UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA
E MATEMATICA APPLICATA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Progetto Cybersecurity

Gruppo: Security Team

Landi Thomas, t.landi3@studenti.unisa.it, 0622701813

Paolillo Alessandro, a.paolillo26@tudenti.unisa.it, 0622701812

Santorelli Francesco, f.santorelli8@studenti.unisa.it, 0622701809

Savarese Marco, m.savarese18@studenti.unisa.it, 0622701847

Anno Accademico 2021 – 2022

Workpackage	*Task*	*Responsabile*
WP1	Modello	Thomas Landi

Il voto elettronico è, in generale, il voto espresso tramite apparecchi elettronici di diverso tipo. Una particolare specie di voto elettronico che desta maggiore curiosità e perplessità è rappresentata dall'internet voting, cioè un sistema che si caratterizza per l'utilizzo di postazioni remote dotate di una connessione di rete internet attraverso la quale i voti vengono espressi e trasmessi. Il voto elettronico porta con sé indiscutibili vantaggi:

- Completa eliminazione delle schede nulle. La macchina, infatti, impedisce errori di compilazione della scheda e l'apposizione di eventuali segni di riconoscimento sulla stessa guidando l'elettore nella procedura di voto. In questo modo le schede sono necessariamente compilate nel modo corretto e, di conseguenza, non possono essere nulle.
- Offrire la possibilità agli elettori di votare online in modo sicuro può favorire la possibilità di esprimere il voto in qualsiasi momento e luogo durante tutto il periodo elettorale aumentando, di conseguenza, l'affluenza al seggio 'virtuale'.
- Costi minori rispetto al voto in presenza: l'organizzazione delle elezioni non richiederebbe l'impiego attivo ai seggi delle forze dell'ordine, la presenza di scrutatori e presidenti di seggi, l'utilizzo e lo sfruttamento di strutture (come le scuole) dedicate ad altre attività.
- Maggiore privacy, soprattutto in caso di elezioni che coinvolgono poche migliaia di votanti.

Nonostante ciò, da un punto di vista meno tecnico e più giuridico, l'internet voting potrebbe sollevare problemi di segretezza e libertà del voto, dato che è materialmente impossibile verificare che ogni elettore voti senza la presenza di terzi, i quali potrebbero influire, anche illegalmente, sul voto stesso. In Italia, questi problemi assumono rilievo costituzionale, dal momento in cui i principi di segretezza e libertà sono sanciti dall'art. 48, comma 2 della Costituzione. Lo stesso articolo, poi, introduce nell'ordinamento anche i principi di personalità e uguaglianza del voto, per cui se volessimo introdurre un sistema di voto elettronico, questo dovrebbe anche garantire che il voto di ogni elettore non venga conteggiato più di una volta. A tal fine un sistema di internet voting, per evitare il fenomeno della coercizione, dovrebbe consentire di modificare la preferenza espressa dall'elettore in qualsiasi momento all'interno della finestra temporale stabilita per votare. Ovviamente, per garantire che ogni elettore non voti più di una volta, bisogna tenere in conto solo l'ultima preferenza espressa. Questi principi sono oggi tutelati tramite l'identificazione dell'elettore effettuata dal personale di seggio, che annota nel registro i nominativi di coloro che hanno già votato, assicurandosi che non votino una seconda volta. Nel caso di votazione elettronica, il cittadino si potrebbe identificare tramite una carta d'identità elettronica o metodi di identificazione biometrici.
È inoltre necessario che un sistema di voto remoto/elettronico sia trasparente nel senso che non debba affidarsi eccessivamente ad una presunta parte fidata, ma abbia invece una progettazione ed analisi che permetta a tutti di verificarne la sua bontà limitando il danno che può essere causato da un qualunque avversario.
L'Estonia è uno dei pochi Paesi che utilizza un sistema di internet voting per il quale il cittadino ha la possibilità di votare nei dieci giorni antecedenti alle elezioni anche più di una volta. Il voto più recente annulla quelli precedenti. Inoltre, per evitare eventuali influenze e condizionamenti durante l'espressione del voto non presidiato, viene data la possibilità di votare anche in seggi presidiati con le tradizionali modalità del voto cartaceo e il voto espresso al seggio prevale su quello eventualmente già espresso su internet. Ci sono poi dei paesi, come la Norvegia e la Finlandia, che hanno deciso di abbandonare il voto elettronico per ragioni legate più che altro alla sicurezza o alla scarsa fiducia degli elettori sul sistema di voto. La Germania, invece, è tornata al voto cartaceo a seguito della sentenza costituzionale che dichiara incostituzionale il voto per mancanza di pubblicità delle operazioni elettorali.

Ciò premesso elenchiamo gli attori coinvolti nel sistema che si intende realizzare con votazione referendum:

- Utenti del sistema da realizzare: cittadini aventi il diritto di voto e correttamente registrati presso il Comune della città di residenza o online. Devono essere in possesso di un dispositivo digitale dotato di connessione internet.
- *Authority* (AU): ramo del governo responsabile "delle votazioni". E' costituita da più membri che lavorano per lo stesso scopo.
- Ministero della digitalizzazione (MD): entità controllata dal governo; possiede ingenti risorse finanziarie e dispone quindi di connessioni a larga banda e server molto potenti.
- Giustizia (J): può essere coinvolta solo nel caso in cui vi siano situazioni esterne al sistema digitalizzato. Ad esempio, un utente viene minacciato o corrotto affinché esprima un determinato voto.

Si assume che tutti gli aventi diritti al voto siano già in possesso dei rispettivi parametri di autenticazione.

Ogni attore in gioco potrebbe avere obiettivi di voto differenti e potrebbero chiaramente introdurre rischi di brogli che possono provenire da singoli o coalizioni di attori.

Assumiamo che in casi occasionali si possa coinvolgere anche la giustizia, che ha un'identità ben nota. Ha senso che J venga utilizzato solo in caso di controversia, evitando quindi il più possibile il suo coinvolgimento.

Completeness.

La votazione elettronica si svolge in due fasi. La prima fase ha una finestra temporale T0-T1 la quale è prestabilita dal governo. Durante la prima finestra temporale T0-T1, ogni cittadino avente diritto al voto effettua la registrazione. Durante la seconda fase, T1-T2, gli elettori possono votare esprimendo la propria preferenza. Infine durante la fase T2-T3, vengono determinati dal sistema i risultati del referendum. La durata di questa fase varia a seconda del volume di informazioni e della performance del sistema.

Sia ***v1,…,vn*** l'insieme dei potenziali votanti, sia ***xi*** la preferenza espressa dall'i-esimo votante. Ciascun votante ***v*** può partecipare, tra il tempo T1 e T2, ad una votazione esprimendo la propria scelta attraverso un input ***xi***. Tale scelta è modificabile fino al tempo T2.

Allo scadere del tempo T2, l'ultima preferenza espressa da ciascun votante viene definitivamente inviata e memorizzata all'interno del sistema.

Il ministero della digitalizzazione ***MD*** può essere coinvolto in qualsiasi momento.

La giustizia ***J*** può essere invocata a seguito di qualsiasi segnalazione di illeciti per verificarne la veridicità.

Si assume che l'Authority **AU** non commetta nessun tipo di errore nello stabilire se una persona possa votare o meno.

L'utente **v** non potrà più accedere a quella specifica votazione[1] ma potrà visionare la sua scelta in qualunque momento. Solo l'ultima votazione dell'utente **v** sarà presa in considerazione.

Threat model.

Avversario che cambia i voti. Durante la trasmissione del voto al sistema, un avversario potrebbe essere interessato ad intercettare il voto stesso per modificarlo a suo piacimento. Questo avversario può avere una potenza di calcolo discretamente forte.

Avversario che vota per un'altra persona. Avversario che accede al sistema con credenziali rubate a terze persone per votare a loro nome. Questo avversario non ha una potenza di calcolo, sfrutta tecniche di ingegneria sociale per impadronirsi dei dati all'insaputa dell'utente.

[1] Si intende che l'utente potrà modificare la propria preferenza solo eseguendo nuovamente e da capo tutto il processo di votazione.

Avversario che vuole sapere i voti degli altri. Avversario che ha lo scopo di scoprire il voto espresso dagli elettori, violando la privacy del voto. Tenta di intercettare la trasmissione del voto e può risalire all'identità dell'elettore qualora riuscisse a ottenere le loro credenziali di accesso al sistema. Questo avversario può avere una potenza di calcolo discretamente forte.

Avversario che elimina i voti. Durante la trasmissione del voto al sistema, un avversario potrebbe essere interessato ad intercettare il voto stesso per eliminarlo se non è di suo piacimento. Questo avversario può avere una potenza di calcolo discretamente forte.

Cittadino scorretto. Un cittadino scorretto può essere interessato a portare un attacco di tipo DDoS (tecnicamente la versione distribuita del ***Denial of Service***) interrompendo i servizi del sistema e, di conseguenza, la votazione. Questo avversario può avere una potenza di calcolo discretamente forte.

Membro dell'authority corrotto: può essere interessato a violare la privacy del voto, manipolare il conteggio e/o modificare i voti. Questo avversario può avere una potenza di calcolo discretamente forte.

Integrity.

In presenza di avversari, il sistema dovrebbe garantire:

- I.1 **veridicità**. Il risultato finale deve essere determinato in base a tutti i voti legittimi, senza alcun tipo di alterazione durante o dopo la trasmissione dei voti.
- I.2 il **corretto conteggio** delle votazioni, garantendo che sia conteggiata solo l'ultima preferenza espressa da ogni utente una sola volta.
- I.3 l'**ingiustizia** dovrebbe essere improbabile o almeno scoraggiata mediante sanzioni.
- I.4 **idoneità**: solo gli aventi diritto possono richiedere le credenziali d'accesso per poter accedere alla votazione

Privacy.

In presenza di avversari, il sistema dovrebbe garantire:

- P.1 **riservatezza**. Non è possibile risalire in alcun modo ad informazioni sulla preferenza espressa da altri utenti.
- P.2 **incoercibilità**. Non è possibile imporre un determinato voto a un votante.

Workpackage	*Task*	*Responsabile*
WP2	Soluzione	Marco Savarese

Si assume che i computer o i device degli elettori non siano compromessi. Ci si può fidare, inoltre, del ministero della digitalizzazione MD. Una blockchain proof-of-work può funzionare correttamente solo se meno del 50% delle risorse di calcolo nella rete stanno cercando di imbrogliare cambiando la blockchain in modo dannoso. 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). E' stata adottata una blockchain di tipo permissionless, le cui motivazioni con relativi benefici e svantaggi saranno analizzati nel WP3. Infine, la comunicazione sulla blockchain per l'inoltro delle transazioni avviene mediante il protocollo TLS 1.3.

L'Authority (AU) pubblica uno smart contract in merito al referendum sulla blockchain "Ref-chain" specificando:

- il referendum per cui si vota;
- la finestra temporale entro cui gli elettori possono registrarsi per accedere al voto indicando l'indice di blocco iniziale T0 e l'indice di blocco finale T1 della Ref-Chain;
- la finestra temporale in cui i voti possono essere accettati indicando l'indice di blocco iniziale T1 e l'indice di blocco finale T2 della Ref-chain;
- un indice di blocco aggiuntivo T3 che corrisponde al tempo entro il quale deve essere calcolato l'esito del voto;
- una chiave pubblica di AU PKau e un corrispondente certificato;

Lo smart contract si annuncia pubblicamente utilizzando i canali standard, proprio come ogni pubblica amministrazione annuncia un bando.
Il processo di voto è diviso in due parti: una prima in cui il votante **v** esegue il processo di registrazione e autenticazione. Contestualmente **AU** verifica che l'elettore autenticatosi abbia i requisiti necessari per votare. Successivamente, nella seconda parte, l'elettore effettuerà la propria votazione.

Fase 1: registrazione

Ogni elettore interessato al referendum, per poter votare, deve disporre di un ID valido. Tale ID può essere ottenuto in due modi:

- Recandosi presso l'ufficio comunale di appartenenza con un'identificazione fisica;
- Accendendo al sito online utilizzando la propria identità digitale. L'utilizzo dello SPID, in questo caso, implica che l'utente si fidi dei gestori dell'identità digitale (soggetti privati per la gestione delle identità digitali degli utenti) e dei fornitori dei servizi (service provider, organizzazioni pubbliche o private che consentono la fruizione del servizio). Lo SPID fornisce 3 livelli di sicurezza: accesso tramite nome utente e password, accesso tramite credenziali e una OTP, accesso tramite credenziali e dispositivi fisici.

L'ID di ciascun avente diritto al voto viene trasmesso e memorizzato nel database dell'AU.

Dopo aver ottenuto l'ID elettorale, ogni elettore deve generare la propria coppia di chiavi. Tale coppia sarà utile successivamente per poter firmare il messaggio m creato e formalizzare il voto.
L'algoritmo di generazione delle chiavi è il seguente:

1. Si sceglie un parametro di sicurezza di 2048 bits (dimensione consigliata per implementazioni reali);
2. Si sceglie un numero intero primo **q** lungo quanto il parametro di sicurezza;
3. Si sceglie un numero intero **p** pari a 2q+1;
4. Si sceglie un numero intero **g**, il quale è un generatore di q;
5. Si sceglie un numero intero primo **s** lungo quanto il parametro di sicurezza;
6. Si sceglie un numero intero **h = g^s mod p**.

La chiave pubblica **è Pkv=(p,q,g,h)** mentre quella privata è **Skv=(s).**

Dopo aver generato la propria coppia di chiavi, ogni elettore deve registrare la propria chiave pubblica (e tenere segreta per sé quella privata) all'Authority utilizzando l'ID elettorale ottenuto precedentemente. AU verifica l'ID di ciascun elettore e registra la corrispondente chiave pubblica Pkv in una lista pubblica.

Fase 2: procedura di voto

Un elettore v con chiave segreta **Skv** e chiave pubblica **Pkv**, notando l'annuncio e quindi lo smart contract, verifica che sia ben formato (AU è l'identità attesa) e che l'ultimo blocco di Ref-Chain precede T2 e può quindi decidere di votare.

L'elettore **v** crea un messaggio del tipo:

$$m = (Ts; x; R)$$

dove:

- **Ts** è un timestamp che mostra data e ora della creazione del messaggio, tale parametro consente all'elettore di votare più volte in quanto verrà presa in considerazione sono la votazione più recente;
- **x** è la preferenza espressa da v. E' un intero pari a -1 per votare "no", 0 per astenersi e 1 per votare "sì";
- **R** è un valore random necessario per prevenire attacchi. Tale elemento permette di impedire a un utente malintenzionato di indovinare la coppia di chiavi di v intercettando gli m crittografati creati da v stesso.

Dopo aver creato il messaggio, quest'ultimo viene cifrato utilizzando la chiave pubblica di AU Pkau:

$$C = Enc(PKau; m) = (u,v)$$

La funzione di cifratura prende in input il messaggio e la chiave pubblica di AU Pkau. Sceglie un **r** in Zq random di lunghezza pari al parametro di sicurezza e restituisce in output una coppia (u,v) dove **u=g^r** e **v=m*h^r**.

ZK proof: affinché la computazione sia corretta, assumiamo l'esistenza di una ZK proof che forzi il votante (prover) a mandare un ciphertext corretto. Per far sì che ciò accada, la preferenza espressa non può essere diversa da 1, 0 o -1.
L'informazione comune a prover e verifier è la PKau. Il prover, partendo dalla randomness (segreto R in suo possesso) può restituire al verifier la "prova", quindi il ciphertext. Ovviamente senza una ZK proof si potrebbe fornire la randomness al verifier per dimostrargli che il voto è corretto, ma ciò introdurrebbe un grave problema di privacy.
Il verifier prende l'input pubblico (Pkau) e la prova restituita dal prover per verificare che il ciphertext è "well-formed".

A questo punto l'elettore calcola:

$$\mathbf{H = SHA256(C \,||\, k)} \text{ con } k = \{0,1\}^n$$

Ed infine firma il messaggio con la propria chiave privata Skv per rivendicarne la proprietà:

$$\mathbf{S = Sig(Skv, H)}$$

A questo punto tramite transazione Ref-Chain, l'utente chiede di aggiungere la tripla (Pkv, C, S) allo stato dello smart contract. Lo smart contract prima di aggiornare il proprio stato effettua tre controlli:

- Verifica che il blocco corrente in Ref-Chain non abbia ancora raggiunto T2;
- Verifica che la Pkv sia presente nella lista pubblica contenente tutti le Pkv appartenenti agli elettori idonei alla votazione;
- Prende la Pkv ed esegue **Ver(Pkv, H, S)** per verificare che la firma sia effettivamente quella dell'elettore che ha creato il messaggio e che non ci sia nessun tentativo di truffa.

Se tutti i controlli hanno esito positivo lo smart contract aggiorna il proprio stato.

El Gamal: omomorfismo

Sulla falsariga dell'utilizzo di UTXO nella blockchain Bitcoin, si ipotizza l'esistenza nella Ref-Chain di un database che contiene Pk, C e Ts associata ad ogni avente diritto al voto che ha espresso la propria preferenza. In tal modo, così come UTXO cerca di evitare il double spending, potremmo evitare che il voto di un votante che ri-esegue la votazione sia conteggiato 2 volte. Se nel database è presente, infatti, già la Pk del votante che prova a votare nuovamente, la transazione associata al voto precedente viene eliminata e sostituita dal nuovo blocco col nuovo voto.
Ovviamente ciò è possibile solo se ci si trova ancora all'interno della finestra temporale dedicata alla votazione.

Allo scadere di quest'ultima, sfruttiamo l'operazione omomorfica per combinare tutti i ciphertext presenti sulla blockchain in un unico ciphertext **Cfin** che sarà comunicato ai membri dell'organizzazione dell'AU.

Shamir Secret Sharing (n,n)

L'organizzazione AU in generale, può essere considerato come il dealer. Esso possiede infatti la SkAU con la quale è possibile decifrare Cfin.
Si ipotizza l'esistenza di un computer isolato sul quale è in esecuzione il software del dealer corretto su un sistema operativo non corrotto. Tutti i membri dell'organizzazione possono verificare che il software è corretto.
Sfruttiamo lo Shamir Secret Sharing distribuito tra 2 membri dell'AU in modo che tutti debbano collaborare affinchè sia decifrabile il risultato finale del referendum.

1° soluzione con SSS

Preparazione: il dealer è in possesso del segreto SkAU e si sceglie a caso a_i definendo quindi il polinomio casuale con termine noto SkAU:

$$p(x) = SkAU + a_i x^i$$

Dove la singola share per ogni membro dell'AU è $p(x_i)$

Ricostruzione: i 2 membri, avendo le 2 coppie $(x_i, p(x_i))$ ed effettuando l'interpolazione posso ricostruire il polinomio, in cui p(0) equivale poi a SkAU.

2° soluzione con SSS

Preparazione: il dealer è in possesso del segreto SkAU e si sceglie 2 stringhe s_i da distribuire ai partecipanti tali che

$$s1 \text{ XOR } s2 = SkAU$$

Ricostruzione: i 2 membri, avendo le 2 shares, possono comunicarsele tra loro in modo tale da fare lo XOR e ricavare il segreto SkAU

A questo punto, una volta ricostruita SkAU, è possibile calcolarsi il risultato finale dell'elezione ricavando y

$$y = Dec (Skau, Cfin)$$

Dove la decifratura avviene mediante l'operazione **v*u^-s = m**

A questo punto dopo aver decifrato, viene pubblicato l'esito del referendum:

- se il risultato è un numero intero positivo, allora ha vinto il "sì";
- se il risultato è un numero intero negativo o esattamente 0, allora ha vinto il "no";

Eventuali reclami possono essere fatti coinvolgendo la giustizia J e il ministero della digitalizzazione ed analizzando le transazioni incriminate che per costruzione della blockchain sono pubbliche a tutti.

Workpackage	*Task*	*Responsabile*
WP3	Analisi	Francesco Santorelli

Il sistema basato su blockchain proposto può gestire condizioni avverse riguardanti un qualsiasi sistema centralizzato, come: attacchi ai server, problemi di rete, problemi di alimentazione elettrica.
Per come è realizzata la blockchain, tutti i partecipanti (nodi o miners) hanno più o meno lo stesso database. Qualsiasi attacco riuscito ad alcuni nodi non annullerà l'intero processo di votazione.
Finché un elettore ha accesso alla rete blockchain, può sempre votare.
Pertanto, migliora la disponibilità di un sistema di voto in quanto può gestire condizioni avverse decentralizzando i dati.

Completezza

Si può osservare che se tutti i giocatori seguono il protocollo descritto nel WP2, non ci sarebbero conflitti nè nessun coinvolgimento di J e l'opzione più votata vince il referendum. Si noti che la giustizia è coinvolta solo quando ci sono problemi legati al mondo fisico e questo è inevitabile poiché il sistema digitalizzato non può prevenire gli illeciti nel mondo reale. E' importante, tuttavia, che il sistema di e-voting fornisca prove evidenti di comportamenti disonesti.

Integrità

Nel paragrafo, per ogni avversario, che può portare minacce all'integrità dell'informazione, si evidenziano le contromisure adottate dal sistema.

Avversario che cambia i voti: un avversario non può facilmente manomettere i voti, in quanto non possiede la chiave segreta del votante e di conseguenza non può replicare la firma. Inoltre il messaggio di voto, **m** è cifrato. L'attaccante, prima di tutto, dovrebbe capire come decifrare il messaggio per manometterlo.

Avversario che vota per un'altra persona: un avversario di questo tipo non è contrastabile nel mondo virtuale in quanto ha agito nel mondo fisico. Tuttavia, un attacco di questo tipo è facilmente identificabile in quanto le transazione sulla blockchain sono pubbliche e visibili da tutti, quindi qualora un utente scopra una votazione non sua può immediatamente rivolgersi alla giustizia e all'authority per generare nuove credenziali private.

Avversario che elimina i voti: un attacco di questo tipo è fortemente scoraggiato in quanto per eliminare una transazione e quindi un voto, l'avversario dovrebbe possedere oltre il 50% della potenza di calcolo della blockchain ma ciò è irrealistico in quanto il ministero della digitalizzazione garantisce che oltre il 50% delle risorse di calcolo della blockchain è onesta.

Riservatezza

Avversario che vuole sapere i voti degli altri: un avversario di questo tipo con i mezzi a disposizione non è in grado di decifrare il voto di un elettore in quanto non conosce la chiave segreta dell'Authority fondamentale per la decifratura nè tantomeno tutte le singole share dei membri dell'AU necessarie per la ricostruzione.

Membro dell'authority corrotto: un avversario che fa parte di questo ramo del governo con i mezzi a disposizione non potrebbe da solo decifrare nè l'intero risultato del voto, per cambiarne i risultati, nè ogni singolo voto per violare la privacy dei cittadini.

Tuttavia, l'utilizzo di una blockchain di tipo permissionless rende inevitabile che chiunque possa sapere se un amico o un conoscente abbia partecipato o meno alla votazione. Ad esempio, se Alice dice a Bob di non aver partecipato alla votazione, Bob se conoscesse la Pk di Alice può facilmente verificare se Alice stia mentendo o meno controllando tutti i blocchi.

Efficienza

Il sistema è stato progettato per essere abbastanza efficiente. In effetti, autorità come MD, J non partecipano alle normali attività; pertanto, non è necessario che siano online e reattive. Gli unici attori che richiedono un coinvolgimento frequente sono quelli che intrinsecamente devono contribuire al referendum, vale a dire i votanti e l'Authority. Gli unici calcoli coinvolti consistono in calcoli di funzioni hash crittografiche che sono molto veloci. Esistono anche alcuni esponenti modulari per calcolare firme e crittografie, ma i votanti devono solo eseguire un numero costante di tali operazioni, mentre l'Authority che è comunque dotato di hardware più potente necessita solo di un numero lineare di tali operazioni.

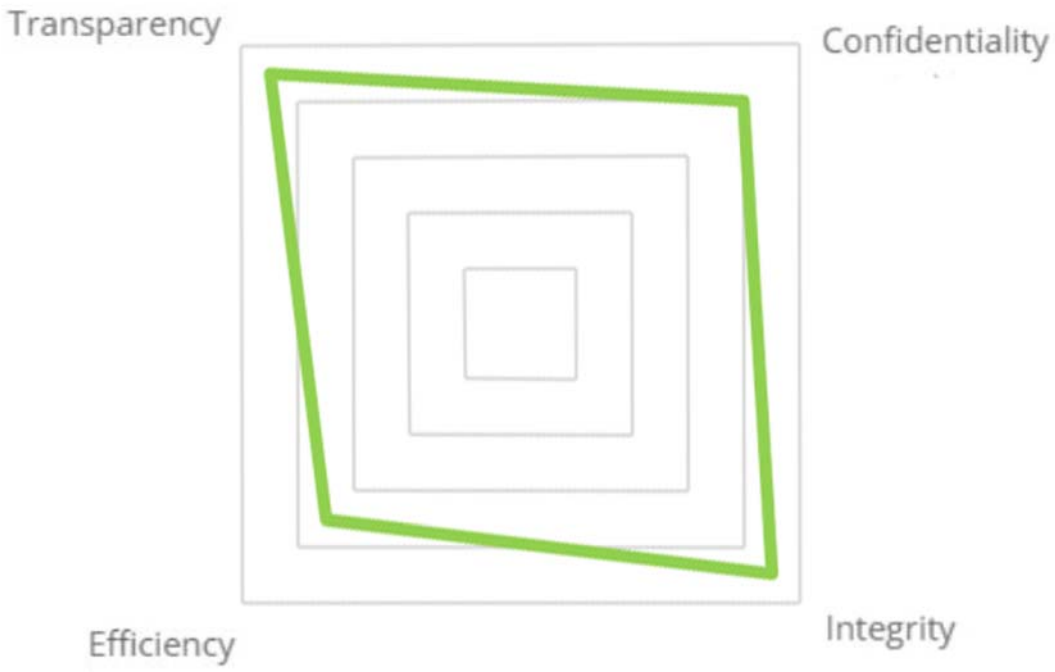
La possibilità da parte dei membri dell'AU di dover decifrare un unico ciphertext per determinare il risultato delle elezioni è più efficiente rispetto a dover decifrare tutti i ciphertext dei votanti singolarmente.

Tuttavia, una penalizzazione per l'efficienza è dovuta all'evidente latenza di transazione dovute all'utilizzo di Ref-Chain permissionless, ma d'altronde questo rende il sistema molto trasparente.

Trasparenza

Il sistema presentato non è basato su algoritmi segreti. Considera, inoltre, la piena corruzione di cittadini o membri del governo, minimizzando il ruolo delle altre parti che quindi saranno coinvolte solo in caso di controversia. Tutte le operazioni vengono eseguite con la verificabilità pubblica di Ref-Chain assicurandosi quindi che gli illeciti non possano essere nascoste modificando le informazioni utilizzate durante le varie fasi del referendum. In quanto tale il sistema fornisce un'elevata trasparenza.

Tool used in WP2	Consequences	Used for property
Servizio SPID (oppure identificazione fisica presso ufficio comunale)	Solo gli aventi diritto riescono a votare	I.4
Cifratura del voto	La preferenza espressa resta segreta	P.1
Possibilità di votare più volte	Il fenomeno della coercizione è mitigato	P.2
Blockchain	Non è possibile eseguire un "double voting" attack	I.2
ZK-proof	Non è possibile creare voti illegittimi	I.1
Segreto di Shamir	Favorisce la decentralizzazione e il calcolo del risultato finale	I.1


Transparency
Confidentiality
Efficiency
Integrity

Workpackage	*Task*	*Responsabile*
WP4	Implementazione	Alessandro Paolillo

La fase di implementazione ha riguardato lo sviluppo del:

- Server Dealer responsabile dell'invio delle chiavi segrete parziali ai due server di Authorization e voting.
- Server di Authorization che convalida l'identità digitale dell'utente consentendogli di accedere alla fase di votazione.
- Server di voting che gestisce la fase di votazione dell'utente correttamente autenticato.
- Server della giustizia (Recostruction) responsabile della ricostruzione della chiave segreta per poter effettuare, una volta terminato il tempo di voting, la fase di scrutinio e il conteggio finale del risultato per il referendum.
- Votante.

Tutte le comunicazioni sono gestite con l'utilizzo di certificati digitali, rilasciati dal Governo, per poter avere la certezza che l'entità con la quale avviene lo scambio di informazioni sia quella desiderata. Tali certificati sono stati generati mediante curve ellittiche e sono:

- sslServerDealer.cer
- sslServerAuth.cer
- sslServerVoting.cer
- sslServerRec.cer

Per la realizzazione dei meccanismi di cifratura, decifratura, creazione delle chiavi segrete parziali e omomorfismo di ElGamal sono state inserite le classi:

- ElGamal.java
- ElGamalPK.java
- ElGamalSK.java
- ElGamalCT.java

Esempio di esecuzione

FASE 1: Dealer distribuisce la SKAU alle due entità Authorization e Voting

```
public static void main(String[] args) throws Exception {

    if (System.getProperty(
            "javax.net.ssl.keyStore") == null || System.getProperty("javax.net.ssl.keyStorePassword") == null) {
        // set keystore store location
        System.setProperty("javax.net.ssl.keyStore", "keystoreServerDealer");
        System.setProperty("javax.net.ssl.keyStorePassword", "serverDealer");
    }
    // create socket
    SSLServerSocketFactory sockfact = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault(); //
    SSLServerSocket sSock;
    SSLSocket[] sslSock = new SSLSocket[num_authority];
    sSock = (SSLServerSocket) sockfact.createServerSocket(4000); // bind to port 4000

    ElGamalSK Params = SetupParameters(64); // in real implementation set the security parameter to at least 2048 bits
    //there is some non-trusted entity that generates the parameters

    // we now suppose there are 2 authorities
    ElGamalSK[] SK = new ElGamalSK[num_authority];
    for (int i = 0; i < 2; i++) {
        SK[i] = Setup(Params);
    }

    ElGamalPK[] PartialPK = new ElGamalPK[num_authority];
    for (int i = 0; i < num_authority; i++) {
        PartialPK[i] = SK[i].getPK();
    }

    ElGamalPK PKAU = AggregatePartialPublicKeys(PartialPK);
    for (int i = 0; i < num_authority; i++) {
        System.out.println("Waiting for connections...");
        sslSock[i] = (SSLSocket) sSock.accept(); // accept connections
        System.out.println("Connection to Authority Server\n");
        Protocol(sslSock[i], SK[i], PKAU);
        System.out.println("Partial secret key sent successfully\n");
    }
```

Figura 1: Server Dealer

Il Dealer crea le chiavi segrete parziali e attende la connessione dei due server di Authorization e Voting. Una volta avvenuta la connessione procede con l'esecuzione del suo protocollo inviando la SK parziale e PKAU.

```
public class DealerSKAU {

    private static final int num_authority = 2;

    /**
     * Protocol execution Dealer
     * @param sSock
     * @param SKP
     * @param PKAU
     * @throws Exception
     */
    static void Protocol(Socket sSock, ElGamalSK SKP, ElGamalPK PKAU) throws Exception {

        OutputStream out = sSock.getOutputStream();

        try {

            ObjectOutputStream objectOut;

            objectOut = new ObjectOutputStream(out);

            objectOut.writeObject(SKP);
            objectOut.flush();
            objectOut.writeObject(PKAU);
            objectOut.flush();

            out.close();
            sSock.close(); // close connection
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

Figura 2: Protocollo Dealer

```java
public static void main(String[] args) throws Exception {

    if (args.length != 2) {
        printUsage();
        return;
    }
    if (System.getProperty("javax.net.ssl.trustStore") == null || System.getProperty("javax.net.ssl.trustStorePassword") == null) {
        System.setProperty("javax.net.ssl.trustStore", "truststoServerDealer");
        System.setProperty("javax.net.ssl.trustStorePassword", "serverDealer");
    }
    try {
        SSLSocketFactory sslsocketfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
        SSLSocket sslsocket = (SSLSocket) sslsocketfactory.createSocket(args[0], Integer.parseInt(args[1]));
        sslsocket.startHandshake();
        protocolSKPartialPKAU(sslsocket);
        System.out.println("Partial secret key acquired");
    } catch (IOException ex) {
        Logger.getLogger(ServerAuthenticationDealer.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private static void printUsage() {
    System.out.println("Usage:\n\tjava client.SocketClient [address] [port]");
}

private static void protocolSKPartialPKAU(SSLSocket sslsocket) throws IOException {
    InputStream in = sslsocket.getInputStream();
    try {
        ObjectInputStream inputStream;

        inputStream = new ObjectInputStream(in);
        ElGamalSK SKP = (ElGamalSK) inputStream.readObject();
        ElGamalPK PKAU = (ElGamalPK) inputStream.readObject();
        Utils.writeSKByte(SKP, SKAuthFile);
        Utils.writePKByte(PKAU, PKAUFile);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            in.close();
            sslsocket.close();
        } catch (IOException ex) {
            Logger.getLogger(ServerAuthenticationDealer.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Figura 3: Protocollo acquisizione SK parziale Server Authentication (Analogo protocollo per Server Voting)

Il server di Authorization si mette in comunicazione con il Dealer e avvia il protocollo di acquisizione della SK parziale e della PKAU. Il codice utilizzato per l'acquisizione del SK parziale e della PKAU del server di Voting è analogo a quello di Authorization.

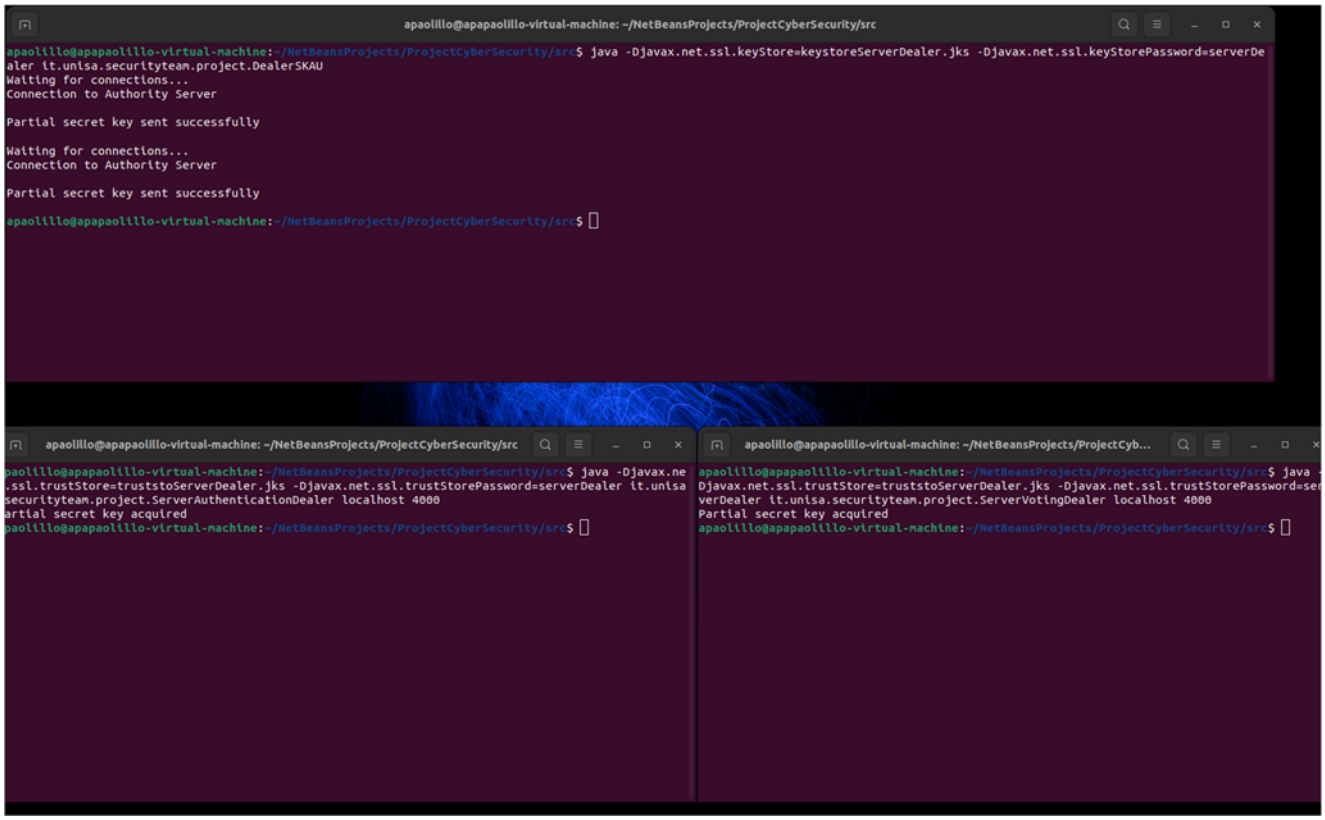
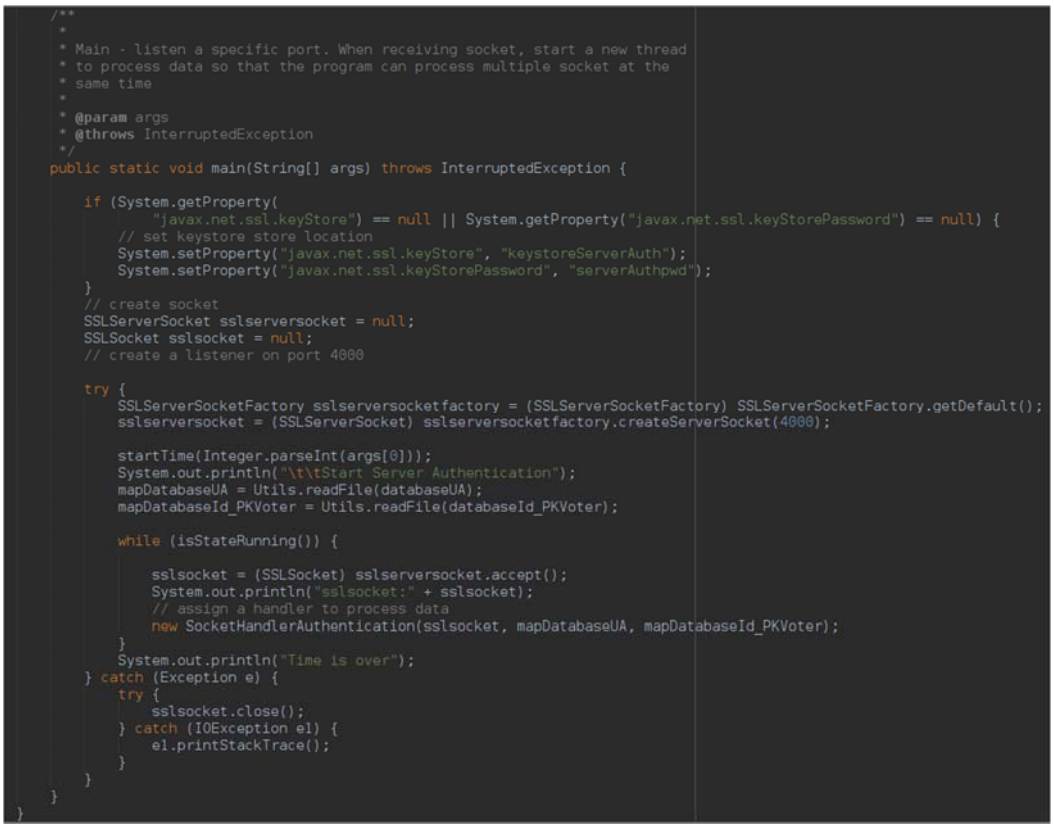


Figura 4: Esecuzione codice Secret Sharing ElGamalSK SKAU

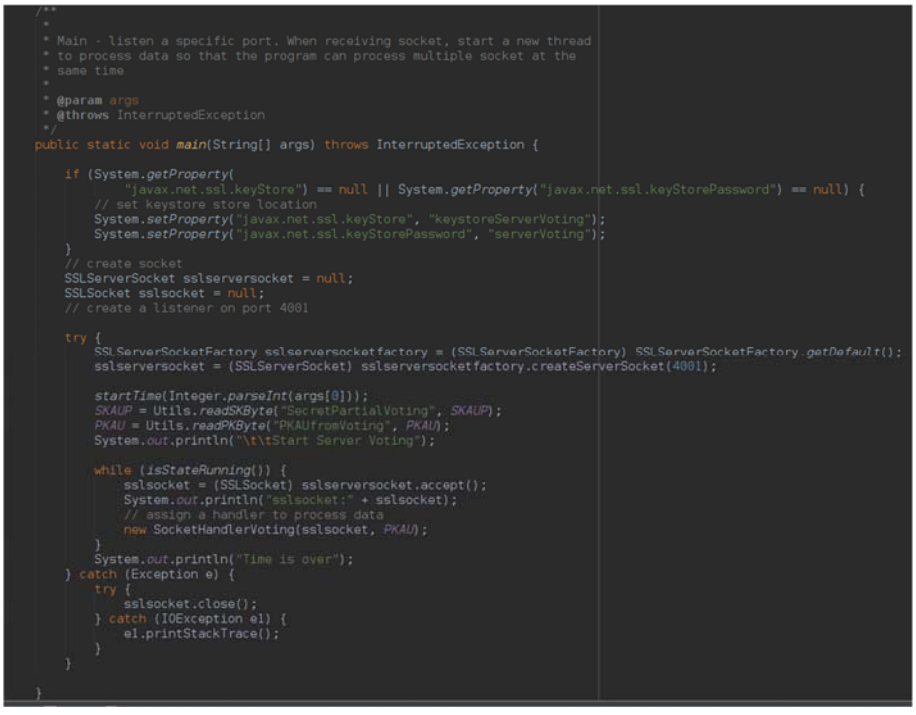
FASE 2: Avvio dei Server Authorization e Voting: procedura votazione

I Server di Authorization e Voting sono stati implementati seguendo la logica del multithreaded server. Tale logica prevede che i server siano capaci di gestire più comunicazioni contemporaneamente. Inoltre, tali server presentano un timeStopVoting, definito dal Governo, che indica il tempo durante il quale i server sono online e quindi, di fatto, il tempo totale del referendum.



```
    /**
     *
     * Main - listen a specific port. When receiving socket, start a new thread
     * to process data so that the program can process multiple socket at the
     * same time
     *
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {

        if (System.getProperty(
                "javax.net.ssl.keyStore") == null || System.getProperty("javax.net.ssl.keyStorePassword") == null) {
            // set keystore store location
            System.setProperty("javax.net.ssl.keyStore", "keystoreServerAuth");
            System.setProperty("javax.net.ssl.keyStorePassword", "serverAuthpwd");
        }
        // create socket
        SSLServerSocket sslserversocket = null;
        SSLSocket sslsocket = null;
        // create a listener on port 4000

        try {
            SSLServerSocketFactory sslserversocketfactory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
            sslserversocket = (SSLServerSocket) sslserversocketfactory.createServerSocket(4000);

            startTime(Integer.parseInt(args[0]));
            System.out.println("\t\tStart Server Authentication");
            mapDatabaseUA = Utils.readFile(databaseUA);
            mapDatabaseId_PKVoter = Utils.readFile(databaseId_PKVoter);

            while (isStateRunning()) {

                sslsocket = (SSLSocket) sslserversocket.accept();
                System.out.println("sslsocket:" + sslsocket);
                // assign a handler to process data
                new SocketHandlerAuthentication(sslsocket, mapDatabaseUA, mapDatabaseId_PKVoter);
            }
            System.out.println("Time is over");
        } catch (Exception e) {
            try {
                sslsocket.close();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

Figura 5: Esecuzione Server Authentication



```
    /**
     *
     * Main - listen a specific port. When receiving socket, start a new thread
     * to process data so that the program can process multiple socket at the
     * same time
     *
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {

        if (System.getProperty(
                "javax.net.ssl.keyStore") == null || System.getProperty("javax.net.ssl.keyStorePassword") == null) {
            // set keystore store location
            System.setProperty("javax.net.ssl.keyStore", "keystoreServerVoting");
            System.setProperty("javax.net.ssl.keyStorePassword", "serverVoting");
        }
        // create socket
        SSLServerSocket sslserversocket = null;
        SSLSocket sslsocket = null;
        // create a listener on port 4001

        try {
            SSLServerSocketFactory sslserversocketfactory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
            sslserversocket = (SSLServerSocket) sslserversocketfactory.createServerSocket(4001);

            startTime(Integer.parseInt(args[0]));
            SKAUP = Utils.readSKByte("SecretPartialVoting", SKAUP);
            PKAU = Utils.readPKByte("PKAUfromVoting", PKAU);
            System.out.println("\t\tStart Server Voting");

            while (isStateRunning()) {
                sslsocket = (SSLSocket) sslserversocket.accept();
                System.out.println("sslsocket:" + sslsocket);
                // assign a handler to process data
                new SocketHandlerVoting(sslsocket, PKAU);
            }
            System.out.println("Time is over");
        } catch (Exception e) {
            try {
                sslsocket.close();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }

    }
```

Figura 6: Esecuzione Server Voting

Entrambi gli ascoltatori lanciano il thread startTime.

```java
/**
 * This function launches a thread that, based on the timeStopVooting value,
 * stops the server authentication request.
 *
 * @param timeStopVoting
 * @throws InterruptedException
 */
private static void startTime(int timeStopVoting) throws InterruptedException {
    if (timeStopVoting < 0) {
        return;
    }
    stateRunning = true;

    Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(timeStopVoting);
                stateRunning = false;
            } catch (InterruptedException ex) {
                Logger.getLogger(SocketListenerAuthentication.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    });
    t.start();
}

/**
 *
 * @return Boolean
 */
private static boolean isStateRunning() {
    return stateRunning;
}
```

Figura 7: Thread che, una volta scaduto il tempo per il Referendum, mette i server offline.

Fase 2.1

Procedura di Autenticazione del votante

Il votante che desidera votare deve prima autenticarsi, pertanto si connette al server di Authentication.

```java
public static void main(String[] args) throws Exception {

    if (args.length != 2) {
        printUsage();
        return;
    }
    if (System.getProperty("javax.net.ssl.trustStore") == null || System.getProperty("javax.net.ssl.trustStorePassword") == null) {
        System.setProperty("javax.net.ssl.trustStore", "truststoServerAuth");
        System.setProperty("javax.net.ssl.trustStorePassword", "serverAuthpwd");
    }
    try {
        SSLSocketFactory sslsocketfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
        SSLSocket sslsocket = (SSLSocket) sslsocketfactory.createSocket(args[0], Integer.parseInt(args[1]));
        sslsocket.startHandshake();
        System.out.println("sslsocket=" + sslsocket);
        protocolAuth(sslsocket);

    } catch (IOException ex) {
        Logger.getLogger(SocketClientAuthentication.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private static void printUsage() {
    System.out.println("Usage:\n\tjava client.SocketClient [address] [port]");
}
```

Figura 8: Utente che avvia la comunicazione con il Server di Authentication per procedere all'autenticazione

ed esegue il protocollo di autenticazione. La procedura prevede che l'utente inserisca il

- FiscalCode
- UserName
- Password[2]

Quando l'utente inserisce le prime due informazioni il server di Authorization verifica che l'utente sia registrato all'interno del sistema. Se la verifica risulta essere negativa il sistema informa l'utente con:

- "User not present in database".

Altrimenti l'utente inserisce la password per poter terminare l'autenticazione. Se la password corrispondente a quell'utente non è corretta, il sistema informa l'utente con:

- "Password error, You still have --- attempts".

[2] Queste informazioni sono state acquisite dall'UA nella fase 1 discussa nel WP2

Se l'utente commette più di 3 errori allora il sistema blocca l'utente:

- "Attempts exhausted, Contact Assistance for Unlock".

Se la procedura di autenticazione è andata a buon fine allora il sistema informa l'utente con:

- "You have access!"

Il sistema verifica se l'utente ha già effettuato precedentemente un accesso controllando l'esistenza della PK corrispondente a quell'ID dell'utente appena autenticato. Se l'utente ha già effettuato precedentemente l'accesso allora non viene eseguito il protocolFirstAccess. Tale protocollo prevede che l'utente generi la ElGamalSK e ElGamalPK, memorizzi la ElGamalSK e mandi al server la ElGamalPK. Il server attende la ElGamalPK dell'utente correttamente autenticato e memorizza tale chiave pubblica nella sua mappa che ha per chiave l'ID Voter e valore ElGamalPK Voter.

```
*
 Constructor - initialize variables

 @param sslsocket
 @param mapDatabaseUA
 @param mapDatabaseId_PKvoter
/
blic SocketHandlerAuthentication(SSLSocket sslsocket, HashMap<String, String> mapDatabaseUA, HashMap<String, String> mapDatabaseId_PKvoter) {
  this.sslsocket = sslsocket;
  this.mapDatabaseUA = mapDatabaseUA;
  this.mapDatabaseId_PKvoter = mapDatabaseId_PKvoter;
  try {
      start();
  } catch (Exception e) {
      e.printStackTrace();
  }
```

Figura 9: Costruttore del gestore del Server multithreading Authentication

```
@Override
public void run() {
    OutputStream out = null;
    InputStream in = null;
    ObjectOutputStream objectOut;
    ObjectInputStream inputStream;
    try {
        out = sslsocket.getOutputStream();
        in = sslsocket.getInputStream();

        objectOut = new ObjectOutputStream(out);
        inputStream = new ObjectInputStream(in);

        objectOut.writeObject("\tInsert Fiscal Code:");
        objectOut.flush();

        String fiscalCode = (String) inputStream.readObject();

        objectOut.writeObject("\tInsert UserName:");
        objectOut.flush();

        String userName = (String) inputStream.readObject();

        if (!checkExisting(fiscalCode, userName)) {
            objectOut.writeBoolean(false);
            objectOut.flush();
            objectOut.writeObject("\nUser not present in database");
            objectOut.flush();
        } else {
            objectOut.writeBoolean(true);
            objectOut.flush();

            int repetition = 3;
            while (repetition > 0) {
                objectOut.writeObject("\tPlease insert password for voting");
                objectOut.flush();
```

```
    String psw = (String) inputStream.readObject();

    if (checkID(userName, psw)) {
        objectOut.writeBoolean(true);
        objectOut.flush();
        objectOut.writeObject(IDVoter);
        objectOut.flush();
        objectOut.writeObject("\nYou have access!");
        objectOut.flush();

        if (Utils.firstAccessClient(mapDatabaseId_PKvoter, IDVoter)) {
            objectOut.writeBoolean(true);
            objectOut.flush();
            protocolFirstAccess(inputStream);
        } else {
            objectOut.writeBoolean(false);
            objectOut.flush();
        }
        break;
    }
    objectOut.writeBoolean(false);
    repetition--;
    if (repetition == 0) {
        objectOut.writeObject("\nAttempts exhausted, Contact Assistance for Unlock");
        objectOut.flush();
    } else {
        objectOut.writeObject("\nPassword error, You still have " + repetition + " attempts");
        objectOut.flush();
    }
```

Figura 10: Protocollo Server Authentication

```java
private static void protocolAuth(SSLSocket sslsocket) {
    OutputStream out = null;
    InputStream in = null;
    ObjectOutputStream objectOut;
    ObjectInputStream inputStream;
    try {
        out = sslsocket.getOutputStream();
        in = sslsocket.getInputStream();

        objectOut = new ObjectOutputStream(out);
        inputStream = new ObjectInputStream(in);
        Scanner scanner = new Scanner(System.in);

        System.out.println((String) inputStream.readObject());
        String fiscalCode = scanner.next();
        objectOut.writeObject(fiscalCode);
        objectOut.flush();

        System.out.println((String) inputStream.readObject());
        String userName = scanner.next();
        objectOut.writeObject(userName);
        objectOut.flush();

        if (inputStream.readBoolean()) {
            int repetition = 3;
            while (repetition > 0) {
                System.out.println((String) inputStream.readObject());
                String psw = scanner.next();
                objectOut.writeObject(psw);
                objectOut.flush();

                if (inputStream.readBoolean()) {
                    IDVoter = (String) inputStream.readObject();
                    System.out.println((String) inputStream.readObject());
                    if (inputStream.readBoolean()) {
                        protocolFirstAccess(objectOut);
                    }
                    break;
                }
                repetition--;
                if (repetition == 0) {
                    System.err.println((String) inputStream.readObject());

                } else {
                    System.err.println((String) inputStream.readObject());
                }

            }
        } else {
            System.err.println((String) inputStream.readObject());
        }
```

Figura 11: Protocollo Client Authentication

```java
private static void protocolFirstAccess(ObjectOutputStream objectOut) {
    ElGamalSK SK = Setup(64); //questioni di tempo a 64 altrienti 2048
    try {
        objectOut.writeObject(SK.getPK().getP());
        objectOut.flush();

        objectOut.writeObject(SK.getPK().getQ());
        objectOut.flush();

        objectOut.writeObject(SK.getPK().getG());
        objectOut.flush();

        objectOut.writeObject(SK.getPK().getH());
        objectOut.flush();

        objectOut.writeObject(SK.getPK().getSecurityparameter());
        objectOut.flush();
    } catch (IOException ex) {
        Logger.getLogger(SocketClientAuthentication.class.getName()).log(Level.SEVERE, null, ex);
    }
```

```java
private void protocolFirstAccess(ObjectInputStream inputStream) {
    try {
        BigInteger p = (BigInteger) inputStream.readObject();
        BigInteger q = (BigInteger) inputStream.readObject();
        BigInteger g = (BigInteger) inputStream.readObject();
        BigInteger h = (BigInteger) inputStream.readObject();
        int securityparameter = (Integer) inputStream.readObject();
        mapDatabaseId_PKvoter.put(IDVoter, Utils.createStringPKElGamal(new ElGamalPK(p, q, g, h, securityparameter)));
    } catch (IOException ex) {
        Logger.getLogger(SocketHandlerAuthentication.class.getName()).log(Level.SEVERE, null, ex);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(SocketHandlerAuthentication.class.getName()).log(Level.SEVERE, null, ex);
    }
    updateMapAuthFinish();
```

Figura 12: Protocollo First Access Server-Client

Fase 2.2

Procedura di Votazione del votante

Il votante, una volta autenticato correttamente, si connette al server di Voting ed esegue il protocollo di votazione.

```
public static void main(String[] args) throws Exception {

    if (args.length != 2) {
        printUsage();
        return;
    }
    if (System.getProperty("javax.net.ssl.trustStore") == null || System.getProperty("javax.net.ssl.trustStorePassword") == null) {
        System.setProperty("javax.net.ssl.trustStore", "truststoServerVoting");
        System.setProperty("javax.net.ssl.trustStorePassword", "serverVoting");
    }
    try {
        SSLSocketFactory sslsocketfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
        SSLSocket sslsocket = (SSLSocket) sslsocketfactory.createSocket(args[0], Integer.parseInt(args[1]));
        sslsocket.startHandshake();
        System.out.println("sslsocket=" + sslsocket);
        SK = Utils.readSKByte(fileClientSK, SK);
        IDVoter = Utils.readIDVoterByte(fileClientID, IDVoter);
        protocolVote(sslsocket);

        //protocol(args[0], Integer.parseInt(args[1]));
    } catch (IOException ex) {
        Logger.getLogger(SocketClientVoting.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private static void printUsage() {
    System.out.println("Usage:\n\tjava client.SocketClient [address] [port]");
}
```

Figura 13: Votante che avvia la comunicazione con il Server di Voting per procedere alla votazione per il Referendum

La procedura prevede che l'utente invii al Server di Voting le informazioni della sua ElGamalPK e il suo ID Voter per un'ulteriore conferma della sua entità. Il sistema verifica se l'utente ha già votato verificando se esiste la sua ElGamalPK all'interno del database degli SmartContracts. Se esiste, allora significa che l'utente ha già effettuato in precedenza una scelta e il sistema informa l'utente con:

"This user has already voted", "Do you want to vote with a new vote?"

L'utente può decidere se rivotare. Se desidera rivotare scrive "yes" altrimenti "no" e il server termina la comunicazione tra le due parti.

Se l'utente non ha mai effettuato una votazione in precedenza o desidera effettuare una nuova votazione si procede al protocollo di votazione. Tale procedura prevede che il Server informi l'utente delle tre possibili scelte di voto:

- Yes
- No
- White

L'utente effettua la sua scelta ed invia al Server le informazioni come descritte nella fase 2 del WP2. Se le informazioni vengono aggiunte correttamente allo SmartContract il Server informa l'utente della corretta procedura con:

"Vote added".

```java
/**
 * Constructor - initialize variables
 *
 * @param sslsocket
 * @param SKUA
 */
public SocketHandlerVoting(SSLSocket sslsocket, ElGamalPK PKUA) {
    this.sslsocket = sslsocket;
    this.PKUA = PKUA;
    try {
        start();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 14:: Costruttore del gestore del Server multithreading Voting

```java
@Override
public void run() {
    OutputStream out = null;
    InputStream in = null;
    ObjectOutputStream objectOut;
    ObjectInputStream inputStream;
    try {
        out = sslsocket.getOutputStream();
        in = sslsocket.getInputStream();

        objectOut = new ObjectOutputStream(out);
        inputStream = new ObjectInputStream(in);

        ElGamalPK PKVoter = (ElGamalPK) inputStream.readObject();

        String IDVoter = (String) inputStream.readObject();

        if (checkPKVoter(PKVoter, IDVoter)) {
            objectOut.writeBoolean(true);
            objectOut.flush();
            if (Utils.alreadyVoting(smartContracts, PKVoter)) {
                objectOut.writeBoolean(true);
                objectOut.flush();
                objectOut.writeObject("This user has already voted\n");
                objectOut.flush();
                objectOut.writeObject("Do you want to vote with a new vo
                objectOut.flush();
                String x = (String) inputStream.readObject();
                if (choise(x)) {
                    objectOut.writeBoolean(true);
                    protocolVoting(objectOut, inputStream, PKVoter);
                } else {
                    objectOut.writeBoolean(false);
                    objectOut.writeObject("Closure");
                    objectOut.flush();
                }
            } else {
                objectOut.writeBoolean(false);
                objectOut.flush();
                protocolVoting(objectOut, inputStream, PKVoter);
            }
        } else {
            objectOut.writeBoolean(false);
            objectOut.writeObject("The user can not vote");
            objectOut.flush();

        }
```

```java
private void protocolVoting(ObjectOutputStream objectOut, ObjectInputStream inputStream, ElGamalPK PKVoter)
    try {
        objectOut.writeObject("The user can vote");
        objectOut.flush();

        objectOut.writeObject(PKUA);
        objectOut.flush();
        objectOut.writeObject("Choose your voting preference for the referendum:\nyes\t\twhite\t\t\tno");
        objectOut.flush();

        ElGamalCT CTMsg = (ElGamalCT) inputStream.readObject();
        SchnorrSig s = (SchnorrSig) inputStream.readObject();
        if (Verify(PKVoter, s, CTMsg.toString())) {
            objectOut.writeBoolean(true);
            objectOut.flush();

            protocolUpdateSmartContracts(PKVoter, CTMsg);
            objectOut.writeObject("Vote added");
            objectOut.flush();
        } else {
            objectOut.writeBoolean(false);
            objectOut.flush();

        }
    } catch (IOException ex) {
        Logger.getLogger(SocketHandlerVoting.class.getName()).log(Level.SEVERE, null, ex);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(SocketHandlerVoting.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
```

Figura 15: Protocollo Server di Voting

```java
private static void protocolVote(SSLSocket sslsocket) {
    OutputStream out = null;
    InputStream in = null;
    ObjectOutputStream objectOut;
    ObjectInputStream inputStream;
    try {
        out = sslsocket.getOutputStream();
        in = sslsocket.getInputStream();

        objectOut = new ObjectOutputStream(out);
        inputStream = new ObjectInputStream(in);
        Scanner scanner = new Scanner(System.in);

        objectOut.writeObject(SK.getPK());
        objectOut.flush();
        objectOut.writeObject(IDVoter);
        objectOut.flush();

        if (inputStream.readBoolean()) {

            if (inputStream.readBoolean()) {
                System.out.println((String) inputStream.readObject());
                System.out.println((String) inputStream.readObject());
                String x = scanner.next();
                objectOut.writeObject(x);
                objectOut.flush();
                if (inputStream.readBoolean()) {
                    protocolVoting(objectOut, inputStream, scanner);
                } else {
                    System.out.println((String) inputStream.readObject());
                }
            } else {
                protocolVoting(objectOut, inputStream, scanner);
            }
        } else {
            System.out.println((String) inputStream.readObject());
        }
```

```java
private static void protocolVoting(ObjectOutputStream objectOut, ObjectInputStream inputStream, Scanner scanner) {
    try {
        System.out.println((String) inputStream.readObject());

        ElGamalPK PKUA = (ElGamalPK) inputStream.readObject();
        System.out.println((String) inputStream.readObject());

        //BigInteger x = scanner.nextBigInteger(); Start
        String str = scanner.next();
        while (!(str.compareToIgnoreCase("yes") == 0 || str.compareToIgnoreCase("no") == 0 || str.compareToIgnoreCase("white") == 0)) {
            System.out.println("Choose your voting preference for the referendum:\nyes\t\twhite\t\t\tno");
            str = scanner.next();
        }
        BigInteger x;
        if (str.compareToIgnoreCase("yes") == 0) {
            x = new BigInteger("1");
        } else if (str.compareToIgnoreCase("yes") == 0) {
            x = new BigInteger("-1");
        } else {
            x = new BigInteger("0");
        }

        //End
        ElGamalCT CTMsg = EncryptInTheExponent(PKUA, x);
        objectOut.writeObject(CTMsg);
        objectOut.flush();

        SchnorrSig s = Sign(SK, CTMsg.toString());

        objectOut.writeObject(s);
        objectOut.flush();

        if (inputStream.readBoolean()) {
            System.out.println("\tRequest to add vote");
            System.out.println((String) inputStream.readObject());
        } else {
            System.out.println("\tRequest to add vote denied");
        }
```

Figura 16: Protocollo Client Voting

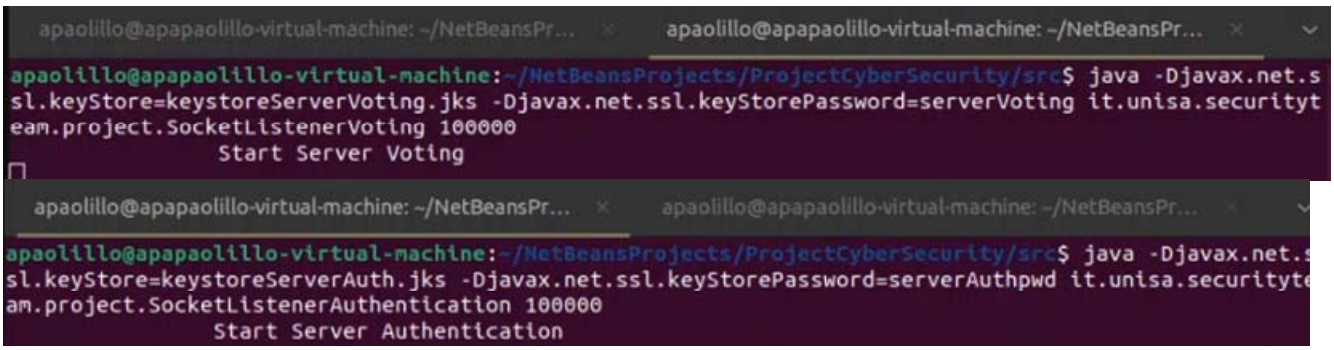


Figura 17: Avvio Server Authentication e Voting

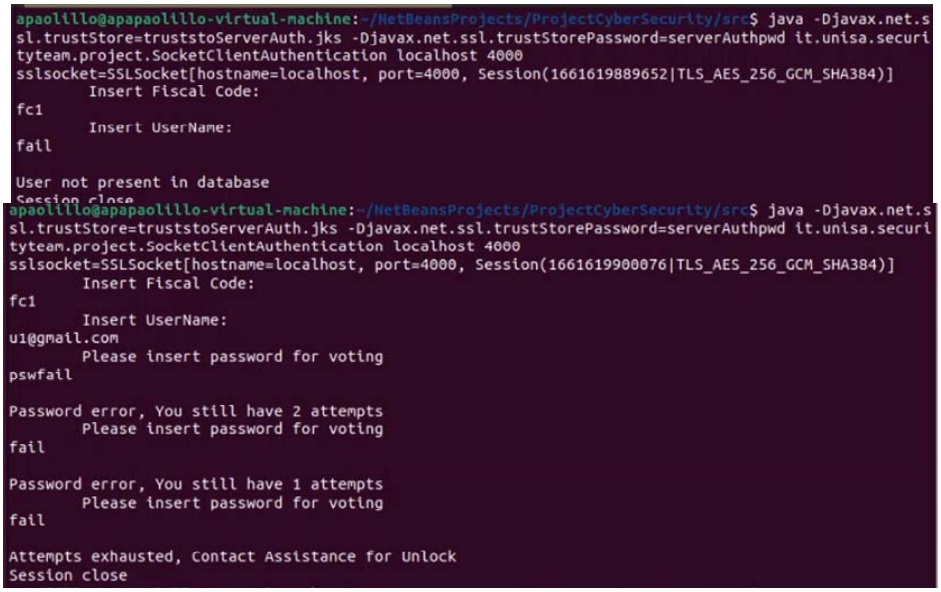
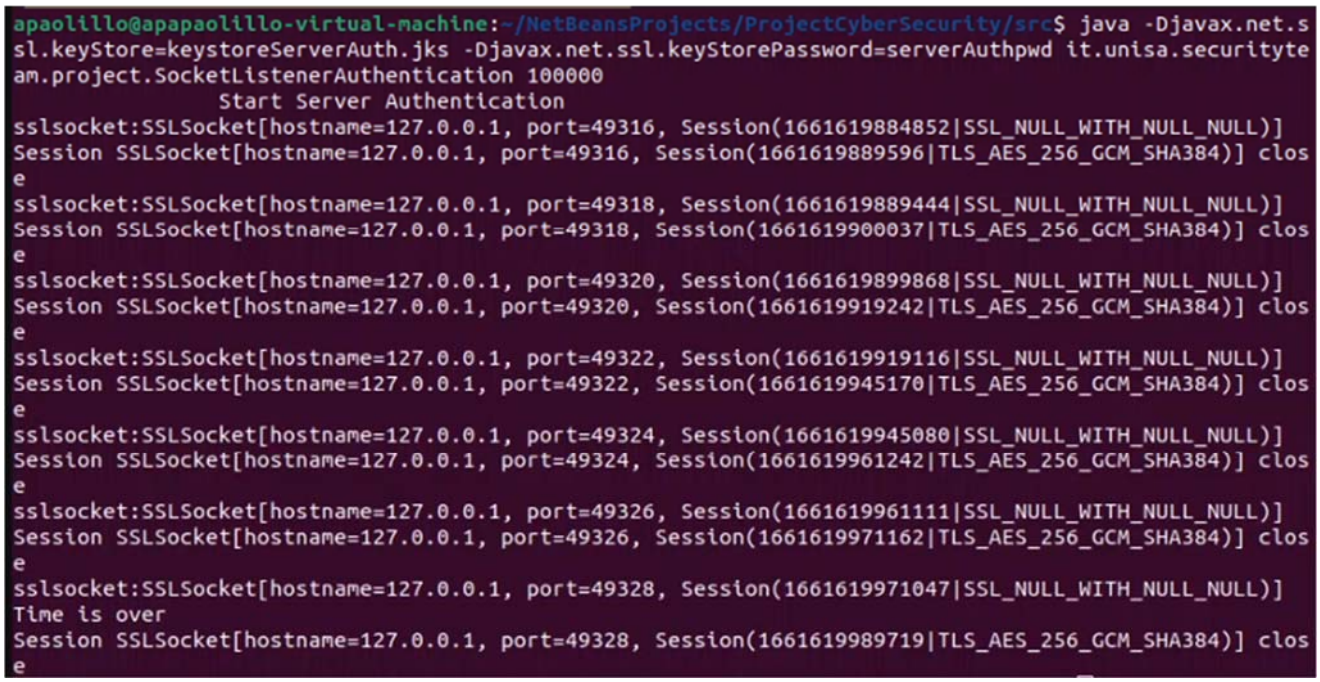


Figura 18: Esempio comunicazione Client-Authentication Errors



```
apaolillo@apapaolillo-virtual-machine:~/NetBeansProjects/ProjectCyberSecurity/src$ java -Djavax.net.s
sl.keyStore=keystoreServerVoting.jks -Djavax.net.ssl.keyStorePassword=serverVoting it.unisa.securityt
eam.project.SocketListenerVoting 100000
               Start Server Voting
sslsocket:SSLSocket[hostname=127.0.0.1, port=39574, Session(1661619886768|SSL_NULL_WITH_NULL_NULL)]
Session SSLSocket[hostname=127.0.0.1, port=39574, Session(1661619933431|TLS_AES_256_GCM_SHA384)] clos
e
sslsocket:SSLSocket[hostname=127.0.0.1, port=39578, Session(1661619933235|SSL_NULL_WITH_NULL_NULL)]
Session SSLSocket[hostname=127.0.0.1, port=39578, Session(1661619938635|TLS_AES_256_GCM_SHA384)] clos
e
sslsocket:SSLSocket[hostname=127.0.0.1, port=39582, Session(1661619938526|SSL_NULL_WITH_NULL_NULL)]
Session SSLSocket[hostname=127.0.0.1, port=39582, Session(1661619956329|TLS_AES_256_GCM_SHA384)] clos
e
sslsocket:SSLSocket[hostname=127.0.0.1, port=39584, Session(1661619956205|SSL_NULL_WITH_NULL_NULL)]
Session SSLSocket[hostname=127.0.0.1, port=39584, Session(1661619981291|TLS_AES_256_GCM_SHA384)] clos
e
sslsocket:SSLSocket[hostname=127.0.0.1, port=39586, Session(1661619981198|SSL_NULL_WITH_NULL_NULL)]
Time is over
Session SSLSocket[hostname=127.0.0.1, port=39586, Session(1661619994381|TLS_AES_256_GCM_SHA384)] clos
e
```

Figura 19: Tutte le comunicazioni che i Server Authentication e Voting hanno avviato e gestito durante il timeVoting

```
sl.trustStore=truststoServerAuth.jks -Djavax.net.ssl.trustStorePassword=serverAuthpwd it.unisa.secur
tyteam.project.SocketClientAuthentication localhost 4000
sslsocket=SSLSocket[hostname=localhost, port=4000, Session(1661619919260|TLS_AES_256_GCM_SHA384)]
        Insert Fiscal Code:
fc2
        Insert UserName:
u2@gmail.com
        Please insert password for voting
psw2

You have access!
Session close
```

```
sl.trustStore=truststoServerVoting.jks -Djavax.net.ssl.trustStorePassword=serverVoting it.unisa.secur
ityteam.project.SocketClientVoting localhost 4001
sslsocket=SSLSocket[hostname=localhost, port=4001, Session(1661619933493|TLS_AES_256_GCM_SHA384)]
The user can vote
Choose your voting preference for the referendum:
yes             white                   no
yes
        Request to add vote
Vote added
        Session close
```

```
This user has already voted

Do you want to vote with a new vote?
yes
The user can vote
Choose your voting preference for the referendum:
yes             white                   no
no
        Request to add vote
Vote added
        Session close
```

Figura 20: Procedura di autenticazione, voto e rivoto dell'utente u2

```
apaolillo@apapaolillo-virtual-machine:~/NetBeansProjects/ProjectCyberSecurity/src$ java -Djavax.net.
sl.trustStore=truststoServerAuth.jks -Djavax.net.ssl.trustStorePassword=serverAuthpwd it.unisa.secur
tyteam.project.SocketClientAuthentication localhost 4000
sslsocket=SSLSocket[hostname=localhost, port=4000, Session(1661619945182|TLS_AES_256_GCM_SHA384)]
        Insert Fiscal Code:
fc3
        Insert UserName:
u3@gmail.com
        Please insert password for voting
psw3

You have access!
```

```
sl.trustStore=truststoServerVoting.jks -Djavax.net.ssl.trustStorePassword=serverVoting it.unisa.secur
ityteam.project.SocketClientVoting localhost 4001
sslsocket=SSLSocket[hostname=localhost, port=4001, Session(1661619956359|TLS_AES_256_GCM_SHA384)]
The user can vote
Choose your voting preference for the referendum:
yes             white                   no
yes
        Request to add vote
Vote added
        Session close
```

Figura 21: Procedura autenticazione e voto utente u3

```
sl.trustStore=truststoServerAuth.jks -Djavax.net.ssl.trustStorePassword=ser
tyteam.project.SocketClientAuthentication localhost 4000
sslsocket=SSLSocket[hostname=localhost, port=4000, Session(1661619971184|TL
        Insert Fiscal Code:
fc4
        Insert UserName:
u4@gmail.com
        Please insert password for voting
psw4

You have access!
Session close
```

```
sl.trustStore=truststoServerVoting.jks -Djavax.net.ssl.trustStorePassword=serverVoting it.unisa.s
ityteam.project.SocketClientVoting localhost 4001
sslsocket=SSLSocket[hostname=localhost, port=4001, Session(1661619981332|TLS_AES_256_GCM_SHA384)]
The user can vote
Choose your voting preference for the referendum:
yes             white                   no
yes
        Request to add vote
Vote added
        Session close
```

Figura 22: Procedura autenticazione e voto utente u4

```
apaolillo@apapaolillo-virtual-machine:~/NetBeansProjects/ProjectCyberSecurity/src$ java -Djavax.net.
sl.keyStore=keystoreServerRec.jks -Djavax.net.ssl.keyStorePassword=serverRec it.unisa.securityteam.p
oject.ServerRecostructionResult
The referendum has ended. Waiting for authority connection
Waiting for connections...
```

```
apaolillo@apapaolillo-virtual-machine:~/NetBeansProjects/ProjectCyberSecurity/src$ java -Djavax.net.s
sl.trustStore=truststoServerRec.jks -Djavax.net.ssl.trustStorePassword=serverRec it.unisa.securitytea
m.project.ServerVotingRecostruction localhost 4000
Partial secret key sent successfully
```

```
apaolillo@apapaolillo-virtual-machine:~/NetBeansProjects/ProjectCyberSecurity/src$ java -Djavax.net.s
sl.trustStore=truststoServerRec.jks -Djavax.net.ssl.trustStorePassword=serverRec it.unisa.securitytea
m.project.ServerAuthenticationRecostruction localhost 4000
Partial secret key sent successfully
```

```
The referendum has ended. Waiting for authority connection
Waiting for connections...
new connection

Partial secret key acquired
Waiting for connections...
new connection

Partial secret key acquired
Scrutiny Phase started
The yes won the referendum.
Number of yes: 1
Scrutiny Phase ended
```

Figura 23: Procedura Ricostruzione SK e calcolo risultato Referendum

```
Partial secret key acquired
Scrutiny Phase started
The no won the referendum.
Number of no: 2
Scrutiny Phase ended
```

```
Partial secret key acquired
Scrutiny Phase started
The referendum did not have a majority
Result: 0
Scrutiny Phase ended
```

Figura 24: Altri casi di risultato per il Referendum

Comandi per la generazione dei certificati:

keytool -genkey -noprompt -trustcacerts -keyalg ec -groupname secp256r1 -sigAlg SHA256withECDSA -alias sslAuth -dname "cn=localhost, ou=DIEM, o=unisa, c=IT" -keypass serverAuthpwd -keystore keystoreServerAuth.jks

keytool -export -alias sslAuth -storepass serverAuthpwd -file sslServerAuth.cer -keystore keystoreServerAuth.jks

keytool -import -v -trustcacerts -alias sslAuth -keystore truststoServerAuth.jks -file sslServerAuth.cer -keypass serverAuthpwd

****
keytool -genkey -noprompt -trustcacerts -keyalg ec -groupname secp256r1 -sigAlg SHA256withECDSA -alias sslVoting -dname "cn=localhost, ou=DIEM, o=unisa, c=IT" -keypass serverVoting -keystore keystoreServerVoting.jks

keytool -export -alias sslVoting -storepass serverVoting -file sslServerVoting.cer -keystore keystoreServerVoting.jks

keytool -import -v -trustcacerts -alias sslVoting -keystore truststoServerVoting.jks -file sslServerVoting.cer -keypass serverVoting

****
keytool -genkey -noprompt -trustcacerts -keyalg ec -groupname secp256r1 -sigAlg SHA256withECDSA -alias sslDealer -dname "cn=localhost, ou=DIEM, o=unisa, c=IT" -keypass serverDealer -keystore keystoreServerDealer.jks

keytool -export -alias sslDealer -storepass serverDealer -file sslServerDealer.cer -keystore keystoreServerDealer.jks

keytool -import -v -trustcacerts -alias sslDealer -keystore truststoServerDealer.jks -file sslServerDealer.cer -keypass serverDealer

****
keytool -genkey -noprompt -trustcacerts -keyalg ec -groupname secp256r1 -sigAlg SHA256withECDSA -alias sslRec -dname "cn=localhost, ou=DIEM, o=unisa, c=IT" -keypass serverRec -keystore keystoreServerRec.jks

keytool -export -alias sslRec -storepass serverRec -file sslServerRec.cer -keystore keystoreServerRec.jks

keytool -import -v -trustcacerts -alias sslRec -keystore truststoServerRec.jks -file sslServerRec.cer -keypass serverRec

Comandi per l'esecuzione del progetto eseguiti in sequenza

Attenzione eseguire sempre PreliminarySetting.java prima di ogni simulazione e correggerre dirName in PreliminarySetting.java

Step One

java -Djavax.net.ssl.keyStore=keystoreServerDealer.jks -Djavax.net.ssl.keyStorePassword=serverDealer it.unisa.securityteam.project.DealerSKAU

java -Djavax.net.ssl.trustStore=truststoServerDealer.jks -Djavax.net.ssl.trustStorePassword=serverDealer it.unisa.securityteam.project.ServerAuthenticationDealer localhost 4000

java -Djavax.net.ssl.trustStore=truststoServerDealer.jks -Djavax.net.ssl.trustStorePassword=serverDealer it.unisa.securityteam.project.ServerVotingDealer localhost 4000

Step Two

```
java -Djavax.net.ssl.keyStore=keystoreServerAuth.jks -Djavax.net.ssl.keyStorePassword=serverAuthpwd
it.unisa.securityteam.project.SocketListenerAuthentication 100000
java -Djavax.net.ssl.trustStore=truststoServerAuth.jks -Djavax.net.ssl.trustStorePassword=serverAuthpwd
it.unisa.securityteam.project.SocketClientAuthentication localhost 4000

java -Djavax.net.ssl.keyStore=keystoreServerVoting.jks -Djavax.net.ssl.keyStorePassword=serverVoting
it.unisa.securityteam.project.SocketListenerVoting 100000
java -Djavax.net.ssl.trustStore=truststoServerVoting.jks -Djavax.net.ssl.trustStorePassword=serverVoting
it.unisa.securityteam.project.SocketClientVoting localhost 4001
```

Step Three

```
java -Djavax.net.ssl.keyStore=keystoreServerRec.jks -Djavax.net.ssl.keyStorePassword=serverRec
it.unisa.securityteam.project.ServerRecostructionResult
java -Djavax.net.ssl.trustStore=truststoServerRec.jks -Djavax.net.ssl.trustStorePassword=serverRec
it.unisa.securityteam.project.ServerAuthenticationRecostruction localhost 4000
java -Djavax.net.ssl.trustStore=truststoServerRec.jks -Djavax.net.ssl.trustStorePassword=serverRec
it.unisa.securityteam.project.ServerVotingRecostruction localhost 4000
```