E-VOTING IMPLEMENTÁCIÓ ANDROID PLATFORMON

Önálló laboratórium beszámoló

Miklós Richárd – JOVJ11

Konzulens: Dr. Goldschmidt Balázs

2020. 05. 31.

Tartalomjegyzék

1. Bevezető

1.1. A feladat

Feladatom egy elosztott kriptográfiai alkalmazás fejlesztése volt Android platformon. Ezen belül tetszőleges témát választhattam. Az “e-voting” mellett döntöttem, tehát egy Androidos szavazó alkalmazás implementálása volt a feladat, megfelelő kriptográfiai biztonságot szem előtt tartva.

1.2. Érdeklődés

Már egy ideje nagyon érdekel a kiberbiztonság. Gondolkodtam rajta, hogy az infokommunikáció specializációt választom, mert ott jobban el tudnám mélyíteni az IT biztonság tudásomat, de úgy gondoltam, hogy a szoftverfejlesztés egy biztosabb út, ami általánosságban közelebb áll hozzám, mint az infokommunikáció.

Ezért tehát megörültem, amikor megláttam ezt az önálló laboratórium témát.

1.3. Felkészülés

Azzal kezdtem a munkát, hogy megismerkedtem a kriptográfia alapjaival.

Rövid kutakodás után arra jutottam, hogy ennek a legjobb módja kivételesen nem az internet végtelen böngészése, hanem egy olyan írás elolvasása, ami egy, a szakterületén elismert, tapasztalt ember tollából származik. Ezt figyelembe véve olyan könyvet kerestem, ami nem egy-két kiválasztott témáról ír, hanem a legtöbb kriptográfiai alapfogalmat részletekbe bemenően kifejt. Az ajánlások különböző fórumokon végül elvezettek egy sokat emlegetett könyvig, aminek a fentiek mellett hatalmas előnye, hogy csupán 3 éve adták ki. Ez azért fontos a kriptográfia esetén, mert így kevesebb eséllyel tartalmaz elavult technikákat, protokollokat, tehát nagyrészt *state-of-the-art* megoldásokat mutat a problémákra.

Választásom Jean-Philippe Aumasson *Serious Cryptography* című könyvére esett. Megismertem alapvető fogalmakat, a kriptográfia építőköveit (blokkrejtjelezés, hashek, autentikált titkosítás, szimmetrikus, aszimmetrikus kulcsú rejtjelezés, kulcscsere algoritmusok, stb.), valamint egy komplett példát, ami ezen különálló eszközökből épít fel egy összetett protokollt, a TLS-t.

Ekkor ugyan még nem volt tervem arra, hogy pontosan hogyan fogom megvalósítani az e-voting alkalmazásom kriptográfiai részét, de elkezdtem implementálni egy szimmetrikus kulcsú rejtjelezést használó teszt programot, ami a Google Firebase adatbázisa segítségével tárol titkosított üzeneteket, majd azok lekéri, és dekódolja őket. Ez csupán egy ízelítő volt, hogy hogy is fog zajlani az implementációja a valós feladatnak. Hamar rájöttem, hogy kell egy konkrét terv. Egy e-voting protokoll helyes

megtervezése nem kezdőnek való feladat. Számos, a mai napig elméletben működő protokoll létezik, amiket különböző szakemberek felmutattak az elmúlt néhány évtizedben. Mivel a feladatom fókusza az implementáció volt, ezért választottam egy kész protokollt, amit már csak le kell fejleszteni.

A következő fejezet az elektronikus szavazásról fog szólni, valamint a téma kriptográfiai hátterét és az általam választott protokollt mutatja be.

2. E-voting

2.1. Motiváció

Általánosságban elmondható, hogy a szavazói részvétel folyamatosan csökken világszinten a nyolcvanas évek óta [1]. A szavazói részvétel csökkenése egy demokráciában nem kívánt jelenség. Habár valószínűleg számos oka van ennek a jelenségnek, legtöbben egyetértenek, hogy azt ellensúlyozná egy, a szavazás menetét megkönnyítő, és kényelmesebbé tevő technikai újítás, mint az elektronikus szavazás, ha az megfelelően van kivitelezve.

2.2. Problémák

Egy e-voting rendszer kivitelezésének sajnos vannak technológiai és társadalmi akadályai is.

Utóbbira példa Németország: 2005-ben zajlott le az első elektronikus szavazás, ahol olyan szavazógépeket használtak, amik majdnem megegyeztek a géppel, amit 2006-ban sikeresen feltörtek holland hackerek [2]. Ezt követően 2006-ban különböző óvintézkedésekkel bővítették a választásokat, és egy tervet is készítettek a jövőbeli szavazásokra. A terveket viszont keresztezte, hogy 2007-ben több forrás publikálta eredményeit arról, hogy a rendszer sebezhető volt, és egy általános bizalmatlanság alakult ki az emberekben a rendszerrel kapcsolatban. Ennek eredményeképp a terveket elvetették, és 2009-ben Németország visszatért a papíralapú szavazáshoz.

Az új technológiában való bizonytalanság visszavezethető annak gyakorlati eredményeire, azaz a technológiai akadályokra.

Online szavazás esetén egy egyszerű DDoS támadással meg lehet akadályozni a rendszer működését. Az efféle támadás kivédésére léteznek megoldások, de ezek magukkal hoznak újabb problémákat [3]. Egy ilyen úgynevezett „DDoS mitigation” védekezés azzal jár, hogy a szolgáltatást nyújtó fél lényegében dekódol minden kérést a szavazó szerver felé, és megvizsgálja azt, hogy valós-e a kérés. Ahhoz, hogy ez működjön, a privát TLS kulcsnak világszerte különböző szervereken jelen kell lennie. Ez rengeteg úgynevezett „single point of failuret” (SPOF) hoz létre, mert ha egy ilyen szervert feltörnek, és hozzájutnak a privát TLS kulcshoz, akkor lényegében akár módosíthatják is a szavazás eredményét.
Pont ez történt 2017-ben Ausztráliában [4].

Egy másik eset, ezúttal Észtországban [5], pedig többek között azt mutatja, hogy a szoftveres és hardveres implementáción felül fontos, hogy a rendszert üzemeltető személyek is rendelkezzenek megfelelő tudással és felügyelettel, annak érdekében, hogy megfelelően működjön a rendszer.

Végül, de nem utolsó sorban, a teljes biztonság elérését szinte lehetetlenné teszi a tény, hogy a dedikált szavazógépek kompromittálhatók. A 2019-es DEF CON (legnagyobb hacker konferencia) jelentéséből kiderül [6], hogy a résztvevő hackerek sikeresen feltörték mind a 100 szavazógépet, amit eléjük tettek.
Ez a probléma online szavazás esetén még jobban erősödik, mert a szavazók különféle eszközei számtalan lehetőséget nyújtanak a támadók számára.

2.3. Követelmények

Egy e-voting rendszer biztonsági követelményeit többféle módon meg lehet adni, de alapvetően egy ideális protokoll esetén az alábbiak fogalmazhatóak meg (a szakmájában elismert kriptográfus, Bruce Schneier, 1996-os *Applied Cryptography* c. könyvéből) [7]:

1. Csak autorizált szavazók szavazhatnak.
2. Senki sem szavazhat egynél többször.
3. Senki sem derítheti ki, hogy valaki más kire szavazott.
4. Senki sem duplikálhatja valaki más szavazatát.
5. Senki sem változtathatja meg valaki más szavazatát, anélkül, hogy erre fény derülne.
6. Minden szavazó meg tud bizonyosodni arról, hogy beleszámították a szavazatát a végső eredménybe.

2.4. Központi és elosztott szavazás

2.4.1. Ismertető

Az e-voting protokollok két nagy csoportra oszthatóak. Központi és elosztott alapúra. Mindkét oldalnak vannak előnyei és hátrányai, valamint mindkét oldal esetében létezik olyan helyzet, ahol azt jobban preferálják.

2.4.2. Központi rendszerek

Központi szavazási rendszernek nevezik az olyan protokollon alapuló rendszereket, amelyek a hagyományos, papír alapú szavazáshoz hasonlóan egy vagy több központi egység segítségével bonyolítják le a szavazás menetét.

Az ilyen rendszerek legnagyobb hátránya az, hogy a központi egységeket nyújtó szervek túl nagy hatalommal rendelkeznek a szavazás felett.

Tegyük fel, hogy a valóságban látottakkal ellentétben a rendszer szoftveres és hardveres megvalósítása teljesen *open source*, és valamilyen *checksum*mal van

publikálva. Ekkor elmondható, hogy nagy eséllyel ellenőrizhető a központi szerv becsületessége, feltéve, hogy nem történt checksum hamisítás.

Ettől függetlenül a központi szervek hatalma megmarad, mert például bármikor dönthetnek úgy, hogy nem publikálják a szavazás eredményét, valamilyen hamis indoklás mellett.

A központi rendszerek előnye viszont, hogy jól skálázódnak.

2.4.3. Elosztott rendszerek

Elosztott szavazási rendszernek nevezik az olyan protokollon alapuló rendszereket, ahol a szavazók *peer-to-peer* kommunikálnak egymással, nincs semmiféle központi szerv.

Ezek általában *blockchain* alapú protokollok. A blockchain, hasonlóan a kriptovalutánál való alkalmazásához, az e-votingnál is hasonló előnyöket nyújt.

Legfőbb előnye, hogy csak magában a szoftverben kell megbíznia a szavazónak, ami ebben az esetben *open source.* Nincs tehát potenciálisan "tisztességtelen" (dishonest) központi szerv.

Emellett az egész szavazási folyamat sokkal olcsóbban is kivitelezhető.

Hátránya, hogy a bizalom problémáját most a szavazóknak egyénileg kell megoldaniuk, ami nagyobb számítási igénnyel jár. Országos méretű szavazásokon tehát ez a megközelítés sajnos nem skálázódik jól, viszont például céges / egyetemi környezetben használják.

2.5. FOO protokoll

2.5.1. Ismertető

Az általam választott "FOO" protokoll [8] egy központi szavazási rendszert vázol fel, ami a korábbi protokollok által nyújtott előnyöket tartalmazza anélkül, hogy azok különféle problémákat idéznének elő. Két központi, egymástól független egységet (adminisztrátor, számláló) használ a céljai elérésére.

Nevét alkotóiról kapta: Atsushi Fujioka, Tatsuaki Okamoto, és Kazuo Ohta.

A három japán származású alkotó 1992-ben, az éves *Advances in Cryptology* konferencián publikáltak egy papírt a protokollról.

A papírban állítják, hogy a felvázolt szavazási rendszer megállja a helyét nagyobb részvételi számú választásokon, és megoldja a korábbi protokollok alábbi problémáit:

- Az adminisztrátor és a számláló tisztességtelen összefogása esetén sem kell a szavazónak felfednie titkai (pl. szavazatát) annak érdekében, hogy bizonyítsa a szavazatáról, hogy az érvényes és mégsem lett számlálva.
- Becsületességet biztosít azzal, hogy senki sem tudhatja meg a választás vége előtt a részeredményeket.
- Továbbá a szavazó és az adminisztrátor sem tud csalni.

A protokollnak van viszont egy hátránya, ami elég kényelmetlenné teszi a szavazás menetét a felhasználó számára, ugyanis kétszer kell interakcióba lépnie a rendszerrel. Egyszer amikor elküldi a szavazatát, és egyszer a szavazás lezárása után, amikor "kinyitja a céduláját tartalmazó borítékot".

2.5.2. Működés

A protokoll feltételezi, hogy már létezik egy publikus kulcsú rejtjelező infrastruktúra, ahol minden szavazó, valamint az adminisztrátor is rendelkezik egy-egy kulcspárral, amiknek publikus része nyilvános mindenki számára. Előkövetelmény továbbá egy anonim kommunikációs csatorna a szavazók és a számláló között.

Nagy vonalakban a következő lépések szerint zajlik egy szavazás a szavazó szempontjából [9]:

1. Választ egy jelöltet, készít egy "borítékolt" szavazócédulát, majd aláírja azt, miközben elment magának egy titkos, véletlenszerű számot, ami a "borítékolás" melléktermeke volt.
2. Autentikálja magát az adminisztrátornál, és szerez egy "vak aláírást" (blind signature) a szavazócédulára.
3. Elküldi az adminisztrátor által aláírt cédulát egy anonim csatornán a számlálónak.
4. Miután a szavazás lezárult, elküldi az első lépésben generált titkos számát a számlálónak, szintén anonim csatornán.

Ezekhez a lépésekhez a protokoll a következő kriptográfiai eszközöket (primitíveket) használja: digitális aláírás, vak aláírás (blind signature), elkötelezés (commitment scheme).

Digitális aláírás:

Egy digitális aláírás séma három műveletből áll.

Az első művelet generál egy kulcspárt a választott publikus kulcsú rejtjelezés szerint.

A második művelet az aláírás, ami vesz egy dokumentumot és a generált privát kulcsot inputként, és visszatér egy aláírással. Ekkor az aláírást végző fél eljuttatja a dokumentumot, valamint mellette az új aláírást egy másik, fogadó félnek.

Ezután amikor a másik fél megkapja a dokumentumot és az aláírást, veszi ezekhez a küldő fél által az első műveletben generált publikus kulcsot. Ez a három dolog lesz a harmadik művelet, az ellenőrzés bemenete. Kimenete 1 ha az aláírás érvényes, 0 ha érvénytelen.

Vak aláírás:

A vak aláírás a digitális aláírásra épít, ahhoz két további műveletet adva: az egyik eltakarja az üzenetet, a másik felfedi. A két művelet értelmezhető egy boríték lezárásának és felnyitásának, abban az értelemben, hogy nem látszik mi van benne.

Az eltakarás (blinding) művelet bemenetei az aláíráshoz generált publikus kulcs, és az eltakarni kívánt üzenet, kimenetei pedig az eltakart üzenet, valamint egy véletlenszerű szám (unblinding factor), ami a felfedéshez lesz szükséges.

A felfedés (unblinding) bemenetként kapja az eltakaráshoz használt publikus kulcsot, a közben aláírt eltakart üzenetet, és az eltakaráskor generált véletlen számot.

A módosított digitális aláírás tehát így zajlik a FOO protokoll esetében:

1. Kulcsgenerálás (előfeltétele a protokollnak, pl. szavazó regisztrálja magát az adminisztrátornál)
2. Szavazó eltakarja (blind) a szavazatát, elküldi azt az adminisztátornak
3. Adminisztrátor aláírja az eltakart üzenetet, elküldi az aláírást a szavazónak
4. Szavazó felfedi (unblind) az aláírást
5. Ellenőrzés (számláló végzi)

Elkötelezés (commitment):

Egy commitment séma három műveletből áll. Felkészülés, elkötelezés, felnyitás.
Itt is alkalmazhatjuk a boríték analógiát [9]. Az első két művelet során egy átlátszatlan borítékba helyezzük az üzenetet, majd azt átadjuk egy kezelőnek. Ekkor tehát senki sem tudja elolvasni az üzenetet, valamint azt megváltoztatni sem lehet (elkötelezés). Felnyitni csak az tudja a borítékot, aki elhelyezte benne az üzenetet.

A felkészülés (setup) művelet generál a kimenetére egy paramétert.

Az elkötelezés (commit) művelet bemenetként kapja a generált paramétert, és az üzenetet, majd kiadja az elkötelezett "borítékot" (commitment), és egy felnyitáshoz szükséges kulcsot.

A felnyitás (open) művelet bemenetként várja az első művelet által generált paramétert, az elkötelezett üzenetet, az eredeti üzenetet, és a második műveletben kapott kulcsot. Kimenete pedig 1, ha a kapott eredeti üzenet valóban az elkötelezett üzenethez tartozik, ellenkező esetben 0.

A fentiekben látott primitívekkel a protokoll kriptográfiai biztonsága megoldható. A következő fejezetben az implementáció állomásairól és nehézségeiről lesz szó, valamint a kész program bemutatásra kerül.

3. Implementáció

3.1. Áttekintés, nehézségek

3.1.1. Fókusz

Fő fókuszom a félév során a protokoll helyes implementációja volt. Csak miután meggyőződtem arról, hogy az helyesen működik, kezdtem el a felhasználói felület fejlesztését az addig csupán parancssoros megjelenés helyettesítésére.

3.1.2. Komponensek kommunikációja

Az általam eredetileg elképzelt alkalmazás komponensei Firebase adatbázis segítségével kommunikáltak volna egymással. Főként azért gondolkodtam ilyen megoldásban, mert eredetileg ott szerettem volna publikálni a szavazások eredményeit is, így azt hittem kényelmes lesz.
Rövid használat után viszont túl komplikálttá, feleslegessé vált ez a megoldás, mert minden üzenettípust az adatbázis struktúra egy külön ágára kellett volna küldeni ahhoz, hogy azt értelmezni lehessen. Egyébként sem tűnt helyes megoldásnak függeni egy külső adatbázistól egy kriptográfiai alkalmazás esetén.

Áttértem tehát egy sokkal tisztább megoldásra, ez pedig a java socketes kommunikáció volt. Az adminisztrátor és a számláló is (programomban ezeket "authority"-nek és "counter"-nek nevezem) egyaránt megnyit egy-egy szerver socketet különböző portokon. A kliens ezekre csatlakozik, ha kommunikálni szeretne a központi egységekkel.

3.1.3. Bouncy Castle

A fejlesztés során több olyan akadályba is ütköztem, aminek forrása a Java alapértelmezett kriptográfiai *provider*ének (a *Sun*-nak) hiányossága volt. Nem csak a szerver egységek, hanem az Androidos kliens esetén is fennállt ez a probléma, mert utóbbi is a standard Java könyvtárat használta. A hiányosság legnagyobb része a protokollhoz szükséges két fő kriptográfiai primitívben merült ki: a vak aláírásban és az elköteleződési sémában. Mivel ezeket valószínűleg kezdőként nem tudnám hiba nélkül implementálni, úgy döntöttem, hogy egy külső könyvtárat használok a legtöbb kriptográfiai művelet elvégzéséhez.

Ez volt a Bouncy Castle. Ausztrál készítői megelégelték, hogy minden alkalommal amikor új cégnél dolgoztak újra fel kellett találniuk bizonyos kriptográfiai könyvtárakat, ezért megalkották ezt az összetett API kollekciót [10].

3.1.4. Szövegből kód, kódból szöveg

Talán a legtöbb időmet felemésztő része volt az alkalmazásom készítésének a publikus és privát kulcsok szöveges tárolása, pontosabban azok szöveges reprezentációja és a Java objektumok közötti átváltás.

Az egyszerűség végett "beégetett" kulcs stringeket használtam a fejlesztés során a generálás helyett. Ez sajnos még mindig így van (lásd 3.3. fejezet).
Továbbá az adminisztrátor tárolja az összes regisztrált szavazó publikus kulcsát szöveges (string) formátumban, de azokat használni is szeretné byte vagy objektum formában. Ez a két ok arra, hogy különböző átváltásokat használtam.

Alapvetően minden kulcsot PEM (Privacy-Enhanced Mail) formában tárolunk, ami Base64 kódolást használ a tényleges kulcs megjelenítésére, valamint egyértelműen fejlécekkel jelöli a kulcs elejét és végét (1. ábra). Ezen kívül létezik még a DER bináris forma, és az XML forma.

```
-----BEGIN FOO BAR KEY-----
MIIBgjAcBgoqhkiG9w0BDAEDMA4ECKZesfWLQOiDAgID6ASCAWBu7izm8N4V
2puRO/Mdt+Y8ceywxiC0cE57nrbmvaTSvBwTg9b/xyd8YC6QK7lrhC9Njgp/
...
-----END FOO BAR KEY-----
```

Egy PEM fájl sablonja

A nehézség abból adódott, hogy a kulcsokat többféle módon lehet tárolni, ezért azok PEM megjelenítései is különbözők.

Publikus RSA kulcs esetében, ha csupán a nyers kulcsot tároljuk, akkor azt a *PKCS#1* szabvány szerint ajánlott tenni. Ilyenkor a PEM fejléce és lábléce sorra 5-5 kötőjelbe zárt "BEGIN RSA PUBLIC KEY" és "END RSA PUBLIC KEY".

Azonban később megjelent egy olyan tárolási módja a publikus kulcsoknak, ahol a nyers kulcs mellett tárolunk különböző információkat is a kulcsról. Ezt hívják *X.509* szabványnak.
RSA esetében ez csupán azt jelenti, hogy tároljuk azt az információt, hogy milyen algoritmussal van kódolva a kulcs (RSA), és magát a kulcsot. Ez a két információ van Base64 kódolva PEM esetében. Egy X.509 *certificate*-en belül a nyers kulcs még mindig PKCS#1 formában van.
X.509 RSA esetén a PEM fejléc és lábléc kötőjelek közé zárt "BEGIN PUBLIC KEY" és "END PUBLIC KEY", valamint maga a Base64 string hosszabb.

Privát RSA kulcsokat, hasonlóan a publikus kulcsokhoz, lehet egy nyersebb, valamint egy X.509-hez hasonló összetett formában tárolni.

A nyers formát itt is a PKCS#1 szabvány írja le. PEM fejléce és lábléce sorra "BEGIN RSA PRIVATE KEY" és "END RSA PRIVATE KEY".

Az összetett formát a *PKCS#8* szabvány írja le. Értelemszerűen itt is hosszabb lesz a PEM, a fejléc és lábléc pedig sorra "BEGIN PRIVATE KEY" és "END PRIVATE KEY".

A különböző formák különböző Java és Bouncy Castle osztályok segítségével kezelendőek, de szerencsére a Bouncy Castle nyújt lehetőséget a nyers és az összetett formák közötti átváltásra.

3.1.5. Állapot tárolása

A szerveregységek állapotait célszerű lenne adatbázisokban tárolni, és adatbázis műveletekkel hozzáférni azokhoz. Én egy egyszerűbb, kevésbé hatékony megoldást használtam: JSON fájlokat.

Az adminisztrátor (authority) 3 fájlban tárolja állapotát. Ezek a *polls*, *voters*, és *votes* fájlok.
A polls tartalmaz minden szavazást a következő felépítéssel: szavazás azonosítója, neve, lejárati ideje, jelöltek nevei, szavazás résztvevői.
A voters tartalmazza a szavazókat: szavazó azonosító, szavazó publikus aláírás kulcsa.
A votes tartalmazza azokat a szavazatokat, amiket az adminisztrátor elfogadott, és aláírt. Egy szavazat felépítése: szavazás azonosító, szavazó azonosító, klienstől kapott vak aláírással eltakart elkötelezés (blinded commitment), valamint az ehhez tartozó aláírás (adminisztrátor aláírása).

A számláló (counter) 4 fájlban tárolja az állapotát: *ballots, already_opened, valid_votes, tally.*
Ballots minden szavazáshoz eltárolja a kliens által küldött, adminisztrátor által aláírt elkötelezést, feltéve, ha az aláírás sikeresen validálva lett. Egy szavazócédula (ballot) felépítése: cédula azonosító, elkötelezés, elkötelezéshez tartozó adminisztrátori aláírás.
Az already_opened tartalmazza azokat a cédulákat, amik már "ki lettek nyitva" a szavazó által (szavazó második interakciója a rendszerrel). Minden szavazáshoz tárolva van egy cédula azonosító lista.
A valid_votes minden szavazáshoz tárolja a szavazatokat, amiket sikeresen kinyitott cédulák tartalmaztak. Egy bejegyzés felépítése: jelölt neve, érkezett szavazatok száma.
A tally fájl felépítése ugyan az mint a valid_votes-é, a különbség a kettő között az, hogy ez csak a lezárt szavazásokat tartalmazza.

A kliens nem ment el semmilyen állapotot, a szavazásokat mindig lekérdezi az adminisztrátortól.

3.2. Működés

Az alkalmazás ideális működése a következőképpen zajlik.

Amint az adminisztrátor és a számláló szerverek elindulnak, betöltik az állapotukat, majd várnak a kliens csatlakozására:

```
Reading voters file completed successfully.
Reading polls file completed successfully.
Reading voters file completed successfully.
```

Adminisztrátor: állapot betöltése

```
Reading ballots file completed successfully.
Reading valid votes file completed successfully.
Reading 'already opened' file completed successfully.
```

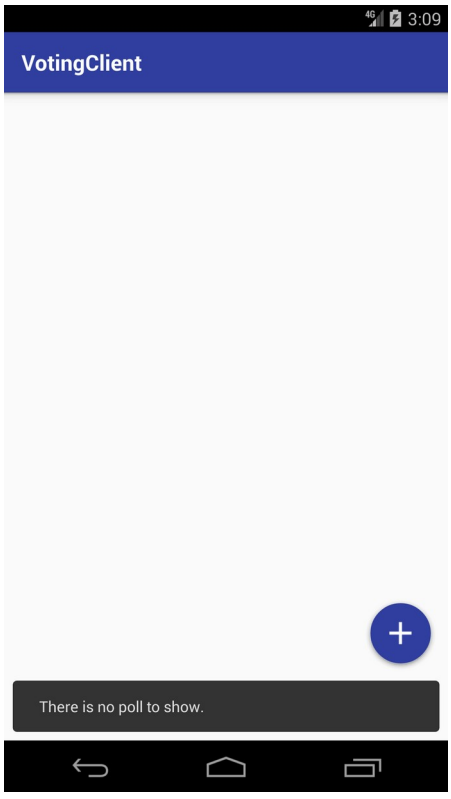
Számláló: állapot betöltése

```
Waiting for client to connect...
```

Admin és számláló: kliensre vár

Az állapotbetöltés során olyan index (hash) struktúrákat hoznak létre, amik megkönnyítik az azonosító alapján való keresést, legyen szó a szavazókról, szavazásokról, vagy cédulákról.

Amint egy kliens elindítja az Androidos applikációt, az lekérdezi az adminisztrátortól a szavazások listáját.

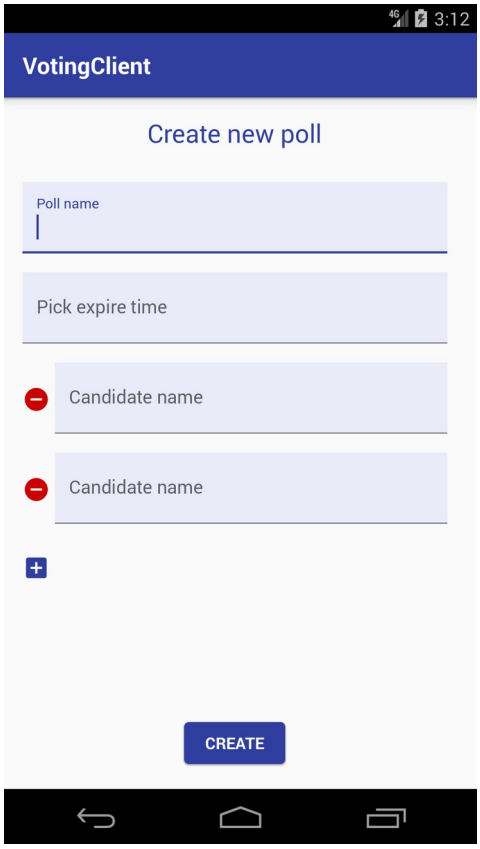


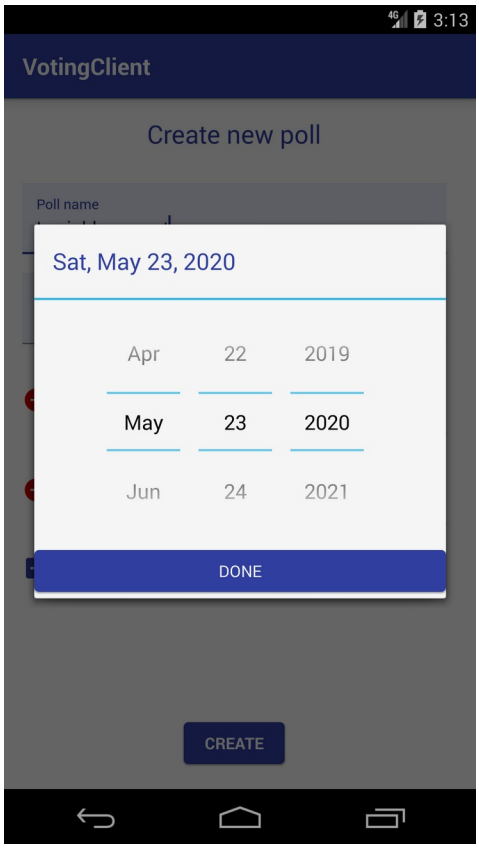
Kliens: főoldal

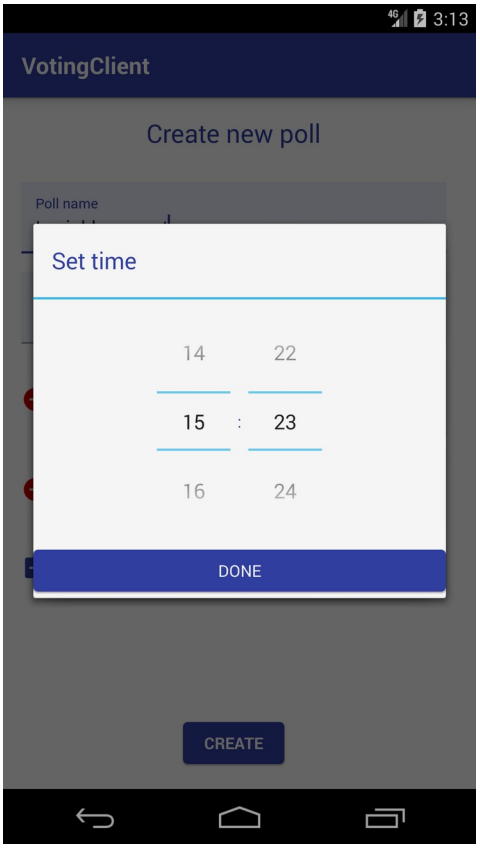
```
Client connected
Waiting for client to connect...
Waiting for data...
Received data from client.
Sending polls to client...
Polls sent
```

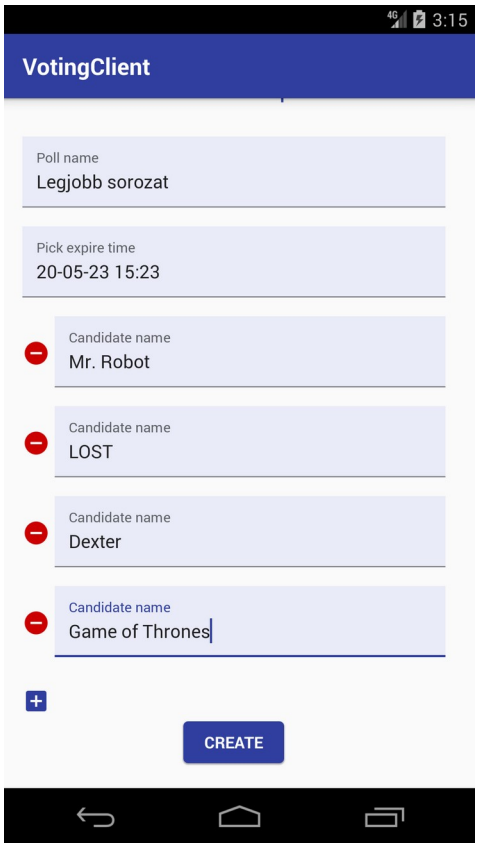
Admin: szavazások lekérdezése

A kliens létrehozhat egy új szavazást, ha a főképernyőn rányom a + gombra. A létrehozás oldalon meg kell adnia a szavazás nevét, egy lejárati időpontot, valamint a jelölteket. A + gombbal új jelölteket adhat hozzá. A "CREATE" gombra kattintva a kliens elküldi a létrehozási kérést az adminisztrátornak, majd visszakerül a főképernyőre.



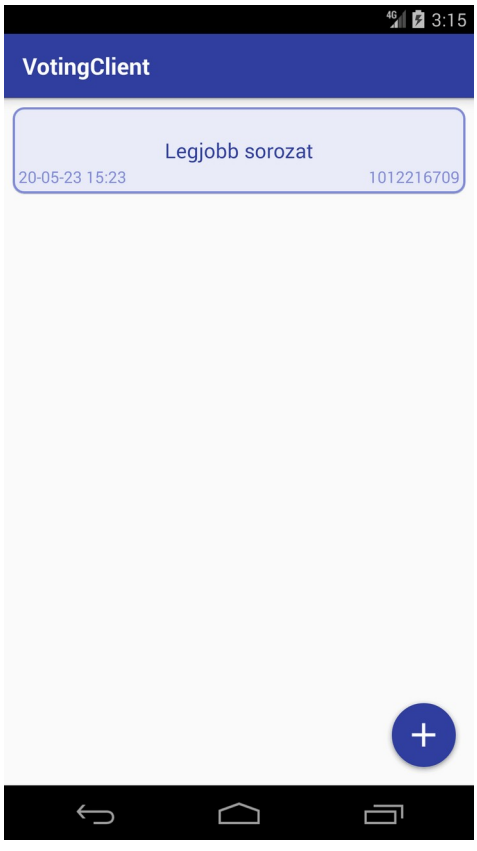




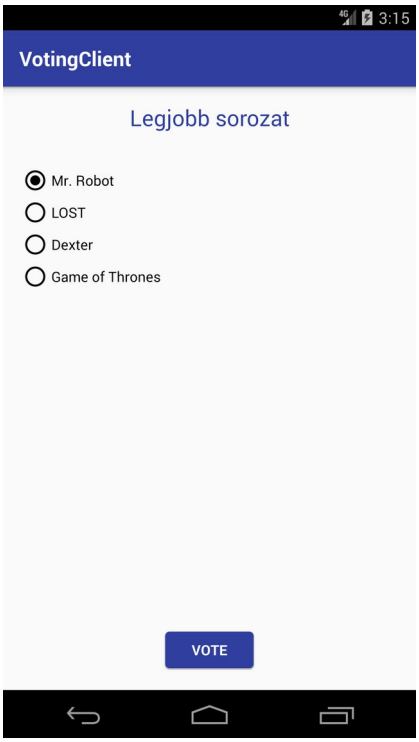



```
Client connected
Waiting for client to connect...
Waiting for data...
Received data from client.
Poll created with ID: 1012216709
Writing polls file completed successfully.
```

Admin: új szavazás lértehozása



Ha egy kék színnel jelölt aktív szavazásra kattint a felhasználó, akkor eljut a szavazás képernyőre. Itt csupán választania kell egy jelöltet, és rányomni a "VOTE" gombra.



Ennek hatására a kliens készít egy elkötelezést, azt elfedi (blind), majd aláírja. Eután elküldi az adminisztrátornak az alábbi adatokat: szavazás azonosító, szavazó (kliens) azonosító, elfedett (vak) elkötelezés a választott jelöltre, digitális aláírás az elfedett elkötelezéshez. Elment továbbá egy titkos értéket, ami az elkötelezés során jött létre.

A kapott értékek alapján az adminisztrátor megnézi, hogy az adott kliens részt vehet-e az adott szavazáson, megvizsgálja a kapott aláírást, valamint hogy szavazott-e már a kliens ezen a szavazáson a múltban. Ha minden megfelel, aláírja a fedett elkötelezést, majd visszaküldi a kliensnek az aláírást.

```
Client connected
Waiting for client to connect...
Waiting for data...
Received data from client.
Poll ID: 1012216709
Client ID: 12345678
Blinded commitment: D81+Eaf8UHjbKX903zFosfZRTSBYYaHEd3EnHY5ePh+M7kH4xTAPVQ2TTesktnIbGmqktIYvq6HYZINFEMa+8KTDokInSY0wwA
Signature of blinded commitment: AfmJcf+Gkp41MuULsTTlThzXQJcUxyxG/7TF5wp9KZ1H8SGahJAAVsSPTPK3e6vMyrqLKmYvj8hxxNHf63vkt
Client is eligible to vote.
Signature verified.
Client hasn't voted before.
Writing votes file completed successfully.
Blinded commitment signed by authority: GiEE+Yt10QYiy/0nYSyG5PkYg4bMVlv+q/5bYwZQ4SyKJLC0MNTh7TKX3+yVs1uzio2GbBbFxDzDxo
Sending to client...
Data sent
```

Adminisztrátor: szavazat kezelése

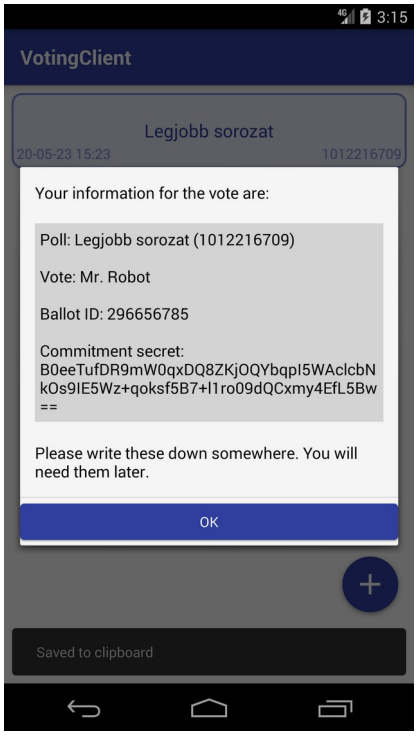
Rosszabb esetben a kliens kap egy választ, hogy valami nem stimmel. Ezt a választ a kliens egy felugró sávban látja. Ha "gyanús" a kliens (pl. nem jó az aláírása), akkor nem is küld választ az adminisztrátor, csak bontja a kapcsolatot.

```
Client connected
Waiting for client to connect...
Waiting for data...
Poll ID: 1012216709
Commitment: oQ4ngYZwGfn356EEsw7JSpA3v/YZonRURWmksH/5jhs=
Signature: UqNni5BWw/KQPzZdtaHJcrbDQWQtqxL5BY45Z8T8jKsguC6WCMh0ta0pWbJSWxWQh/XqiaQ76G5e5HNT/LffCD3jB1cwk4og0rIHByCPcYn
Connecting to authority...
Connected successfully
Fetching polls...
Request sent
Waiting for polls...
Received polls
Authority's signature on commitment valid.
Ballot saved with ID: 296656785
Sending identifier to client...
Identifier sent
Writing ballots file completed successfully.
```

Számláló: szavazat kezelése

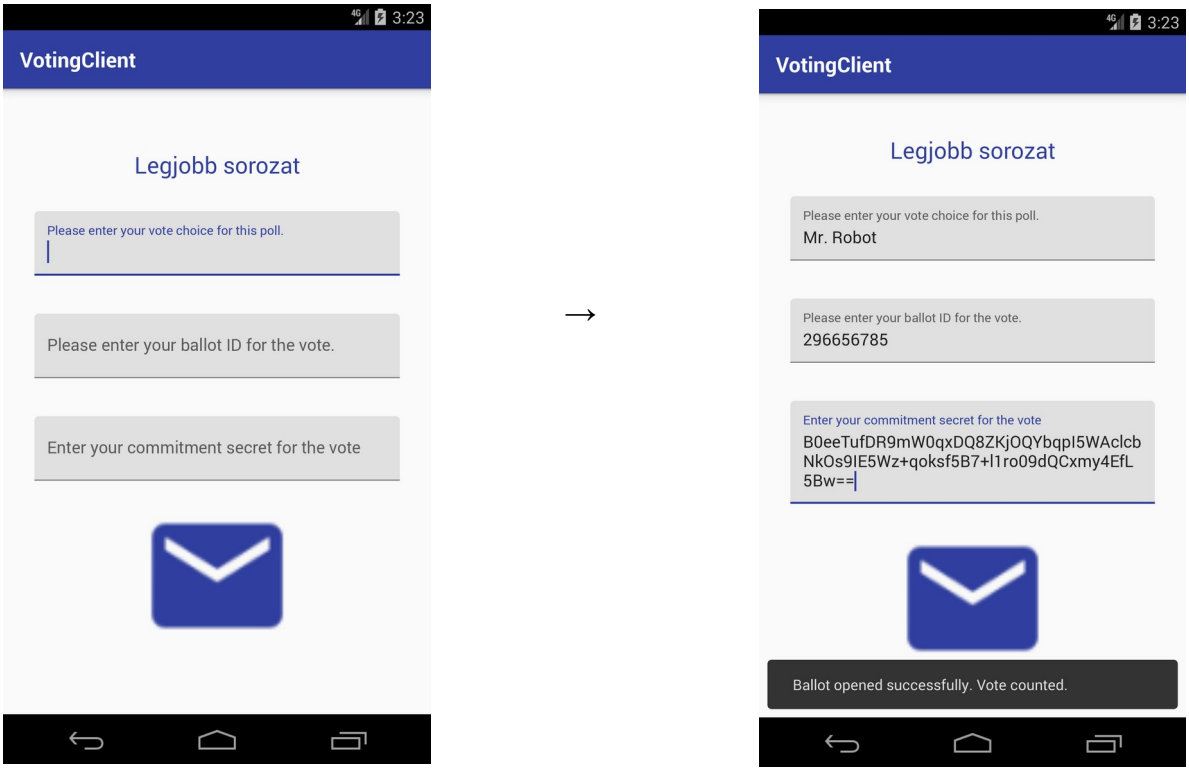
Amint a klienshez megérkezett az adminisztrátor aláírása, felfedi azt (unblind), így megkapja az eredeti elkötelezésre szóló aláírást. Ezt küldi tovább a számlálónak néhány másik adat mellett. A küldött adatok: szavazás azonosító, elkötelezés, adminisztrátor aláírása az elkötelezésre. A számláló megnézi, hogy van-e a kapott szavazásról bejegyzése. Ha nincs, lekérdezi azt az adminisztrátortól (ahogy azt a lenti példa is mutatja). Ha még nem járt le a szavazás, leellenőrzi az adminisztrátor aláírását. Ha érvényes az aláírás, akkor készít egy új cédulát, amit feljegyez magának, majd elküldi a cédula azonosítóját a kliensnek.

A kliensnél ekkor megjelenik egy üzenet a szavazat adataival, amit fel kell jegyeznie:



Megjegyzés: Ezeket az adatokat el lehetne tárolni titkosítva a kliens eszközén is, ezzel kényelmesebbé téve a folyamatot.

A szavazás lejárta után annak háttere a kliens alkalmazásban pirossá válik. Ekkor egy bizonyos időn belül a szavazóknak "ki kell nyitniuk" a számlálónál lévő elkötelezésüket. A piros hátterű szavazatra kattintva megnyílik az alábbi képernyő, amit ki kell tölteni a megfelelő adatokkal. A levél ikonra kattintva a kliens elküldi az alábbi adatokat a számlálónak: szavazás azonosító, cédula azonosító, jelölt, elkötelezéskor generált titkos érték.



A számláló néhány ellenőrzés (létezik a szavazás, létezik a cédula, nincs-e kinyitva már az elkötelezés), kinyitja az elkötelezést, és megvizsgálja, hogy az érvényes-e a kapott adatok alapján. Ha igen akkor hozzáadja a kinyitottak listájához, és növeli a számlálót a megfelelő jelöltnél:

```
Client connected
Waiting for client to connect...
Waiting for data...
Poll ID: 1012216709
Ballot ID: 296656785
Vote: Mr. Robot
Commitment secret: B0eeTufDR9mW0qxDQ8ZKjOQYbqpI5WAclcbNkOs9IE5Wz+qoksf5B7+l1ro09dQCxmy4EfL5Bw==
Writing valid votes file completed successfully.
Writing 'already opened' file completed successfully.
Vote was valid.
Sending to client...
Data sent
```

Számláló: cédula (elkötelezés) nyitás kezelése

Ha egy szavazás kinyitási ideje lejárt, a kliens eszközén az szürke háttérrel jelenik meg. Az alábbi kép egyben azt is szemléltetni szeretné, hogy a szavazások színek szerint csoportosítva jelennek meg:

A számláló minden új ciklusban, tehát minden kliens interakció után kiírj a végeredményeket a *tally.json* fájlba. Ha épp nincs interakció, akkor egy percenként teszi meg ugyanezt.

```
Socket accept timed out after 1 minute.
Connecting to authority...
Connected successfully
Fetching polls...
Request sent
Waiting for polls...
Received polls
Writing tally file completed successfully.
```

Számláló: eredmények tally fájlba írása

```
[
    {
        "poll id": 1012216709,
        "votes": [
            {
                "vote": "Mr. Robot",
                "count": 1
            }
        ]
    }
]
```

tally.json fájl tartalma

3.3. Ami kimaradt

Az alkalmazásom legnagyobb hiányossága az, hogy többfelhasználós környezetben nem állja meg a helyét. A szerver egységek ugyan külön szálban kezelik a bejövő kéréseket, de idő hiányában nem lett semmiféle szálbiztosság implementálva, ezért a közös erőforrások írása és olvasása biztos, hogy problémát okozna.

A választott FOO protokoll dokumentációja részletesen leírja annak működését, viszont az elkészítendő feladathoz nem teljesen illeszkedik, továbbá nem old meg önmagában minden problémát:

1. A protokoll egyetlen szavazásra van tervezve, annak kriptográfiai biztonsága egy szavazáson felül nincs bizonyítva, valamint ajánlás sincs arra, hogy lehetne megvalósítani egy többszavazásos rendszert.
2. Előkövetelménye a protokollnak egy már létező publikus kulcsú infrastruktúra, ahol a szavazók regisztrálva vannak az adminisztrátornál, aki magánál tartja a szavazók publikus kulcsait.
3. A protokoll feltételez egy anonim kommunikációs csatornát a szavazók és a számláló között. Enélkül a szavazók és a szavazatok összeköthetőek. Ilyenkor nem csak a szavazók magánszférája sérül, hanem a szavazatok értékesíthetővé válhatnak.

Az alkalmazásom ugyan több szavazással is működik, de az 1. pontban felvetett probléma fennáll: ennek biztonsága nincs kriptográfiailag bizonyítva.

A 2. pontban említett regisztrációs folyamat egyelőre nincs implementálva, helyette a szavazók és a hozzájuk tartozó kulcsok, valamint kliens oldalon és a számlálónál az adminisztrátor kulcsa "be vannak égetve" a kódba.

A 3. pont problémája nincs megoldva alkalmazásomban. Egy lehetséges megoldás lenne a *mix-net*ek használata [11]. Minden szavazó átadja egy megbízható "keverőnek" a rejtjelezett szavazatát, aki visszaadja a szavazatok egy permutációját, valamint egy bizonyítékot arra, hogy a keverés megtörtént. Végül a számláló dekódolja az immár kevert szavazatokat, és publikálja az eredményt. Ha nincs megbízható keverő, akkor több keverőt lehet használni úgy, hogy azokból egy pipelinet alkotunk.

4. Irodalomjegyzék

[1] R. G. Niemi and H. F. Weisberg, “Controversies in Voting Behavior,” 5th Edition, CQ Press, Washington DC, 2010, 31. oldal

[2] https://en.wikipedia.org/wiki/Electronic_voting_by_country#Germany

[3] https://www.csoonline.com/article/3269297/online-voting-is-impossible-to-secure-so-why-are-some-governments-using-it.html

[4] https://arxiv.org/pdf/1708.00991.pdf

[5] https://estoniaevoting.org/press-release/

[6] https://www.defcon.org/html/defcon-27/dc-27-index.html

[7] Bruce Schneier and Phil Sutherland. 1995. Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd. ed.). John Wiley & Sons, Inc., USA.

[8] Fujioka A., Okamoto T., Ohta K. (1993) A practical secret voting scheme for large scale elections. In: Seberry J., Zheng Y. (eds) Advances in Cryptology — AUSCRYPT '92. AUSCRYPT 1992. Lecture Notes in Computer Science, vol 718. Springer, Berlin, Heidelberg

[9] Bernhard D., Warinschi B. (2014) Cryptographic Voting — A Gentle Introduction. In: Aldini A., Lopez J., Martinelli F. (eds) Foundations of Security Analysis and Design VII. FOSAD 2013, FOSAD 2012. Lecture Notes in Computer Science, vol 8604. Springer, Cham

[10] https://en.wikipedia.org/wiki/Bouncy_Castle_(cryptography)

[11] https://crypto.stanford.edu/pbc/notes/crypto/voting.html