Univerzita Pavla Jozefa Šafárika v Košiciach

Prírodovedecká fakulta

Ústav informatiky

# ELEKTRONICKÉ VOĽBY

**Košice 2009** **Radoslav Frankovič**

# Obsah

# 1 Elektronické voľby

## 1.1 Definícia

E-voľby, alebo elektronické voľby („e-voting“), môžeme definovať ako voľby, v ktorých má volebný hlas výhradne elektronickú podobu. Teda proces, v ktorom je akt voľby občanom uskutočnený priamo prostredníctvom elektronického zariadenia a jeho výsledok odovzdaný na spracovanie prostredníctvom elektronického prenosového média (komunikačnej siete) a ďalej spracovaný výhradne elektronickou cestou. Za elektronické voľby teda nemožno považovať voľby, v ktorých vystupujú informačné technológie iba vo fáze spracovania výsledkov.

Z hľadiska vykonania je možné e-voľby rozdeliť do dvoch skupín:

1. Prvú skupinu tvoria voľby uskutočnené v presne definovanom čase prostredníctvom zákonom definovaného elektronického zariadenia umiestneného na zákonom definovanom mieste (vo volebnej miestnosti). Táto skupina riešení je spravidla označovaná ako „poll-site electronic voting“.

2. Druhú skupinu tvoria voľby uskutočnené v presne definovanom čase prostredníctvom ľubovoľného elektronického zariadenia, splňujúceho technické požiadavky kompatibility so zvoleným volebným systémom, umiestneným na ľubovoľnom mieste planéty s nutnou dostupnosťou komunikačného prostredia. Táto skupina riešení je spravidla označovaná ako „remote electronic voting“, v terminológii používanej EU ako „e-voting“ alebo slovenským ekvivalentom „ivoľby“.

V tejto práci sa budem zaoberať iba druhou z popísaných skupín elektronických volieb.

## 1.2 Elektronické voľby „na diaľku“ - Remote electronic voting

Táto skupina systému elektronických volieb představuje riešenie, ktoré prináša do systému nezávislosť na mieste, jedná sa teda o systém „vzdialených volieb“ teda

„remote voting“. Táto skutočnosť so sebou prináša najväčší problém ivolieb a to spôsob overenia identity voliča a jeho oprávnenia voliť. Hlavnou výhodou oproti systému „poll-site electronic voting“ je výrazné zvýšenie komfortu pre voliča (ako časovo, teda umožnením voľby bez ohľadu na „pracovnú dobu“ fyzických volebních miestností, tak aj tým, že občan môže voliť odkiaľkoľvek) a ďalej možnosť teoreticky neobmedzeného rastu počtu volieb pri nákladoch blízkých organizácii jedných volieb. Medzi ďalšie výhody samozrejme patrí zníženie nákladov na tlač hlasovacích lístkov, znížení náročnosti sčítania hlasov a teda aj zníženie počtu chýb a zvýšenie rýchlosti spracovánia výsledkov volieb a možné logické kontroly v rámci aplikácie, ktoré znižujú počet neplatných hlasov. Nevýhodou je určite väčšia technická a procesná náročnosť pri overovaní identity voliča, jeho oprávnenia voliť, zaistenie anonymity hlasu a s tým súvisiace bezpečnostné riziká. Ďalším problémovým okruhom je veľký rozdiel oproti klasickému spôsobu volieb, čo môže viesť k nedôvere voličov voči novému systému volieb a teda k ich neúčasti na voľbách.

## 1.3 Typy elektronických volieb

- Áno/Nie voľby
- 1 z L - volič má L možností, z ktorých si vyberie jednu voľbu
- K z L - volič vyberie K z L prvkov
- K z L - usporiadane - volič vyberie usporiadanú K-ticu z L prvkov
- 1-L-K - volič vyberie jednu z L množín a z tejto množiny vyberie K prvkov
- Write in - volič formuluje vlastnú odpoveď, takže voľba je reťazec určitej max. dĺžky

## 1.4 Bezpečnostné požiadavky na elektronické voľby

1. Oprávnenosť (angl. eligibility) - len registrovaní voliči môžu voliť, voľba musí byť jedinečná

2. Súkromie (angl. privacy) - nie je možné vytvoriť spojenie medzi jednotlivou voľbou a voličom
3. Overiteľnosť - možnosť overiť, či hlas bol zaznamenaný a započítaný do výsledku volieb
    - individuálna - samotný volič vie overiť svoj hlas
    - univerzálna - ktokoľvek môže overiť, či hlasy boli správne spočítané
4. Nespochybniteľnosť(angl. dispute-freenes) - schéma by mala poskytovať mechanizmy na vyriešenie nezrovnalostí v každej fáze
5. Správnosť (angl. accuracy) - hlasy musia byť správne zaznamenané a započítané
6. Spravodlivosť - nikto by nemal byť schopný vypočítať čiastočné výsledky, pokiaľ prebiehajú voľby
7. Robustnosť - schéma je maximálne robustná, ak je potrebná spolupráca všetkých autorít na volebný podvod resp. chybu
8. Bezdokladovosť (angl. receipt-freenes) - volič nie je schopný poskytnúť dôkaz o svojej voľbe niekomu inému
9. Nedonutiteľnosť (angl. incoercibility) - útočník by nemal byť schopný prinútiť voliča k určitej voľbe
10. Škálovatelnosť
11. Praktickosť - schéma by mala byť realizovateľná

# 2 Protokol elektronických volieb

Protokol elektronických volieb popísaný v tejto kapitole je navrhnutý v článku[1].

Značenie pre účastníka $X$:
$Pk_X^E(Pk_X^S)$ - verejný kľúč pre šifrovanie (podpisovanie)
$Sk_X^E(Sk_X^S)$ - súkromný kľúč pre šifrovanie (podpisovanie)
Šifrovanie správy $m$ verejným kľúčom $Pk_X^E$ budeme označovať ako $Pk_X^E(m)$. Podpisovanie správy $m$ účastníkom $X$ použitím súkromného kľúča $Sk_X^S$ budeme označovať ako $Sk_X^S(m)$.

## 2.1 Schéma založená na slepých podpisoch

Pre zabezpečenie vlastnosti *súkromia* elektronických volieb tzn. aby nebolo možné určiť spojenie medzi voličom a voľbou použijeme schému založenú na slepých podpisoch. Slepý podpis rieši problém, keď žiadateľ o podpis chce získať podpis správy $m$ bez toho aby podpisujúci vedel, alebo rozumel, tomu čo podpisuje. Správa $m$ pre podpisujúceho $A$ s verejným podpisovacím kľúčom $Pk_X^S$ je „zaslepená“ žiadateľom použitím blinding funkcie $Bl(m, r, Pk_A^S)$ s náhodným parametrom $r$. Podpisujúci podpíše „zaslepenú“ správu použitím svojho súkromného podpisovacieho kľúča $S_{Sk_A^S}(Bl(m, r, Pk_A^S))$ a vráti správu späť žiadateľovi o podpis. Žiadateľ získa požadovaný podpis použitím „odslepujúcej“ funkcie

$$Unbl(S_{Sk_A^S}(Bl(m, r, Pk_A^S)), r, Pk_A^S) = S_{Sk_A^S}(m)$$

.

### Registračná fáza

Volič $V_i$ získa verejné kľúče všetkých servrov a okrem nich aj verejný parameter $g_T^t$ počítacieho servra jedinečný pre každé voľby. Súkromný parameter volieb $t$ počítacieho servra je zdieľaný viacerými autoritami napr. volebnou komisiou. $g_T \in G$ je genrátor cyklickej grupy $G$, na ktorú je možné namapovať množinu asymetrických kľúčov volieb.

Volič $V_i$ vyberie svoju voľbu $vote_i$. Potom si zvolí náhodný parameter $v_i$ a vypočíta asymetrický kľúč $K_i = (g_T^t)^{v_i}$ pre dešifrovanie a odpovedajúci asymetrický kľúč $K_i^{-1}$ na zašifrovanie svojej voľby.

Volič $V_i$ si pripraví svoj balíček s hlasom $b_i = E_{K_i^{-1}}(vote_i), g_T^{v_i}$ a vypočíta jeho hash $h_{i_1} = H(b_i)$ použitím hashovacej funkcie $H$. Blinds tento hash ako $bl_{i_1} = Bl_{Pk_{RS_1}^S}(h_{i_1}, r_{i_1})$ pomocou náhodného parametra $r_{i_1}$. Volič sa zaregistruje u registračného servra $RS_1$ a nechá si ním podpísať správu s hlasom nasledujúcou výmenou správ:

1. $V_i \rightarrow RS_1 : E_{Pk_{RS_1}^E}(V_i, S_{Sk_{V_i}^S}(ID_{voting}, V_i, bl_{i_1}))$
2. $RS_1 \rightarrow V_i : S_{Sk_{RS_1}^S}(bl_{i_1})$

Po obdržaní a dešifrovaní prvej správy registračný server $RS_1$ skontroluje či volič $V_i$ je na zozname voličov pre voľby s číslom $ID_{voting}$ a overí jeho podpis správy. Ak je podpis správy platný, potom registračný server $RS_1$ podpíše blinded správu $bl_{i_1}$ a výsledok pošle ako správu späť voličovi. Volič unblinds túto správu použitím náhodného parametra $r_{i_1}$ a získa tak podpis hash-hodnoty jeho balíčka s voľbou: $S_{Sk_{RS_1}^S}(h_{i_1})$.

V druhej časti registračnej fázy sa musí volič $V_i$ zaregistrovať u registračného servra $RS_2$, aby získal *„token“*, ktorý použije vo volebnej fáze. Najprv volič vytvorí správu $m_{i_2} = E_{Pk_{RS_1}^E}(E_{Pk_T^E S}(b_i, S_{Sk_{RS_1}^S}(h_{i_1})))$ a vypočíta hash-hodnotu tejto správy ako $h_{i_2} = H(m_{i_2})$. Potom blinds tento hash pomocou náhodného parametra $r_{i_2}$ a získa $bl_{i_2} = Bl_{Pk_{RS_2}^S}(h_{i_2}, r_{i_2})$. Volič sa zaregistruje u registračného servra $RS_2$ nasledujúcou výmenou správ:

1. $V_i \rightarrow RS_2 : E_{Pk_{RS_2}^E}(V_i, S_{Sk_{V_i}^S}(ID_{voting}, bl_{i_2}, V_i))$
2. $RS_2 \rightarrow V_i : S_{Sk_{RS_2}^S}(bl_{i_2})$

Po obdržaní a dešifrovaní prvej správy registračný server $RS_2$ skontroluje či volič $V_i$ je na zozname voličov pre voľby s číslom $ID_{voting}$ a overí jeho podpis správy. Ak je podpis správy platný, potom registračný server $RS_2$ podpíše blinded správu $bl_{i_2}$ a výsledok pošle ako správu späť voličovi. Volič unblinds túto správu použitím náhodného parametra $r_{i_2}$ a získa tak $S_{Sk_{RS_2}^S}(h_{i_2})$, čo bude *„tokenom“*, ktorý volič $V_i$ použije vo volebnej fáze volieb.

**Volebná fáza**

Po popísanej registrácii sa volič $V_i$ zúčastní na samotných voľbách tak, že v čase do ukončenia volieb pošle nasledujúcu správu:

1. $V_i \rightarrow RS_1 : E_{Pk^E_{RS_1}}(m_{i_2}, S_{Sk^S_{RS_2}}(h_{i_2}))$

Registračný server $RS_1$ dešifruje obdržanú správu, overí podpis hash-hodnoty správy $m_{i_2}$ servra $RS_2$ a uloží správu $m_{i_2}$ spolu s jej podpísanou hash-hodnotou $S_{Sk^S_{RS_2}}(h_{i_2})$ do svojej lokálnej databázy.

**Sčítacia fáza**

Po ukončení volieb pošle registračný server $RS_1$ *lexikograficky usporiadané* správy $m_{i_2}$ všetkých zúčastnených voličov $V_i$ spolu s odpovedajúcimi podpismi $S_{Sk^S_{RS_2}}(h_{i_2})$ správ registračnému servru $RS_2$. Registračný server $RS_1$ taktiež zverejní zoznam všetkých podpisov správ. Registračný server $RS_2$ skontroluje, či je jeho podpis každej správy platný. Ak áno, tak $RS_2$ dešifruje všetky správy $m_{i_2}$ a pošle ich *lexikograficky usporiadané* počítaciemu servru $TS$ spolu s podpisom hash-hodnoty celého zoznamu posielaných správ. Registračný server $RS_2$ taktiež zverejní zoznam všetkých podpisov správ, ktoré posielal počítaciemu servru $TS$.

1. $RS_1 \rightarrow RS_2 : m_{i_2}, S_{Sk^S_{RS_2}}(h_{i_2})$
2. $RS_2 \rightarrow TS : E_{Pk^E_T S}(b_i, S_{Sk^S_{RS_1}}(h_{i_1}))$

Počítací server $TS$ skontroluje podpis servra $RS_2$ celého zoznamu obdržaných správ, dešifruje všetky správy a skontroluje podpis servra $RS_1$ každého balíčka $b_i$. Počítací server $TS$ obdrží zdieľaný súkromný parameter $t$ volieb a pre každý balíček s voľbou $b_i$ vypočíta kľúč $K_i = (g_T^{v_i})^t$ potrebný na dešifrovanie voľby voliča $V_i$. Po dešifrovaní všetkých volieb počítací server $TS$ zverejní $t$ a zoznam $S_{Sk^S_{RS_1}}(h_{i_1})$, $(E_{K_i^{-1}}(vote_i), g_T^{v_i})$, $vote_i$.

## 2.2 Bezdokladová (receipt-free) verzia schémy

Schéma sa bude líšiť v tom, že namiesto $b_i$ definovaného v základnej schéme zavedieme označenie $b_i^{RF}$ balíčka s hlasom, v ktorom budeme posielať bit commitment voľby voliča a popierateľným šifrovaním budeme šifrovať náhodný parameter

potrebný na otvorenie bit commitmentu. Takto vie sčítací server otvoriť bit commitment a volič vie otvoriť bit commitment ako ľubovoľnú zvolenú voľbu použitím príslušného vypočítaného parametra. Na zaistenie ešte väčšej robustnosti schémy v požiadavke spravodlivosti nám stačí šifrovať popierateľným šifrovaním kľúč $K_i$ definovaný v základnej schéme.

### Trapdoor bit commitment

Využijeme parametre $p, q, g$ vygenerované a zverejnené systémom, kde $p$ a $q$ sú prvočísla, $q|p-1, g \in \mathbb{Z}_p^*$ a $q$ je rádu $g$. Volič $V_i$ pomocou súkromného parametra $\alpha_i$ vypočíta $G_i = g^{\alpha_i} \mod p$. Definujeme bit commitment $BC(vote_i, r_i) = g^{vote_i}.G_i^{r_i} \mod p$, kde $vote_i$ je voľba voliča $V_i$ a $r_i$ je náhodný parameter. Volič dokáže otvoriť bit commitment ako hociktorú hodnotu $vote^c$ použitím $\alpha_i$ a z rovnosti $vote_i + \alpha_i r_i = vote^c + \alpha_i r_i^c \mod p$ vie vypočítať $r_i^c$, tak aby platilo $BC(vote_i, r_i) = BC(vote^c, r_i^c)$.

Predpokladáme, že počet kandidátov nie je príliš veľký a že sčítací server dokáže pre každého kandidáta $c$ vypočítať a uložiť hodnotu $g^{vote^c}$. Na otvorenie bit commitmentu bude postačovať ak volič pošle $BC(vote_i, r_i), r_i, G_i$. Sčítací server vypočíta $G_i^{r_i} = g^{\alpha.r_i} \mod p$ a následne inverznú hodnotu $(G_i^{r_i})^{-1} \mod p$. Hodnotu $g^{vote_i}$ vypočíta ako $g^{vote_i} = BC(vote_i, r_i).(G_i^{r_i})^{-1} = g^{vote_i}.G_i^{r_i}.(G_i^{r_i})^{-1} \mod p$ a na získanie hodnoty $vote_i$ porovná $g^{vote_i}$ s predvypočítanými hodnotami $g^{vote^c}$ pre každého kandidáta $c$.

### Popierateľné šifrovanie

Popierateľné šifrovanie má spĺňať nasledovné požiadavky:

- iba príjemca pozná dešifrovací kľúč a schéma je sémanticky bezpečná
- správa získaná dešifrovaním správy príjemcom neobsahuje žiadne zmenené bity
- odosielateľ musí poznať efektívny zavádzací algoritmus $\phi$, taký že pre ľubovoľný daný šifrovaný text $s$, ktorý je zašifrovaním správy $m$ pomocou náhodného parametra $l (s = DE(m, l))$ a pre danú zavádzajúcu správu $m_f$ vie vypočítať $l' = \phi(s, m, l, m_f)$ také, že $s = DE(m_f, l')$.

**Bezdokladová (receipt-free) verzia protokolu**

Volič $V_i$ si podobne ako v základnej schéme zvolí náhodné $v_i$ a vypočíta asymetrický kľúč $K_i = (g_T^t)^{v_i}$ pre dešifrovanie a odpovedajúci kľúč $K_i^{-1}$ na za3ifrovanie bit commitmentu. Pripraví si hlas $b_i^{RF} = (E_{K_i^{-1}}(BC(vote_i, r_i)), g_T^{v_i}, G_i, DE_{Pk_{TS}^{DE}}(r_i, l_i))$. Pre každého kandidáta $c$ vypočíta $r_i^c$ použitím svojho súkromného parametra $\alpha_i$ a rovnosti $vote_i + \alpha_i r_i = vote^c + \alpha_i r_i^c \mod p$. Takto vie volič otvoriť bit commitment $BC(vote_i, r_i)$ ako voľbu hociktorého kandidáta $c$ vypočítaním odpovedajúcej hodnoty $r_i^c$. Pre každý parameter $r_i^c$ vie volič pomocou zavádzajúceho algoritmu $\phi$ zo schémy popierateľného šifrovania vypočítať hodnotu $l_i^c = \phi(DE_{Pk_{TS}^{DE}}(r_i, l_i), r_i, l_i, r_i^c)$. Pri prinútení voliča iným voličom aby hlasoval tak ako chcel on, vie volič ukázať vhodné $l_i^c$ také, že $DE_{Pk_{TS}^{DE}}(r_i^c, l_i^c) = DE_{Pk_{TS}^{DE}}(r_i, l_i)$ a $BC(vote_i, r_i) = BC(vote_i^c, r_i^c)$ pre každého kandidáta $c$.

V sčítacej fáze sčítací server overí podpis $RS1$ hash-hodnoty hlasu $b_i^{RF}$. Získa parameter $t$, pre každý hlas vypočíta kľúč $K_i = (g_T^{v_i})^t$ na dešifrovanie $E_{K_i^{-1}}(BC(vote_i, r_i))$ a získa tak bit commitment $BC(vote_i, r_i)$. Potom dešifruje parameter $r_i$ šifrovaný popierateľným šifrovaním a otvorí bit commitment ako je to popísané vyššie.

Je potrebné zabezpečiť aby volič alebo hocikto iný mohol overiť korektnosť sčítania hlasov vo voľbách. Použijeme na to *zero-knowledge proof*. Sčítací server zverejní zoznam bit commitmentov $bc_i = (BC(vote_i, r_i))$ s odpovedajúcimi podpismi registračného servra $RS1$. Ďalej zverejní zoznam volieb $vote_i'$ v náhodnom poradí použitím *náhodnej permutácie* $\pi$ takej, že $vote_i' = vote_{\pi(i)}$. Presnejšie, sčítací server rozdelí rozdelí všetky voľby do disjunktných skupín tak aby každá skupina obsahovala aspoň jedného kandidáta, ak je to možné a pre každú skupinu zverejní zoznam bit commitmentov $bc_1, \ldots, bc_k$ a zoznam volieb $vote_1', \ldots, vote_k'$. Pouitím neinteraktívnej verzie zero-knowledge proof overi, že pozná permutáciu $\pi$ a náhodné parametre $r_i$ na otvorenie bit commitmentov také, že $bc_i = (BC(vote_i, r_i))$ a $vote_i' = vote_{\pi(i)}$ bez odhalenia $\pi$ a $r_i$.

Známe schémy popierateľného šifrovania však správy príliš zväčšujú. Pre naše potreby potrebujeme šifrovať parameter $r_i$ potrebný na otvorenie bit commitmentu. Môžme použiť jednoduchý trik, v ktorom vygenerujeme náhodné kľúče malej dĺžky $SK_i^c$ na symetrické zašifrovanie parametrov $r_i^c$ pre každého kandidáta $c$. Použijeme popierateľné šifrovanie len na „pravý“ kľúč $SK_i$ použitý na šifrovanie

parametra $r_i$ na otvorenie voľby $vote_i$. Sčítací server dešifruje symetrický kľúč $SK_i$ a pokúsi sa dešifrovať každý parameter. Server vie (na základe určitej redundancie) rozoznať „pravý“ parameter $r_i$ a korektne otvoriť bit commitment ako voľbu $vote_i$ voliča. Na druhej strane dokáže volič ukázať vhodný kľúč $SK_i^c$ na dešifrovanie $E_{SK_i^c}(r_c)$ a otvoriť bit commitment ako pre kandidáta $c$.

# 3 Netradičné kryptografické primitíva pre protokol elektronických volieb

## 3.1 Slepý podpis

Slepý podpis je typom digitálneho podpisu, pri ktorom podpisujúci nepozná obsah podpisovanej správy. Obsah správy je totiž žiadateľom podpisu pred samotným podpísaním „zaslepený“ (z angl. blinded). Podpis musí byť nefalšovateľný(iba podpisujúci môže podpísať správu) a verejne overiteľný(hocikto môže overiť, že daný podpis správy je korektný).

Napríklad ak podpisujúci vlastní verejný RSA kľúč $(n, e)$ a odpovedajúci súkromný RSA kľúč $d$, tak môže podpísať správu $m$, $m \in Z_n$ ako $s = m^d \mod n$. Takto môže hocikto overiť či $m \stackrel{?}{=} s^e \mod n$ a keďže súkromný kľúč vlastní a pozná iba podpisujúci, tak podpis je nefalšovateľný.

Ak žiadateľ o podpis nechce, aby podpisujúci pri podpisovaní vedel čo podpisuje, použije nasledovný postup, aby získal „slepý podpis“ správy:

1. Žiadateľ o podpis najprv zaslepí správu $m$ ako $m' = mr^e \mod n$ pomocou náhodného parametra $r, r \in Z_n$ a pošle správu $m'$ podpisujúcemu.

2. Podpisujúci podpíše zaslepenú správu $m'$ a pošle podpis $s = m'^d \mod n$ späť žiadateľovi.

3. Žiadateľ získa požadovaný podpis $s$ správy $m$ nasledovne:

$$s = \frac{s'}{r} = \frac{m'^d}{r} = \frac{m^d r^{ed}}{r} = \frac{m^d r}{r} = m^d \mod n$$

### Implementácia v jazyku JAVA

Trieda BlindSignature má 2 konštruktory:
*BlindSignature(RSAPublicKey)* - tento konštruktor vytvára inštanciu triedy slepý podpis pre žiadateľa o podpis alebo overovateľ podpisu. Parametrom je podpisovateľov verejný kľúč. Konštruktor vygeneruje náhodné číslo a vypočíta inverzné číslo k vygenerovanému číslu modulo modulo verejného kľúča podpisovateľa.

*BlindSignature(RSAPrivateKey)* - tento konštruktor vytvára inštanciu triedy slepý podpis pre podpisovateľa. Parametrom je súkromný kľúč podpisovateľa.

*byte[] BlindSignature.blind(byte[])* - metóda na zaslepenie správy danej parametrom ako pole byte-ov. Vracia zaslepenú správu ako pole byte-ov. Metódu volá žiadateľ o podpis.

*byte[] BlindSignature.sign(byte[])* - metóda na podpísanie zaslepenej správy danej parametrom ako pole byte-ov. Vracia podpísanú zaslepenú správu ako pole byte-ov. Metódu volá podpisovateľ.

*byte[] BlindSignature.unblind(byte[])* - metóda na odslepenie podpísanej zaslepenej správy danej parametrom ako pole byte-ov. Vracia podpísanú správu ako pole byte-ov. Metódu volá žiadateľ o podpis.

*boolean BlindSignature.verify(byte[], byte[])* - metóda na overenie podpisu. Vracia *true*, ak pole byte-ov dané prvým parametrom je podpisom správy danej druhým parametrom ako pole byte-ov.

## 3.2 Optimal asymmetric encryption padding (OAEP)

Optimal asymmetric encryption padding (OAEP) je padding schéma často používaná s RSA šifrovaním. OAEP algoritmus, ktorý je formou Feistelovej schémy, používa náhodné orakulá G a H na úpravu plaintextu pred asymetrickým šifrovaním. V tejto schéme sa pri paddingu pridáva náhodný parameter, čím sa deterministické šifrovanie (napr. RSA) mení na nedeterministické.
*Popis schémy:*

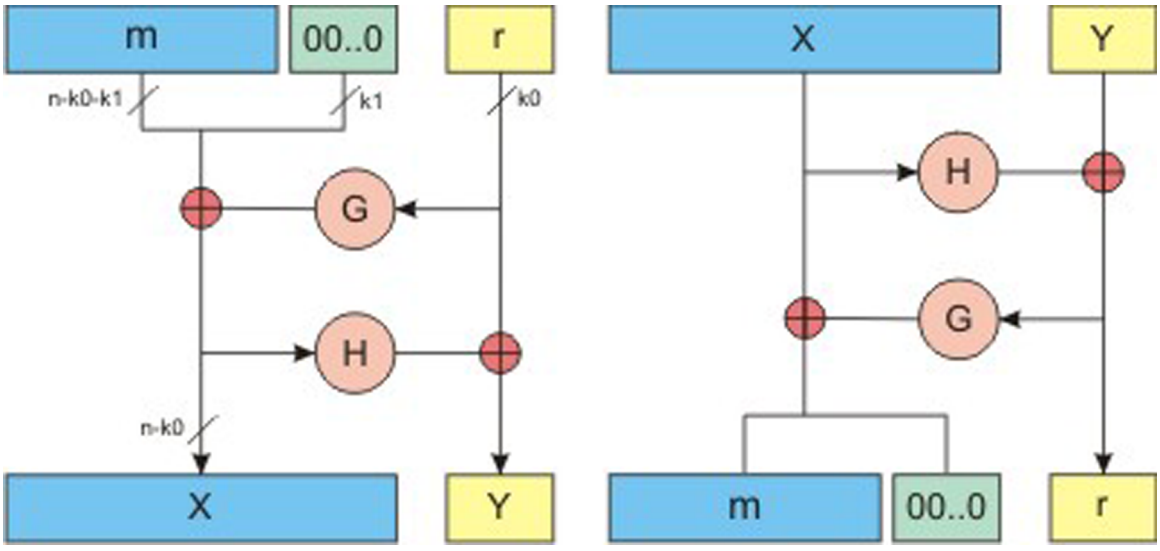
- $n$ je veľkosť (počet bitov) modula RSA šifrovania
- $k_0$ a $k_1$ sú kladné čísla pevne určené protokolom
- $m$ je plaintext, dĺžky $(n - k_0 - k_1)$ bitov
- $G$ a $H$ sú hashovacie funkcie určené protokolom

*Kódovanie:*

1. správa $m$ je doplnená $k_1$ nulovými bitmi, vznikne tak reťazec dĺžky $n - k_0$ bitov
2. vygenerujeme náhodný reťazec r dĺžky $k_0$ bitov
3. hashovacia funkcia $G$ rozšíri r dĺžky $k_0$ na reťazec dĺžky $n - k_0$
4. $X = m00..0 \oplus G(r)$
5. $H$ zredukuje $X$ dĺžky $n - k_0$ bitov na reťazec dĺžky $k_0$ bitov
6. $Y = r \oplus H(X)$
7. výstupom je reťazec $X||Y$ dĺžky $n$ bitov

*Dekódovanie:*

1. vypočítame $r = Y \oplus H(X)$
2. vypočítame $m00..0 = X \oplus G(r)$




Obr. 1: OAEP - kódovanie a dekódovanie

**Implementácia v jazyku JAVA**

Trieda OAEPPaddingUtils obsahuje dve metódy:
*byte[] OAEPPaddingUtils.pad(byte[])* - metóda na kódovanie (doplnenie) správy zadanej ako pole bytov ako argument metódy algoritmom OAEP pre nedeterministické šifrovanie. Vracia pole byte-ov - kódovanú správu.

*byte[] OAEPPaddingUtils.unpad(byte[])* - metóda na dekódovanie správy zadanej ako pole bytov ako argument metódy algoritmom OAEP pre nedeterministické šifrovanie. Vracia pole byte-ov - dekódovanú správu.

## 3.3 RSA-OAEP

Šifrovanie systémom RSA-OAEP prebieha tak, že správa pripravená na šifrovanie sa najprv zakóduje algoritmom OAEP. Vznikne takto správa dĺžky modula RSA-kľúča. Následne sa táto zakódovaná správa zašifruje algoritmom RSA.

Trieda RSAOAEPCipher obsahuje tieto metódy:
*byte[][] splitByteArray(byte[], int)* - metóda rozdelí pole byte-ov zadané ako argument metódy na polia byte-ov dĺžky zadanej ako druhý argument metódy, alebo vráti to isté pole byte-ov, ak je kratšie ako dĺžka zadaná druhým argumentom

*byte[] mergeByteArrays(byte[][])* - spojí polia byte-ov zadané ako argument metódy jedno pole byte-ov

*byte[] encryptBlock(byte[], RSAPublicKey)* - metóda najprv doplní jeden blok správy zadaný ako argument metódy algoritmom OAEP a potom ho zašifruje algoritmom RSA pre asymetrické šifrovanie verejným kľúčom zadaným ako druhý argument metódy

*byte[] encrypt(byte[], RSAPublicKey)* - metóda najprv rozdelí správu zadanú ako pole byte-ov v prvom argumente metódy na viacero blokov, každý z nich potom zašifruje metódou *encryptBlock()* verejným kľúčom zadaným v druhom ar-

gumente metódy a následne tieto zašifrované bloky spojí do jedného poľa byte-ov, ktoré vráti ako návratovú hodnotu metódy

*byte[] decryptBlock(byte[], RSAPrivateKey)* - metóda najprv dešifruje jeden blok správy zadaný ako argument metódy algoritmom RSA pre asymetrické šifrovanie súkromným kľúčom zadaným v druhom argumente metódy a tento blok následne odkóduje algoritmom OAEP

*byte[] encrypt(byte[], RSAPrivatecKey)* - metóda najprv rozdelí správu zadanú ako pole byte-ov v prvom argumente metódy na viacero blokov, každý z nich potom dešifruje metódou *decryptBlock()* súkromným kľúčom zadaným v druhom argumente metódy a následne tieto dešifrované bloky spojí do jedného poľa byte-ov, ktoré vráti ako návratovú hodnotu metódy

## 3.4 Bit-Commitment

Ak odosielateľ nechce aby prijímateľ hneď zistil aký bit poslal. Prijímateľ však požaduje aby odosielateľ nemohol neskôr, ešte však pred odhalením posielaného bitu, zmenil posielaný bit. Odosielateľ nejako zašifruje bit $b$ a pošle ho prijímateľovi. Prijímateľ nevie dešifrovať $b$ pokiaľ mu odosielateľ nepošle kľúč. Vo všeobecnosti je *bit commitment* funkciou $\xi : \{0,1\} \times X \rightarrow Y$. Zašifrovaním bitu $b$ je hodnota $\xi(b,k), k \in X$. Schéma bit commitment spĺňa nasledovné požiadavky:

- prijímateľ nevie získať $b$ z $\xi(b,k)$
- odosielateľ vie otvoriť bit commitment použitím $b$ a $k$. Odosielateľ nevie otvoriť bbit commitment ako 0 aj ako 1.

Ak odosielateľ posiela reťazec bitov tak spraví bit commitment každého bitu nezávisle. Bit commitment, ktorý vie odosielateľ otvoriť ako 0 aj ako 1 sa nazýva *trapdoor bit commitment.*

Bit commitment sa dá realizovať nasledovne:
Majme veľké prvočíslo $p$, generátor $g$ cyklickej grupy $\mathbb{Z}_p$ a $G = g^a, G \in \mathbb{Z}_p$. Odosielateľ ani prijímateľ nepoznajú $a$. $G$ môže byť zvolené náhodne. Potom

$\xi : \{0,1\} \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ je hodnota $\xi(b,k) = g^k G^b$. Bit commitment môže byť otvorený ako $b$ odhalením $k$, alebo môže byť otvorený ako $\neg b$ odhalením $k - a$ ak $b = 0$ alebo $k + a$ ak $b = 1$. Ak odosielateľ nepozná $a$ tak nevie otvoriť bit commitment ako $\neg b$. Podobne, ak prijímateľ nepozná $b$, tak nevie otvoriť bit commitment zo znalosti $\xi(b,k) = g^k G^b$. Ak odosielateľ pozná $a$, tak môže vytvoriť trapdoor bit commitment. Ak prijímateľ pozná $a$, tak vie oklamať tretiu stranu o tom aký bit prijal, tým že bude tvrdiť, že obdržal $k - a$ alebo $k + a$. V tom prípade hovoríme o *chameleónskom bit commitmente*.

Namiesto vytvárania bit commitmentu pre každý bit bitového reťazca $s$ nezávisle, odosielateľ môže vytvoriť $\xi(s,k) = g^k G^s$, kde $0 \leq s < p$. Ak odosielateľ pozná $a$, tak vie otvoriť $\xi(s,k)$ ako ľubovoľné $s'$ tak, že vypočíta príslušné $k'$ pomocou rovnosti $as + k = as' + k'$.

## 3.5 Popierateľné šifrovanie založené na kvadratických zvyškoch

Odosielateľ správy zašifruje správu verejným kľúčom prijímateľa. Správa je šifrovaná po bitoch. Prijímateľ má vlastné RSA modulo $n = pq$, kde $p$ a $q$ sú veľké prvočísla. $n$ je jeho verejný kľúč a $p$ a $q$ sú tajné. Odosielateľ nepotrebuje súkromný ani verejný kľúč.

### Šifrovanie

Odosielateľ zašifruje a pošle jeden bit správy nasledovne($k$ je bezpečnostný parameter):

- Ak odosielateľ chce poslať 1(jednotku), pošle $k$ náhodných kvadratický zvyškov: náhodne zvolí $x_i \in \mathbb{Z}_n^*, i = 1, \ldots, k$ a pošle $y_i = x_i^2$ prijímateľovi.

- Ak odosielateľ chce poslať 0(nulu), pošle $k$ náhodných celý čísel $y_i \in J_n, i = 1, \ldots, k$: zvolí si $y_i \in \mathbb{Z}_n^*$ a overí, či $(\frac{y_i}{n}) = 1$.

### Dešifrovanie

Prijímateľ dešifruje správu nasledovne:

- Ak všetky prijaté $y_i$ sú kvadratické zvyšky, tak správu dešifruje ako 1(jednotku).

- Ak aspoň jedno prijaté $y_i$ nie je kvadratickým zvyškom, tak správu dešifruje ako 0(nulu).

Keďže iba prijímateľ vie rozoznať kvadratický zvyšok, tak len on vie správu dešifrovať.

Ak odosielateľ pošle 1, potom všetky $y_i$ sú kvadratickými zvyškami a prijímateľ dešifruje správu korektne. Ak odosielateľ pošle 0, tak je len malá pravdepodobnosť, že všetky náhodne vybraté $y_i$ budú kvadratickými zvyškami. Takto by prijímateľ nesprávne dešifroval správu ako 1. Pravdepodobnosť, že všetky $y_i$ budú kvadratickými zvyškami je $(1/2)^k$.

Odosielateľ interpretuje šifrovaný text útočníkovi nasledovne:

- Ak poslaný bit bol 1, tak ho dešifruje ako 1 odhalením kvadratických zvyškov, alebo ho dešifruje ako 0 tvrdením, že $y_i$ boli vybraté náhodne.

- Ak poslaný bit bol 0, tak tento bit vie dešifrovať len ako 0.

Keďže útočník nevie odlíšiť kvadratické zvyšky, tak musí veriť tvrdeniu, že odoslané $y_i$ boli zvolené náhodne.

Odosielateľ nevie zavádzať, ak odoslaný bit bol 0. To sa dá vyriešiť rozšírením šifrovacej schémy nasledovne:

1. Odosielateľ zvolí náhodný reťazec bitov $s$ dĺžky $l$:

    - Ak chce odosielateľ poslať 0, tak zvolí $s$ s párnym počtom 1.
    - Ak chce odosielateľ poslať 1, tak zvolí $s$ s nepárnym počtom 1.

2. Odosielateľ vytvorí a pošle $s$ pre každý bit správy.

Takto narastie veľkosť posielaných dát z $k$ na $lk$. Ak prijímateľ príjme $s$ s nepárnym počtom 1, tak dekóduje tento bit ako 1. Inak ho dekóduje ako 0.

Pri útoku tak stačí odosielateľovi zmeniť jediný posielaný bit z reťazca z 1 na 0 a tým reťazec bitov dekóduje ako ako opačný bit oproti správnemu bitu. Tento spôsob sa nedá uskutočniť ibe v tom prípade, ak $s$ pozostáva zo samých 0.

## 3.6 Dôkaz znalosti náhodnej permutácie $\pi$

Budeme označovať overovateľa ako $T$.

T vytvorí disjunktné množiny zo všetkých hlasov také, že každá z množín obsahuje všetky možné hlasy(kandidátov) (v našom prípade o veľkosti 5, napr. $v_1, v_2, v_3, v_4, v_5$). T vytvorí náhodnú permutáciu hlasov $\pi$ a zverejní zoznamy $(m_1, m_2, m_3, m_4, m_5)$ a $(v'_1, v'_2, v'_3, v'_4, v'_5) = (v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)}, v_{\pi(4)}, v_{\pi(5)})$. $T$ následne spraví neinteraktívny dôkaz $\sigma$, že $(v'_1, v'_2, v'_3, v'_4, v'_5) = (v_1, v_2, v_3, v_4, v_5)$. **Vstup:** $(m_1, m_2, m_3, m_4, m_5)a(v'_1, v'_2, v'_3, v'_4, v'_5)$

Potrebujeme overiť, že T pozná $\pi$ a $r_i$ také, že $m_i = BC(v_i, r_i)$ a $v'_i = v_{\pi(i)}$.

Nasledovný postup opakujeme $k$ krát:

1. $T$ vygenerujenáhodnú permutáciu $\delta$ a $u_i, s_i, t_i \in \mathbb{Z}_p$.

   $$Z_i = m_i G_i^{s_i} h^{u_i} \mod p$$

   $$W_i = g^{v'_{\pi(i)}} h^{t_i} \mod p$$

   kde $i = 1, \ldots, 5$.

   $T$ pošle $X = (Z_1, \ldots, Z_5, W_1, \ldots, W_5)$ overovateľovi dôkazu.

2. Overovateľ náhodne zvolí bit $e \in \{0, 1\}$ a pošle ho $T$.

3. Ak $e = 0$, tak T pošle $Y = \delta, u_i, s_i, t_i$ overovateľovi.

   Ak $e = 1$, tak T vypočíta:

   $$\rho = \pi^{-1} \circ \delta^{-1}$$

   $$x_i = r_i + s_i \mod p$$

   $$y_i = u_i - t_{\rho(i)} \mod p$$

   a pošle $Y = (\rho, x_i, y_i)$ overovateľovi.

4. Ak $e = 0$, tak overovateľ skontroluje, či platí nasledovné:

   $$Z_i = m_i G_i^{s_i} h^{u_i} \mod p$$

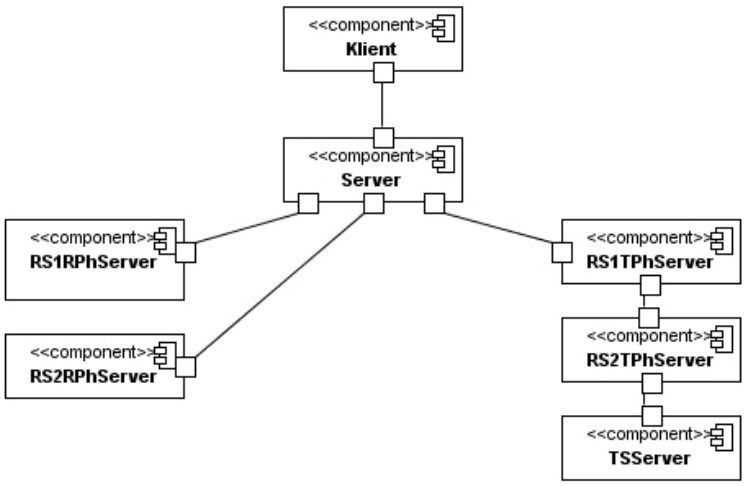
   $$W_i = g^{v'_{\pi(i)}} h^{t_i} \mod p$$

   Ak $e = 1$, tak overovateľ skontroluje, či platí nasledovné:

   $$W_{\rho(i)} G_i^{x_i} h^{y_i} \equiv Z_i( \mod p)$$

# 4 Návrh a implementácia volebného systému

## 4.1 Komponenty systému




Obr. 2: Diagram komponentov

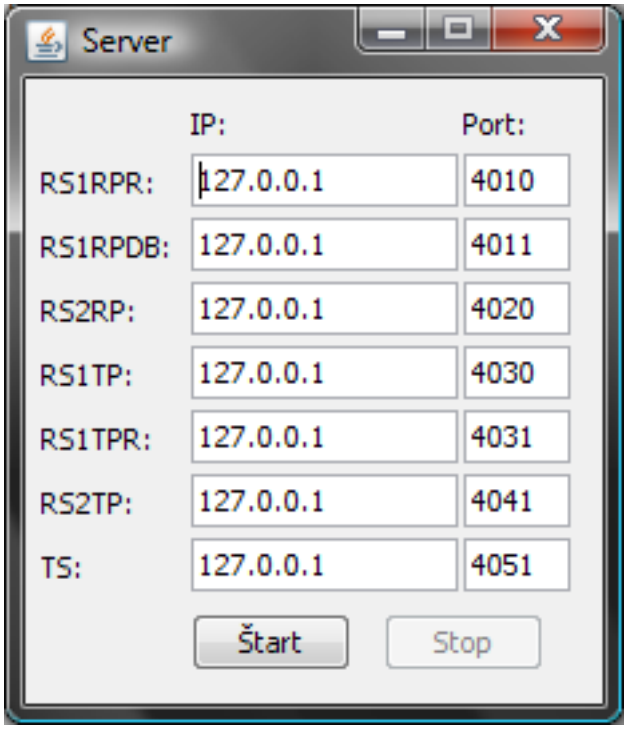
### 4.1.1 Server

Komponent systému fungujúci ako proxy-server medzi klientskou a servrovou časťou systému. *Server* preposiela správy medzi klientskou aplikáciou a servrovými aplikáciami systému.

***gui.IPSettingDialog*** - reprezentuje grafické rozhranie komponentu. Pozostáva z jedného okna obsahujúceho niekoľko textových polí, do ktorých sa nastavujú IP adresy a porty, na ktorých beží komunikácia so servrami $RS1$, $RS2$ a $TS$ systému. Grafické rozhranie ďalej obsahuje 2 tlačidlá pomocou, ktorých sa *Server* spúšťa a zastavuje.

***api.Server*** - hlavná trieda aplikačného rozhrania *Servra*

*void api.Server.startServer()* - volaná na stlačenie tlačidla na spustenie *Servra*. Vytvára štyri aplikačné vlákna. Prvé z nich vytvára inštanciu triedy *DatabasePhas-*




Obr. 3: Nastavenie IP adries a portov

*eServer*, druhé vytvára inštanciu triedy *RegistrationPhaseServer*, tretie vytvára inštanciu triedy *VoteCastingPhaseServer* a štvrté vytvára inštanciu triedy *ResultServer*.

*boolean api.Server.set()* - vracia *true* ak IP adresy a porty(argumenty metódy) vyplnené v textových poliach sú v správnom tvare

***api.DatabasePhaseServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií s klientom pred registračnou fázou volieb. Preposiela správy obsahujúce parametre volieb.

*void api.DatabasePhaseServer.run()* - metóda vytvárajúca vlákna na obsluhu komunikácie s viacerými klientmi naraz

***api.DatabasePhaseServerWorker implements Runnable*** - trieda obsluhu-

júca komunikáciu medzi klientom a registračným servrom *RS1* pred registračnou fázou volieb

*void api.DatabasePhaseServerWorker.run()* - metóda preposiela správy prijaté od klientskej aplikácie aplikácii *RS1RPHServer*

***api.RegistrationPhaseServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií s klientom počas registračnej fázy volieb

*void api.RegistrationPhaseServer.run()* - metóda vytvárajúca vlákna na obsluhu komunikácie s viacerými klientmi naraz

***api.RegistrationPhaseServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi klientom a registračnými servrami *RS1* a *RS2* počas registračnej fázy volieb

*void api.RegistrationPhaseServerWorker.run()* - metóda preposiela správy prijaté od klientskej aplikácie aplikáciam *RS1RPHServer* a *RS2RPHServer*

***api.VoteCastingPhaseServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií s klientom počas volebnej fázy volieb

*void api.VoteCastingPhaseServer.run()* - metóda vytvárajúca vlákna na obsluhu komunikácie s viacerými klientmi naraz

***api.VoteCastingPhaseServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi klientom a registračným servrom *RS1* počas volebnej fázy volieb

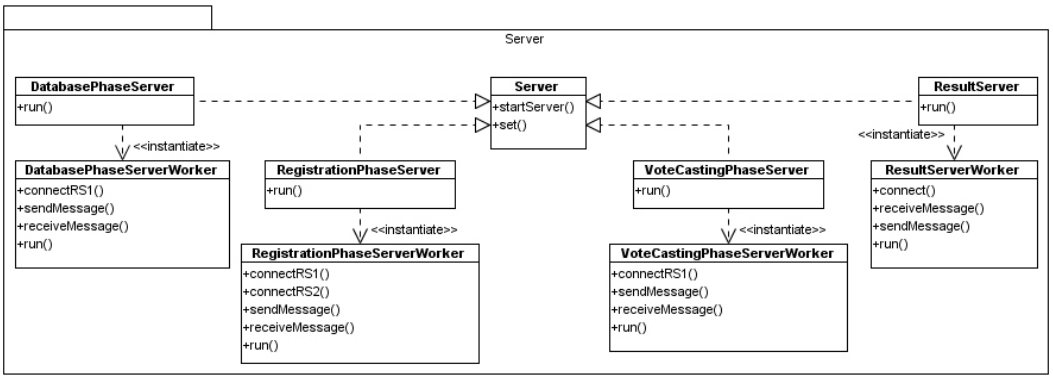
*void api.VoteCastingPhaseServerWorker.run()* - metóda preposiela správy prijaté od klientskej aplikácie aplikácii *RS1RPHServer*

***api.ResultServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií s klientom po ukončení volebnej fázy volieb. Aplikácia preposiela výsledky volieb a kontrolné zoznamy pre voliča.

*void api.ResultServer.run()* - metóda vytvárajúca vlákna na obsluhu komunikácie s viacerými klientmi naraz

***api.ResultServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi klientom, registračnými servrami *RS1*, *RS2* a počítacím servrom *TS* po ukončení volebnej fázy volieb

*void api.ResultServerWorker.run()* - metóda preposiela správy medzi klientskou aplikáciou, registračnými servrami *RS1*, *RS2* a počítacím servrom *TS*




Obr. 4: Server - diagram tried

### 4.1.2 RS1RPhServer

Komponent systému fungujúci ako registračný server *RS1* v registračnej fáze volieb. Získava z databázy voľby, možnosti volieb a parametre volieb, posiela ich klientskej aplikácii a vykonáva časť registrácie voliča tak, ako je to popísané v protokole.

***gui.MainDialog*** - reprezentuje grafické rozhranie komponentu. Pozostáva z jed-

ného okna obsahujúceho 2 tlačidlá pomocou, ktorých sa *RS1RPhServer* spúšťa a zastavuje.

***api.RS1RPhServer*** - hlavná trieda aplikačného rozhrania *RS1RPhServra*

*void api.RS1RPhServer.startServer()* - volaná na stlačenie tlačidla na spustenie *RS1RPhServra*. Vytvára dve aplikačné vlákna. Prvé z nich vytvára inštanciu triedy *RS1RPhRegServer*, druhé vytvára inštanciu triedy *RS1RPhDBServer*.

***api.RS1RPhDBServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií so *Servrom* pred registračnou fázou volieb. Posiela správy obsahujúce parametre volieb.

*void api.RS1RPhDBServer.run()* - metóda vytvárajúca vlákna na obsluhu komunikácie so *Servrom* preposielajúcim správy od klientskej aplikácie

***api.RS1RPhDBServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi *RS1RPhServrom* a *Servrom* pred registračnou fázou volieb

*Object[][] api.RS1RPhDBServerWorker.ziskajVolby()* - metóda získava z databázy zoznam volieb, v ktorých môže volič žiadajúci o tento zoznam voliť

*byte[][] api.RS1RPhDBServerWorker.retrieveVotingKeys()* - metóda získava z databázy parametre volieb potrebné na vytvorenie volebného kľúča voliča

*Object[][] api.RS1RPhDBServerWorker.vyberVolieb()* - metóda získava z databázy možnosti volieb pre voľby, ktoré si zo zoznamu volieb vybral volič

*void api.RS1RPhDBServerWorker.run()* - metóda posiela zoznam volieb, parametre volieb a možnosti volieb pre voľby, ktoré si zo zoznamu volieb vybral volič

***api.RS1RPhRegServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií so *Servrom* počas registračnej fázy volieb

*void api.RS1RPhRegServer.run()* - metóda vytvára vlákna na obsluhu komunikácie so *Servrom* preposielajúcim správy od klientskej aplikácie.

***api.RS1RPhRegServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi *RS1RPhServrom* a *Servrom* počas registračnej fázy volieb

*boolean api.RS1RPhRegServerWorker.skontrolujUcast()* - metóda, ktorá skontroluje, či volič môže voliť v ním zvolených voľbách. Vracia *true*, ak volič môže v daných voľbách voliť.

*void api.RS1RPhRegServerWorker.run()* - vykonáva časť registrácie voliča tak, ako je to popísané v protokole

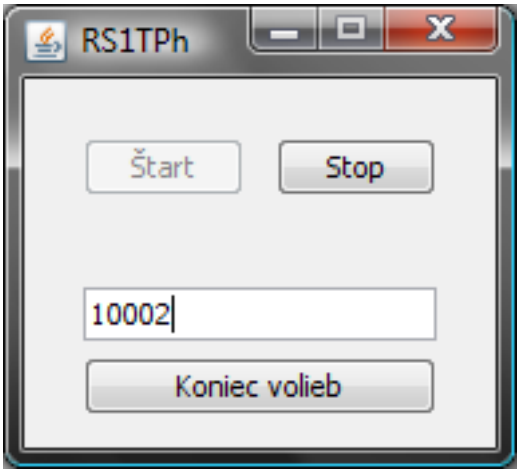
### 4.1.4 RS1TPhServer

Komponent systému fungujúci ako registračný server *RS1* vo volebnej fáze volieb. Vykonáva časť volieb a počítacej fázy volieb tak, ako je to popísané v protokole.

***gui.MainDialog*** - reprezentuje grafické rozhranie komponentu. Pozostáva z jedného okna obsahujúceho 2 tlačidlá pomocou, ktorých sa *RS1TPhServer* spúšťa a zastavuje. Grafické rozhranie ďalej obsahuje textové pole a tlačidlo *„Ukončenie volieb“*. Tlačidlo *„Ukončenie volieb“* volá metódu na aplikačnom rozhraní, ktorá ukončí voľby s číslom, ktoré je vyplnené v textovom poli.

***api.RS1TPhServer*** - hlavná trieda aplikačného rozhrania *RS1TPhServra*

*void api.RS1TPhServer.loadKeyStore()* - metóda zo súboru načíta PKCS#12 keystore, obsahujúci verejný a súkromný kľúč servra *RS1* a taktiež verejné kľúče ostatných servrov a všetkých voličov




Obr. 5: RS1TPhServer - grafické rozhranie

*void api.RS1TPhServer.startServer()* - volaná na stlačenie tlačidla na spustenie *RS1TPhServra*. Metóda vytvára vlákna na obsluhu komunikácie so *Servrom* preposielajúcim správy od klientskej aplikácie.

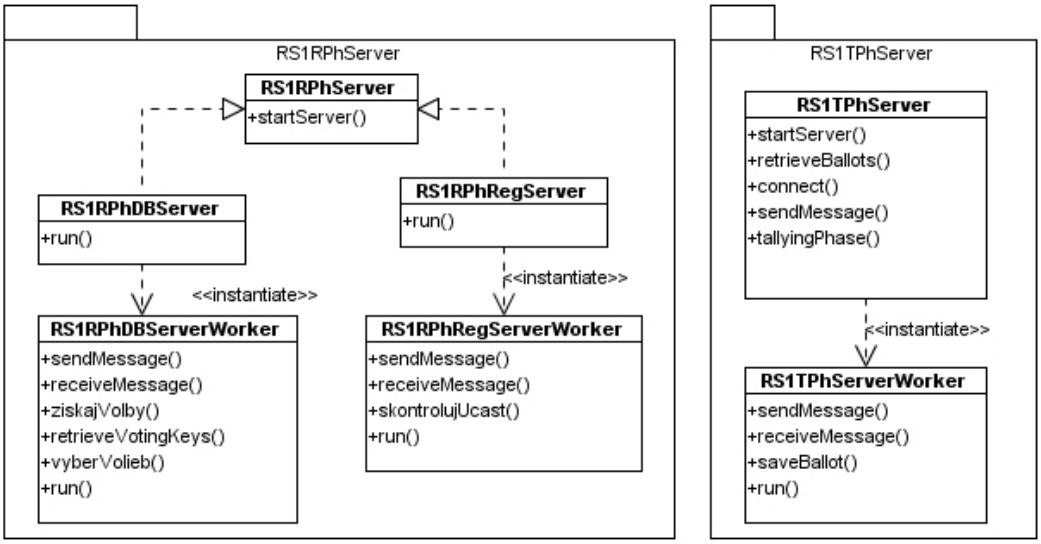
*byte[][][] api.RS1TPhServer.retrieveBallots(String)* - metóda na získanie balíčkov s hlasmi a ich digitálnych podpisov z databázy volaná z metódy *tallyingPhase()*. Parametrom metódy je číslo volieb, ktorých balíčky s hlasmi sa vyberajú.

*void api.RS1TPhServer.tallyingPhase(String)* - metóda volaná na stlačenie tlačidla *„Ukončenie volieb“*. Metóda vykonáva časť počítacej fázy vykonávanej servrom *RS1* tak, ako je to popísané v protokole.

***api.RS1TPhServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi *RS1TPhServrom* a *Servrom* počas volebnej fázy volieb

*void api.RS1TPhServerWorker.saveBallot()* - metóda na uloženie balíčka s hlasom a jeho digitálneho podpisu do databázy volaná z metódy *run()*. Parametrami metódy sú čísloVolieb, pole byte-ov obsahujúce balíček s hlasom a pole byte-ov obsahujúce digitálny podpis balíčka s hlasom.

*void api.RS1TPhServerWorker.run()* - vykonáva časť voľby voliča, v ktorej server *RS1* komunikuje s klientskou aplikáciou tak, ako je to popísané v protokole




Obr. 6: RS1Server - diagram tried

### 4.1.3 RS2RPhServer

Komponent systému fungujúci ako registračný server *RS2* v registračnej fáze volieb. Vykonáva časť registrácie voliča tak, ako je to popísané v protokole.

***gui.MainDialog*** - reprezentuje grafické rozhranie komponentu. Pozostáva z jedného okna obsahujúceho 2 tlačidlá pomocou, ktorých sa *RS2RPhServer* spúšťa a zastavuje.

***api.RS2RPhServer*** - hlavná trieda aplikačného rozhrania *RS2RPhServra*

*void api.RS2RPhServer.loadKeyStore()* - metóda zo súboru načíta PKCS#12 keystore, obsahujúci verejný a súkromný kľúč servra *RS2* a taktiež verejné kľúče ostatných servrov a všetkých voličov

*void api.RS2RPhServer.startServer()* - volaná na stlačenie tlačidla na spuste-

nie *RS2RPhServra*. Metóda vytvára vlákna na obsluhu komunikácie so *Servrom* preposielajúcim správy od klientskej aplikácie.

***api.RS2RPhServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi *RS2RPhServrom* a *Servrom* počas registračnej fázy volieb

*boolean api.RS2RPhRegServerWorker.skontrolujUcast()* - metóda, ktorá skontroluje, či volič môže voliť v ním zvolených voľbách. Vracia *true*, ak volič môže v daných voľbách voliť.

*void api.RS2RPhServerWorker.run()* - vykonáva časť registrácie voliča tak, ako je to popísané v protokole

### 4.1.5 RS2TPhServer

Komponent systému fungujúci ako registračný server *RS2* v počítacej fáze volieb. Vykonáva časť počítacej fázy tak, ako je to popísané v protokole.

***gui.MainDialog*** - reprezentuje grafické rozhranie komponentu. Pozostáva z jedného okna obsahujúceho 2 tlačidlá pomocou, ktorých sa *RS2TPhServer* spúšťa a zastavuje.

***api.RS2TPhServer*** - hlavná trieda aplikačného rozhrania *RS2TPhServra*

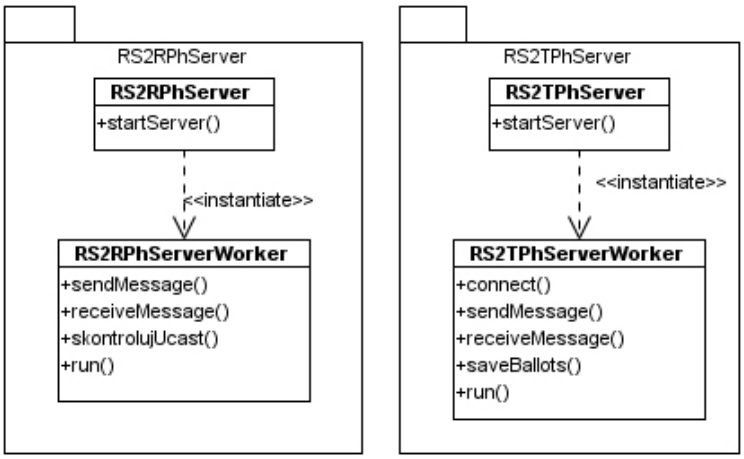
*void api.RS2TPhServer.loadKeyStore()* - metóda zo súboru načíta PKCS#12 keystore, obsahujúci verejný a súkromný kľúč servra *RS2* a taktiež verejné kľúče ostatných servrov a všetkých voličov

*void api.RS2TPhServer.startServer()* - volaná na stlačenie tlačidla na spustenie *RS2TPhServra*. Metóda vytvára vlákna na obsluhu komunikácie s *RS1TPhServrom* v počítacej fáze volieb.

***api.RS2TPhServerWorker implements Runnable*** - trieda obsluhujúca

komunikáciu medzi *RS2TPhServrom* a *RS1TPhServrom* počas počítacej fázy volieb

*void api.RS2TPhServerWorker.run()* - vykonáva časť počítacej fázy *RS2* volieb tak, ako je to popísané v protokole




Obr. 7: RS2Server - diagram tried

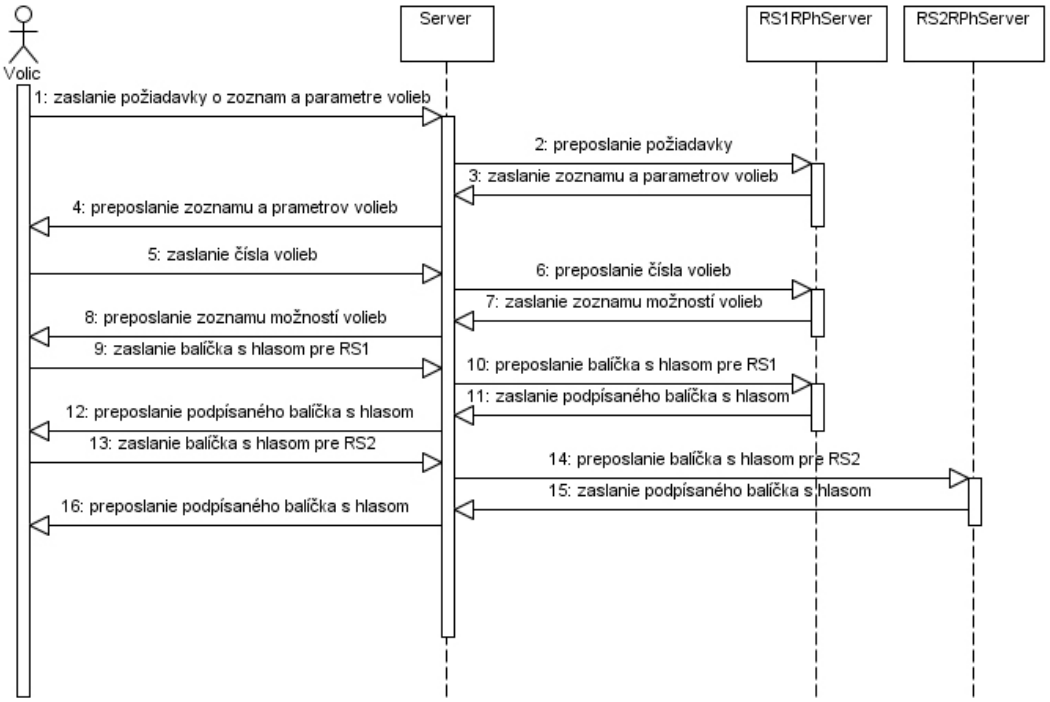
### 4.1.6 TSServer

Komponent systému fungujúci ako počítací server. Vykonáva časť počítacej fázy servra tak, ako je to popísané v protokole a odpovedá klientovi na požiadavku o zaslanie výsledkov volieb.

***gui.MainDialog*** - reprezentuje grafické rozhranie komponentu. Pozostáva z jedného okna obsahujúceho 2 tlačidlá pomocou, ktorých sa *TSServer* spúšťa a zastavuje.

***api.TSServer*** - hlavná trieda aplikačného rozhrania *TSServra*

*void api.TSServer.startServer()* - volaná na stlačenie tlačidla na spustenie *TSServer*. Vytvára dve aplikačné vlákna. Prvé z nich vytvára inštanciu triedy *TSCountServer*, druhé vytvára inštanciu triedy *TSResultServer*.




Obr. 8: Sekvenčný diagram - registračná fáza

***api.TSCountServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií s *RS2TPhServrom* počas sčítacej fázy volieb.

*void api.TSCountServer.loadKeyStore()* - metóda zo súboru načíta PKCS#12 keystore, obsahujúci verejný a súkromný kľúč servra *TS* a taktiež verejné kľúče ostatných servrov a všetkých voličov

*void api.TSCountServer.run()* - metóda vytvára vlákna na obsluhu komunikácie s *RS2TPhServrom* v počítacej fáze volieb.

***api.TSCountServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi *TSServrom* a *RS2TPhServrom* počas počítacej fázy volieb

*BigInteger[] api.TSCountServerWorker.ziskajParametreVolieb(String)* - metóda na získanie parametrov volieb na vytvorenie dešifrovacích kľúčov použitých na

dešifrovanie balíčkov s hlasmi

*Integer[] api.TSCountServerWorker.spracujHlas(byte[])* - metóda dekóduje hlas zakódovaný voličom do tvaru, v ktorom sa posiela (zašifrovaný) registračnému servru

*HashMap api.TSCountServerWorker.zratajHlasy(byte[][][])* - metóda zrátava hlasy z obdržaných balíčkov

*StringBuffer api.TSCountServerWorker.vytvorDokumentSVysledkami(String, HashMap)* - metóda na vytvorenie dokumentu s výsledkami, ktorý sa uloží do databázy a ktorý sa posiela voličovi, ak o to po skončení volieb požiada

*void api.TSCountServerWorker.ulozVysledkyDoDB(String, HashMap)* - metóda na uloženie výsledkov volieb a dokumentu s výsledkoami volieb do databázy

*void api.TSCountServerWorker.ukoncVysledkyVDB(String)* - metóda na ukončenie volieb v databáze

*void api.TSCountServerWorker.run()* - vykonáva časť počítacej fázy volieb počítacieho servra tak, ako je to popísané v protokole

***api.TSResultServer implements Runnable*** - trieda reprezentujúca vlákno vlákien komunikácií so *Servrom* po ukončení sčítacej fázy volieb.

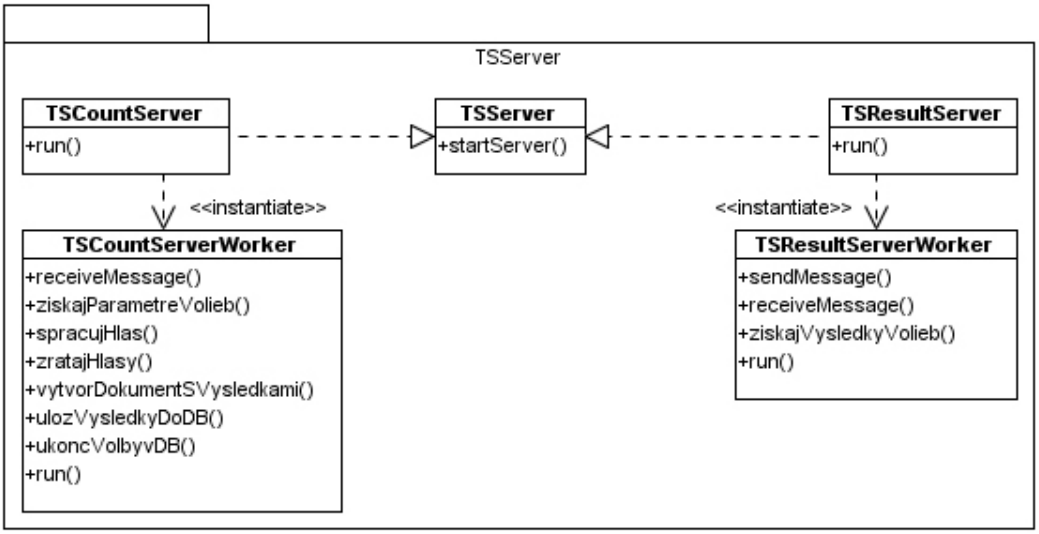
*void api.TSResultServer.run()* - metóda vytvára vlákna na obsluhu komunikácie so *Servrom* vo fáze volieb po ich ukončení.

***api.TSResultServerWorker implements Runnable*** - trieda obsluhujúca komunikáciu medzi *TSServrom* a *Servrom* po ukončení volieb

*byte[] api.TSResultServerWorker.ziskajVysledkyVolieb()* - metóda na získavanie

výsledkov volieb z databázy na ich preposlanie klientovi

*void api.TSResultServerWorker.run()* - metóda na odoslanie výsledkov klientovi o nich žiadajúcemu

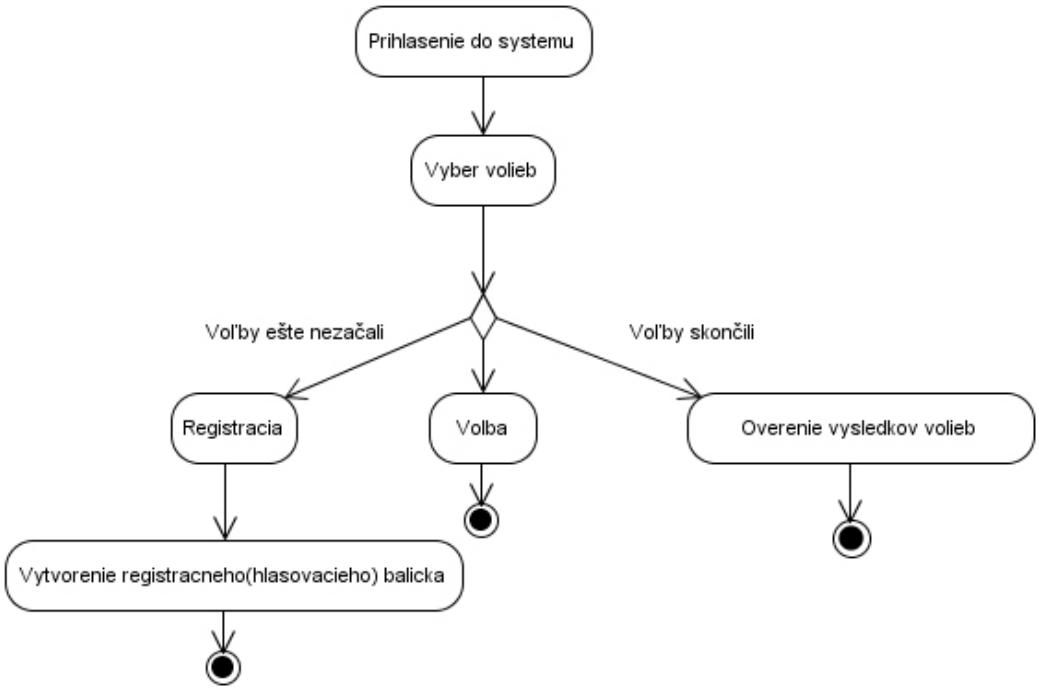



Obr. 9: TSServer - diagram tried

### 4.1.7 Klient

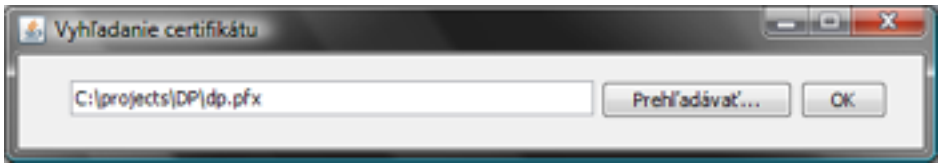
***gui.KlientDialog*** - abstraktná trieda, od ktorej dedia triedy grafického rozhrania klientskej aplikácie

***gui.KeyStoreBrowseDialog extends KlientDialog*** - trieda grafického rozhrania pre vyhľadanie PKCS#12 keystore súboru, ktorý obsahuje verejný a súkromný kľúč voliča a verejné kľúče sevrov. Obsahuje textové pole, do ktorého sa zapíše cesta k súboru a dve tlačidlá *„Prehľadávať...“* a *„OK“*. Na stlačenie tlačidla *„Prehľadávať...“* sa zobrazí vyhľadavací dialóg, v ktorom volič vyberie súbor s keystore-om. Po zatvorení dialógu sa do textového poľa zapíše cesta k vybratému súboru. Na stlačenie tlačidla *„OK“* sa zo súboru, cesta ku ktorému je zapísaná v textovom poli, načíta keystore a zobrazí sa nasledujúce okno.




Obr. 10: Aktivita voliča vo voľbách

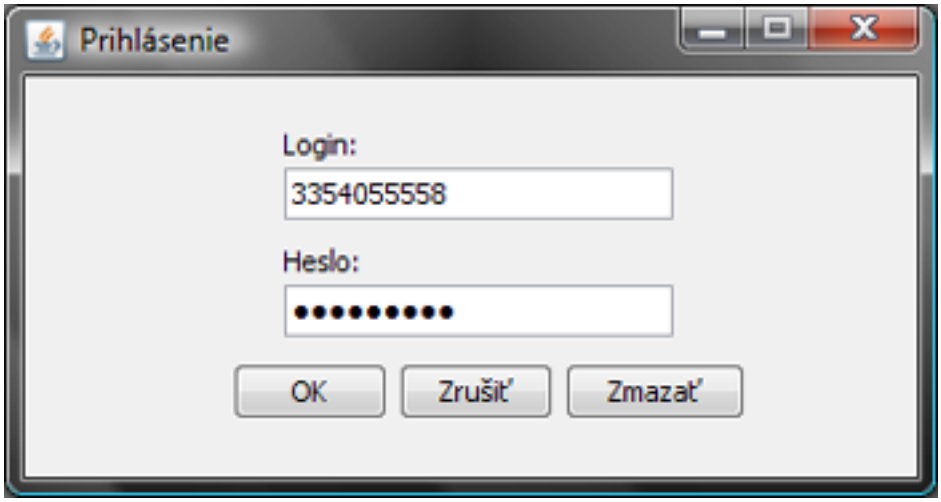
***gui.LoginDialog extends KlientDialog*** - trieda grafického rozhrania pre




Obr. 11: Vyhľadanie súboru s kľúčom

získanie súkromného kľúča voliča. Grafické rozhranie obsahuje dve textové polia, do ktorých sa vpisuje alias (jednoznačný identifikátor každého voliča) a heslo k súkromnému kľúču v keystore. Grafické rozhranie ďalej obsahuje tri tlačidlá. Na stlačenie tlačidla *„OK“*, ak je dvojica alias - heslo správna, tak sa z keystore-u načíta súkromný a verejný kľúč voliča s daným aliasom a zobrazí sa hlavné okno voličskej aplikácie. Na stlačenie tlačidla *„Zrušiť“* sa aplikácia zatvorí. Na stlačenie tlačidla *„Zmazať“* sa vymažú údaje zapísané v textových poliach.

***gui.MainDialog extends KlientDialog*** - hlavná trieda grafického rozhrania



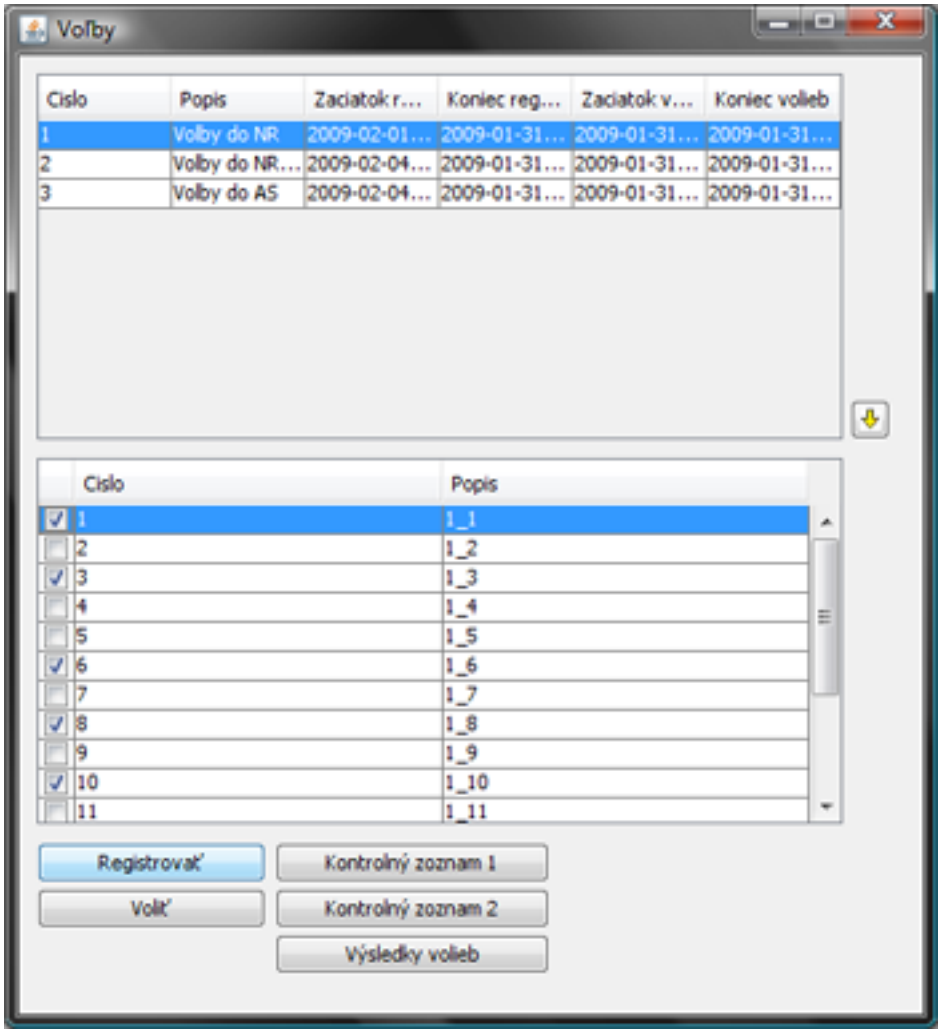

Obr. 12: Prihlásenie do systému

pre vykonanie registrácie a samotnej voľby. Obsahuje dve tabuľky(gridy). V prvom(hornom) gride sa zobrazuje zoznam volieb, ktorých sa môže volič zúčastniť v registračnej alebo volebnej fáze. V druhom(dolnom) gride sa zobrazuje zoznam možností, z ktorých si môže volič vybrať vo voľbách, ktoré si zvolil v hornom gride. Grafické rozhranie ďalej obsahuje tlačidlo na naplnenie dolného gridu možnosťami na voľbu vo voľbách. Tlačidlo *„Registrovať“* volá metódu na aplikačnom rozhraní na zaregistrovanie voliča vo vybraných voľbách. Po ukončení registrácie sa zobrazí dialógové okno, pomocou ktorého si volič uloží súbor obsahujúci balíček s hlasom a jeho digitálny podpis. Tlačidlo *„Voliť“* zobrazí dialógové okno, pomocou ktorého volič vyberie súbor uložený pri registrácii. Po vybratí súboru sa zavolá metóda na aplikačnom rozhraní na vykonanie voľby voliča.

*void gui.MainDialog.init()* - metóda na naplnenie horného gridu. Volá metódu na aplikačnom rozhraní na získanie zoznamu volieb.

***cls.Ballot implements Serializable*** - trieda používaná na uloženie balíčka s hlasom a jeho digitálneho podpisu, vytvoreného pri registrácii voliča vo voľbách.

***cls.VotingKeys*** - trieda používaná na uchovanie šifrovacieho a odpovedajúceho dešifrovacieho kľúča vytvorených voličom pre dané voľby




Obr. 13: Klient - grafické rozhranie

***api.Klient*** - hlavná trieda aplikačného rozhrania voličskej aplikácie

*Object[][] api.Klient.ziskajVolby(String)* - metóda volaná z metódy *gui.MainDialog.init()* grafického rozhrania. Zasiela na *RS1RPhDBServer* požiadavku a obdrží od neho zoznam volieb, ktorých sa môže volič, s idVolica zadaným ako argument metódy, zúčastniť.

*Object[][] api.Klient.vyberVolieb(String)* - metóda volaná na stlačenie tlačidla po vybratí volieb, v ktorých chce volič voliť. Zasiela na *RS1RPhDBServer* požia-

davku a obdrží od neho parametre volieb a zoznam možností volieb, z ktorých si volič vyberá vo vybratých voľbách.

*cls.VotingKeys api.Klient.createVotingKeys(BigInteger, BigInteger, BigInteger)* - vytvorí šifrovací a odpovedajúci dešifrovací kľúč jedinečný pre dané voľby pomocou dvoch náhodne zvolených parametrov

*byte[] api.Klient.vytvorHlasy(String)* - zakóduje zoznam vybratých možností volieb do tvaru aby ich *TSServer* vedel dekódovať

*cls.Ballot api.Klient.registrationPhase(String)* - metóda volaná na stlačenie tlačidla *Registrovať*. Vykonáva registráciu voliča vo vybratých voľbách tak, ako je popísaná v protokole. Metóda vracia inštanciu triedy *cls.Ballot* pripravenú na uloženie. Parametrom metódy je reťazec s číslom voľby.

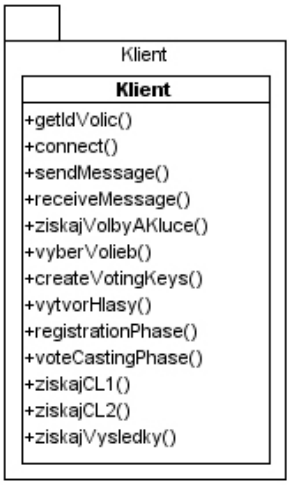
*void api.Klient.voteCastingPhase(cls.Ballot)* - metóda volaná na stlačenie tlačidla *Voliť*. Vykonáva voľbu voliča vo vybratých voľbách tak, ako je popísaná v protokole. Parametrom metódy je inštancia triedy *cls.Ballot* uložená pri registrácii voliča.

*String api.Klient.ziskajVysledky(cls.Ballot)* - metóda volaná na stlačenie tlačidla *Výsledky volieb*. Metóda získava výsledky vybraných volieb od *TSServra* a ukladá ich do *.html* súboru prezerateľného v internetovom prehliadači.

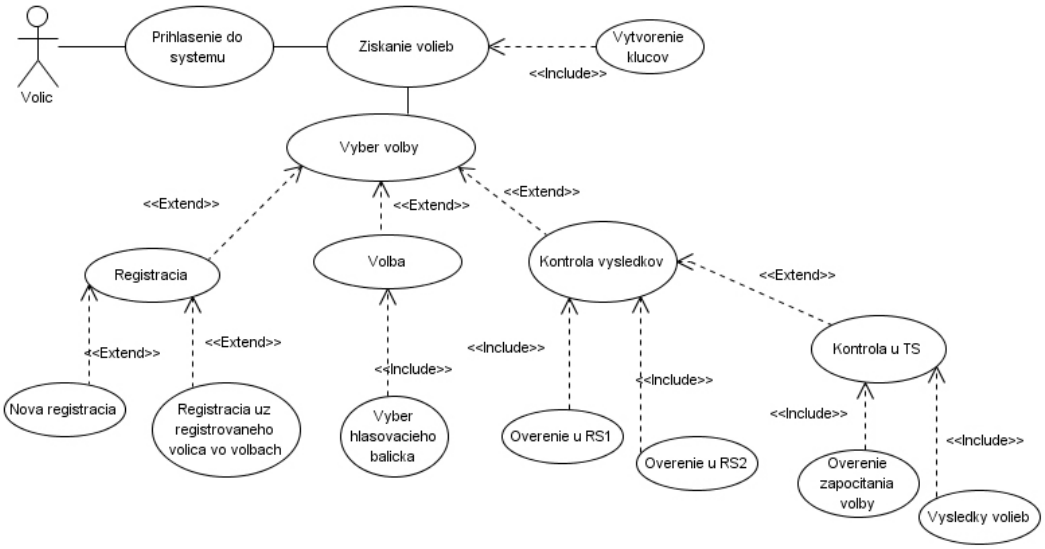
## 4.2 Databázový model

Databáza obsahuje nasledujúcich 5 tabuliek na ukladanie dát:

1. Volic - ukladajú sa v nej voliči a má nasledujúce stĺpce:
    - id
    - cislo - jedinečný identifikátor voliča
2. Volby - ukladajú sa v nej voľby a má nasledujúce stĺpce:

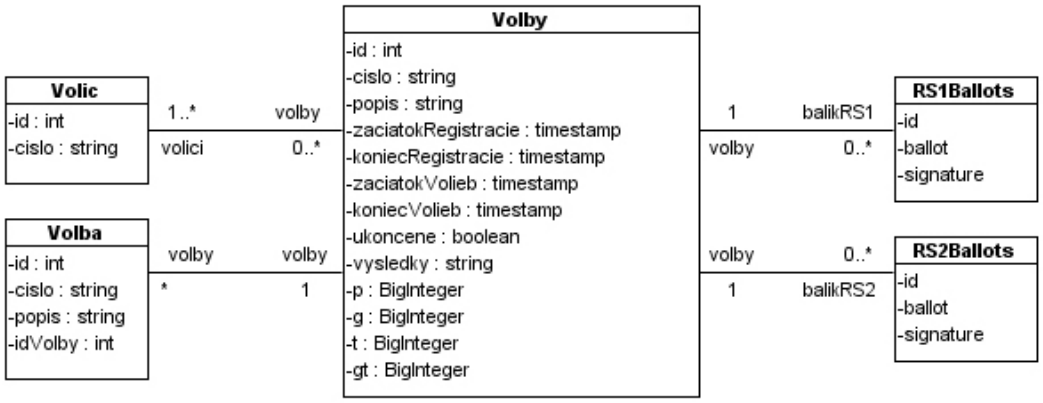



Obr. 14: Klient - diagram tried




Obr. 15: UseCase volby

- id
- cislo - jedinečný identifikátor volieb
- popis - stručný popis, o aký typ volieb ide
- zaciatokRegistracie
- koniecRegistracie
- zaciatokVolieb




Obr. 16: Databázový model

- koniecVolieb
- ukoncene
- p, g, gt - parametre volieb potrebné k vytvoreniu voličských kľúčov pre dané voľby
- t - súkromný parameter volieb TSServra

3. VolbyVolica - ukladajú sa v nej informácie o tom, v ktorých voľbách môže volič voliť a má nasledujúce stĺpce:
   - id
   - idVolby - cudzí kľúč tabuľky Volby
   - idVolic - cudzí kľúč tabuľky Volic
4. Volba - ukladajú sa v nej možnosti volieb v jednotlivých voľbách a má nasledujúce stĺpce:
   - id
   - cislo - jedinečný identifikátor voľby
   - popis - stručný popis voľby
   - idVolby - cudzí kľúč tabuľky Volby
5. RS1Ballots - ukladajú sa v nej balíčky s hlasmi a ich digitálne podpisy vytvorené vo volebnej fáze volieb a má nasledujúce stĺpce:

- id
- cisloVolieb - jedinečný identifikátor voľby
- ballot - balíček s hlasom
- signature - digitálny podpis balíčka s hlasom

6. RS2Ballots - ukladajú sa v nej balíčky s hlasmi a ich digitálne podpisy vytvorené vo volebnej fáze volieb a má rovnaké stĺpce ako tabuľka RS1Ballots.

# Literatúra


[1] Novotný, M.: *Design and analysis of a practical e-voting protocol*, to be published in the Proceedings of FIDIS/IFIP Internet Security & Privacy Summer School 2008, Springer IFIP series

[2] Rosner, I. M., Rosner, G.: *Electronic Voting Protocols and Schemes*, The Hebrew University of Jerusalem, Israel, 2002

[3] Rjašková, Z.: *Electronic voting schemes*, Diplomová práca, Bratislava, 2002

[4] Chaum, D.: *Elections with Unconditionally-Secret and Disruption Equivalent to Braking RSA*, Center for Mathematics and Computer Science, Kruislaan, Amsterdam, 1998

[5] Chaum, D.: *Blind signatures for untraceable payments*, Advances in Cryptology - Crypto, Springer-Verlag, 1983

[6] Okamoto, T.: *An electronic voting scheme*, IFIPWorld Conference on IT Tools, Chapman & Hall, 1996

[7] Okamoto, T.: *Receipt-Free Electronic Voting Schemes for Large Scale Elections*, Security Protocols Workshop 1997, Lecture Notes in Computer Science, vol. 1361, Springer Verlag, 1998