PROJECT WORK

Algoritmi e Protocolli per la Sicurezza

L.M. Ingegneria Informatica

Docenti: V. Iovino- I. Visconti

A.A. 2021-2022

WP1

Il voto elettronico

Il voto elettronico è un tema su cui si è molto discusso nell'ultimo ventennio, sia in Italia, sia nel resto del mondo. La digitalizzazione del voto ha molti obiettivi quali: migliorare l'efficienza del sistema e la partecipazione dei cittadini, diminuire i rischi di brogli e di inaccuratezza e, soprattutto, ridurre i costi. Non è possibile raggiungere tutti gli obiettivi senza scendere a compromessi con alcuni aspetti, anche perché alcuni dei requisiti/desideri sono tra loro contrastanti. Le sfide principali sono fornire una soluzione sicura e ottenere la fiducia degli elettori nell'utilizzarla.

Le elezioni portano con sé varie criticità e contraddizioni, spesso difficili da risolvere, come per esempio l'idoneità e la segretezza; infatti, l'elettore deve essere autenticato nel sistema come idoneo ma al tempo stesso deve risultare anonimo in esso; in aggiunta, il sistema deve considerare il tentativo di ripetere il voto. Ancora, la possibilità per l'elettore di verificare il proprio voto in contrasto con l'assenza di ricevuta. Infine, ci potrebbero essere anche ulteriori insidie durante l'invio del voto in quanto possibili avversari potrebbero intercettarlo e volerne carpire informazioni.

Il sistema di voto elettronico presentato cercherà di soddisfare quei requisiti che classificheremo come 'requisiti fondamentali' (o requisiti di base) del sistema, identificare le proprietà di resilienza del sistema in presenza di attacchi, discutere i possibili avversari (threat model) interessati a compromettere il sistema e misurare la bontà di una progettazione che prova a realizzare una tale funzionalità in presenza di avversari. Infine, implementare il sistema di voto remoto/elettronico progettato.

Scenario di voto

Esistono diverse tipologie di elezioni pubbliche, in questa trattazione andremo ad analizzare uno schema di voto con vari candidati su comunità di decine o centinaia di migliaia di persone. Più nello specifico, uno scenario reale che si adatta bene al seguente schema è rappresentato dalle elezioni comunali nelle quali gli elettori hanno la possibilità di votare per il candidato al ruolo di sindaco.

Fasi del voto elettronico

La procedura di voto elettronico è tipicamente divisa in quattro fasi:

1) $[T_0, T_1]$, rappresenta la fase di pre-voto, nella quale è prevista l'identificazione del votante presso un Identity Provider (IP) (mediante operazioni di registrazione, rilascio di eventuali credenziali e/o scambio di chiavi e segreti);
2) $[T_1, T_2]$, rappresenta la fase di votazione, nella quale viene presentata sullo schermo del votante la scheda elettorale elettronica; l'elettore esprime la sua preferenza e ne verifica la corrispondenza a video;
3) $[T_2, T_3]$, rappresenta la fase di post-voto, nella quale è concessa una finestra temporale di due giorni utile per eventuali contestazioni;
4) $[T_3, T_4]$, rappresenta la fase di pubblicazione, nella quale viene effettuato il conteggio dei voti e viene effettuata la pubblicazione dei risultati.

Assumiamo che in casi occasionali si possa coinvolgere la giustizia, che per semplicità indicheremo come G. Tale attore potrà essere invocato solo su richiesta di fronte ad un'evidenza di brogli, evitando quindi il più possibile il suo coinvolgimento. Seppur il sistema progettato cercherà di essere il più trasparente possibile, potrebbero nascere delle contestazioni in merito all'esito elettorale. In caso di contestazione contro l'esito elettorale, quindi, è possibile proporre ricorso, che deve essere redatto

con motivi specifici che saranno esaminati dalle autorità competenti. Quest'ultima parte è fuori dal sistema digitalizzato, di conseguenza non verrà approfondita nella trattazione.

Formalizzazione

Esiste una ***macchina della giustizia*** $\boldsymbol{MG}$, la quale è un dispositivo completamente affidabile e gestito dalla **giustizia** $\boldsymbol{G}$. Tale dispositivo ha la funzione di generare e dividere un segreto tra vari attori.

Siano $\boldsymbol{C_1}, \dots, \boldsymbol{C_n}$ gli ipotetici **candidati** eleggibili. Siano $\boldsymbol{CT_1}, \dots, \boldsymbol{CT_j}$ i **cittadini** aventi diritto di voto. Ogni cittadino diventa elettore dopo opportuna richiesta e approvazione. Ogni **elettore** $\boldsymbol{E_i} \in \{E_1, \dots, E_j\}$ può esprimere un **voto** $\boldsymbol{V_i}$. Tale voto può essere espresso più volte, ma solo l'ultimo sarà considerato valido.

Ciascun partecipante fisico alle elezioni (candidati, cittadini e autorità varie) si assume sia in possesso di un'**identità digitale** $\boldsymbol{SID_i}$ che viene rilasciata dopo opportune verifiche da un ente governativo.

Ciascun cittadino CT_i avrà a disposizione una fascia temporale $[T_0, T_1]$ durante la quale potrà identificarsi con l'identità digitale SID_i ad un portale certificato.

Una permissioned **blockchain** $\boldsymbol{BEV}$ (Blockchain E-Voting) fornisce i metodi necessari per la votazione. La governance della blockchain BEV è affidata ad ogni regione della nazione. Per ogni regione sono presenti diversi nodi denominati BN (Blockchain Node) $\in \{BN_1, \dots, BN_{k \cdot l}\}$, dove k è il numero delle regioni e l è il numero di nodi per regione. Le operazioni di scrittura su $\boldsymbol{BEV}$ sono regolate solamente dalle autorità, mentre le operazioni di lettura sono ammesse senza restrizioni.

Si precisa che $\boldsymbol{BEV}$ verrà utilizzata esclusivamente per la votazione. Al termine delle votazioni la blockchain viene conservata da tutti i nodi partecipanti in modo tale da mantenere traccia delle votazioni e rimanere accessibile per successive consultazioni.

Un qualsiasi BN, dopo opportune verifiche, abiliterà il cittadino alla generazione di credenziali randomiche e segrete e gli consentirà di diventare un elettore a tutti gli effetti. La chiave pubblica dell'elettore E_i verrà memorizzata nella blockchain in modo tale da consentire l'autenticazione in futuro.

Una volta autenticato sul portale, l'elettore E_i, durante la finestra temporale $[T_1, T_2]$, ha diritto di votare per un candidato eleggibile, segnando il relativo contrassegno. L'elettore E_i si identifica presso un nodo BN_i per poter inoltrare il proprio voto. Il nodo BN_i pubblicherà tale voto generando una nuova transazione sulla blockchain BEV. L'intervallo temporale tra l'istante di invio del voto e l'istante di pubblicazione è variabile e deciso randomicamente in tempo di esecuzione. Il voto V_i può essere controllato e validato da ogni elettore E_i guardando lo storico della blockchain. Il sistema di voto ammette la possibilità di votare rieseguendo la procedura. Nella blockchain verrà considerata valida solo l'ultima transazione contenente il voto dell'elettore E_i.

In seguito alla votazione, durante la fascia temporale $[T_2, T_3]$, si attende un tempo prefissato (ipotizzato di due giorni) in modo tale che ogni elettore E_i abbia la possibilità di aprire una controversia se sospettoso di eventuali frodi o se il suo voto è stato in qualche modo manomesso.

Infine, durante la fascia temporale $[T_3, T_4]$, ogni nodo BN effettua la somma cifrata dei voti $\mathrm{R} = \Sigma_{i=1}^{j} V_i$; tale somma verrà verificata tramite consenso e resa nota come transazione sulla blockchain. La correttezza del risultato potrà essere verificata pubblicamente da chiunque, dal momento che ogni voto cifrato è pubblico. La responsabilità della rivelazione del risultato delle elezioni è affidata al seguente gruppo di attori:

- ***n* Candidati**, che per semplicità indicheremo con la notazione $C_1, \dots, C_n$; tali attori sono considerati poco affidabili. Se considerati avversari, essi vorrebbero manomettere le elezioni per trarne vantaggio personale.
- ***n* Persone di Legge**, che per semplicità indicheremo con la notazione $PL_1, \dots, PL_n$; tali attori sono considerati affidabili al 65%. Essi sono esponenti giudiziari (magistrati, giudici, ecc) o esponenti delle forze dell'ordine (Questore, Capo della Polizia, ecc). Essi non gioverebbero nel compromettere la votazione dato che nel caso in cui lo facessero, la loro carriera cesserebbe, oltre che essere severamente puniti da G.
- ***Un Presidente della Provincia***, che per semplicità indicheremo con la notazione $PProv$; tale attore è considerato affidabile al 90%. Esso è un esponente politico di alto rango, il quale non è direttamente coinvolto nell'elezione. Compromettere la votazione, per lui, vorrebbe dire subire conseguenze che sarebbero disastrose e contro producenti per la sua reputazione e carriera.
- ***Un Presidente della Regione***, che per semplicità indicheremo con la notazione $PReg$; tale attore è considerato affidabile al 90%. Esso è un esponente politico di alto rango, il quale non è direttamente coinvolto nell'elezione. Esso non trarrebbe alcun vantaggio nel compromettere la votazione dato che le conseguenze che subirebbe sarebbero disastrose e contro producenti per la sua reputazione e carriera.
- ***Il Presidente della Repubblica,*** che per semplicità indicheremo con la notazione $PRep$; tale attore è considerato affidabile al 99% (a meno di governo corrotto). Esso è il capo dello stato, il quale non è direttamente coinvolto nell'elezione. Esso non trarrebbe alcun vantaggio nel compromettere la votazione dato che le conseguenze che subirebbe sarebbero disastrose, sia per sé stesso, sia per la nazione di cui è a capo, e contro producenti per la sua reputazione e carriera.

Infine, tutti gli attori responsabili della rivelazione collaborano per ottenere la decifratura del risultato senza ricostruirne il segreto. Tutte le informazioni rilasciate da ogni attore saranno di dominio pubblico tramite blockchain e chiunque potrà validare il risultato eseguendo la decifratura in modo autonomo.

N.B. Ovviamente nessun attore potrà mai ottenere il segreto, né tantomeno decifrare singoli voti.

Threat model

Di seguito si riportano gli avversari (A = attivo, P = passivo) individuati che potrebbero compromettere la sicurezza del sistema:

- ***The Meddler*** (P)**,** il ficcanaso è avversario interessato a scoprire la preferenza di singoli votanti o gruppi di essi o individuare la corrispondenza tra cittadino ed elettore. Un tale attaccante potrebbe carpire informazioni utili sull'orientamento politico o sull'identità dell'elettore ed eventualmente venderle a committenti che ne hanno fatto richiesta. Questo avversario opera sul canale di comunicazione utilizzato per inviare il voto o sulla blockchain (attraverso un'attenta analisi della sua storia). Si suppone che la sua potenza di calcolo sia superiore al dispositivo di un elettore, ma limitata comunque alle possibilità economiche di un singolo individuo, e che lavori su architetture classiche.
- ***The Dishonest Adviser*** (A)*:* l'avversario disonesto è un avversario interessato a modificare il voto di un elettore. Questo avversario opera sulla blockchain e sul canale di comunicazione utilizzato per inviare il voto, cercando di convincere un blockchain node di essere l'elettore effettivo (senza possedere, però, le credenziali). Si suppone che la sua potenza di calcolo sia superiore al dispositivo di un elettore, ma limitata comunque alle possibilità economiche di un

singolo individuo, e che lavori su architetture classiche. Questo attacco ha senso solo se il rapporto tra sforzo richiesto e valore ottenuto è vantaggioso.

- ***The Fake Doppleganger*** $(\boldsymbol{A})$*:* Il finto sosia, un avversario interessato a votare al posto di un elettore. Esso vorrebbe spacciarsi per un blockchain node in modo tale da modificare il voto dell'elettore forgiandone uno nuovo per poi inoltrarlo ad un reale blockchain node. Si suppone che la sua potenza di calcolo sia superiore al dispositivo di un elettore, ma limitata comunque alle possibilità economiche di un singolo individuo, e che lavori su architetture classiche.
- ***The Unfair Voter*** $(\boldsymbol{A})$*:* il votante sleale, un avversario interessato a votare più volte e far considerare validi entrambi i voti in modo da favorire o sfavorire un determinato candidato. Si suppone che la sua potenza di calcolo sia superiore al dispositivo di un elettore, ma limitata comunque alle possibilità economiche di un singolo individuo, e che lavori su architetture classiche.
- ***The Evil Citizen (A):*** cittadino malvagio, un avversario che vorrebbe essere ammesso al voto pur non essendo legittimato ad esserlo. Il suo scopo è quello di convincere un blockchain node di essere in possesso di un SID valido e di non aver effettuato la richiesta delle credenziali in precedenza. Si suppone che la sua potenza di calcolo sia superiore al dispositivo di un elettore, ma limitata comunque alle possibilità economiche di un singolo individuo, e che lavori su architetture classiche.
- ***The Crafty Administrator*** $(\boldsymbol{P})$*:* l'amministratore furbo, un avversario interessato a scoprire i voti prima che essi vengano contati. Si tratta di un avversario che lavora all'interno di una sede dove risiede un blockchain node. Tale avversario tramite i suoi privilegi da dipendente potrebbe ottenere e divulgare queste informazioni. Questo avversario non ha accesso al canale di comunicazione utilizzato per inviare il voto, ma accede ai voti guardando il flusso di informazioni che arrivano ad un blockchain node. Questo avversario non dispone di una potenza di calcolo elevata poiché utilizza soltanto dispositivi aziendali.
- ***The Smily Persuader*** $(\boldsymbol{A})$*:* il persuasore, un avversario in grado di avere un'influenza sull'elettore. Esso potrebbe convincere un elettore a votare per un candidato a sua scelta o di non votare. Questo avversario non utilizza mezzi digitali ma opera applicando pressione psicologica o fisica sull'elettore. Si suppone che la sua capacità di persuasione sia elevata.
- ***The Thief*** $(\boldsymbol{A})$***:*** il ladro, un avversario interessato a rubare le credenziali di accesso di un elettore. Esso vorrebbe appropriarsi delle credenziali di un elettore effettuando una copia fisica (foto o trascrizione) delle credenziali. Questo avversario non dispone di una potenza di calcolo, si tratta tuttavia di una figura estremamente discreta e scaltra nel furto.
- ***The Corrupt Actor*** $(\boldsymbol{A})$***:*** la persona di legge corrotta, un avversario che è in possesso di informazioni riservate e potrebbe decidere, sotto retribuzione, di cedere tali informazioni a un candidato o ad un'altra tipologia di avversario. La potenza computazionale di questo avversario non è elevata poiché non è interessato ad altro se non svelare informazioni che dovrebbero rimanere confidenziali.
- ***The Unfair Candidate*** $(\boldsymbol{A})$***:*** il candidato sleale, un avversario interessato a manomettere le elezioni coinvolgendo uno o più avversari sotto retribuzione. Si assume che tale avversario disponga di una elevata somma di denaro ed una enorme potenza computazionale.
- ***The Overloaders*** $(\boldsymbol{A})$***:*** gruppo di avversari intenzionati a sovraccaricare il sistema durante la fase di voto (effettuando numerose richieste di voto in successione), in modo da impedire agli elettori di votare, rendendo il servizio inutilizzabile. Questi avversari hanno una potenza di calcolo estremamente superiore al dispositivo di un elettore in quanto potrebbero essere capaci di mettere insieme più architetture di calcolo ottenendo una potenza di calcolo estremamente maggiore. Dispongono, inoltre, di considerevoli risorse economiche.

- ***The Jackals*** $(\boldsymbol{A})$*:* gruppi di avversari di cui sopra, intenzionati a collaborare per arrivare ad un obiettivo comune. Questi avversari hanno una potenza di calcolo estremamente superiore al dispositivo di un elettore in quanto potrebbero essere capaci di mettere insieme più architetture di calcolo ottenendo una potenza di calcolo estremamente maggiore. Dispongono, inoltre, di considerevoli risorse economiche.

Proprietà

Si assume che in presenza di avversari il sistema considera la correttezza del risultato prioritaria rispetto ad altre proprietà riguardanti la privacy e la coercizione.

Integrità

In presenza di avversari il sistema di voto dovrebbe comunque garantire le seguenti proprietà:

- I.1 È necessario che la preferenza espressa da un elettore venga conteggiata correttamente.
- I.2 Il votante deve essere sicuro che il proprio voto sia integro per tutta la durata del seggio elettorale. Ciò significa che per nessun soggetto coinvolto nel sistema deve essere possibile, a partire da un voto lecitamente immesso nel sistema, forgiare un nuovo voto che appare come valido (indipendentemente dal momento in cui è stata effettuata la votazione).
- I.3 Il voto rilasciato da un elettore deve essere ritenuto valido solo se rappresenta l'ultima scelta effettuata (eventuali voti precedenti devono essere considerati nulli o eliminati dal sistema).
- I.4 La preferenza deve essere espressa secondo la libera scelta del votante, contrastando la possibilità della coercizione che un avversario potrebbe esercitare su un elettore.
- I.5 Sono idonei a votare solo gli utenti che si sono precedentemente registrati e che ne hanno il diritto.
- I.6 La proprietà di integrità del sistema deve permanere anche in caso di comportamenti malevoli dei soggetti coinvolti, anche basati su collusione di ragionevole dimensione.

Privacy

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

- P.1. L'identità dell'elettore deve essere riservata e non associabile alla preferenza scelta, né in quel momento, né in futuro.
- P.2 Le credenziali di accesso di un elettore devono essere completamente segrete tramite opportune contromisure.
- P.3. Le credenziali di accesso di un elettore, seppur rese pubbliche in modo malevolo, non devono mai ricondurre ad un cittadino.
- P.4. Il voto è segreto, solo l'elettore può verificare il voto effettivo che ha inviato; nessuno dei soggetti coinvolti nella procedura elettorale, può scoprire il voto di un elettore o collegare il voto all'identità di un cittadino.
- P.5. La proprietà di privacy deve permanere anche in caso di comportamenti malevoli dei soggetti coinvolti, anche basati su collusione di ragionevole dimensione.
- P.6. Non deve essere possibile avere informazioni parziali sul risultato del voto prima del conteggio ufficiale.
- P.7 Le credenziali dell'identità digitale fornite da un funzionario devono essere completamente segrete tramite opportune contromisure fisiche (sigillo, carta oscuratrice, ecc.).

Trasparenza

- T.1. Il sistema non dovrebbe essere basato su algoritmi segreti.
- T.2. Qualunque soggetto, compreso un osservatore passivo, può verificare che il risultato dell'elezione è corretto.

Efficienza

- E.1. Il sistema deve assicurare un funzionamento efficiente anche in caso di elevato numero di partecipanti.
- E.2. Il sistema deve assicurare il servizio anche in caso di guasti.

WP2

Trust assumption

Per garantire il corretto funzionamento del sistema, si considerano valide le seguenti assunzioni:

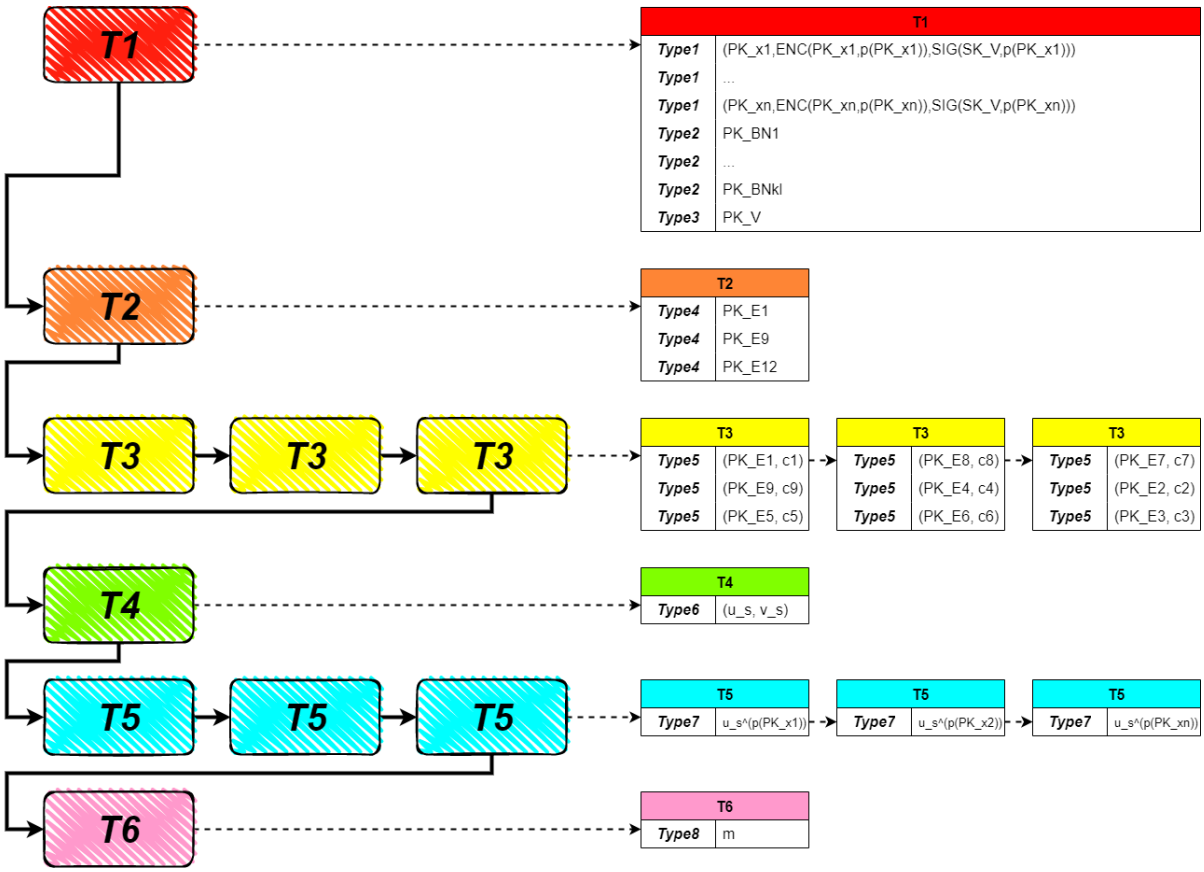
- Tutti i cittadini hanno sufficiente dimestichezza con la tecnologia e quindi possono utilizzare dispositivi per votare elettronicamente da casa;
- I dispositivi di chi è onesto non sono corrotti, ovvero funzionano correttamente senza essere controllati da malware/virus/trojans;
- Non si può avere la totale fiducia sull'onestà dell'elettore perché potrebbe vendere il suo voto dimostrando il contenuto della scheda elettorale (es. con foto o video)
- Il candidato non è considerato onesto. Esso può rientrare in due diverse tipologie di avversari: attivo (obbligando tramite minaccia gli elettori); passivo (utilizzando pubblicità ingannevole screditando gli altri candidati).

Per semplicità ogni volta che è coinvolto un certificato si fa effettivamente riferimento all'elenco concatenato dei certificati fino alla radice (cioè l'autorità di certificazione).

La privacy e la sicurezza dei dati per le comunicazioni su Internet tra Client e Server viene fornita dal protocollo TLS, che crea un canale sicuro tra i due endpoints e permette di identificarne le parti.

Strutturazione della blockchain

La governance della blockchain BEV è affidata ad ogni regione della nazione. Per ogni regione sono presenti diversi nodi denominati BN (Blockchain Node) $\in \{BN_1, \dots, BN_{k \cdot l}\}$, dove k è il numero delle regioni e l è il numero di nodi per regione. Nel caso in esame, il numero delle regioni è $k = 20$ con $l = 20$ numero di nodi per regione. BEV è costituita da diverse tipologie di blocchi e transazioni.



Possiamo suddividerle come segue:

T1. ***Blocco genesi***: esso è il primo blocco della blockchain generato da MG contenente tre tipologie di transazioni:
   a. $Type1 = c_{x_i} = \left(PK_{x_i}, ENC\left(PK_{x_i}, p\left(PK_{x_i}\right)\right), SIG(SK_V, p(PK_{x_i}))\right)$, transazioni contenenti la parte del segreto dell'attore x_i cifrata con la chiave pubblica PK_{xi} e la firma digitale;
   b. $Type2 = PK_{BNj}$, transazioni contenenti le chiave pubbliche dei nodi BN;
   c. $Type3 = PK_V$, transazione contenete la chiave pubblica generata da MG che verrà utilizzata per la cifratura dei voti.

T2. Tale blocco contiene le chiavi pubbliche degli elettori E_i che sono stati certificati dai nodi BN e ammessi al voto.
   a. $Type4 = PK_{E_i}$.

T3. Tale blocco contiene le transazioni dei voti espressi dagli elettori E_i:
   a. $Type5 = V_{E_i} = (PK_{E_i}, c_{E_i})$.

T4. Tale blocco è formato da un'unica transazione contenente la somma cifrata dei voti:
   a. $Type6 = c_s = \prod_i^n c_{E_i} = (u_s, v_s)$.

T5. Tale blocco è formato dalle transazioni contenenti i contributi calcolati dagli attori che hanno una share del segreto:
   a. $Type7 = u_s^{p\left(PK_{x_i}\right)}$.

T6. Tale blocco è formato da un'unica transazione contenente il risultato in chiaro delle elezioni:
   a. $Type8 = m; dove\ g^m = \frac{V_s}{w}$.

È importante sottolineare che ogni transazione che viene inserita nella blockchain deve essere firmata ed approvata da tutti i BN. Per semplicità nella trattazione viene omessa.

Fase di Pre-Voto

Ottenimento del SID (Sistema di Identità Digitale)

Ogni cittadino deve essere in possesso di un SID, ovvero un'identità digitale certificata da un Centro di Autorità Governativo. Tale identità digitale viene fornita in un tempo antecedente al voto, anche di diversi mesi o anni. Lo scopo del SID è quello di ottenere le credenziali di voto in modalità telematica evitando intermediari fisici, i quali sono ad alto rischio di corruzione.

Il SID viene interpretato come una coppia chiave segreta, chiave pubblica: $SID = (SK_{CT_i}, PK_{CT_i})$. La PK_{CT_i} viene autenticata dal Centro di Autorità Governativo come mezzo sicuro e affidabile per istaurare una comunicazione sicura con il cittadino CT_i.

Una possibile interpretazione (escludendo il threat model) del modello del rilascio delle credenziali è il seguente:

1. ***Richiesta di rilascio del SID***: Il cittadino CT_i si reca in una filiale del *Centro di Autorità Governativo* CAG per effettuare una richiesta di ottenimento del SID mostrando un documento di riconoscimento in corso di validità (carta d'identità o patente, tessera sanitaria).
2. ***Rilascio del SID***: il funzionario rilascia al cittadino CT_i una busta chiusa, appositamente realizzata per nascondere nel modo più efficace le credenziali del *SID*.
3. ***Inserimento in Banca Dati del SID***: nello stesso lasso di tempo in cui il funzionario rilascia il SID, esso inserisce in piattaforma l'associazione (e quindi l'autenticità) tra le informazioni (cifrate con PK_{CT_i}) del cittadino fisico CT_i e la chiave pubblica PK_{CT_i}.

Le possibili minacce a tale modello sono esenti da questa trattazione. L'obiettivo di questo modello consiste nell'avere uno strumento ben definito per un'identificazione sicura e affidabile del cittadino CT_i, il quale diventerà una nuova figura, ovvero quella dell'elettore E_i. È importante specificare che i vari attori sono considerati anch'essi cittadini (candidati, persone di legge, ecc).

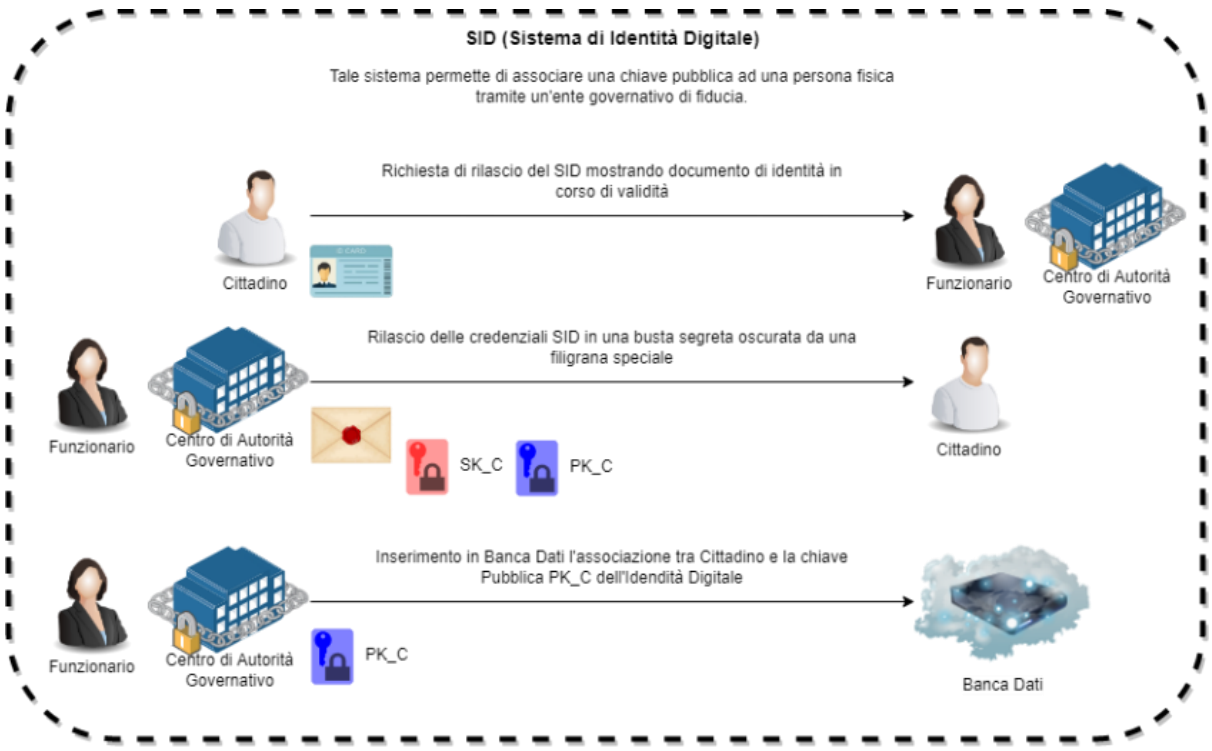


Figura 1 Modello di rilascio del SID

Scambio del segreto

Uno dei fattori che causano sfiducia negli attori che utilizzano un sistema di voto elettronico, è la centralizzazione di un attore chiave in grado di gestire e leggere in chiaro tutte le informazioni private (ovvero il voto scelto).

Per evitare la centralizzazione, quindi, il segreto non è più posseduto da un singolo attore ma viene suddiviso tra attori di diversa tipologia.

L'idea si basa su un software open-source di dominio pubblico che genera e distribuisce il segreto, il quale gira su un dispositivo non connesso ad internet e completamente sicuro contro tutte le possibili minacce. Dato che ognuno può testarlo in ogni minima parte, esso suscita nei vari attori che lo utilizzano una maggiore fiducia delle sue caratteristiche di sicurezza ad attacchi di qualunque tipo.

La responsabilità, quindi, si divide in M attori di diverso tipo. Dati $x_i \in \{C_1, \dots, C_N\} \cup \{PL_1, \dots, PL_N\} \cup \{PRep, PReg, PProv\}$, essi sono:

1. N Candidati;
2. N Persone di Legge;
3. 1 Presidente della Repubblica;
4. 1 Presidente della Regione;
5. 1 Presidente della Provincia;

Questi attori sono stati scelti per un motivo ben preciso.

Dato $M = 2N + 3$ (ovvero il totale degli attori coinvolti nel segreto), si è deciso di utilizzare lo strumento di condivisione del segreto basato sui polinomi "Shamir Secret Sharing (t, n)", dove i parametri sono $t = M - 2, n = M$.

È necessario effettuare alcune considerazioni sul valore di M, il quale non è esponenzialmente alto, dato che sia per elezioni comunali, sia per altre tipologie di elezioni, il numero di candidati difficilmente supera l'ordine delle decine.

Ricordiamo l'affidabilità di tali attori:

1. ***Presidente della Repubblica***: è l'unico attore affidabile al 99%. Esso è il capo dello stato, il quale non è direttamente coinvolto nell'elezione.
2. ***Presidente della Regione e della Provincia***: tali attori sono considerati affidabili al 90%. Essi sono esponenti politici di alto rango, i quali non sono direttamente coinvolti nell'elezione.
3. ***Persone di Legge***: tali attori sono considerati affidabili al 65%. Essi sono esponenti giudiziari (magistrati, giudici, ecc) o esponenti delle forze dell'ordine (Questore, Capo della Polizia, ecc).
4. ***Candidati***: tali attori sono considerati poco affidabili. Essi sono attori che, se considerati avversari, vorrebbero manomettere le elezioni per trarne vantaggio personale.

Il motivo per cui è stato scelto questo insieme, è perché è altamente improbabile che $M - 2$ attori corrotti collaborino per ottenere il segreto. Il Presidente della Regione o il Presidente della Provincia difficilmente, se non altamente improbabile, si alleerebbero con Persone di Legge o con i Candidati per poter scoprire il voto anticipatamente, questo perché, come detto in precedenza, non andrebbe verso i loro interessi (carriera rovinata e gravi conseguenze penali). Lo stesso vale per le Persone di Legge, le quali difficilmente si alleerebbero con i Candidati, dato che anch'essi subirebbero una perdita di notevole valore, oltre che andare contro i propri interessi. I Candidati, invece, potrebbero corrompere qualche Persona di Legge o qualche esponente di alto rango, ma un'alleanza tra loro sarebbe improbabile, questo perché sarebbe controproducente ai fini dell'elezione (ognuno vorrebbe

vincere le elezioni). Infine, l'aggiunta del Presidente della Repubblica favorisce una maggiore affidabilità.

Nel caso estremo che tutti i candidati e tutte le persone di legge siano corrotte, servirebbe il consenso di almeno uno dei tre attori più affidabili ($PRep, PReg, PProv$) per poter ricostruire il segreto o decifrare un determinato voto. Se nessuno dei tre attori viene corrotto, secondo la proprietà di Shamir Secret Sharing, non è possibile in alcun modo ricostruire il segreto.

Lo scambio del segreto avviene in due macro-fasi: la prima consiste nell'eseguire il software open-source isolato che si occupa di generare, dividere, cifrare e distruggere il segreto, la seconda, invece, si occupa di distribuire le cifrature delle share tra i vari attori in gioco tramite la permissioned blockchain BEV.

La macchina su cui gira il software di generazione del segreto è $\boldsymbol{MG}$ (Macchina della Giustizia).

La prima macro-fase si articola nei seguenti passi:

1. ***Ottenimento delle chiavi pubbliche degli attori***: G si occupa di ottenere le varie chiavi pubbliche degli attori PK_{x_i} che condivideranno il segreto, le quali sono fornite in input a MG.
2. ***Generazione del segreto***: MG genera una coppia di chiavi $k = (SK_V, PK_V)$ usate successivamente per la cifratura del voto (PK_V) e per la decifratura (SK_V) in modo distribuito.
3. ***Creazione del polinomio***: MG utilizzando Shamir Secret Sharing $(t = M - 2, n = M)$, costruisce un polinomio di grado $t - 1$, il cui termine noto è il segreto (SK_V). Il polinomio è il seguente: $p(x) = SK_V + \sum_{i=1}^{t-1} a_i x^i$, dove ogni coefficiente a_i è completamente random (scelto in Z_q).
4. ***Generazione delle coppie***: MG calcola n punti del polinomio $p(x)$ e crea delle triple cifrate $c_{x_i} = (PK_{x_i}, ENC(PK_{x_i}, p(PK_{x_i})), SIG(SK_V, p(PK_{x_i})))$.
5. ***Distruzione della chiave segreta e del polinomio***: MG distrugge in maniera permanente sia il segreto SK_V, sia il polinomio $p(x)$, in modo tale da rendere la chiave segreta irrecuperabile. Dopo l'eliminazione, MG viene connesso ad internet e diventa capace di inviare messaggi verso l'esterno.

La seconda macro-fase si articola nei seguenti passi:

1. ***Creazione del blocco genesi***: MG annuncia il primo blocco della blockchain *BEV* (blocco genesi) di tipo T1 specificando le seguenti transazioni:
   i. ***La share del segreto per ogni attore***: $c_{x_i} = (PK_{x_i}, ENC(PK_{x_i}, p(PK_{x_i})), SIG(SK_V, p(PK_{x_i})))$. Tale informazione può essere decifrata solo tramite chiave segreta dell'attore x_i, ovvero tramite SK_{x_i}. Questo permette agli attori di ottenere lo share del segreto in modo ufficiale. Ogni attore decifra il proprio valore di $p(PK_{x_i}) = DEC(SK_{x_i}, c_{x_i})$ ottenendo così una parte del segreto. Tale valore deve essere al sicuro!
   ii. ***Le chiavi pubbliche delle autorità della blockchain***: $PK_{BN_1}, \dots, PK_{BN_{k \cdot l}}$. Tali chiavi pubbliche saranno utilizzate dagli elettori per potersi collegare (tramite HTTPS) ad un servizio che consente la votazione da remoto.
   iii. ***La chiave pubblica per le votazioni***: PK_V. Tale chiave pubblica verrà utilizzata dagli elettori per poter effettuare la votazione.
2. ***Comunicazione tramite canali ufficiali***: MG annuncia tramite canali ufficiali il blocco genesi di BEV in modo tale da consentire l'inizio delle votazioni.

Formalizzazione

Il modello sopra citato può essere formalmente definito come segue:

1. G ottiene le chiavi pubbliche PK_{x_i} *via HTTPS.*
2. MG genera la coppia di chiavi usando $El\ Gamal\ Encryption$

$$k = (PK_V, SK_V) = ((p, q, g, y), x) = Gen(1^n)$$

con n parametro di sicurezza,
$y = g^x$ chiave pubblica,
x chiave segreta.

3. MG calcola il polinomio usando $Shamir\ Secret\ Sharing\ (t, n)$

$$p(x) = SK_V + \sum_{i=1}^{t-1} a_i x^i$$

con a_i coefficiente random del polinomio scelto in Z_q.

4. MG calcola i punti del polinomio

$$(PK_{x_i}, ENC(PK_{x_i}, p(PK_{x_i})),) = GenPairs(p(x), n)$$

con n numero di attori coinvolti nel segreto.

5. MG distrugge il segreto e il polinomio:

$$destroy(SK_V, p(x))$$

6. MG genera il blocco genesi di BEV contenente le transazioni di tipo $Type1, Type2, Type3$.
7. Ogni attore x_i coinvolto nel segreto decifra e ottiene lo share del segreto

$$p(PK_{x_i}) = DEC(SK_{x_i}, c_{x_i})$$

Ottenimento delle credenziali di voto

La settimana antecedente all'inizio delle votazioni, ogni cittadino CT_i abilitato al voto, deve fare richiesta delle credenziali di voto. Tali credenziali sono completamente differenti dal SID, il quale è lo strumento necessario affinché il cittadino CT_i possa ottenere le suddette credenziali.

Nota. Il cittadino CT_i d'ora in poi diverrà elettore e sarà etichettato con il termine E_i.

Il modello dell'ottenimento delle credenziali di voto può essere espresso nei seguenti passi:

1. ***Istaurazione di una comunicazione con un nodo BN***: il cittadino CT_i richiede il certificato del nodo BN e istaura una connessione con esso tramite HTTPS. Il protocollo TLS consente ai due attori di comunicare in maniera confidenziale, autentica e integra.
2. ***Identificazione tramite SID***: una volta istaurata una connessione sicura, il cittadino CT_i effettua la richiesta delle credenziali di voto, dimostrando tramite firma digitale che esso è in possesso della chiave segreta SK_{CT_i}.
3. ***Richiesta al Centro di Autorità Governativo CAG:*** il nodo BN dopo aver verificato la PK_{CT_i}, richiede al Centro di Autorità Governativo CAG se tale PK_{CT_i} appartiene ad un cittadino avente diritto al voto.
4. ***Abilitazione per la generazione delle credenziali:*** Se il Centro di Autorità Governativo CAG conferma il diritto al voto, BN comunica a CT_i che è abilitato a generare le credenziali di voto.
5. ***Generazione delle credenziali:*** il cittadino CT_i genera le credenziali di voto randomiche $Cred = (SK_{E_i}, PK_{E_i})$ e invia al nodo BN la chiave pubblica PK_{E_i} con la relativa firma digitale. Se tale cittadino provasse ad effettuare una seconda richiesta, quest'ultima verrebbe rifiutata dal CAG, il quale tiene traccia delle richieste precedentemente effettuate.
6. ***Inserimento in Blockchain:*** BN crea una nuova transazione sulla blockchain creando un blocco di tipo T2 contenente PK_{E_i}, in modo tale da verificare nella fase successiva se l'elettore E_i è ammesso al voto.

Formalizzazione

Il modello sopra citato può essere formalmente definito come segue:

1. CT_i avvia una comunicazione con un nodo BN *via HTTPS.*
2. CT_i si identifica come reale possessore di PK_{CT_i}, richiedendo di generare le credenziali di voto.
3. BN verifica tramite firma digitale la veridicità dell'identità del cittadino CT_i

$$Vrfy(SIG, m, PK_{CT_i}) == 1$$

4. Se è dimostrato tramite firma digitale, BN richiede al CAG se la chiave pubblica PK_{CT_i} è abilitata al voto

$$canVoteCitizen(PK_{CT_i}) == 1$$

5. In caso di risposta affermativa da parte del CAG, BN comunica a CT_i che è abilitato al voto.
6. Il cittadino CT_i genera la coppia di chiavi usando $El\ Gamal\ Encryption$

$$k = (PK_{E_i}, SK_{E_i}) = ((p, q, g, y), x) = Gen(1^n)$$

con n parametro di sicurezza,
$y = g^x$ chiave pubblica,
x chiave segreta.

7. Il cittadino CT_i invia al BN la PK_{E_i} e la firma digitale.
8. BN verifica tramite firma digitale la veridicità della PK_{E_i} dell'elettore E_i

$$Vrfy(SIG, m, PK_{E_i}) == 1$$

9. Se è dimostrato tramite firma digitale, BN genera un nuovo blocco su BEV di tipo $T2$ contenente la transazione di tipo $Type4$.

Fase di voto

In questa fase avviene l'invio del voto da parte dell'elettore E_i.

La fase di voto si articola nei seguenti passi:

1. ***Istaurazione della comunicazione***: l'elettore E_i istaura una comunicazione sicura con un nodo BN attraverso HTTPS. BN consente, tramite un apposito portale web, l'accesso da parte dell'elettore in modo tale da votare nel modo più semplice possibile.
2. ***Dimostrazione che il cyphertext non violi il protocollo:*** E_i dimostra, tramite ZKP, a BN che il voto m_i ha il seguente formato: $00 \dots 00b_1||00 \dots 00b_2||\dots||00 \dots 00b_{n_c-1}||00 \dots 00b_{n_c}$. Dove solo una delle b_i può assumere valore 1, il resto deve essere 0. In caso di scheda bianca, tutte le b_i assumono valore 0. Il numero di bit per candidato varia a seconda del contesto, nel caso delle elezioni comunali, il comune più popoloso ha circa 2'700'000 abitanti $\approx 2^{22}$.
3. ***Cifratura e invio del messaggio***: E_i crea un cyphertext $c_i = (u_i = g^{r_i}, v_i = g^{m_i} y^{r_i})$, dove r_i è una stringa casuale scelta in Z_q, g è il generatore del gruppo, m_i è il voto e y è PK_V (chiave pubblica della votazione generata dalla macchina della giustizia MG).
4. ***Dimostrazione che*** $\boldsymbol{E_i}$ ***sia il legittimo possessore della chiave segreta*** $\boldsymbol{SK_{E_i}}$***:*** E_i invia sul canale la tripla $(c_i, TIME\_STAMP, SIG(SK_{E_i}, c_i||TIME\_STAMP))$ in modo tale che BN possa verificare che E_i sia il legittimo possessore della chiave segreta, mentre il $TIME\_STAMP$ consente di evitare dei reply attack.
5. ***Inserimento del voto nella blockchain***: BN alla ricezione di c_i, sceglie un valore random di tempo $1min \leq t_r \leq 60min$ nelle ore diurne, mentre nelle ore notturne $1min \leq t_r \leq 240min$ (a causa del basso afflusso di voti). Al termine di questo tempo t_r, inserisce la transazione in un nuovo blocco (in attesa di aggiunta alla coda della blockchain) la coppia $V_{E_i} = (PK_{E_i}, c_i)$, la quale è visibile a tutti, in modo tale da consentire la verifica ad ogni E_i di aver inviato correttamente il proprio voto. Se t_r supera il tempo limite per il voto, allora sullo scattare tutte le coppie rimanenti vengono aggiunte al blocco finale di tipo T3. Attendere un tempo t_r prima della pubblicazione, consente di effettuare uno shuffle delle varie ricezioni dei messaggi, in modo tale che non si riesca a risalire alla sessione di invio del voto conoscendo l'istante di pubblicazione.

Formalizzazione

Il modello sopra citato può essere formalmente definito come segue:

1. E_i avvia una comunicazione con un nodo BN *via HTTPS.*
2. E_i esprime la sua preferenza di voto m_i.
3. E_i calcola $Enc(PK_V, m_i)$ ottenendo:

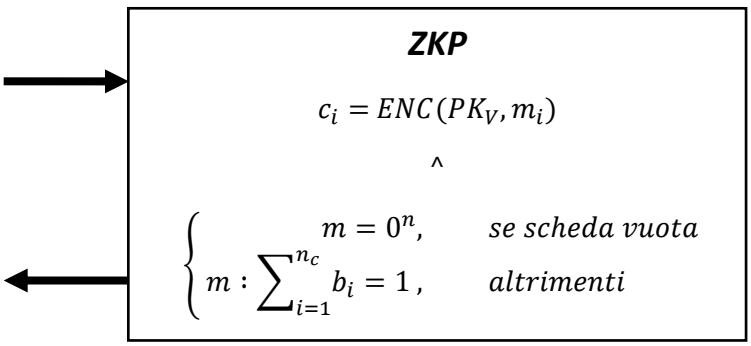
$$c_i = (u_i, v_i) = (g^{r_i}, g^{m_i} y^{r_i})$$

4. E_i invia sul canale la tripla

$$(c_i, TIMESTAMP, SIG(SK_{E_i}, c_i || TIMESTAMP))$$

5. E_i dimostra a BN tramite ZKP che il voto inviato V_i ha il seguente formato:

$$00 \dots 00b_1 || 00 \dots 00b_2 || \dots || 00 \dots 00b_{n_c-1} || 00 \dots 00b_{n_c}.$$



Il voto m_i e la randomness r_i sono conosciuti solo dal votante (il quale ha il ruolo di ***prover***) mentre il BN che ha ricevuto il voto e gli altri BN che convalidano il blocco sono i ***verifier*** i quali conoscono solo il *ciphertext* c_i e la *proof*.

6. BN attende un tempo $1min \leq t_r \leq 60min$ nelle ore diurne o un tempo $1min \leq t_r \leq 240min$ nelle ore notturne.
7. BN genera un nuovo blocco su BEV di tipo $T3$ contenente la transazione di tipo $Type5$.

Fase di pubblicazione

In questa fase si prevede il conteggio dei voti e la decifratura del risultato in chiaro.

La fase di pubblicazione si articola come segue:

1. ***Conteggio dei voti:*** Ogni BN effettua la moltiplicazione dei vari cyphertext $c_i = (u_i = g^{r_i}, v_i = g^{m_i} y^{r_i})$ presenti nei blocchi di tipo T3 della blockchain contenenti i voti degli elettori, producendo un nuovo cyphertext $c_s = \left(u_s = g^{\sum_{i=1}^{n} r_i}, v_s = g^{\sum_{i=1}^{n} m_i} y^{\sum_{i=1}^{n} r_i}\right)$ che contiene la somma dei voti, sfruttando l'omomorfismo additivo di ElGamal. È importante sottolineare che se sono presenti cyphertext con la stessa chiave pubblica (c'è stato un rivoto) verrà considerato nel conteggio solo il più recente. Questa operazione può essere effettuata, e quindi verificata, da chiunque.
2. ***Pubblicazione del cyphertext somma c_s:*** il primo BN che completa la computazione del conteggio, genera un nuovo blocco di tipo T4 contenente c_s in attesa di consenso. Gli altri BN al termine della computazione, verificano se è già stato calcolato c_s da un altro BN. In caso affermativo, se quello che è in attesa di consenso corrisponde a quello calcolato allora viene confermato il consenso.
3. ***Invio dei contributi degli attori***: ogni attore coinvolto nella decifratura, calcola il contributo $u_s^{p\left(PK_{x_i}\right)}$ e tramite un BN lo inserisce nella blockchain come una transazione di un nuovo blocco di tipo T5, dimostrando tramite ZKP che tale contributo sia veritiero ottenendo il consenso dalle autorità della governance.
4. ***Decifratura di c_s:*** Ogni BN cerca di decifrare il cyphertext somma c_s sfruttando l'interpolazione di Lagrange, ovvero: $w = \prod_{j=1}^{n-2} u_s^{p\left(PK_{x_j}\right)\cdot d(i)} = u_s^{\sum_{j=1}^{n-2} p\left(PK_{x_j}\right)\cdot d(i)} = u_s^{x} = y^{r_s}$. Successivamente si ricava g^m secondo la seguente formula: $g^m = \frac{v_s}{w}$. Successivamente, tramite una ricerca bruteforce si riesce a ricavare il messaggio m. Questa operazione, seppur onerosa, può essere effettuata, e quindi verificata, da chiunque.
5. ***Pubblicazione del risultato in chiaro***: il primo BN che completa la computazione della decifratura, genera un nuovo blocco di tipo T6 contenente m in attesa di consenso. Gli altri BN al termine della computazione, verificano se è già stato calcolato m da un altro BN. In caso affermativo, se quello che è in attesa di consenso corrisponde a quello calcolato allora viene confermato il consenso.

Al termine delle elezioni, la giustizia G attende un tempo t per eventuali contestazioni. Allo scadere di t, impone a tutti gli attori aventi il segreto di avviare una procedura verificata per distruggere in completa affidabilità le share del segreto.

Formalizzazione

Il modello sopra citato può essere formalmente definito come segue:

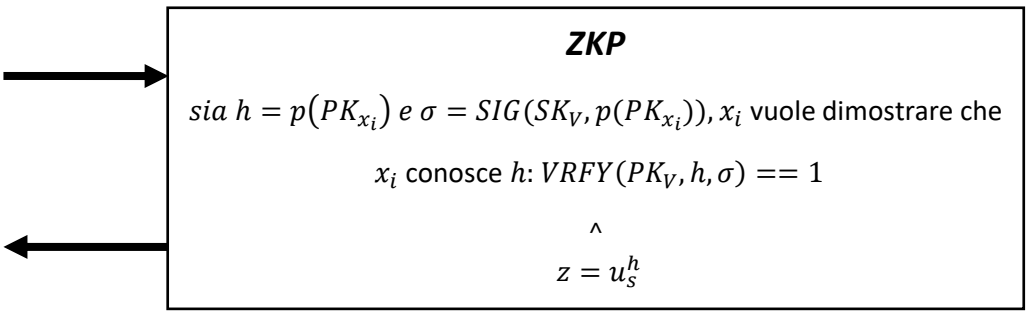
1. Ogni BN calcola il cyphertext contenente la somma dei voti

$$c_s = \left(u_s = g^{\sum_{i=1}^{n} r_i}, v_s = g^{\sum_{i=1}^{n} m_i} y^{\sum_{i=1}^{n} r_i}\right)$$

2. Il primo BN che effettua la computazione di c_s genera un blocco su BEV di tipo T4 contenente una transazione di tipo $Type6$. Gli altri BN danno il consenso se la computazione da loro effettuata corrisponde a quella del primo BN.
3. *Ogni* x_i avvia una comunicazione con un nodo BN *via HTTPS*.
4. *Ogni* x_i invia sul canale la tupla contenente il contributo firmato:

$$\left(z = u_s^{p\left(PK_{x_i}\right)}, SIG\left(SK_{x_i}, u_s^{p\left(PK_{x_i}\right)}\right)\right)$$

dimostrando tramite ZKP che il contributo inviato è corretto.



Lo share $p\left(PK_{x_i}\right)$ è conosciuto solo dall'attore x_i (il quale ha il ruolo di ***prover***) mentre il BN che ha ricevuto il contributo e gli altri BN che convalidano il blocco sono i ***verifier***, i quali conoscono solo il *contributo* z, la firma $SIG(SK_V, p(PK_{x_i}))$ (ricavabile in blockchain) e la *proof*.

5. BN genera un nuovo blocco su BEV di tipo $T5$ contenente la transazione di tipo $Type7$.
6. Al termine del raccoglimento dei contributi, ogni BN calcola il risultato delle votazioni

$$g^m = \frac{V_S}{w}, dove\ w = \prod_{j=1}^{n-2} u_s^{p\left(PK_{x_j}\right)\cdot d(i)} = u_s^{\sum_{j=1}^{n-2} p\left(PK_{x_j}\right)\cdot d(i)} = u_s^x = y^{r_s}.$$

Successivamente tramite ricerca brute force viene calcolato il risultato delle elezioni m.

7. Il primo BN che effettua la computazione di m genera un blocco su BEV di tipo T6 contenente una transazione di tipo $Type8$. Gli altri BN danno il consenso se la computazione da loro effettuata corrisponde a quella del primo BN.
8. Ogni attore coinvolto può consultare BEV per ottenere i risultati delle elezioni.

WP3

Analisi della completezza

Nell'ipotesi in cui tutti gli attori in gioco eseguano il protocollo descritto, non ci siano contestazioni e nessun coinvolgimento della giustizia G e supponendo che tutti i tool crittografici utilizzati siano corretti, le elezioni proclameranno il nuovo sindaco.

Si noti che la giustizia è coinvolta quando ci sono problemi legati al mondo fisico (es. consegna del SID, sanzioni civili e penali) e questo è inevitabile poiché il sistema digitalizzato non può prevenire le frodi nel mondo reale. Inoltre, la giustizia viene coinvolta anche durante l'inizio e la fine delle elezioni. Essa avrà il compito di scegliere gli attori che condivideranno il segreto e di assicurarsi che le elezioni terminino correttamente.

È importante, tuttavia, che il sistema digitalizzato dia prove evidenti di comportamenti disonesti e ciò viene rispettato dal sistema realizzato.

Analisi delle proprietà in presenza di attaccanti

Discutiamo uno per uno gli avversari che potrebbero cercare di attaccare il sistema e analizziamo le proprietà che il sistema garantisce in presenza di tali attacchi.

Si assume che i dispositivi di chi è onesto non siano corrotti; quindi, che l'hardware non sia compromesso ed il software non sia controllato da malware/virus/trojans; seguirà comunque una breve discussione su cosa può accadere in caso contrario.

Integrità

Dishonest adviser

Il **dishonest adviser** non può agire sul canale di comunicazione tra l'elettore E_i e un blockchain node BN e tentare di manipolare il voto perché quest'ultimo prevede l'utilizzo del protocollo TLS che garantisce l'integrità e l'autenticazione. L'avversario potrebbe pensare, tuttavia, di spacciarsi per un elettore E_i utilizzando la chiave pubblica PK_{E_i}. Questo tipo di attacco verrebbe contrastato dalla firma digitale prevista durante la fase di invio, la quale convince il blockchain node BN che colui che ha inviato il voto è il legittimo proprietario di tale chiave pubblica PK_{E_i}. Inoltre, l'avversario non riuscirebbe a ricorrere ad un attacco di tipo reply perché il messaggio firmato contiene sia il voto, sia il timestamp. Nel caso in cui venisse inviato un messaggio vecchio, il blockchain node lo scarterebbe e lo renderebbe nullo. In nessun caso il dishonest adviser potrebbe modificare i voti pubblicati sulla blockchain poiché tali voti vengono pubblicati come transazioni firmate con un algoritmo di firma digitale *unforgeable* su una blockchain: tali transazioni non possono essere né cancellate né modificate. Tali contromisure adottate consentono al sistema di rispettare le proprietà di integrità **I.1**. e **I.2**.

Fake Doppleganger

L'elettore non viene tratto in inganno da blockchain node BN fittizi poiché la transazione di tipo $Type2$ fornisce le PK dei blockchain node ammessi a riceve i voti, i quali sono gli unici affidabili. Di conseguenza, un semplice controllo di questa informazione permetterebbe all'elettore di non essere raggirato e impedirebbe al fake doppleganger di avere successo. Seppure quest'ultimo riuscisse a ingannare l'elettore riuscendo ad ottenere un voto e, a partire da esso, provasse a forgiarne uno nuovo per la proprietà di omomorfismo di ElGamal, non riuscirebbe a convincere un blockchain node perchè

non è in grado di firmare il voto forgiato, in quanto non è il reale possessore della chiave segreta SK_{E_i}. Per tale motivo viene rispettata la proprietà di integrità **I.2.**

Unfair voter

L'**unfair voter** è interessato ad esprimere più volte la propria preferenza. Il primo metodo che ha per farlo consiste nella generazione di nuove chiavi pubbliche con le quali si vorrebbe far identificare dal blockchain node BN come un legittimo elettore. Tale comportamento viene negato dal blockchain node BN dal momento che è in possesso della lista di tutte le chiavi pubbliche degli elettori ammessi al voto. Se l'avversario tentasse di autenticarsi con una chiave pubblica fittizia, il blockchain node invaliderebbe immediatamente tale procedura. Un secondo metodo che l'unfair voter può adottare consiste nel compromettere il formato del voto e, ad esempio, votare per due candidati anziché uno. Questo non può verificarsi poiché l'elettore deve necessariamente fornire una ZKP valida sul corretto formato del voto. Un terzo metodo che l'avversario può utilizzare per generare nuovi voti consiste nel cercare di compromettere la normale procedura d'invio del voto. Per fare ciò, l'avversario può pensare di inviare voti ripetutamente a più blockchain node e far in modo che lo stesso voto venga conteggiato più di una volta. Ogni sforzo sarebbe, tuttavia, vano dal momento che, se sono presenti cyphertext con la stessa chiave pubblica, verrà considerato nel conteggio solo l'ultimo voto inviato. Questa operazione può essere effettuata, e quindi verificata, da chiunque e ogni difformità può essere rilevata in qualsiasi momento. Infine, l'ultimo metodo consiste nella richiesta continua di nuove credenziali: a partire da un SID valido, un cittadino, una volta approvato anche dal CAG, potrebbe pensare di effettuare una seconda richiesta di generazione delle credenziali per ottenere in tal modo ulteriori identità digitali ammesse al voto. Tale richiesta verrebbe tempestivamente rifiutata dal CAG, il quale tiene traccia delle richieste precedentemente effettuate.

Tale soluzione consente di rispettare le proprietà di integrità **I.1**, **I.3** e **I.5.**

Evil Citizen

Si assume che nessun cittadino può ingannare il sistema di rilascio del SID poiché tale sistema è stato considerato inattaccabile e incorruttibile. Inoltre, tale modello applica delle contromisure digitali efficaci (protocolli affidabili) e contromisure fisiche (metodi di stampa e di rilascio) certificate e approvate da leggi governative. Questa assunzione consente al sistema il soddisfacimento della proprietà di privacy ***P.7***. Poiché non può ottenere illecitamente un SID valido, l'unico modo che ha l'evil citizen per essere ammesso al voto, è la generazione di un SID casuale: il SID altro non è che una coppia chiave segreta, chiave pubblica $SID = (SK_{CT_i}, PK_{CT_i})$, dunque, supponendo che l'avversario riuscisse a generare un SID casuale e ingannare il blockchain node BN di essere il reale possessore della chiave pubblica PK_{CT_i} (poiché in possesso della chiave segreta SK_{CT_i}) e passare il primo step per la generazione delle credenziali di voto, verrebbe successivamente bloccato dal CAG il quale verifica, una volta ricevuta richiesta dal BN, se tale PK_{CT_i} appartiene ad un cittadino ammesso al voto.

Tali contromisure adottate consentono al sistema di rispettare la proprietà di integrità **I.5**.

Smily persuader

Lo Smily persuader ha l'obiettivo di influenzare la scelta di uno o più elettori tramite pressione psicologica e/o fisica. Il sistema realizzato non garantisce la protezione totale da questo tipo di attacco, ma permette di mitigarlo implementando un meccanismo di rivoto. Tale meccanismo consiste nel dare la possibilità all'elettore di votare diverse volte nell'intervallo T1-T2 (ognuno distanziato da un intervallo temporale di 60 minuti). Ogni voto inviato viene inserito tramite transazione (Type5) nei

blocchi di tipo T3 della blockchain e, successivamente, nella fase di conteggio T3-T4, se sono presenti cyphertext con la stessa chiave pubblica (c'è stato una rivoto) viene considerato nel conteggio solo il più recente.

Nel caso in cui questa forzatura avvenga negli ultimi istanti di tempo disponibili per esprimere il proprio voto (meno di un'ora prima dalla fine delle votazioni) è presente una fase di post-voto T2-T3. In questa fascia temporale si attende un tempo prefissato (ipotizzato di due giorni) in modo tale che ogni elettore abbia la possibilità di aprire una controversia se sospettoso di eventuali frodi o se il suo voto è stato in qualche modo manomesso.

Così facendo, il sistema tenta di contrastare al meglio l'attacco dello Smily persuader e di rispettare la proprietà **I.4**, sebbene sia impossibile garantirla con certezza.

Jackals

L'unfair voter non può in alcun modo generare nuove chiavi pubbliche per poter essere ammesso al voto più di una volta e quindi non può aiutare l'unfair candidate a vincere le elezioni incrementando in maniera illecita il numero di consensi per quel candidato. Così come una coalizione tra unfair voter e unfair candidate sarebbe inconcludente, anche una coalizione tra Unfair Candidate e Smily persuader diverrebbe infruttuosa, infatti, pur supponendo che un avversario intimidatore riesca ad applicare costrizione psicologica o, nel caso più estremo, fisica, sull'elettore, costringendolo a votare per il candidato con il quale collude in cambio di denaro, non riuscirebbe ad arrivare al suo scopo poiché il sistema ammette il rivoto. Seppur si cercasse in qualche modo di impedire all'elettore di rivotare una volta espresso il voto, coinvolgendo ad esempio, un terzo tipo di avversari, gli overloaders, che avrebbero il compito di congestionare, in un secondo momento, il sistema, in modo da rendere il servizio inutilizzabile e da non permettere all'elettore di rivotare ponendo rimedio alla coercizione da lui subita, non si riuscirebbe a portare a casa il risultato sperato. Come sarà descritto in seguito, un attacco da parte degli overloaders non avrebbe successo poiché tali attaccanti non sono in grado di sovraccaricare il sistema. La proprietà di integrità **I.6** potrebbe essere parzialmente colpita da una grande collusione che coinvolge tutti i candidati, tutte le persone di legge e almeno uno dei tre attori più affidabili ($PRep, PReg, PProv$), cosa che però è abbastanza improbabile che accada a causa dei rischi di collusione degli attori istituzionali e dell'effettivo guadagno dei collusi. Ciò permette al sistema di conservare la proprietà di integrità **I.6**.

Privacy

Meddler

Un attacco da parte del **Meddler** sarebbe subito scoraggiato, infatti il sistema prevede che il canale sul quale viene inviato il voto dell'elettore sia sicuro perché protetto dal protocollo TLS. Tale protocollo garantisce confidenzialità, integrità e autenticazione. Una volta inibito l'attacco sul canale, l'avversario potrebbe riuscire ad ottenere l'indirizzo IP dal quale è stata instaurata la connessione con il blockchain node e osservando la blockchain prima e dopo l'invio del voto dell'utente potrebbe risalire, così, all'identità dell'elettore (ad esempio conoscendo l'indirizzo IP dell'elettore e sfruttando altri siti sui quali è stato inserito nome e cognome da quell'indirizzo IP). Questo scenario viene reso altamente improbabile dal momento che non si può determinare a priori il momento di inserimento del voto nella blockchain dato che l'intervallo di tempo t_r per l'inserimento viene scelto randomicamente. Essendo il numero di voti ricevuti dal blockchain node (in quel lasso di tempo) elevato, non è possibile risalire a quale delle varie identità corrisponde quell'indirizzo IP. Questa soluzione permette di soddisfare la proprietà di privacy ***P.1*** e la proprietà ***P.4*** parzialmente (*collegare il voto all'identità di un cittadino*).

Inoltre, se le credenziali di accesso di un elettore venissero rese pubbliche, sarebbero del tutto indipendenti dall'identità del cittadino CT_i dovuto al fatto che al momento del rilascio, la procedura consente di disaccoppiare totalmente l'identità digitale del cittadino CT_i dalle credenziali di accesso dell'elettore E_i. Infatti, tali credenziali vengono generate ex-novo dal cittadino durante tale fase e sono esenti di alcuna informazione riguardante colui che le ha generate (il cittadino CT_i). Questa soluzione permette di soddisfare la proprietà di privacy ***P.3.***
Infine, il meddler non ha la possibilità di conoscere il voto di altri votanti prima del conteggio ufficiale né di ottenere alcuna informazione guardando il cyphertext associato alla PK dell'elettore (se per qualche ragione ne venisse a conoscenza) in quanto i voti vengono cifrati utilizzando uno schema di cifratura a chiave pubblica di ElGamal con la chiave pubblica della votazione $\mathrm{PK_V}$. La chiave segreta SK_V è stata generata dalla macchina della giustizia MG (piena fiducia) per poi essere suddivisa e distrutta in maniera irreversibile. Poiché è altamente improbabile che tale avversario riesca ad ottenere tutti i contributi, non è in grado di effettuare la decifratura del messaggio cifrato con la $\mathrm{PK_V}$ e pertanto l'attacco fallisce e l'elettore resta l'unico in grado di verificare se il voto è stato inserito correttamente in blockchain, controllando che l'ultimo cyphertext associato alla sua PK corrisponde al cyphertext calcolato in fase di voto (opportunamente salvato in precedenza). Questa soluzione permette di soddisfare la proprietà di privacy ***P.6*** e la proprietà ***P.4*** parzialmente (*scoprire il voto di un elettore*)***.***

Corrupt actor

L'attore corrotto può sempre decidere di cedere informazioni riservate per un tornaconto personale, poiché si tratta di un attacco fisico che non utilizza mezzi digitali. Tuttavia, seppur uno degli attori possessori del segreto decidesse di svelare informazioni altamente riservate, non minaccerebbe in nessun modo la privacy del sistema dal momento che i voti vengono cifrati con uno schema di cifratura a chiave pubblica Threshold ElGamal e nessuna coalizione di meno di t attori otterrebbe informazioni sul segreto dalle loro condivisioni collettive (oltre a tutte le informazioni che avevano già sul segreto). Dunque, l'unica ipotesi per la quale questo attacco ha successo è quella in cui tutti i candidati, tutte le persone di legge e almeno uno dei tre attori più affidabili ($PRep, PReg, PProv$) colludono per ricostruire il segreto, ma tale coalizione sarebbe altamente improbabile da realizzare per i motivi già discussi in precedenza.

Nella fase di ricostruzione, inoltre, un attore malintenzionato non può presentare una quota diversa da quella assegnatagli dalla macchina della giustizia MG o decidere di non pubblicare il proprio contributo di decifratura, e quindi pregiudicare il segreto recuperato (sotto assunzione che il dealer, che in questo caso è la macchina della giustizia, sia onesto) poiché ogni attore deve dimostrare, tramite ZKP, che il contributo inviato corrisponde a quello ricevuto in fase di condivisione del segreto, in aggiunta al fatto che, qualsiasi azione illecita è visibile sulla blockchain e di conseguenza perseguibile.

Tale soluzione permette di soddisfare le proprietà di privacy **P.4** e **P.6**.

Unfair Candidate

Il candidato sleale potrebbe colludere, per esempio, con un corrupt actor in possesso di una share del segreto in modo da ottenere quella share. Il corrupt actor, dal suo canto, sarebbe interessato a cedergliela in cambio di denaro. Tuttavia, un tale attacco sarebbe pressoché vano se non altamente improbabile a causa della contrapposizione degli interessi delle figure che dovrebbero allearsi. Quando il dealer è onesto (e nel nostro caso si assume che la macchina della giustizia lo sia), gli attori

corrotti $t-1$ non apprendono nulla del segreto dalle loro azioni e da qualsiasi informazione pubblica fornita dal dealer. Poiché vogliamo che ci siano t attori non corrotti anche se $t-1$ attori sono corrotti, assumiamo che la maggioranza degli attori rimanga incorrotta. Tale assunzione consente al sistema di rispettare la proprietà di privacy **P.5**. La soluzione a questa minaccia è stata già discussa per il corrupt actor.

Crafty Administrator

Un attacco da parte del crafty administrator è scoraggiato dal fatto che per la cifratura dei voti viene utilizzata la chiave pubblica della votazione PK_V. La chiave segreta SK_V corrispondente, è stata generata dalla giustizia G (piena fiducia) e viene decentralizzata tra M attori (per poi essere distrutta in maniera irreversibile) in modo tale che i voti possano essere decifrati solo se i M attori mettono insieme le loro parti del segreto. Attraverso questo approccio nessuno conosce la chiave segreta SK_V, nemmeno i vari blockchain node, a meno che i t attori non colludano per ricostruirla, ma abbiamo discusso già in precedenza i motivi per il quale questo tipo di attacco è improbabile che si verifichi. Tale soluzione permette di soddisfare le proprietà di privacy **P.4** e **P.6**.

Thief

Il ladro, intenzionato a ottenere le credenziali di accesso, potrebbe provare ad agire nell'intervallo T0-T1 in due fasi: nella fase di ottenimento del SID e nella fase di ottenimento delle credenziali di voto. Avendo assunto il SID essere uno strumento d'identificazione sicuro e affidabile e avendo escluso da questa trattazione le possibili minacce ad esso, assumiamo che l'avversario non riesca in alcun modo a rubare le credenziali SID. Per quanto riguarda le credenziali di voto, esse vengono approvate da BN ma generate dall'elettore E_i. La chiave segreta SK_{E_i} non viene mai divulgata in alcuna comunicazione, nemmeno su un canale cifrato con protocollo TLS. L'unico modo per ottenere le credenziali è quello di minacciare fisicamente l'elettore E_i, ma in tal caso l'elettore potrebbe ricorrere alla giustizia G. Inoltre, anche nell'ipotesi in cui il ladro possa trarre qualche tipo di vantaggio nel rendere pubbliche tali credenziali, queste ultime sarebbero del tutto indipendenti dall'identità del cittadino CT_i e l'unico modo per conoscere quale cittadino si cela dietro l'identità dell'elettore è quello della violenza fisica. Questo comporta che le proprietà **P.2** e **P.3** siano soddisfatte.

Nel caso in cui il ladro attaccasse il dispositivo dell'elettore iniettando su di esso malware/virus/trojans (dispositivo dell'elettore corrotto), il sistema non garantisce il rispetto delle precedenti proprietà.

Trasparenza

Il sistema è stato progettato per fornire un'elevata trasparenza, infatti, tutte le operazioni vengono eseguite con la verificabilità pubblica di tutte le figure coinvolte. Ciò assicura che le frodi non possano essere nascoste modificando le informazioni utilizzate durante le varie fasi dell'elezione. Qualunque soggetto, compreso un osservatore passivo, ad esempio, può verificare che il risultato dell'elezione è corretto semplicemente guardando alla storia della blockchain e accertandosi che il cyphertext che viene decifrato corrisponde effettivamente alla somma di tutti i cyphertext presenti sulla blockchain, rispettando in questo modo la proprietà di trasparenza **T.2**. Per quanto riguarda la proprietà **T.1**, essa è implicitamente rispettata in quanto tutti gli algoritmi di cifratura utilizzati nel sistema sono noti e pubblicamente verificabili.

Efficienza

Il sistema è stato progettato per fornire un'elevata integrità, confidenzialità e trasparenza delle informazioni di tutti i soggetti coinvolti nelle elezioni e delle operazioni eseguite da essi. Ne consegue

una prevedibile penalizzazione dell'efficienza del sistema, la quale non riuscirebbe ad essere migliorata senza svantaggiare integrità, confidenzialità e trasparenza. D'altronde, tale penalizzazione sembrerebbe essere un giusto compromesso per un sistema che viene utilizzato occasionalmente e che non necessita di performance elevate. Cerchiamo di capire meglio in quali punti del sistema l'efficienza risulta penalizzata.

Sicuramente l'utilizzo di uno schema di cifratura asimmetrica ElGamal esponenziale sfavorisce l'efficienza perché, in fase di decifratura, consente di ricavare g^m e non direttamente m (come avviene per lo schema di cifratura asimmetrica ElGamal standard). Ne consegue una necessaria ricerca Bruteforce per calcolare il messaggio m ovvero il risultato delle elezioni, operazione onerosa in termini di calcoli, la cui complessità è $O(2^n)$, con n pari al numero di bit per candidato per il numero di candidati. L'utilizzo di un algoritmo più efficiente apporterebbe una leggera miglioria all'efficienza del sistema, che resterebbe comunque oneroso.

Un'altra penalizzazione dell'efficienza è dovuta all'utilizzo di Blockchain Node BN per l'inserimento del voto nella blockchain. L'attesa di un valore random di tempo t_r prima della pubblicazione del voto, da parte di BN, nella blockchain, consente di effettuare uno shuffle delle varie ricezioni dei messaggi, in modo tale che non si riesca a risalire alla sessione di invio del voto conoscendo l'istante di pubblicazione. Ovviamente tale operazione causa dei rallentamenti ulteriori, ma d'altronde questo contribuisce ad accrescere la confidenzialità del sistema.

Infine, l'efficienza del sistema è sicuramente influenzata dall'utilizzo di una blockchain che introduce una latenza dovuta alla somma delle latenze di operazioni come l'approvazione delle transazioni, l'aggiunta delle transazioni sulla blockchain e l'utilizzo di un collegamento internet per lo scambio di informazioni oltre ad una minor efficienza in termini di memoria occupata. Ciò, tuttavia, rende il sistema molto trasparente.

Essendoci diversi blockchain node sparsi sul territorio nazionale, in caso di guasto ad uno di essi, le richieste verranno reindirizzate ad altri BN. Così facendo viene rispettata la proprietà ***E.2.***

Overloaders

All'interno del sistema è previsto che un elettore, una volta inviato il proprio voto, possa rivotare solo dopo un periodo pari a 60 min. Questo implica che un attacco da parte degli overloaders non avrebbe successo. Cerchiamo di formalizzare meglio di seguito.

Ipotizzando che:

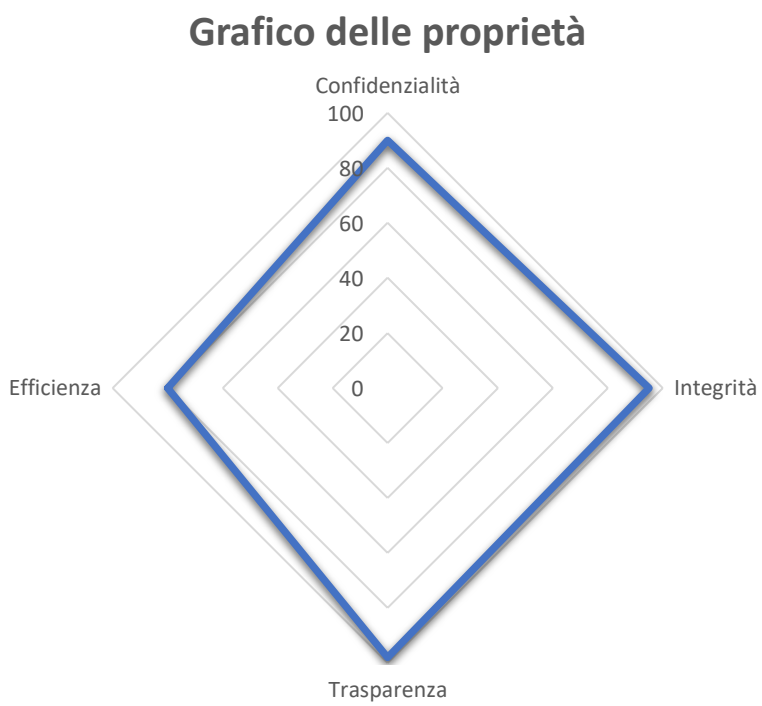
- Il numero massimo di elettori è di circa 2'700'000 (in riferimento al comune più popoloso in Italia);
- La frequenza di voti esprimibili per ogni elettore è pari a 1 voto all'ora.

Si nota che il numero di voti totali esprimibili in media in un'ora è pari a 2'700'000, ovvero 750 voti al secondo, i quali corrispondono ad un numero equivalente di transazioni sulla blockchain. Tale valore è altamente supportato da blockchain permissioned, le quali riescono a supportare anche decine di migliaia di transazioni al secondo.

Per tali motivazioni, gli overloaders non riuscirebbero a sovraccaricare in alcun modo il sistema rispettando la proprietà ***E.1.***

Tool crittografici utilizzati

Tool usato in WP2	Usato per proprietà
Protocollo Transport Layer Security (TLS)	I.2 (parzialmente), P.1
ZKP sul corretto formato del cyphertext	I.1, I.6
ZKP sulla veridicità del contributo degli attori	I.6
Schema di cifratura asimmetrica ElGamal esponenziale (con proprietà di omomorfismo additivo)	I.6, P.4, P.6
Schema Shamir secret-sharing (t, n)-threshold	I.6, P.4, P.6
Schema di firma di Schnorr	I.2
Permissioned Blockchain	I.1, I.2, I.3, I.5, I.6



WP4

Semplificazioni

Il progetto è stato sviluppato tenendo conto delle seguenti semplificazioni:

- Utilizzo di certificati self-signed basati su cifratura RSA invece di curve ellittiche.
- La permissioned blockchain BEV è stata rappresentata con file di testo e una classe Blockchain, la quale controlla l'accesso e le operazioni che possono avvenire su di essa. Si è scelto di suddividere la blockchain in 8 file di testo, ognuno contente una tipologia di transazione ammissibile.
- Le transazioni sulla blockchain non sono state firmate e approvate dai diversi BN.
- La macchina della giustizia MG è stata rappresentata come una classe in grado di:
  - Creare il blocco genesi;
  - Ottenere informazioni dal CAG;
  - Creare il polinomio per la distribuzione del segreto tramite Shamir Secret Sharing(t,n);
- I BN, gli attori e gli elettori, sono stati gestiti come tre diversi applicativi i quali sono:
  - PW_Shacalli_BlockchainNode: rappresenta un generico BN. Tutti gli attori ed elettori fanno riferimento ad esso per la comunicazione, inoltre, esso è il responsabile per la gestione della blockchain;
  - PW_Shacalli_Attori: rappresenta tutti gli Attori coinvolti nel segreto;
  - PW_Shacalli_Elettori: rappresenta tutti gli Elettori ammessi alle elezioni.
- La ZKP è stata implementata come una black-box, la quale, data una ZK Proof, afferma che essa è verificata sempre;
- Il SID è stato generato in runtime dagli elettori, il quale viene considerato sempre ammesso al voto dal CAG, il quale è stato implementato nella classe della macchina della giustizia MG.

Inizializzazione del sistema

Il sistema è stato inizializzato con la generazione, per tutte le entità coinvolte, dei certificati, i trust store ed i key store. La creazione di questi ultimi è avvenuta tramite l'utilizzo del Java Keytool, uno strumento utilizzato per la gestione dei certificati che consente di poter creare le chiavi pubbliche, private e dei certificati che le contengono. Questi certificati vengono salvati in file con estensione ".jks" e risultano essere necessari per generare le catene di fiducia.

Creazione del keystore e del certificato del BlockchainNode definito come $BN01$:

```
keytool -genkey -noprompt -trustcacerts -keyalg RSA -alias BN01 -dname "cn=BN01" -keypass BN01pass
-keystore keystoreBN01.jks

keytool -export -alias BN01 -storepass BN01pass -file BN01.cer -keystore keystoreBN01.jks
```

Creazione del keystore e del certificato degli attori definito come $Attori$:

```
keytool -genkey -noprompt -trustcacerts -keyalg RSA -alias Attori -dname "cn=Attori" -keypass
Attoripass -keystore keystoreAttori.jks

keytool -export -alias Attori -storepass Attoripass -file Attori.cer -keystore keystoreAttori.jks
```

Creazione del keystore e del certificato degli elettori definito come $Elettori$:

```
keytool -genkey -noprompt -trustcacerts -keyalg RSA -alias Elettori -dname "cn= Elettori " -
keypass Elettoripass -keystore keystoreElettori.jks
```

```
keytool -export -alias Elettori -storepass Elettoripass -file Elettori.cer -keystore
keystoreElettori.jks
```

Scambio dei certificati tra le entità che devono comunicare tra di loro.

Aggiunta del certificato del BN01 nel trust store degli Attori

```
keytool -import -v -trustcacerts -alias BN01 -keystore truststoreAttori.jks -file BN01.cer -
keypass BN01pass -storepass Attoripass
```

Aggiunta del certificato degli Attori nel trust store del BN01

```
keytool -import -v -trustcacerts -alias Attori -keystore truststoreBN01.jks -file Attori.cer -
keypass Attoripass -storepass BN01pass
```

Aggiunta del certificato del BN01 nel trust store degli Elettori

```
keytool -import -v -trustcacerts -alias BN01 -keystore truststoreElettori.jks -file BN01.cer -
keypass BN01pass -storepass Elettoripass
```

Aggiunta del certificato degli Elettori nel trust store del BN01

```
keytool -import -v -trustcacerts -alias Elettori -keystore truststoreBN01.jks -file Elettori.cer
-keypass ElettoriPass -storepass BN01pass
```

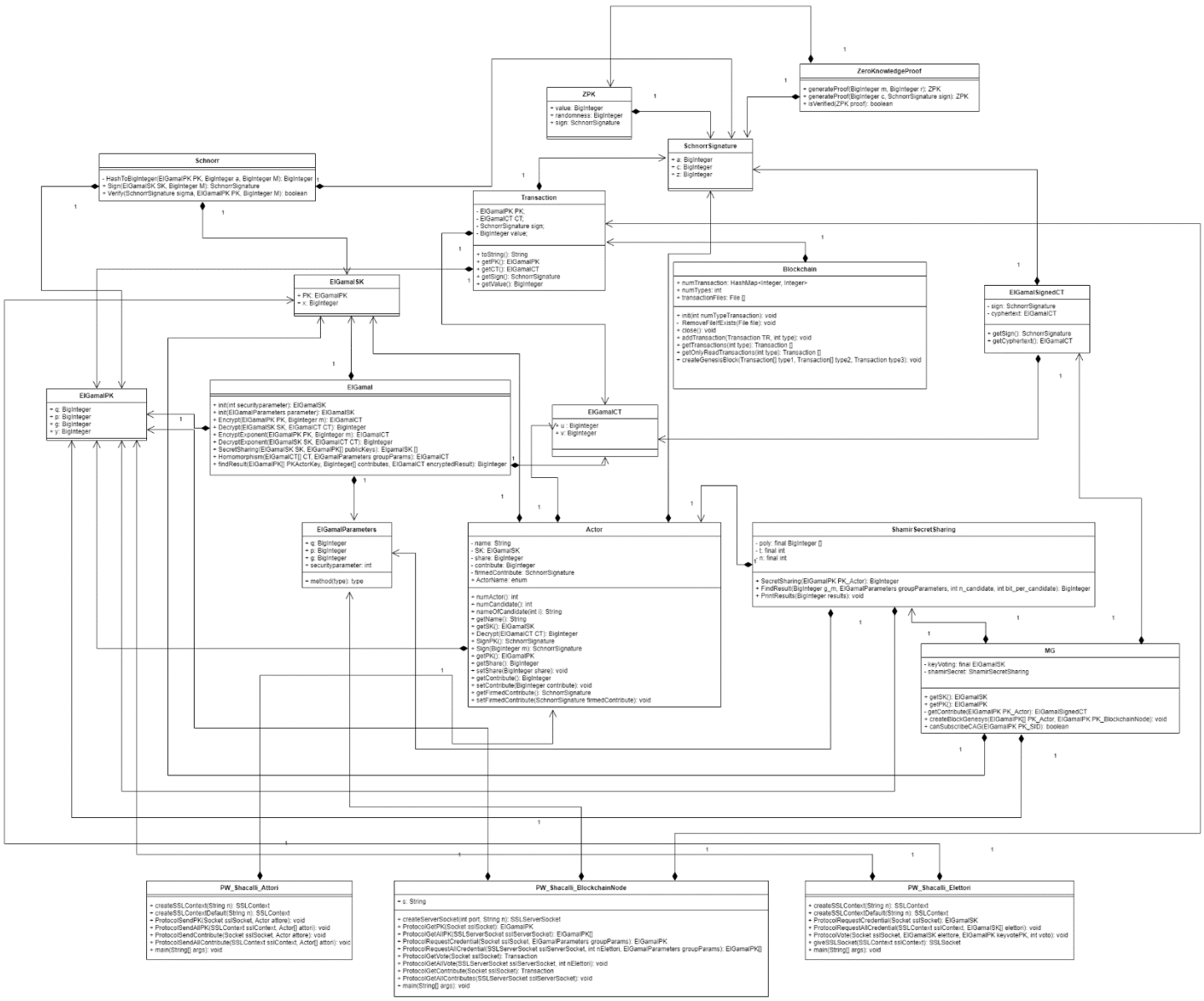
Descrizione dell'architettura software

L'architettura software è stata realizzata tenendo conto di tutti gli algoritmi e gli strumenti crittografici descritti in precedenza. Ogni strumento è stato implementato usando una classe apposita. Tali classi sono le seguenti:

- ***Actor***: contiene le informazioni sui vari attori, tra cui la chiave segreta, i contributi e il nome.
- ***Blockchain***: permette di gestire e leggere le transazioni sulla blockchain.
- ***ElGamal***: è la responsabile della creazione di coppie di chiavi di ElGamal, della cifratura e della decifratura di messaggi e, infine, della decifratura di un cyphertext usando i contributi dei vari attori.
- ***ElGamalCT***: rappresenta il cyphertext di ElGamal.
- ***ElGamalPK***: rappresenta la chiave pubblica di ElGamal
- ***ElGamalParameters***: contiene tutti i parametri del gruppo per poter istanziare le diverse chiavi di ElGamal.
- ***ElGamalSK***: rappresenta la chiave segreta di ElGamal.
- ***ElGamalSignedCT***: rappresenta la coppia cyphertext-firma.
- ***MG***: tale classe rappresenta la macchina di giustizia, la quale genera la chiave pubblica e segreta della votazione, il polinomio, le share e la creazione del blocco genesi. Inoltre, emula anche la richiesta al CAG per apprendere se un cittadino è ammesso al voto.
- ***PW_Shacalli_Attori***: rappresenta la classe principale per la gestione dei vari attori.
- ***PW_Shacalli_BlockchainNode***: rappresenta la classe principale per la gestione dei blockchain node.
- ***PW_Shacalli_Elettori***: rappresenta la classe principale per la gestione degli elettori.
- ***Schnorr***: consente di creare una firma digitale di Schnorr e di verificarne la validità.
- ***SchnorrSignature***: rappresenta la firma digitale di Schnorr.
- ***ShamirSecretSharing***: fornisce i metodi per la generazione del polinomio, il calcolo dei punti di quest'ultimo e di effettuare la bruteforce per determinare il risultato delle elezioni.

- ***Transaction***: rappresenta tutte le possibili transazioni ammissibili sulla blockchain.
- ***Utils***: contiene alcuni metodi utili per il funzionamento dei vari tool.
- ***ZKP***: rappresenta le diverse ZK Proof.
- ***ZeroKnowledgeProof***: consente di generare una ZK Proof e di verificarne la validità.

UML delle classi



Snippet di codice rilevanti

Classe Blockchain.java

```
1.  /*
2.   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
    license
3.   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4.   */
5.
6.  import java.io.EOFException;
7.  import java.io.File;
8.  import java.io.FileInputStream;
9.  import java.io.FileNotFoundException;
10. import java.io.FileOutputStream;
11. import java.io.IOException;
12. import java.io.ObjectInputStream;
13. import java.io.ObjectOutputStream;
14. import java.util.ArrayList;
15. import java.util.HashMap;
16.
17. /**
18.  *
19.  * @author Shacalli
20.  */
21. public class Blockchain {
22.
23.     static HashMap<Integer, Integer> numTransaction;
24.     static int numTypes;
25.     static File[] transactionFiles;
26.
27.     public static void init(int numTypeTransaction) throws FileNotFoundException, IOException {
28.         transactionFiles = new File[numTypeTransaction];
29.         for (int i = 0; i < numTypeTransaction; i++) {
30.             String path = "C:\\blockchain\\";
31.             String nameFile = path + "type" + (i + 1) + ".txt";
32.             transactionFiles[i] = new File(nameFile);
33.             RemoveFileIfExists(transactionFiles[i]);
34.         }
35.
36.         numTransaction = new HashMap();
37.         for (int i = 0; i < numTypeTransaction; i++) {
38.             numTransaction.put(i + 1, 0);
39.         }
40.         numTypes = numTypeTransaction;
41.     }
42.
43.     public static void initReading() throws FileNotFoundException, IOException {
44.         transactionFiles = new File[8];
45.         for (int i = 0; i < 8; i++) {
46.             String path = "C:\\blockchain\\";
47.             String nameFile = path + "type" + (i + 1) + ".txt";
48.             transactionFiles[i] = new File(nameFile);
49.         }
50.     }
51.
52.     private static void RemoveFileIfExists(File file) {
53.         if (file.exists()) {
54.             file.delete();
55.         }
56.     }
57.
58.     public static void close() throws IOException {
59.         for (int i = 0; i < transactionFiles.length; i++) {
60.             RemoveFileIfExists(transactionFiles[i]);
61.         }
62.     }
63.
64.     public static void addTransaction(Transaction TR, int type) throws Exception {
65.         Transaction[] oldTransaction = getTransactions(type);
66.
67.         ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(transactionFiles[type
    - 1]));
68.
69.         if (oldTransaction != null) {
70.             for (int i = 0; i < oldTransaction.length; i++) {
71.                 out.writeObject(oldTransaction[i]);
72.             }
73.         }
74.
75.         out.writeObject(TR);
76.
77.         Integer num = numTransaction.get(type) + 1;
78.         numTransaction.put(type, num);
```

```
79.         out.close();
80.     }
81.
82.     public static Transaction[] getTransactions(int type) throws FileNotFoundException,
    IOException, ClassNotFoundException {
83.         try ( ObjectInputStream stream = new ObjectInputStream(new
    FileInputStream(transactionFiles[type - 1]))) {
84.             Transaction[] trs = new Transaction[numTransaction.get(type)];
85.             for (int i = 0; i < trs.length; i++) {
86.                 trs[i] = (Transaction) stream.readObject();
87.             }
88.             return trs;
89.         } catch (FileNotFoundException e) {
90.             return null;
91.         }
92.     }
93.
94.     public static Transaction[] getOnlyReadTransactions(int type) throws FileNotFoundException,
    IOException, ClassNotFoundException {
95.         ArrayList<Transaction> array = new ArrayList<Transaction>();
96.
97.         ObjectInputStream stream = new ObjectInputStream(new FileInputStream(transactionFiles[type
    - 1]));
98.
99.         while (true) {
100.            try {
101.                Transaction tran = (Transaction) stream.readObject();
102.                array.add(tran);
103.            } catch (EOFException e) {
104.                break;
105.            }
106.        }
107.        stream.close();
108.        return array.toArray(new Transaction[0]);
109.    }
110.
111.    public static void createGenesisBlock(Transaction[] type1, Transaction[] type2, Transaction
    type3) throws IOException, Exception {
112.        for (int i = 0; i < type1.length; i++) {
113.            addTransaction(type1[i], 1);
114.        }
115.        for (int i = 0; i < type2.length; i++) {
116.            addTransaction(type2[i], 2);
117.        }
118.        addTransaction(type3, 3);
119.    }
120.}
121.
```

Classe ElGamal.java

```
1.  /*
2.   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
    license
3.   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4.   */
5.
6.  import java.math.BigInteger;
7.  import java.security.SecureRandom;
8.  import org.bouncycastle.asn1.oiw.ElGamalParameter;
9.
10. /**
11.  *
12.  * @author Shacalli
13.  */
14. public class ElGamal {
15.
16.     public static ElGamalSK init(int securityparameter) {
17.         BigInteger p, q, g, y;
18.         SecureRandom src = new SecureRandom();
19.         while (true) {
20.             q = BigInteger.probablePrime(securityparameter, src);
21.             p = BigInteger.valueOf(2).multiply(q).add(BigInteger.ONE);
22.             if (p.isProbablePrime(50)) {
23.                 break;
24.             }
25.         }
26.
27.         g = new BigInteger("4");
28.         BigInteger x = new BigInteger(securityparameter, src).mod(q);
29.         y = g.modPow(x, p);
30.         ElGamalPK PK = new ElGamalPK(q, p, g, y, securityparameter);
31.         return new ElGamalSK(x, PK);
32.     }
```

```
33.
34.     public static ElGamalSK init(ElGamalParameters parameter) {
35.         SecureRandom src = new SecureRandom();
36.         BigInteger x = new BigInteger(parameter.securityparameter, src).mod(parameter.q);
37.         BigInteger y = parameter.g.modPow(x, parameter.p);
38.         ElGamalPK PK = new ElGamalPK(parameter.q, parameter.p, parameter.g, y,
    parameter.securityparameter);
39.         return new ElGamalSK(x, PK);
40.     }
41.
42.     public static ElGamalCT Encrypt(ElGamalPK PK, BigInteger m) {
43.         SecureRandom sc = new SecureRandom();
44.         BigInteger r = new BigInteger(PK.securityparameter, sc).mod(PK.q);
45.         BigInteger v = m.multiply(PK.y.modPow(r, PK.p));
46.         v = v.mod(PK.p);
47.         BigInteger u = PK.g.modPow(r, PK.p);
48.         return new ElGamalCT(u, v);
49.     }
50.
51.     public static BigInteger Decrypt(ElGamalSK SK, ElGamalCT CT) {
52.         BigInteger tmp = CT.u.modPow(SK.x, SK.PK.p);  // tmp=u^s mod p
53.         tmp = tmp.modInverse(SK.PK.p);
54.
55.         BigInteger M = tmp.multiply(CT.v).mod(SK.PK.p); // M=tmp*C mod p
56.         return M;
57.     }
58.
59.     public static ElGamalCT EncryptExponent(ElGamalPK PK, BigInteger m) {
60.         SecureRandom sc = new SecureRandom();
61.         BigInteger M = PK.g.modPow(m, PK.p); // M=g^m mod p
62.         BigInteger r = new BigInteger(PK.securityparameter, sc).mod(PK.q);
63.         BigInteger v = M.multiply(PK.y.modPow(r, PK.p)).mod(PK.p);
64.         BigInteger u = PK.g.modPow(r, PK.p);
65.         return new ElGamalCT(u, v);
66.     }
67.
68.     public static BigInteger DecryptExponent(ElGamalSK SK, ElGamalCT CT) {
69.         BigInteger tmp = CT.u.modPow(SK.x, SK.PK.p).modInverse(SK.PK.p);
70.         BigInteger res = tmp.multiply(CT.v).mod(SK.PK.p);
71.
72.         BigInteger M = new BigInteger("0");
73.         while (true) {
74.             if (SK.PK.g.modPow(M, SK.PK.p).compareTo(res) == 0) {  // if g^M=res stop and return M
75.                 return M;
76.             }
77.             M = M.add(BigInteger.ONE); // otherwise M++
78.         }
79.     }
80.
81.     public static ElGamalCT Homomorphism(ElGamalCT[] CT, ElGamalParameters groupParams) {
82.         BigInteger u = BigInteger.ONE;
83.         BigInteger v = BigInteger.ONE;
84.
85.         for (ElGamalCT ct : CT) {
86.             u = u.multiply(ct.u).mod(groupParams.p);
87.             v = v.multiply(ct.v).mod(groupParams.p);
88.         }
89.
90.         return new ElGamalCT(u, v);
91.     }
92.
93.     public static BigInteger findResult(ElGamalPK[] PKActorKey, BigInteger[] contributes,
    ElGamalCT encryptedResult) {
94.         ElGamalParameters groupParams = new ElGamalParameters(PKActorKey[0]);
95.
96.         BigInteger w = BigInteger.ONE;
97.         int i = 0;
98.         for (ElGamalPK i_PK : PKActorKey) {
99.             BigInteger dj = BigInteger.ONE;
100.            for (ElGamalPK j_PK : PKActorKey) {
101.                if (j_PK.y != i_PK.y) {
102.                    dj = dj.multiply(j_PK.y.subtract(i_PK.y).modInverse(groupParams.q));
103.                    dj = j_PK.y.multiply(dj).mod(groupParams.q);
104.                }
105.            }
106.            w = w.multiply(contributes[i].modPow(dj.mod(groupParams.q),
    groupParams.p)).mod(groupParams.p);
107.            i++;
108.        }
109.
110.        BigInteger g_m =
    w.modInverse(groupParams.p).multiply(encryptedResult.v).mod(groupParams.p);
111.
112.        BigInteger plaintextResult = ShamirSecretSharing.FindResult(g_m, groupParams,
    PKActorKey.length - 2, PKActorKey.length);
113.        return plaintextResult;
114.    }}
```

Classe MG.java

```
1.  import java.io.File;
2.  import java.io.IOException;
3.  import java.math.BigInteger;
4.
5.  /**
6.   *
7.   * @author Shacalli
8.   */
9.  public class MG {
10.
11.     private final ElGamalSK keyVoting;
12.     private final ShamirSecretSharing shamirSecret;
13.
14.     public MG(ElGamalParameters groupParams, int numActors) {
15.         this.keyVoting = ElGamal.init(groupParams);
16.         this.shamirSecret = new ShamirSecretSharing(numActors - 2, numActors, groupParams,
    keyVoting.x);
17.     }
18.
19.     /**
20.      * Get the value of SK
21.      *
22.      * @return the value of SK
23.      */
24.     private ElGamalSK getSK() {
25.         return keyVoting;
26.     }
27.
28.     /**
29.      * Get the value of PK
30.      *
31.      * @return the value of PK
32.      */
33.     public ElGamalPK getPK() {
34.         return getSK().PK;
35.     }
36.
37.     /**
38.      * Get the value of PK
39.      *
40.      * @return the value of PK
41.      */
42.     private ElGamalSignedCT getContribute(ElGamalPK PK_Actor) {
43.         BigInteger secret = this.shamirSecret.SecretSharing(PK_Actor);
44.         return new ElGamalSignedCT(ElGamal.Encrypt(PK_Actor, secret), Schnorr.Sign(keyVoting,
    secret));
45.     }
46.
47.     public void createBlockGenesys(ElGamalPK[] PK_Actor, ElGamalPK PK_BlockchainNode) throws
    Exception {
48.         ElGamalSignedCT[] encryptedContribute = new ElGamalSignedCT[PK_Actor.length];
49.
50.         for (int i = 0; i < PK_Actor.length; i++) {
51.             encryptedContribute[i] = this.getContribute(PK_Actor[i]);
52.             System.out.println(encryptedContribute[i].getCyphertext().v + "\t-> Contributo cifrato
    " + Actor.ActorName.values()[i]);
53.         }
54.
55.         Blockchain.init(8);
56.
57.         Transaction transactionKeyVotePK = new Transaction(keyVoting.PK);
58.
59.         Transaction[] transactionBlockchainNodePK = new Transaction[1];
60.         transactionBlockchainNodePK[0] = new Transaction(PK_BlockchainNode);
61.
62.         Transaction[] transactionActorPK = new Transaction[PK_Actor.length];
63.         for (int i = 0; i < PK_Actor.length; i++) {
64.             transactionActorPK[i] = new Transaction(PK_Actor[i],
    encryptedContribute[i].getCyphertext(), encryptedContribute[i].getSign());
65.         }
66.
67.         Blockchain.createGenesisBlock(transactionActorPK, transactionBlockchainNodePK,
    transactionKeyVotePK);
68.
69.         System.out.println("Blocco genesi creato\n");
70.     }
71.
72.     public static boolean canSubscribeCAG(ElGamalPK PK_SID) {
73.         //Il Centro di Autorità Governativo mi indica se quel cittadino
74.         //può essere ammesso al voto
75.
76.         return true;
77.     }}
```

Classe Schnorr.java

```
1.  /*
2.   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
    license
3.   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4.   */
5.
6.  import java.math.BigInteger;
7.  import java.security.MessageDigest;
8.  import java.security.SecureRandom;
9.
10. /**
11.  *
12.  * @author Shacalli
13.  */
14. public class Schnorr {
15.
16.     private static BigInteger HashToBigInteger(ElGamalPK PK, BigInteger a, BigInteger M) {
17.         String msg = PK.g.toString() + PK.y.toString() + a.toString() + M.toString(); // Hash
    PK+a+M to a BigInteger
18.         try {
19.             MessageDigest h = MessageDigest.getInstance("SHA256"); // hash a String using
    MessageDigest class
20.             h.update(Utils.toByteArray(msg));
21.             BigInteger e = new BigInteger(h.digest());
22.             return e.mod(PK.q);
23.         } catch (Exception E) {
24.             E.printStackTrace();
25.         }
26.
27.         BigInteger e = new BigInteger("0");
28.         return e;
29.     }
30.
31.     public static SchnorrSignature Sign(ElGamalSK SK, BigInteger M) {
32.         SecureRandom sc = new SecureRandom(); // generate secure random source
33.         BigInteger r = new BigInteger(SK.PK.securityparameter, sc); // choose random r
34.         BigInteger a = SK.PK.g.modPow(r, SK.PK.p); // a=g^r mod p
35.         BigInteger c = HashToBigInteger(SK.PK, a, M); // c=H(PK,a,M)
36.         BigInteger z = r.add(c.multiply(SK.x).mod(SK.PK.q)).mod(SK.PK.q); // z=r+cs mod q
37.         return new SchnorrSignature(a, c, z); // (a,e,z) is the signature of M
38.     }
39.
40.     public static boolean Verify(SchnorrSignature sigma, ElGamalPK PK, BigInteger M) {
41.         BigInteger c2 = HashToBigInteger(PK, sigma.a, M); // we compute the the challenge c
    c=H(PK,a,M)
42.         BigInteger tmp = sigma.a.multiply(PK.y.modPow(c2, PK.p)).mod(PK.p); // tmp=ay^c2
43.
44.         if (tmp.compareTo(PK.g.modPow(sigma.z, PK.p)) == 0) { // compare tmp with g^z mod p
45.             return true;
46.         }
47.         return false;
48.     }
49. }
50.
```

Classe ShamirSecretSharing.java

```
1.  import java.math.BigInteger;
2.  import java.security.SecureRandom;
3.
4.  /**
5.   *
6.   * @author Shacalli
7.   */
8.  public class ShamirSecretSharing {
9.
10.     private final BigInteger[] poly;
11.
12.     private final int t;
13.     private final int n;
14.
15.     public ShamirSecretSharing(int t, int n, ElGamalParameters GroupParam, BigInteger secret) {
16.         this.t = t;
17.         this.n = n;
18.
19.         SecureRandom src = new SecureRandom(); // oggetto secure random
20.
21.         // Nuovo polinomio
22.         poly = new BigInteger[t];
23.
```

```
24.         //Generazione del polinomio
25.         poly[0] = secret;
26.
27.         String out = "Polinomio: " + poly[0].toString();
28.
29.         for (int i = 1; i < t; i++) {
30.             poly[i] = new BigInteger(GroupParam.securityparameter, src).mod(GroupParam.q);
31.             out = out + " " + poly[i].toString() + "x^" + i;
32.         }
33.
34.         System.out.println(out);
35.     }
36.
37.     public BigInteger SecretSharing(ElGamalPK PK_Actor) {
38.         BigInteger share; // parti del segreto per ogni attore
39.
40.         share = this.poly[0];
41.         for (int j = 1; j < this.t; j++) {
42.             // a*x^j
43.             share = share.add(this.poly[j].multiply(PK_Actor.y.pow(j)).mod(PK_Actor.q));
44.         }
45.         share = share.mod(PK_Actor.q);
46.
47.         return share;
48.     }
49.
50.     public static BigInteger FindResult(BigInteger g_m, ElGamalParameters groupParameters, int
    n_candidate, int bit_per_candidate) {
51.         BigInteger i = BigInteger.ZERO;
52.         BigInteger limit = BigInteger.valueOf(2).pow(n_candidate*bit_per_candidate);
53.
54.         for(; i.compareTo(limit) < 0; i=i.add(BigInteger.ONE)){
55.             if(groupParameters.g.modPow(i, groupParameters.p).compareTo(g_m) == 0){
56.                 return i;
57.             }
58.         }
59.         return null;
60.     }
61.
62.     static void PrintResults(BigInteger results){
63.         //4 candidati, 4 bit per candidato
64.         BigInteger maschered = BigInteger.valueOf(2).pow(4).add((BigInteger.ONE.negate()));
65.
66.         System.out.println("\nRISULTATI ELEZIONI\n");
67.
68.         for(int i=0; i<(Actor.numActor()-3)/2; i++){
69.             System.out.println(Actor.ActorName.values()[i] + ":
    "+results.and(maschered).shiftRight(i*4)+" voti");
70.             maschered = maschered.multiply(BigInteger.valueOf(2).pow(4));
71.         }
72.     }
73. }
74.
```

Classe ZeroKnowledgeProof.java

```
1.  /*
2.   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
    license
3.   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4.   */
5.  import java.math.BigInteger;
6.
7.  /**
8.   *
9.   * @author Shacalli
10.  */
11. public class ZeroKnowledgeProof {
12.     public static ZKP generateProof(BigInteger m, BigInteger r){
13.         return new ZKP(m,r);
14.     }
15.
16.     public static ZKP generateProof(BigInteger c,SchnorrSignature sign){
17.         return new ZKP(c,sign);
18.     }
19.
20.     public static boolean isVerified(ZKP proof){
21.         return true;
22.     }
23. }
```

Descrizione del flusso di codice e dei protocolli

Di seguito verrà descritto il flusso di codice tra le varie entità in gioco, descrivendo i protocolli di comunicazione utilizzati tra essi.

Inizialmente viene eseguito il codice del blockchain node, il quale avvia un server SSL sulla porta 4000.

PW_Shacalli_BlockChainNode.java

```
static SSLServerSocket createServerSocket(int port, String n) throws Exception {
    String truststore = "truststore" + n + ".jks";
    String trustpass = n + "pass";

    System.setProperty("javax.net.ssl.trustStore", truststore);
    System.setProperty("javax.net.ssl.trustStorePassword", trustpass);

    String keystore = "keystore" + n + ".jks";
    String keypass = n + "pass";
    System.setProperty("javax.net.ssl.keyStore", keystore);
    //specifing the password of the trustStore file
    System.setProperty("javax.net.ssl.keyStorePassword", keypass);

    SSLServerSocketFactory sslServerSocketfactory = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
    SSLServerSocket sslServerSocket = (SSLServerSocket) sslServerSocketfactory.createServerSocket(port);
    sslServerSocket.setNeedClientAuth(true);

    return sslServerSocket;
}
```

Gli attori, invece, effettuano una nuova richiesta di connessione al blockchain node.

PW_Shacalli_Attori.java

```
static SSLContext createSSLContext(String n) throws Exception {
    String truststore = "truststore" + n + ".jks";
    String trustpass = n + "pass";

    System.setProperty("javax.net.ssl.trustStore", truststore);
    //specifing the password of the trustStore file
    System.setProperty("javax.net.ssl.trustStorePassword", trustpass);

    KeyManagerFactory keyFact = KeyManagerFactory.getInstance("SunX509");
    KeyStore clientStore = KeyStore.getInstance("JKS");

    String keystore = "keystore" + n + ".jks";
    String keypass = n + "pass";

    clientStore.load(new FileInputStream(keystore), keypass.toCharArray());

    keyFact.init(clientStore, keypass.toCharArray());

    SSLContext sslContext = SSLContext.getInstance("TLS");
    sslContext.init(keyFact.getKeyManagers(), null, null);

    return sslContext;
}
```

In seguito, il blockchain node si mette in attesa delle varie PK degli attori.

PW_Shacalli_BlockChainNode.java

```
static ElGamalPK[] ProtocolGetAllPK(SSLServerSocket sslServerSocket) throws Exception {
    ElGamalPK[] attori = new ElGamalPK[Actor.numActor()];
    for (int i = 0; i < Actor.numActor(); i++) {
        SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept();

        System.out.println(Actor.nameOfCandidate(i) + " connesso");
        attori[i] = ProtocolGetPK(sslSocket);
    }
    return attori;
}

static ElGamalPK ProtocolGetPK(Socket sslSocket) throws Exception {
    System.out.println("session GETPK started.");
    ObjectInputStream inputStream = new ObjectInputStream(sslSocket.getInputStream());

    ElGamalPK PK = (ElGamalPK) inputStream.readObject();
    SchnorrSignature Sign = (SchnorrSignature) inputStream.readObject();

    sslSocket.close();
```

```
    if (Schnorr.Verify(Sign, PK, PK.y)) {
        System.out.println("PK dell'attore ricevuta: " + PK.y);
        System.out.println("session GETPK closed.\n");
        return PK;
    }

    System.out.println("session GETPK closed.");
    System.out.println("PK dell'attore non conforme alla firma.\n");
    return null;
}
```

Gli attori, dopo essersi connessi al blockchain node, inviano le proprie PK.

PW_Shacalli_Attori.java

```
static void ProtocolSendAllPK(SSLContext sslContext, Actor[] attori) throws Exception {
    int serverPort = 4000;
    String serverName = "BN01";
    for (int i = 0; i < Actor.numActor(); i++) {
        SSLSocketFactory sslClientSocket = sslContext.getSocketFactory();
        SSLSocket sslSocket = (SSLSocket) sslClientSocket.createSocket(serverName, serverPort);
        sslSocket.startHandshake();

        ProtocolSendPK(sslSocket, attori[i]);

        sslSocket.close();
    }
}

static void ProtocolSendPK(Socket sslSocket, Actor attore) throws Exception {
    ObjectOutputStream outputStream = new ObjectOutputStream(sslSocket.getOutputStream());

    //Invio della PK dell'attore
    outputStream.writeObject(attore.getPK());

    //Invio la firma della PK dell'attore
    outputStream.writeObject(attore.SignPK());
}
```

Il blockchain node, dopo aver ricevuto le PK, invoca la macchina della giustizia MG affinché possa generare il segreto e creare il blocco genesi. In seguito, il blockchain node si mette in attesa dei vari cittadini che intendono fare richiesta delle credenziali di voto.

PW_Shacalli_BlockchainNode.java

```
static ElGamalPK[] ProtocolRequestAllCredential(SSLServerSocket sslServerSocket, int nElettori,
ElGamalParameters groupParams) throws Exception {
    ElGamalPK[] elettori = new ElGamalPK[nElettori];
    for (int i = 0; i < nElettori; i++) {
        SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept();

        System.out.println("Cittadino connesso");
        elettori[i] = ProtocolRequestCredential(sslSocket, groupParams);

        Blockchain.addTransaction(new Transaction(elettori[i]), 4);
    }
    return elettori;
}

static ElGamalPK ProtocolRequestCredential(Socket sslSocket, ElGamalParameters groupParams) throws
Exception {
    System.out.println("session GETPK Voters started.");

    ObjectOutputStream outputStream = new ObjectOutputStream(sslSocket.getOutputStream());
    ObjectInputStream inputStream = new ObjectInputStream(sslSocket.getInputStream());

    //Invio i parametri del gruppo
    outputStream.writeObject(groupParams);

    //Ottengo la PK del SID
    ElGamalPK PK_SID = (ElGamalPK) inputStream.readObject();

    //Ottengo la sign(PK_SID)
    SchnorrSignature SID_sign = (SchnorrSignature) inputStream.readObject();

    if (Schnorr.Verify(SID_sign, PK_SID, PK_SID.y) == false) {
        sslSocket.close();
        System.out.println("session GETPK Voters closed.");
        System.out.println("PK del cittadino non conforme alla firma.\n");
        return null;
    }
```

```
    if (MG.canSubscribeCAG(PK_SID) == false) {
        sslSocket.close();
        System.out.println("Cittadino non ammesso al voto!");
        System.out.println("session GETPK Voters closed.\n");
        return null;
    }

    //Ottengo la PK
    ElGamalPK PK = (ElGamalPK) inputStream.readObject();

    //Ottengo la sign(message)
    SchnorrSignature sign = (SchnorrSignature) inputStream.readObject();

    sslSocket.close();

    if (Schnorr.Verify(sign, PK, PK.y)) {
        System.out.println("PK dell'elettore generata: " + PK.y);
        System.out.println("session GETPK Voters closed.\n");
        return PK;
    }

    System.out.println("session GETPK Voters closed.");
    System.out.println("PK dell'elettore non conforme alla firma.\n");

    return null;
}
```

Gli elettori che intendono fare richiesta avviano una connessione al blockchain node e, dopo essersi identificati, fanno richiesta delle credenziali di voto.

PW_Shacalli_Elettori.java

```
static void ProtocolRequestAllCredential(SSLContext sslContext, ElGamalSK[] elettori) throws Exception {
    int serverPort = 4000;
    String serverName = "BN01";
    for (int i = 0; i < elettori.length; i++) {
        SSLSocketFactory sslClientSocket = sslContext.getSocketFactory();
        SSLSocket sslSocket = (SSLSocket) sslClientSocket.createSocket(serverName, serverPort);
        sslSocket.startHandshake();

        System.out.println("Richiesta di ammissione al voto del cittadino "+(i+1));

        elettori[i] = ProtocolRequestCredential(sslSocket);

        sslSocket.close();
    }
}

static ElGamalSK ProtocolRequestCredential(Socket sslSocket) throws Exception {
    ObjectOutputStream outputStream = new ObjectOutputStream(sslSocket.getOutputStream());
    ObjectInputStream inputStream = new ObjectInputStream(sslSocket.getInputStream());

    //Ottengo i parametri del gruppo
    ElGamalParameters groupParams = (ElGamalParameters) inputStream.readObject();

    //Crea le chiavi
    ElGamalSK SID = ElGamal.init(groupParams);
    ElGamalSK newKey = ElGamal.init(groupParams);

    //Invio il SID
    outputStream.writeObject(SID.PK);

    //Firmo la PK con SK e invio
    SchnorrSignature SID_sign = Schnorr.Sign(SID, SID.PK.y);
    outputStream.writeObject(SID_sign);

    //Invio la PK
    outputStream.writeObject(newKey.PK);

    //Firmo la PK con SK e invio
    SchnorrSignature sign = Schnorr.Sign(newKey, newKey.PK.y);
    outputStream.writeObject(sign);

    return newKey;
}
```

Al termine delle diverse richieste delle credenziali di voto, si passa alla fase successiva, ovvero la fase di voto. Il blockchain node si mette in attesa dei vari elettori in modo tale da consentire l'invio dei voti.

PW_Shacalli_BlockchainNode.java

```
static void ProtocolGetAllVote(SSLServerSocket sslServerSocket, int nElettori) throws Exception {
    ElGamalPK[] elettori = new ElGamalPK[nElettori];
    for (int i = 0; i < nElettori; i++) {
        SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept();

        System.out.println("Elettore connesso");
        Transaction tr = ProtocolGetVote(sslSocket);

        Blockchain.addTransaction(tr, 5);
    }
}

static Transaction ProtocolGetVote(Socket sslSocket) throws Exception {
    System.out.println("session Get Vote started.");

    ObjectOutputStream outputStream = new ObjectOutputStream(sslSocket.getOutputStream());
    ObjectInputStream inputStream = new ObjectInputStream(sslSocket.getInputStream());

    //Ottengo la PK
    ElGamalPK PK = (ElGamalPK) inputStream.readObject();

    //Controllo se l'elettore è ammesso al voto
    Boolean canVote = false;
    for (Transaction tr : Blockchain.getOnlyReadTransactions(4)) {
        if (tr.getPK().y.compareTo(PK.y) == 0) {
            canVote = true;
        }
    }

    if (canVote == false) {
        sslSocket.close();
        System.out.println("session Get Vote closed.\n");
        System.out.println("Elettore non ammesso al voto!\n");
        return null;
    }

    //Ottengo il voto, TIMESTAMP, sign(voto|TIMESTAMP)
    ElGamalCT votoCifrato = (ElGamalCT) inputStream.readObject();
    Long timestamp = inputStream.readLong();
    SchnorrSignature sign = (SchnorrSignature) inputStream.readObject();

    //Calcolo voto|TIMESTAMP
    BigInteger votePlusTimestamp = new BigInteger(String.valueOf(votoCifrato.v) +
String.valueOf(timestamp));

    //Ottengo la ZKP del formato del voto
    ZKP zkpVote = (ZKP) inputStream.readObject();

    sslSocket.close();

    if (Schnorr.Verify(sign, PK, votePlusTimestamp)) {
        if (ZeroKnowledgeProof.isVerified(zkpVote)) {
            //Attendo un tempo r
            long r = 1;
            //TimeUnit.SECONDS.sleep(r);
            TimeUnit.MILLISECONDS.sleep(r);
            System.out.println("Voto espresso correttamente");
            System.out.println("session Get Vote closed.\n");
            return new Transaction(PK, votoCifrato);
        }
        System.out.println("session Get Vote closed.\n");
        System.out.println("ZKP non verificata!\n");
        return null;
    }

    System.out.println("session Get Vote closed.\n");
    System.out.println("Firma non verificata!\n");
    return null;
}
```

Gli elettori, invece, inviano il voto e attendono il risultato delle elezioni.

PW_Shacalli_Elettori.java

```
static void ProtocolVote(Socket sslSocket, ElGamalSK elettore, ElGamalPK keyvotePK, int voto) throws
Exception {
    ObjectOutputStream outputStream = new ObjectOutputStream(sslSocket.getOutputStream());
    ObjectInputStream inputStream = new ObjectInputStream(sslSocket.getInputStream());

    //Invio PK dell'elettore
    outputStream.writeObject(elettore.PK);

    BigInteger votoBinario = BigInteger.ONE.shiftLeft((voto - 1) * 4);
```

```
    ElGamalCT votoCifrato = new ElGamalCT(ElGamal.EncryptExponent(keyvotePK, votoBinario));

    //TIMESTAMP
    Timestamp time = new Timestamp(System.currentTimeMillis());

    long timestamp = time.getDateTime();

    //voto|TIMESTAMP
    BigInteger votePlusTimestamp = new BigInteger(String.valueOf(votoCifrato.v) +
String.valueOf(timestamp));

    //Invio voto, TIMESTAMP, sign(voto|TIMESTAMP)
    outputStream.writeObject(votoCifrato);
    outputStream.writeLong(timestamp);
    outputStream.writeObject(Schnorr.Sign(elettore, votePlusTimestamp));

    //ZKP del formato del voto
    ZKP zkpVote = ZeroKnowledgeProof.generateProof(votoCifrato.v, BigInteger.ONE);
    outputStream.writeObject(zkpVote);

    System.out.println("\nVoto inviato!!!\n");
}
```

Al termine dell'invio dei voti, il blockchain node crea il cyphertext somma e lo pubblica in blockchain. Non appena disponibile, gli attori calcolano il proprio contributo in riferimento al cyphertext somma e lo inviano al blockchain node.

PW_Shacalli_Attori.java

```
static void ProtocolSendAllContribute(SSLContext sslContext, Actor[] attori) throws Exception {
    int serverPort = 4000;
    String serverName = "BN01";
    for (int i = 0; i < Actor.numActor(); i++) {
        SSLSocketFactory sslClientSocket = sslContext.getSocketFactory();
        SSLSocket sslSocket = (SSLSocket) sslClientSocket.createSocket(serverName, serverPort);
        sslSocket.startHandshake();

        ProtocolSendContribute(sslSocket, attori[i]);

        sslSocket.close();
    }
}

static void ProtocolSendContribute(Socket sslSocket, Actor attore) throws Exception {
    ObjectOutputStream outputStream = new ObjectOutputStream(sslSocket.getOutputStream());

    //Invio la PK, il contributo cifrato e la firma dello stesso
    outputStream.writeObject(attore.getPK());
    outputStream.writeObject(attore.getContribute());
    outputStream.writeObject(attore.Sign(attore.getContribute()));

    //ZKP del formato del voto
    ZKP zkpVote = ZeroKnowledgeProof.generateProof(attore.getContribute(), attore.getFirmedContribute());
    outputStream.writeObject(zkpVote);
}
```

PW_Shacalli_BlockchainNode.java

```
static void ProtocolGetAllContributes(SSLServerSocket sslServerSocket) throws Exception {
    BigInteger[] contributiCifrati = new BigInteger[Actor.numActor()];
    for (int i = 0; i < Actor.numActor(); i++) {
        SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept();
        System.out.println("Attore connesso");

        Transaction tr = ProtocolGetContribute(sslSocket);

        Blockchain.addTransaction(tr, 7);
    }
}

static Transaction ProtocolGetContribute(Socket sslSocket) throws Exception {
    System.out.println("session Get Contribute started.");
    ObjectInputStream inputStream = new ObjectInputStream(sslSocket.getInputStream());

    //Ottengo la PK, il contributo cifrato e la firma dello stesso
    ElGamalPK PK = (ElGamalPK) inputStream.readObject();
    BigInteger contributoCifrato = (BigInteger) inputStream.readObject();
    SchnorrSignature sign = (SchnorrSignature) inputStream.readObject();

    //Ottengo la ZKP del contributo
    ZKP zkpContribute = (ZKP) inputStream.readObject();

    sslSocket.close();
```

```
    if (Schnorr.Verify(sign, PK, contributoCifrato)) {
        if (ZeroKnowledgeProof.isVerified(zkpContribute)) {
            System.out.println("Contributo ricevuto correttamente");
            System.out.println("session Get Contribute closed.\n");
            return new Transaction(contributoCifrato);
        }
        System.out.println("session Get Contribute closed.\n");
        System.out.println("ZKP non verificata!\n");
        return null;
    }

    System.out.println("session Get Contribute closed.\n");
    System.out.println("Firma non verificata!\n");

    return null;
}
```

Il blockchain node ottiene i vari contributi e avvia la fase di decifratura del risultato. Appena compiuta la decifratura, esso pubblica in blockchain il risultato in chiaro dando fine alle elezioni.

Simulazione del prototipo

La simulazione che verrà mostrata in seguito ha le seguenti entità coinvolte:

- 11 attori: 4 candidati, 4 persone di legge, il presidente della provincia, il presidente della regione e il presidente della Repubblica;
- 1 blockchain node: BN01;
- 10 elettori.

All'avvio del blockchain node, esso attende il collegamento dei vari attori che custodiranno le share del segreto:

PW_Shacalli_BlockchainNode

```
Avvio server BlockchainNode

Avviato

Inizio raccoglimento PK degli attori
```

Quando i vari attori si collegheranno, inviano le proprie PK e attendono fino al momento in cui si devono riunire per decifrare il risultato delle elezioni:

PW_Shacalli_Attori

```
Invio PK completato

Digita "go" per decifrare il risultato delle elezioni
```

Il blockchain node, invece, salva le PK che verranno poi inserite nel blocco genesi:

PW_Shacalli_BlockchainNode

```
Candidate1 connesso

PK dell'attore ricevuta: 27270719296884880916


[...]


Candidate4 connesso

PK dell'attore ricevuta: 7898336977050788241
```

```
PersonOfLaw1 connesso
PK dell'attore ricevuta: 15190576026272706924

[...]

PersonOfLaw4 connesso
PK dell'attore ricevuta: 25591658638586673248

PRep connesso
PK dell'attore ricevuta: 1134169922582563970

PReg connesso
PK dell'attore ricevuta: 9951459249244373063

PProv connesso
PK dell'attore ricevuta: 58364756769466261 43

PK degli attori raccolte
```

Il blockchain node, una volta ottenute le PK degli attori, avvia la procedura di creazione del polinomio per poter condividere il segreto tra i vari attori e attende che la macchina della giustizia crei il blocco genesi:

PW_Shacalli_BlockchainNode

```
Inizio generazione e distribuzione del segreto

24914470526642643458    -> Contributo cifrato Candidate1
[...]
8657003075244797658     -> Contributo cifrato Candidate4
9945411459953701963     -> Contributo cifrato PersonOfLaw1
[...]
20159007870318623327    -> Contributo cifrato PersonOfLaw4
11281313018079062001    -> Contributo cifrato PRep
7621208197271358157     -> Contributo cifrato PReg
15784352620999484009    -> Contributo cifrato PProv
Blocco genesi creato
```

Al termine, si passa alla fase di autenticazione dei cittadini, i quali diventeranno elettori:

PW_Shacalli_Elettori

```
Richiesta di ammissione al voto del cittadino 1
[...]
Richiesta di ammissione al voto del cittadino 10

Richiesta credenziali completato
```

PW_Shacalli_BlockchainNode

```
INIZIO GENERAZIONE DELLE CREDENZIALI

Cittadino connesso
PK dell'elettore generata: 10424535674567270474

[...]

Cittadino connesso
PK dell'elettore generata: 8162542874916198681

FINE GENERAZIONE DELLE CREDENZIALI
```

Al termine della procedura di generazione delle credenziali, si passa al voto:

PW_Shacalli_BlockchainNode

```
INIZIO VOTAZIONI

Elettore connesso
Voto espresso correttamente

[...]

Elettore connesso
Voto espresso correttamente

FINE VOTAZIONI
```

PW_Shacalli_Elettori

```
Votazione dell'elettore 1
Digita:
"1" per votare il candidato Candidate1
Digita:
"2" per votare il candidato Candidate2
Digita:
"3" per votare il candidato Candidate3
Digita:
"4" per votare il candidato Candidate4
2

[...]

Voto inviato!!!

Votazione dell'elettore 10
Digita:
"1" per votare il candidato Candidate1
Digita:
"2" per votare il candidato Candidate2
Digita:
"3" per votare il candidato Candidate3
Digita:
"4" per votare il candidato Candidate4
4

Voto inviato!!!
```

Dopo aver concluso la fase di voto, il blockchain node crea il cyphertext somma e attende che i vari attori inviano i vari contributi in riferimento a quel determinato cyphertext:

PW_Shacalli_Attori

```
LETTURA BLOCKCHAIN
Ottengo le share
Share ottenute!
```

PW_Shacalli_BlockchainNode

```
INIZIO CREAZIONE CYPHERTEXT SOMMA


FINE CREAZIONE CYPHERTEXT SOMMA


INIZIO OTTENIMENTO CONTRIBUTI DAGLI ATTORI


Attore connesso
Contributo ricevuto correttamente


[...]


Attore connesso
Contributo ricevuto correttamente


FINE OTTENIMENTO CONTRIBUTI DAGLI ATTORI
```

Infine, il blockchain node, in possesso di tutti gli elementi necessari, decifra il risultato delle elezioni:

PW_Shacalli_BlockchainNode

```
RISULTATI ELEZIONI
Candidate1: 0 voti
Candidate2: 1 voti
Candidate3: 1 voti
Candidate4: 8 voti
```

Lo stesso fanno i vari elettori che al termine della decifratura, prendono atto del risultato delle elezioni:

PW_Shacalli_Elettori

```
Digita "go" per sapere il risultato delle elezioni
go


RISULTATI ELEZIONI
Candidate1: 0 voti
Candidate2: 1 voti
Candidate3: 1 voti
Candidate4: 8 voti
```

Modifiche effettuate

Rispetto alla consegna del 50%, sono state effettuate delle modiche. Di seguito si riportano le modifiche effettuate:

1. **Sostituzione del sistema di voto con la blockchain:** rispetto alla consegna del 50 %, il sistema di voto è stato sostituito con la blockchain BEV. Motivo di questa scelta è stato il fatto che utilizzare un singolo server, nel nostro caso il sistema di voto, significava anche avere un punto di centralizzazione nel nostro sistema. Quindi se il server venisse attaccato ciò porterebbe ad una completa rottura del sistema. Invece utilizzando una blockchain, questo tipo di attacco viene scoraggiato dal fatto che, essendo la blockchain un sistema decentralizzato, anche se un singolo dispositivo che forma la blockchain venisse attaccato, ciò non porterebbe alcun danno al sistema.
2. **All'interno degli attori è stato aggiunto il presidente della repubblica al posto del sistema di voto:** rispetto alla consegna del 50%, è stata effettuata una modifica riguardante gli attori che sono coinvolti nel segreto. Come detto nel punto precedente, il sistema di voto è stato sostituito con la blockchain BEV. Quindi di conseguenza vi è stata anche una modifica degli attori coinvolti nel segreto sostituendo il sistema di voto con una nuova entità cioè il presidente della repubblica. Questo rappresenta una maggiore sicurezza all'interno del sistema perché rispetto al sistema di voto centralizzato che può essere facilmente attaccato, è meno probabile invece corrompere il presidente della repubblica in quanto questo non gioverebbe alla sua reputazione e carriera.
3. **Aggiunta dell'avversario the overloaders:** rispetto alla consegna del 50%, è stato aggiunto un ulteriore avversario chiamato the overloaders. Questa tipologia di avversario è intenzionata a sovraccaricare il sistema effettuando molte richieste di voto impedendo di conseguenza ai cittadini di votare.
4. **Modifica della proprietà di integrità I.2:** rispetto alla consegna del 50%, la proprietà di integrità I.2 è stata tolta dalle proprietà di integrità ed è stata aggiunta come premessa iniziale, questo in quanto rappresenta un compromesso tra integrità e privacy.
5. **Sono stati tolti i bit per candidato:** rispetto alla consegna del 50%, è stata effettuata una modifica riguardante il formato del voto. In precedenza, ad ogni candidato era associato un numero di bit, nel caso peggiore ad ogni candidato erano associati 22 bit. Questi bit sono stati tolti per i seguenti motivi: la prima motivazione è dovuta al fatto che, avendo utilizzato ElGamal esponenziale, quando andiamo ad effettuare la decifratura del messaggio, per ottenere il risultato delle elezioni è necessario effettuare una ricerca Brute Force. Avere troppi bit all'interno del messaggio si tradurrebbe con un tempo di ricerca molto lungo e questo porterebbe ad una minore efficienza all'interno del sistema. La seconda motivazione è dovuta al fatto che all'interno dello schema di ElGamal, la sicurezza viene data principalmente dalla randomness. Quando calcoliamo un ciphertext di ElGamal, quello che facciamo è calcolare la tupla $(u = g^r, v = g^m y^r)$ con $y = g^x$. Quindi anche se il nostro messaggio fosse formato da pochi bit, all'esponente di v otterremmo sempre un valore molto grande in Z_q.
6. **Nel WP2 nella fase di ottenimento delle credenziali di voto, la ZKP è stata sostituita con la firma digitale:** all'interno del WP2, nella consegna del 50%, nella fase di ottenimento delle credenziali di voto il cittadino quando effettuava la richiesta delle credenziali, doveva inviare anche una ZKP, dimostrando di conoscere la chiave segreta SK_{CTi}. A questo è stata effettuata una modifica sostituendo la ZKP con la firma digitale per il seguente motivo: quando andiamo a firmare un messaggio, la firma viene effettuata utilizzando la chiave segreta. Quindi se la

firma corrispondente al messaggio è corretta allora questo dimostra che il cittadino è in possesso della chiave segreta SK_{CTi}. Invece, se la firma non fosse verificata, al cittadino non verrebbe concesso di generare le credenziali di voto.

7. **Modifiche formali**: come detto in precedenza nel punto 1, rispetto alla consegna del 50% il sistema di voto è stato sostituito con la blockchain BEV. Questo ha portato a delle modifiche formali all'interno del progetto per adattare quanto fatto nel 50% all'utilizzo della blockchain.
8. **Modifica proprietà di integrità**: rispetto a quanto consegnato nel 50%, è stata effettuata un'ulteriore modifica alle proprietà di integrità, in particolare la proprietà I.7 è stata tolta e inglobata nella proprietà P.2.