Federico Ginosa, federico@menxit.com, 457026

Introduzione

Le elezioni sembrano semplici, ma in realtà richiedono di ottemperare un insieme di requisiti di sicurezza unici. In questo lavoro sono state analizzate queste proprietà, in relazione a un sistema di votazione elettronico basato sul protocollo Bitcoin [1]. Inoltre per comprendere al meglio quali siano le proprietà rispettate o meno dal sistema, il protocollo è stato reimplementato; in particolare non è stata utilizzata la piattaforma Bitcoin, ma Ethereum, ma ai fini della discussione delle proprietà il discorso non cambia.

Requisiti richiesti e soddisfatti

	Descrizione	**Soluzione proposta**
Ammissibilità e autenticazione	in un sistema di votazione elettronico deve poter votare solo chi è autorizzato	Sì
Verificabilità e Auditabilità	è possibile verificare che tutti i voti siano stati correttamente contabilizzati nel conteggio finale	Sì
Unicità	nessun elettore può votare più di una volta	Sì
Precisione	i sistemi elettorali dovrebbero registrare correttamente i voti, con una tolleranza agli errori estremamente ridotta	Sì
Integrità	i voti non devono poter essere modificati, falsificati o eliminati senza rilevamento	Sì
Votare in maniera anonima	né le autorità elettorali né nessuno dovrebbe essere in grado di determinare il modo in cui un individuo ha votato. L'anonimato del voto deve essere garantito anche nel caso in cui l'elettore stesso volesse infrangerlo, questo per evitare la vendita di voti	Nì
Conteggio e riconteggio	il risultato dell'elezione deve poter essere accessibile solo al termine delle votazioni.	Sì
Risultato solo a urne chiuse	il risultato dell'elezione deve poter essere accessibile solo al termine delle votazioni.	No

Introduzione a Ethereum

Ethereum è una piattaforma simile a Bitcoin. In Bitcoin gli UTXO vengono bloccati e sbloccati sulla base di un locking script e di un unlocking script. Il linguaggio di programmazione utilizzato da Bitcoin per scrivere questi script prende il nome di Script. Questo linguaggio appartiene alle grammatiche di Chomsky di tipo 2 (Automa a pila); dunque Script non è un linguaggio di programmazione turing completo. Il fatto che Script non sia un linguaggio di programmazione turing completo per molti è considerato un fattore positivo dal punto di vista della sicurezza. Infatti uno dei mantra della sicurezza è la semplicità, più un sistema è semplice, più è facile da comprendere, minori sono le probabilità di inserire al suo interno degli errori. D'altra parte scrivere dei contratti (anche complessi) con Script risulta difficile e a volte impossibile. Ethereum propone un sistema simile a Bitcoin, ma permette di scrivere dei locking e degli unlocking script con un linguaggio di programmazione turing completo, che viene eseguito dalla macchina virtuale di Ethereum. In Ethereum il locking script prende il nome di smart contract; uno smart contract ricorda fortemente il concetto di classe della OOP. Il concetto di unlocking script è stato invece sostituito con quello di invocazione dei metodi.

Ethereum Virtual Machine **(Solidity : Java = JVM : ETH VM)**

Solidity è il linguaggio di programmazione utilizzato per scrivere gli smart contract. Come per Java la vera fortuna di Java non è stato Java, ma la JVM, per Ethereum la vera fortuna di Ethereum non è stato Solidity, ma la ETH VM.
Quando scriviamo uno smart contract, esso viene compilato in un linguaggio macchina, interpretabile dalla ETH VM. Il compilatore di Solidity prende il nome di solc.

`hello, smart contract\n`

Il titolo del paragrago è un inganno, il nostro primo esempio non ricorderà per niente quello di Dennis Ritchie, ma avrà il pregio di mostrare al lettore come funziona nel dettaglio la ETH VM. Il seguente smart contract ha una sola proprietà, di tipo uint256 , di nome "a" e un costruttore. Il costruttore è un metodo particolare dello smart contract, esso viene richiamato un unica volta, nel momento in cui si effettua il deploy del contratto. In particolare questo costruttore prende la proprietà a e la inizializza ad 1.

```
contract HelloSmartContract {
   uint256 a;
   function HelloSmartContract() {
     a = 1;
   }
}
```

Uno smart contract oltre ad essere visto come una classe, può essere visto come un Database. Le informazioni memorizzate nelle proprietà, verranno infatte memorizzate permanentemente nella block chain di ethereum.

Ora possiamo compilare lo smart contract con il seguente comando:

```
solc --bin --asm HelloSmartContract.sol
```

Ottenendo questo bytecode interpretabile dalla ETH VM:
`60606040523415600e57600080fd5b5b6001600081905550`5b5b6036806026
6000396000f30060606040525b600080fd00a165627a7a72305820af3193f6
fd31031a0e0d2de1ad2c27352b1ce081b4f3c92b5650ca4dd542bb770029

La maggior parte di questo codice è ripetitivo e si trova in tutti gli smart contract. Andiamo ad analizzare solamente la parte di codice relativa all'inizializzazione della variabile, ovvero l'unico pezzo di codice dello smart contrat che va a modificare permanentemente lo storage.

```
a = 1
```

Questa assegnazione viene descritta dal seguente bytecode: **`6001600081905550`**

In particolare, nella ETH VM le istruzioni vengono eseguite dall'alto verso il basso e la memoria volatile viene gestita come uno stack. La ETH VM mette a disposizione un insieme di comandi grazie ai quali è possibile eseguire delle operazioni sullo stack. Di seguito vediamo alcuni dei principali.

60 x: pusha il valore x sullo stack
81: duplica il secondo elemento dello stack
90: swappa i primi due elementi affioranti sullo stack
55: prende i primi due elementi e effettua lo store del secondo elemento nella posizione indicata nel primo elemento e poi cancella dallo stack tutti e due valori
50: fa il pop del valore affiorante

```
60 01
60 00
81
90
55
50
```

A questo punto possiamo simulare il comportamento dello stack:

`60 01`	Pusha 1 sullo stack	`[ 1 ]`
`60 00`	Pusha 0 sullo stack	`[ 0 1 ]`
`81`	Copia il secondo elemento dello stack (1) e lo pusha.	`[ 1 0 1 ]`
`90`	Swappa i primi due elementi affioranti dello stack.	`[ 0 1 1 ]`
`55`	Effettua lo store del secondo elemento dello stack (1) nella posizione indicata dal primo elemento dello stack (0) e poi cancella tutti e due i valori.	`[ 1 ]` Store Blockchain: 0 => 1
`50`	Fa il pop dell'elemento affiorante. n.b. l'esecuzione di uno smart contract termina quando lo stack è vuoto.	`[]` Store Blockchain: 0 => 1

Paperopoli alle urne

A Paperopoli si stanno organizzando le elezioni del prossimo sindaco, ma la classe politica è stufa di sperperare i soldi in questa attività. Inoltre ogni anno Rockerduck pretende il riconteggio dei voti, in quanto non si capacita di come possa perdere e teme dei brogli elettorali; tuttavia questa attività risulta lunga e laboriosa. Servirebbe una soluzione: economica, che permetta il conteggio e riconteggio in maniera veloce, mantenendo però tutte le proprietà di sicurezza necessarie a buon un sistema di votazione. A quanto pare sembra che Archimede abbia trovato una soluzione!

Fase di pre-voting

In questa prima fase avviene il processo di approvazione dei candidati e degli elettori idonei. Nel progetto la digitalizzata del processo di approvazione dei candidati non è stata realizzata; semplicemente in fase di votazione l'elettore riceverà gli indirizzi pubblici dei candidati.

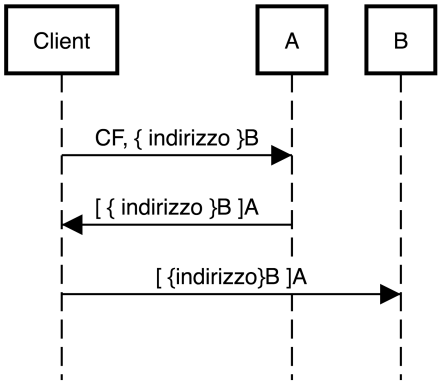
Per quanto riguarda l'approvazione degli elettori, essa è stata digitalizzata. In particolare un elettore può registrarsi, conoscendo il proprio codice fiscale. L'elettore dunque mostra il proprio codice fiscale e il proprio indirizzo pubblico e successivamente è autorizzato a votare.

Una delle proprietà fondamentali in un sistema elettorale è l'anonimato, ma visto che le transazioni sulla blockchain sono pubbliche, in questo modo sarebbe di pubblico dominio chi ha votato chi.

A tal proposito nel paper in discussione viene proposto di utilizzare Kerberos o la blind signature per marginare questo problema. Per semplificare e velocizzare l'implementazione è stato realizzato un protocollo alternativo; chiaramente in futuro si può pensare di migliorare la sicurezza di questo sistema adottando un protocollo già testato.

Il protocollo che viene proposto, identifica tre elementi:

- Client (applicazione che permette all'elettore di registrarsi)
- Verificatore delle credenziali (A)
- Registratore dell'indirizzo (B)



L'idea è questa: inizialmente il client invia un primo pacchetto ad A. Questo pacchetto contiene due informazioni:

1. CF: il codice fiscale del client

2. { indirizzo }B: l'indirizzo del proprio wallet criptato con la chiave pubblica di B

A questo punto A ricevendo queste due informazioni verifica che il codice fiscale sia valido e successivamente, cripta con la propria chiave privata { indirizzo }B.

Ora il client può presentarsi da B con [{ indirizzo }B]A, come prova del fatto che il suo codice fiscale è stato valutato come valido da A. In questo modo A può approvare il CF senza conoscere l'indirizzo di A, mentre B può registrare l'indirizzo di A come valido, senza conoscere il suo CF.

B successivamente esporrà su un endpoint pubblico tutti gli indirizzi pubblici che hanno l'autorizzazione per votare.

Fase di voting

Finalmente a Paperopoli è giorno di elezioni. Ogni elettore si è fatto approvare il proprio indirizzo come valido e può votare. Come già detto la registrazione del voto avverrà sulla blockchain di ethereum. Di seguito illustriamo lo smart contract utilizzato a tale scopo:

```
contract ElezioniAPaperopoli {

    mapping(address => bool) votazioni;
    mapping(address => address[]) candidati;

    function vota(address candidato) public {
        require(!votazioni[msg.sender]);
        votazioni[msg.sender] = true;
        candidati[candidato].push(msg.sender);
    }

    function votiByCandidato(address candidato) public returns(address[]) {
        return candidati[candidato];
    }

}
```

Lo smart contract in questione è molto semplice. In particolare sono presenti due variabili di storage:

- **votazioni**: è una classica mappa che ha come chiave un indirizzo (quello dell'elettore) e come valore un booleano. Quando un elettore vota, il valore della mappa corrispondente alla sua chiave viene settato a true. Questo serve a impedire che possa votare più di una volta.

- **candidati**: anche qui abbiamo una mappa, questa mappa ha come chiave un indirizzo (quello del candidato) e come valore un array di indirizzi (quelli dei relativi elettori).

Questo smart contract inoltre ha due metodi pubblici: vota e votiByCandidato.

Il metodo "vota" permette agli elettori di votare il loro candidato ideale. Questo metodo accetta come input l'indirizzo del candidato. Inizialmente si verifica che l'elettore non abbia già votato. In caso contrario viene sollevata un'eccezione. Successivamente si inizializza il valore relativo alla sua chiave nella mappa votazioni (questo gli impedirà di votare una successiva volta) e infine viene pushato il suo indirizzo nell'array relativo al candidato che vuole votare.

```
function vota(address candidato) public {
    require(!votazioni[msg.sender]);
    votazioni[msg.sender] = true;
    candidati[candidato].push(msg.sender);
}
```

Fase di post-voting

Le elezioni sono terminate e si vuole conoscere il vincitore. A differenza del passato, il conteggio questa volta sarà immediato. Di seguito si illustrato i vari step:

1. Come primo step occorrerà scaricare tutti gli indirizzi pubblici che hanno il diritto di votare. In particolare basterà effettuare una richiesta HTTP GET all' endpoint pubblico di B.
2. Successivamente occorrerà richiamare il metodo votiByCandidato per ciascun candidato.
3. Infine occorrerà verificare la validità di ciascun elettore e in caso positivo conteggiare il voto.

```
Promise.all([
    fetch('http://b.paperopoli.net/elettori_autorizzati.json'),
    smartContract.votiByCandidato(paperinoAddress),
    smartContract.votiByCandidato(paperoneAddress),
    smartContract.votiByCandidato(rockerduckAddress),
])
.then(([ elettoriValidi, votiPaperino, votiPaperone, votiRockerduck ]) => {

    const f = (elettoriValidi, voti) => {
```

```
        return voti
            .filter(v => elettoriValidi.indexOf(v) != -1)
            .length;
    }

    return {
        paperino:   f(elettoriValidi, votiPaperino),
        paperone:   f(elettoriValidi, votiPaperino),
        rockerduck: f(elettoriValidi, votiRockerduck),
    }
})
```

Conclusioni

Questo sistema elettorale garantisce gran parte dei requisiti necessari a un sistema elettorale elettronico. Alcuni dei punti critici sono i seguenti:

- **Autenticazione**: L'autenticazione in questa implementazione è stata realizzata con un protocollo "giocattolo". Se si dovesse andare in produzione con un sistema del genere, occorrerebbe certamente utilizzare un protocollo già testato, come la blind signature.
- **Anonimato**: L'anonimato del voto deve essere garantito anche nel caso in cui l'elettore stesso volesse infrangerlo, questo per evitare la vendita di voti. Ma da questo punto di vista purtroppo non si può fare nulla per evitare fenomeni di compravendita dei voti. Infatti la blockchain è pubblica e l'elettore possiede sia la chiave pubblica che quella privata; egli può dunque mostrare la sua chiave privata come prova del voto.
- **Risultato solo a urne chiuse**: Altra proprietà non rispettata da questo sistema elettorale è il fatto che il risultato delle votazioni si dovrebbe sapere solo a urne chiuse, mentre invece questo non è vero. Infatti le transazioni vengono pubblicate in diretta. Questo problema potrebbe in realtà essere superato. Una soluzione è quella di far creare la transazione all'elettore, fargliela firmare e infine invece che rilasciarla sulla rete P2P, inviarla a una terza entità C. Questa entità conserverà gelosamente queste transazioni già firmate e al termine delle elezioni le rilascerà sulla mainnet tutte assieme. In questo modo il risultato delle elezioni non sarebbe pubblico fino al termine dell'elezioni stesse.

Demo

Qui è presente una demo dell'applicazione: https://rebrand.ly/tesina-sicurezza

[1] Stefano Bistarelli, Marco Mantilacci, Paolo Santancini, and FrancescoSantini. An end-to-end voting-system based on bitcoin. InProceedingsof the Symposium on Applied Computing, pages 1836–1841. ACM, 201.