Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Hosszútávú archiválás elektronikus aláírással

Szerző:

Kollár Balázs

Konzulensek:

Szigeti Szabolcs, Krasznay Csaba

2006. május 17.

1. Diplomaterv feladatkiírás és nyilatkozat

Az e-közigazgatás, e-kormányzat, e-business rohamos térhódítása szükségessé teszi az elektronikus formában létező információ hosszú távú, megbízható, hitelesíthető archiválását. Az archiválás egyik fontos feladata az adathordozók és a rajtuk tárolt adatok fizikai megőrzése, amely már a jelenlegi technológiákkal is megoldott feladatnak tekinthető. Azonban ugyanilyen fontos kérdés az információ hitelességének és sértetlenségének megőrzése, azaz annak biztosítása, hogy az elektronikusan aláírt információ sértetlensége hosszú távon is megőrizhető és bizonyítható legyen, abban az esetben is, ha az eredeti aláíró, hitelesítés-szolgáltató már nem létezik, vagy az eredetileg használt aláíró algoritmus idő közben elavul.

Elvégzendő feladatok:

1. Tekintse át az elektronikus archiválással kapcsolatos lehetőségeket, a feladatokat, és a jelenlegi megoldásokat.
2. Dolgozzon ki megoldást a hosszú távú archiválás megoldására.
3. A megoldás működőképességének bemutatására készítse el az archiváló rendszer főbb elemeit, és értékelje az elvégzett munkát.

Alulírott Kollár Balázs, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem.

Sajátkezű aláírás

Tartalom

2. Kivonat

A közelmúltban a Magyar Elektronikus Aláírási Szövetség (MELASZ) elkészítette a digitális aláírások formátumára vonatkozó közös ajánlását, mely nemzetközi szabványok [1],[2],[9] alapján ad útmutatást a hazai fejlesztőknek. Ez az ajánlás mérföldkőnek tekinthető az elektronikus iratkezelés hazai elterjedésének szempontjából, amely számos előnye ellenére még várat magára, habár a jogi szabályozás már 2001 óta lehetővé teszi a váltást. Az ajánlás megteremti az alapját az elektronikus iratkezelésre épülő termékek és szolgáltatások hazai piacának. Ugyanis az intézmények iratkezelésük megszervezésekor szabványos felületen kapcsolódó alkalmazások közül válogathatnak, melyek egymást kiegészítve komplex iratkezelő rendszerek kialakítását teszik lehetővé.

Kutatásaim célja kezdetben a PKI rendszerek és a digitális aláírás nemzetközi szabványainak beható tanulmányozása, ezt követően nemzeti megvalósítások, valamint a hazai jogi szabályozás megismerése volt. Később megvizsgáltam a gyakorlati felhasználás lehetőségeit, tájékozódtam a már megvalósult alkalmazásokról.

Vizsgálódásaim eredményeként célul tűztem ki egy saját alkalmazás elkészítését, amely képes a MELASZ ajánlásának megfelelő dokumentumok kezelésére, és amelyre később összetett szolgáltatásokat lehet építeni, különös tekintettel az elektronikus archiválásra. Követtem a korszerű szoftver-technológiai irányvonalakat, így az alkalmazás Web-alapú, több-felhasználós, többrétegű, relációs adatbázisra és XML állományokra épül, hardware platform független és objektum-orientált.

A szoftvert egy éve fejlesztem, mely idő alatt elkészült minden kritikus funkciója.

Az aláírás és állomány fogalmakat gyakran szinonimaként fogom használni, mivel az XML állomány általában tartalmazza az aláírt dokumentumot is.

3. Abstract

In recent past the Hungarian Association for Electronic Signature (MELASZ) completed its paper describing the format of digital signatures, which gives guidance for software developers based on international standards [1],[2],[9]. This paper can be a milestone in the spreading of electronic document management systems in Hungary, which, despite of its numerous advantages and its legal support, is still awaited. The paper provides the technological base for products and services on the electronic document management market in the following way. Institutions replanning their document management will be able to choose from a palette of applications communicating on standard interfaces.

At the beginning the aim of my investigations was to review the international standards of PKI systems and digital signatures, followed by the examination of foreign applications, and the domestic legal regulations. Later on I analysed the current and possible future applications.

As a result of my investigation I put the aim of developing an application, which is able to manage MELASZ-formatted documents, and later can serve as a base for complex services, especially for electronic archiving applications. Following up-to-date software technology trends, the application is web-based, multi-user, multilayered, stores data in relational database and XML files, hardware platform independent, and object oriented.

I'm developing the system since February 2005, so the critical functions are completed.

4. Bevezetés

4.1. A feladat értelmezése

A fejlesztés célja egy olyan alkalmazás létrehozása, amely képes a MELASZ[1] ajánlásnak megfelelő aláírás típusokkal kapcsolatos feladatok elvégzésére, különös tekintettel az archiválására. Környezetét és interakcióit tekintve a rendszer lehet:

- egy vállalati informatikai környezet egyik belső kiszolgálója. Ez esetben nagy igény mutatkozhat egy másik, például workflow rendszerrel történő szoros összekapcsolásra.
- egy nyilvános hálózaton elérhető, sok magánszemélyt és kisvállalkozást kiszolgáló szolgáltató alaprendszere. Ez esetben elengedhetetlen egy interaktív felület, amelyen a felhasználók követhetik az állományaik állapotát, valamint újakat tölthetnek fel. Továbbá praktikus lehet, ha a rendszer a feltöltött dokumentumokat könnyen letölthetővé teszi a felhasználóinak, így az támogathatja a felhasználók biztonsági mentéssel kapcsolatos tevékenységét is.

Fontos értelmezési kérdés, hogy a rendszer a dokumentumok tárolását és feldolgozását összetartozó, vagy két külön feladatként kezeli. Elképzelhető olyan környezet is, amikor már üzemel egy dokumentumkezelő rendszer, amely már üzembiztosan megoldja a tárolással kapcsolatos problémákat. Ekkor az archiváló rendszer egyfajta átfolyó rendszerben működne. A bemenetén megkapná az állományokat (például aláírt szerződéseket), a kimenetén pedig (valamekkora késleltetéssel) továbbadná azokat a már üzemelő dokumentumkezelő rendszernek.

4.2. A tervezés célja

A mostani tervező munka a rendszer újratervezését célozza. Ehhez felhasználom az elmúlt egy éves fejlesztés során szerzett tapasztalatokat,

[1] A MELASZ a hazai elektronikus aláíró alkalmazást fejlesztő cégeket tömörítő civil szervezet.

valamint az időközben megszerzett vállalati tapasztalataimat. A tervezés végén egy olyan átgondolt, rugalmas rendszer terve készül el, amely széles körben, különböző felépítésű rendszerkörnyezetekben is használható, beépíthető.

4.3. A feladat indokoltsága

A feladat indokoltságát és aktualitását több tényező együttesen adja.

A nemzetközi technológiai vonalon megtörtént az XML aláírás szabványosítása [1], majd a különféle gyártók beépítették szoftver-fejlesztő környezeteikbe ennek támogatását. Így mind a Java, mind a .NET platformon rendelkezésre áll szabványos XML aláírást kezelő csomag.

Az európai szabványosítási folyamat tovább ment ennél, és elkészült az XML aláírás kiterjesztése, az XAdES [2]. Az Európai Unió továbbá irányelvet adott ki, amely szerint a tagállamok közigazgatási rendszereiben biztosítani kell az elektronikus ügyintézést.

Hazai jogi vonalon az Országgyűlés 2001-ben megalkotta az elektronikus aláírásról szóló törvényt [3] (Eat.), majd 2004-ben elfogadta annak módosítását [4], amelyben többek között szabályozza az archiválás-szolgáltatók tevékenységét. Több kiegészítő rendelet is született, például a papíralapú dokumentumok elektronikus megőrzésérének feltételeiről. A szabályozási folyamat következő lépcsője az új Közigazgatási eljárásról szóló törvény (KET), amely kimondja, hogy a hivatalok ügyfelei online kapcsolatba léphetnek az ügyeiket intéző szervekkel egy központi rendszeren keresztül [5]. A központi rendszer az Ügyfélkapu, amelyen számos ügyet lehetne intézni elektronikus aláírással, de jelenleg sajnos csak a regisztrációhoz használható. Azonban vannak budapesti kerületi pozitív példák, ahol az elektronikus aláírással iratot lehet hitelesíteni.

Hazai technológiai vonalon az aláíró szoftverek fejlesztői és a hitelesítés-szolgáltatók együttesen dolgozták ki a XAdES szabvány hazai használatát szabályozó ajánlást a termékek jobb együttműködése érdekében. Ez a MELASZ ajánlás [6]. A diplomaterv feladatom egy olyan rendszer megtervezése, mely ennek az ajánlásnak megfelelően működik, így összekapcsolható más hazai szoftverekkel.

4.4. A diplomaterv felépítése

A diplomaterv felépítése igyekszik követni a megelőző fél éves kutató és egy éves fejlesztő munka időbeliségét.

A munkám a hetedik féléves önálló laboratórium keretében kezdődött, így vele párhuzamosan hallgattam az Adatbiztonság című tárgyat. Ez nagyban megkönnyítette a kutatómunkát, mivel az adatbiztonság tárgyból megszerzett matematikai-elméleti alapok fényében a gyakorlatban alkalmazott módszerek és szabványok könnyen megérthetők. A digitális aláírás és a dokumentum- és iratkezelés jogi hátterét is ebben a félévben ismertem meg. Az „előzmények és a kutatómunka ismertetése” című fejezet 6.1, 6.2, 6.3-ik alfejezetei ezt a két témakört ismertetik.

A nyolcadik félévben kaptam feladatul egy archiváló rendszer kifejlesztését. A munkát néhány kísérleti „pilot” feladattal kezdtem. Ezek segítségével derítettem fel, hogy a választott – Java – környezetben hogyan lehet kisebb részfeladatokat kezelni, megvalósítani. Ezeket a kísérleteket írja le a 6.4-ik alfejezet.

A nyolcadik félév végére elkészült a rendszer egy előzetes változata, majd a kilencedik félév közepére – a novemberi TDK konferenciára – már egy alapfunkcióiban teljes változattal készültem. A mostani tavaszi félév a rendszer átgondolt újratervezéséről szól, amely a 6.5-ik fejezet következtetéseinek felhasználásával a 7. fejezetben olvasható.

A 8.-ik fejezet a tervek megvalósult részeit mutatja be, majd a 9.-ik fejezetben értékelem a rendszert.

5. A feladatkiírás pontosítása és részletes elemzése

Értelmezésem szerint az aláírt állományok a felhasználói oldalról jutnak a rendszerbe valamilyen továbbító közegen keresztül. A rendszer rendszerezi, megőrzi az állományokat, továbbá megfelelő időpontokban műveleteket hajt rajtuk végre.

5.1. Használati esetek

A rendszer használati eset (use case) diagramja az ***első mellékletben*** látható. A rendszerrel felhasználók és rendszeradminisztrátorok kerülhetnek kapcsolatba.

A *felhasználók* elsődleges célja, hogy feltöltsék az állományaikat a rendszerbe. A későbbiekben ezeket letölthetik. A hiteles archiváláshoz szükség van a felhasználók tanúsítványaira, ezért ezeket is feltöltik.

A *rendszer adminisztrátorai* kezelik a paramétereket és a felhasználókat, valamint az esetleges feldolgozási hibák okát megkeresik, és az akadályokat elhárítják. Ezt a tevékenységet a napló támogatja.

5.2. Entitások

A rendszerben az állományokon kívül más entitásokat is kezelni kell. Szükség van *felhasználók* kezelésére, hogy a különböző felhasználóktól érkező állományokat egymástól logikailag el lehessen különíteni.

Szükséges továbbá a felhasználók nyilvános kulcsait tartalmazó *tanúsítványok* tárolása is, mivel csak ezekkel lehetséges a feltöltött állományokon szereplő aláírások ellenőrzése. Mivel a tanúsítványokon is szerepel aláírás, amelyet ellenőrizni kell – a kibocsátó hitelesítés-szolgáltató aláírása – ezért a hitelesítés-szolgáltatók tanúsítványait is tárolni kell, és így tovább a gyökér hitelesítés-szolgáltatókkal bezárólag.

A tanúsítványokat kibocsátó szolgáltatók visszavonási listáit is tárolnia kell a rendszernek sok évre visszamenőleg. Így azokat rendszeresen be is kell szereznie. Azért van szükség a régi visszavonási listák tárolására – nem csak a legfrissebbére – mert a szolgáltatók az aktuális listájukban csak

a még le nem járt érvényességi idejű tanúsítványok sorszámait adják közre, a régieket nem.

A nyilvános kulcsú infrastruktúrában a tanúsítványok tulajdonosának, kibocsátójának gyűjtő neve „principal”. Érdemes lehet ezt a fogalmat is megjelölni entitásként.

5.3. Egy állomány életciklusa

Egy állomány vagy a felhasználónál keletkezik, vagy kapja azt valahonnan, például elektronikus számlaként. A felhasználó felelőssége, hogy az állományai bizonyos belépési feltételeknek megfeleljenek. Ezeket a feltételeket a MELASZ ajánlás rögzíti, így a hazai digitális aláírás készítő alkalmazások a rendszer számára megfelelő bemenetet állítanak elő. A felhasználó tehát feltölti az állományát a rendszerbe.

A rendszer az állományok fogadását követően azonnal *fogadáskori ellenőrzést* végez, amely biztosítja, ellenőrzi a további lépések előfeltételeit. Az úgynevezett kivárási idő letelte után a rendszer beszerzi a szükséges szolgáltatói visszavonási listákat és tanúsítványokat, majd ezek ismeretében újabb ellenőrzést, ún. *kezdeti ellenőrzést* hajt végre az aláíráson. Amennyiben ezen a második ellenőrzésen is érvényesnek látszik az aláírás, a rendszer csatolja a későbbi ellenőrzéshez szükséges adatokat az aláíráshoz, és érvényesnek fogadja el azt.

A rendszer a későbbiekben *archiválási lépést* is végezhet, amelynek során egy archív időbélyeget tesz az aláírásra. Archív időbélyeget többször is lehet az állományon elhelyezni, akár más-más kriptográfiai algoritmussal is. Az időbélyeget egy ún. megbízható időbélyegzés-szolgáltatótól szerzi be, aki az időpecsétet a saját titkos kulcsával aláírja, így vállal garanciát annak pontosságára.

A *fogadáskori ellenőrzés* lépései közül a legfontosabbak a következők.

- Aláírás időbélyegének ellenőrzése.
- Aláíró tanúsítvány meglétének ellenőrzése.
- Visszavonási listákra mutató hivatkozások ellenőrzése.

A *kezdeti ellenőrzés* lépései a következők.

- Az aláírás lánc érvényességének ellenőrzése a megszerzett visszavonási listák alapján.

5.4. Technikai követelmények

A rendszer a következő technikai követelményeket elégíti ki.

- Lehetővé teszi több felhasználó párhuzamos, független elérését.
- A felhasználóknak hitelesíteni kell magukat ahhoz, hogy a funkcióit használják.
- Szolgáltatásai változatos felületeken keresztül érhetők el.
- Külön üzemeltető felülettel rendelkezik.
- Naplózza a működését.
- Adatait adatbázisban tárolja, melyhez külső adatbázis szervert használ.

6. Előzmények és a kutatómunka ismertetése

2004. év őszén kezdtem digitális aláírásokkal foglalkozni az önálló labor keretében. Kutatásaim célja kezdetben a PKI technológia, és az erre épülő rendszerek megismerése volt. Ezután kezdtem bele a digitális aláírás nemzetközi szabványainak beható tanulmányozásába, ezt követően nemzeti megvalósítások, valamint a hazai jogi szabályozás megismerése volt a feladatom. Később megvizsgáltam a megszerzett ismeretek hazai gyakorlati felhasználásának lehetőségeit, tájékozódtam a már megvalósult alkalmazásokról.

Az archiválással kapcsolatos konkrét feladat önálló labor témaként született meg 2005 tavaszán. A megelőző őszi félévben összegyűjtött ismereteket használtam fel az első négy hónapos fejlesztés során, hogy felállíthassam a rendszer egy korai, kísérleti formáját, amelyen már körvonalazódtak a jövőbeni funkciók. Ekkor ismerkedtem meg a szerveroldali alkalmazások fejlesztésének mikéntjével, így ez az első verzió még számos, később zsákutcának minősült megoldást tartalmazott, például nem használt adatbázis-kezelőt. Az ilyen egyszerűsítések célja az volt, hogy az általános programozás technológiai feladatok helyett az XML és a digitális aláírás technológiákra koncentrálhassak. A félév végére a rendszer már képes volt fogadni az XML aláírásokat, és kisebb feldolgozásokat helyesen végrehajtani azokon. Így például helyesen hajtotta végre az XML kanonizálást, valamint az időbélyeg kérést.

A következő fejlesztési ciklus 2005 őszén következett. A novemberi TDK konferenciára az alkalmazás jelentős fejlődésen ment keresztül. Szoftvertechnológiai oldalról elkezdtem a már terhes kezdetleges megoldásokat robosztusakra cserélni, például egy adatbázis szervert kapcsoltam a rendszerhez, ezzel kiváltottam a korábbi objektumsorosítás, fájlmentés párost. Működési oldalról a fejlődés úgy jelent meg, hogy a kis elemi feldolgozásokat a MELASZ ajánlásnak megfelelő folyamatokká szerveztem. Így a rendszer által kezelt aláírás állományok életciklusa egyszerűvé és világossá vált, megjelentek a rendszer magas szintű funkciói.

A mostani – 2006-os – tavaszi félévben a következő célokat tűztem ki:

- A rendszert újra kell gondolni, és a kezdeti időkből megmaradt gyenge implementációs megoldásokat korszerűekre kell cserélni.
- A rendszert ki kell egészíteni egy XML WebService felülettel, amelynek eredményeképp a rendszer jól illeszthetővé válik más alkalmazásokhoz.
- A rendszer biztonsági megoldásait is át kell gondolni, és az implementáció szintjén korszerű, szép megoldással kell megvalósítani. A szerver-kliens kommunikációt biztonságos HTTP fölé kell áthelyezni.
- A rendszert minél több aláírás készítő szoftver állományaival kell tesztelni.
- El kell készíteni az üzemeltető felületet, amelyen látható a rendszer állapota, és esemény naplója. Az üzemeltető felületen továbbá kezelhetők a felhasználók, az állományok, a tanúsítványok és a visszavonási listák.

6.1. A digitális aláírás matematikai háttere

A digitális aláírás gyakorlati használhatóságához elengedhetetlen, hogy megbízhatóságát tudományos tényekkel lehessen alátámasztani. Mivel a módszer alapja matematikai kutatások eredménye, ez könnyen megtehető. Bemutatom az aláírás ma elterjedt módszerének elemeit, nevezetesen a lenyomatképző függvényeket, és az aszimmetrikus kódolást. E kettő lépésből áll az ún. „Hash and Sign” módszer, amelyre a mai implementációk épülnek, és amelyet az elfogadott [6] MELASZ szabvány is javasol.

6.1.1. Lenyomatképző függvények

A lenyomatképző függvények nevüket az ujjlenyomat fogalma után kapták. Ha találunk egy ujjlenyomatot, valószínűsíthetjük, hogy az pontosan egy emberhez tartozik, mivel két embernek nincs hasonló ujjlenyomata, de legalábbis valószínűtlen hogy ilyen párost találnánk. Továbbá ha van tippünk, hogy kié lehet az ujjlenyomat, azt gyorsan le is ellenőrizhetjük egy nyilvántartás vagy az illető segítségével.

A lenyomatképző függvények ugyanerre képesek. A lenyomat esetükben valamiféle szám vagy számsor, a lenyomat tulajdonosa pedig szintén egy számsor, ami a lenyomatnál sokkal hosszabb is lehet. A digitális világban sok mindent leírhatunk számsorokkal, így lenyomatot képezhetünk bármilyen számokkal ábrázolt adathoz, így tipikusan képhez, hanghoz, szerkesztett szöveghez, tömörített fájlhoz. Az ujjlenyomathoz hasonlóan egy erősnek tartott lenyomatképző függvény esetén sem fordulhat elő a gyakorlatban, hogy két különböző számsorhoz ugyanaz a lenyomat tartozzon, sőt, a bemenő számsor egy pici változása is nagymértékű változást eredményez a lenyomatban.

6.1.2. Aszimmetrikus kriptográfia

Az írott információ elrejtésére már igen régen felmerült az igény. Bizonyítékok már hétezer évvel korábbról, az Óegyiptomi Birodalom idejéből is származnak erre vonatkozóan. Az első komolynak tekinthető, ismert rejtjelező módszer azonban négy évezreddel későbbről származik, és a héber ábécéhez készült. Elve az ábécé visszájára fordítása. [10]

Az alapvető feladat - katonai hasonlattal élve - tulajdonképpen az, hogy úgy írjuk le a mondandónkat a szövetségeseinknek, hogy az ellenség ne tudja azt elolvasni, ha véletlenül hozzá érkezne meg az üzenet. Ennek érdekében az üzenetet mi rejtjelezzük, a szövetségesünk pedig dekódolja. Ennek sikeréhez mindkét fél tud egy-egy olyan titkot, amit az ellenség reményük szerint nem tud.

A különbség a szimmetrikus és az aszimmetrikus kriptográfia között abban áll, hogy ez a két titok egyforma, vagy különböző (de természetesen valamilyen módon még mindig összetartozó). Előbbi esetben mindkét fél ugyanazzal a kulccsal végzi a rejtjelezést és a feloldást is, utóbbi esetben viszont mindkét félnek két-két kulcsa van (kulcs pár), az egyik nyilvános, a másikat titokban kell tartania. Egy A fél úgy üzenhet rejtjelezve a másik B félnek, hogy B nyilvános kulcsával rejtjelezve küldi el az üzenetet. B ezt a titkos kulcsával dekódolja, és olvassa is. Az aszimmetria abban áll, hogy a nyilvános és a titkos kulcs különböző, és egyikből a másikat támadó nem tudja rekonstruálni.

Tehát, mindkét fél tud rejtjelezve üzenni a másiknak anélkül, hogy közös titkon osztoznának, amit előre egyeztetniük kellene. A nyilvános kulcsokat valamilyen módon ki kell cserélniük egymás között. A cserét nem szükséges titkosítva lebonyolítani, de a kulcs pár és az állítólagos tulajdonos összetartozóságáról meg kell győződni. Hiszen, ha egy támadó a saját nyilvános kulcsára cserélné a másik fél kulcsát, akkor ellophatja titkos üzeneteinket. A digitális aláírás aszimmetrikus kriptográfiát használ.

6.1.3. Tanúsítványok

Egy nyilvános kulcs és tulajdonosa összetartozását ún. tanúsítványok rögzítik, melyeket különleges szervezetek, a hitelesítés-szolgáltatók bocsátanak ki. Tanúsítványt bármilyen jogi entitás, magánszemély vagy szervezet igényelhet. A kibocsátott tanúsítványok eredetiségét a hitelesítés-szolgáltató a saját digitális aláírásával szavatolja, amelyet ellenőrizve megbizonyosodhatunk a kapott nyilvános kulcs és a tulajdonos összetartozásáról.

Ily módon egy felhasználónak elég a hitelesítés-szolgáltatói tanúsítványokat beszereznie ahhoz, hogy különféle felhasználói tanúsítványokat fogadhasson, és azokban megbízhasson.

6.1.4. A „Hash and Sign” paradigma

Ebben a fejezetben vezetem be a digitális aláírás fogalmát, amely egyrészt a lenyomatképzésre, másrészt az aszimmetrikus rejtjelezés használatának megfordítására épül.

Az első lépésben az aláírandó dokumentumot egy alkalmas lenyomatképző függvényen átengedve megkapjuk a dokumentum lenyomatát, amely az aláírás alapja. A lenyomatképzés azt szavatolja, hogy az aláírás nem egy teljesen másik dokumentumhoz, vagy nem a szóban forgó dokumentum egy kicsit módosított variánsához tartozik. Az első esetben az adja a biztosítékot, hogy két különböző dokumentum lenyomata különböző, a második esetben pedig arra lehet építeni, hogy az alapdokumentum egy pici módosítása is nagy változást okoz annak lenyomatában.

A második lépésben az aláíró fél a dokumentum lenyomatát saját titkos kulcsával rejtjelezi. Ezt hívtam fentebb az aszimmetrikus rejtjelezés megfordításának, hiszen az üzenetrejtjelezéssel ellentétben a titkos kulcsot most rejtjelezésre, nem pedig kikódolásra használjuk fel. Az aszimmetrikus RSA rejtjelező algoritmus esetében ez semmiféle problémát nem okoz. Ez a tulajdonság nem feltétlenül igaz minden aszimmetrikus rejtjelező algoritmusra. Nem minden algoritmussal lehet olyan kulcs párt generálni, amellyel lehet a tulajdonosnak rejtjeles üzeneteket küldeni, valamint digitális aláíró kulcsként is használható.

A digitális aláírás ezzel a két lépéssel el is készíthető. Érdemes megvizsgálni, hogy milyen biztonságot nyújt az ilyen aláírás. Álljon itt néhány szemléltető magyarázat.

Letagadhatatlanság

Az aláíró a lenyomatot rejtjelezi, amit később nem tagadhat le. Az aláírás rábizonyításához a bizonyító fél képzi a dokumentum lenyomatát, majd az aláírást kirejtjelezi az aláíró nyilvános kulcsával. A nyilvános kulcsot az aláíró fél tanúsítványából lehet megbízható módon megtudni. Ha a képzett és a kirejtjelezett lenyomat megegyezik, azzal bizonyított az aláírás ténye, és a tanúsítványkiadás szabályai miatt az aláírásnak jogi súlya lehet.

Hitelesség

Az aláíró személye egyértelmű is. A matematikai módszerek szavatolják, hogy nem lehetséges egy titkos kulcs nélkül eredetinek látszó aláírást létrehozni.

Integritás

Az integritás követelménye arra vonatkozik, hogy a dokumentum észrevétlenül nem módosítható az aláírás után. A követelmény teljesül a lenyomatképző függvény felhasználásával. Ugyanis bármilyen kis módosítás a függvény bemenetén megváltoztatja a kimenetet, így az ellenőrzéskor az eredeti és a képzett lenyomat el fog térni egymástól.

6.2. A jogi környezet

6.2.1. 2001. évi XXXV. törvény és módosítása

A [3] és [4], az úgynevezett Elektronikus aláírás törvény (továbbiakban Eat.) rendelkezik arról, hogy (néhány kivétellel) legalább fokozott biztonságú elektronikus aláírással ellátott elektronikus iratok elfogadhatók ott, ahol jogszabály írásba foglalást ír elő. Ezzel az elektronikus iratok „polgárjogot nyertek” Magyarországon.

Az Eat. kétféle, joghatással bíró elektronikus aláírást definiál:

a) fokozott biztonságú elektronikus aláírás,

b) minősített elektronikus aláírás.

A fokozott biztonságú elektronikus aláírás egyértelműen azonosítja az aláírót, továbbá felismerhetővé teszi, ha a dokumentum tartalma az aláírás elhelyezése óta megváltozott.

A minősített elektronikus aláírás a fokozott biztonságúhoz képest „erősebb”, mivel minősített tanúsítvány tartozik hozzá, és ún. biztonságos aláírás-létrehozó (BALE) eszközzel hozták létre, ennél fogva több joghatás kapcsolódik hozzá.

Az Eat. az elektronikus aláírás gyakorlati használatához négy szolgáltatástípust definiál:

a) elektronikus aláírás hitelesítés-szolgáltatás,

b) időbélyegzés,

c) aláírás-létrehozó eszközön az aláírás-létrehozó adat elhelyezése,

d) elektronikus archiválás-szolgáltatás.

A hitelesítés-szolgáltató tanúsítványokat bocsát ki. A tanúsítvány kiadásának előfeltétele, hogy az igénylő hitelesen igazolja a személyazonosságát. Ettől fogva a tanúsítvány lejáratáig vagy visszavonásáig az igénylő képes a személyazonosságát elektronikusan is igazolni, valamint képes a tanúsítványhoz tartozó egyedi és titkos aláírás-létrehozó eszközzel iratokon elektronikus aláírást elhelyezni.

A hitelesítés-szolgáltató továbbá nyilvántartja, és nyilvánosan elérhetővé teszi a nála kibocsátott tanúsítványok állapotát, díjakat szab a tanúsítvány fenntartásáért, és meghatározza a tanúsítványokkal létrehozott aláírásért vállalt anyagi felelősségének felső határait. A minősített

elektronikus aláírásokhoz jellemzően magasabb értékhatárok tartoznak a fokozott elektronikus aláírásokhoz képest.

Az érvényes tanúsítványok hitelességének megőrzése céljából az aláíró köteles haladéktalanul tájékoztatni a hitelesítés-szolgáltatót, ha adatai megváltoztak, vagy a titkos magánkulcs kitudódott, esetleg elveszett. Ekkor a hitelesítés-szolgáltató a tanúsítvány érvényességét felfüggeszti, esetleg végleg visszavonja, továbbá ennek tényét közzéteszi. Visszavonás történik akkor is, ha a tanúsítvány érvényességi ideje lejár.

Az időbélyegzés-szolgáltató egy dokumentumhoz az aktuális pontos időt rendeli hozzá.

Az archiválás-szolgáltató dokumentumokat, és az elektronikus aláírások hosszú távú ellenőrizhetőségéhez szükséges érvényességi láncot tárolja megbízható módon. Továbbá igazolást adhat ki arról, hogy egy elektronikus aláírás a létrehozás pillanatában érvényes volt.

6.3. A szabványok

6.3.1. X.509 PKI Certificate and CRL profile (RFC 3280)

Az X.509 egy ITU-T szabvány nyilvános kulcsú infrastruktúra számára. Az első változatot 1988-ban adták ki az X.500 szabványhoz kapcsolódódóan. Mivel az X.500 sosem valósult meg teljesen, valamint szigorú hierarchikus modellje nem illett rá a világhálóra, az IETF átdolgozta az X.509-et az internet rugalmas szerkezetéhez illeszkedően, így az X.509 végül IETF ajánlásként vált ismertté.

Az X.509 megszabja az interneten működő nyilvános kulcsú infrastruktúrában használt tanúsítványok és visszavonási listák szerkezetét, valamint ad egy algoritmust a hitelesítési útvonal érvényességének eldöntésére.

Egy X.509 tanúsítvány alapvető mezői a következők:

- Verziója (Version). A tanúsítvány formátumának verziója.
- Sorszáma (Serial Number). A kibocsátóval együtt azonosítja a tanúsítványt.

- Algoritmus azonosító (Algorithm ID). Annak az algoritmusnak a neve, amellyel a kibocsátó aláírta a tanúsítványt.
- Kibocsátó X.500 neve (Issuer). A sorszámmal együtt azonosítja a tanúsítványt.
- Érvényességi idő (Validity).
- Tárgy (Subject). A tanúsítvány tárgyának X.500 neve. Joghatással bíró tanúsítvány esetén az a jogi vagy magánszemély, akihez a tanúsítvány a kulcspárt hozzárendeli.
- A tanúsított nyilvános kulcs (Subject Public Key Info). A tárgyhoz hozzárendelt nyilvános kulcs értéke, és annak az algoritmusnak az azonosítója, amellyel a kulcs használható.
- Aláíró algoritmus (Certificate Signature Algorithm). Annak az algoritmusnak az azonosítója, amellyel a kibocsátó a tanúsítványt aláírja.
- Aláírás (Certificate Signature). A kibocsátó aláírása a tanúsítványon.

Egy X.509 visszavonási lista szerkezete a következő:

- Verziója (Version). A visszavonási lista formátumának verziója.
- Algoritmus azonosító (Algorithm ID). Annak az algoritmusnak a neve, amellyel a kibocsátó aláírta a visszavonási listát.
- Kibocsátó X.500 neve (Issuer). A kibocsátási idővel együtt azonosítja a visszavonási listát.
- Kibocsátási idő (thisUpdate). A visszavonási lista kibocsátási ideje. A kibocsátó nevével együtt azonosítja a visszavonási listát.
- Következő frissítés (nextUpdate). A következő visszavonási lista kiadásának ideje.
- Visszavont tanúsítványok listája (revokedCertificates). A lista minden eleme kötelezően tartalmazza a visszavont tanúsítvány sorszámát a kibocsátónál, a visszavonás dátumát, valamint választható egyéb mezőket, mint például a visszavonás kiváltó okát.

- Aláíró algoritmus (Certificate Signature Algorithm). Annak az algoritmusnak az azonosítója, amellyel a kibocsátó a visszavonási listát aláírja.
- Aláírás (Certificate Signature). A kibocsátó aláírása a visszavonási listán.

6.3.2. XML Signature (RFC 3275)

Miután elkészült a dokumentumhoz az elektronikus aláírás, és összegyűltek az egyéb szükséges információk, egységbe kell zárni őket. Ennek eszköze az XML elektronikus aláírás (XMLDSIG), melyet a [1] dokumentum ír le.

Az XML elektronikus aláírás egy XML dokumentum, melynek formája az alábbi.

```
<Signature ID?>
   <SignedInfo>
     <CanonicalizationMethod/>
     <SignatureMethod/>
     (<Reference URI? >
       (<Transforms>)?
       <DigestMethod>
       <DigestValue>
     </Reference>)+
   </SignedInfo>
   <SignatureValue>
  (<KeyInfo>)?
  (<Object ID?>)*
</Signature>
```

1. ábra – XML aláírás szerkezete

Az aláírás a teljes SignedInfo elem lenyomatának a rejtjelezésével keletkezik, és értéke a SignatureValue elembe kerül. Az aláírás tetszőleges számú dokumentumra vonatkozhat, ezek lenyomata külön-külön Reference elemek alá kerül, egy-egy DigestValue elembe.

Az aláírás hordozhatja az aláírás-ellenőrző adatokat is a KeyInfo elemben. A KeyInfo elem közvetlenül nem kerül aláírásra, mert nem része a SignedInfo elemnek. Lehetőség van azonban egy Reference elemen keresztül közvetett módon bevonni az aláírt elemek halmazába.

A Reference elemek opcionálisan hivatkozhatnak arra a dokumentumra, amelynek a lenyomatát hordozzák sőt, az XML aláírás akár magában is hordozhatja e dokumentumokat egy Object elemben.

Az aláírás elkészítéséhez először a Reference elemeket kell létrehozni. A dokumentumokat a Transforms tagban rögzített transzformációknak kell alávetni, majd a DigestMethod elemben megadott algoritmussal el kell készíteni a lenyomatot, és azt a DigestValue elembe kell tenni. Ezután a SignedInfo elemet kell a CanonicalizationMethod elemben rögzített algoritmussal kanonizálni, majd a SignatureMethod elem szerint alá kell írni, az aláírást pedig a SignatureValue elembe kell tenni.

Az aláírás ellenőrzéséhez először a Reference elemekhez tartozó lenyomatokat kell elkészíteni, majd összevetni az XML aláírásban levőkkel. Ha mindegyik lenyomat helyes, akkor a dokumentumok nem módosultak az aláírás óta. Ezután el kell készíteni a SignedInfo elem kanonizált formáját, majd annak lenyomatát, és össze kell vetni a SignatureValue elem értékének feloldásával. A rejtjelezés feloldását a SignatureMethod szerinti algoritmussal lehet elvégezni az aláírás-ellenőrző adat ismeretében.

6.3.3. XML Advanced Electronic Signature (ETSI TS 101903)

Az XML elektronikus aláírás tartalmára tett eddigi megkötések még nem elegendőek ahhoz, hogy az a gyakorlatban jól használható legyen. További követelmények fogalmazódnak meg, amelyek megoldására különböző elektronikus aláírás formátumokat dolgoztak ki. Ezeket a [2] XAdES szabvány írja le. A meghatározott aláírás-típusok az alábbiak.

1. BES, az alapszintű elektronikus aláírás. A BES tartalmaz egy digitális aláírást, az aláíró tanúsítványát aláírt formában, és egyéb aláírt/nem aláírt opcionális jellemzőket. Ez a formátum megfelel az Eat. fokozott biztonságú aláírásának, hitelesítést és integritásvédelmet biztosít, de letagadható, mivel nem rögzíti az aláírás időpontját, és így nem bizonyítható, hogy az aláíró tanúsítvány érvényes volt az aláírás létrehozásának időpontjában.
2. EPES, az egyértelműen szabályozott elektronikus aláírás. Az EPES egy alapszintű aláírásra épül, és kiegészíti azt egy aláírási szabályzat

azonosítójával, amit kötelező aláírni. Ezáltal egyértelművé teszi az aláírás érvényesítési módját. Az aláírási szabályzat fogalma a dokumentum elején olvasható. Az EPES a BES-hez hasonlóan hiteles, integritásvédett, de letagadható.

3. XAdES-T, az aláírás időpontjával kiegészített elektronikus aláírás. Az XAdES-T az előző két forma egyikét egészíti ki egy (megbízható) időbélyegzés-szolgáltatótól származó időpecséttel. A megbízható időbélyeg egy kezdeti lépést jelent az aláírás hosszú távra szóló érvényességének biztosítására. Az időpecsét egy aláírt dokumentum, mely az időbélyegzés-szolgáltató órája által mutatott időből, és egy dokumentum lenyomatából áll. A lenyomatot az ügyfél küldi el a szolgáltatónak, az aláírás pedig a szolgáltató tanúsítványával történik.
4. XAdES-C, a teljes körű érvényesítő adatokkal kiegészített elektronikus aláírás. Az XAdES-T formához képest hivatkozásokat tartalmaz a teljes érvényességi láncra, és a szükséges tanúsítvány-visszavonási listákra. Ezek beszerzése időigényes lehet, ezért ilyen aláírást létrehozni is időt vesz igénybe. Előnye, hogy az aláírás későbbi érvényesítéséhez minden adat rendelkezésre áll. Erre a típusra akkor lehet szükség, ha az aláíró, és esetleg a hitelesítés-szolgáltató tanúsítványának lejárta után is szükség lesz az aláírás ellenőrzésére. Az XAdES-C aláírást nem lehet „azonnal” létrehozni, mivel annak eldöntése, hogy egy tanúsítvány egy időpillanatban érvényes-e, csak egy bizonyos kivárási idő eltelte után lehetséges teljes bizonyossággal, hiszen például a hitelesítés-szolgáltatónak is idő kell a visszavonási listák frissítéséhez. A kivárási idő az [8] ITKTB ajánlásban az elvárt biztonsági szinttől függően 1-3 nap.
5. XAdES-X, a XAdES-C elektronikus aláírás kibővített változata. Három fajtáját határozták meg.

   A XAdES-X Long nem csak hivatkozásokat tartalmaz az érvényesítő adatokra, hanem magukat az adatokat is magába foglalja. Erre akkor

lehet szükség, ha fennáll a veszélye, hogy a tanúsítvány-láncok és visszavonási listák elvesznek.

   a. A XAdES-X Type 1 formátum a XAdES-C-t egészíti ki egy időbélyeggel, mely a teljes elektronikus aláírásra készül el. Ezzel a típussal kezelhetők azok az esetek, amikor kompromittálódik vagy az aláíró tanúsítványt kibocsátó hitelesítés-szolgáltató, vagy az érvényesítő adatokat aláíró szolgáltató aláíró kulcsa. Ez a forma kombinálható az elsővel, ekkor XAdES-X Long Type 1 a neve.

   b. A XAdES-X Type 2 forma szintén a szolgáltatók kulcsának kompromittálódása ellen véd úgy, hogy az egyes érvényesítő adatokra külön-külön tesz időbélyeget. Ez a forma kombinálható az elsővel, ekkor XAdES-X Long Type 2 a neve.

6. XAdES-A, az archív érvényesítő adatokkal kiegészített elektronikus aláírás. A XAdES-A formátum valamelyik XAdES-X aláírás-fajtára épülhet, és azokra az esetekre biztosítja az elektronikus aláírás hitelességét, integritását és letagadhatatlanságát, amikor az aláírás bármely eleme meggyengült. Ez lehet például az aláírás elkészítéséhez használt algoritmus, kulcs, tanúsítvány. A XAdES-A egy időbélyeggel látja el a teljes aláírást, amelynek kriptográfiai erőssége mindig elég erős lehet a kor technológiai követelményeihez igazodva. Ennek haszna abban van, hogy ha egy korábban – az állományon – alkalmazott algoritmus meggyengül, akkor az állomány hitelessége megvédhető, hiszen az aláírás ellenőrzésekor az összes archív időbélyeget is ellenőrizni kell.

Az Eat. az archiválási-szolgáltatók számára a következőt írja elő:

„[4] Eat. 16/G. § (2) A szolgáltató köteles a Hatóság határozata szerinti, elfogadott kriptográfiai algoritmuson alapuló minősített elektronikus aláírást elhelyezni és minősített szolgáltató által kibocsátott időbélyegzőt elhelyezni vagy elhelyeztetni az érvényességi láncon:

1. a szolgáltatási szabályzatban meghatározott időközönként;
2. a határozatban előírt időpontban.”

Az utóbbi eset például akkor fordulhat elő, ha egy széles körben használt aláíró algoritmusról kiderül, hogy feltörhető.

A [8] dokumentum a XAdES-C formátum használatát javasolja a magyar közigazgatás elektronikus kommunikációjában, ha a hosszú távú letagadhatatlanság követelmény. Ekkor az aláíró alkalmazások legalább a XAdES-EPES formátum létrehozását kell, hogy támogassák, amelyből majd vagy az aláíró, vagy az aláírást továbbító szolgáltatás, vagy az aláírást fogadó/feldolgozó/ellenőrző címzett készíti el a XAdES-C formátumot, amely már tartalmaz minden szükséges információra vonatkozó hivatkozást a hosszú távú letagadhatatlanság biztosításához.

A dokumentum azért nem javasolja a XAdES-X vagy XAdES-A formátum használatát, mert azokra előreláthatólag még egy évtizedig nem lesz szükség, valamint terjedelmük is jóval nagyobb a XAdES-C változaténál. Ha mégis szükség mutatkozna a használatukra, akkor a meglévő XAdES-C aláírásokból készíthető X vagy A típusú XAdES aláírás, mivel minden szükséges hivatkozás része az aláírásnak.

A ***második melléklet*** az XMLDSIG és a különböző XAdES formátumok viszonyát tekinti át.

6.3.4. MELASZ XAdES profil

A Magyar Elektronikus Aláírás Szövetség (MELASZ) 2005. szeptemberében fogadta el az ETSI XAdES szabványra alapuló aláírás profilját.

A profil tartalmazza a XAdES szabvány minden kötelező elemét, a választható elemek jelentős részének használatát azonban nem engedi meg. A szűkítést a magyarországi felhasználói és szolgáltatói szempontok figyelembevételével tették meg. Szűkítették a használható XML elemek, valamint a hivatkozható algoritmusok körét. Ezzel egyszerűsödik a profilnak megfelelő alkalmazások megvalósítása, hiszen kevesebb elem és algoritmus kezelésére kell felkészülni, míg az alkalmazás funkcionalitása ezzel nem csökken.

Kikerültek a használható elemek köréből a következő elemek:

- SignatureProductionPlace,
- SignerRole,

- CommitmentTypeIndication,
- AllDataObjectsTimeStamp,
- IndividualDataObjectsTimeStamp,
- AttributeCertificateRefs,
- AttributeRevocationRefs.

A használható algoritmusok köre is szűkült, így csak a következők használhatók:

- Kanonizálásra csak a megjegyzések nélküli XML kanonizáció,
- aláírásra csak az SHA-1 lenyomatképzéssel kombinált RSA algoritmus,
- lenyomatképzésre pedig csak az SHA-1 algoritmus használható.

Azon transzformációk köre is szűkült, amelyeket az aláírt dokumentumon el lehet végezni:

- Az XPath szűrés, mint transzformáció nem megengedett.
- Az Enveloped Signature Transform, mint transzformáció nem megengedett.
- Az XSLT transzformáció nem megengedett.

6.3.5. Time-Stamp Protocol (RFC 3161)

Az időbélyegek használata kulcsfontosságú az elektronikus aláírásban. Ezek segítségével bizonyítható, hogy egy tetszőleges adat – például egy aláírással ellátott szerződés – a tanúsított időpontban már létezett, így valósítható meg a teljesen letagadhatatlan[2] elektronikus aláírás.

Megbízható időbélyeget az Eat. Törvény szerint megbízható időbélyegzés-szolgáltató állíthat ki. A folyamat szerint a szolgáltatóval szerződött ügyfél összeállít a lebélyegzendő adataiból egy lenyomatot, majd azt egy szabványos [11] kérésbe csomagolva eljuttatja a szolgáltatóhoz.

[2] Bizonyítható az aláírás ténye és az aláírás időpontja is.

A szolgáltató elkészít egy szintén szabványos választ, amelyben a következő elemeket helyezi el:

- Az időbélyegzés alatt álló lenyomat értéket.
- Az időbélyeg sorszámát.
- Az időbélyeg készítésének lehető legpontosabb időpontját.
- Az időpont értékének lehetséges hibáját.
- A saját szolgáltatói titkos kulcsával készített digitális aláírását, amellyel hitelesíti az időbélyeget.
- A saját szolgáltatói nyilvános kulcsát tartalmazó X.509 tanúsítványát, amellyel később az időbélyeg hitelessége ellenőrizhető.

6.4. Szoftver technológiai kísérletek és tapasztalatok

A diplomaterv elkészítését hosszú fejlesztési időszak előzte meg. Ezt az időszakot legjobban a „code and fix paradigma” jellemezte, amely egy hosszú kísérletező-kutató időszaknak tekinthető a diplomaterv szempontjából. Ez idő alatt több részfeladatnak több megoldási módját is kipróbáltam, megtapasztaltam azok előnyeit és hátrányait.

6.4.1. Webes alkalmazások

A hálózat fölött működő alkalmazások koncepciója a hálózatok megjelenésével egyidősnek tekinthető. Amíg a nagygépes és mainframe világban a hálózat fölötti alkalmazásfuttatás (elosztott működés) a rendszer architektúrája miatt mindig is magától értetődő és természetes jelenség volt, addig az asztali számítógépek világában az elosztott alkalmazásfejlesztés csak az elmúlt néhány évben terjedt el.

A koncepció lényege hasonló a nagygép-terminál viszonyhoz, habár a központi gép valamivel többet vár el a felhasználó oldali géptől, mint a régi karakteres (esetleg grafikus) termináloktól volt szokás. A felhasználók tehát a saját mikroszámítógépüket használják a rendszer eléréséhez. A központi kiszolgáló gép a hálózatra kapcsolódik, a felhasználóknak pedig csak a központ hálózati címét kell ismerniük, hogy elérjék a rendszert.

Annak felülete a felhasználó gépén futó web böngészőben jelenik meg, így a rendszer használata semmilyen előzetes telepítést, felkészülést nem

igényel, hiszen web böngésző minden asztali gépen van. A szerver-kliens kommunikáció HTTP protokollon keresztül folyik, az adatbevitel pedig HTML űrlapokon keresztül történik. Ezeknek az űrlapoknak a mezőit tölti ki a felhasználó, ha adatot akar bevinni a rendszerbe, majd elküldi azt a szervernek.

6.4.2. Szervlet technológia alkalmazása

A rendszer fejlesztésével párhuzamosan kezdtem el a Java-alapú, hálózaton működő alkalmazások fejlesztéséhez kifejlesztett szervlet technológia tanulmányozását. A szervlet többféle protokoll fölött is működhet. Én a HTTP fölötti változatot használtam, de megvalósítható bármilyen más TCP/IP fölötti szolgáltatás is szervet alapon.

A HTTP szervlet technológia a következő fontos szolgáltatásokat nyújtja a fejlesztő számára.

- Egyszerű session kezelés. Mivel a HTTP protokoll ún. állapot nélküli kapcsolatokat létesít a kliens és a szerver között, viszont a webes alkalmazásokban szükség van a felhasználó és a szerver közötti ún. munkamenet létesítésére, ezért a legtöbb technológia támogatja ezt. A szervlet technológiában minden munkamenetet egy HttpSession objektum képvisel, amely saját kulcs-érték szervezésű tárolóval rendelkezik. Ebben a tárolóban lehet a bejelentkezett felhasználóhoz tartozó egyedi objektumokat tartani, például a bejelentkezése idejét, vagy webes áruház esetében a kosarát.
- Egyszerű HTTP kezelés. Lehetővé teszi a felhasználói böngészőtől érkező űrlapok paramétereinek könnyű dekódolását, továbbá a kérés és a válasz is egy-egy objektumként érhetők el, amelyek metódusain keresztül jól kezelhető a HTTP kommunikáció minden jellemzője.

A klienstől érkező kéréseket egy dedikált, a szerveren futó metódus szolgálja ki. A technológia nagy előnye, hogy ez a metódus egy Java virtuális gépen fut, így minden további nélkül használható a Java platformra írt összes osztály szervlet környezetből is. A platformnak ez a jellemzője lehetővé teszi azt is, hogy a webes alkalmazáshoz írt Java

alkalmazáslogika – megfelelő tervezés esetén – könnyen használható legyen nem hálózati környezetben is, például egy egyszerű asztali alkalmazásban.

Visszatérve a dedikált metódusra, ebben a függvényben (diszpécser) szokás a kérést a paraméterei alapján átirányítani a megfelelő kiszolgáló metódushoz. Itt több tervezési lehetőség is választható, amely döntés nagyban befolyásolja a rendszer felépítését. A döntés abban nyilvánul meg, hogy a szervlet kiszolgáló metódusa, illetve maga a szervlet osztály mekkora részét tartalmazza az alkalmazáslogikának. Másképp megfogalmazva: az alkalmazáslogika működőképes-e a szervlet nélkül? A gyakorlat azt mutatja, hogy célszerű úgy tervezni a rendszert, hogy a vezérlés és az alkalmazáslogika jól elkülönüljenek.

Az első kísérleteimben a rendszer alkalmazáslogikájának egy része a szervlet osztály metódusaiban kapott helyet. Ugyanitt kapott helyet a felhasználók azonosítását, be- és kijelentkeztetését végző kódrészlet is. Továbbá, a klienstől érkező kérés azonosítását, és a megfelelő kiszolgálóhoz küldését is a szervlet osztály végezte. Ennek eredményeképp a szervlet osztály kódja elérte a nyolcszáz soros méretet, amitől nehezen áttekinthetővé vált.

A megoldást a szervlet technológia lehetőségeinek jobb kihasználása és az előre megírt vezérlő szervlet használata jelentette. A leghíresebb ilyen vezérlő a Jakarta Struts technológia, amely tulajdonképpen az MVC (Model View Controller) paradigma webes környezetbe ültetése. Ebben az elrendezésben a modell (Model) alkalmazási terület sajátosságainak megfelelő egyedi fejlesztés, a nézet (View) a felhasználó böngészőjében megjelenő HTML oldal. A vezérlő (Controller) pedig a következő részekből áll össze:

- Egy előre megírt paraméterezhető Java osztály.
- A fejlesztő által írt paraméterállomány, melyben az egyes web címeket lehet akció osztályokhoz kapcsolni. Például a szerver/Login.do akciót a Login akció kezeli le.
- A fejlesztő által írt ún. akció (Action) osztályok, amelyeknek egyetlen (execute) metódusuk van.

A Struts technológia alkalmazásával kiválthatók a sematikus – switch-case – formájú diszpécser osztályok, ahol általában a HTTP kérések feloldása és továbbítása történik.

Ily módon elválasztottam az alkalmazáslogikát a vezérlő rétegtől, amelynek eredményeként a rendszer hordozhatóbbá és strukturáltabbá vált.

6.4.3. Adatbázis elérése szervletből

[7] 23. fejezete alapján.

A webes alkalmazások legtöbbje az alkalmazási területre jellemző adatokat halmoz fel, amelyeket adatbázisban szokás tárolni. Az adatbázis elérésének módja lényeges kérdés a rendszer tervezése szempontjából, ezért –néhány próbálkozások után – külön kutatást igényelt a helyes eljárás elsajátítása.

Java környezetben adatbázisokat a JDBC technológiával lehet használni. A folyamat lépései általában egy kapcsolat (Connection) objektum megszerzése, egy SQL utasítás (Statement) végrehajtása, majd lekérdezés esetén az eredmény (ResultSet) feldolgozása.

A rendszer teljesítményére nagy hatással van a kapcsolat és az utasítás objektumok létrehozásával eltöltött idő, mivel ezek költséges műveletek. Asztali alkalmazások esetében ez nem okoz problémát, mivel az alkalmazás indulásakor létrejön a kapcsolat objektum, és a programpéldány teljes életciklusa alatt – leállításáig – rendelkezésre áll. A program általában legfeljebb egy adatbázis műveletet hajt végre egyszerre, így egy darab kapcsolat objektum elegendő.

Egy szerver alkalmazásban a helyzet bonyolultabb. Időben párhuzamosan több felhasználó is hajthat végre adatbázis műveletet, amelyek egymástól függetlenek, így könnyen lehetnek konkurensek. Három szempontot kell figyelembe venni a kapcsolat objektumok szervezésével kapcsolatban:

- Létrehozása sok idő, tipikusan egy-két másodperc. Ezalatt hálózati kapcsolatot kell létesíteni a szerverrel, amelynek el kell végezni a felhasználó azonosítását, és több adatstruktúrát is létre kell hoznia.
- Egyet egyszerre csak egy felhasználó használhat.
- Nyitva tartása is költséges az adatbázis szerver oldaláról.

Az egyik megoldás – ha a kapcsolat implementáció támogatja ezt –, hogy a szervlet objektum tartalmaz egy kapcsolat objektumot, majd minden érkező kérés azt használja. Ehhez a kapcsolat implementációnak kell tudnia többszálúan is biztonságosan működni. Ekkor a kérések időben sorban egymás után rendezve futnak le (egyszerre mindig csak egy), ami miatt a várakozási idő könnyen megugorhat. Egy másik megoldás, ha minden belépett felhasználónak saját kapcsolat objektumot hozunk létre, és azt a munkamenetéhez rendeljük. Ennek természetesen az a hátránya, hogy a drágán fenntartott kapcsolat objektum kihasználtsága nagyon alacsony lesz. Hiszen, amíg a felhasználó az előző lekérdezés eredményét nézi a képernyőjén, addig a kapcsolat objektum kihasználatlanul vár.

A legjobb választható lehetőség az ún. kapcsolat készlet (connection pool) használata. A készlet kapcsolat objektumokat tartalmaz, amelyeken a JSP oldalak és a szervletek közösen osztoznak. Minden adatbázis hozzáférés egy kapcsolat objektumot kivesz a készletből, majd használat után visszateszi azt.

Ez a megoldás teljesítőképesség szempontjából nagyon jó, azonban felvet egy problémát. A kapcsolatokat az adatbázis szerver azonosítja, általában felhasználónévvel. A webes alkalmazás felhasználóit ezért lehetséges a felhasználókat az adatbázis szerverbe egyenként felvenni, így kiforrott hozzáférést lehet kialakítani. A kapcsolat készlet használata azonban ezt lehetetlenné teszi, hiszen a kapcsolat objektumokat nem lehet többé felhasználóhoz kötni.

6.4.4. A JSP megjelenítő technológia

A webes alkalmazások megjelenítő felülete a web böngészők által megjelenített HTML oldal, melyeket a szerver generál a felhasználó kéréseinek megfelelően. Ezeknek a válasz oldalak tartalmaznak állandó és változó elemeket is. Az oldalak fej- és lábléce például ritkán, az oldalak középső (tartalmi) része pedig minden kéréskor megváltozik.

A HTML kód generálható közvetlenül a szervlet osztály kódjából is a PrintWriter osztály metódusain keresztül. Ennek a megoldásnak számos hátránya van, melyek közül a nehéz karbantarthatóság, a rossz skálázhatóság, és a körülményes fejlesztés a legfontosabbak.

A JSP technológia speciális tagokkal tűzdelt HTML oldalakat használ a megjelenítendő oldal előállításához. A tagok paraméteres Java kódrészletek, amelyek számos feladatot el tudnak végezni az oldal előállításakor a hagyományos vezérlő szerkezetektől az adatbázis lekérdezéseken át a különböző – Java Gyűjtemény Keretrendszerbe illeszkedő – adatszerkezetek bejárásig.

A felhasználói kérések feldolgozásának folyamatában a JSP oldalaknak vagy a szervlet továbbítja a kérést, vagy a felhasználó közvetlenül hívja meg őket a szervletek kihagyásával. Mindkét módon lehet jó webes alkalmazást készíteni, mivel a technológia rendkívül rugalmas. Valójában a JSP oldalakból egy fordító Java kódot generál, és ez a program fut le, amikor a JSP oldalhoz kerül a futás joga.

A JSP oldalak fejlesztése viszonylag kevés programozói tudást igényel, valamint jól elkülöníthető a szervlet fejlesztésétől – amennyiben a rendszer tervezésekor a két réteg közötti kapcsolat jól meghatározott.

6.4.5. XML kezelés

Az XML leírónyelv a XAdES alapja, így minden aláírás ilyen formában érkezik, készül, és tárolódik a fájlrendszerben.

Kezdetben a World Wide Web Consortium (W3C) által specifikált Document Object Model (DOM) implementációját használtam. Ez a csomag a specifikáció univerzalitása, nyelvfüggetlensége miatt – hasonlóan az OMG CORBA modellhez – nem használja ki a Java speciális előnyeit, amelyek a gyors, hibamentes kódolást segítik elő. Java környezetben használata nehézkes, idegennek hat. Aránytalanul sok időt vitt el az XML-kezelő részek megírása, melyek épp az alkalmazás központi elemét képezik. Ezért másik API-t kerestem.

Végül a JDOM API-t választottam ki, amelynek kialakításánál kihasználták a Java nyelv minden lehetőségét. Az API használja a Java Gyűjtemény Keretrendszert (Collections), és az 1.5-ös Java verzióban megjelenő típusos gyűjteményeket. Használatával jól érthető, jól olvasható, tömör, biztonságos, lényegretörő kódot lehet írni a W3C DOM-hoz képest töredék idő alatt.

Példaként álljon itt egy sokszor idézett kódrészlet. A feladat egy új dokumentum létrehozása egy gyökérelemmel és abban egy szöveges elemmel. A különbség magáért beszél.

W3C DOM megoldás:

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element root = doc.createElement("root");
Text text = doc.createText("This is the root");
root.appendChild(text);
doc.appendChild(root);
```

JDOM megoldás:

```
Document doc = new Document(
    new Element("root").setText("This is the root"));
```

6.4.6. Időbélyeg kérés

A rendszer működése során többször is előfordul, hogy XML állományokra időbélyeget kell, kérjen. A legnehezebb feladat az időbélyegzésre kerülő kriptográfiai lenyomat helyes előállítása, hiszen egyetlen bit eltérés az elkészítés során is teljesen megváltoztatja a kapott időbélyeget.

A bélyegzés előtt az XML fa megfelelő ágait a megfelelő sorrendben kanonizálni kell, majd a kanonizálással kapott bájtsorozatot át kell adni a választott lenyomatképző algoritmusnak. Az időbélyeg fajtája határozza meg, hogy az XML fa mely ágait kell feldolgozni. Az így kapott lenyomatot szabványos kérés formájában kell az időbélyegzés-szolgáltatónak elküldeni, aki visszaküldi az időbélyeget.

Ezt az adatot azután megfelelő XML elemben kell elhelyezni, és további 'Include' elemeket kell melléhelyezni, amelyek rögzítik az időbélyeg által védett elemek listáját.

7. Logikai rendszerterv

7.1. Adatmodell

A ***harmadik mellékletben*** láthatók a logikai adatmodellt ábrázoló diagramok. A következő felsorolásban az egyedek kulcs mezőit folytonos vonallal, az idegen kulcsokat szaggatott vonallal húztam alá.

7.1.1. Alkalmazási területhez kapcsolódó adatok

Az alkalmazási terület modellezése során feltártam a szükséges entitásokat, és a közöttük lévő kapcsolatokat. A rendszer szempontjából legfontosabb kezelendő fogalom az aláírás állomány, amely tartalmazhatja az aláírt dokumentumot is. Az adatmodellben „sigs” néven szerepelnek az **aláírás állományok**, és a következő mezőkkel bírnak:

- Azonosító. Egyedi azonosító mező, amely az entitásokat egyértelműen azonosítja.
- Tulajdonos felhasználó. Az állomány tulajdonosának felhasználói azonosítója, idegen kulcs.
- Állomány neve. Az állomány eredeti fájlneve.
- Státusz. Az állomány státusza, amely a feldolgozottságának állapotát rögzíti. A mező a következő értékeket veheti fel.
  10 Fogadott, nem ellenőrzött
  20 Sikeresen fogadott állomány
  30 Hosszú távú MELASZ állomány
  40 Archív MELASZ állomány
- Feltöltés ideje.
- Leírás. A felhasználó által megadható állomány leírás.
- Az aláírás állomány bájtfolyamként.
- Tartalmazó mappa. A tartalmazó mappa azonosítója. Idegen kulcs.

A rendszer működésében nagy szerepet játszanak a tanúsítványok. Ezeket a feldolgozási lépések során gyakran fel kell használni, így fontos a jól strukturált tárolásuk. A tanúsítványokat egyedien azonosítja a kibocsátójuk és a sorszámuk, így ez a két mező lesz a kulcs.

A következő mezők jellemeznek egy **tanúsítványt**:

- Tulajdonos felhasználó. Az állomány tulajdonosának felhasználói azonosítója.
- Kibocsátó egyedi azonosítója. A kibocsátót reprezentáló X.500 entitás azonosítója, idegen kulcs.
- Kibocsátó által adott sorozatszám. A kibocsátó szolgáltató által a tanúsítványba írt sorszám.
- Lejárat dátuma.
- Állomány neve. Az állomány eredeti fájl neve.
- Gyökér tanúsítvány indikátor. Logikai mező, amely a gyökér tanúsítványokat azonosítja. Pontosan akkor igaz, ha a kibocsátó megegyezik a tanúsítottal.
- Tanúsított egyedi azonosítója. A tanúsítottat reprezentáló X.500 entitás azonosítója, idegen kulcs.
- A tanúsítvány állomány bájtfolyamként.
- Visszavont állapot indikátor. A lejáratuk előtt visszavont tanúsítványok esetében igaz.
- CRL elérési helye. A letöltés helyet URL formában kell megadni.

A tanúsítványok alanyai és a tanúsítvány kiállítók egységes szerkezetű névtérből kapják nevüket. Ez a névtér az X.500, amelyet eredetileg egy világméterű elektronikus levelezőrendszer címzési módszerének szántak, de végül nem terjedt el. Azonban kormányzati informatikai rendszerek címtáraiban és a PKI szabványokban ezt alkalmazzák címzésre. Egy cím több kulcs=érték összerendelés halmaza, ahol a kulcs meghatározott értékeket vehet fel. Ezeknek a neveknek létezik karakterfolyam formája is, amelynek szerkezetét, képzésének módját RFC ajánlás rögzíti. Például a MÁV Informatika kibocsátó X.500 neve a következő.

```
E = ica@mavinformatika.hu
PostalCode = 1012
STREET = Krisztina krt. 37/A
CN = Trust&Sign Root CA v1.0
OU = PKI Services BU
O = MAV INFORMATIKA Kft.
L = Budapest
C = HU
```

Az **X.500 entitások** neve angolul principal, így a rendszerben is ilyen név alatt szerepelnek. Jellemzőik a következők:

- Azonosító. Egyedi azonosító mező, amely az entitásokat egyértelműen azonosítja. Idegen kulcsként szolgál a visszavonási listák és a tanúsítványok között is.
- Teljes név. Az entitás X.500 neve az RFC 2253 szabványnak megfelelő formában.

A **tanúsítvány visszavonási listák** feladata azon tanúsítványok felsorolása, amelyeket a lejáratuk előtt visszavontak. Így beszerzésük és tárolásuk szükséges a későbbi hitelesség igazoláshoz. A CRL állományok a következő attribútumokkal rendelkeznek:

- Azonosító. Egyedi azonosító mező, amely a visszavonási listákat egyértelműen azonosítja.
- Kibocsátó X.500 entitás azonosítója. Idegen kulcs.
- Kibocsátás ideje.
- Lejárat. A következő visszavonási lista kibocsátásának várható ideje.
- A visszavonási lista állomány bájtfolyamként.

7.1.2. Segédadatok

A rendszer **felhasználó**-orientált, így szükséges a felhasználók – mint entitások – figyelembe vétele. A következő jellemzőkkel bírnak:

- A felhasználó belépési neve. Az aláírás állományok és a tanúsítványok idegen kulcsként használják.
- Jelszó lenyomat. A felhasználó „sózott” jelszavának MD5 lenyomata.
- Jelszóhoz tartozó só[3]. A lenyomatképzés előtt a jelszóhoz fűzött véletlen adat.
- Jogosultság maszk. A felhasználó szerepeit rögzítő 32 elemű bitminta.

[3] A sózás (salting) technika lényege az, hogy a jelszóhoz véletlen számot fűznek lenyomatképzés előtt, így védve ki a lenyomat elleni szótáras támadást. Ugyanis, a gyakran használt szavak lenyomatát viszonylag kis szótárban is tárolni lehet, így az adatbázisból sok jelszó visszafejthető.

A rendszer működése naplózott, amelynek bejegyzései szintén adatbázisba kerülnek. A naplóbejegyzések típus és felhasználónév szerint leválogathatók. A **bejegyzések típusai** a következő jellemzőkkel rendelkeznek:

- Azonosító. Egyedi azonosító mező, amely a bejegyzés típust egyértelműen azonosítja.
- Név. A bejegyzés típus szöveges elnevezése.

A **naplóbejegyzések** a következő jellemzőkkel bírnak:

- Azonosító. Egyedi azonosító mező, amely a naplóbejegyzést egyértelműen azonosítja.
- Típus. Idegen kulcs a bejegyzés típusok között.
- Felhasználónév. A bejegyzéshez kapcsolható felhasználó neve, amennyiben nem rendszerüzenet. Utóbbi esetben nem kötelező kitölteni. Idegen kulcs a felhasználók között.
- A bejegyzés keletkezésének ideje.
- Szöveg. A naplóbejegyzés szövege.
- Adat. A naplóbejegyzéshez kapcsolódó bináris adat. Lehet például egy hibát okozó aláírás állomány vagy tanúsítvány.

A felhasználók az állományaikat **mappák**ba rendezhetik, ezért van egy mappára mutató idegen kulcs az aláírás állomány jellemzői között. A mappák hierarchiát alkothatnak. A mappák egy részének ezért lesz szülő mappája. A mappáknak a következő jellemzőik vannak:

- Azonosító. Egyedi azonosító mező, amely a mappát egyértelműen azonosítja.
- Tulajdonos felhasználó. Annak a felhasználónak az azonosítója, akié a mappa. Idegen kulcs.
- Név. A mappa neve.
- Szülő mappa. Ha a mappa nem a gyökér mappában van, akkor ide kerül be a szülő mappa azonosítója. Idegen kulcs.

7.2. Dinamikus viselkedés

A rendszer dinamikus viselkedésének három oka lehet, amelyeket három különféle eszköztár szolgál ki:

1. Életciklus események.
2. Szolgáltatás kérés (HTTP vagy WebService hívás).
3. Ütemezett események.

7.2.1. Életciklus események

Az **életciklus események** a rendszer indulásakor és leállásakor történnek. Ekkor indulnak el a rendszer háttér-szolgáltatásai, valamint az ütemezett feladatok is ekkor születnek meg.

A következő szolgáltatások indulnak el a rendszerrel:

- Tanúsítványtár.
- Visszavonási lista tár.
- X.500 entitás tár.
- Felhasználótár.
- Aláírás állomány tár.
- Aláírás állomány feldolgozó motor.
- Napló.

A következő feladatok ütemezése indul el:

- Visszavonási listák periodikus frissítése.
- Aláírás állományok periodikus feldolgozása.

7.2.2. Szolgáltatás kérések

Szolgáltatást a rendszer kliensei kérnek a HTTP vagy a WebService felületen. A következő szolgáltatások állnak rendelkezésre:

- Bejelentkezés
- Kijelentkezés
- Aláírás állomány feltöltés
- Aláírás állomány letöltés
- Aláírás állomány utólagos ellenőrzése
- Tanúsítvány feltöltés
- Tanúsítvány letöltés

A következő szolgáltatásokat a rendszer ütemezett feladatai végzik, de kézzel is indíthatók:

- Aláírás állomány fogadása
- Aláírás állomány kezdeti ellenőrzése
- Aláírás állomány archiválása
- Visszavonási listák frissítése

7.2.3. Ütemezett események

Ütemezett események a rendszer háttér feldolgozásai, amelyek a szolgáltatás kérésekkel ellentétben nem interaktívak, hanem a háttérben futnak le:

- Visszavonási listák frissítése. Ebben a lépésben a rendszer megkísérli letölteni a hitelesítés-szolgáltatók által kiadott legfrissebb tanúsítvány visszavonási listákat. Ha sikerrel járt, a lista alapján a tanúsítványtárban tárolt tanúsítványok státuszait frissíti.
- Aláírás állományok feldolgozása. Ez a folyamat az aláírás állományok feldolgozottságát igyekszik megnövelni, amelynek során többször végigolvassa a bemenetét.

7.3. Segédfunkciók

7.3.1. Bejelentkezés/kijelentkezés a webes felületen

A felhasználók a hitelesítő adatainak megadásával tud a rendszerbe belépni. Ehhez megadja a felhasználónevét és a jelszavát, amely alapján a rendszer azonosítja, vagy elutasítja. Azonosítás esetén létrejön a rendszer és a felhasználó számítógépe között egy munkamenet (session).

7.3.2. Felhasználó hozzáadása, módosítása, törlése

A rendszer adminisztrátorainak joguk van felhasználókat a rendszerbe regisztrálni, vagy meglévő felhasználók adatait módosítani. A törlés szintén megoldható technikailag, de egy megvalósult rendszerben ez nem mindig szükséges, sok problémát vethet fel.

7.3.3. A rendszer paramétereinek átállítása

A rendszer működése több paraméterrel szabályozható. Ezeket a paramétereket a rendszer adminisztrátorai megváltoztathatják. Ez két lépésben zajlik. Először a megfelelő fájlban vagy adattáblában az adminisztrátor megváltoztat értékeket, majd értesíti a rendszert, hogy olvassa be azokat, és változtasson a működésén. Esetleg a rendszert újra is kell indítani.

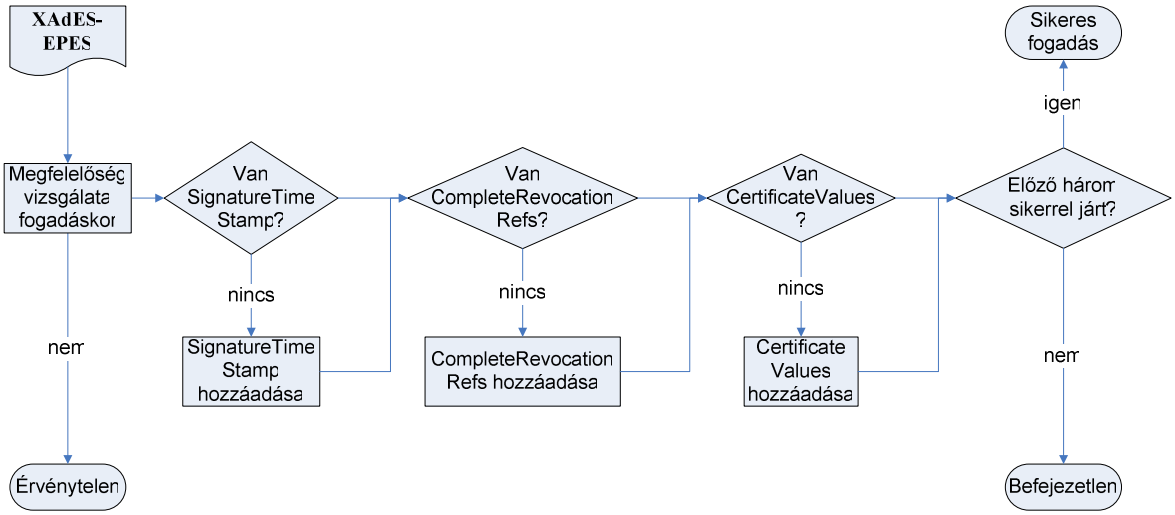
7.3.4. Napló megtekintése

A rendszer működése naplózott. Minden felhasználó megtekintheti a saját magára vonatkozó bejegyzéseket, ezen túl a rendszer adminisztrátorai a teljes naplót megnézhetik, kereshetnek benne, szűrhetik azt.

7.4. Alkalmazási területhez kapcsolódó műveletek

A rendszer a kezelt aláírásokon az alábbi műveleteket képes végrehajtani, melyek összhangban vannak a [2] MELASZ szabvány hatodik fejezetével. Az egyes műveleteket folyamatábrák szemléltetik, a bennük szereplő **segédfunkciókat** a következő fejezet részletezi.

7.4.1. Az aláírás fogadása



2. ábra - Az aláírás fogadás utáni ellenőrzése

Amikor egy aláírás bekerül a rendszerbe, számos feltételnek kell meg felelnie, amelyeket a rendszer ellenőriz. A szabvány szerint ezeket a feltételeket az aláírást létrehozó alkalmazásnak kell biztosítania. Ha az

elvárt minimális feltételek nem teljesülnek, a folyamat eredménye „érvénytelen”. Megfelelőség esetén három további elem meglétéről kell gondoskodni, amelyeket nem kötelezően az aláírást létrehozó alkalmazás is csatolhat.

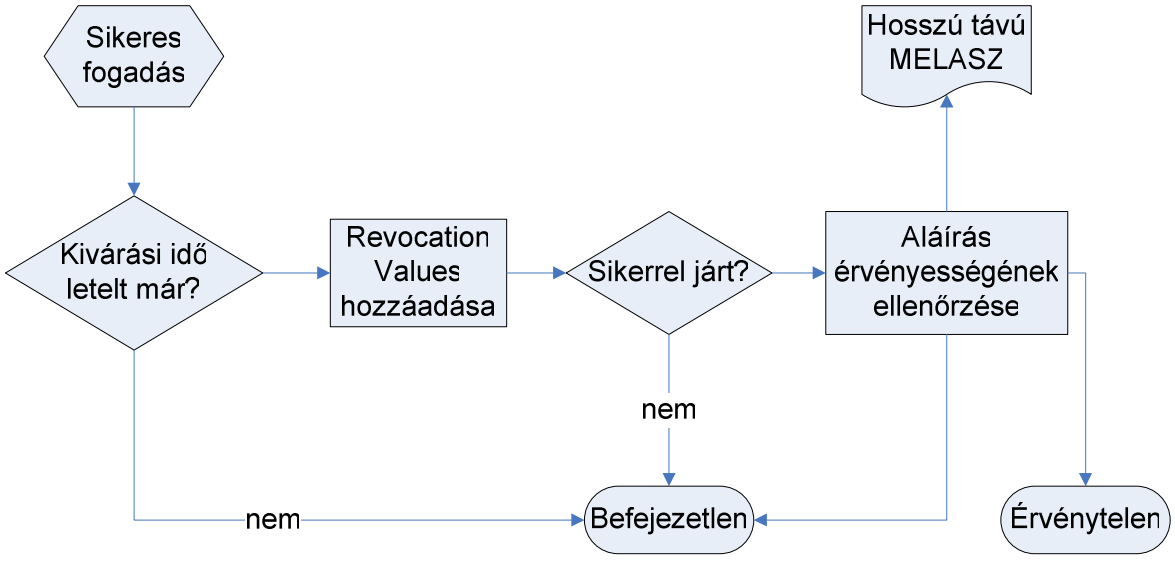
Az első az aláírás időbélyege (SignatureTimeStamp), amely hitelesen igazolja, hogy az aláírás létezett egy bizonyos időpillanat előtt. Ettől számítva a kivárási idő letelte után lehet az aláírás kezdeti ellenőrzését, azaz a következő folyamatot végrehajtani. Emiatt célszerű az időbélyegzést minél korábban végrehajtani.

A második elem hivatkozásokat tartalmaz mindazon tanúsítványok visszavonási listájára, amelyek a hitelesítési láncban szerepelnek (CompleteRevocationRefs).

A harmadik elem tartalmazza a hitelességi láncban szereplő tanúsítványokat.

Ha mind a három elemet sikerül csatolni, akkor az aláírás fogadása sikerrel zárult.

7.4.2. Az aláírás kezdeti ellenőrzése



3. ábra - Aláírás kezdeti ellenőrzése

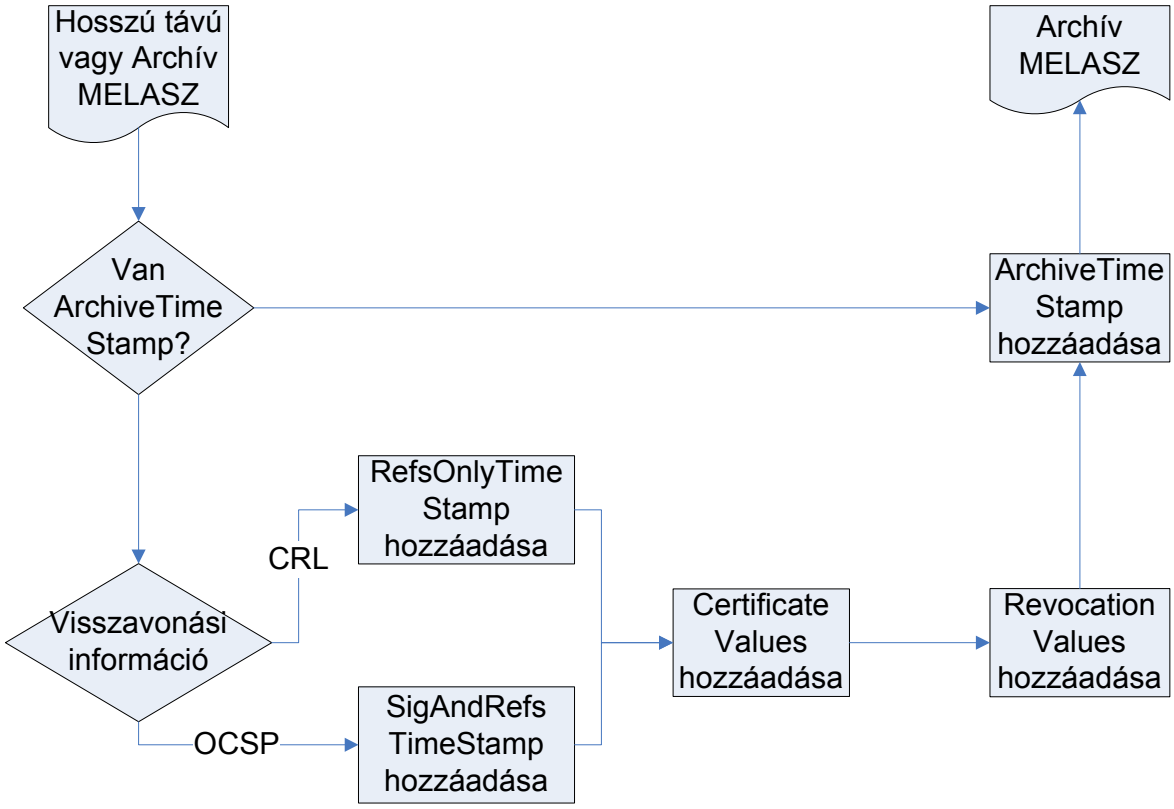
A sikeresen fogadott aláírások feldolgozásának következő lépése az ún. kezdeti ellenőrzés. Ez a folyamat a kivárási idő letelte után hajtható végre sikerrel. A kivárási időszak kezdete az aláírás első hiteles időbélyegzésétől számítandó, hossza CRL visszavonási listák használata esetében 24 óra,

OCSP azonnali tanúsítvány állapotot szolgáltató szerver használata esetén 30 perc.

Ha a kötelező kivárási idő még nem telt le, az aláírás kezdeti ellenőrzése befejezetlen marad.

Amennyiben a kötelező kivárási idő már eltelt, be kell szerezni a hitelességi láncban szereplő tanúsítványok állapotát igazoló adatokat. Ezután ellenőrzést kell végrehajtani az aláíráson. Ha az ellenőrzés megerősíti az aláírás érvényességét, azaz a kriptográfiai aláírás érvényes, és az aláíró tanúsítványt sem veszítette el az érvényességét a kivárási idő alatt, akkor az aláírás érvényes hosszú távú MELASZ aláírás.

7.4.3. Az aláírás archiválása



4. ábra - Aláírás archiválása

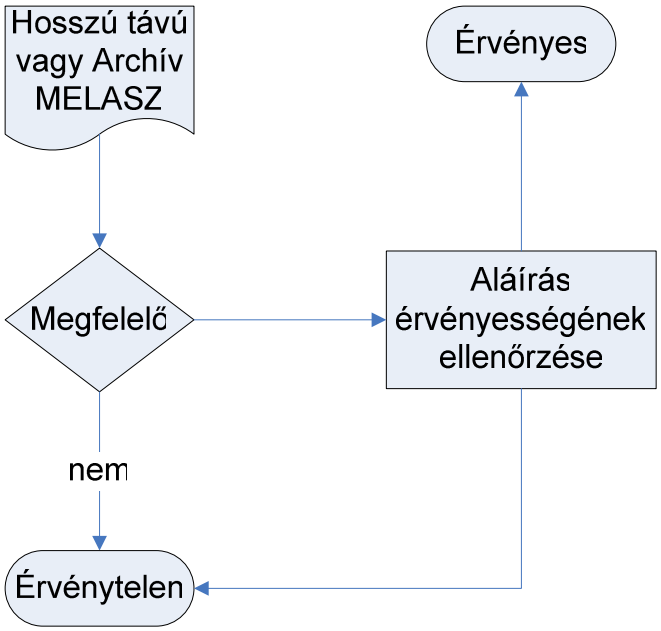
Archív MELASZ aláírás előállításához egy érvényes hosszú távú, vagy egy érvényes archív MELASZ aláírásra van szükség.

Az első eset akkor fordul elő, ha az aláírást először kerül archiválása. Ekkor ki kell egészíteni a visszavonási listájának megfelelő időbélyeggel, amely CRL lista használata esetén egy RefsOnlyTimeStamp elembe kerül, vagy azonnali tanúsítvány állapot információ (OCSP) használata esetén egy

SigAndRefsTimeStamp. Ezután hozzá kell adni az aláíráshoz a tanúsítványokat a CertificateValues elemben, majd a visszavonási információkat a RevocationValues elemben. Végül egy archív időbélyeget (ArchiveTimeStamp elemet) kell készíteni a teljes aláírásra. Ez az időbélyeg abban különbözik az előzőektől, hogy az aláírás minden elemét védi ellentétben a többi időbélyeggel, amelyek nem védik az összes csatolt információt.

A második esetben az aláírás már volt archiválva, és a mostani archiválás archív felülbélyegzést jelent. Erre akkor lehet szükség, ha az előző archiváláskor használt algoritmus vagy kulcs hitelét veszíti. Ebben az esetben csak az archív időbélyeget (ArchiveTimeStamp elemet) kell az aláírásra elhelyezni.

7.4.4. Aláírás utólagos ellenőrzése

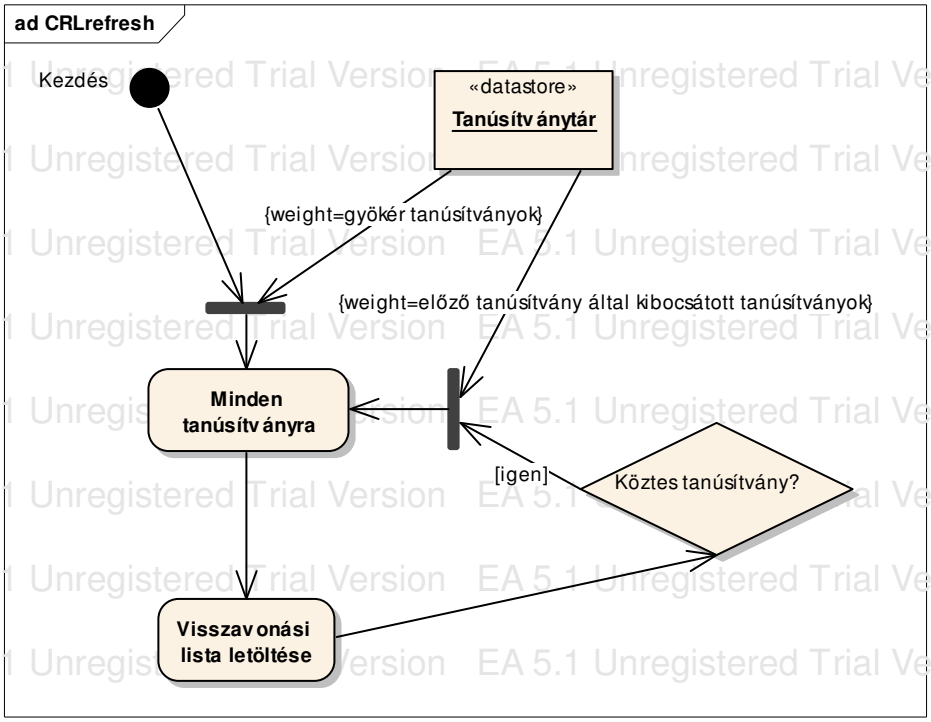


5. ábra - Aláírás utólagos ellenőrzése

Aláírás utólagos ellenőrzésekor a már az aláíráshoz csatolt hitelesítő adatok alapján kell annak érvényességét eldönteni. Első lépésben a formátum helyességét kell megvizsgálni. Lényeges, hogy a kötelező mezők megléte és helyes tartalma. Második lépésben az aláírás kriptográfiai ellenőrzését kell elvégezni. Ha az aláírás mindkét ellenőrzésen átmegy, akkor érvényes, ellenkező esetben érvénytelen.

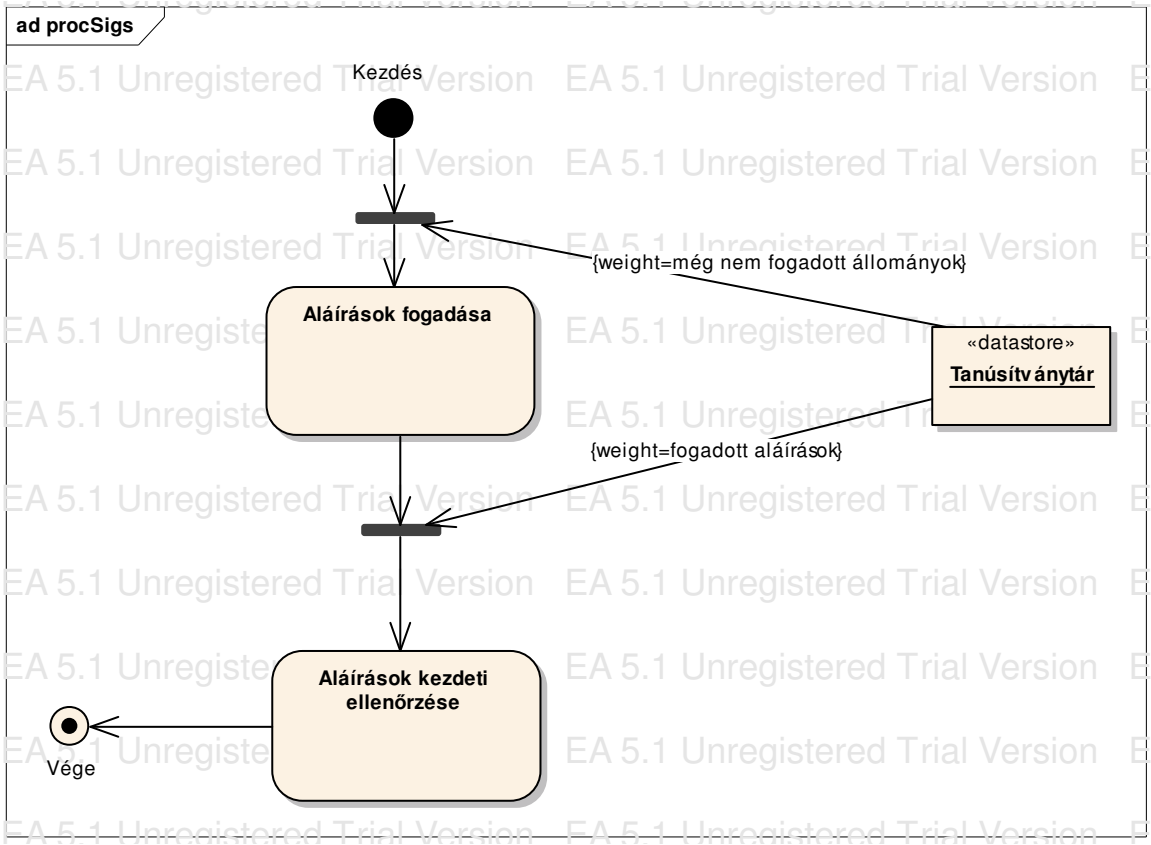
7.4.5. Visszavonási listák frissítése

Ebben a lépésben a rendszer kikeresi a tanúsítványtárból az aktuálisan érvényes kibocsátói tanúsítványokat, és a hozzájuk bejegyzett URL címen keresi a visszavonási listákat. Ha egy tanúsítványhoz a tanúsítványtárban található cím nem elérhető, vagy nincs kitöltve, akkor a folyamat a naplóban hibát jelez. A visszavonási lista elérési cím az adatbázisban kézzel kijavítható. A megtalált visszavonási listák a visszavonási lista tárba kerülnek.



7.4.6. Aláírás állományok feldolgozása

Az első körben a frissen érkezett állományok fogadáskori ellenőrzését végzi el. Második lépésben kiválogatja azokat az állományokat, amelyek már átestek a fogadáskori ellenőrzésen, és az aláírás létrehozása óta a kivárási idejük is eltelt, majd megkísérli ezeket az állományokat hosszú távú MELASZ formátumra hozni. Amennyiben egy feldolgozás sikertelen, bejegyzést tesz a naplóba.



7.5. Alkalmazási területhez kapcsolódó segédfunkciók

Ez a fejezet tartalmazza azon funkciók leírását, amelyekre az előző fejezetben megadott műveletek épülnek.

7.5.1. Aláírás megfelelőségének vizsgálata fogadáskor

Ez a vizsgálat azokat az elemeket ellenőrzi, amelyeket az aláírást létrehozó alkalmazásnak kell az aláírásban elhelyeznie. Egy helyes aláírásban helyes és végleges tartalommal kell szerepelniük a következő elemeknek, amelyeket az XMLDSIG szabvány definiál:

- SignedInfo elem: Egyrészt hivatkozásokat tartalmaz az aláírt adatok elérhetőségére vonatkozóan, másrészt leírja, hogy a hivatkozott adatok az aláírás során milyen algoritmusokkal milyen feldolgozási lépéseken mentek keresztül. Ilyen lépések a kanonizálás, a lenyomatképzés, és az aláírás.
- SignatureValue elem: Az aláírás base64 algoritmussal kódolt értékét tartalmazza.

- KeyInfo elem: Az aláírás érvényesítéséhez szükséges tanúsítványokat, vagy azokra mutató hivatkozásokat tartalmaz. Választhatóan tanúsítvány-visszavonási információ is elhelyezhető ebben az elemben.

A következő elemeket a XAdES szabvány definiálja, és az aláírásban helyes és végleges tartalommal kitöltve kell szerepelniük:

- SigningTime elem: Az aláíró által állított aláírási időpontot tartalmazza.
- SigningCertificate elem: Az aláírást létrehozó tanúsítványra mutató hivatkozást, valamint biztonsági okokból a tanúsítvány lenyomatát tartalmazza.
- SignaturePolicyIndentifier elem: Hivatkozást tartalmaz az aláírási szabályzatra, amely azon szabályok összessége, amelyeket be kell tartani, hogy az aláírás érvényes legyen.
- DataObjectFormat elem: Az egyes aláírt adatok formátumára vonatkozó információkat tartalmazza.
- CompleteCertificateRefs elem: A hitelesítési lánc elemeire mutató hivatkozásokat tartalmazza.

Az aláírást létrehozó alkalmazás választhatóan a következő három elemet is elhelyezheti az aláírásban:

- SignatureTimeStamp elem.
- CompleteRevocationRefs elem.
- CertificateValues.

7.5.2. SignatureTimeStamp hozzáadása

Ebben a feldolgozási lépésben egy időbélyegzés-szolgáltató által kiállított időbélyeget kell az aláíráson elhelyezni. Az időbélyeg elkészítéséhez képezni kell a ds:SignatureValue elemben tárolt aláírás érték lenyomatát, majd azt a szolgáltatónak el kell küldeni. A szolgáltató visszaküldi az időbélyeget, amelyet base64-gyel kódolva kell a SignatureTimeStamp elemben elhelyezni.

7.5.3. CompleteRevocationRefs hozzáadása

Ez a lépés visszavonási információkra vonatkozó hivatkozásokat helyez el az aláírásban. A hivatkozások az aláíró tanúsítványra, valamint az „őt” hitelesítő tanúsítvány láncra vonatkoznak. Egy hivatkozás mutathat időszakosan frissülő CRL listára, vagy azonnali tanúsítvány állapotot szolgáltató (OCSP) szerverre.

Egy CRL hivatkozás tartalmazza a hitelesítés-szolgáltató X.500 formátumú nevét, a lista kibocsátásának időpontját, valamint a lista sorszámát, ha a hitelesítés-szolgáltató ezt a mezőt kitölti.

Az OCSP hivatkozás tartalmazza a szolgáltató szerver címét, a tanúsítvány állapotáról szóló igazolás időpontját, valamint az igazolás kriptográfiai lenyomatát a lenyomatképző algoritmus azonosítójával együtt.

7.5.4. CertificateValues hozzáadása

Ebben a lépésben az aláíráshoz kell csatolni az aláíró tanúsítványt, valamint a hitelesítési láncban szereplő tanúsítványokat. Minden tanúsítvány egy EncapsulatedX509Certificate elembe kerül base64 algoritmussal kódolva. Az egyes EncapsulatedX509Certificate elemek a CertificateValues elembe kerülnek.

7.5.5. RevocationValues hozzáadása

Ez a lépés a hiteles tanúsítvány visszavonási információkat tartalmazza az aláíró tanúsítványra és a hitelesítési lánc elemeire vonatkozóan. Az egyes CRL listák vagy OCSP válaszokat base64-gyel kódolva kell az aláíráshoz mellékelni a RevocationValues elembe.

7.5.6. Aláírás érvényességének ellenőrzése

Az első lépés a tanúsítványlánc érvényességének ellenőrzése. Meg kell vizsgálni, hogy a kivárási időn belül a hitelesítési lánc valamely tanúsítványát visszavonták-e. Ha igen, akkor az aláírás érvénytelen.

A második lépés a kriptográfiai ellenőrzés. Az aláírás ds:SignatureValue elemben található értékét kell leellenőrizni, melyhez szükséges az aláírt dokumentum és a hitelesítési lánc minden tanúsítványa. A dokumentum vagy része az aláírásnak, vagy külön áll tőle, de a

felhasználónak valamilyen formában mindenképp el kell juttatnia a rendszerbe. A hitelesítési lánc elemeit a CertificateValues elemnek kell tartalmaznia. Ezek felhasználásával ellenőrizni kell a láncban szereplő aláírásokat. Ha a lánc valamelyik eleme érvénytelen, akkor az aláírás is érvénytelen.

Ha mindkettő ellenőrzés érvényes eredményt ad, akkor az aláírás érvényes.

7.5.7. RefsOnlyTimeStamp hozzáadása

Ez a lépés az aláírás ellenőrző adatokra mutató hivatkozásokat védi időbélyeg elhelyezésével. Akkor használatos, amikor a visszavonási információk CRL típusúak. Egy ilyen időbélyeget biztonságos időbélyegzés-szolgáltatónak kell elkészítenie a következő elemekre.

- CompleteCertificateRefs
- CompleteRevocationRefs.

Az időbélyeg base64-gyel kódolt értékét a RefsOnlyTimeStamp elemben kell elhelyezni.

7.5.8. SigAndRefsTimeStamp hozzáadása

Ez a lépés is időbélyeget helyez el. Abban az esetben használatos, amikor OCSP válasz igazolja egy tanúsítvány állapotát. Ekkor az időbélyeget a következő elemekre kell kérni.

- ds:Signature,
- SignatureTimeStamp,
- CompleteCertificateRefs,
- CompleteRevocationRefs.

Az időbélyeg Base64 kódolt értékét a RefsOnlyTimeStamp elemben kell elhelyezni.

7.5.9. CertificateValues hozzáadása

Az aláírás későbbi ellenőrzése szempontjából szükséges *tanúsítványokat* – egészen a gyökér tanúsítványokig visszamenőleg – kell ebben a lépésben az XML megfelelő pontjában Base64 kódolással elhelyezni.

7.5.10. RevocationValues hozzáadása

Az aláírás későbbi ellenőrzése szempontjából szükséges *tanúsítvány visszavonási listákat* – egészen a gyökér kibocsátókig visszamenőleg – kell ebben a lépésben az XML megfelelő pontjában Base64 kódolással elhelyezni. A visszavonási lista célszerűen a kivárási idő letelte utáni legelső visszavonási lista.

7.5.11. ArchiveTimeStamp hozzáadása

Az archív időbélyeg hozzáadásakor hiteles időbélyegzés-szolgáltatóval kell az időbélyeget elkészíttetni az összes olyan elemre, amelyben tárolt információ hitelessége elveszhet a kriptográfiai algoritmusok meggyengülése miatt.

8. Fizikai rendszerterv

Ez a fejezet a logikai rendszerterv egy fizikai megvalósításának terveit írja le.

8.1. Tervezési döntések

- A rendszer J2EE alkalmazásszerveren fog futni, és ún. Pooled DataSource-on keresztül kapcsolódik az adatbázishoz. Ez tehermentesíti a kódot az adatbázis kapcsolatok kiépítésétől.
- A rendszer naplózni fog, hogy nyomon lehessen követni a működését.
- JSP nézetek lesznek, amelyek saját lekérdezésekkel fordulnak az adatbázis szerverhez.
- A rendszer vezérlő eleme a Jakarta Struts szervlet lesz, nem pedig saját vezérlő szervlet. A műveletek így ún. akció osztályokban kapnak helyet, a vezérlő pedig egy ún. paraméter állományból tudja, hogy milyen URL kérés esetén mely akció osztályt kell meghívnia. Mivel egy webes alkalmazás vezérlése kevés egyedi megoldást igényel, ezért ez a választás nagyban megkönnyíti a rendszert fejlesztését.
- A rendszer a felhasználóival HTTPS protokollon keresztül fog kommunikálni.
- A rendszer paraméterei egy XML állományban kapnak helyet. Az állományt a rendszer induláskor és külön kérésre olvassa be, és érvényesíti a beállításokat.

8.2. Adatbázis

A rendszer hátteréül szolgáló adatbázis a logikai adatmodell alapján valósítható meg a kiválasztott adatbázis kezelő kiszolgáló szoftverrel. A fizikai adatmodell a ***negyedik mellékletben*** látható.

A választás a Microsoft SQL Server ingyenes asztali változatára esett, mivel megbízható és könnyen használható a grafikus kezelőfelülete segítségével. Ez a választás a platform-függetlenséget nagyban érinti, mivel a szoftver csak Windows platformon érhető el. Azonban, a rendszer

az SQL utasítások kis módosításával más adatbázis szoftverekkel is működőképes, hiszen nem használja ki az Microsoft SQL Server speciális lehetőségeit, csak az SQL nyelv alapvető elemeit.

Az adattáblák fizikai megvalósítása során a grafikus kezelőfelületen elérhető diagram-alapú tervezőkörnyezetet használtam, így állítottam be az idegen kulcsok rendszerét, valamint az adatmezők típusait. A következő adattípusokat rendeltem a különböző attribútumokhoz.

- Az egyedi azonosító (ID) mezők 4 bájtos *int* típusúak.
- A felhasználónév mezők legfeljebb 50 hosszú *varchar* szöveges típusúak.
- A tanúsítványok sorszám mezői *bigint* típusúak, mivel a Java Biztonsági Keretrendszer X.509 implementációjában a sorszám BigInteger típusú.
- Az időpontot tároló mezők *datetime* típusúak.
- A bájtfolyam mezők – az aláírás állomány, tanúsítvány, visszavonási listák táblákban – *image* típusúak, így tetszőleges hosszúságú bináris adatot tudnak tárolni.
- A felhasználói jelszavak lenyomatait tároló mezőnek bináris adatot kell tárolnia. A mező szükséges hossza függ a lenyomatképző algoritmus megválasztásától. Az MD5 például 128, az SHA-1 160 bites, az SHA-512 pedig 512 bites lenyomatot képez. Ez utóbbi algoritmus tehát 64 bájtot igényel, így ezt választottam mezőméretnek.

8.3. Felhasznált csomagok

O'Reilly Servlet

Ennek a csomagnak két fontos funkcióját használtam. Tartalmaz egy Base64 kódoló és dekódoló metódust, továbbá támogatja az ún. multipart-form-data MIME típus feldolgozását. Ez az a MIME típus, amit a böngészők egy fájl feltöltésekor használnak a válasz kódolására.

Apache XML Security

Ez a csomag az XML kanonizáló algoritmusával járul hozzá a rendszer működéséhez.

JDOM

A JDOM csomag az XML kezelést teszi rendkívül egyszerűvé Java környezetben.

Bouncy Castle Crypto API

A Bouncy Castle Crypto csomagot használtam az RFC 3161 szabványnak megfelelő időbélyeg kérések előállítására és a válaszok feldolgozására.

8.4. Osztályok

A rendszer osztálydiagramja az ***ötödik mellékletben*** látható.

8.4.1. archiver::XAdESTransformer

public receiveSignature(String userName, int fileID, List<String> ret) : void

Ez a metódus paraméterül kapott aláírás állomány fogadását végzi el. Első lépésben ellenőrzi, hogy az állomány megfelel-e azon feltételeknek, amelyeket a MELASZ ajánlás szerint az aláírást létrehozó szoftvernek kell biztosítani. Ha megfelel, gondoskodik az aláírás hiteles időbélyegzéséről, a visszavonási lista referenciák meglétéről, és arról, hogy az aláíró, és a hitelességi lánc tanúsítványai az állományban benne legyenek. Ha hiba történik, azt kivétellel jelzi.

A metódus pszeudokódja a következő.

```
XAdES sig = sigs.get(userName, fileID);
if (sig.isReceivable()) {
        checkSignatureBasic(sig);
        QualifyingProperties qp = sig.getQP();
        if (null == qp.getSignatureTimeStamp())
            qp.addSignatureTimeStamp();
        addRevocationRefs(sig);
        if (qp.getCertRefs().size() >
            qp.getCertValues().size())
            addCertValues(sig);
}
```

public initialCheck(String userName, int fileID, List<String> ret) : void

Ez a metódus a paraméterül kapott állomány kezdeti ellenőrzését végzi el a kivárási idő alatt beszerzett visszavonási listák felhasználásával. Ha az ellenőrzés sikeres – amihez szükségesek a friss visszavonási listák is – akkor az aláíráshoz csatolja a friss visszavonási listákat is. Ha hiba történik, azt kivétellel jelzi.

A metódus pszeudokódja a következő.

```
XAdES sig = sigs.get(userName, fileID);
long timeElapsed = currentTime - sigs.getUploadTime(...);
if (timeElapsed > gracePeriod)
     if (checkSignatureFully(sig))
         addRevocationValues(sig);
```

public archiveSignature(String userName, int fileID, List<String> ret) : void

Ez a metódus a paraméterül kapott állomány archiválását végzi el. Ennek során először gondoskodik róla, hogy a hitelességi lánc minden tanúsítványa az állomány része legyen. Ha ez teljesül, két időbélyeget helyez el az aláíráson. Az első időbélyeg a következő elemekbe kerülhet.

- CRL visszavonási információ esetén egy RefsOnlyTimeStamp elembe.
- OCSP esetén egy SigAndRefsTimeStamp elembe.

A második időbélyeg egy ArchiveTimeStamp elembe kerül. A metódus pszeudokódja a következő. Ha hiba történik, azt kivétellel jelzi.

A metódus pszeudokódja a következő.

```
XAdES sig = sigs.get(userName, fileID);
QualifyingProperties qp = sig.getQP();
if (qp.getCertRefs().size() > gp.getCertValues().size())
    addCertValues(sig);
if (null == qp.getRefsOnlyTimeStamp(sig) &&
    0 < qp.readCRLRefs().size())
    qp.addRefsOnlyTimeStamp();
if (null == qp.getSigAndRefsTimeStamp() &&
    0 < qp.readOCSPRefs().size())
    qp.addSigAndRefsTimeStamp();
qp.addArchiveTimestamp();
```

private addRevocationRefs(XAdES sig, List<String> ret) : void

Ez a metódus a paraméterül kapott állomány visszavonási lista hivatkozásait ellenőrzi és szükség esetén kiegészíti azokat. A visszavonási lista hivatkozásoknak meg kell lenni minden – a hitelességi láncban érintett – kiállítóhoz. Kétféle hivatkozás lehetséges, CRL és OCSP típusú. Ha hiba történik, azt kivétellel jelzi.

private addCertValues(XAdES sig, List<String> ret) : String

Ez a metódus a paraméterül kapott állomány tanúsítvány értékeit ellenőrzi és egészíti ki szükség esetén. Először lekéri a tanúsítvány hivatkozásokat (CompleteCertificateRefs), majd összehasonlítja a tanúsítvány értékekkel (CertificateValues). A hiányzó tanúsítványokat a tanúsítványtár alapján megkísérli pótolni. Ha hiba történik, azt kivétellel jelzi.

private checkSignatureBasic(XAdES sig) : void

Ez a metódus az aláírás klasszikus kriptográfiai ellenőrzését végzi el. Ez minden aláírás állomány esetében elvégezhető, mivel csak az aláírt dokumentum és az aláíró tanúsítvány megléte szükséges hozzá. Jogi értelemben ez az ellenőrzés még nem hitelesít egy aláírást – hiszen nem tudható, hogy visszavonták-e az aláíró tanúsítványt az aláírás létrehozás egyébként szintén nem ismert pillanatában – de a primitív hamisítványokat azonnal kiszűri. Ezt az ellenőrzést az aláírás fogadó metódus használja. Ha hiba történik, azt kivétellel jelzi.

private checkSignatureFully(XAdES sig) : void

Ez a metódus az aláírás teljes körű ellenőrzését végzi el, beleértve a teljes hitelességi láncot, visszavonási listákon szereplő aláírásokat, és az időbélyegek értékeit is. A következők ellenőrizendők.

- Új lenyomatképzéssel az aláírt adatok lenyomatértéke, majd az aláíró nyilvános tanúsítványával a lenyomat kriptográfiai kódoltja.
- Az aláíró tanúsítványon szereplő kibocsátói aláírás, a kibocsátói tanúsítványon szereplő kibocsátói aláírás, és így tovább a gyökér tanúsítványig, tehát a hitelességi lánc kriptográfiai szempontból.
- Minden időbélyeg – beleértve az archív időbélyegeket is – kriptográfiai szempontból az időbélyegzés-szolgáltató nyilvános tanúsítványának segítségével.
- A visszavonási listák felhasználásával a hitelességi lánc tanúsítványainak érvényessége – azaz, hogy nem vonták-e őket vissza az aláírás pillanata plusz a kivárási idő előtt. Ez tehát nem kriptográfiai ellenőrzés.
- A visszavonási listákon szereplő kibocsátói aláírás kriptográfiai szempontból.

Ez a metódus tehát minden lehetséges szempontból ellenőrzi az aláírás állományt. Hiba esetén kivételt dob.

private addRevocationValues(XAdES sig) : void

Ez a metódus a paraméterül kapott állomány visszavonási lista értékeit ellenőrzi és egészíti ki szükség esetén. Amennyiben egy visszavonási lista hivatkozásnak (CRLRef XML elem) megfelelő visszavonási lista állomány (RevocationValue XML elem) nincs az aláíráshoz csatolva, a visszavonási lista tár alapján megpróbálja azt hozzá csatolni.

8.4.2. archiver::xades::QualifyingProperties

Ez az objektum egyértelműen kapcsolódik egy XAdES objektumhoz. A 'QualifyingProperties' XML részfát reprezentálja és kezeli.

public addArchiveTimeStamp() : String

Ez a metódus az archív MELASZ formátum eléréséhez szükséges 'ArchiveTimeStamp' időbélyeget helyezi el az aláírás állományon.

public addCertValue(byte[] bytes) : void

Ez a metódus a paraméterül kapott bájttömbben tárolt tanúsítványt helyezi el az aláírás állomány 'CertificateValues' részfájába egy 'EncapsulatedX509Certificate' elembe Base64 kódolással.

public addRevocationValue(byte[] bytes) : void

Ez a metódus a paraméterül kapott bájttömbben tárolt visszavonási listát helyezi el az aláírás állomány 'RevocationValues' részfájába egy 'EncapsulatedCRLValue' elembe Base64 kódolással.

public addRevocationRef(String issuer, Date issueTime) : void

Ez a metódus egy visszavonási listára mutató hivatkozást szúr be az aláírás állományba. A hivatkozás egy 'CRLRef' elembe kerül a 'CompleteRevocationRefs' elem alá.

public addSignatureTimeStamp : String

Ez a metódus a XAdES-T formátum eléréséhez szükséges időbélyeget helyezi el az aláírás állományon, amelyet a ds:SignatureValue XML elemre kell kérni. Az időbélyeg kérésnél kötelező kérni az időbélyegzés-szolgáltatót, hogy a tanúsítványát csatolja az időbélyeghez.

public addRefsOnlyTimeStamp() : String

Ez a metódus az archív MELASZ formátum eléréséhez szükséges 'RefsOnlyTimeStamp' időbélyeget helyezi el az aláírás állományon.

public addSigAndRefsTimeStamp() : String

Ez a metódus az archív MELASZ formátum eléréséhez szükséges 'SigAndRefsTimeStamp' időbélyeget helyezi el az aláírás állományon.

public getCertRefs() : List<CertKey>

Ez a metódus az aláíráshoz tartozó tanúsítvány hivatkozások listáját adja vissza. A lista elemei CertKey objektumok, amelyek egy a tanúsítványtár kulcsai, kibocsátó és sorszám szerepel bennük.

public getCertValues() : List<byte[]>

Ez a metódus visszaadja az aláírás állomány 'CertificateValues' részfája alatti tanúsítvány értékek – Base64 dekódolás utáni – binárisan kódolt formáját.

public getCRLValues() : List<byte[]>

Ez a metódus visszaadja az aláírás állomány 'CRLValues' részfája alatti tanúsítvány értékek – Base64 dekódolás utáni – binárisan kódolt formáját.

public getCRLRefs() : List<Map<String,String>>

Ez a metódus az aláírás állomány 'CompleteRevocationRefs' elem alatti 'CRLRefs' részfáját, azaz a hivatkozott visszavonási listákat adja vissza Java struktúrákban ábrázolva. A lista minden eleme egy Map interfészt megvalósító objektum, amely megfelel egy 'CRLRef' elemnek.

A leképezés kulcsai a következők.

- Issuer. A visszavonási lista kibocsátója.
- IssueTime. A visszavonási lista kibocsátási időpontja.
- Number. A visszavonási lista sorszáma. Nem kötelező elem.
- DigestMethod. A visszavonási lista lenyomatának elkészítéséhez használt algoritmus neve.
- DigestValue. A lenyomat értéke.

public getDataObjectFormat() : List<Element>

Visszaadja a DataObjectFormat XML elemek listáját, amelyek meghatározzák a csatolt Object elemek MIME típusát és kódolását.

public getOCSPRefs() : List<Map<String,String>>

Ez a metódus az aláírás állomány 'CompleteRevocationRefs' elem alatti 'OCSPRefs' részfáját, azaz a hivatkozott azonnali tanúsítvány állapot válaszokat (Online Certificate Status Protocol válaszokat) adja vissza Java

struktúrákban ábrázolva. A lista minden eleme egy Map interfészt megvalósító objektum, amely megfelel egy ’OCSPRef’ elemnek.

A leképezés kulcsai a következők.

- DigestMethod. A visszavonási lista lenyomatának elkészítéséhez használt algoritmus neve.
- DigestValue. A lenyomat értéke.
- ResponderID. A tanúsítvány státusz választ kibocsátó szerver neve.
- ProducedAt. A válasz kibocsátás ideje.

public getPath(Element e) : String

Ez a rekurzív segédmetódus egy XML fa elemének adja meg a gyökértől mért azonosítóját. Az azonosítóban az elem őseinek neve szerepel ’–’ jellel elválasztva, és szinten belüli sorszámmal kiegészítve. A sorszámra azért van szükség, hogy az egy szinten lévő (testvér) ugyanolyan nevű testvér csomópontokat meg lehessen különböztetni.

Például ’Signature-0-SignedInfo-0-Reference-1’ egy olyan elemnek az azonosítója, amelynek neve ’Reference’, a közös ősük egy ’SignedInfo’ elem, amelynek őse az XML fa gyökere, egy ’Signature’ elem. A ’Signature’ elem értelemszerűen az első és egyben az utolsó is, ezért sorszáma ’0’, a ’SignedInfo’ elem az első a szinten, a ’Reference’ elem pedig a második ilyen ezen a szinten, tehát van egy testvére, amely megelőzi.

Erre az azonosítóra az összes olyan metódusnak szüksége van, amelyik új elemet szúr be az XML struktúrába, és – a MELASZ ajánlás szerint – kötelezően ki kell töltenie az elem ’Id’ attribútumát.

public getSignaturePolicyIdentifier() : Element

Ez a metódus visszaadja az aláíráshoz tartozó SignaturePolicyIdentifier XML elemet, amely tartalmazza az aláírási politika egyértelmű azonosítóját, vagy null-t, ha nincs ilyen elem.

public getRefsOnlyTimeStamp() : Element

Ez a metódus visszaadja az aláírás állomány RefsOnlyTimeStamp elemét.

public getSigAndRefsTimeStamp() : Element

Ez a metódus visszaadja az aláírás állomány SigAndRefsTimeStamp elemét.

public getSignatureTimeStamp() : Element

Ez a metódus visszaadja az aláíráshoz tartozó SignatureTimeStamp XML elemet, amely egy megbízható időbélyegzés-szolgáltatató (TSA) által kiállított hiteles időpecsét, vagy null-t, ha nincs ilyen elem.

public getSigningCertificate() : Element

Ez a metódus visszaadja az aláíráshoz tartozó aláírás létrehozó tanúsítványt tartalmazó XML elemet, vagy null-t, ha nincs ilyen elem.

public getSigningTime() : Element

Ez a metódus visszaadja az aláíráshoz tartozó SigningTime elemet, amely az aláíró által állított, de nem ellenőrizhető időpont, vagy null-t, ha nincs ilyen elem.

public getXPath(Element e) : String

Ez a rekurzív segédmetódus egy XML fa elemének adja meg az XPath kifejezését, amellyel elérhető. Erre a kifejezésre az időbélyegző metódusoknak van szüksége, a kanonizáló algoritmus használatához.

private timeStamp(byte[] digest) : byte[]

Ezt a privát segédmetódust az időbélyegzést végző metódusok használják, amikor az elkészült – XML részfa összeállítása, kanonizálása és lenyomatkészítése után kapott – lenyomatot megbízható időbélyegzés-szolgáltatóval szeretnék lepecsételtetni. A metódus a QualifyingProperties objektum TSAaddresses mezőjéből veszi az időbélyegzés-szolgáltatók URL elérhetőségét.

8.4.3. archiver::xades::XAdES

public isReceivable(List<String> ret) : void

Ez a metódus az aláírás állomány fogadáskori ellenőrzését végzi el. Az eredményeket a paraméterül kapott listába írja, hiba esetén kivételt dob.

8.4.4. archiver::CertContainer

Ez az osztály a tanúsítványok adatbázisba mentését és visszatöltését végzi, amelynek során mentéskor bájtokká alakítja a tanúsítvány fájlt, betöltéskor pedig Java objektumot épít a bájtokból.

public add(User u, File f, String fileName) : int

Ez a metódus a paraméterül kapott fájlt menti el az adatbázis megfelelő táblájába.

public get(int issuerID, BigInteger serial) : X509Certificate

Ez a metódus visszaadja a paraméterek által meghatározott tanúsítványt az adatbázisból. A tároló belül tartalmaz egy gyorsító tárat, így az egymást követő lekérés műveletek hatására egy tanúsítványból csak a legelső híváskor épít X509Certificate objektumot, a későbbi híváskor a gyorsítótárba bejegyzett objektumot adja eredményül. A gyorsítótár egy WeakHashMap objektum, amelynek használata biztosítja, hogy a már nem használt tanúsítvány objektumokat a Java virtuális gép szemétgyűjtője felszabadíthassa.

8.4.5. archiver::CRLContainer

Ez az osztály a tanúsítvány visszavonási listák adatbázisba mentését és visszatöltését végzi, a CRLContainer osztályhoz teljesen hasonló módon.

public get(int issuerID, Date issueTime) : X509CRL

Ez a metódus – adatbázisból, vagy a belső gyorsítótárából – visszaadja a paramétereknek megfelelő visszavonási listát, vagy null-t, ha nem talált a paramétereknek megfelelőt.

public getNewest(int issuerID) : X509CRL

Ez a metódus visszaadja a paraméterül kapott kibocsátóhoz tartozó legfrissebb tanúsítvány visszavonási listát az adatbázisból.

public getFirstAfter(int issuerID, Date issueTime) : X509CRL

Ez a metódus – adatbázisból, vagy a belső gyorsítótárából – visszaadja a paraméterül kapott időpont utáni legfrissebb visszavonási listát az adott kibocsátóhoz.

public refreshCRL(String url) : File

Ez a segédmetódus a paraméterül kapott URL címről letölti az ott található fájlt, majd visszaadja azt.

8.4.6. archiver::Principals

public getPrincipalId(X500Principal p) : int

Lásd a következő metódust.

public getPrincipalId(String s) : int

Ez a segédmetódus a paraméterül kapott X.500 entitás adatbázisbeli azonosítóját keresi ki és adja vissza. Ha nem talál megfelelőt az adatbázisban, akkor automatikusan létrehoz egy neki megfelelő bejegyzést a 'principals' táblában, és az ahhoz tartozó azonosítót adja vissza.

8.4.7. archiver::User

Ez az osztály a rendszer egy regisztrált felhasználóját modellezi.

public setPasswd(String passwd) : void

Ez a metódus a paraméterül kapott jelszó „sózott" lenyomatára cseréli a felhasználó előző jelszavának lenyomatát.

public getPasswdHash() : byte[]

Ez a metódus visszaadja a felhasználó „sózott" jelszavának tárolt lenyomatát.

public isPasswdOk(String passwd) : boolean

Ez a metódus ellenőrzi a paraméterül kapott jelszót. Ehhez a jelszóhoz fűzi a salt mezőben tárolt értéket, képzi a lenyomatot, és összehasonlítja a tárolt lenyomattal.

8.4.8. archiver::UserContainer

Ez az osztály tárolja a bejelentkezett felhasználókat modellező objektumokat, és végez el minden felhasználókkal kapcsolatos műveletet.

public static isUserAuthorized(User user, String URI) : boolean

Ez a metódus a felhasználók hozzáférésének finom szabályozására való. Az archiver::servlets::AuthenticationFilter osztály használja fel, amikor dönt a szolgáltatás kérés továbbengedéséről.

public login(String name, String pass) : User

Ez a metódus megpróbálja bejelentkeztetni a paraméterül kapott felhasználót. Ellenőrzi a jelszavát, majd siker esetén feljegyzi bejelentkezett felhasználóként, és visszaadja az objektumát. Hiba esetén kivételt dob.

public logout(User u) : void

Ez a metódus kijelentkezteti a paraméterül kapott felhasználót.

public add(String name, String passwd, boolean isAdmin) : int

Ez a metódus új felhasználót jegyez be az adatbázisba a kapott paraméterek alapján. Visszatérési értéke siker esetén 1, egyébként 0, vagy kivételt is dobhat.

8.4.9. archiver::XAdESContainer

Ez az osztály az aláírás állomány tárat valósítja meg, alapvető műveletei a hozzáadás és a lekérdezés. A törlés művelet nem támogatott.

public getStatus(String userName, int fileID) : int

Ez a metódus a paraméterekkel adott aláírás állomány státuszát adja vissza.

public setStatus(String userName, int fileID, int newStatus) : int

Ez a metódus a paraméterekkel adott aláírás állomány státuszát növeli meg *newStatus* értékre. Ha az új státusz kisebb volna a réginél, a metódus kivételt dob.

8.4.10. archiver::util::Canonicalizer

Ez az osztály az időbélyegzéshez szükséges XML kanonizáló műveleteket valósítja meg. A működés az Apache XML Security csomagra épül.

private writeToFile(org.jdom.Document doc, File file) : void

Ez a segédmetódus a paraméterül kapott JDOM XML dokumentum reprezentációt írja ki a paraméterül kapott fájlba. Erre azért van szükség, mert a kanonizáló metódus kizárólag DOM XML dokumentum reprezentációt fogad el.

public convertToDOMDocument(org.jdom.Document jdomDocument) : org.w3c.dom.Document

Ez a metódus az előző metódus felhasználásával először ideiglenes fájlba írja a paraméterül kapott JDOM dokumentumot, majd abból W3C DOM dokumentumot épít, amit visszaad.

public canonicalize(org.w3c.dom.Document w3cDocument, String xpath) : byte[]

Ez a metódus a paraméterül kapott W3C DOM dokumentum egyik részfáját kanonizálja a „http://www.w3.org/TR/2001/REC-xml-c14n-20010315" URI azonosítójú algoritmus szerint. Az eredményt bájt tömb formájában adja vissza.

8.4.11. archiver::util::CertKey

Ez az osztály a tanúsítványtár kulcsa. Két mezője van, a tanúsítvány kibocsátója és a tanúsítvány sorszáma. Valójában egy egyszerű csomagoló objektum, amely kanonizált formában tárolja az X.500 nevet. Továbbá, segít kivédeni a Java X500Principal osztályának hiányosságait. Például a Java implementáció az „emailAddress” mező nevét csak „EMAILADDRESS” formában fogadja el, a „postalCode” mezőt pedig csak „OID.2.5.4.17” néven ismeri fel. Ezért az osztály a konstruktorában minden karakterláncként adott X.500 nevet először X500Principal objektummá alakít, majd abból képez újra karakterláncot.

8.4.12. archiver::servlets::AuthenticationFilter

Ez az osztály a szervlethez érkező kérések előtét szűrője. Feladata a felhasználók jogosultságának ellenőrzése, és a „time-out” kezelés.

public doFilter(ServletRequest request, ServletResponse response, FilterChain chain) : void

Ez a metódus a paraméterül kapott kérést vizsgálja. Ha a kérés egy munkafolyamat része, továbbá a munkafolyamathoz tartozik hitelesített felhasználó, akkor tovább engedi a kérést, ellenkező esetben átirányítja azt a bejelentkező oldalra. A szűrő továbbá még megvizsgálja, hogy a bejelentkezett felhasználó jogosult-e annak az oldalnak a használatára, amit a kérés tartalmaz.

Ezenkívül, a metódus minden futásakor beleírja az aktuális munkamenetbe a pontos időt. Ez alapján meg tudja vizsgálni, hogy egy kéréskor nem telt-e el túl sok idő az előző művelet óta. A timeout hosszát a rendszer paraméter állományából tudja meg. Amennyiben túl sok idő telt el, kilépteti a felhasználót és visszaküldi a bejelentkező oldalra.

8.4.13. archiver::servlets::ServletListener

Ez az objektum két metódusával a szervlet életciklusát kíséri, gyakorlatilag induláskor létrehozza a különböző tárolókat.

8.5. JSP nézetek

A rendszer képernyőtervei a ***hatodik mellékletben*** találhatók.

8.5.1. Tanúsítványok

Ez a nézet a bejelentkezett felhasználó saját tanúsítványait jeleníti meg. A következő akciók hajthatók innen végre.

- Tanúsítvány feltöltése
- Tanúsítvány letöltése
- Kibocsátó legfrissebb visszavonási listájának letöltése a szolgáltatótól

8.5.2. Visszavonási listák

Ez a nézet a tanúsítvány visszavonási listákat jeleníti meg. A következő akciók hajthatók innen végre.

- Visszavonási lista letöltése

8.5.3. Karbantartás

Ez a nézet a rendszer felhasználóit és a rendszer paramétereit jeleníti meg. A következő akciók hajthatók innen végre.

- Új felhasználó felvétele
- Rendszer paramétereinek módosítása

8.5.4. Aláírás állományok

Ez a nézet a felhasználó aláírás állományait jeleníti meg. A következő akciók hajthatók innen végre.

- Állomány feltöltése
- Állomány letöltése
- Állomány fogadása
- Állomány kezdeti ellenőrzése
- Állomány archiválása
- Állomány utólagos ellenőrzése

8.5.5. Napló megtekintése

Ebben a nézetben a napló tekinthető meg különböző nézetekben és szűrő feltételekkel. A felhasználók csak a saját bejegyzéseiket láthatják, a rendszer adminisztrátorok a teljes naplót.

8.6. Akciók

A logikai rendszertervben felsorolt nem ütemezett szolgáltatásokat a kódban egy-egy Jakarta Struts akció objektum valósítja meg. A háttérben futó ütemezett feladatokat a java.util.Timer osztály segítségével ütemezem. A „protected” könyvtár alatt lévő akciókat az AuthenticationFilter objektum védi, tehát csak bejelentkezett felhasználók számára érhetők el. Amennyiben hitelesített munkamenet nélküli számítógép próbálja elérni a védett akciókat, a szűrő átirányítja az üdvözlő oldalra a felhasználót.

/Login

Ezt az akciót hívja meg a bejelentkező űrlap a „Bejelentkezés” gomb megnyomására. Az akció ellenőzi a beírt felhasználónevet és jelszót.

Ha helyes, akkor létrehozza a felhasználói munkamenetet, felveszi a felhasználót a bejelentkezett felhasználók listájára (amelyet a Felhasználótár kezel), majd továbbítja a felhasználót a /protected/loggedin.jsp nézetre, ahol üdvözlő képernyő várja.

Helytelen jelszó esetén az akció visszairányítja a felhasználót a bejelentkező űrlapra.

/protected/Logout

A kijelentkezés akció megszakítja a felhasználói munkamenetet, kiveszi a felhasználót a bejelentkezett felhasználók listájáról, végül átirányítja a böngészőt az üdvözlőképernyőre.

/protected/DownloadCert

/protected/DownloadCRL

/protected/DownloadSignature

Ez a három akció egy tanúsítványt / visszavonási listát / aláírás állományt küld el a felhasználó böngészőjének. Fontos, hogy az akció beállítsa a következő két HTTP fejléc tartalmát. Az első fejléc értesíti a böngészőt, hogy nem szöveges, hanem bináris adat érkezik, amelyet ne próbáljon közvetlenül megjeleníteni, hanem mentsen fájlba. A második fejléc mondja meg a böngészőnek a fájl eredeti nevét. A két fejléc a következő:

- ContentType=application/octet-stream
- Content-Disposition=”fájlnév”

/protected/Upload

A feltöltés akciót a feltöltésre szolgáló űrlap „feltöltés” gombjával hívja meg a felhasználó. Az űrlapon szerepel egy fájl választó mező, egy típus-választó mező, és egy leírás mező. A böngésző a fájlt is tartalmazó

űrlapokat más formában küldi el a szervernek, mint a többi űrlapot, ezért ez az akció speciális feldolgozást igényel.

A böngészők a „multipart/form-data” MIME típust használják, mivel a W3C HTML szabvány ezt írja elő a számukra. Az adat több részre van osztva. Van paraméter rész, amelyben a nem bájtfolyam-jellegű mezők tartalma található (például a fájlok neve), és van fájl rész, amelyben a kiválasztott fájlok tartalma található. Ezt a struktúrát kell az akció osztálynak dekódolnia, és a fájlt a tanúsítvány vagy az aláírás állomány tárba elhelyezni.

/protected/ReloadSystemParams

Az akció arra utasítja a rendszert, hogy a paramétereit tartalmazó XML állományt olvassa be, és érvényesítse az ott talált beállításokat. A következő paraméterek lehetnek az állományban

- Az időbélyegzés szolgáltatók URL elérhetőségei.
- A kivárási idő, ami után egy aláírás kezdeti ellenőrzése elvégezhető.
- A különféle háttér feldolgozások ütemezési gyakorisága.
- A felhasználói „timeout” ideje, amennyi inaktivitás után a munkamenetet a rendszer megszünteti.

/protected/ArchiveSignature

/protected/InitialCheckSignature

/protected/PosteriorCheckSignature

/protected/ReceiveSignature

/protected/RefreshCRL

/protected/RegisterUser

Ezek az akciók érdemi feldolgozást nem végeznek, mindössze továbbítják a HTTP protokollon érkező kérést a megfelelő Java metódusnak.

8.7. Kódolási és hibakezelési szabályok

- A különböző tárak adatbázishoz forduló metódusai a következő szabályok szerint íródnak.
    - Try-finally blokkban kell adatbázishoz fordulni.
    - A finally blokkban be kell zárni a Statement és a Connection objektumokat.
    - A try blokk elején az adott tár DataSource objektumától kell Connection objektumot kérni.
    - A Statement végrehajtása után kapott ResultSet objektumot használat után be kell zárni.
    - A menet közben keletkező kivételeket az adatbázist használó metódus tovább dobja, ezt a throws kulcsszó után kell deklarálni.
- Általában a metódusok a kritikus kivételt tovább dobják a magasabb szintek felé, és a legfelsőbb szinten történik a hibák kezelése. Nem jó a kivételek „lenyelése” alacsonyabb szinteken.

9. Tesztelési terv

A rendszer tesztelése során a következő vizsgálatokat kell elvégezni.

- Modulok funkcionális tesztelése referencia XAdES állományokkal, és valós szolgáltatói állományokkal.
    - Tanúsítvány és visszavonási lista kezelés. A különböző szolgáltatók tanúsítványait a rendszer képes-e kezelni?
    - Hitelességi lánc helyes felépítése és a lánc hitelességének ellenőrzése.
    - Az állományon szereplő aláírás kriptográfiai ellenőrzése.
    - Az időbélyegek ellenőrzése helyesen működik-e? Nem ad-e hamis negatív eredményt.
- Integrációs teszt a más alkalmazásokkal való együttműködés behangolására különféle alkalmazásokkal előállított XAdES, és valós szolgáltatói állományok segítségével.
    - XAdES feldolgozás rugalmasságának tesztelése. Az aláírás állományok a szabványokon belül is változatos formákat ölthetnek.
    - A kriptográfiai ellenőrző funkciók működése. Rendkívül fontos, hogy az ellenőrzések ne adjanak hamis negatív eredményt. Ez nagyon sokszor elő fog fordulni, hiszen egy bit eltérés az ellenőrzés során is negatív eredményt produkálhat. További problémák forrása lehet, hogy a feldolgozandó állományokat több különböző alkalmazás több különféle verziója állítja elő, így szinte biztosan lesznek eltérések az egyes formátumok között.
- Biztonsági tesztelés. A rendszer támadható-e, illetéktelenek hozzáférhetnek-e.
- Felhasználói felület tesztje annak ellenőrzésére, hogy a felület funkciói megfelelően működnek-e.

10. Értékelés

Az elmúlt egy másfél évben folytatott kutató-fejlesztő tevékenységemet a következőképpen értékelem a feladat megfogalmazásától kezdve a fejlesztés lezárásáig. A következő feladatokat teljesítettem:

- Áttekintettem az elektronikus archiválás lehetőségeit, és a megvalósításból adódó feladatait.
- A hosszú távú archiválás megoldására kerestem egy alkalmas technológiát, majd rendszert tervezte köré.
- A terveket megvalósítottam, a rendszer főbb elemei logikailag helyesen működnek.

Amit fontosnak tartok megjegyezni:

- A rendszer határainak és környezetének meghatározása sok fejtörést okozott, mivel ismert igény nem volt a rendszer felhasználására. Marketing felmérés hiányában a rugalmas, „weben elérhető alkalmazás” koncepció köré építettem a rendszert.
- Az átgondolt tervezés a munka kései szakaszában kezdődött meg, mivel kezdetben a használt technológiákban nem volt gyakorlatom. A rendszer szerkezetét többször átalakítottam, végül az utolsó átírás kapcsán ismertem meg a Jakarta Struts technológiát, amely a végleges formát adta.
- Éles projektnél elengedhetetlennek tartom a megalapozottabb technológiai felkészülést, és a felhasználói igények pontos felmérését.

Összességében elmondható, hogy a kiírásban foglaltakat teljesítettem, és a tervekben szereplő szoftver is működik.

11. Köszönetnyilvánítások

Szeretnék köszönetet mondani:

Dr. Frigó József tanár úrnak, aki előadásai során megismertetett a korszerű szoftverfejlesztés filozófiájával és módszereivel.

Konzulenseimnek, Krasznay Csabának és Szigeti Szabolcsnak, amiért segítettek eligazodni a szabványok és technológiák útveszőiben, majd e hozzám illő feladatot javasolták kidolgozásra, és végül hasznos tanácsokkal segítettek a munkám során.

12. Irodalom

[1] RFC 3275 XML-Signature Syntax and Processing

[2] ETSI TS 101 903 XML Advanced Electronic Signatures (XAdES)

[3] 2001. évi XXXV. törvény az elektronikus aláírásról

[4] 2004. évi LV. törvény az elektronikus aláírásról szóló 2001. évi XXXV. törvény módosításáról

[5] A közigazgatási hatósági eljárásról és szolgáltatásról szóló 2004. CXL törvény

[6] Egységes MELASZ formátum elektronikus aláírásokra; verzió: 1.0

[7] JavaServer Pages, 2nd Edition, Hans Bergsten, O'Reilly 2002.

[8] Aláírási szabályzatra vonatkozó elvárások a magyar elektronikus közigazgatásban. *forrás:* Információs Társadalom Koordinációs Tárcaközi Bizottság, Elektronikus Közigazgatás Albizottság, http://www.itktb.hu/engine.aspx?page=ias

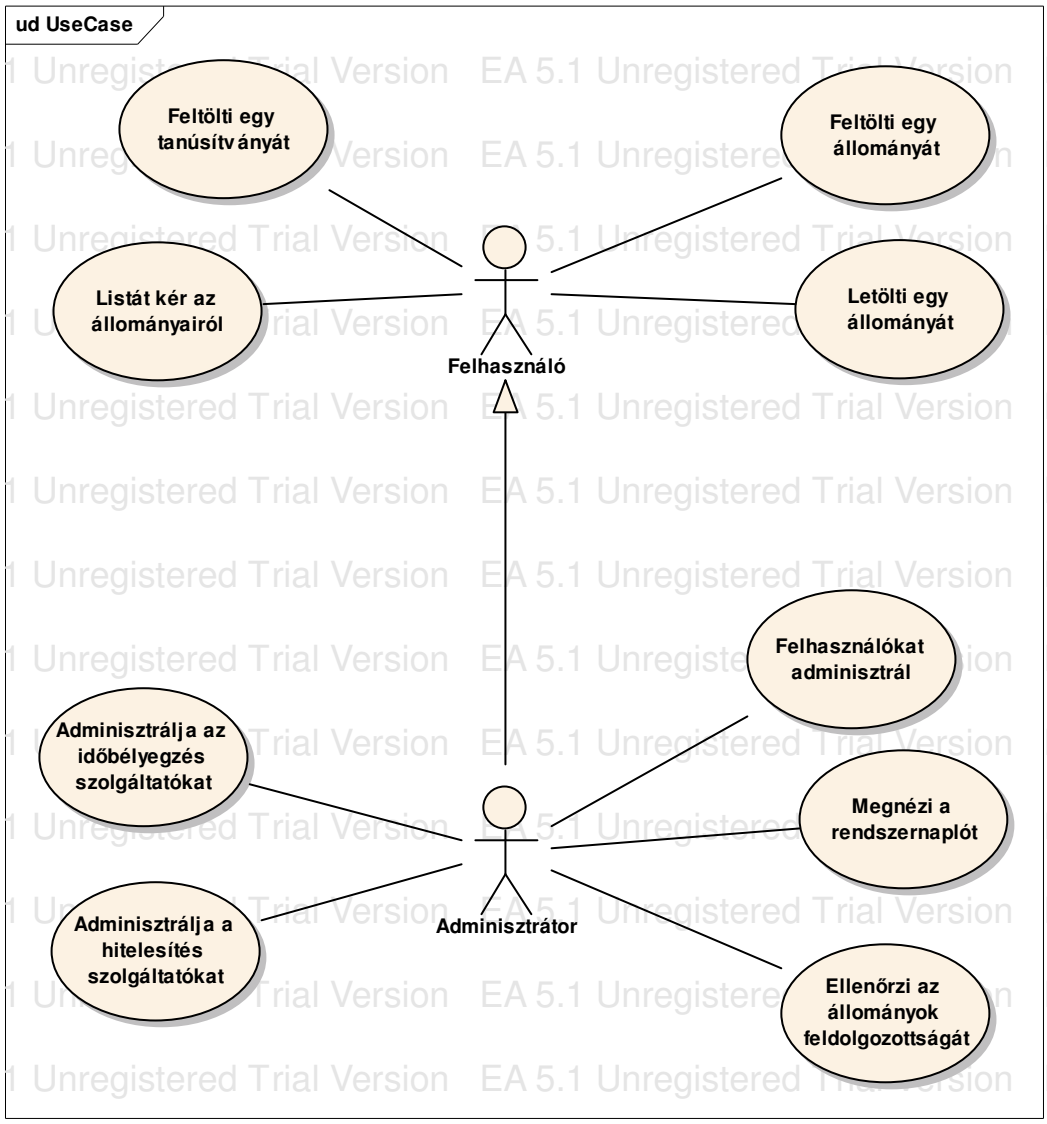
[9] RFC 2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile

[10] http://en.wikipedia.org/wiki/History_of_cryptography

[11] RFC 3161 Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)

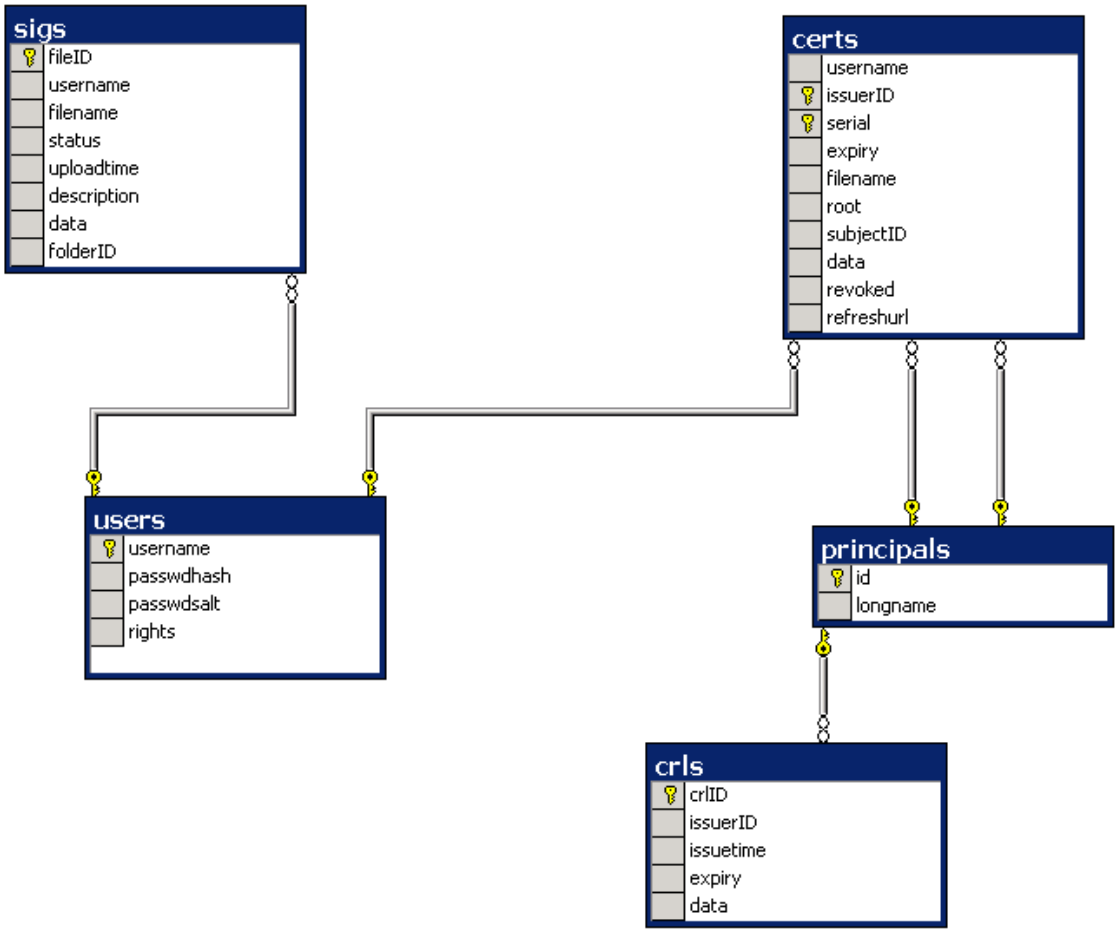
13. Mellékletek

1. melléklet A rendszer használati eset (Use Case) diagramja

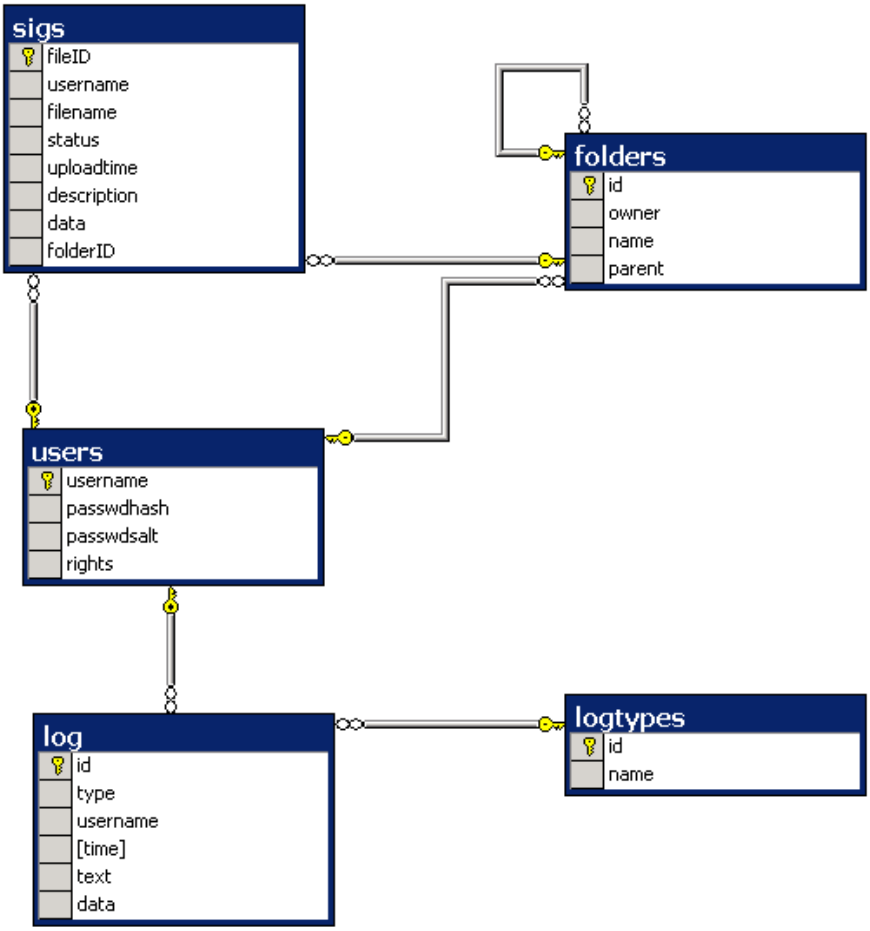


2. melléklet Az XMLDSIG és az XAdES típusok egymásra épülnek

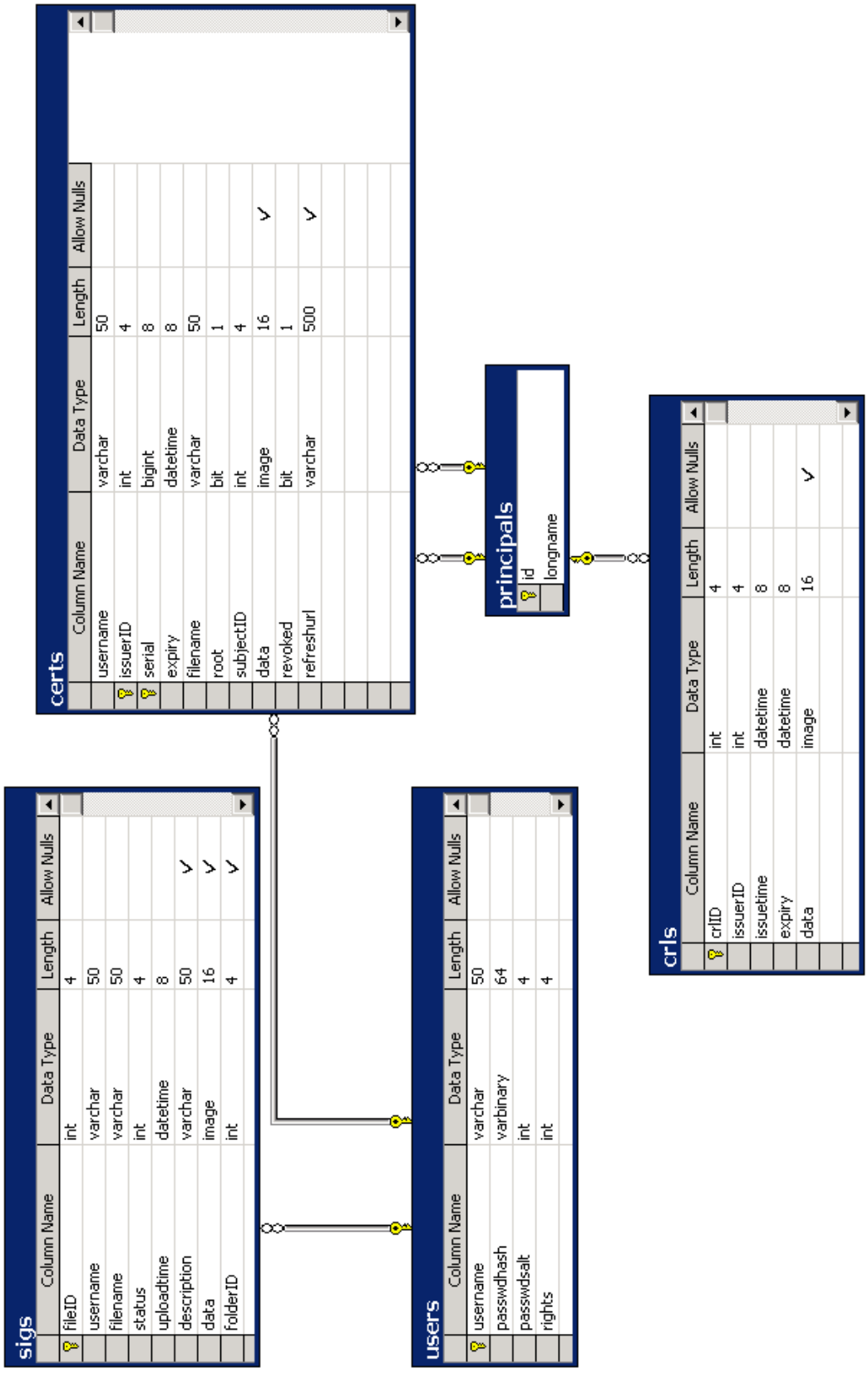
3. melléklet Logikai adatmodell



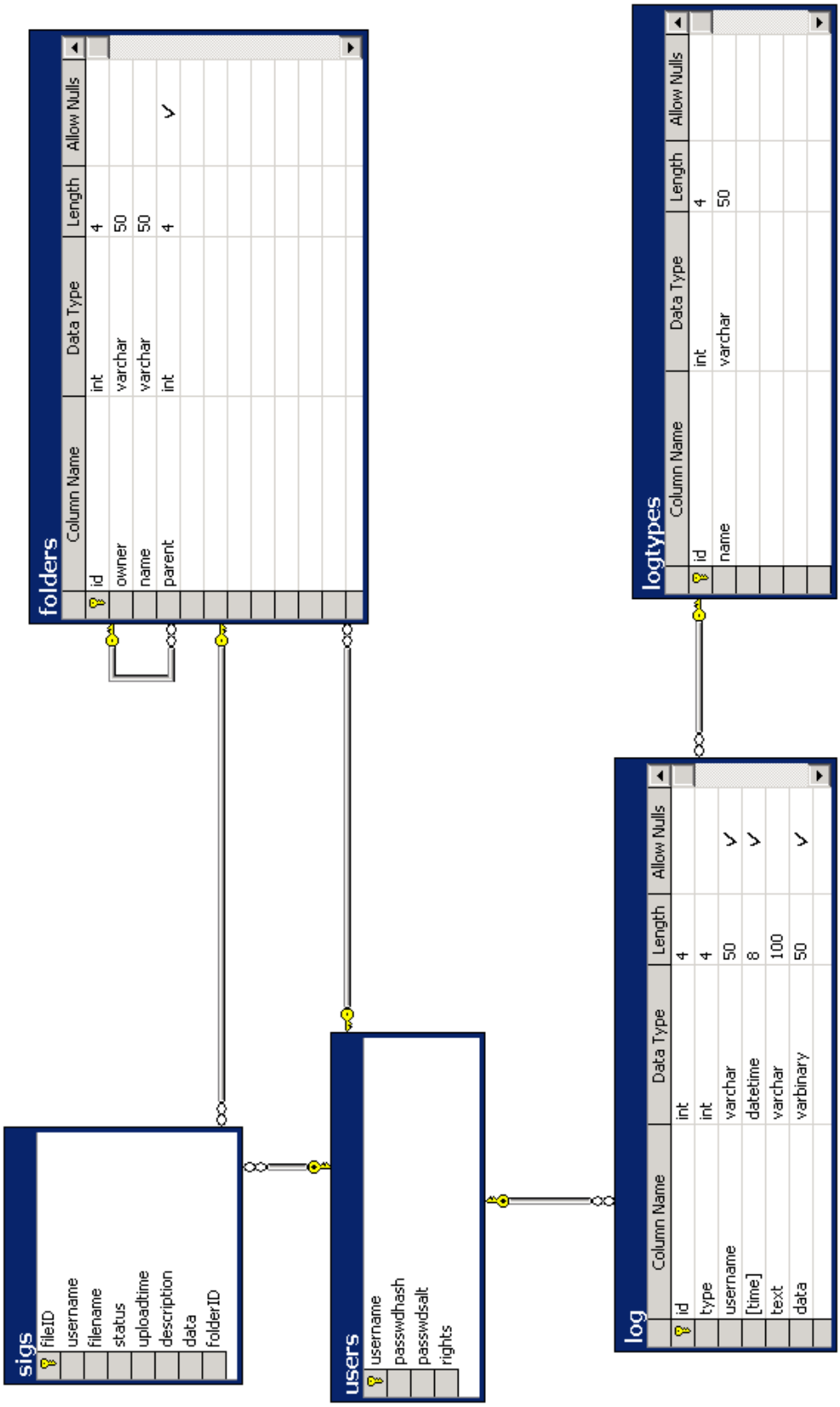
Segédtáblák



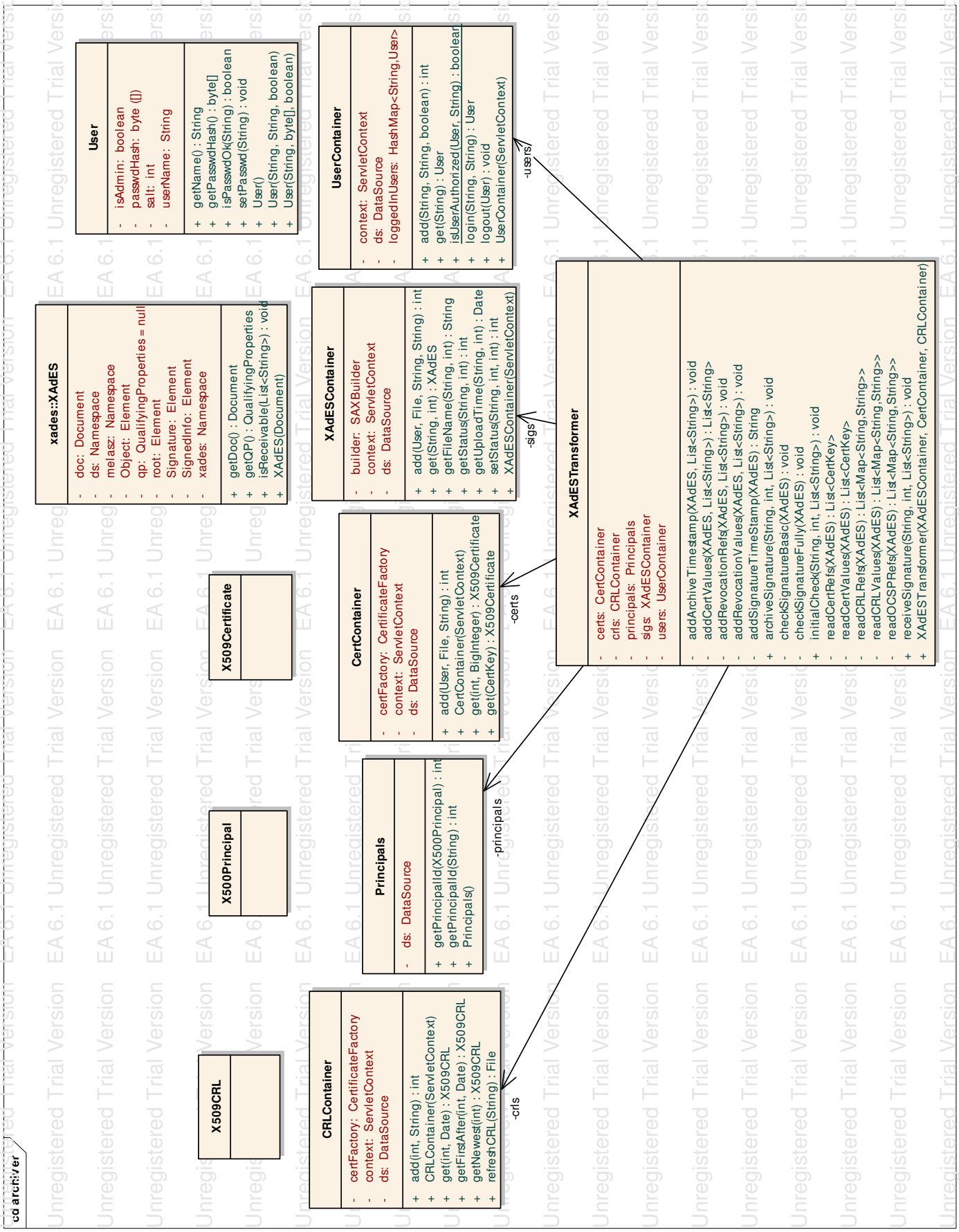
4. melléklet Fizikai adatmodell – Microsoft SQL Server 2000 diagram



Segédtáblák



5. melléklet Osztálydiagram

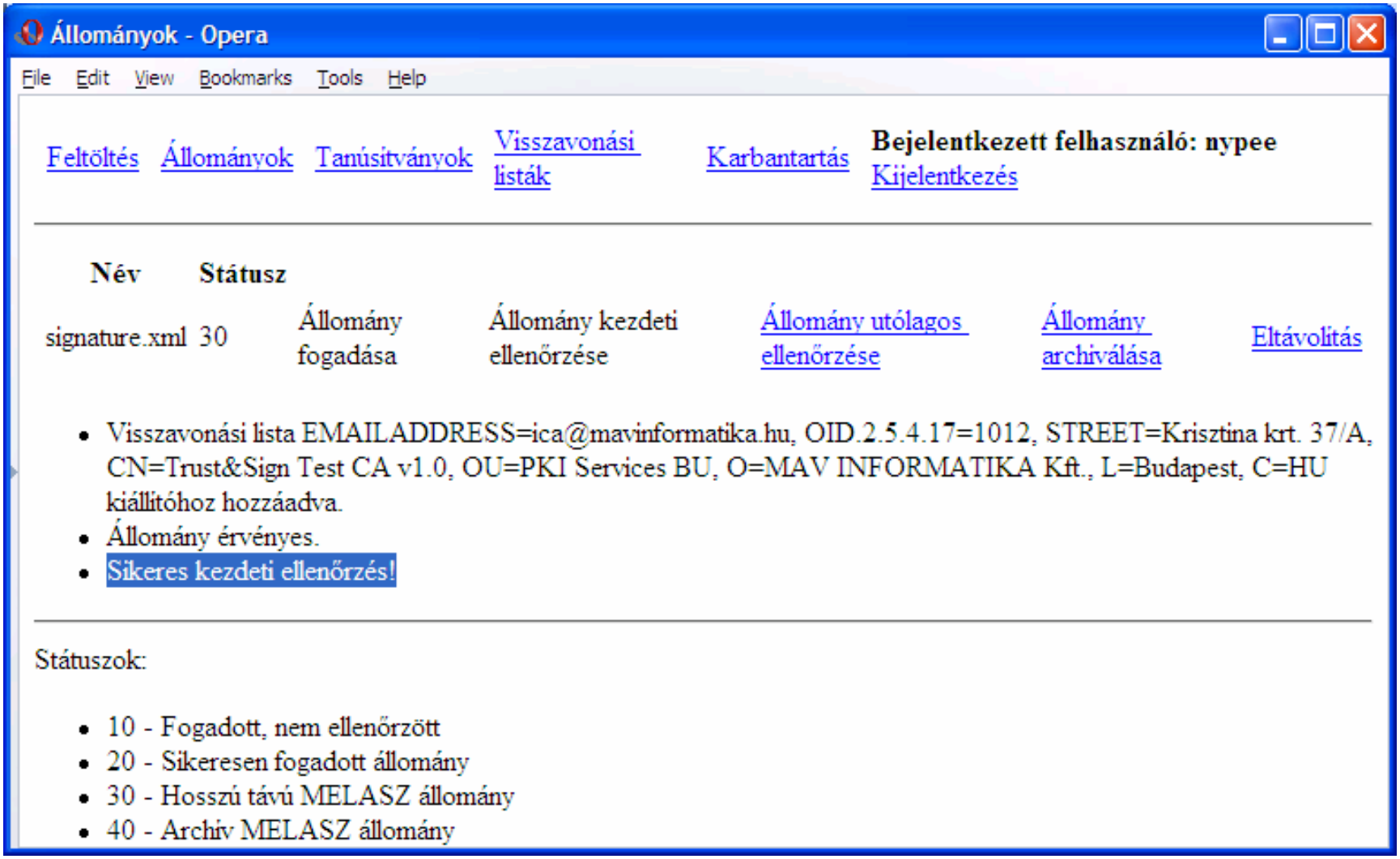


6. melléklet Képernyőtervek

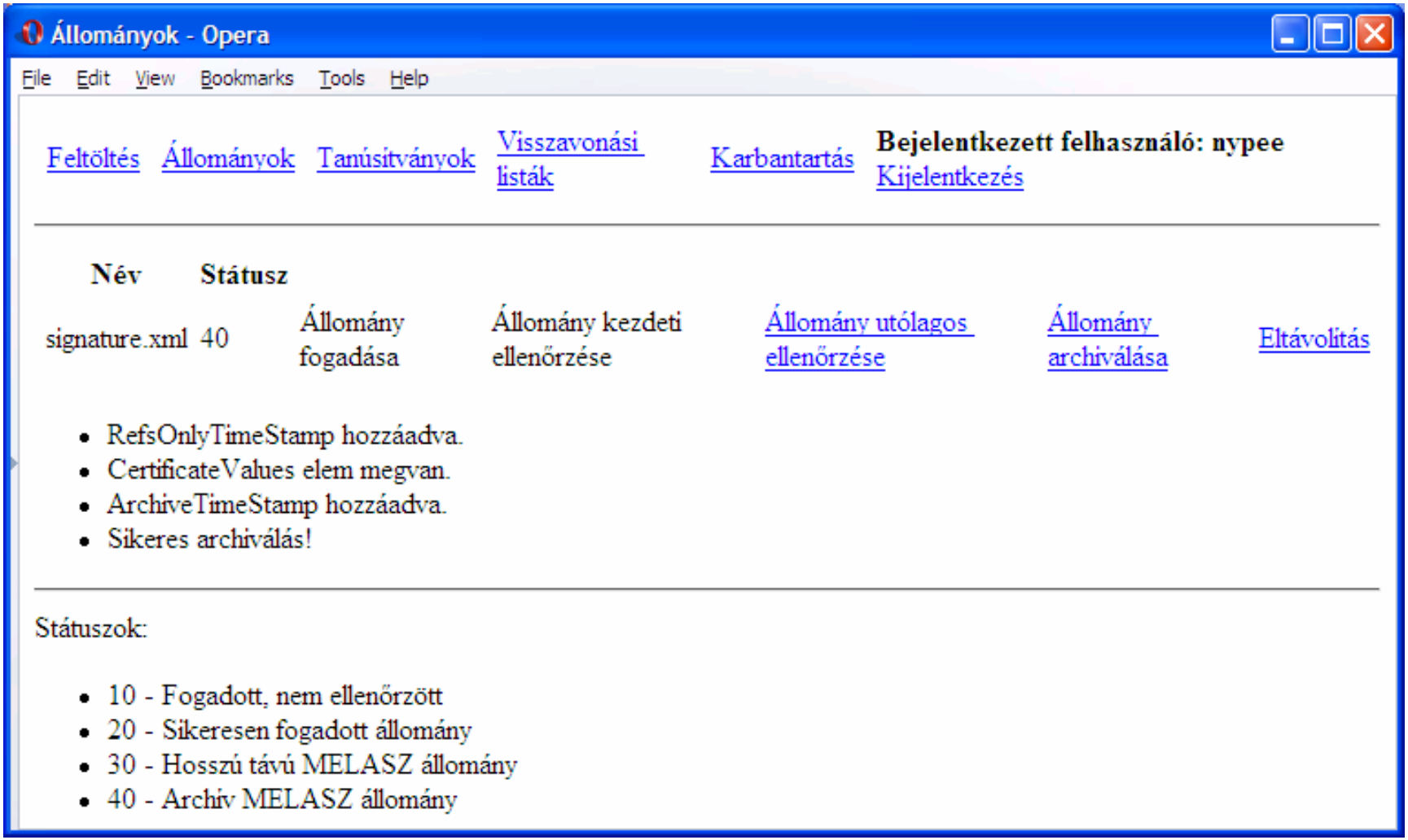
Állomány feltöltése

Állomány állapota feltöltés után

Állomány fogadása


Állományok - Opera
File Edit View Bookmarks Tools Help
Feltöltés Állományok Tanúsítványok Visszavonási listák Karbantartás
Bejelentkezett felhasználó: nypee
Kijelentkezés
Név Státusz
signature.xml 30 Állomány fogadása Állomány kezdeti ellenőrzése Állomány utólagos ellenőrzése Állomány archiválása Eltávolítás
Visszavonási lista EMAILADDRESS=ica@mavinformatika.hu, OID.2.5.4.17=1012, STREET=Krisztina krt. 37/A, CN=Trust&Sign Test CA v1.0, OU=PKI Services BU, O=MAV INFORMATIKA Kft., L=Budapest, C=HU kiállítóhoz hozzáadva.
Állomány érvényes.
Sikeres kezdeti ellenőrzés!
Státuszok:
10 - Fogadott, nem ellenőrzött
20 - Sikeresen fogadott állomány
30 - Hosszú távú MELASZ állomány
40 - Archív MELASZ állomány

Állomány utólagos ellenőrzése


Állományok - Opera
File Edit View Bookmarks Tools Help
Feltöltés Állományok Tanúsítványok Visszavonási listák Karbantartás
Bejelentkezett felhasználó: nypee
Kijelentkezés
Név Státusz
signature.xml 40 Állomány fogadása Állomány kezdeti ellenőrzése Állomány utólagos ellenőrzése Állomány archiválása Eltávolítás
RefsOnlyTimeStamp hozzáadva.
CertificateValues elem megvan.
ArchiveTimeStamp hozzáadva.
Sikeres archiválás!
Státuszok:
10 - Fogadott, nem ellenőrzött
20 - Sikeresen fogadott állomány
30 - Hosszú távú MELASZ állomány
40 - Archív MELASZ állomány