PROJECT WORK Algoritmi e Protocolli per la Sicurezza

L.M. Ingegneria Informatica

Docenti: I. Visconti - V. Iovino *A.A. 2021-2022*

OBIETTIVO

Realizzare un sistema di voto elettronico/remoto per elezioni con vari candidati anche su comunità di poche migliaia di persone.

CARPENTIERI EUGENIO 0622701804
CASABURI ADOLFO 0622701814
FALCONE ROBERTO 0622701790
FERRAIOLI LUIGI 0622701853

Sommario

PROJECT WORK Algoritmi e Protocolli per la Sicurezza 0
WP1 3
Il sistema 3
Completeness 3
Threat model 4
INTEGRITÀ 5
PRIVACY 5
VIVEZZA 5
AUTENTICITÀ 5
INCOERCIBILITÀ 6
WP2 6
Premessa 6
Fase di registrazione 6
Preparazione della struttura (fase *T0-T1*) 7
Preparazione al voto 8
Fase di voto effettivo 9
Fase di shuffling 9
Fase di checking 10
Fase di apertura e conteggio 10
WP3 11
COMPLETEZZA 11
INTEGRITÀ 11
PRIVACY 11
VIVEZZA 12
AUTENTICITÀ 12
INCOERCIBILITÀ 12
EFFICIENZA 12
TRASPARENZA 12
Server provider corrotto 13
Imbroglioni in collusione 13
Gestori d'identità digitale come avversari 13
GRAFICI 14
WP4 15
Classi 15
MAIN.JAVA 16

UML 17
CODICE JAVA 17
Anonymizer 17
Authenticator 19
BulletinBoard 22
CentralServer 23
ElGamalCT 24
ElGamalPK 24
ElGamalSK 25
ElGamalUtils 25
Elettore 26
InfoVoto 28
KeyUtils 29
Tallier 30
Main 32
INDICE FIGURE 35
INDICE TABELLE 35

WP1

Il sistema

L'impostazione desiderata comprende la presenza di un elettore il cui obiettivo è quello di esprimere una preferenza tra vari candidati nell'ambito di un'elezione comunale. Sussiste il vincolo che un elettore possa esprimere un'unica preferenza, la quale può essere vista come un messaggio inviato dall'elettore al sistema elettorale.

Le motivazioni principali degli attacchi potrebbero essere le esigenze contrastanti del sistema elettorale, il cui auspicio è di mantenere la regolarità dell'elezione, e degli individui coinvolti (candidati, elettori o individui/gruppi di essi, etc.) che vorrebbero influenzare l'esito per il proprio tornaconto.

L'elezione si svolge in quattro fasi. In primo luogo, c'è una finestra temporale *T0-T1* in cui vengono effettuate operazioni di registrazione e preparazione della struttura. Dopodiché, durante l'intervallo *T1-T2* ogni elettore esprime e invia la propria preferenza. In una finestra temporale successiva *T2-T3*, le preferenze degli elettori vengono raggruppate e contate in via preliminare. Infine, dopo *T3*, i risultati dei vari gruppi vengono sommati ottenendo così il risultato finale dell'elezione.

In alcuni casi può anche essere coinvolta la giustizia. Se vengono riscontrare irregolarità durante le votazioni, le autorità o i privati cittadini possono sporgere denuncia. Ha senso che l'entità nota J (Giustizia) venga chiamata in causa solo in caso di controversia, evitando quindi il suo coinvolgimento il più possibile. Tuttavia, quest'ultima parte è fuori dal sistema digitalizzato, che quindi può solo aiutare la giustizia a rilevare eventuali controversie.

Completeness

Siano $E_1, \dots, E_n$ i potenziali elettori, $A_1, \dots, A_k$ i server Authenticator delegati ad accertarsi dell'idoneità al voto, B la bulletin board, $Az_1, \dots, Az_j$ i server Anonymizer che si occupano di eliminare la corrispondenza tra l'indirizzo IP dell'elettore e la sua preferenza, $T_1, \dots, T_m$ i server Tallier incaricati di contare le preferenze degli elettori, S_c il Central Server incaricato di effettuare il conteggio finale e pubblicare il risultato dell'elezione.

Trattandosi di un sistema di voto realizzato per un'elezione comunale, possiamo assumere che per ogni tipologia di server coinvolto si mettono a disposizione circa tanti server quanti sono i seggi utilizzati in quel comune durante delle elezioni tradizionali.

Dati i vari candidati, ogni elettore E_i può partecipare all'elezioni esprimendo una preferenza all'interno della finestra temporale *T1-T2*, fornendo un input che rappresenta il candidato per il quale vuole esprimere la propria preferenza.

I server Authenticator si occupano di controllare se l'elettore abbia già espresso una preferenza e in caso contrario lo autorizzano al voto. Dopodiché gli elettori inviano la preferenza autorizzata alla Bulletin Board.

Una volta terminata la finestra temporale *T1-T2*, i server Anonymizer prelevano tutti i voti dalla Bulletin Board e, solo dopo averli mescolati, li inviano ai vari server Tallier, i quali effettuano il conteggio delle preferenze ricevute assegnando un punteggio a ciascuno dei candidati e pubblicando infine la lista dei voti conteggiati.

In seguito, al termine della finestra temporale *T2-T3*, il Central Server somma i risultati dei server Tallier, ottenendo così il risultato finale dell'elezione, ovvero quale tra i vari candidati ha ricevuto più preferenze.

I server Authenticator e i server Tallier costituiscono dei sistemi distribuiti, ovvero insiemi di calcolatori in grado di condividere risorse e attività in modo che agli utenti appaiano come un unico sistema coerente.

La giustizia J può essere invocata ogni qualvolta viene rilevata un'irregolarità durante le votazioni e anche interrompere il processo di voto elettronico annullando i risultati.

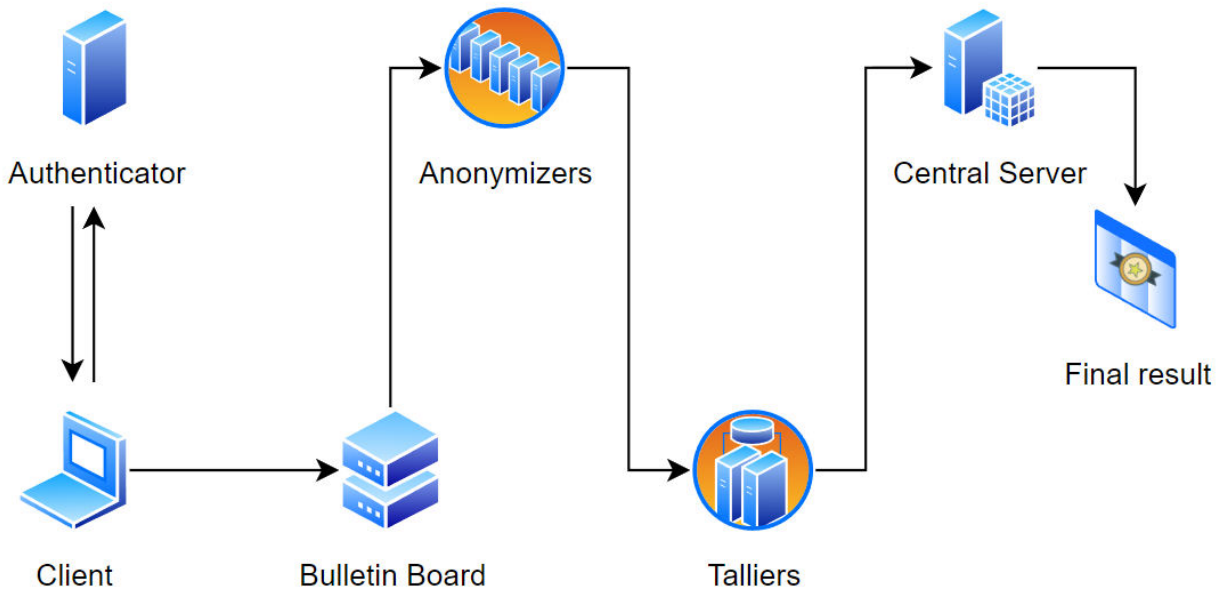


Figura 1, Rappresentazione grafica delle fasi dell'elezione

Threat model

- ***Elettore furbo***. Un elettore è ovviamente interessato a far vincere l'elezione al proprio candidato preferito. Un elettore avversario potrebbe tentare di votare molteplici volte, anche esprimendo preferenze diverse, all'interno della finestra temporale *T1-T2*. Non ha accesso a nessun canale di comunicazione, ma ha solo accesso al client.
- ***Avversario catfish***. Un qualsiasi avversario potrebbe essere interessato a fingersi un'altra persona per avvalersi del diritto di voto, qualora non ce l'abbia, oppure per fornire più di una preferenza in modo da avvantaggiare uno specifico candidato, sfruttando ad esempio gli astenuti. Non ha accesso a nessun canale di comunicazione, ma potrebbe avere accesso al client o ad uno dei server Authenticator.
- ***Avversario intimidatorio***. Un avversario potrebbe cercare di obbligare un elettore ad esprimere una determinata preferenza contro la sua volontà oppure a fornirgli le sue credenziali. Non ha accesso a nessun canale di comunicazione, ma potrebbe avere contatto diretto con l'elettore.
- ***Intruso impiccione***. Un qualsiasi avversario, interno o esterno al sistema, potrebbe essere interessato a intercettare la preferenza di un elettore, violando così il suo anonimato. Ha accesso ai canali di comunicazione o anche ai server coinvolti nella finestra temporale *T2-T3*.
- ***Intruso pettegolo***. Un qualsiasi avversario, interno o esterno al sistema, potrebbe essere interessato a carpire informazioni su risultati intermedi dell'elezione, allo scopo, ad esempio, di influenzarne l'andamento. Ha accesso ai canali di comunicazione tra i server Tallier e il Central Server oppure ai dati dei server Tallier.
- ***Intruso malintenzionato***. Un qualsiasi avversario, interno o esterno al sistema, potrebbe essere interessato a modificare o annullare la preferenza di un elettore oppure interromperne l'invio (attacchi *DoS,* specialmente distribuiti). Ha accesso ai canali di comunicazione e può manomettere attivamente i dati scambiati e memorizzati sui server coinvolti. Questo avversario può avere una potenza computazionale realisticamente forte.

- ***Intruso ladro***. Un qualsiasi avversario, interno o esterno al sistema, potrebbe provare a rubare credenziali di accesso degli elettori. Ha accesso ai canali di comunicazione oppure al server che mantiene le credenziali degli elettori. Questo avversario può avere una potenza computazionale realisticamente forte da provare un attacco forza bruta.
- ***Autorità del sistema di voto astuta***. Un'autorità del sistema di voto avversaria potrebbe decidere di voler modificare il conteggio totale delle preferenze e comunicare un risultato diverso da quello reale. Ha accesso ai canali di comunicazione e ai server coinvolti sia nella finestra temporale *T2-T3* sia dopo *T3*.
- ***Autorità del sistema di voto corrotta***. Un'autorità del sistema di voto avversaria potrebbe colludere con un intruso malintenzionato per modificare o eliminare preferenze, nonché aggiungere voti falsi. Ha accesso ai canali di comunicazione e ai server coinvolti sia nella finestra temporale *T2-T3* sia dopo *T3*.
- ***Server provider corrotto***. La società che si occupa della fornitura dei server potrebbe essere corrotta o attaccata in modo da sabotare l'andamento dell'elezione. Ad esempio, il server Bulletin Board o uno dei server Tallier potrebbero essere preimpostati per scartare alcuni voti che gli vengono inviati. Può intervenire durante lo svolgimento dell'elezione, ma anche prima della finestra temporale *T1-T2*, fornendo server corrotti.
- ***Imbroglioni in collusione***. Sottoinsiemi dei suddetti avversari possono colludere per raggiungere i loro obiettivi.

INTEGRITÀ

In presenza di avversari il sistema di voto dovrebbe comunque garantire che:

- **I.1** una volta espressa, una preferenza non venga modificata.
- **I.2** alla chiusura della finestra temporale *T2-T3* il conteggio totale delle preferenze non sia modificabile.

PRIVACY

In presenza di avversari il sistema di voto dovrebbe comunque garantire che:

- **P.1** l'identità degli elettori resti anonima e separata dalla preferenza espressa, cosicché nessun attore, sia onesto che disonesto, possa ricondurre una preferenza all'identità dell'elettore che l'ha espressa.
- **P.2** nessun attore, onesto o disonesto che sia, conosca risultati intermedi prima dell'inizio della finestra temporale *T2-T3*, in modo tale che il conteggio dei voti non influenzi la votazione.

VIVEZZA

In presenza di avversari il sistema di voto dovrebbe comunque garantire che:

- **V.1** le preferenze espresse dagli elettori siano registrate entro la finestra temporale *T1-T2*.

AUTENTICITÀ

In presenza di avversari il sistema di voto dovrebbe comunque garantire che:

- **A.1** un elettore debba essere in grado di esprimere la propria preferenza al più una volta.
- **A.2** un elettore debba essere in grado di verificare che la preferenza espressa sia stata conteggiata correttamente.
- **A.3** tutti gli attori abbiano la possibilità di verificare che il conteggio finale sia realmente la somma delle preferenze inviate.
- **A.4** non sia permesso a nessuno di votare per altri.
- **A.5** sia permesso di votare solo agli aventi diritto.

INCOERCIBILITÀ

In presenza di avversari il sistema di voto dovrebbe comunque garantire che:

- **IN.1** sia permesso di votare liberamente malgrado ci sia qualcuno interessato a far votare secondo sue indicazioni.

WP2

Premessa

Questo sistema di voto elettronico si basa su un'infrastruttura a chiave pubblica (PKI) che permette agli attori in gioco di essere certi che le altre parti con cui comunicano siano "al sicuro", ovvero che le loro identità e le loro chiavi siano valide e affidabili. Per far ciò i server coinvolti devono disporre di un certificato digitale rilasciato da una Certification Autority (CA). Durante qualsiasi comunicazione che comprende lo scambio di messaggi, i server coinvolti devono inviare il proprio certificato digitale, in particolare tutta la catena di certificati fino alla CA radice. In questo modo i client possono verificare che il certificato ricevuto sia valido, ovvero che la connessione stabilita sia con il server corretto. Inoltre, i client hanno la certezza che la chiave pubblica possa essere utilizzata nel modo per cui è stata certificata dalla CA.

Sia la scelta delle credenziali d'accesso da parte degli elettori sia l'inizializzazione di tutte le chiavi pubbliche e private vengono rieseguite ad ogni nuova elezione.

Da questo momento in poi assumiamo che qualsiasi comunicazione che comprende uno scambio di messaggi avvenga attraverso l'utilizzo di TCP/IP + TLS e comprenda l'autenticazione del server mediante il proprio certificato digitale.

Fase di registrazione

Ai potenziali elettori $\mathrm{E}_1, \ldots, \mathrm{E}_n$ viene fornito in anticipo (durante la fase di preparazione, nell'intervallo di tempo *T0-T1*) un e-token.

L'e-token funziona da certificato elettorale digitale che accerta l'idoneità al voto. Il suo scopo è quello di identificare in modo univoco il singolo elettore, tramite informazioni personali (nome, cognome, data di nascita, luogo di nascita, codice fiscale, residenza, etc.) e viene rilasciato da gestori d'identità digitale.

In caso ci sia bisogno di revocare il diritto al voto di un elettore, spetta ai gestori d'identità digitale inserire l'e-token all'interno di una "certificate revocation list" in modo da escludere l'elettore dal sistema di voto.

Identificandosi tramite l'e-token fornito l'elettore può scegliere le credenziali di username e password per accedere alla piattaforma di voto. Una volta confermata la scelta di username e password il sistema genera una stringa casuale $rand$. A questo punto le credenziali e la stringa casuale vengono inviate ad uno dei server Authenticator.

Il sistema distribuito costituito dai server Authenticator mantiene memorizzati, all'interno del proprio database, tutti gli e-token rilasciati. Inoltre, possiede una corrispondenza tra e-token e credenziali scelte dall'elettore, questo gli permette di verificare se l'elettore si sia già registrato o meno. Le informazioni sulle credenziali degli elettori vengono mantenute all'interno del database come una tripla $(username, SHA256(rand \,||\, password), rand)$ in modo tale da, in caso di autenticazione di un elettore, poter verificare se la password è corretta e, al contempo, proteggerla da un potenziale attaccante. Infine, conserva per ogni elettore un'informazione che consente di appurare se abbia già votato o meno.

Ciascuno stato può scegliere il metodo di autenticazione digitale più opportuno, basandosi ad esempio su quello in vigore al momento, come *SPID* per il governo italiano o *Alicem* per quello francese, o ancora generare un sistema di certificato elettorale digitale apposito. Nel primo caso, le credenziali di accesso alla piattaforma di voto coincidono con quelle del sistema di identità digitale utilizzato.

Preparazione della struttura (fase *T0-T1*)

Tutti gli Authenticator $A_1, \dots, A_k$ possiedono un'unica chiave privata SK_A e un'unica chiave pubblica PK_A generate secondo lo schema di cifratura RSA. Dato un generico server Authenticator A_i, scelto casualmente, esso:

- Esegue l'algoritmo $\mathcal{G}$ di generazione, quindi definisce il gruppo ciclico $\mathcal{G}_q$, sottoinsieme del gruppo moltiplicativo Z_p^*, un elemento g che lo genera, nonché due numeri primi q e p, con p taglia del gruppo Z_p^*, e q legato a p attraverso la relazione: $p = 2q + 1$.
- Calcola $n = p \cdot q$ e $\phi(n) = (p - 1)(q - 1)$;
- Sceglie un numero $1 < e < \phi(n)$ tale che e è co-primo con $\phi(n)$ (non è necessario che e sia primo);
- Calcola $d = e - 1 \, mod \, \phi(n)$;
- La chiave pubblica è $PK_A = (e, n)$.
- La chiave segreta è $SK_A = (d, n)$.

Tutti gli Anonymizer $Az_1, \dots, Az_j$ eseguono l'algoritmo di generazione di chiavi PK_{Az_i} e SK_{Az_i}, per $i = 1, \dots, j$, per lo schema di cifratura ElGamal, essi:

- Esegue l'algoritmo $\mathcal{G}$ di generazione, quindi definisce il gruppo ciclico $\mathcal{G}_{q'}$, sottoinsieme del gruppo moltiplicativo $Z_{p'}^*$, un elemento g' che lo genera, nonché due numeri primi q' e p', con p' taglia del gruppo $Z_{p'}^*$, e q' legato a p attraverso la relazione: $p = 2q + 1$.
- Sceglie casualmente un intero x nel gruppo additivo $Z_{q'}$
- Calcola $y' = g^x \, mod \, p'$
- La chiave pubblica è $PK_{AZ_i} = (p', q', g', y')$
- La chiave segreta à $SK_{AZ_i} = x$

Inoltre, la stessa procedura effettuata dai server Anonymizer viene ripetuta anche dal Central Server S_c, il quale ottiene la chiave pubblica PK_{S_c} e la chiave segreta SK_{S_c}. Quest'ultima viene condivisa tra tutti i Tallier $T_1, \dots, T_m$ in modo tale che ciascuno di essi possieda una share sh_i con $i = 1, \dots, m$, ottenuta come segue:

- S_c sceglie casualmente a_i nel gruppo additivo Z_q, con $1 \leq i \leq m$
    - Definendo quindi il polinomio $p(x) = SK_{S_c} + \sum_{1 \leq i \leq m} a_i x^i$
- $sh_i = p(x_i)$, con $x_i = i$

Preparazione al voto

L'elettore E, durante la finestra temporale *T1-T2*, effettua il login-in sulla piattaforma di voto. Se correttamente autenticato, può iniziare la fase di preparazione al voto, utilizzando i parametri pubblici $z = (p, q, g)$ forniti dal Central Server S_C e inviando la preferenza espressa al server Authenticator A_i assegnatogli casualmente tra i vari $A_i, \dots, A_k$.

Formalmente:

- E prepara il plaintext M con all'interno la preferenza espressa
- Cifra M con la chiave pubblica del Central Server PK_{S_c}
    - E calcola il ciphertext $x(M) =< u, v >$ su input M
        - $u = g^r$
        - $v = M \cdot PK_{S_c}^r$
            - dove r è random in Z_q
- Applica il blind factor con la chiave pubblica del server A_i
    - $b\big(x(M)\big) = x(M) \cdot s^e \ mod\ n$
        - dove s è un valore a caso tra 1 e n
        - dove e è l'esponente pubblico di PK_A
- Invia ad A_i userID, password e blind in modo tale da ottenere l'autorizzazione a votare
    - $E \rightarrow A_i : \ userID, password, b(x(M))$

A_i verifica userID e password; successivamente, se tutto va a buon fine, controllando se non ha già firmato il suo certificato elettorale, si accerta che E non abbia già votato e in tal caso restituisce il voto firmato con la propria chiave privata SK_A autorizzando E a votare.

Formalmente :

- A_i verifica se userID e password sono corretti
    - Accedendo al database, trova l'userID e, attraverso la stringa $rand$ memorizzata e la password ricevuta, calcola $SHA(rand \,||\, password)$ e lo confronta con il valore associato.
- A_i controlla se ha già firmato la preferenza di E
    - Nel caso in cui E non abbia già votato, firma $b(x(M))$ con la propria chiave privata SK_A, lo memorizza nel database associandolo alla tripla relativa ad E e glielo restituisce.
        - $A_i \rightarrow E : S_A(b(x(M))$
            - Dove $S_A(b\big(x(M)\big) = b\big(x(M)\big)^d \ mod\ n$
                - Dove d è l'esponente privato di SK_A
    - Altrimenti, non autorizza il voto

In entrambi i casi A_i invia una notifica all'elettore comunicando l'esito del controllo. In questo modo, qualora E non abbia ancora votato ma riceva una notifica di autorizzazione al voto, allora può sempre appellarsi alla giustizia J.

Il sistema distribuito formato dai server $A_1, \dots, A_k$ mantiene memorizzata e rende pubblica alla fine della finestra temporale *T1-T2* la lista:

…	…	…
$\boldsymbol{userID}$	$\boldsymbol{b(x(M))}$	$\boldsymbol{S_A(b(x(M))}$
…	…	…

Tabella 1, Lista pubblicata dai server Authenticator

Verifica della firma

L'elettore E verifica che la firma di A_i sia autentica, usando la chiave pubblica PK_A su $S_A(b(x(M)))$, per riottenere $b(x(M))$:

- $[S_A\left(b(x(M))\right)]^e mod\ n = \left[b(x(M))^d\right]^e = b(x(M))^{d\cdot e} = b(x(M))\ mod\ n$

Se il controllo va a buon fine E toglie il blind factor ottenendo il certificato originale. Infatti, dato che:

- $S_A(b(x(M)) = b(x(M))^d\ mod\ n = (x(M)\cdot s^e)^d\ mod\ n$
- $S_A(b(x(M)) = x(M)^d \cdot s\ mod\ n$

Allora1:

- $S_A(x(M)) = \frac{S_A(b(x(M))}{s}\ mod\ n = x(M)^d\ mod\ n$

Se il controllo fallisce, E rivendica l'errore dimostrando che SK_A non è autentica.

Fase di voto effettivo

L'elettore E cifra, tante volte quanti sono i server Anonymizer, la coppia $< x(M), S_A(x(M)) >$ con le loro chiavi pubbliche PK_{Az_i} per $i = 1, \dots, j$.

Formalmente:

$for\ i = 1\ to\ j$:

- $C_i =< x(M), S_A(x(M)) >$
- $u_i = g^{r_i'}$
- $v_i = C_i \cdot PK_{Az_i}^{r_i'}$
    - dove r_i' è random in Z_q
- $y_i(C_i) =< u_i, v_i >$

Una volta terminate le j cifrature si ottiene l'output $y_j(C_j) =< u_j, v_j >$ che viene inviato da E alla Bulletin Board B, il cui contenuto è pubblicamente leggibile.

Fase di shuffling

Una volta terminata la finestra temporale ***T1-T2*** il server Anonymizer Az_j preleva tutti i ciphertext all'interno della Bulletin Board B, li decifra utilizzando la propria chiave privata Sk_{Az_j}, applica una permutazione casuale e li invia al server successivo Az_{j-1}. Questo procedimento viene ripetuto per tutti i server Anonymizer fino ad arrivare a Az_1.

Formalmente:

- Per tutti i ciphertext $y_{j_1}(C_{j_1}), \dots, y_{j_n}(C_{j_n})$, inviati dagli n elettori alla Bulletin Board B, il server Anonymizer Az_j rimuove un livello di crittografia utilizzando la propria chiave privata SK_{Az_j}
    - $Dec\left(y_j(C_j), SK_{Az_j}\right) = v_j \cdot u_j^{-SK_{Az_j}} = y_{j-1}(C_{j-1})$
        - Dove $y_{j-1}(C_{j-1})$ è la cifratura del server Az_{j-1}
- Applica una permutazione casuale su tutti i ciphertext risultanti dal passaggio precedente
- Invia il risultato a Az_{j-1};
- Lo stesso procedimento viene ripetuto per ogni Az_i, fino ad ottenere, in output all'ultimo Anonymizer della Mix-net, i ciphertext $C_1, \dots, C_n$ in ordine casuale.

Successivamente, l'ultimo Anonymizer distribuisce casualmente questi n ciphertext tra i server Tallier $T_1, \dots, T_m$.

Ad ogni passaggio l' $Az\ i$-esimo deve dimostrare e rendere pubblico, in ZK, che il risultato dell'operazione effettuata è corretto.

- Ciò che l'$Az\ i$-esimo vuole mantenere segreto è la propria chiave privata Sk_{Az_i}
- Ciò che l'$Az\ i$-esimo vuole dimostrare è che la decifratura che effettua su $x_i(M_i)$ equivale a $x_{i-1}(M_{i-1})$
    - Dove sia $x_i(M_i)$ che $x_{i-1}(M_{i-1})$ sono ciphertext cifrati rispettivamente con le chiavi pubbliche PK_{AZ_i} e $PK_{AZ_{i-1}}$
- Deve dimostrare che $Dec\big(x_i(M_i), SK_{Az_i}\big)$ è una decifratura di $x_i(M_i)$ tale che:

$$Dec\big(x_i(M_i), SK_{Az_i}\big) = v_i \cdot u_i^{-SK_{Az_i}} = x_{i-1}(M_{i-1})$$

Fase di checking

Ogni server Tallier T_i, per ogni ciphertext ricevuto $C_i =< x_i(M_i), S_A(x_i(M_i)) >$, verifica che la firma $S_A(x_i(M_i))$ sia autentica, seguendo la stessa procedura effettuata dall'elettore in precedenza. Se questo controllo fallisce il server Tallier T_i verifica tutte le zero-knowledge proof pubblicate dagli Anonymizer.

Dopo di che, se da una di queste zero-knowledge proof emerge che un server Anonymizer non ha seguito il protocollo, questo viene eliminato, altrimenti, presupponendo che non siano stati riscontrati problemi nella fase di preparazione al voto, si deduce che l'errore si trovi nel ciphertext memorizzato da B. In entrambi i casi il voto viene escluso dalle fasi successive.

Fase di apertura e conteggio

Una volta esclusi tutti i voti non validi, i vari server Tallier ricostruiscono la chiave segreta SK_{S_c} :

- Tutti i Tallier $T_1, \dots, T_m$ condividono le m coppie (x_i, sh_i)
- Eseguono l'interpolazione di Lagrange : $SK_{S_c} = p(0) = \sum_{1 \le i \le m} p(x_i) \cdot d_i$
    - dove $d_i = \prod_{1 \le i \le m, j \ne i} \frac{j}{j-i}$

Se meno di m Tallier si mettono insieme non ottengono alcuna informazione su SK_{S_c}.

Ogni Tallier decifra la parte degli $x_1(M_1), \dots, x_n(M_n)$ ciphertext assegnatagli utilizzando SK_{S_c}:

- $M_i = v_i \cdot u_i^{-SK_{S_c}}$

Una volta decifrati i voti, i Tallier procedono con la fase di counting, eliminano le preferenze che non rappresentano uno dei candidati, inviano il risultato del conteggio al Central Server S_c e pubblicano la lista di tutti i voti ricevuti, specificando la motivazione qualora venissero scartati, nonché il conteggio risultante.

…	…	…
$\boldsymbol{S_A(x(M_i))}$	$\boldsymbol{x(M_i)}$	$\boldsymbol{M_i}$
…	…	…

Tabella 2, Lista pubblicata dal Central Server

Infine, S_C, durante la fase T2-T3, effettua un'unica ultima somma e pubblica il risultato dell'elezione.

WP3

COMPLETEZZA

Premesso che tutte le parti in causa siano oneste, ovvero seguano il protocollo descritto, allora tutti i voti validi sono conteggiati correttamente, non sussiste nessun coinvolgimento di J e il risultato finale della votazione è affidabile.

Diversamente, nel caso in cui ci siano comportamenti devianti dal protocollo, il sistema fornisce prove evidenti del comportamento disonesto riscontrato, fondamentali per chiamare in causa J.

INTEGRITÀ

- L' intruso malintenzionato può intervenire in diversi instanti della procedura di voto:
  - Durante la fase di preparazione al voto potrebbe modificare $b(x(M))$ oppure $S_A(b(x(M))$.
  - Durante le fasi di voto effettivo e di shuffling potrebbe modificare rispettivamente i ciphertext mantenuti dalla Bulletin Board B oppure quelli inviati ai server Tallier $T_1, \dots, T_m$.

  Tuttavia, le sue azioni non passeranno inosservate, poiché sia un elettore sia un server Tallier onesto potranno accorgersi di eventuali modifiche sulle preferenze espresse nel momento in cui verificano la firma del server Authenticator. In questo caso la proprietà I.1 è soddisfatta.
- L'autorità del sistema di voto corrotta potrebbe aiutare l'intruso malintenzionato nel far passare inosservate, durante la fase di checking, le modifiche dei ciphertext inviati ai server Tallier $T_1, \dots, T_m$. In questo caso la proprietà I.1 è a rischio, in quanto i Tallier potrebbero deviare dal protocollo non eseguendo la verifica sopra citata. Tuttavia, è da prendere in considerazione che questa collusione comporterebbe rischi alla reputazione di entrambi, visto che non necessariamente si fidano gli uni degli altri.
- L'autorità del sistema di voto astuta si trova in difficoltà dato che un eventuale conteggio scorretto verrà sicuramente rilevato con controlli sulle liste pubblicate dai server Authenticator e dai server Tallier. In questo modo la proprietà I.2 non viene violata.

PRIVACY

- La blind signature effettuata dall'elettore serve per fare in modo che gli Authenticator certifichino i voti senza poterne vedere il contenuto, nascondendo così la corrispondenza tra voto e identità dell'elettore durante la finestra temporale *T1-T2* a qualsiasi avversario interessato a violare l'anonimato.
- L'intruso impiccione non ha modo di ricavare la corrispondenza tra una preferenza espressa e l'identità di un elettore, grazie all'impiego di una Mixnet costituita dai server Tallier. Questo, assieme all'utilizzo delle blind signature, assicura che la proprietà P.1 sia soddisfatta.
- L' intruso pettegolo potrebbe compromettere la proprietà P.2 qualora avesse accesso alla chiave privata del Server Centrale e ad un sottoinsieme dei voti inviati ai server Tallier dall'ultimo server Anonymizer; tuttavia, in questo caso, riuscirebbe ad ottenere solo un conteggio parziale che non gli fornirebbe informazioni significative sui risultati finali dell'elezione. Al contrario, P.2 viene violata completamente se e soltanto se l'intruso pettegolo ottiene l'accesso a tutti i server Tallier e di conseguenza riesce a ricostruire la chiave segreta del Central Server, poiché, in questo caso può aprire e contare i voti anticipatamente. Tuttavia, l'utilizzo di molteplici server Tallier rende questo scenario meno probabile.

VIVEZZA

L'unico avversario che potrebbe minacciare la vivezza del sistema, quindi la proprietà V.1, è l'intruso malintenzionato, qualora decidesse di effettuare attacchi DoS o simili sui server coinvolti nella finestra temporale *T1-T2*, escludendo una parte dei voti dalla fase di conteggio. Il problema è stato arginato utilizzando il sistema distribuito costituito dai server Authenticator, ma potrebbe essere ulteriormente contenuto adottando la medesima soluzione anche per la Bulletin Board B.

AUTENTICITÀ

- La fase di preparazione al voto permette di tutelare l'integrità delle proprietà A.1, A.4 e A.5, contrastando l'elettore furbo, l'avversario catfish e l'intruso ladro. La firma del server Authenticator è necessaria per accertarsi che l'elettore non abbia già votato in precedenza. Sia l'utilizzo di credenziali di accesso sia il sistema di notifiche che comunica l'esito dei controlli del server Authenticator, invece, hanno permesso di scoraggiare eventuali furti d'identità. Infine, l'utilizzo di un e-token per gli elettori accerta l'idoneità al voto di quest'ultimi.
- Per garantire la pubblica verificabilità, e quindi le proprietà A.2 e A.3, è stato previsto che i server Authenticator e i server Tallier pubblichino le liste *Tabella 1* e *Tabella 2*. In questo modo le azioni dell'autorità del sistema di voto astuta o corrotta non passeranno inosservate.

INCOERCIBILITÀ

Sebbene non sia presente nessun tipo di ricevuta che attesti pubblicamente la preferenza espressa da un elettore, in quanto neanche attraverso le liste pubblicate dai vari server è possibile ricavare tale informazione, la proprietà IN.1 non è comunque soddisfatta, dato che il sistema non ha controllo sulle situazioni relative al mondo fisico. Possibili soluzioni per garantire l'incoercibilità del sistema sarebbero: il re-voting, ossia la possibilità per l'elettore di poter sovrascrivere la propria preferenza; evitare di rendere pubbliche liste di voti ricevuti (*Tabella 1*, *Tabella 2*). Tuttavia, queste soluzioni porterebbero ad una violazione delle proprietà di autenticità, pertanto è stato deciso di preferire la pubblica verificabilità piuttosto che tentare di risolvere problemi irrisolti anche nelle elezioni tradizionali.

EFFICIENZA

- Gli interventi di J sono stati per quanto possibile limitati solo ai casi in cui sussiste un'evidenza di imbrogli da parte di attori o avversari.
- Il sistema è stato concepito per essere quanto più user friendly possibile. È stato utilizzato un meccanismo di autenticazione tramite username e password piuttosto che uno basato su chiavi pubbliche e firme digitali, in quanto le password sono scelte dagli utenti stessi e più facili da memorizzare. Inoltre, il sistema prevede che l'elettore sia coinvolto solo in due circostanze, una per votare, obbligatoria, e un'altra di controllo nel momento in cui viene pubblicata dai server Tallier la lista dei voti conteggiati.
- Gli unici calcoli coinvolti per calcolare firme e crittografie sono esponenziazioni modulari, le quali sono operazioni efficienti ma non efficientissime, dato che richiedono comunque un certo effort; di conseguenza questo problema potrebbe essere arginato tramite l'approccio del Key/Data Encapsulation Mechanism. Inoltre, tutti i server del sistema di voto hanno a disposizione hardware più potente e sono implementati in modo tale da distribuire il carico computazionale tra diverse unità (es. ogni server Tallier decifra solo un sottoinsieme di tutti i voti ricevuti).
- Un possibile collo di bottiglia per le prestazioni potrebbe essere rappresentato dalla serie di cifrature a cascata eseguite per i server della Mixnet, in quanto ognuna di queste cifrature ha bisogno di una nuova stringa random. Per mitigare questo problema si potrebbero utilizzare tecniche come block cipher o stream chipher.

TRASPARENZA

Come già evidenziato nell'analisi delle proprietà di autenticità, il sistema garantisce la pubblica verificabilità attraverso la divulgazione di liste intermedie e finali relative a voti certificati, contati e scartati (*Tabella 1*, *Tabella 2*).

Server provider corrotto

Questo avversario potrebbe effettivamente minare molteplici proprietà del sistema. Tuttavia, è ragionevole assumere che sia le liste pubblicate che la reputazione della società che si occupa della fornitura dei server rappresentino un deterrente per questa tipologia di attacchi.

In questo caso si potrebbero presentare diversi situazioni, nelle quali tutti i server a disposizione potrebbero essere corrotti oppure, più realisticamente, solo un sottoinsieme di essi. Si riportano alcuni scenari ipotizzati:

- Server Authenticator e server Tallier in collusione per violare la privacy. In questo caso l'anonimato degli elettori viene comunque preservato, dato che, anche se insieme, non hanno le informazioni necessarie per poter ricondurre una preferenza all'elettore che l'ha espressa.
- Server Bulletin Board e server Tallier in collusione per scartare voti. In questo scenario, grazie alla pubblicazione della lista *Tabella 1* da parte dei server Authenticator è facile accorgersi di differenze sostanziali con la lista *Tabella 2* pubblicata dai server Tallier.
- Un Central Server corrotto in combutta con uno o più server Tallier fornisce l'intera chiave segreta e non una singola share a questi ultimi in modo da permettergli di aprire i voti senza il contributo degli altri. Tuttavia, questo attacco non avrebbe conseguenze gravi sul corretto svolgimento delle elezioni, dato che i server Tallier ricevono i voti cifrati solo al termine della finestra temporale *T1-T2* e di conseguenza il conteggio dei voti non influenzerebbe la votazione.
- Una soluzione per arginare questo attacco potrebbe essere fare in modo che il Central Server calcoli la chiave segreta e distribuisca le share all'inizio della finestra temporale *T2-T3*, anziché nella fase di preparazione della struttura, in modo che il conteggio dei voti non influenzi la votazione
- Server Authenticator firmano e votano a nome di astenuti. Per risolvere questo attacco una possibile miglioria del protocollo proposto potrebbe essere quella di utilizzare la tecnica del (n,n) threshold encryption anche per i server Authenticator, in modo tale che un elettore, per essere autorizzato al voto, debba raccogliere le firme di tutti i server Authenticator.

Imbroglioni in collusione

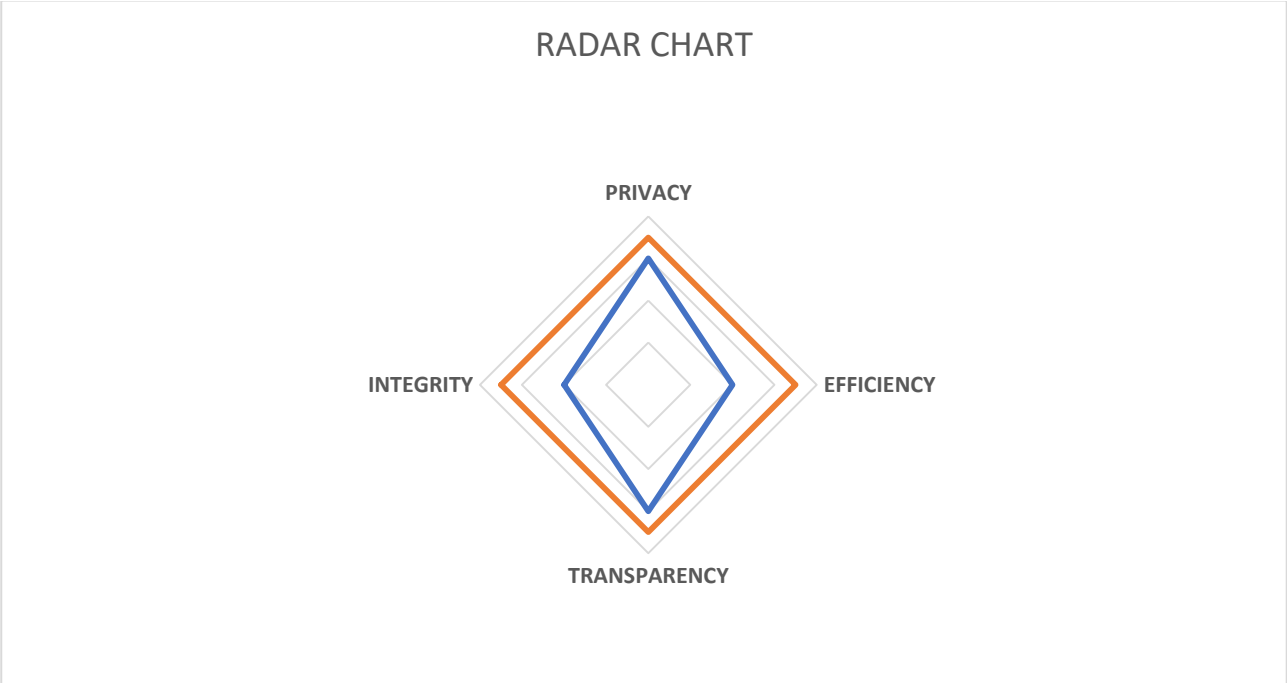
Risulta ovvio che diversi avversari interessati a violare delle priorità del sistema possano colludere. Tuttavia, questo tipo di attacchi combinati potrebbe esporre rischi alla loro reputazione ed inoltre risulta poco probabile mantenere una coalizione del genere.

Gestori d'identità digitale come avversari

Non abbiamo supposto che i gestori d'identità digitale possano rappresentare una minaccia per il sistema di voto elettronico presentato dato che questo sarebbe altamente minatorio per la loro reputazione. Infatti, è chiaro che se un solo gestore di identità dimostra di avere dei buchi, tutto il sistema viene rimesso in discussione con gravi danni di reputazione.

GRAFICI

Figura 2, Radar chart



STRUMENTO UTILIZZATO IN WP2	**USATO PER**
SHA256	Memorizzare con sicurezza le password degli elettori
Cifratura ElGamal	Cifrare i voti degli elettori
Blind signature	Permettere che gli Authenticator certifichino i voti senza poterne vedere il contenuto
Decryption Mixnet	Eliminare collegamento elettore-preferenza espressa
ZPK	Verificare che i server della Mixnet decifrino correttamente

Threshold Encryption e secret sharing	Evitare che i voti vengano aperti e contati anticipatamente

Tabella 3, Strumenti utilizzati

WP4

Classi

Il sistema progettato è stato realizzato attraverso le seguenti classi java:

- **Authenticator.java**: rappresenta l'entità del server Authenticator, attraverso i seguenti metodi:
  - *addElettore(…)* : permette di registrare le credenziali di un nuovo elettore.
    - *shaPasswordGenerator(…)* : consente di effettuare l'operazione $SHA256(password||rand)$.
  - *isElettoreLogin(…)* : consente di verificare se l'elettore intento ad effettuare il login è presente all'interno dei dati del server.
  - *signElettore(…)* : permette di firmare il blind inviato dall'elettore.
    - *isElettoreSigned(…)* : consente di verificare che l'Authenticator non abbia già firmato una preferenza a nome di uno specifico elettore.

  Inoltre, sono presenti le seguenti classi innestate:
  - InfoElettore : rappresenta le informazioni che devono essere memorizzate relativamente ad un determinato elettore.
  - PublicInfoElettore : rappresenta le informazioni che devono essere pubblicate dal server Authenticator relativamente ad un determinato elettore al termine della finestra temporale *T1-T2*.
- **CentralServer.java**: rappresenta l'entità del server CentralServer, attraverso i seguenti metodi:
  - *buildShares(…)* : permette di costruire le share relative alla chiave segreta che dovranno essere distribuite tra i vari server Tallier.
  - *finalCount(…)* : effettua il conteggio finale sui risultati intermedi forniti dai server Tallier.
- **BulletinBoard.java**: rappresenta una semplice astrazione delle funzionalità che dovrebbe svolgere il server Bulletin Board.
- **Anonymizer.java**: rappresenta un'astrazione delle funzionalità che dovrebbe svolgere un server della Mixnet. Presenta il seguente metodo:
  - *decypher(…)* : effettua, in seguito ad una permutazione casuale, la decifratura dei ciphertext prelevati dalla Bulletin Board o dal server della Mixnet precedente.
- **Tallier.java**: : rappresenta l'entità del server Tallier, attraverso i seguenti metodi:
  - *reconstructSecret(…)* : consente, a partire dalle share ricevute dagli altri server Tallier, di ricostruire la chiave segreta del CentralServer, implementando l'interpolazione di Lagrange.
  - *decPreferences(…)* : si occupa della decifratura della porzione di preferenze assegnategli.
    - *verifySignedMessage(…)* : permette di verificare che la firma associata al voto ricevuto sia autentica.
  - *countPreferences(…)* : effettua il conteggio della porzione di preferenze assegnategli.

  Inoltre, è presente la seguente classe innestata:
  - publicInfoVoto : rappresenta le informazioni che devono essere pubblicate dal server Tallier relativamente ad un determinato elettore al termine della fase *T2-T3*.

- **Elettore.java**: rappresenta il singolo elettore avente diritto al voto. Contiene i seguenti metodi:
    - *blindPreference(…)* : permette di cifrare la preferenza espressa dall'elettore con la chiave pubblica del CentralServer, nonché di applicare il blind factor al ciphertext risultante.
    - *unblindPreference(…)* : permette di rimuovere il blind factor sulla preferenza firmata dal server Authenticator
        - *verifySignedBlind(…)* : consente di verificare che la firma ricevuta dall'Authenticator sia autentica.
    - *mixNetCipher(…)* : implementa la cifratura a cascata attraverso le chiavi pubbliche dei server Anonymizer.
- **InfoVoto.java**: rappresenta la coppia $< x(M), S_A(x(M)) >$.
- **ElGamalCT.java**: implementazione della struttura di un ciphertext ElGamal.
- **ElGamalSK.java**: implementazione della struttura di una chiave segreta ElGamal.
- **ElGamalPK.java**: implementazione della struttura di una chiave pubblica ElGamal.
- **ElGamalUtils.java**: contiene i metodi di cifratura e decifratura secondo lo schema ElGamal.
- **KeyUtils.java**: contiene i metodi di generazione delle chiavi per gli schemi ElGamal e RSA.

MAIN.JAVA

All'interno della main class Main.java è presente una simulazione del sistema progettato.

La simulazione parte con la registrazione di quattro elettori all'interno del server Authenticator e con l'inizializzazione dei server coinvolti, con relativa generazione delle loro chiavi.

In seguito, vengono espresse due preferenze a nome di due elettori registrati nel server Authenticator e vengono richieste a linea di comando delle credenziali di accesso. Qualora le credenziali fornite coincidessero con una di quelle presenti nel server, viene allora richiesto all'utente di esprimere la propria preferenza.

Successivamente, tale preferenza viene cifrata con la chiave pubblica del Central Server, nascosta con un blind factor e inviata al server Authenticator per essere firmata. Se la firma ricevuta risulta corretta, viene rimosso il blind factor e viene eseguita la sequenza di cifrature attraverso le chiavi pubbliche dei server Anonymizer. Il ciphertext risultante dalle cifrature a cascata viene inviato alla Bulletin Board.

Una volta raccolte le tre preferenze, l'Authenticator stampa su standard output la propria lista di informazioni e si può passare alla fase di shuffling.

A questo punto, i server Anonymizer effettuano una permutazione casuale e le decifrature a cascata sui ciphertext prelevati dalla Bulletin Board, per poi distribuirli casualmente tra i vari server Tallier. Quest'ultimi ricostruiscono la chiave segreta dal Central Server e decifrano le preferenze ricevute. Infine, ognuno effettua il conteggio sui voti che gli sono stati assegnati.

I risultati dei conteggi intermedi vengono stampati su standard output e inviati successivamente al Central Server, che si occupa di effettuare l'ultimo conteggio e di stampare il risultato finale dell'elezione.

Le stampe su standard output citate simulano la pubblicazione delle suddette informazioni.

Per semplicità la simulazione prevede solo due server Anonymizer e due server Tallier e non è stato implementato il meccanismo di comunicazione su TLS.

UML

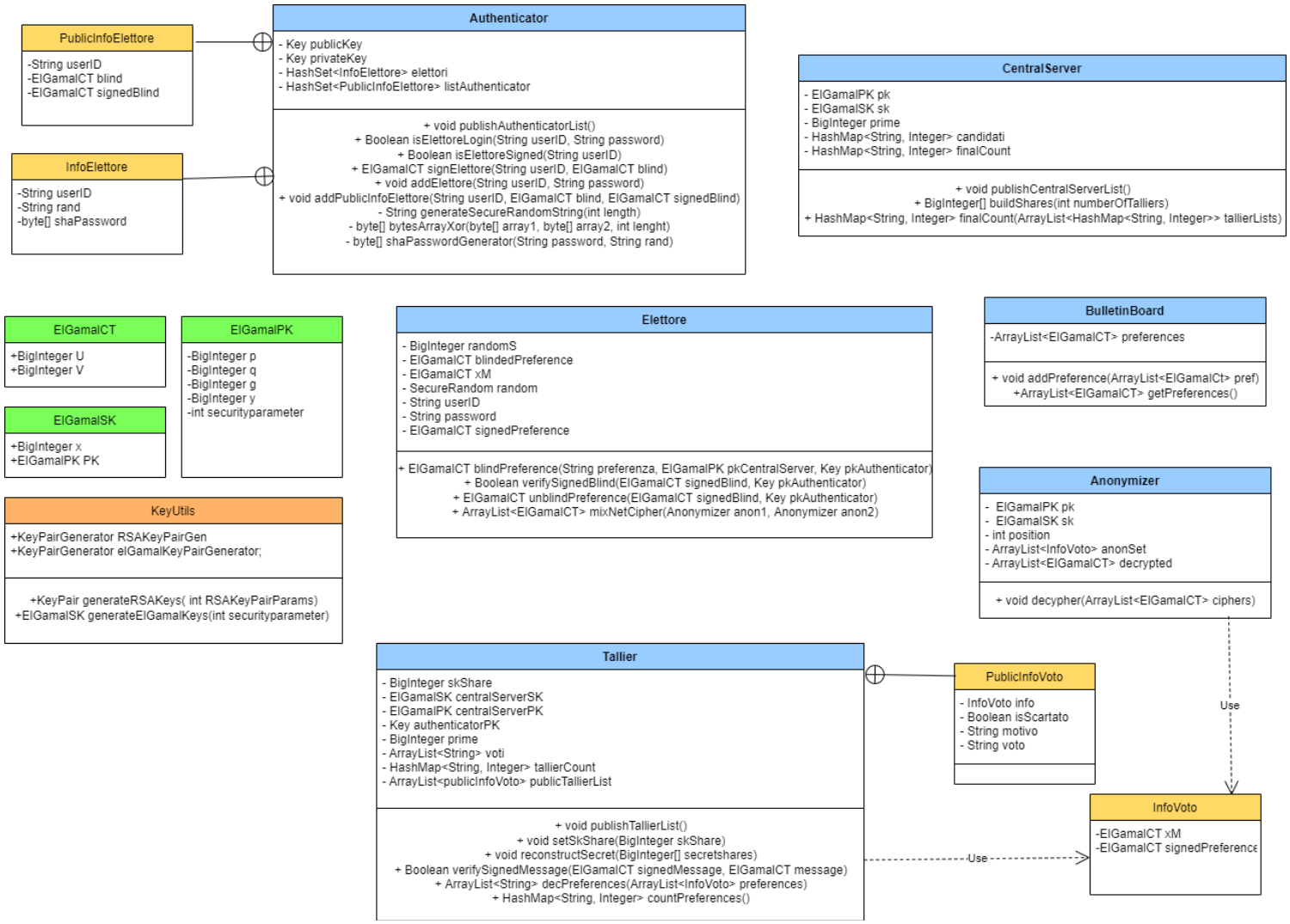


Figura 3, Diagramma UML

CODICE JAVA

Anonymizer

```
package default_package;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Collections;

public class Anonymizer {

    private final ElGamalPK pk;
    private final ElGamalSK sk;
    private int position;
    private ArrayList<InfoVoto> anonSet;
    private ArrayList<ElGamalCT> decrypted;

    public Anonymizer(int position, int secureparameter) {
        this.sk = KeysUtils.generateElGamalKeys(secureparameter);
        this.pk = sk.getPK();
        this.position = position;
        this.anonSet = new ArrayList<>();
        this.decrypted = new ArrayList<>();
    }
```

```
public ElGamalPK getPk() {
    return pk;
}

public ArrayList<ElGamalCT> getDecrypted() {
    return decrypted;
}

public ArrayList<InfoVoto> getAnonSet() {
    return anonSet;
}

public void decypher(ArrayList<ElGamalCT> ciphers) {
    ArrayList<ArrayList<ElGamalCT>> blocks = new ArrayList<>();
    if (position == 2) {
        for (int i = 0; i < ciphers.size(); i += 8) {
            ArrayList<ElGamalCT> tempList = new ArrayList<>();
            tempList.add(ciphers.get(i));
            tempList.add(ciphers.get(i + 1));
            tempList.add(ciphers.get(i + 2));
            tempList.add(ciphers.get(i + 3));
            tempList.add(ciphers.get(i + 4));
            tempList.add(ciphers.get(i + 5));
            tempList.add(ciphers.get(i + 6));
            tempList.add(ciphers.get(i + 7));
            blocks.add(tempList);
        }

        Collections.shuffle(blocks);

        for (ArrayList<ElGamalCT> block : blocks) {
            ElGamalCT newMessageUU = block.get(0);
            ElGamalCT newMessageUV = block.get(1);
            ElGamalCT newMessageVU = block.get(2);
            ElGamalCT newMessageVV = block.get(3);
            ElGamalCT newSignedUU = block.get(4);
            ElGamalCT newSignedUV = block.get(5);
            ElGamalCT newSignedVU = block.get(6);
            ElGamalCT newSignedVV = block.get(7);

            //DEC
            BigInteger decMessageUU = ElGamalUtils.Decrypt(newMessageUU, sk);
            BigInteger decMessageUV = ElGamalUtils.Decrypt(newMessageUV, sk);
            BigInteger decMessageVU = ElGamalUtils.Decrypt(newMessageVU, sk);
            BigInteger decMessageVV = ElGamalUtils.Decrypt(newMessageVV, sk);

            BigInteger decSignedUU = ElGamalUtils.Decrypt(newSignedUU, sk);
            BigInteger decSignedUV = ElGamalUtils.Decrypt(newSignedUV, sk);
            BigInteger decSignedVU = ElGamalUtils.Decrypt(newSignedVU, sk);
            BigInteger decSignedVV = ElGamalUtils.Decrypt(newSignedVV, sk);

            //RECONSTRUCT
            decrypted.add(new ElGamalCT(decMessageUV, decMessageUU)); //newMessageUf
            decrypted.add(new ElGamalCT(decMessageVV, decMessageVU)); //newMessageVf
            decrypted.add(new ElGamalCT(decSignedUV, decSignedUU)); //newSignedUf
            decrypted.add(new ElGamalCT(decSignedVV, decSignedVU)); //newSignedVf
        }

    } else if (position == 1) {

        for (int i = 0; i < ciphers.size(); i += 4) {
            ElGamalCT newMessageUf = ciphers.get(i);
            ElGamalCT newMessageVf = ciphers.get(i + 1);
            ElGamalCT newSignedUf = ciphers.get(i + 2);
            ElGamalCT newSignedVf = ciphers.get(i + 3);

            //DEC
            BigInteger decMessageU = ElGamalUtils.Decrypt(newMessageUf, sk);
            BigInteger decMessageV = ElGamalUtils.Decrypt(newMessageVf, sk);
            BigInteger decSignedU = ElGamalUtils.Decrypt(newSignedUf, sk);
            BigInteger decSignedV = ElGamalUtils.Decrypt(newSignedVf, sk);

            ElGamalCT messageFinal = new ElGamalCT(decMessageV, decMessageU);
            ElGamalCT signedFinal = new ElGamalCT(decSignedV, decSignedU);

            anonSet.add(new InfoVoto(messageFinal, signedFinal));
        }
        Collections.shuffle(anonSet);
```

```
        }

    }

}
```

Authenticator

```
package default_package;

import default_package.exceptions.ElettoreNotLogged;
import default_package.exceptions.ElettoreAlreadySigned;
import static default_package.KeysUtils.generateRSAKeys;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.security.KeyPair;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Objects;
import org.bouncycastle.jcajce.provider.asymmetric.rsa.BCRSAPrivateKey;

public class Authenticator {

    private final Key publicKey;
    private final Key privateKey;
    private HashSet<InfoElettore> elettori;
    private HashSet<PublicInfoElettore> listAuthenticator;
    private static MessageDigest digest;
    private static String chrs = "0123456789abcdefghijklmnopqrstuvwxyz-_ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    public Authenticator() {
        this.elettori = new HashSet();
        this.listAuthenticator = new HashSet();
        KeyPair kp = generateRSAKeys(256);
        this.publicKey = kp.getPublic();
        this.privateKey = kp.getPrivate();
        try {
            digest = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException ex) {
            System.err.println(ex.getMessage());
        }
    }

    public void publishAuthenticatorList(){
        System.out.println("========= AUTHENTICATOR LIST =========");
        for (PublicInfoElettore info : listAuthenticator){
            System.out.println(info.getUserID() + " | " + info.getBlind() + " | " +
info.getSignedBlind());
        }
    }

    public void printElettori() {
        for (InfoElettore e : elettori) {
            System.out.println(e);
        }
    }

    public Boolean isElettoreLogin(String userID, String password) throws ElettoreNotLogged {
        for (InfoElettore e : elettori) {
            if (e.getUserID().compareTo(userID) == 0 && password.length() == e.getRand().length()) {
                String rand = e.getRand();
                byte[] sha = e.getShaPassword();

                byte[] resultSha = shaPasswordGenerator(password, rand);

                return Arrays.toString(sha).compareTo(Arrays.toString(resultSha)) == 0;
            }
        }
        throw new ElettoreNotLogged("Elettore non presente nel database!");
    }
```

```
    public Boolean isElettoreSigned(String userID) {
        for (PublicInfoElettore e : listAuthenticator) {
            if (e.getUserID().compareTo(userID) == 0) {
                return true;
            }
        }
        return false;
    }

    public ElGamalCT signElettore(String userID, ElGamalCT blind) throws ElettoreAlreadySigned {
        if (!isElettoreSigned(userID)) {
            BCRSAPrivateKey privKey = (BCRSAPrivateKey) this.privateKey;
            BigInteger modulus = privKey.getModulus();
            BigInteger d = privKey.getPrivateExponent();

            BigInteger U = blind.U;
            BigInteger V = blind.V;
            BigInteger signedU = U.modPow(d, modulus);
            BigInteger signedV = V.modPow(d, modulus);

            ElGamalCT signedBlind = new ElGamalCT(signedV, signedU);

            listAuthenticator.add(new PublicInfoElettore(userID, blind, signedBlind));

            return signedBlind;
        } else {
            throw new ElettoreAlreadySigned("Elettore already signed!");
        }

    }

    private String generateSecureRandomString(int lenght) {
        SecureRandom r = null;

        try {
            r = SecureRandom.getInstanceStrong();
        } catch (NoSuchAlgorithmException ex) {
            System.err.println(ex.getMessage());
        }

        String randomString = r.ints(lenght, 0, chrs.length()).mapToObj(i -> chrs.charAt(i))
                .collect(StringBuilder::new, StringBuilder::append,
StringBuilder::append).toString();
        return randomString;
    }

    private byte[] bytesArrayXor(byte[] array1, byte[] array2, int lenght) {
        byte[] xor = new byte[lenght];
        for (int i = 0; i < lenght; i++) {
            xor[i] = (byte) (array1[i] ^ array2[i]);
        }
        return xor;
    }

    private byte[] shaPasswordGenerator(String password, String rand) {
        byte[] passwordBytes = password.getBytes(StandardCharsets.UTF_8);
        byte[] randBytes = rand.getBytes(StandardCharsets.UTF_8);

        byte[] xor = bytesArrayXor(passwordBytes, randBytes, password.length());

        return digest.digest(xor);
    }

    public void addElettore(String userID, String password) {
        String rand = generateSecureRandomString(password.length());
        byte[] shaPassword = shaPasswordGenerator(password, rand);

        elettori.add(new InfoElettore(userID, rand, shaPassword));
    }

    public void addPublicInfoElettore(String userID, ElGamalCT blind, ElGamalCT signedBlind) {
        listAuthenticator.add(new PublicInfoElettore(userID, blind, signedBlind));
    }

    public Key getPublicKey() {
        return publicKey;
    }
```

```
    private class InfoElettore {

        private String userID;
        private String rand;
        private byte[] shaPassword;

        public InfoElettore(String userID, String rand, byte[] shaPassword) {
            this.userID = userID;
            this.rand = rand;
            this.shaPassword = shaPassword;
        }

        public String getUserID() {
            return userID;
        }

        public void setUserID(String userID) {
            this.userID = userID;
        }

        public String getRand() {
            return rand;
        }

        public void setRand(String rand) {
            this.rand = rand;
        }

        public byte[] getShaPassword() {
            return shaPassword;
        }

        public void setShaPassword(byte[] shaPassword) {
            this.shaPassword = shaPassword;
        }

        @Override
        public int hashCode() {
            int hash = 7;
            hash = 59 * hash + Objects.hashCode(this.userID);
            return hash;
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj) {
                return true;
            }
            if (obj == null) {
                return false;
            }
            if (getClass() != obj.getClass()) {
                return false;
            }
            final InfoElettore other = (InfoElettore) obj;
            return Objects.equals(this.userID, other.userID);
        }

        @Override
        public String toString() {
            return "InfoElettore{" + "userID=" + userID + ", rand=" + rand + ", shaPassword=" +
shaPassword + '}';
        }

    }

    private class PublicInfoElettore {

        private String userID;
        private ElGamalCT blind;
        private ElGamalCT signedBlind;

        public PublicInfoElettore(String userID, ElGamalCT blind, ElGamalCT signedBlind) {
            this.userID = userID;
            this.blind = blind;
            this.signedBlind = signedBlind;
        }
```

```
        public String getUserID() {
            return userID;
        }

        public void setUserID(String userID) {
            this.userID = userID;
        }

        public ElGamalCT getBlind() {
            return blind;
        }

        public void setBlind(ElGamalCT blind) {
            this.blind = blind;
        }

        public ElGamalCT getSignedBlind() {
            return signedBlind;
        }

        public void setSignedBlind(ElGamalCT signedBlind) {
            this.signedBlind = signedBlind;
        }

        @Override
        public int hashCode() {
            int hash = 7;
            hash = 97 * hash + Objects.hashCode(this.userID);
            return hash;
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj) {
                return true;
            }
            if (obj == null) {
                return false;
            }
            if (getClass() != obj.getClass()) {
                return false;
            }
            final PublicInfoElettore other = (PublicInfoElettore) obj;
            return Objects.equals(this.userID, other.userID);
        }

        @Override
        public String toString() {
            return "ListAuthenticator{" + "userID=" + userID + ", \nblind=" + blind + ",
\nsignedBlind=" + signedBlind + '}';
        }

    }
}
```

BulletinBoard

```
package default_package;

import java.util.ArrayList;

public class BulletinBoard {

    public ArrayList<ElGamalCT> preferences;

    public BulletinBoard() {
        this.preferences = new ArrayList<>();
    }

    public void addPreference(ArrayList<ElGamalCT> pref) {
        preferences.addAll(pref);
    }

    public ArrayList<ElGamalCT> getPreferences() {
        return preferences;
    }

}
```

CentralServer

```
package default_package;

import java.util.HashMap;
import static default_package.KeysUtils.generateElGamalKeys;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.security.Security;
import java.util.ArrayList;

public class CentralServer {

    private final ElGamalPK pk;
    private final ElGamalSK sk;
    private HashMap<String, Integer> finalCount;
    private SecureRandom sc;
    private BigInteger prime;
    private HashMap<String, Integer> candidati;

    public CentralServer() {
        Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
        this.finalCount = new HashMap<>();
        this.sk = generateElGamalKeys(128);
        this.pk = this.sk.getPK();
        this.sc = new SecureRandom();
        this.candidati = new HashMap<>();
        initCandidati();
    }

    public void initCandidati() {
        candidati.put("Carpentieri", 0);
        candidati.put("Casaburi", 0);
        candidati.put("Falcone", 0);
        candidati.put("Ferraioli", 0);
        candidati.put("Greco", 0);
    }

    public void publishCentralServerList(){
        System.out.println("========= CENTRAL SERVER LIST =========");
        for (String s : candidati.keySet()) {
            System.out.println(s + " : " + candidati.get(s));
        }
    }

    public ElGamalPK getPk() {
        return pk;
    }

    public BigInteger getPrime() {
        return prime;
    }

    public BigInteger[] buildShares(int numberOfTalliers) {
        BigInteger[] secretshares = new BigInteger[numberOfTalliers];
        BigInteger elGamalSkInteger = this.sk.getX();
        this.prime = new BigInteger(elGamalSkInteger.bitLength() + 1, 256, sc);
        BigInteger[] coeff = new BigInteger[numberOfTalliers];
        coeff[0] = elGamalSkInteger;

        for (int i = 1; i < numberOfTalliers; i++) {
            BigInteger r;
            while (true) {
                r = new BigInteger(prime.bitLength(), sc);
                if (r.compareTo(BigInteger.ZERO) > 0 && r.compareTo(prime) < 0) {
                    break;
                }
            }
            coeff[i] = r;
        }

        for (int x = 1; x <= numberOfTalliers; x++) {
            BigInteger accum = elGamalSkInteger;

            for (int exp = 1; exp < numberOfTalliers; exp++) {
                accum =
accum.add(coeff[exp].multiply(BigInteger.valueOf(x).pow(exp).mod(prime))).mod(prime);
            }
```

```
            secretshares[x - 1] = accum;
        }

        return secretshares;
    }

    public HashMap<String, Integer> finalCount(ArrayList<HashMap<String, Integer>> tallierLists) {

        for (HashMap<String, Integer> list : tallierLists) {
            for (String k : list.keySet()) {
                candidati.put(k, list.get(k) + candidati.get(k));
            }
        }

        return candidati;
    }

}
```

ElGamalCT

```
package default_package;

import java.math.BigInteger;

public class ElGamalCT {

    BigInteger V, U;

    public ElGamalCT(BigInteger V, BigInteger U) {
        this.V = V;
        this.U = U;

    }

    public ElGamalCT(ElGamalCT CT) {
        this.V = CT.V;
        this.U = CT.U;

    }

    @Override
    public String toString() {
        return "{" + "V=" + V + ", U=" + U + '}';
    }

}
```

ElGamalPK

```
package default_package;

import java.math.BigInteger;

public class ElGamalPK {

    private final BigInteger g, y, p, q; // description of the group and public-key y=g^s
    int securityparameter; // security parameter

    public ElGamalPK(BigInteger p, BigInteger q, BigInteger g, BigInteger y, int securityparameter)
{
        this.p = p;
        this.q = q;
        this.g = g;
        this.y = y;
        this.securityparameter = securityparameter;

    }

    public BigInteger getG() {
        return g;
    }

    public BigInteger getY() {
        return y;
    }
```

```
    public BigInteger getP() {
        return p;
    }

    public BigInteger getQ() {
        return q;
    }

    public int getSecurityparameter() {
        return securityparameter;
    }

}
```

ElGamalSK

```
package default_package;


import java.math.BigInteger;
//structures for ElGamal secret-key
//Vincenzo Iovino

public class ElGamalSK { // Secret-key of El Gamal

    private final BigInteger x;
    // x is random BigInteger from 1 to q where q is the order of g (g is in the PK)

    private final ElGamalPK PK; // PK of El Gamal

    public ElGamalSK(BigInteger x, ElGamalPK PK) {
        this.x = x;
        this.PK = PK;

    }

    public BigInteger getX() {
        return x;
    }

    public ElGamalPK getPK() {
        return PK;
    }

}
```

ElGamalUtils

```
package default_package;

import java.math.BigInteger;
import java.security.SecureRandom;

public class ElGamalUtils {

    public static ElGamalCT Encrypt(ElGamalPK PK, BigInteger M) {
        SecureRandom sc = new SecureRandom(); // create a secure random source

        BigInteger r = new BigInteger(PK.securityparameter, sc); // choose random r of lenght
security parameter
        // V=[y^r*M mod p, g^r mod p].

        BigInteger V = M.multiply(PK.getY().modPow(r, PK.getP())); // V=M*(y^r mod p)
        V = V.mod(PK.getP()); // V=V mod p
        BigInteger U = PK.getG().modPow(r, PK.getP());  // U=g^r mod p
        return new ElGamalCT(V, U);   // return CT=(V,U)

    }
```

```
    public static BigInteger Decrypt(ElGamalCT CT, ElGamalSK SK) {
        // V=[V,U]=[y^r*M mod p, g^r mod p].
        // y=g^x mod p

        BigInteger tmp = CT.U.modPow(SK.getX(), SK.getPK().getP());  // tmp=U^x mod p
        tmp = tmp.modInverse(SK.getPK().getP());
        // if tmp and p are BigInteger tmp.modInverse(p) is the integer x s.t.
        // tmp*x=1 mod p
        // thus tmp=U^{-x}=g^{-rx} mod p =y^{-r}

        BigInteger M = tmp.multiply(CT.V).mod(SK.getPK().getP()); // M=tmp*V mod p
        return M;

    }

}
```

Elettore

```
package default_package;

import default_package.exceptions.AuthenticatorSignatureFailure;
import java.math.BigInteger;
import java.security.Key;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.bouncycastle.jcajce.provider.asymmetric.rsa.BCRSAPublicKey;

public class Elettore {

    private BigInteger randomS;
    private ElGamalCT blindedPreference;
    private ElGamalCT xM;
    private SecureRandom random;

    private String userID;
    private String password;

    private ElGamalCT signedPreference;

    public Elettore(String userID, String password) {
        try {
            this.random = SecureRandom.getInstanceStrong();
        } catch (Exception ex) {
            Logger.getLogger(Elettore.class.getName()).log(Level.SEVERE, null, ex);
        }
        this.userID = userID;
        this.password = password;
    }

    public String getUserID() {
        return userID;
    }

    public String getPassword() {
        return password;
    }

    public ElGamalCT getxM() {
        return xM;
    }

    public ElGamalCT getSignedPreference() {
        return signedPreference;
    }

    public ElGamalCT blindPreference(String preferenza, ElGamalPK pkCentralServer, Key
pkAuthenticator) {
        BigInteger plainText = new BigInteger(preferenza.getBytes());

        //ELGAMAL
        ElGamalCT CT = ElGamalUtils.Encrypt(pkCentralServer, plainText);

        //BLIND
        BCRSAPublicKey pubKey = (BCRSAPublicKey) pkAuthenticator;
        BigInteger modulus = pubKey.getModulus();
        BigInteger e = pubKey.getPublicExponent();
```

```
        BigInteger s = new BigInteger(modulus.bitLength(), random);
        while (s.compareTo(modulus) >= 0) {
            s = new BigInteger(modulus.bitLength(), random);
        }

        this.randomS = s;

        this.xM = CT;

        BigInteger newU = (s.modPow(e, modulus).multiply(xM.U)).mod(modulus);
        BigInteger newV = (s.modPow(e, modulus).multiply(xM.V)).mod(modulus);

        ElGamalCT newCT = new ElGamalCT(newV, newU);

        this.blindedPreference = newCT;

        return blindedPreference;
    }

    public Boolean verifySignedBlind(ElGamalCT signedBlind, Key pkAuthenticator) {
        BCRSAPublicKey pubKey = (BCRSAPublicKey) pkAuthenticator;
        BigInteger modulus = pubKey.getModulus();
        BigInteger e = pubKey.getPublicExponent();

        BigInteger U = signedBlind.U;
        BigInteger V = signedBlind.V;
        BigInteger resultU = U.modPow(e, modulus);
        BigInteger resultV = V.modPow(e, modulus);

        ElGamalCT newCT = new ElGamalCT(resultV, resultU);

        return blindedPreference.U.compareTo(newCT.U) == 0 && blindedPreference.V.compareTo(newCT.V)
== 0;
    }

    public ElGamalCT unblindPreference(ElGamalCT signedBlind, Key pkAuthenticator) throws
AuthenticatorSignatureFailure {
        if (!verifySignedBlind(signedBlind, pkAuthenticator)) {
            throw new AuthenticatorSignatureFailure("The Authenticator' signature isn't valid!");
        }
        BCRSAPublicKey pubKey = (BCRSAPublicKey) pkAuthenticator;
        BigInteger modulus = pubKey.getModulus();
        BigInteger signedPreferenceU =
randomS.modInverse(modulus).multiply(signedBlind.U).mod(modulus);
        BigInteger signedPreferenceV =
randomS.modInverse(modulus).multiply(signedBlind.V).mod(modulus);

        this.signedPreference = new ElGamalCT(signedPreferenceV, signedPreferenceU);

        return signedPreference;
    }

    public ArrayList<ElGamalCT> mixNetCipher(Anonymizer anon1, Anonymizer anon2) {

        ArrayList<ElGamalCT> result = new ArrayList<>();

        ElGamalCT newMessageU = ElGamalUtils.Encrypt(anon1.getPk(), xM.U);
        ElGamalCT newMessageV = ElGamalUtils.Encrypt(anon1.getPk(), xM.V);
        ElGamalCT newSignedU = ElGamalUtils.Encrypt(anon1.getPk(), signedPreference.U);
        ElGamalCT newSignedV = ElGamalUtils.Encrypt(anon1.getPk(), signedPreference.V);

        //U MESSAGE
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newMessageU.U)); //newMessageUU
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newMessageU.V)); //newMessageUV
        //V MESSAGE
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newMessageV.U)); //newMessageVU
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newMessageV.V)); //newMessageVV

        //U SIGNED
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newSignedU.U)); //newSignedUU
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newSignedU.V)); //newSignedUV
        //V SIGNED
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newSignedV.U)); //newSignedVU
        result.add(ElGamalUtils.Encrypt(anon2.getPk(), newSignedV.V)); //newSignedVV

        return result;
```

```
    }

}
```

InfoVoto

```
package default_package;

import java.util.Objects;

public class InfoVoto {

    private ElGamalCT xM;
    private ElGamalCT signedPreference;

    public InfoVoto(ElGamalCT xM, ElGamalCT signedPreference) {
        this.xM = xM;
        this.signedPreference = signedPreference;
    }

    public ElGamalCT getxM() {
        return xM;
    }

    public ElGamalCT getSignedPreference() {
        return signedPreference;
    }

    public void setxM(ElGamalCT xM) {
        this.xM = xM;
    }

    public void setSignedPreference(ElGamalCT signedPreference) {
        this.signedPreference = signedPreference;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 97 * hash + Objects.hashCode(this.signedPreference);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final InfoVoto other = (InfoVoto) obj;
        return Objects.equals(this.signedPreference, other.signedPreference);
    }

    @Override
    public String toString() {
        return "InfoVoto{" + "xM=" + xM + ", signedPreference=" + signedPreference + '}';
    }

}
```

KeyUtils

```
package default_package;

import java.math.BigInteger;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.security.Security;
import java.util.logging.Level;
import java.util.logging.Logger;

public class KeysUtils {

    public static KeyPairGenerator RSAKeyPairGen;
    public static KeyPairGenerator elGamalKeyPairGenerator;

    public static KeyPair generateRSAKeys( int RSAKeyPairParams) {
        Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
        KeyPair rsaKeys = null;
        try {
            RSAKeyPairGen = KeyPairGenerator.getInstance("RSA", "BC");
            RSAKeyPairGen.initialize(RSAKeyPairParams);
            rsaKeys = RSAKeyPairGen.generateKeyPair();
        } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(KeysUtils.class.getName()).log(Level.SEVERE, null, ex);
        } catch (NoSuchProviderException ex) {
            Logger.getLogger(KeysUtils.class.getName()).log(Level.SEVERE, null, ex);
        }

        return rsaKeys;
    }

    public static ElGamalSK generateElGamalKeys(int securityparameter) {
        BigInteger p, q, g, y;

        SecureRandom sc = new SecureRandom(); // create a secure random source

        while (true) {
            q = BigInteger.probablePrime(securityparameter, sc);

            // method probablePrime returns a prime number of length securityparameter
            // using sc as random source
            p = q.multiply(BigInteger.TWO);
            p = p.add(BigInteger.ONE);  // p=2q+1

            if (p.isProbablePrime(50) == true) {
                break;      // returns an integer that is prime with prob. 1-2^-50
            }
        }
        // henceforth we have that p and q are both prime numbers and p=2q+1
        // Subgroups of Zp* have order 2,q,2q
        g = new BigInteger("4"); // 4 is quadratic residue so it generates a group of order q
        // g is a generator of the subgroup the QR modulo p
        // in particular g generates q elements where q is prime

        BigInteger x = new BigInteger(securityparameter, sc); // x is the secret-key
        y = g.modPow(x, p); // y=g^x mod p

        ElGamalPK PK = new ElGamalPK(p, q, g, y, securityparameter);

        return new ElGamalSK(x, PK);

    }

}
```

Tallier

```
package default_package;

import java.math.BigInteger;
import java.security.Key;
import java.security.Security;
import java.util.ArrayList;
import java.util.HashMap;
import org.bouncycastle.jcajce.provider.asymmetric.rsa.BCRSAPublicKey;

public class Tallier {

    private BigInteger skShare;
    private ElGamalSK centralServerSK;
    private ElGamalPK centralServerPK;
    private Key authenticatorPK;
    private BigInteger prime;
    private ArrayList<String> voti;
    private HashMap<String, Integer> tallierCount;
    private ArrayList<publicInfoVoto> publicTallierList;

    public Tallier(CentralServer centralServer, Key authenticatorPK) {
        Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
        this.voti = new ArrayList<>();
        this.tallierCount = new HashMap<>();
        this.publicTallierList = new ArrayList<>();
        this.centralServerPK = centralServer.getPk();
        this.prime = centralServer.getPrime();
        this.authenticatorPK = authenticatorPK;
        initCandidati();
    }

    public void initCandidati() {
        tallierCount.put("Carpentieri", 0);
        tallierCount.put("Casaburi", 0);
        tallierCount.put("Falcone", 0);
        tallierCount.put("Ferraioli", 0);
        tallierCount.put("Greco", 0);
    }

    public void publishTallierList(){
        System.out.println("========= TALLIER LIST =========");
        for (publicInfoVoto info : publicTallierList){
            System.out.println(info.getInfo().getSignedPreference() + " | " + info.getInfo().getxM()
+ " | " + info.getVoto() + " | " + info.getMotivo());
        }
    }

    public void setSkShare(BigInteger skShare) {
        this.skShare = skShare;
    }

    public void reconstructSecret(BigInteger[] secretshares) throws Exception {
        BigInteger accum = BigInteger.ZERO;
        BigInteger tmp = null;
        BigInteger value = null;
        for (int j = 0; j < secretshares.length; j++) {
            BigInteger numerator = BigInteger.ONE;
            BigInteger denominator = BigInteger.ONE;
            for (int i = 0; i < secretshares.length; i++) {
                if (j != i) {
                    int startposition = j + 1;
                    int nextposition = i + 1;

                    numerator =
numerator.multiply(BigInteger.valueOf(nextposition).negate()).mod(prime); // (numerator * -
nextposition) % prime;
                    denominator = denominator.multiply(BigInteger.valueOf(startposition -
nextposition)).mod(prime); // (denominator * (startposition - nextposition)) % prime;
                }
            }
            value = secretshares[j];
            BigInteger di = numerator.multiply(denominator.modInverse(prime));
            tmp = value.multiply(di);
            accum = prime.add(accum).add(tmp).mod(prime); //  (prime + accum + (value * numerator *
modInverse(denominator))) % prime;
        }
```

```
        this.centralServerSK = new ElGamalSK(accum, centralServerPK);
    }

    public Boolean verifySignedMessage(ElGamalCT signedMessage, ElGamalCT message) {
        BCRSAPublicKey pubKey = (BCRSAPublicKey) authenticatorPK;
        BigInteger modulus = pubKey.getModulus();
        BigInteger e = pubKey.getPublicExponent();

        BigInteger U = signedMessage.U;
        BigInteger V = signedMessage.V;
        BigInteger resultU = U.modPow(e, modulus);
        BigInteger resultV = V.modPow(e, modulus);

        ElGamalCT newCT = new ElGamalCT(resultV, resultU);

        return message.U.compareTo(newCT.U) == 0 && message.V.compareTo(newCT.V) == 0;
    }

    public ArrayList<String> decPreferences(ArrayList<InfoVoto> preferences) {
        for (InfoVoto voto : preferences) {

            ElGamalCT signedPreference = voto.getSignedPreference();
            ElGamalCT xM = voto.getxM();

            if (verifySignedMessage(signedPreference, xM)) {
                BigInteger plain = ElGamalUtils.Decrypt(xM, centralServerSK);
                String pref = new String(plain.toByteArray());
                if (!tallierCount.containsKey(pref)) {
                    publicTallierList.add(new publicInfoVoto(voto, true, "Candidato non valido",
pref));
                } else {
                    publicTallierList.add(new publicInfoVoto(voto, false, "OK", pref));
                    voti.add(pref);
                }
            }

        }
        return voti;
    }

    public HashMap<String, Integer> countPreferences() {
        for (String voto : voti) {
            tallierCount.put(voto, tallierCount.get(voto) + 1);
        }
        return tallierCount;
    }

    private class publicInfoVoto {

        private InfoVoto info;
        private Boolean isScartato;
        private String motivo;
        private String voto;

        public publicInfoVoto(InfoVoto info, Boolean isScartato, String motivo, String voto) {
            this.info = info;
            this.isScartato = isScartato;
            this.motivo = motivo;
            this.voto = voto;
        }

        public InfoVoto getInfo() {
            return info;
        }

        public void setInfo(InfoVoto info) {
            this.info = info;
        }

        public Boolean getIsScartato() {
            return isScartato;
        }

        public void setIsScartato(Boolean isScartato) {
            this.isScartato = isScartato;
        }
```

```
        public String getMotivo() {
            return motivo;
        }

        public void setMotivo(String motivo) {
            this.motivo = motivo;
        }

        public String getVoto() {
            return voto;
        }

        public void setVoto(String voto) {
            this.voto = voto;
        }

        @Override
        public String toString() {
            return "publicInfoVoto{" + "info=" + info + ", isScartato=" + isScartato + ", motivo=" +
motivo + ", voto=" + voto + '}';
        }

    }
}
```

Main

```
package default_package;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws Exception {
        ArrayList<HashMap<String, Integer>> finalMap = new ArrayList<>();
        HashMap<String, Integer> csCount;

        //Initializing system entites
        Authenticator auth = new Authenticator();

        BulletinBoard board = new BulletinBoard();

        Anonymizer anon1 = new Anonymizer(1, 256);
        Anonymizer anon2 = new Anonymizer(2, 512);

        CentralServer cs = new CentralServer();

        //Building central server shares
        BigInteger bg[] = cs.buildShares(2);

        Tallier t1 = new Tallier(cs, auth.getPublicKey());
        Tallier t2 = new Tallier(cs, auth.getPublicKey());

        //Sharing central server key
        t1.setSkShare(bg[0]);
        t2.setSkShare(bg[1]);

        //Adding some voters to the authenticator dataset
        auth.addElettore("rob_falc", "eugenioNapoli");
        auth.addElettore("carpe_eug", "robertoLewis");
        auth.addElettore("ado_cas", "semanos365");
        auth.addElettore("l.ferr", "cavese1919");

        //FIRST VOTER
        Elettore elettore1 = new Elettore("rob_falc", "eugenioNapoli");

        if (auth.isElettoreLogin(elettore1.getUserID(), elettore1.getPassword())) {
            ElGamalCT blind = elettore1.blindPreference("Carpentieri", cs.getPk(),
auth.getPublicKey());
            ElGamalCT signed = auth.signElettore("rob_falc", blind);
            ElGamalCT signedPreference = elettore1.unblindPreference(signed, auth.getPublicKey());
            ArrayList<ElGamalCT> cryptedPref = elettore1.mixNetCipher(anon1, anon2);
            board.addPreference(cryptedPref);
        }
```

```
        //SECOND VOTER
        Elettore elettore2 = new Elettore("carpe_eug", "robertoLewis");

        if (auth.isElettoreLogin(elettore2.getUserID(), elettore2.getPassword())) {
            ElGamalCT blind2 = elettore2.blindPreference("Ferraioli", cs.getPk(),
auth.getPublicKey());
            ElGamalCT signed2 = auth.signElettore("carpe_eug", blind2);
            ElGamalCT signedPreference2 = elettore2.unblindPreference(signed2, auth.getPublicKey());
            ArrayList<ElGamalCT> cryptedPref = elettore2.mixNetCipher(anon1, anon2);
            board.addPreference(cryptedPref);
        }

        //THIRD VOTER USER INPUT
        System.out.println("Welcome back sir!");
        Scanner scan = new Scanner(System.in);

        System.out.println("Insert userID: ");
        String userID = scan.next();

        System.out.println("Insert password: ");
        String password = scan.next();

        Elettore elettore3 = new Elettore(userID, password);

        if (auth.isElettoreLogin(userID, password)) {
            System.out.println("Insert preference: ");
            String pref = scan.next();

            //Applying blind factor on expressed preference
            ElGamalCT blind3 = elettore3.blindPreference(pref, cs.getPk(), auth.getPublicKey());
            //Requesting the Authenticator signature
            ElGamalCT signed3 = auth.signElettore(elettore3.getUserID(), blind3);
            //Unblind Authenticator signature
            ElGamalCT signedPreference3 = elettore3.unblindPreference(signed3, auth.getPublicKey());
            //Crypting the message to send to the Bulletin Board with the MixNet keys
            ArrayList<ElGamalCT> cryptedPref = elettore3.mixNetCipher(anon1, anon2);
            //Sending the vote and the unblinded Authenticator signature to the Bulletin Board
            board.addPreference(cryptedPref);
        } else {
            System.out.println("Wrong UserID or Password!");
        }

        //Publishing Authenticator list
        auth.publishAuthenticatorList();

        //Taking the votes list from the Bulletin Board
        ArrayList<ElGamalCT> finalPrefs = board.getPreferences();

        //MixNet decryption
        anon2.decypher(finalPrefs);
        ArrayList<ElGamalCT> intCryptedPref = anon2.getDecrypted();
        anon1.decypher(intCryptedPref);
        ArrayList<InfoVoto> finalInfoVoto = anon1.getAnonSet();

        //Reconstructing secret
        t1.reconstructSecret(bg);
        t2.reconstructSecret(bg);

        //Splitting the votes list among the Tallier servers
        ArrayList<InfoVoto> list1 = new ArrayList<>(finalInfoVoto.subList(0, 1));
        ArrayList<InfoVoto> list2 = new ArrayList<>(finalInfoVoto.subList(1, 3));

        //Decrypting and opening the votes
        t1.decPreferences(list1);
        t2.decPreferences(list2);

        //Partial counting from the Tallier servers
        finalMap.add(t1.countPreferences());
        finalMap.add(t2.countPreferences());

        //Publishing Tallier lists
        t1.publishTallierList();
        t2.publishTallierList();

        //Final counting from the Central Server
        csCount = cs.finalCount(finalMap);
```

```
        //Printing the results
        cs.publishCentralServerList();

        //Code to print the winner
        //System.out.println("Winner: " + Collections.max(csCount.entrySet(),
Comparator.comparingInt(Map.Entry::getValue)).getKey());
    }

}
```

INDICE FIGURE

Figura 1, Rappresentazione grafica delle fasi dell'elezione 4
Figura 2, Radar chart 14
Figura 3, Diagramma UML 17

INDICE TABELLE

Tabella 1, Lista pubblicata dai server Authenticator 8
Tabella 2, Lista pubblicata dal Central Server 10
Tabella 3, Strumenti utilizzati 14