyiben a mégse gombot választja, akkor hamis értékkel térünk vissza, megszakítva így a levél elvesztéséhez vezető folyamatot.

public abstract **void** afterVisible()

Átdefiniálása kötelező, az a műveletsor, ami a panel teljes megjelenítése után végzendő, pl. valamelyik komponenshez a fókusz hozzárendelése.

`mik.client.view.KeyPanel`

A nyomkövetés menü alatt található, a tárolt kulcsok részleteit megmutató panel. Mivel lehetőséget biztosít új kulcsok letöltésére is (a vezérlő GetKeyAction osztályával), megváltozhat a tárolt kulcsok listája, ezért implementálja az `Observer` interfészt, amin keresztül értesül a modell változásáról és ekkor újra feltölti a táblázat adatait.

A sorok megjelenítését a `KeyTableCellRenderer` végzi.

`mik.client.view.KeyTableCellRenderer`

A kulcsokat felsoroló, illetve az érkezett üzeneteket felsoroló táblázat azon sorát, ami a saját kulcsunk zölddel, míg a szerver kulcsát pirossal volt kívánatos kiemelni, ezért egy külön `KeyTableCellRenderer` osztályt is létrehoztam, ami biztosítja ezt, újrafelhasználva a megjelenítés lényegi részét elvégző `DefaultTableCellRenderer` Java Swing osztályt.

`mik.client.view.LoginPanel`

A SecMS bejelentkező-képernyőjét és a nyelvválasztást megvalósító panel.

A lényegi funkcionalitást, a bejelentkezési adatok ellenőrzését, illetve a regisztrációt a vezérlő csomag `RegisterAction` és `LoginAction` osztályai végzik.

A konstruktor egyetlen bonyolult része a vezérlők billentyűzettel való navigálási sorrendjét beállító rész. A `ContainerOrderFocusTraversalPolicy` osztály egy olyan névtelen alosztályát kell itt létrehozni, ami felüldefiniálva a `getComponentAfter` és `getComponentBefore` metódusokat leírja a megfelelő, hagyományostól (nevezetesen a balról jobbra olvasás szabályaitól) eltérő, de a konkrét ablakban logikusabb és gyorsabb navigálási sorrendet.

A `getComponentBefore` a `getComponentAfter` segítségével egy elhanyagolható futási idejű lineáris kereséssel megkapható, így a kívánt sorrendet csak egyszer és egyirányba kell átgondolni és leprogramozni.

A megvalósított sorrenddel támasztott követelmények a következők:

- A nyelvválasztás ritkán használt funkció, a navigálásból kihagyható, elegendő, hogy van gyorsbillentyűje.
- Mivel jóval többször jelentkeznek be a felhasználók, mint regisztrálnak, az első négy komponens a következő legyen: gépnév, portszám, felhasználói név, jelszó, majd a bejelentkezés gomb következzen és csak utána a regisztráció.
- Azonban ha a regisztrációt aktivizálták, boruljon fel ez a sorrend, ekkor már a gépnévre és portszámra nem kell, hogy sor kerüljön, a felhasználói név után a megjelenített név következzen és így a jelszó kétszer egymásután adható meg, ahogy az mindenhol megszokott, majd ugorjunk a regisztráció gombra, a regisztráció megszakítása lehet a legutoljára elért funkció.

A követelmények alapján az elágazásrendszer már könnyen megérthető.

`mik.client.view.MessageListPanel`

A modellben aktuálisan tárolt leveleket megjelenítő panel, lehetőséget ad a modell új, most érkezett levelekkel való feltöltésére is, ezért megvalósítja az `Observer` interfészt, hogy értesülhessen a változásokról.

A táblázat feladó oszlopának megjelenítését a `KeyTableCellRenderer` végzi. A levelek frissítését, illetve a válasz kezdeményezését a vezérlő csomag `RefreshAction` és `ReplyAction` osztályai biztosítják.

`mik.client.view.MessageNewPanel`

Ezzel a panellel lehet új üzenetet írni, mivel a címzett megadásakor lehetőség van még nem tárolt, új kulcs letöltésére is, változtathatja a modellt és ezért megvalósítja az `Observer` interfészt, hogy értesülve a modell változásáról újra feltölthesse a címzett-választódobozt.

A címzett-választódobozban a saját kulcs zöldre színezésének megvalósítása a `KeyTableCellRenderer` osztályhoz nagyon hasonló, azonban itt listáról és nem táblázatról van szó, ezért a `DefaultListCellRenderer` osztály `getListCellRendererComponent` metódusának átdefiniálására volt szükség.

public final **boolean** beforeDestroy()

Ezt a metódust hívja meg a vezérlő, mielőtt az üzenetküldő-panel másikra cserélődne vagy a program végetérne, így átdefiniálásával elérem, hogy ha az üzenet egy részét a felhasználó már megírta, akkor figyelmeztető ablak jelenjen meg, ahol a mégsét választva még megszakítható a művelet. Ezt a vezérlőnek visszaadott hamis érték mutatja.

`mik.client.view.View`

A `View` osztály az alkalmazás/applet főosztálya, abban az értelemben, hogy a végrehajtás mindig ennek az osztálynak valamelyik metódusával kezdődik.

Alkalmazás esetében a statikus `main(String[] args)` metódussal, aminek a feladata csak az, hogy létrehoz egy főablakot és azt beállítja úgy, hogy rögtön a belsejében az „applet induljon el”. Így érjük el, hogy ugyanaz a forráskód fut alkalmazásként és appletként is.

Furcsa lehet, hogy egy az MVC tervminta szerint fejlesztett alkalmazás a nézet komponens végrehajtásával és nem a vezérlőével kezdődik, ennek oka az, hogy appletként futva a Java nyelv megkötése az, hogy az első indított osztály kell, hogy a grafikus felületet nyújtó `JApplet` leszármazott legyen. Ez a feladat viszont egyértelműen a megjelenítési réteghez tartozik.

Szintén az appletek miatt a konstruktort itt az `init()` és `start()` metódusok helyettesítik, ezeket hívja meg a `main` is és ezek végzik a felhasználói

felület felépítését, a vezérlő és a modell inicializációját.

«package private» **void** toggleLanguage()
Ez a metódus valósítja meg a nyelvváltást angol és magyar között, a `LoginPanel` gombja egyszerűen csak ezt hívja.

`mik.client.view.i18n`

Ez az alosztály tartalmazza az egyes nyelvekhez tartozó `ResourceBundle` leszármazottakat `Messages_xx` névvel, ahol `xx` a nyelv kódja, pl. a magyar osztály neve `Messages_hu`. Ezeknek az osztályoknak rögtön a bináris bájtkód alakját generálja a gettext megfelelő segédprogramja, ezt korábban tárgyaltuk. Egyedül az angol nyelvhez kellett egy kis segédosztályt írni, mivel maga a programkód angolul készült el és csúnya megoldás lett volna készíteni egy olyan „fordítást", ami angolról angolra fordít. Ez a segédosztály (`Messages_en`) a lehető legszükségesebb metódusait implementálja csak (a ResourceBundle ősben absztraktakat) a lehető legtriviálisabban (mindig `null`, illetve `false` értéket visszaadva), mivel így a felhasználó gettext csomagba tartozó metódus az eredeti angol szöveget adja vissza.

`mik.client.view.uic`

Az alosztály igyekszik kijavítani pár hibát és pótolni egy-két hiányosságot a Swing programozói felületében és a létrejövő felhasználói felületek működésében.

`mik.client.view.uic.KeyTextField`

Olyan beviteli mező (TextField leszármazott), ami csak hexadecimális számjegyek bevitelét engedélyezi, a `createDefaultModel()` metódust átdefiniálja, olyan dokumentummodellt hozva létre, ami csak az említett karakterek beillesztését engedélyezi.

`mik.client.view.uic.Mnemonic`

Az osztály statikus metódusaival egyszerűbben lehet beállítani gyorsbillentyűvel rendelkező gombok szövegét, ugyanis a szöveg részeként (a megfelelő betű előtt) csak jelezni kell egy & jellel, hogy melyik gomb legyen a gyorsbillentyű. Ennek különösen akkor van jelentősége, amikor a gombot több nyelvre is lefordítják, ugyanis ekkor a fordítás részeként (programozói tudás nélkül) kiosztásra kerülhetnek az adott nyelvhez tartozó gyorsbillentyűk.

public static **void** setText(AbstractButton `b`, String `text`)

A `b` gomb feliratát `text`-re állítja, ha a szövegben található gyorsbillentyű, akkor azt beállítja.

public static **JButton** newMnemonicButton(String `text`)
Létrehoz egy új nyomógombot, amelynek feliratát a gyorsbillentyűkkel együtt inicializálja.

public static **JButton** newMnemonicButton(String `text`, Action `act`)
Az előző függvényen hatásán túl még a gombot az `act`-hoz rendeli.

public static **JMenu** newMnemonicMenu(String `text`)
Létrehoz egy új menüt, amelynek feliratát a gyorsbillentyűkkel együtt inicializálja.

public static **JMenuItem** newMnemonicMenuItem(String `text`)
Létrehoz egy új menüelemet, amelynek feliratát a gyorsbillentyűkkel együtt inicializálja.

`mik.client.view.uic.NumberField`

Olyan beviteli mező (TextField leszármazott), ami csak a konstruktorban megadott értékhatárokon belüli intervallum értékeket fogad el. Egy új fókuszfigyelő segítségével, amikor a mező aktivizálódik, megjegyzi az aktuális értéket, majd a fókusz elvesztésekor ezt visszaállítja, ha a bevitt új érték nem megfelelő.

public **«constructor»** NumberField(int `number`, int `minVal`, int `maxVal`)
Ez a konstruktor létrehoz egy intervallum-ellenőrzött beviteli mezőt, `number` kezdőértékkel és `minVal-maxVal` értékhatárokkal (inkluzíve).

`mik.client.view.uic.PasswordField`

Olyan jelszóbeviteli mező, ami ha a billentyűzet segítségével kap fókuszt (pl. gyorsbillentyű vagy a tab billentyűvel való „lépegetés”), akkor kijelöli az egész jelszót, így ha navigálás nélkül kezd el a felhasználó gépelni, akkor a szokásos viselkedésnek megfelelően, ezzel teljesen felülírja a régi adatot.

A megvalósítás részletei a `TextField` megvalósításával egyezők (ld. később).

`mik.client.view.uic.Table`

Olyan JTable leszármazott, ami a konstruktorában beállítja az általunk használt táblázatok kívánt kinézetét és viselkedését, nevezetesen az egysoros kijelölést, valamint azt, hogy a tab és shift+tab billentyűk hagyományos módon a fókuszváltást idézzék elő és ne a táblázaton belül navigáljanak.

mik.client.view.uic.TableSorter

Speciális (a Java on-line dokumentációjából másolt) táblamodell, ami lehetővé teszi, hogy a felhasználó a címsor oszlopelemeinek megkattintásával változtassa a sorok rendezettségét. Sajnos a nyelv készítőinek sem sikerült ezt a megoldást a lehető legkielégítőbben kidolgozni, pl. a kijelölt sor elveszik a rendezettség megváltoztatásakor. A korrekt megoldást a Java nyelv 1.6-os verziója már beépítve tartalmazza, azonban ez a szakdolgozatom írásakor még éppen csak megjelent, számos kliensgépen (és így sok SecMS felhasználó számára) nem érhető el.

mik.client.view.uic.TextField

Olyan szövegbeviteli mező, ami ha a billentyűzet segítségével kap fókuszt (pl. gyorsbillentyű vagy a tab billentyűvel való „lépegetés"), akkor kijelöli az egész szöveget, így ha navigálás nélkül kezd el a felhasználó gépelni, akkor a szokásos viselkedésnek megfelelően, ezzel teljesen felülírja a régi adatot.

Mindezt úgy érjük el, hogy a felülbírált konstruktorokban meghívunk egy `initListener()` nevű segédfüggvényt, ami engedélyezi az egér eseményeinek figyelését (és a figyelő beállít egy jelzőbitet, ha kattintás történik), illetve egy olyan fókuszkezelőt vezet be, ami aktivizálódáskor kijelöli az egész szöveget (ha nem jelez a jelzőbit korábbi kattintást).

mik.client.view.uic.TimeJLabel

A JLabel kiterjesztése, ami a `SetTextErr` és `SetTextOK` metódusaival lehetővé teszi hiba, illetve információs üzenetek megjelenítését egy adott adott időre, ami után a címke eredeti üzenete lesz ismét látható. Ehhez természetesen a `SetText` utasítást is felül kellett definiálni, de annak szignatúrája nyilván nem változott, ezért csak a felhasználás szempontjából érdekes, új metódusokat mutatjuk be:

public **void** setTextErr(String `errorMessage`, final int `timeout`)

Piros színnel megjeleníti az `errorMessage` által megadott feliratot, de `timeout` ezredmásodperc elteltével újra az eredeti, legutolsó `setText(String text)` hívás által beállított értéket teszi láthatóvá.

public **void** setTextOK(String infoMessage, final int timeout)

Zöld színnel megjeleníti az `errorMessage` által megadott feliratot, de `timeout` ezredmásodperc elteltével újra az eredeti, legutolsó `setText(String text)` hívás által beállított értéket teszi láthatóvá.

3.5. Tesztelés

3.5.1. Automatikus tesztelés

Minden komolyabb algoritmuson vagy adatszerkezeten alapuló programkódot szükséges automatikusan tesztelhetővé tenni, mert ha a tesztek elegendően sok esetet lefednek, akkor a kód hibás változtatásakor nagy valószínűséggel valamelyik teszt felhívja majd a hibára a figyelmünket, így a hibákra hamarabb és kisebb hatókörben fény derülhet.

Ugyanakkor felhasználói interfészeket, illetve hálózati tevékenységet végző kódokat jóval nehezebb automatikusan tesztelni, ezért a SecMS esetében automatikus tesztelést a kriptográfiai algoritmusokat megvalósító Secms Commons csomagnál végzünk a JUnit[11] rendszer segítségével.

3.5.2. Kitűzött tesztfolyamat

A kliensprogramot (és ezáltal a szervert) egy előre meghatározott tesztfolyamattal ellenőrizzük minden fontosabb mérföldkő után. A szakdolgozatban megvalósított funkciókhoz illeszkedő tesztfolyamat, amit természetesen a szakdolgozat verziójának véglegesítése előtt végig is ellenőriztem, a következő (α, β két különböző elindított kliensprogramot jelöl, célszerű, hogy az egyik appletként, a másik alkalmazásként fusson):

α) Alice nevű felhasználó regisztrálása, majd az érkezett üzenetek panel kiválasztása, jelenleg üres az üzenet lista,

β) azonos felhasználói névvel és jelszóval, azonos/különböző megjelenített névvel a regisztráció újbóli kísérlete hibát kell, hogy adjon,

β) bejelentkezés a még nem regisztrált Bob felhasználóval hibát kell, hogy adjon,

β) Bob regisztrálása, belépés után az új üzenet ablakban minnél többféle nem létező, illetve hibás szintaxisú kulcs letöltése hibát kell, hogy adjon,

β) Alice kulcsának a letöltése,

β) üres üzenet elküldése hibát kell, hogy adjon,

β) nem üres üzenet esetén a panel elhagyása figyelmeztetést kell, hogy adjon,

[11] http://junit.sourceforge.net/

β) üzenet elküldése, majd egy másik üzenet küldése saját magunknak,

α) Alice kliensprogramjában a frissítés gombra megjelenik az új üzenet, válaszadás tesztelése,

β) Bob kliensprogramjában az érkezett üzenetek között kettő üzenet kell, hogy legyen, amiből a saját magunk által küldött zöld. Ellenőrizzük a táblázat rendezhetőségét, az elemek közti váltogatást!

β) A nyomkövetés menü tárolt kulcsok menüpontjának hatására megjelenő panelben a nyilvános kulcsok, ujjlenyomatok másolhatók és a vágólapra helyezhetők, a szerver kulcsa piros. A saját (zöld) kulcsunk esetében a titkos kulcs is megjelenik.

β) Jelentkezzünk ki, a szerver adatbázisából kézzel töröljük ki az Alice nevű felhasználót.

β) Bobként újra bejelentkezve a üzenetek listájában Alice válasz üzenete már (Alice kulcsának) hiányában nem jeleníthető meg, de ennek ellenére kiválasztásakor, rendezéskor a program logikusan viselkedik.

3.5.3. Felhasználók bevonásával

Amikor a SecMS első már demonstrálható kliensváltozata, illetve szervere elkészült, akkor egy szemináriumon keretében az ELTECRYPT kutatócsoport tagjai számára tartottunk egy bemutatót, ami előtt a felhasználóknak a szoftver használatához kapcsolódó oktatást nem adtunk, a rendszernek csak a feladatát és kriptográfiai hátterét ismerték, a felhasználói felület használatát az intuíciójukra bíztuk.

Az ekkor megfigyelt nehézségek, illetve elhangzott kérdések, panaszok sokat segítettek a felület egyértelműségének növelésében. Pl. a speciális kulcsok néven túl színnel való megkülönböztetése, a regisztrációs gomb pontos működése és a különböző kulcsadatok vágólapra való helyezésének lehetősége mind olyan ötletek, amik a szemináriumon születtek.

4. fejezet

A CD melléklet tartalma

- `secms-protocol.pdf`: a protokoll angol nyelvű leírása
- `client-src.zip`: a kliensprogram forrása tömörítve
- `server-src.zip`: a szerverprogram forrása tömörítve
- `common-src.zip`: a SecMS Commons forrása tömörítve
- `client/`: a kliensprogram futtatásra kész állapotban, kibontva
- `server/`: a szerverprogram futtatásra kész állapotban, kibontva
- `common/`: a SecMS Commons felhasználásra készen, kibontva
- `secms-cd.md5`: a CD-n lévő fájlok MD5 hibafelismerő kódjai

Az ellenőrizhetőség kedvéért a `client-src.zip`, `server-src.zip` és `common-src.zip` fájlok SHA1, illetve MD5 ellenőrzőösszegeit itt is megadom:

MD5:

```
e92988cf111e86bd489e436772b6ce4e  client-src.zip
bcf37582ecbe222f4c0557dc7826cefa  common-src.zip
1a0a8533ba12841f12273f5d9ceea1e1  server-src.zip
```

SHA1:

```
473b4c5a0987a8305eb600fc189d78642a81da08  client-src.zip
3de988573995f16078eb341fa8dbd82069830e0b  common-src.zip
017bea9f5d3a89c46d985c97614cfaedc295f6b3  server-src.zip
```

Irodalomjegyzék

[OpenPGP] 1.3.7, 1.5.1

OpenPGP Message Format
Jon Callas, Lutz Donnerhacke, Hal Finney, Rodney Thayer
http://www.ietf.org/rfc/rfc2440.txt

[FIPS 186-2] 1.5.1

Digital Signature Standard
http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

[DSA] 1.5.1

Digital Signature Algorithm a Wikipédiában
http://en.wikipedia.org/wiki/Digital_Signature_Algorithm

[WYY SHA1] 1.5.3

Finding Collisions in the Full SHA-1
Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu
http://www.infosec.sdu.edu.cn/paper/sha1-crypto-auth-new-2-yao.pdf

[WFLY RIPE] 1.5.3

Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD
Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu
http://eprint.iacr.org/2004/199.pdf

[FluManSha] 1.5.2

Weaknesses in the Key Scheduling Algorithm of RC4
Selected Areas in Cryptography 2001, pp1-24
Scott R. Fluhrer, Itsik Mantin, Adi Shamir

http://www.wisdom.weizmann.ac.il/~itsik/RC4/Papers/Rc4_ksa.ps

[DDFriFuIm] 1.6.1

Future Imperfect
David D. Friedman
http://patrifriedman.com/prose-others/fi/commented/Future_Imperfect.html

[XMPP] 1.6.3

Extensible Messaging and Presence Protocol
Jabber Software Foundation
http://www.ietf.org/rfc/rfc3920.txt
http://www.ietf.org/rfc/rfc3921.txt

[XMPP E2E] 1.6.3

End-to-end signing and object encryption for XMPP
Jabber Software Foundation
http://www.ietf.org/rfc/rfc3923.txt

[TLSv1] 1.6.3

The TLS Protocol
T. Dierks, C. Allen
http://www.ietf.org/rfc/rfc2246.txt

[S/MIME] 1.6.3

Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification
Ramsdell, B., Ed.
http://www.ietf.org/rfc/rfc3851.txt

[OTR] 1.6.3

Off-the-Record Messaging Protocol version 2
Eric Brewer, Nikita Borisov, Ian Goldberg
http://www.cypherpunks.ca/otr/Protocol-v2-3.0.0.html

[SECMS] 3.3.4, 3.4.2

SecMS Protocol Description
Péter Ligeti, Zsolt Balázs Németh, Gergely Riskó
http://sourceforge.net/projects/secms