



Università degli Studi di Salerno

.DIEM

Corso di Laurea Magistrale in Ingegneria Informatica

APS 2022/2023

Project Work – I guardiani del bit

Iannaccone Martina (0622702102)

Minichiello Giulia (0622702127)

Ricciardi Andrea Vincenzo (0622702009)



<b>WP1. MODELLO</b>	<b>3</b>
1.1 COMPLETENESS	4
1.2 THREAT MODEL	6
1.3 PROPRIETÀ	8
1.3.1 CONFIDENZIALITÀ	8
1.3.2 INTEGRITÀ	8
1.3.3 TRASPARENZA	9
1.3.4 EFFICIENZA	9
<b>WP2. SOLUZIONE</b>	<b>10</b>
2.1 GREEN PASS 2.0	11
2.1.1 FASE DI PRE-GIOCO	14
2.2 GENERAZIONE STRINGHE CASUALI	17
2.3 JOKERCHAIN	21
<b>WP3. ANALISI</b>	<b>25</b>
3.1 CONFIDENZIALITÀ	25
3.2 INTEGRITÀ	26
3.3 TRASPARENZA	29
3.4 EFFICIENZA	30
<b>WP4. IMPLEMENTAZIONE</b>	<b>32</b>
4.1 GENERAZIONE DEI CERTIFICATI	33
4.2 SISTEMA DI GIOCO	35
4.2.1 PLAYER: ANDY	51
4.2.2 PLAYER: MARIO	52
4.2.3 PLAYER: INFANTE	53

# WP1. Modello

Workpackage	Task	Responsabile
WP1	Modello	Minichiello Giulia

Durante la pandemia, il proprietario di una Sala Bingo, Mister Joker, ha deciso di creare delle stanze virtuali online in cui le persone possono partecipare a vari giochi di fortuna, ossia giochi dove sulla base di valori casuali c'è un vincitore. Per realizzare questa idea, contatta un gruppo di studenti del corso di laurea magistrale in ingegneria informatica dell'università di Salerno per creare una funzionalità che generi in modo casuale **stringhe continue**. L'obiettivo è creare un sistema trasparente che eviti imbrogli.

Il governo ha imposto il divieto di partecipare ad eventi sociali online a chi non possiede il Green Pass. Tuttavia, il garante per la protezione dei dati personali ha vietato l'invio del Green Pass attraverso canali telematici. Il governo ha quindi pubblicato una call aperta a tutti per proposte di formato del Green Pass 2.0 che preveda l'unico uso delle informazioni strettamente necessarie per accedere a un servizio. Mister Joker ha chiesto agli studenti di creare una funzionalità per l'accesso alle sale virtuali della sua Sala Bingo, anche se non sa ancora quali informazioni saranno necessarie nel **Green Pass 2.0**. Tuttavia, questo non è un problema, perché gli studenti intendono progettare un sistema dinamico che permette al proprietario del Green Pass 2.0 di usarlo in sicurezza al variare delle politiche nel tempo circa quali dati bisogna possedere per accedere al servizio. Il Green Pass 2.0 progettato dagli studenti viene emesso dal Ministero della Salute e permette agli utenti di identificarsi con la sala Bingo per accedere al proprio profilo e, successivamente, alla funzionalità di generazione continua di stringhe casuali.

Ci sono gli oppositori dell'innovazione, chiamati *no-fox*, che puntano sui rischi delle nuove tecnologie e sono considerati allarmisti e complottisti dai sostenitori dell'innovazione. Per evitare la strumentalizzazione, è necessario che il sistema sia trasparente e aperto alla verifica da parte di tutti. Mister Joker ha sottolineato l'importanza di questa trasparenza, citando una canzone di Franco Califano che diceva: "Non mi fido di nessuno".

## 1.1 Completeness

La procedura di gioco nella sala bingo virtuale è divisa nelle seguenti fasi:

1. La fase  $[T_0, T_1]$  rappresenta la fase di pre-gioco, nella quale è prevista l'identificazione del giocatore presso l'Identity Provider della Sala Bingo di Mr. Joker.
2. La fase  $[T_1, T_2]$  rappresenta la fase di gioco, nella quale l'utente effettua una giocata.
3. La fase  $[T_2, T_3]$  rappresenta la fase di post-gioco, in cui viene mostrato a video il risultato elaborato dal sistema.

Formalizzando, siano  $C = \{C_1, \dots, C_n\}$  l'insieme dei cittadini in possesso di un Green Pass 2.0 e sia  $P = \{P_1, \dots, P_m\} \subseteq C$  l'insieme dei giocatori in possesso dei requisiti necessari per essere ammessi alla sala bingo virtuale. In altre parole, si assume che ogni giocatore  $P_i$  sia in possesso di un Green Pass 2.0 valido rilasciato, dopo opportune verifiche, da un'entità governativa di fiducia, quale **il Ministero della Sanità**. Ogni giocatore  $P_i$  avrà a disposizione una finestra temporale  $[T_0, T_1]$  entro cui identificarsi con il Green Pass 2.0.

Una volta autenticato sul portale, il giocatore  $P_i \in P$  può effettuare, durante la finestra temporale  $[T_1, T_2]$ , al più una giocata  $B_{P_i}$ . Nel caso in cui uno o più giocatori non effettuino una giocata entro una finestra temporale specificata da Mr. Joker nella fase di definizione dei requisiti, verranno esclusi dall'attuale turno di gioco. Si assume che la finestra temporale abbia una durata di 30s.

Infine, durante l'intervallo di tempo  $[T_2, T_3]$ , dopo che il sistema ha elaborato il risultato, quest'ultimo viene mostrato a ciascun giocatore sullo schermo insieme all'esito della giocata (*Vincente* o *Perdente*), elaborata direttamente dal software di gioco. Inoltre, per favorire la trasparenza del sistema, sono rese pubbliche a ogni giocatore le giocate effettuate dagli altri giocatori della sala, in modo da poter verificare successivamente la correttezza del risultato <sup>1</sup>.

---

<sup>1</sup> Come da specifiche, tutti gli aspetti legati alla componente economica (e.g., importo scommessa, eventuali vincite, etc.) sono stati esclusi dalla trattazione.

In caso di provi concrete di brogli nel sistema di gioco, si può ricorrere alla giustizia  $J$ . Anche se il sistema è stato implementato in modo da essere il più trasparente possibile, esiste sempre la possibilità, anche minima, che il sistema possa essere violato. Tuttavia, gli aspetti relativi alle possibili conseguenze penali sono stati esclusi dalla trattazione.

Gli attori coinvolti nel sistema di gioco sono i seguenti <sup>2</sup>:



**1. *m-Giocatori***, indicati con la notazione  $P = \{P_1, \dots, P_m\}$ . Essi sono ritenuti poco affidabili, in quanto potrebbero tentare di manipolare il sistema in modo da ottenere vantaggi illegittimi. Per accedere alla generazione di stringhe casuali (i.e., partecipare ad un gioco della sala bingo virtuale), devono essere in possesso di un Green Pass 2.0 valido. Anche se essi potrebbero trarre vantaggio dal compromettere il sistema, se scoperti, affronterebbero gravi conseguenze penali da parte della giustizia  $J$ .



**2. *Mr. Joker***, che per semplicità indicheremo con la notazione ***Mr Joker***. Questo attore è considerato essere affidabile al 70%, ma non gode di una fiducia totale da parte dei giocatori  $P$ . Compromettere il sistema sarebbe estremamente rischioso per la sua reputazione e la sua carriera professionale. Oltre alle potenziali perdite di fiducia, affronterebbe gravissime conseguenze penali da parte della giustizia  $J$ .



**3. *L’Agenzia delle Dogane e dei Monopoli***, indicata con la notazione ***ADM***. Essa è l’entità governativa che supervisiona e governa la blockchain su cui potrebbe essere implementato il sistema di gioco. Essendo un’entità governativa, è considerata affidabile al 99%. La sua presenza garantisce l’integrità e la trasparenza del sistema di gioco.

---

<sup>2</sup> Come attori del sistema, è possibile anche considerare gli studenti vincitori della competizione del Green Pass 2.0 (i.e., i Guardiani del Bit), i quali sono responsabili della risoluzione di eventuali problemi tecnici che potrebbero sorgere nel sistema da loro implementato. La loro integrità e competenza sono fondamentali per garantire il corretto funzionamento del sistema.

## 1.2 Threat Model

In una sala bingo virtuale possiamo identificare vari avversari, che potrebbero essere intenzionati ad attaccare il sistema. Di seguito, vengono riportati gli avversari individuati, distinguendoli in **avversari attivi** (D, *Dishonest*) e in **avversari passivi** (SH, *Semi-Honest*).

- 1. The Trickster (D):** è un soggetto in grado di alterare la propria giocata in modo da risultare vincitore, sebbene la giocata effettiva abbia avuto esito negativo. Si presume che abbia una capacità di elaborazione superiore rispetto a un utente comune, ma è comunque limitato dalle risorse economiche di un singolo individuo e dalle limitazioni delle architetture classiche.
- 2. The Insider Trader (SH):** è un soggetto interno all'organizzazione della sala bingo virtuale che sfrutta il proprio accesso privilegiato per poter ottenere informazioni sui partecipanti, sconosciute alle entità esterne al sistema. Il suo obiettivo è favorire un giocatore specifico fornendo informazioni altamente sensibili. È importante notare che l'Insider Trader opera tramite dispositivi aziendali, il che significa che potrebbe non avere una potenza di calcolo elevata rispetto ad avversari esterni che possono utilizzare risorse hardware più avanzate.
- 3. The Meddler (SH):** è un avversario interessato ad ottenere informazioni sui partecipanti e sulle loro attività di gioco. Si presume che abbia una capacità di elaborazione superiore rispetto a un utente comune, ma è comunque limitato dalle risorse economiche di un singolo individuo e dalle limitazioni delle architetture classiche.
- 4. The No-Vax (D):** è un avversario che vorrebbe essere ammesso alla Sala Bingo pur non essendo legittimato ad esserlo, in quanto sprovvisto di Green Pass 2.0. Il suo scopo è di convincere, e quindi eludere, il sistema di essere in possesso di un Green Pass 2.0 valido. Si presume che abbia una capacità di elaborazione superiore rispetto a un utente comune, ma

è comunque limitato dalle risorse economiche di un singolo individuo e dalle limitazioni delle architetture classiche.



5. **The Thief (D):** è un avversario interessato a rubare le credenziali di accesso (Green Pass 2.0) di un partecipante. Si presume che abbia una capacità di elaborazione superiore rispetto a un utente comune, ma è comunque limitato dalle risorse economiche di un singolo individuo e dalle limitazioni delle architetture classiche.



6. **The Malicious Programmer (D):** è una persona con una vasta conoscenza ed esperienza nei linguaggi di programmazione in grado di riuscire a compromettere i sistemi informatici, così da creare vantaggi impropri per sé o per altri. Si assume che tale avversario disponga di un'enorme potenza computazionale.



7. **The Overloaders (D):** sono un gruppo di avversari intenzionati a sovraccaricare il sistema durante la fase di sottomissione della giocata effettuando numerose giocate in successione, in modo da impedire agli altri partecipanti di giocare, rendendo il servizio inutilizzabile. Si presume che la loro capacità di elaborazione sia superiore rispetto a quella di un utente comune, in quanto potrebbero essere in grado di combinare più architetture, ottenendo così una potenza di calcolo estremamente elevata. Inoltre, hanno a disposizione considerevoli risorse economiche per sostenere tali attività.



8. **The Bingo Bandits (D):** sono gruppi di avversari di cui sopra, intenzionati a collaborare per arrivare ad un obiettivo comune, ossia trarre profitti. Si presume che la loro capacità di elaborazione sia superiore rispetto a quella di un utente comune, in quanto potrebbero essere in grado di combinare più architetture, ottenendo così una potenza di calcolo estremamente elevata. Inoltre, hanno a disposizione considerevoli risorse economiche per sostenere tali attività.

## 1.3 Proprietà

Per i quattro pilastri fondamentali (**confidenzialità, integrità, trasparenza ed efficienza**), vengono elencate le funzionalità che devono essere preservate in presenza di attacchi.



### 1.3.1 Confidenzialità

In presenza di avversari, il sistema di casinò dovrebbe essere *confidenziale*:

- C1. Informazioni sanitarie dell'utente:** Il sistema deve garantire la protezione delle informazioni sanitarie dell'utente. Un avversario non dovrebbe essere in grado di determinare le ragioni alla base del rilascio del GP 2.0 per un determinato giocatore  $P_i$ .
- C2. Credenziali d'accesso:** Le credenziali di accesso di un giocatore  $P_i$  devono essere completamente segrete.
- C3. Identità Nascosta:** L'identità di ogni giocatori  $P_i$  presenti in una stanza di gioco deve rimanere nascosta a tutti gli altri giocatori  $P$ .

### 1.3.2 Integrità

In presenza di avversari, il sistema di casinò dovrebbe rimanere *integro*:

- I1. Stabilità del sistema:** Il sistema deve essere ben progettato in modo che un attaccante non possa manipolare la correttezza del risultato generato dal banco  $D$ . Ciò è essenziale per garantire che il gioco sia equo e trasparente per tutti i partecipanti.
- I2. Continuità del servizio:** Il sistema deve garantire la continuità del servizio, evitando interruzioni o downtime che potrebbero causare danni ai partecipanti o al gioco stesso.
- I3. Idoneità al gioco:** Il processo di generazione di stringhe casuali deve essere garantito solo a coloro in possesso di un GP 2.0 valido e che rispettano determini requisiti richiesti da *Mr Joker*.

**I4. Unicità della giocata:** Il sistema deve garantire ai giocatori  $P$  che il proprio valore immesso sia integro per tutta la durata della giocata. Ciò significa che per nessun soggetto coinvolto nel sistema deve essere possibile, a partire da un valore lecitamente immesso nel sistema, forgiare un nuovo valore che appare come valido.

### 1.3.3 Trasparenza

Il sistema di casinò dovrebbe essere *trasparente*, ovvero:

**T1. Algoritmi segreti:** Non dovrebbe essere basato su algoritmi segreti.

**T2. Fiducia Cieca:** Per evitare facili strumentalizzazioni da parte dei *no-fox*, il sistema non dovrebbe affidarsi eccessivamente ad una presunta parte fidata.

**T3. Correttezza delle giocate:** Qualunque soggetto, compreso un giocatore  $P_i$  passivo, può verificare, al termine della fase di post-gioco, il risultato prodotto dal sistema e le giocate  $B_{P_i}$  dei vari giocatori della stanza.

### 1.3.4 Efficienza

Il sistema di casinò dovrebbe essere *efficiente*, ovvero:

**E1. Elevato numero di partecipanti:** Il sistema deve assicurare un funzionamento efficiente in caso di un elevato numero di partecipanti.

# WP2. Soluzione

Workpackage	Task	Responsabile
WP2	Soluzione	Ricciardi Andrea Vincenzo

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

- I dispositivi non sono corrotti, ovvero funzionano correttamente senza essere controllati da eventuali malware e virus.
- Ogni stanza di gioco può ospitare un numero limitato di giocatori  $P$ , pari all'ordine delle decine.
- 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). In particolare, viene considerata come Root Certification Authority la **Repubblica Italiana**, seguita da due Intermediate CA: la Certification Authority del **Ministero della Salute** per il rilascio del *GP 2.0* e la Certification Authority del **Ministero dell'Economia e delle Finanze** per il rilascio del certificato del banco  $D$  e dei nodi  $ADM$ .
- La privacy e la sicurezza dei dati per le comunicazioni tra un client e un server sono garantite dal protocollo **TLS**. Il TLS crea un canale sicuro tra i due endpoint, in modo che i dati scambiati non possano essere intercettati o modificati da terzi.

Il capitolo in questione è strutturato in tre paragrafi, ognuno dei quali presenta una descrizione dettagliata dell'argomento trattato, seguita da una fase di formalizzazione. I paragrafi in questione sono:

- **Green Pass 2.0:** include la Fase di Pre-Gioco.
- **Generazione Stringhe Casuali:** include le Fasi di Gioco e di Post-Gioco.
- **JokerChain:** include un'implementazione di *blockchain permissioned* per la gestione delle Fasi di Gioco e di Post-Gioco.

## 2.1 Green Pass 2.0

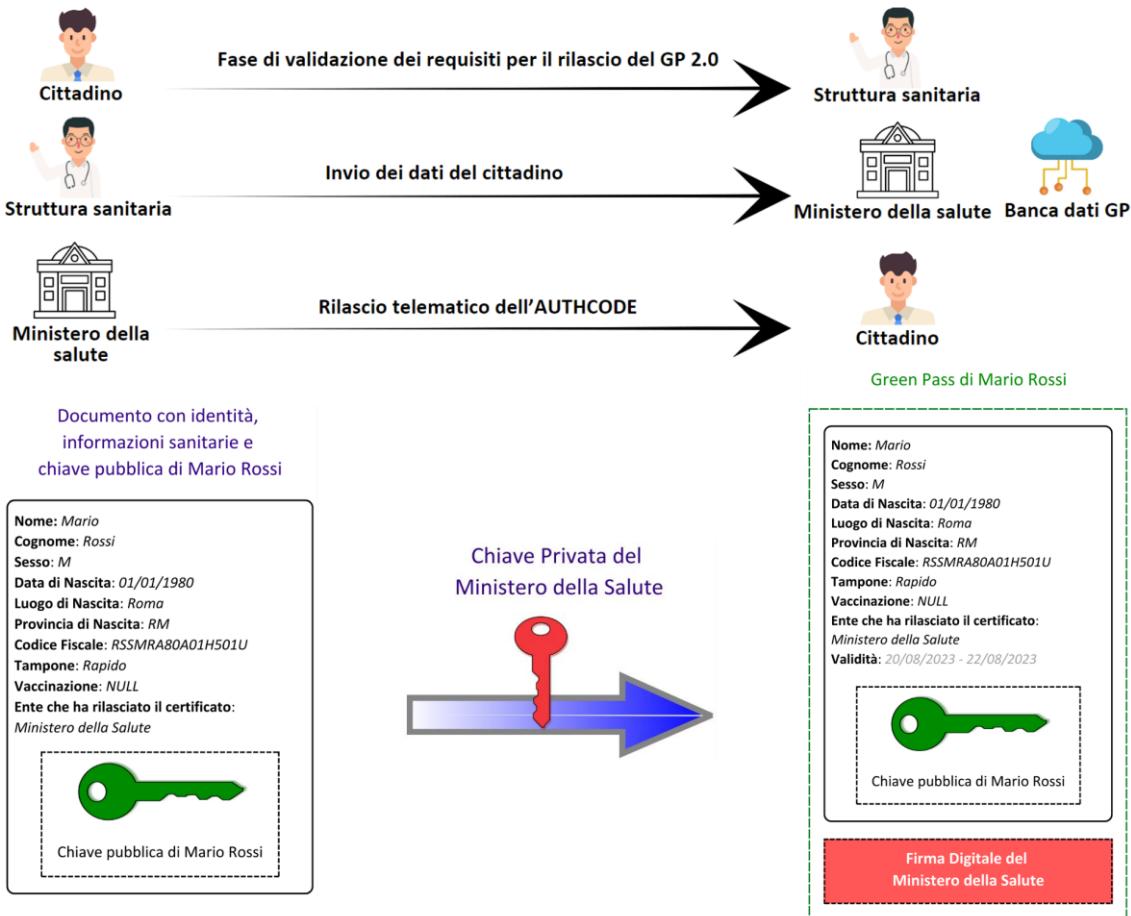
Il Green Pass (*GP*) è un certificato digitale rilasciato da una figura considerata altamente affidabile da tutti i cittadini, quale il **Ministero della Salute** (*MS*).

Per rilasciare un  $GP_{C_i}$ , il *MS* genera una coppia di chiavi  $(p_{k_{M,C_i}}, s_{k_{M,C_i}})$ , dove la chiave pubblica  $p_{k_{M,C_i}}$  è utilizzata per effettuare la verifica del  $GP_{C_i}$  del cittadino  $C_i$  da chiunque, mentre la chiave privata  $s_{k_{M,C_i}}$  viene utilizzata per firmare la richiesta di certificato caricata dall’utente.

Un possibile scenario per il rilascio del  $GP_{C_i}$  può essere descritto come segue:

- 1. Procedura di Validazione Sanitaria.** Per ottenere il  $GP_{C_i}$ , il cittadino  $C_i$  deve recarsi presso un laboratorio medico accreditato per ricevere il vaccino o per effettuare un tampone. Durante la visita, il cittadino dovrà identificarsi esibendo al personale un documento di validazione (e.g., patente, etc.). Raccolti i dati del cittadino  $C_i$ , il personale addetto li inserisce nella Piattaforma del *MS* tramite una connessione HTTPS, insieme all’esito del tampone o al certificato di avvenuta vaccinazione.
- 2. Elaborazione del GP.** Inseriti i dati, il *MS* invia al cittadino  $C_i$  un AUTHCODE come prova dell’autorizzazione a ottenere il rilascio del  $GP_{C_i}$ . Il cittadino  $C_i$  accede alla pagina web del *MS* tramite una connessione HTTPS e dopo aver effettuato l’accesso, inserendo nel form l’AUTHCODE e le ultime 8 cifre della tessera sanitaria, genera una coppia di chiavi  $(p_{k_{C_i}}, s_{k_{C_i}})$ , prima della creazione di una richiesta di certificato ( $CSR_{C_i}$ ). Essa contiene la chiave pubblica  $p_{k_{C_i}}$  e le informazioni che identificano il cittadino  $C_i$  e dovrà essere firmata usando la chiave privata  $s_{k_{C_i}}$ .
- 3. Rilascio del GP.** Ricevuta la richiesta di certificato, il *MS* provvederà a firmare il certificato con la sua chiave privata  $s_{k_{M,C_i}}$  e, quindi, al rilascio del  $GP_{C_i}$  valido, scaricabile dal cittadino  $C_i$  nella sua area privata del sito. Allo stesso tempo, il *MS* renderà pubblica la sua chiave pubblica  $p_{k_{M,C_i}}$  in modo che chiunque possa verificare la validità del  $GP_{C_i}$  di  $C_i$ .

**Fase di richiesta e di rilascio del GP**



**Fig. 2.1** – Possibile scenario per il rilascio del GP. Per semplicità è stato assunto che, indipendentemente dal modo in cui il GP venga ottenuto, tramite vaccinazione o tampone, esista un unico GP contenente le seguenti informazioni:

- **Informazioni Personaliali:** indicano le informazioni personali del cittadino possessore del GP, quali nome, cognome, sesso, data, luogo e provincia di nascita, e codice fiscale.
- **Tampone:** indica il tipo di tampone effettuato: Molecolare o Rapido. Nel caso in cui il GP venga emesso in seguito a una vaccinazione, il campo rimarrà vuoto (NULL).
- **Vaccino:** indica il tipo di vaccino somministrato. Se il GP viene rilasciato a seguito di un tampone, il campo Vaccino rimarrà vuoto (NULL). Invece, nel caso in cui il cittadino, precedentemente positivo al virus e già vaccinato, risulti negativo, verrà generato un nuovo GP valido a partire dalla data dell'ultima somministrazione del vaccino.
- **Validità:** indica il periodo di validazione del GP. Quest'ultimo varia a seconda di come il GP viene ottenuto.
  - Nel caso di un Tampone Rapido, il GP avrà una durata di 48 ore dal rilascio.
  - Nel caso di un Tampone Molecolare, il GP avrà una durata di 72 ore dal rilascio.
  - Nel caso di Vaccino, il GP avrà una durata di 180 giorni dal rilascio.
- **Ente che ha rilasciato la certificazione:** indica l'ente affidabile che ha rilasciato il GP.
- **Identificativo Univoco del Certificato:** indica l'identificativo numerico della chiave pubblica del cittadino.

## Formalizzazione

Di seguito, viene formalizzato il processo di generazione della richiesta del  $GP_{C_i}$  da parte del cittadino  $C_i$  e del rilascio dello stesso da parte del  $MS$ , dopo aver effettuato l'accesso al sito del  $MS$  tramite una connessione HTTPS.

1. Il cittadino  $CT_i$  genera una coppia di chiavi pubblica e privata

$$(p_{k_{C_i}} = \langle G, p, q, g, y = g^x \bmod p \rangle, s_{k_{C_i}} = x) \leftarrow Gen(1^n) \quad x \in \mathbb{Z}^q$$

e crea, poi, una richiesta di certificato  $CSR_{C_i}$  contenente le informazioni che identificano il cittadino  $C_i$  e la chiave pubblica  $p_{k_{C_i}}$ . Tale certificato viene firmato usando la chiave privata del cittadino  $C_i$ :

$$\sigma \leftarrow Sign_{s_{k_{CT_i}}}(CSR_{C_i})$$

2. Il  $MS$  verifica tramite la firma del cittadino  $C_i$  la veridicità della chiave pubblica  $p_{k_{C_i}}$ , ossia  $Vrfy_{p_{k_{C_i}}} (CSR_{C_i}, \sigma) \stackrel{?}{=} 1$ .
3. Una volta verificata, il  $MS$  genera una coppia di chiavi pubblica e privata

$$(p_{k_{M,C_i}} = \langle G, p, q, g, y' = g^{x'} \bmod p \rangle, s_{k_{M,C_i}} = x') \leftarrow Gen(1^n) \quad x' \in \mathbb{Z}^q$$

e provvede, poi, a firmare il  $CSR_{C_i}$ . Nel caso in cui il cittadino  $C_i$  risulti positivo ad un tampone, il certificato emesso per la sua chiave pubblica deve diventare invalido. L'invalidazione è realizzata mediante la revoca esplicita del certificato da parte del  $MS$ . Per gestire la revoca si prevede che il  $MS$  includa un numero seriale in ogni certificato emesso. Pertanto, ogni certificato avrà ora la seguente forma:

$$\sigma' \leftarrow Sign_{s_{k_{M,C_i}}}(CSR_{C_i}, \#\#)$$

dove "###" rappresenta il numero seriale del certificato associato a  $C_i$ .

4. A questo punto, chiunque può verificare l'integrità del certificato, ossia il  $GP_{2.0,C_i}$ , tramite  $Vrfy_{p_{k_{M,C_i}}} (GP_{C_i}, \sigma') \stackrel{?}{=} 1$ , essendo nota  $p_{k_{M,C_i}}$ .

## 2.1.1 Fase di Pre-Gioco

Il **Garante GA** ha imposto il divieto di invio su canali telematici del *GP* per garantire la protezione dei dati personali, in quanto mostra dati personali in eccesso rispetto a quanto strettamente necessario per l'accesso ai servizi in questione. Il governo ha deciso di rilasciare un nuovo formato del *GP*, noto come ***GP 2.0***, il quale continuerà a prevedere la solita sequenza di informazioni del soggetto (cioè i dati presenti nel *GP*), associate ad un'unica firma digitale rilasciata dal *MS*, ma consentirà al cittadino  $C_i$  di esibire telematicamente solo le informazioni strettamente necessarie sulla base del contesto.

Dall'analisi delle informazioni contenute nel  $GP 2.0_{C_i}$ , viene individuato un sottoinsieme  $Sub_{GP\ 2.0_{C_i}}$  di informazioni sufficientemente rappresentative del cittadino  $C_i$ , affinché egli possa accedere al servizio di Sala Bingo Virtuale. Il sottoinsieme di informazioni individuato è il seguente:

- **Data di Nascita:** la data di nascita di un cittadino  $C_i$  gioca un ruolo importante. Questo perché ai cittadini minorenni è impedito l'accesso alla sala bingo virtuale anche se in possesso di un  $GP 2.0_{C_i}$  valido.
- **Codice Fiscale:** è un identificativo univoco assegnato ad ogni cittadino, utilizzato per verificare l'identità di una persona.
- **Validità del GP:** è importante verificare che il  $GP 2.0_{C_i}$  sia ancora valido per l'accesso ai servizi della sala bingo virtuale. Questa informazione è ottenibile controllando la data di scadenza specificata nel Green Pass.

È importante sottolineare che all'interno di una sala bingo virtuale, le informazioni riguardanti il tampone, il tipo di vaccino e l'identificativo univoco del certificato non sono necessari per l'identificazione di una persona. Ciò significa che un cittadino  $C_i$  che desidera mantenere riservatezza sulla modalità in cui ha ottenuto il Green Pass, ad esempio un cittadino che ha scelto di non vaccinarsi, può farlo senza dover divulgare tali informazioni all'interno della sala bingo virtuale. In questo modo, viene garantita la protezione della privacy e la tutela del cittadino che desidera mantenere riservate le proprie scelte personali relative alla propria salute.

## Formalizzazione

La soluzione per il design del *GP 2.0* è ispirata al funzionamento dello **SPID**<sup>3</sup>. In particolare, per accedere ai servizi forniti dalla sala bingo di *Mr Joker*, il giocatore  $P_i$  in possesso del  $GP 2.0_{P_i}$ , rilasciato come specificato in precedenza, si connette, dopo aver verificato la correttezza del certificato digitale della sala bingo, tramite una connessione HTTPS al sito di *Mr Joker*. Il giocatore  $P_i$  viene, quindi, reindirizzato all'IdP del *MS* ed esibisce telematicamente il  $GP 2.0_{P_i}$  insieme ad un *Timestamp* generato al momento della connessione. Il *Timestamp* viene emesso da una TSA (Time Stamping Authority) in modo da prevenire possibili replay attack. Assunto che il *Timestamp* non sia scaduto, se

1.  $[Vrfy_{pk_{M,P_i}}(GP\ 2.0_{P_i}, \sigma') \neq 1]$ , il requisito di correttezza del  $GP 2.0_{P_i}$  non è soddisfatto. Ciò potrebbe essere dovuto al fatto che il  $GP 2.0_{P_i}$  è scaduto oppure che il numero seriale associato a  $GP 2.0_{P_i}$  appartiene alla lista dei certificati revocati (CRL). Pertanto, è impedito l'accesso al sito a  $P_i$ .
2.  $[Vrfy_{pk_{M,P_i}}(GP\ 2.0_{P_i}, \sigma') == 1]$ , il requisito di correttezza del  $GP 2.0_{P_i}$  è soddisfatto. In tal caso, l'IdP del *MS* genera una coppia di chiavi pubblica  $p_{k_{IdP_{MS}}}$  e privata  $s_{k_{IdP_{MS}}}$  che utilizzerà per firmare il sottoinsieme di informazioni  $Sub_{GP\ 2.0_{P_i}}$  ritenuto strettamente necessario dalla sala bingo di *Mr Joker* per accedere ai suoi servizi.

$$\sigma_{Sub_{GP\ 2.0_{P_i}}} \leftarrow Sign_{s_{k_{IdP_{MS}}}}(Sub_{GP\ 2.0_{P_i}})$$

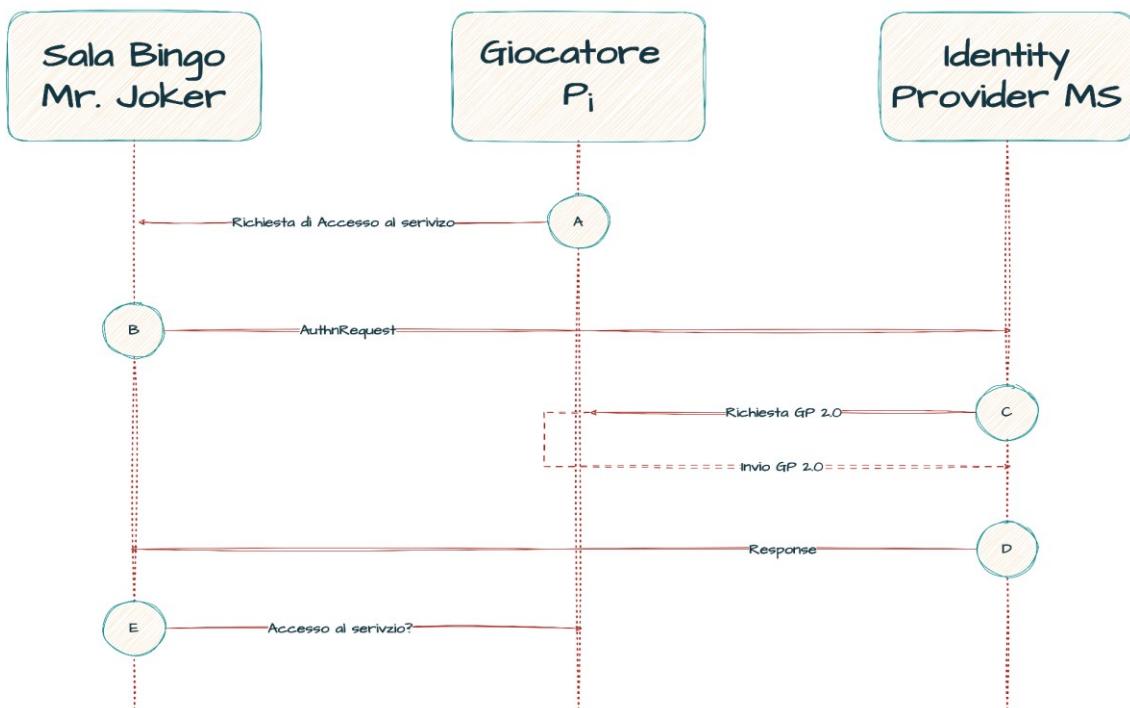
---

<sup>3</sup> L'architettura e il funzionamento dello **SPID** (Sistema Pubblico di Identità Digitale) possono essere compresi analizzando gli attori principali e il loro ruolo:

- Soggetto Interessato: Il cittadino che desidera accedere ai servizi attraverso lo SPID.
- Identity Provider: Ente accreditato che fornisce l'identità digitale (Poste Italiane).
- Fornitore di Servizi: Organizzazione che offre servizi online ai cittadini, che accetta l'identità digitale come metodo di autenticazione.

Durante il processo di autenticazione attraverso SPID, solo alcuni dati minimi e necessari del cittadino vengono inviati al Fornitore di Servizi dall'Identity Provider, dopo che questo ne ha verificato la correttezza, e ciò avviene solo dopo che il cittadino ha dato il proprio consenso. Ad esempio, se consideriamo l'**Università degli Studi di Salerno** come Fornitore di Servizi, saranno trasmessi a quest'ultimo solo i seguenti dati: `codice.identificativo`, `nome`, `cognome` e `codice.fiscale`. Questi dati rappresentano solo una parte degli attributi dello SPID, elencati nel dettaglio al link: <https://docs.italia.it/italia/spid/spid-regole-tecniche/it/stabile/attributi.html>.

Dopo che l'IdP del *MS* ha inviato il sottoinsieme di informazioni  $Sub_{GP\ 2.0_{P_i}}$  insieme ad un *Timestamp'* al sito di *Mr Joker* tramite una connessione HTTPS, quest'ultimo verifica il requisito di correttezza dei dati ricevuti  $Vrfy_{p_{k_{IDP_{MS}}}}(Sub_{GP\ 2.0_i}, \sigma_{Sub_{GP\ 2.0_{P_i}}}) \stackrel{?}{=} 1$ . Assunto che il *Timestamp'* sia valido, sulla base della risposta e dei dati forniti, il sito di *Mr Joker* decide se concedere o negare l'accesso a  $P_i$ , terminando così la fase di pre-gioco. Ad esempio, se il requisito di correttezza è valido ma la data di nascita di  $P_i$  indica che è minorenne, verrà impedito a  $P_i$  l'accesso al sito<sup>4</sup>.



**Fig. 2.2** – Meccanismo di autenticazione per la Sala Bingo di Mr. Joker ispirato a quello SPID. Viene preparato una AuthRequest, ossia una richiesta di autenticazione, che il giocatore  $P_i$  inoltra all'IdP. Eseguita l'autenticazione, l'utente torna presso il sito di Mr. Joker con un'asserzione firmata dal MS contenente gli attributi richiesti he Mr. Joker può usare per autorizzare l'utente in base alle proprie policy ed erogare il servizio richiesto.

<sup>4</sup> Il sistema del GP 2.0, così concepito, presenta una **natura dinamica**. Questo perché se le condizioni di accesso dovessero mutare, non sarebbe necessario dover richiedere al MS di rilasciare un nuovo GP 2.0. Infatti, il sito di Mr. Joker può semplicemente richiedere all'IdP del Ministero della Salute informazioni differenti, a seconda delle politiche in vigore. Ad esempio, nel caso in cui, a causa dell'aggravarsi della pandemia, si vogliano “incoraggiare” i cittadini ad effettuare la vaccinazione, si potrebbe imporre il divieto di partecipare agli eventi, a chi non ha fatto la vaccinazione. Quindi, il sito di Mr. Joker dovrà semplicemente richiedere all'IdP del Ministero della Salute che gli venga fornito anche i dati del campo “Vaccinazione”.

## 2.2 Generazione Stringhe Casuali

Una volta autenticato, al giocatore  $P_i$  viene chiesto di scegliere un nickname per la sessione di gioco. A questo punto, il giocatore  $P_i$  può effettuare, durante la finestra temporale  $[T_1, T_2]$ , al più una giocata  $B_{P_i}$ . Nel caso in cui un giocatore  $P_i$  non effettua una giocata, verrà escluso dall'attuale turno di gioco e, quindi, verrà escluso dalla generazione casuale di stringhe.

Si suppone che, per semplicità, l'unico gioco disponibile all'interno della Sala Bingo virtuale sia la roulette francese. In questo contesto, il giocatore  $P_i$  ha la possibilità di selezionare almeno una delle seguenti opzioni di gioco:

- **Numero:** il giocatore  $P_i$  può scegliere al più un numero da 0 a 36.
- **Rosso vs. Nero:** il giocatore  $P_i$  può scegliere il colore del numero estratto: rosso ( $bit = 1$ ) o nero ( $bit = 0$ ).
- **Pari vs. Dispari:** Il giocatore  $P_i$  ha la possibilità di scegliere se il numero estratto sarà pari ( $bit = 1$ ) o dispari ( $bit = 0$ ).

Una possibile rappresentazione della giocata in bit sarebbe la seguente:

Scelta numero (1 bit)	Numero (6 bit)	Scelta colore (1 bit)	Colore (1 bit)	Scelta parità (1 bit)	Parità (1 bit)
X	XXXXXX	X	X	X	X

dove il bit "Scelta \*\*\*" se settato a 1 indica che il giocatore  $P_i$  ha scommesso sulla giocata " \*\*\* ", viceversa, se settato a 0. Ad esempio, la sequenza di bit  $B_{P_i} = 10010110011$  rappresenta una giocata in cui il giocatore  $P_i$  ha scommesso che il numero estratto sarà il numero 11 o che sarà un numero pari.

Infine, al termine del turno di gioco, dopo che ogni giocatore  $P_i$  ha impegnato la giocata  $B_{P_i}$  effettuata, il banco  $D$  e ogni giocatore  $P_i$  generano una **stringa casuale  $RS_x$**  tramite un PRG, che verrà utilizzata nella fase di post-gioco per determinare il numero estratto per il turno di gioco corrente.

### Formalizzazione

Una volta effettuato l'accesso a una stanza della Sala Bingo, un giocatore  $P_i$  abilitato al gioco può effettuare al più una giocata  $B_{P_i}$ . La fase di gioco, che avviene durante la finestra temporale  $[T_1, T_2]$ , prevede le seguenti fasi:

1. All'inizio di ogni turno di gioco, il banco  $D$  genera una chiave pubblica  $k_D$  e la distribuisce tramite una connessione HTTPS a tutti i giocatori  $P$  della stanza. Questa verrà utilizzata dagli stessi giocatori per impegnare le loro giocate  $B_{P_i}$ :

$$k_D = (\mathbf{p}, \mathbf{q}, \mathbf{g}, \mathbf{h}) \quad \text{con } h \in \langle g \rangle$$

2. Entro la finestra temporale prefissata, ogni giocatore  $P_i$  impegna la propria giocata  $B_{P_i}$  e prende un  $r_{P_i} \in \mathbb{Z}_q$  uniforme. Poi, calcola il commitment, che invierà al banco  $D$ , il quale lo accetta e lo conserva <sup>5</sup>.

$$\text{com} = g^{B_{P_i}} h^{r_{P_i}} \bmod p \in \mathbb{Z}_p^*$$

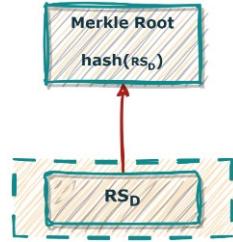
3. All'istante temporale  $T_2$ , quindi subito dopo che ogni giocatore  $P_i$  ha impegnato la giocata  $B_{P_i}$ , il banco  $D$  genera una stringa casuale  $RS_D$  tramite una PRG:

$$RS_D \leftarrow PRG(s_D, \mathbf{1}^L) \quad \text{con } s_D \in \{0,1\}^n, L > n$$

Allo stesso modo, ogni giocatore  $P_i$  che ha effettuato una giocata  $B_{P_i}$  genera una stringa casuale  $RS_{P_i}$  tramite una PRG e la invia attraverso una connessione HTTPS al banco  $D$ :

$$RS_{P_i} \leftarrow PRG(s_{P_i}, \mathbf{1}^L) \quad \text{con } s_{P_i} \in \{0,1\}^n, L > n$$

**Fig. 2.3** – Merkle Tree di altezza 1 prodotto per l'attuale turno di gioco, a cui non partecipa nessun giocatore presente nella sala. Tutto ciò garantisce continuità nel gioco, assicurando un'esperienza utente fluida e coerente. Come per lo schema di commitment, tale scelta tende a emulare l'esperienza di un casinò reale dove il croupier gira sempre la roulette, indipendentemente dal fatto che ci siano scommesse o meno.

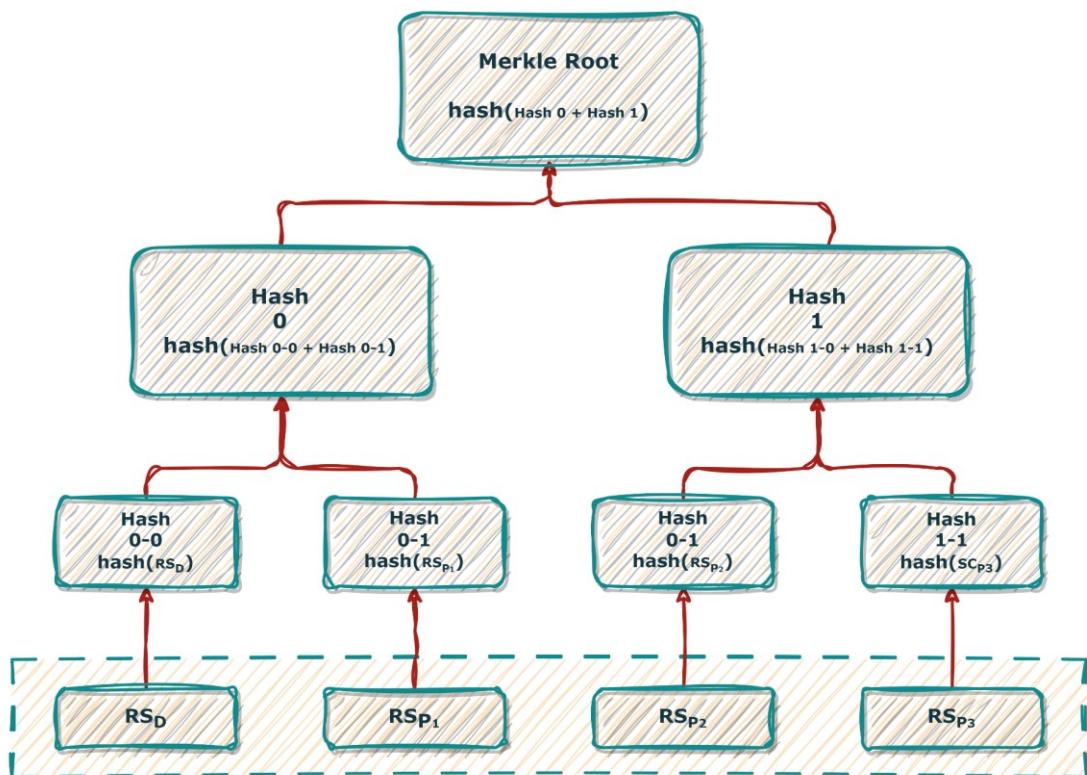



---

<sup>5</sup> La scelta di utilizzare uno schema di commitment è da rimandarsi alla volontà di emulare l'atmosfera di un casinò reale, dove le giocate degli altri giocatori sono visibili per la maggior parte dei giochi di fortuna. Ad esempio, quando un giocatore scommette su un numero alla roulette, gli altri giocatori possono vedere dove sono state piazzate le fiches. Tale meccanismo assicura che, anche in un ambiente digitale, le giocate siano trasparenti e che i giocatori non possano alterare le loro decisioni una volta fatte, proprio come avviene in un casinò reale.

Terminata la fase di gioco, inizia la fase di post-gioco identificata dall'intervallo  $[T_2, T_3]$ . Durante tale fase, ogni stringa casuale prodotta è inserita all'interno di un Merkle Tree costruito dal banco  $D$ . Un esempio di Merkle Tree è visibile nelle Fig. 2.3 e Fig. 2.4. Al completamento del Merkle Tree, si estrae la **Merkle Root**  $MR$ , che sintetizza tutte le stringhe casuali, utilizzata per determinare il numero vincente attraverso un'operazione modulo 37. Il risultato ottenuto viene comunicato a tutti i giocatori  $P$ , i quali invieranno a  $D$  la coppia  $(B_{P_i}, r_{P_i})$  così che il banco possa verificare l'integrità del commitment e, quindi, determinare gli eventuali vincitori dell'attuale turno di gioco.

Infine, al termine della fase di post-gioco ogni giocatore  $P_i$  può effettivamente verificare che non è stata commessa alcuna manomissione, andando a confrontare la Merkle Root calcolata  $MR'$  da  $P_i$  con quella fornita dal banco  $D$ . Se si identifica una discrepanza, allora i giocatori  $P$  possono far intervenire un **meccanismo di giustizia**  $J$  per indagare su potenziali irregolarità. Ciò assicura un certo grado di sicurezza e di fiducia nel sistema per tutti i giocatori<sup>6</sup>.



**Fig. 2.4** – Merkle Tree di altezza 2 prodotto per l'attuale turno di gioco, a cui partecipano 3 giocatori presenti nella sala.

<sup>6</sup> Come già specificato nel WP1, gli aspetti relativi alle possibili conseguenze penali su Mr. Joker, i giocatori P e sul sistema di casinò sono escluse dalla trattazione.

## Formalizzazione

La fase di post-gioco, che avviene durante la finestra temporale  $[T_2, T_3]$ , prevede le seguenti fasi:

1. Il banco  $D$ , una volta raccolte tutte le stringhe casuali dei giocatori  $RS_{P_i}$ , inizia a costruire il Merkle Tree. Le stringhe casuali, compresa quella generata dal banco  $RS_D$ , fungono da foglie del Merkle Tree. Il banco calcola gli hash di ogni coppia di foglie e continua a costruire l'albero fino a quando non rimane un solo hash, che è la **Merkle Root**, utilizzando come funzione hash crittografica  $SHA256$ :

$$MR = SHA256 \left( \dots SHA256 \left( SHA256(RS_D), SHA256(RS_{P_1}) \right), \dots \right)$$

2. Il numero estratto dal banco  $D$  viene calcolato attraverso un'operazione di modulo 37 a partire dal valore del Merkle Root:  $n = MR \bmod 37$ .
3. Il banco  $D$  invia il numero estratto a tutti i giocatori  $P$  attraverso una connessione HTTPS. A questo punto, ogni giocatore  $P_i$  rivela a  $D$  il valore di  $B_{P_i}$  e di  $r_{P_i}$  inviando  $(B_{P_i}, r_{P_i})$  a quest'ultimo.
4. Il banco  $D$  rende pubbliche a ogni giocatore  $P_i$  le giocate  $D_{P_i}$  effettuate dagli altri giocatori della sala, in modo che tutti possano verificare la correttezza del risultato. Inoltre, fornisce anche la Merkle Root e la Merkle Proof, che rappresenta il percorso di hash dalla stringa casuale del giocatore  $P_i$  alla Merkle Root, in modo che ogni  $P_i$  possa verificare che la sua stringa sia stata inclusa nel Merkle Tree confrontando la Merkle Root calcolata con quella fornita dal server.

## 2.3 JokerChain



In termini di efficienza l'approccio proposto è oneroso, il che può portare a ritardi del sistema durante l'esecuzione del programma. Questo perché sono state implementate varie soluzioni crittografiche che implicano un aumento dei costi del sistema, soprattutto al crescere del numero di giocatori. Per mitigare al problema dell'efficienza e garantire un maggior grado di trasparenza, la soluzione è quella di impiegare una blockchain permissioned, nota come **JokerChain**. Non essendo *Mr Joker* un attore affidabile al 100%, la governance della rete viene affidata ad un'entità riconosciuta affidabile e sicura da tutti i giocatori  $P_i$  della sala, come l'**Agenzia delle Dogane e dei Monopoli**, abbreviata con **ADM**, essendo un ente governativo<sup>7</sup>. La blockchain è quindi costituita da diverse tipologie di blocchi e transazioni:



**1. Blocco Genesi:** è il primo blocco della *JokerChain*, il quale contiene due tipologie di transazioni:

a. La transazione delle chiavi pubbliche dei certificati dei nodi ADM, utilizzate in primis dal banco e poi dai giocatori per instaurare una connessione HTTPS con gli stessi nodi *ADM*, così da poter procedere al processo di gioco.

$$Tr_1 = p_{k_{Cert_{ADM}_i}}$$

b. La transazione della chiave pubblica del certificato del banco D, che permette al nodo *ADM*, a cui il banco *D* vuole connettersi, di poter verificare la correttezza del certificato digitale del banco *D*, in modo che questo possa instaurare una connessione con il nodo *ADM*. Ovviamente, il banco *D* deve verificare la correttezza del certificato digitale del nodo banco *ADM* per poter instaurare la connessione.

$$Tr_1 = p_{k_{Cert_D}}$$

---

<sup>7</sup> Possiamo prevedere che i nodi *ADM* vengano distribuiti in varie aree geografiche della penisola, così da poter ridurre la latenza e migliorare l'esperienza dell'utente.

**2. Blocco Chiavi:** è il blocco contenente le transazioni delle chiavi di gioco.

Include la chiave pubblica del Banco  $D$  utilizzata dallo stesso per firmare la propria stringa casuale e le chiavi pubbliche dei giocatori  $P_i$  ammessi alla sala, generate per firmare le rispettive giocate e stringhe casuali.



$$Tr_2 = p_{k_D} \quad \text{e} \quad Tr_2 = p_{k_{P_i}}$$

**3. Blocco Giocate:** è il blocco contenente le transazioni di gioco. Quando

un giocatore  $P_i$  effettua una giocata  $B_{P_i}$ , firmata dallo stesso usando la sua chiave privata  $s_{k_{P_i}}$ , questa viene registrata in una transazione:



$$Tr_3 = B_{P_i}$$

**4. Blocco Stringhe:** è il blocco contenente le transazioni di stringhe casuali.



Sia il banco  $D$  che i giocatori  $P_i$ , che hanno effettuato una giocata  $B_{P_i}$ , generano e inviano al nodo  $ADM$  le stringhe casuali prodotte  $RS_x$  tramite una  $PRG$ , firmate dagli stessi usando le rispettive chiavi private  $s_{k_x}$ :

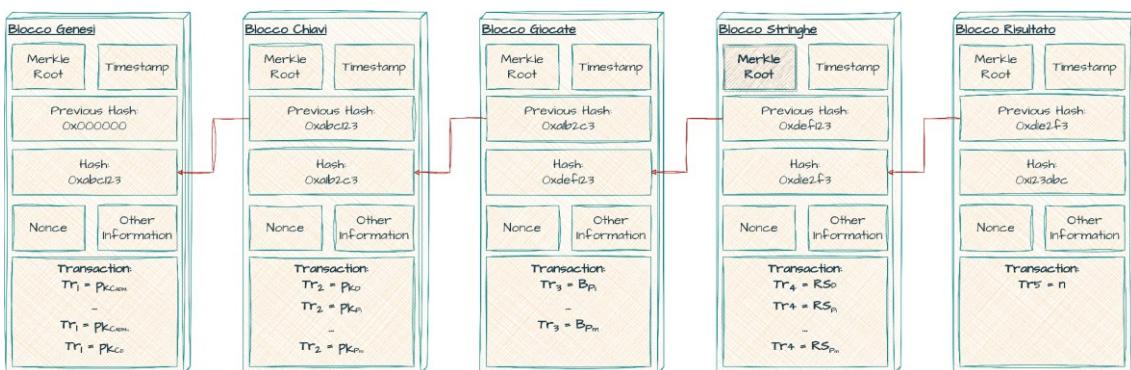
$$Tr_4 = RS_D \quad \text{e} \quad Tr_4 = RS_{P_i}$$

**5. Blocco Risultato:** è il blocco contenente la transazione risultato.



In pratica, il banco legge la Merkle Root  $MR$  memorizzata nell'header del Blocco Stringhe e la sottopone a un'operazione modulo 37, per poter determinare il risultato dell'attuale turno di gioco, che verrà firmato dal banco  $D$  con la sua chiave privata  $s_{k_D}$ .

$$Tr_5 = n \quad (= MR \bmod 37)$$



**Fig. 2.5** – Architettura della JokerChain. Dopo il Blocco Risultato, inizierà un nuovo turno di gioco, il quale richiede la creazione di un nuovo Blocco Chiavi, seguito da un nuovo Blocco Giocate, da un nuovo Blocco Stringhe e un nuovo Blocco Risultato, e così via. Una possibile rappresentazione del Merkle Root del Blocco Stringhe è presente in Fig. 2.4.

## Formalizzazione

Un giocatore  $P_i$ , dopo aver esibito il  $GP 2.0_{P_i}$  sul sito di *MrJoker* e dopo che questo ne ha verificato la correttezza, verifica l'autenticità del certificato del nodo *ADM* prima di connettersi allo stesso. Una volta verificato, il giocatore può stabilire una connessione sicura HTTPS con il nodo *ADM*.

Prima di procedere con qualsiasi azione, il giocatore verifica l'integrità e la cronologia della blockchain assicurandosi che l'ultimo blocco della *JokerChain* sia immediatamente precedente al Blocco Autenticazione. In tal caso, il giocatore  $P_i$  genera una coppia di chiavi, pubblica  $p_{k_{P_i}}$  e privata  $s_{k_{P_i}}$ . Dopo che tutti i giocatori  $P$ , compreso il banco  $D$ , hanno inviato le loro chiavi pubbliche ai rispettivi nodi *ADM*, questi nodi collaborano per raggiungere un consenso e pubblicare il Blocco Chiavi, contenente tutte le chiavi pubbliche generate sotto forma di transazioni<sup>8</sup>.

A questo punto, nella finestra temporale  $[T_1, T_2]$ , il giocatore  $P_i$  può effettuare al più una giocata  $B_{P_i}$  che verrà, dopo essere stata firmata da  $P_i$  con la sua chiave privata  $\sigma_{B_{P_i}} = \text{Sign}_{s_{k_{P_i}}}(B_{P_i})$ , inviata al nodo *ADM* al quale è collegato  $P_i$ . Il nodo *ADM* verifica la correttezza della giocata  $Vrfy_{p_{k_{P_i}}}(B_{P_i}, \sigma_{B_{P_i}}) \stackrel{?}{=} 1$  e, se dimostrata tale, provvederà ad inserirla nel Blocco Giocate, che verrà aggiunto alla *JokerChain* solo dopo che tutti i nodi *ADM* hanno raggiunto il consenso.

All'istante temporale  $T_2$ , il giocatore  $P_i$ , nel caso in cui ha effettuato una giocata  $B_{P_i}$ , genererà una stringa casuale tramite  $PRG$ ,  $RS_{P_i} \leftarrow PRG(s, 1^L)$ , che viene firmata da  $P_i$  con la sua chiave privata  $\sigma_{RS_{P_i}} = \text{Sign}_{s_{k_{P_i}}}(RS_{P_i})$ .

---

<sup>8</sup> Essendo *JokerChain* una blockchain permissioned, il **problema del consenso** è più semplice rispetto ad una blockchain permissionless. Tale architettura prevede che ogni nodo *ADM* riceva transazioni dai giocatori  $P$  o dal banco  $D$  a cui è connesso, di validare alcune di queste transazioni, andando a controllare le firme, e di trasmetterle agli altri nodi *ADM* nella rete. Un **nodo leader ADM** propone un nuovo Blocco X una volta che ha raccolto tutte le transazioni. Gli altri nodi *ADM* ricevono il blocco proposto e lo verificano. Se un nodo *ADM* concorda con il contenuto dei  $X$ , invia un voto positivo al nodo leader *ADM*, altrimenti un voto negativo. Una volta che il nodo leader *ADM* riceve i voti da **almeno 2/3 dei nodi ADM** e la maggioranza di questi è positiva, il blocco  $X$  viene accettato. Per evitare che un singolo nodo diventi un collo di bottiglia o un punto di attacco, il ruolo del leader viene ruotato tra i nodi. Essendo *ADM* affidabile al 90%, la probabilità che un nodo *ADM* agisca in modo malevolo è estremamente bassa. Pertanto, il numero possibile di nodi difettosi è ben al di sotto del limite tollerabile.

Dopo aver generato e firmato le loro stringhe casuali, tutti i giocatori, compreso il banco  $D$ , le inviano ai rispettivi nodi  $ADM$ , che, dopo averne verificato la correttezza, collaborano per raggiungere un consenso e pubblicare il Blocco Stringhe, contenente tutte le stringhe casuali generate sotto forma di transazioni.

Nell'intervallo temporale  $[T_2, T_3]$ , il banco  $D$  legge il Merkle Root del Blocco Stringhe ed effettua un'operazione modulo 37 per stabilire il risultato per il turno di gioco corrente  $n = MR \bmod 37$ . Tale numero, dopo essere stato firmato dal banco  $D$  con la sua chiave privata  $\sigma_n = \text{Sign}_{sk_D}(n)$ , viene inviato al nodo  $ADM$  al quale è collegato  $D$ , il quale verifica la correttezza della giocata e, se dimostrata tale, provvederà ad inserirla nel Blocco Risultato, aggiunto alla *JokerChain* solo dopo che tutti i nodi  $ADM$  hanno raggiunto il consenso.

# WP3. Analisi

Workpackage	Task	Responsabile
WP3	Analisi	Iannaccone Martina

Il WP3 ha lo scopo di effettuare un'analisi della soluzione presentata nel WP2 rispetto al modello presentato nel WP1. In particolare, richiede di discutere il modello proposto in termini dei quattro pilastri fondamentali: confidenzialità, integrità, trasparenza ed efficienza. La struttura analizzata è quella che prevede l'utilizzo della *JokerChain*.

## 3.1 Confidenzialità

Di seguito, vengono discussi uno per uno gli attaccanti che potrebbero compromettere le proprietà di confidenzialità del sistema definite nel WP1:

**C1. Informazioni sanitarie dell'utente:** Il sistema deve garantire la protezione delle informazioni sanitarie dell'utente. Un avversario non dovrebbe essere in grado di determinare le ragioni alla base del rilascio del *GP* 2.0 a un determinato giocatore  $P_i$ .

---

I possibili avversari interessati a compromettere la proprietà C1 sono l'**Insider Trader**, il **Thief** e il **Malicious Programmer**. Per garantirla, il sistema è stampo implementato in modo che il giocatore  $P_i$  che vuole accedere al sito di *Mr Joker* presenti il suo  $GP_{2.0_{P_i}}$ , contenete i dati personali e sanitari, all'Identity Provider del *MS*. In questo modo, il *GP* 2.0 non viene mai completamente passato al sito di *Mr Joker*. Sarà, poi, l'IdP del *MS* a fornire al sito di *Mr Joker* il sottoinsieme delle informazioni  $Sub_{GP_{2.0_{P_i}}}$  necessarie al sito di *Mr Joker* per convalidare o meno l'accesso al sito.

**C2. Credenziali d'accesso:** Le credenziali di accesso di un giocatore  $P_i$  devono essere completamente segrete.

---

I possibili avversari interessati a compromettere la proprietà **C2** sono l'**Insider Trader**, il **Thief** e il **Malicious Programmer**. Tale proprietà viene garantita in quanto quando il giocatore  $P_i$  esibisce il  $GP\ 2.0_{P_i}$  telematicamente, lo fa su una connessione sicura HTTPS che garantisce che il certificato inviato all'IdP del  $MS$  venga cifrato e non venga intercettato da avversari.

**C3. Identità Nascosta:** L'identità di ogni giocatori  $P_i$  presenti in una stanza di gioco deve rimanere nascosta a tutti gli altri giocatori  $P$ .

---

I possibili avversari interessati a compromettere la proprietà **C3** sono l'**Insider Trader**, il **Meddler** e il **Malicious Programmer**. La proprietà è assicurata per il Meddler in quanto è impossibile risalire all'identità associata a  $P_i$ , essendo richiesto a  $P_i$  di utilizzare un nickname per la sessione di gioco corrente. Tuttavia, il Malicious Programmer e l'Insider Trader potrebbero essere in grado di compromettere tale proprietà, soprattutto se il primo dovesse attaccare il dispositivo di  $P_i$ .

## 3.2 Integrità

Di seguito, vengono discussi uno per uno gli attaccanti che potrebbero compromettere le proprietà d'integrità del sistema descritte nel WP1:

**I1. Stabilità del sistema:** Il sistema deve essere ben progettato in modo che un attaccante non possa manipolare la correttezza del risultato generato dal banco  $B$ . Ciò è essenziale per garantire che il gioco sia equo e trasparente per tutti i partecipanti.

---

Un possibile avversario interessato a compromettere la proprietà **I1** è il **Malicious Programmer**. Essa viene assicurata in quanto, essendo la blockchain trasparente, chiunque può controllare il valore del Merkle Root del Blocco Stringhe. Pertanto, anche nel caso in cui un avversario riuscisse a compromettere il risultato, i giocatori possono verificare la correttezza del risultato, essendo questo ricavato applicando al valore del Merkle Root del Blocco Stringhe un'operazione modulo 37. Nel caso in cui fosse evidenziata una discrepanza, allora i giocatori  $P$  possono richiedere l'intervento della giustizia  $J$ .

**I2. Continuità del servizio:** Il sistema deve garantire la continuità del servizio, evitando interruzioni o downtime che potrebbero causare danni ai partecipanti o al gioco stesso.

---

Un possibile avversario interessato a compromettere tale proprietà sono gli **Overloaders**. La continuità del servizio viene garantita dalla specifica per cui il numero massimo di partecipanti per ogni stanza è dell'ordine delle decine. In questo caso, il numero di giocatori in una sala influisce in maniera marginale sull'esperienza di gioco. Pertanto, attacchi di sovraccarico del sistema sono trascurabili.

**I3. Idoneità al gioco:** Il processo di generazione di stringhe casuali deve essere garantito solo a coloro in possesso di un GP 2.0 valido.

---

Un possibile avversario interessato a compromettere la proprietà **I3** è il **No-Vax**. Tale proprietà viene rispettata in quanto affinché avvenga l'accesso all'interno della Sala Bingo è necessario che il giocatore  $P_i$  esibisca un GP 2.0 valido. Tuttavia, il No-Vax potrebbe sferrare un replay attack verso un giocatore  $P_i$  in possesso di un GP 2.0 valido. Pertanto, il sistema è stato implementato in modo che la possibilità di replay attack,

in cui un avversario registra e rinvia nuovamente l'interazione tra il giocatore  $P_i$  e l'IdP del  $MS$ , sia resa impossibile. In particolare, è stato previsto che il mittente aggiunga un *Timestamp*, il quale viene generato e firmato da un'autorità di timestamping (TSA) separata e affidabile. In questo modo, se il *Timestamp* non è entro un certo intervallo di tempo accettabile, la richiesta viene considerata non valida. Questo garantisce che anche se un avversario cattura i dati, non può utilizzarli dopo un certo periodo di tempo.

Tuttavia, nel caso in cui fosse entrato in possesso del *GP* 2.0 di un altro cittadino, il No-Vax riuscirebbe ad accedere alla Sala Bingo, eludendo così il sistema. Ciononostante, se fosse scoperto, incorrerebbe in gravissime conseguenze penali da parte della giustizia  $J$ .

- I4. Unicità della giocata:** Il sistema deve garantire ai giocatori  $P$  che il proprio valore immesso sia integro per tutta la durata della giocata. Ciò significa che per nessun soggetto coinvolto nel sistema deve essere possibile, a partire da un valore lecitamente immesso nel sistema, forgiare un nuovo valore che appare come valido.
- 

I possibili avversari interessati a compromettere la proprietà I4 sono il **Trickster** e il **Malicious Programmer**. Tale proprietà viene rispettata in quanto, una volta che il giocatore  $P_i$  ha sottomesso una giocata  $B_{P_i}$ , questa viene registrata come transazione all'interno del Blocco Giocate della *Jokerchain*. In particolare, il nodo *ADM*, a cui è associato il giocatore  $P_i$ , verifica l'autenticità della giocata  $B_{P_i}$ . Ciò viene fatto controllando la firma digitale di  $B_{P_i}$  realizzata con la chiave privata  $s_{k_{P_i}}$  del giocatore.

Poiché solo  $P_i$  conosce la sua chiave privata, il nodo *ADM* può confermare con certezza che la giocata è stata effettuata da  $P_i$ . Quindi, una volta che la giocata è stata verificata e aggiunta alla *Jokerchain*, essa diventa immutabile.

Le proprietà di confidenzialità e di integrità valgono nel caso di presenza dei **Bingo Bandits**, ossia una collusione dei possibili avversari citati.

## 3.3 Trasparenza

Il sistema implementato gode di un'elevata **trasparenza**, in quanto il sistema di gioco viene implementato attraverso la blockchain *Jokerchain*, garantendo quindi che chiunque all'interno della *Jokerchain* possa esaminare tutte le transazioni a partire dal Blocco Genesi. In questo modo, viene garantita la proprietà **T3**<sup>9</sup>.

Inoltre, Il sistema è stato progettato per garantire un alto livello di affidabilità nel funzionamento equo del gioco. Ciò viene ottenuto nuovamente grazie all'ausilio della blockchain *Jokerchain*. Inoltre, la generazione del risultato è un'operazione matematica molto semplice, che chiunque è in grado determinare. In questo modo viene garantita la proprietà **T1**, facendo fronte alle criticità poste dai *no-fox* (tecnocrati del progresso tecnologico).

Per quanto riguarda la proprietà **T2**, l'*ADM* viene considerato un attore che gode di un'alta credibilità, essendo un ente governativo. Come già espresso in precedenza, la governance di una blockchain permissioned, come *Jokerchain*, deve essere affidata solo ad organizzazioni ben note e autorizzate. *Mr Joker* avrebbe dei validi motivi per compromettere il sistema e, pertanto, non potrà godere mai di una completa fiducia, come può essere per un ente governativo.

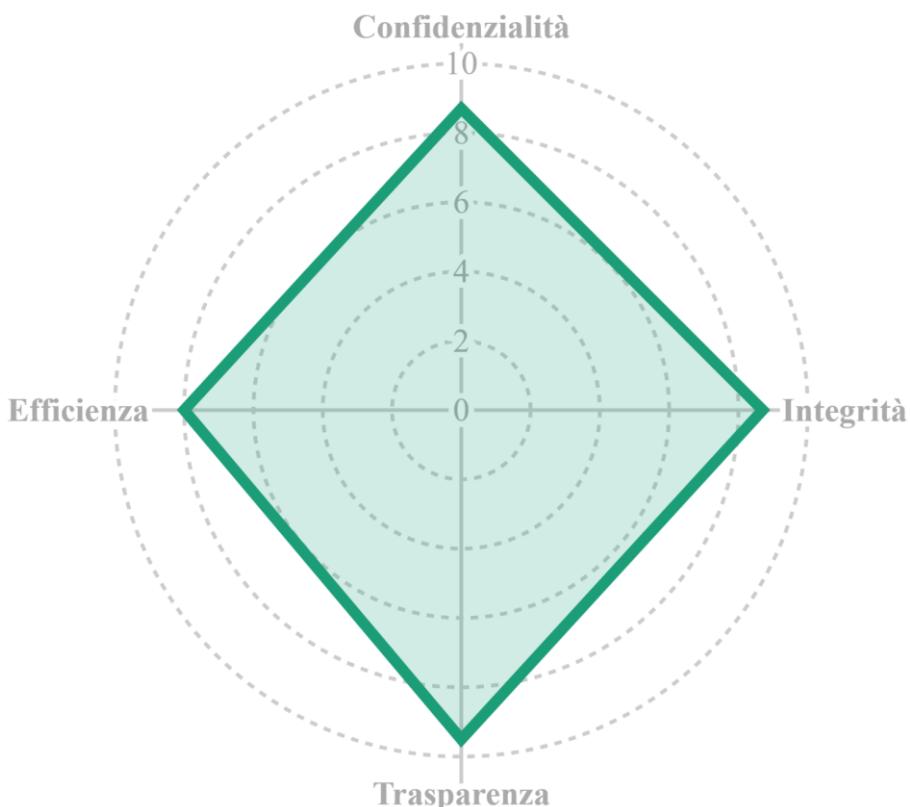
---

<sup>9</sup> Come già indicato nel capitolo precedente, la decisione di non cifrare, all'interno della *Jokerchain*, le giocate e le stringhe casuali prodotte per la generazione del risultato è da rimandarsi alla volontà di emulare l'atmosfera di un casinò reale, dove le giocate degli altri giocatori sono visibili per la maggior parte dei giochi di fortuna.

## 3.4 Efficienza

La soluzione proposta senza blockchain prevedeva l'uso di diverse tecniche crittografiche, come lo schema di commitment, e di numerose comunicazioni tra i client (giocatori  $P$ ) e il server (banco  $D$ ) che imponevano un'elevata operosità al sistema. Per garantire una migliore efficienza e al tempo stesso offrire un maggior grado di confidenzialità, integrità e trasparenza, abbiamo introdotto la blockchain permissioned *Jokerchain*. Tale soluzione, pur offrendo numerosi vantaggi, presenta alcune sfide in termini di efficienza. In particolare, la verifica e l'approvazione delle transazioni, l'aggiunta di queste transazioni alla blockchain e la creazione di nuovi blocchi possono introdurre ritardi, soprattutto nel caso di un alto numero di giocatori collegati. L'utilizzo dell'algoritmo SHA-256, che è relativamente veloce e ampiamente utilizzato, implica che le operazioni relative al calcolo del risultato siano ottimizzate.

### Mr. Joker's Casino



**Fig. 3.1** – Grafico radar dei quattro pilastri fondamentali considerati per l'analisi del sistema.

**Approfondimento – Malware e Virus**

Nel caso in cui il dispositivo di un giocatore  $P_i$  fosse compromesso da malware, virus o trojan, tutte le misure di sicurezza implementate a livello di rete o di piattaforma potrebbero risultare inutili. Ciò perché un avversario  $A$  potrebbe, attraverso il malware, intercettare, alterare o rubare dati direttamente dal dispositivo del giocatore prima che questi vengano trasmessi alla rete.

Per proteggere il dispositivo da tali minacce, un giocatore  $P_i$  può utilizzare un sistema di rilevamento delle intrusioni (IDS, *Intrusion Detection System*). Nello specifico, può impiegare un **HIDS** (Host-based IDS), il quale monitora l'attività all'interno di un dispositivo per rilevare qualsiasi comportamento sospetto. Può rilevare tentativi di accesso non autorizzati, modifiche ai file di sistema o altre attività che potrebbero indicare la presenza di malware. Se un software dannoso tenta di bypassare o disabilitare l'HIDS, il sistema è progettato per rilevarlo e bloccarlo. Per garantire che il database dell'HIDS non venga compromesso, può essere conservato su un supporto fisico separato o su un supporto che non può essere facilmente alterato. Oltre all'HIDS, un **NIDS** (Network-based Intrusion Detection System) può monitorare il traffico di rete per rilevare attività sospette. Infatti, è proprio dalla rete che arrivano i maggiori pericoli. Nello specifico, un giocatore  $P_i$  può impiegare un'Anomaly-based IDS, in cui le anomalie possono essere rilevate attraverso un'analisi comportamentale del traffico osservato in precedenza (euristica), al contrario di quanto accade nei sistemi Signature-based dove la ricerca è fatta tramite regole, tramite la ricerca di pattern o di firme caratteristiche delle violazioni di sicurezza, il quale può essere affetto da falsi negativi (non rileva attacchi classificati) e falsi positivi (allarmi erronei).

Il rilevamento precoce può permettere agli amministratori di sistema di poter intervenire rapidamente, fermare un'attività malevola in corso e prendere misure per rimuovere la minaccia.

# WP4. Implementazione

Workpackage	Task	Responsabile
WP4	Implementazione	Nessuno

Il WP4 si focalizza sull'implementazione dei sistemi progettati in WP2. In particolare, la progettazione del sistema ha tenuto conto complessivamente delle seguenti semplificazioni:

- La permissioned blockchain *JokerChain* è stata implementata come un semplice file di testo, contenente le varie tipologie di transazioni, identificati dai seguenti attributi:
  - **id** – L'**ID della transazione**, che **identifica il turno di gioco**.
  - **type** – Il **tipo di transazione**.
  - **data** – I **dati della transazione**.
  - **hash** – L'**hash della transazione**.
  - **sender** – Il **mittente della transazione**.
- La fase di autenticazione è stata caratterizzata da alcune semplificazioni rispetto alla soluzione proposta in WP2. Infatti, nell'implementazione proposta, il giocatore  $P_i$  (classe *SSLCitizen*), che vuole usufruire dei servizi offerti da *Mr Joker*, si collega all'IdP del *MS* (classe *SSLMS*), il quale provvederà poi ad inviare al fornitore dei servizi (classe *SSLJoker*) solo le informazioni strettamente necessarie del certificato scambiato da  $P_i$  con l'IdP del *MS* durante la fase di handshake. Per semplicità, non abbiamo previsto la firma dei dati inviati. Una volta elaborate, il fornitore di servizi, *Mr Joker*, deciderà se fornire o meno l'accesso al giocatore  $P_i$ .
- La fase di gioco è stata implementata così come descritta nel paragrafo JokerChain, tenendo conto delle semplificazioni sull'implementazione della blockchain. In particolare, il sistema prevede l'inizializzazione del banco *D* (classe *SSLBanco*) e dei player *P* (classe *SSLPlayer*), dove oltre al player  $P_i$  che ha effettuato l'accesso in precedenza, consideriamo due ulteriori player, *Alice* e *Bob*, che assumiamo che siano in possesso dei requisiti previsti. A questo punto inizia la fase di gioco, dove il banco *D* e i player *P* comunicano con il nodo *ADM* (classe *SSLADM*) che funge da server, il quale provvede ad inserire nella blockchain le transazioni

contenenti i dati ricevuti, per alcuni dei quali, così come descritto in WP2, ne verifica la firma in modo da assicurare che i dati (i.e., giocate, stringhe e risultato) provengano da una fonte attendibile, garantendone quindi l'integrità e la non ripudiabilità.

Inoltre, abbiamo previsto che il giocatore  $P_i$  fosse già in possesso di un *GP* 2.0 rilasciato dal *MS*. Quindi, la fase di rilascio ed elaborazione del *GP* 2.0 è stata implementata attraverso i comandi *openssl*, descritti nel successivo paragrafo.

## 4.1 Generazione dei Certificati

Per effettuare l'accesso al sito, il giocatore  $P_i$  deve essere in possesso di un *GP* 2.0 rilasciato dal *MS*. In particolare, abbiamo modificato il file *openssl.cnf*, denominandolo in *opensslGP.cnf*, in modo da aggiungere ulteriori campi al certificato. A tal fine, abbiamo aggiunto i campi previsti dal *GP*, così come mostrato in Fig. 2.1, generando per ognuno di essi un **OID** (Object Identifier).

```

1 # ...
2 # Extra OBJECT IDENTIFIER info:
3 oid_section = new_oids
4 [ new_oids ]
5 # Sottogruppo informazioni personali
6 nome          = 1.2.3.1.1
7 cognome       = 1.2.3.1.2
8 sesso          = 1.2.3.1.3
9 data_di_nascita = 1.2.3.1.4
10 luogo_di_nascita = 1.2.3.1.5
11 provincia_di_nascita = 1.2.3.1.6
12 codice_fiscale = 1.2.3.1.7
13 # Sottogruppo salute
14 tampone       = 1.2.3.2.1
15 vaccino        = 1.2.3.2.2
16 # Organizzazione
17 organizzazione = 1.2.3.3
18
19 #...
20 [ req_distinguished_name ]
21 # Campi X509 classici.
22 # ...
23
24 nome          = "Nome del richiedente"
25 cognome       = "Cognome del richiedente"
26 sesso          = "Sesso del richiedente (M o F)"
27 data_di_nascita = "Data di Nascita del richiedente nel formato dd-mm-yyyy"
28 luogo_di_nascita = "Luogo di Nascita del richiedente"
29 provincia_di_nascita = "Provincia di Nascita del richiedente"
30 codice_fiscale = "Codice Fiscale del richiedente"
31 tampone       = "Tipologia di tampone effettuato dal richiedente (Molecolare, Rapido, NULL)"
32 vaccino        = "Tipologia di vaccino effettuato dal richiedente (BioNTech-Pfizer, Moderna, ..., NULL)"
33 organizzazione = "Ente che rilascia il certificato"

```

**Listing 4.1** – Codice parziale del file *opensslGP.cnf*. La versione completa è disponibile nello zip contenente il codice totale.

Un ulteriore semplificazione è stata quella di considerare il *MS* come CA Root per il rilascio dei *GP* 2.0 e il *ME* come CA Root per il rilascio dei certificati del sito di *Mr Joker*, coincidente con il certificato del banco *D*, e dei nodi *ADM*.

Per la generazione dei certificati rilasciati dal Ministero dell'Economia e delle Finanze, *ME*, abbiamo considerato il classico file *openssl.cnf*, denominandolo per l'occasione in *opensslME.cnf*.

```

1 # Generazione parametri
2 openssl ecparam -name prime256v1 -out prime256v1.pem
3
4 # Generazione Chiavi Private
5 openssl genpkey -paramfile prime256v1.pem -out ms_key.pem
6 openssl genpkey -paramfile prime256v1.pem -out me_key.pem
7 openssl genpkey -paramfile prime256v1.pem -out joker_key.pem
8 openssl genpkey -paramfile prime256v1.pem -out adm_key.pem
9
10
11 # Generazione certificati self-signed
12 openssl req -new -x509 -days 365 -key ms_key.pem -out ms_cert.pem -config "C:\msys64\mingw64\etc\ssl\opensslGP.cnf"
13 openssl req -new -x509 -days 365 -key me_key.pem -out me_cert.pem -config "C:\msys64\mingw64\etc\ssl\opensslME.cnf"
14
15 # Generazioni Cartelle
16 mkdir MS MS/private MS/newcerts
17 touch MS/index.txt MS/serial
18 echo "00" >> MS/serial
19 cp ms_cert.pem MS
20 cp ms_key.pem MS/private
21
22 mkdir ME ME/private ME/newcerts
23 touch ME/index.txt ME/serial
24 echo "00" >> ME/serial
25 cp me_key.pem ME/private
26 cp me_cert.pem ME
27
28 # Generazione Richiesta Certificato Mr.Joker's Casino e Nodo ADM
29 openssl req -new -key joker_key.pem -out joker_request.pem -config "C:\msys64\mingw64\etc\ssl\opensslME.cnf"
30 openssl ca -in joker_request.pem -out joker_cert.pem -policy policy_anything -config
  "C:\msys64\mingw64\etc\ssl\opensslME.cnf"
31
32 openssl req -new -key adm_key.pem -out adm_request.pem -config "C:\msys64\mingw64\etc\ssl\opensslME.cnf"
33 openssl ca -in adm_request.pem -out adm_cert.pem -policy policy_anything -config "C:\msys64\mingw64\etc\ssl\opensslME.cnf"
34
35 # Generazione del Keystore
36 openssl pkcs12 -export -out ms.p12 -inkey ms_key.pem -in ms_cert.pem -name ms -passout pass:aps2023
37 keytool -importkeystore -destkeystore ms_keystore.jks -srckeystore ms.p12 -srcstoretype PKCS12 -alias ms -srcstorepass
  aps2023 -deststorepass aps2023
38
39 openssl pkcs12 -export -out me.p12 -inkey me_key.pem -in me_cert.pem -name me -passout pass:aps2023
40 keytool -importkeystore -destkeystore me_keystore.jks -srckeystore me.p12 -srcstoretype PKCS12 -alias me -srcstorepass
  aps2023 -deststorepass aps2023
41
42 openssl pkcs12 -export -out joker.p12 -inkey joker_key.pem -in joker_cert.pem -name joker -passout pass:aps2023
43 keytool -importkeystore -destkeystore joker_keystore.jks -srckeystore joker.p12 -srcstoretype PKCS12 -alias joker
  -srcstorepass aps2023 -deststorepass aps2023
44
45 openssl pkcs12 -export -out adm.p12 -inkey adm_key.pem -in adm_cert.pem -name adm -passout pass:aps2023
46 keytool -importkeystore -destkeystore adm_keystore.jks -srckeystore adm.p12 -srcstoretype PKCS12 -alias adm -srcstorepass
  aps2023 -deststorepass aps2023
47
48 # Generazione del Truststore
49 openssl x509 -in ms_cert.pem -out ms_cert.der -outform der
50 openssl x509 -in me_cert.pem -out me_cert.der -outform der
51
52 keytool -genkeypair -keyalg RSA -alias dummy -keystore truststore.jks -storepass aps2023 -dname "CN=dummy, OU=dummy,
  O=dummy, L=dummy, S=dummy, C=dummy" -validity 1
53 keytool -delete -alias dummy -keystore truststore.jks -storepass aps2023
54
55 keytool -importcert -file ms_cert.pem -alias ms -keystore truststore.jks -storepass aps2023
56 keytool -importcert -file me_cert.pem -alias me -keystore truststore.jks -storepass aps2023

```

**Listing 4.2** – Script per la generazione dei parametri, delle chiavi private dei certificati, dei keystore e dei truststore, utilizzando il comando *keytool* di Java. L'uso dei **keystore** e dei **truststore** è essenziale per implementare in Java i meccanismi di connessione tra client e server attraverso socket sicure utilizzando come protocollo TLS.

## 4.2 Sistema di Gioco

Il sistema di gioco progettato prevede il coinvolgimento di un unico nodo *ADM*, un banco *D*, due giocatori già presenti nella sala, *alice* e *bob*, e un giocatore *P<sub>i</sub>* che simula l'intero sistema, al quale viene richiesto di effettuare l'accesso tramite il *GP 2.0*. Inoltre, per semplicità non abbiamo tenuto conto di alcun intervallo temporale, così come specificato nei WP precedenti.

Le classi utilizzate per l'implementazione complessiva del sistema, suddivise in tre package all'interno del package **JokerChain**, sono elencate di seguito:

1. Il package **Autentication** contiene le classi preposte all'autenticazione del giocatore *P<sub>i</sub>*. Nello specifico,
  - a. La classe *SSLBase* è una classe astratta che fornisce i principali metodi per la ricezione e l'invio dei dati attraverso una connessione socket sicura utilizzando come protocollo TLS e i metodi per la firma e la verifica dei dati.
  - b. Le classi *SSLCitizen*, *SSLJoker* e *SSLMS* che estendono *SSLBase*. Esse prevedono due metodi principali, uno per abilitare la connessione, in modo che possano fungere da server (e.g., l'IdP del *MS* che si connette al sito di *Mr Joker*), e un altro per avviare la connessione verso un server (e.g., il giocatore *P<sub>i</sub>* si connette all'IdP del *MS*).
2. Il package **JokerChain** contiene le classi preposte all'esecuzione della logica di gioco. Nello specifico,
  - a. Il package **Utils** include la classe *Generator* e la classe *SSLConnection*. In particolare, la classe *Generator* viene utilizzata per la generazione della coppia di chiavi, pubblica e privata, attraverso l'algoritmo RSA e per la generazione delle stringhe casuali. Invece, *SSLConnection* è una classe astratta che estende *SSLBase* ed implementa il metodo *connectTo*. Tale metodo è chiamato da tutti coloro che si connettono ad un nodo *ADM* e contiene dei metodi astratti definiti in maniera differenti nelle classi *SSLBanco* e *SSLPlayer*.

- b. La classe *Giocata* che identifica la giocata effettuata da un player  $P_i$ . In particolare, tale classe contiene il metodo *toBit*, che permette di rappresentare la giocata in termini di bit, così come descritto nel WP2, e il metodo *fromString*, che estrapola le informazioni dal formato in bit, quali numero, colore e parità. Inoltre, contiene la classe *Roulette* che permette di poter stabilire l'esito della giocata di ogni giocatore, come *Vincente* o *Perdente*.
- c. La classe *Player* che identifica i giocatori  $P$  ed il banco  $D$ .
- d. La classe *SSLADM* che estende *SSLBase*. Essa implementa la logica del nodo *ADM*, il quale viene trattato come un server, al quale si connettono i giocatori  $P$  ed il banco  $D$ . Essa prevede che non venga richiesto necessariamente al client un certificato durante la fase di handshake. Essa è responsabile dell'inserimento delle transazioni all'interno della *JokerChain* e, quindi, anche della creazione del blocco genesi. Nel caso in cui il valore ricevuto sia nullo, questo non verrà inserito nella *Jokerchain*. Inoltre, implementa dei meccanismi che permettono di sincronizzare le operazioni tra i player e il banco, evitando quindi la sovrapposizione di transazioni di tipi differenti.
- e. Le classi *SSLBanco* e *SSLPlayer* che estendono *SSLConnection*. Esse implementano i metodi astratti responsabili delle operazioni di popolamento della blockchain. Entrambe le classi implementano allo stesso modo il metodo *handleChiavi* per la generazione delle coppie di chiavi (Transazione  $Tr2$ ). Per il metodo *handleGiocate*, la classe *SSLBanco* invia al nodo *ADM* un valore nullo, in quanto il banco non può effettuare giocate, mentre la classe *SSLPlayer* chiede ai giocatori di effettuare una scelta, che può rilevarsi essere anche una giocata vuota '0000000000' che verrà inserita lo stesso nella blockchain, in modo da evitare di non avere transazioni  $Tr3$ . Il metodo *handleStringheCasuali* viene implementato allo stesso modo

in entrambi le classi, ad eccezione del fatto che *SSLPlayer* impedisce ad un giocatore  $P_i$  che ha effettuato una giocata vuota di poter generare una stringa casuale. Infine, per il metodo *handleRisultato*, la classe *SSLPlayer* invia al nodo *ADM* un valore nullo, in quanto è il banco *D* ad elaborare il risultato. Pertanto, la classe *SSLBanco* legge il valore della Merkle Root, calcolato sull'ultima tipologia di transazioni *Tr4* ed effettua poi un'operazione di modulo 37, così da produrre un valore compreso tra 0 e 36.

- f. La classe *JokerChain* che implementa la *JokerChain* come un file di testo a cui sono aggiunte ad ogni turno di gioco le varie transazioni. Inoltre, tale classe contiene la classe *Transaction* che implementa la logica delle transazioni, identificati dai parametri descritti ad inizio capitolo. Tali transazioni sono inserite nel file di testo *jokerchain.txt* in formato CSV in modo da facilitare l'estrapolazione delle informazioni in esso contenute all'avvio dell'applicazione.

3. Il package **Music** che contiene la classe *MusicPlayer* che permette la riproduzione in background di canzoni in loop<sup>10</sup>.
4. Infine, la classe *JokerCasino* che contiene il *main* ed implementa e definisce i thread e le istanze delle classi precedentemente definite, responsabili della fase di login e della fase di gioco.

Segue una panoramica semplificata delle classi definite, in cui sono stati eliminati i commenti e alcuni metodi non importanti, come i setter, i getter e i costruttori, in modo da offrire una visione d'insieme delle funzionalità del sistema progettato. Infine, viene mostrato un esempio di funzionamento, e quindi degli output prodotti, per tre differenti player.

---

<sup>10</sup> È stato previsto che la musica in background dell'applicazione fosse “La mia libertà” di Franco Califano, così come richiesto da specifiche.

```

1 public abstract class SSLBase {
2
3     private static final String TRUSTSTOREPATH = "Certs/truststore.jks";
4     private static final String PASSWORD = "aps2023";
5
6     public static void sendData(SSLocket socket, Object data) throws IOException {
7         ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
8         outputStream.writeObject(data);
9         outputStream.flush();
10    }
11
12    public static Object receiveData(SSLocket socket) throws IOException, ClassNotFoundException {
13        ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());
14        return inputStream.readObject();
15    }
16
17    private SSLContext sslContext;
18    private KeyStore ks;
19    private String keystorePath;
20
21    public byte[] signData(byte[] g, PrivateKey privateKey){
22        try{
23            Signature signature = Signature.getInstance("SHA256withRSA");
24            signature.initSign(privateKey);
25            signature.update(g);
26            return signature.sign();
27        }catch(Exception e){
28            e.printStackTrace();
29            return null;
30        }
31    }
32
33    public boolean verifySignature(byte[] data, byte[] signatureBytes, PublicKey publicKey) {
34        try {
35            Signature signature = Signature.getInstance("SHA256withRSA");
36            signature.initVerify(publicKey);
37            signature.update(data);
38            return signature.verify(signatureBytes);
39        } catch (Exception e) {
40            e.printStackTrace();
41            return false;
42        }
43    }
44
45    private void initializeSSLContext() throws Exception {
46        this.sslContext = SSLContext.getInstance("TLS");
47        // Carica il truststore
48        KeyStore ts = KeyStore.getInstance("JKS");
49        try (InputStream tis = new FileInputStream(TRUSTSTOREPATH)) {
50            ts.load(tis, PASSWORD.toCharArray());
51        }
52        // Inizializza il TrustManagerFactory
53        TrustManagerFactory tmf = TrustManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
54        tmf.init(ts);
55        if(!this.keystorePath.isEmpty()){
56            // Carica il keystore
57            this.ks = KeyStore.getInstance("JKS");
58            try (InputStream kis = new FileInputStream(this.keystorePath)) {
59                this.ks.load(kis, PASSWORD.toCharArray());
60            }
61            // Inizializza il KeyManagerFactory
62            KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
63            kmf.init(this.ks, PASSWORD.toCharArray());
64            // Inizializza il SSLContext
65            this.sslContext.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
66        }
67        else
68            this.sslContext.init(null, tmf.getTrustManagers(), null);
69    }
70 }

```

**Listing 4.3** – Classe SSLBase.java.

```

1 public class SSLCitizen extends SSLBase{
2
3     private static final int PORT = 3999;
4     private static final String MS_HOST = "localhost";
5     private static final int MS_PORT = 4000;
6     private SSLContext sslContext;
7     private boolean authorized;
8
9     public SSLCitizen(String keystorePath) throws Exception {
10         super(keystorePath);
11         this.sslContext = getSSLContext();
12     }
13
14     public void startConnection() {
15         try {
16             SSLServerSocketFactory ssf = this.sslContext.getServerSocketFactory();
17             SSLServerSocket serverSocket = (SSLServerSocket) ssf.createServerSocket(PORT);
18             serverSocket.setNeedClientAuth(true);
19
20             System.out.println("\u001B[32m(SSLPlayer) In attesa di connessioni ... \u001B[0m");
21             SSLSocket socket = (SSLSocket) serverSocket.accept();
22             System.out.println("\u001B[32m(SSLPlayer) Connesso con Mr.Joker!\u001B[0m");
23
24             String response = (String) receiveData(socket);
25             System.out.println("\u001B[32m(SSLPlayer) Risposta da Mr.Joker: \u001B[1m" + response + "\u001B[0m");
26
27             if(response.startsWith("Accesso Consentito"))
28                 setAuthorized(true);
29             else
30                 setAuthorized(false);
31
32             socket.close();
33             Thread.sleep(100);
34             System.out.println("\u001B[32m(SSLPlayer) Socket chiusa.\n\u001B[0m");
35
36         } catch (Exception ex) {
37             ex.printStackTrace();
38         }
39     }
40
41     public void connectToMS() {
42         try {
43             SSLSocketFactory ssf = this.sslContext.getSocketFactory();
44             SSLSocket socket = (SSLSocket) ssf.createSocket(MS_HOST, MS_PORT);
45             socket.startHandshake();
46
47             socket.close();
48             System.out.println("\u001B[32m(SSLPlayer) Socket chiusa.\u001B[0m");
49
50         } catch (Exception ex) {
51             ex.printStackTrace();
52         }
53     }
54 }

```

**Listing 4.4** – Classe SSLCitizen.java.

```

1 public class SSLMS extends SSLBase{
2
3     private static final int PORT = 4000;
4     private static final int JOKER_PORT = 4001;
5     private static final String JOKER_HOST = "localhost";
6     private SSLContext sslContext;
7
8     public SSLMS(String keystorePath) throws Exception {
9         super(keystorePath);
10        this.sslContext = getSSLContext();
11    }
12
13    public void startConnection() {
14        String message = "";
15        try {
16            SSLServerSocketFactory ssf = this.sslContext.getServerSocketFactory();
17            SSLServerSocket serverSocket = (SSLServerSocket) ssf.createServerSocket(PORT);
18            serverSocket.setNeedClientAuth(true);
19
20            System.out.println("\u001B[34m(SSLMS) In attesa di connessioni ... \u001B[0m");
21            SSLSocket socket = (SSLSocket) serverSocket.accept();
22            System.out.println("\u001B[34m(SSLMS) Connesso con il Player! \u001B[0m");
23
24            // Estrazione delle informazioni essenziali dal GP2.0 del giocatore.
25            SSLSession session = socket.getSession();
26            X509Certificate cert = (X509Certificate) session.getPeerCertificates()[0];
27            System.out.println("\u001B[34m(SSLMS) I seguenti dati stanno per essere inviati a Mr.Joker: Data di Nascita e
28            Codice Fiscale!\u001B[0m");
29            String subjectInfo = cert.getSubjectX500Principal().toString();
30            List<String> oidList = Arrays.asList("OID.1.2.3.1.4", "OID.1.2.3.1.7");
31            String[] fields = subjectInfo.split(",");
32            for(String field : fields){
33                String[] fieldSplit = field.split("=");
34                if(oidList.contains(fieldSplit[0].trim()))
35                    message += fieldSplit[1] + ";";
36            }
37            // Chiusura socket
38            socket.close();
39            Thread.sleep(100);
40            System.out.println("\u001B[34m(SSLMS) Socket chiusa. \u001B[0m");
41        } catch (Exception ex) {
42            ex.printStackTrace();
43        }
44        this.connectToJoker(message);
45    }
46
47    private void connectToJoker(String message) {
48        try {
49            SSLSocketFactory ssf = this.sslContext.getSocketFactory();
50            SSLSocket socket = (SSLSocket) ssf.createSocket(JOKER_HOST, JOKER_PORT);
51            socket.startHandshake();
52            // Invio dati essenziali per l'autenticazione.
53            sendData(socket, message);
54            socket.close();
55        } catch (Exception ex) {
56            ex.printStackTrace();
57        }
58    }
59 }

```

**Listing 4.5** – Classe SSLMS.java.

```

1  public class SSLJoker extends SSLBase {
2
3      private static final int PORT = 4001;
4      private static final int PLAYER_PORT = 3999;
5      private static final String PLAYER_HOST = "localhost";
6      private SSLContext sslContext;
7      private Player p;
8
9      public SSLJoker(String keystorePath, Player p) throws Exception {
10         super(keystorePath);
11         this.sslContext = getSSLContext();
12         this.p = p;
13     }
14
15     public void startConnection() {
16         String message = "";
17         try {
18             // Inizializzazione socket.
19             SSLServerSocketFactory ssf = this.sslContext.getSSLServerSocketFactory();
20             SSLServerSocket serverSocket = (SSLServerSocket) ssf.createSSLServerSocket(PORT);
21             serverSocket.setNeedClientAuth(true);
22             System.out.println("\u001B[35m(SSLJoker) In attesa di connessioni ... \u001B[0m");
23             SSLSocket socket = (SSLSocket) serverSocket.accept();
24             System.out.println("\u001B[35m(SSLJoker) Connesso con l'Identity Provier del Ministero della Salute
... \u001B[0m");
25             // Ricezione dati dall'IdP del MS.
26             message = (String) receiveData(socket);
27             // Chiusura Socket
28             socket.close();
29             Thread.sleep(500);
30             System.out.println("\u001B[35m(SSLJoker) Socket chiuso.\n--- \u001B[0m");
31         } catch (Exception ex) {
32             ex.printStackTrace();
33         }
34
35         sendToPlayer(message);
36     }
37
38     private void sendToPlayer(String message) {
39         String response = "";
40         try {
41             SSLSocketFactory ssf = this.sslContext.getSSLServerSocketFactory();
42             SSLSocket socket = (SSLSocket) ssf.createSSLServerSocket(PLAYER_HOST, PLAYER_PORT);
43             socket.startHandshake();
44
45             // Estrapolazione dati
46             String codiceFiscale = message.split(";")[0];
47             DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
48             LocalDate dataDiNascita = LocalDate.parse(message.split(";")[1], formatter);
49
50             if(isMaggiorenne(dataDiNascita)){
51                 response = "Accesso Consentito. Benvenuto " + codiceFiscale;
52                 this.p.setCodiceFiscale(codiceFiscale);
53                 this.p.setDataDiNascita(dataDiNascita);
54             }
55             else
56                 response = "Accesso Negato. Il gioco è vietato per i minori di 18 anni!";
57
58             sendData(socket, response);
59             socket.close();
60
61         } catch (Exception e) {
62             e.printStackTrace();
63             System.exit(0);
64         }
65     }
66 }
67
68     private static boolean isMaggiorenne(LocalDate dataDiNascita) {
69         LocalDate oggi = LocalDate.now();
70         Period periodo = Period.between(dataDiNascita, oggi);
71         return periodo.getYears() >= 18;
72     }
73 }
74 }
```

Listing 4.6 – Classe SSLJoker.java.

```

1 public class Generator {
2
3     public static KeyPair generateKeyPair() {
4         try {
5             SecureRandom secureRandom = new SecureRandom();
6             KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
7             keyGen.initialize(2048, secureRandom); // Utilizzo di SecureRandom
8             KeyPair keyPair = keyGen.generateKeyPair();
9             return keyPair;
10        } catch (Exception e) {
11            e.printStackTrace();
12            return null;
13        }
14    }
15
16    public static byte[] generatePRG() {
17        SecureRandom secureRandom = new SecureRandom();
18        byte[] randomBytes = new byte[32];
19        secureRandom.nextBytes(randomBytes);
20        return randomBytes;
21    }
22
23 }
```

Generator.java

**Listing 4.7** – Classe Generator.java.

```

1 public abstract class SSLConnection extends SSLBase{
2
3     private static final int ADM_PORT = 4002;
4     private static final String ADM_HOST = "localhost";
5
6     public SSLConnection() throws Exception{
7         super();
8     }
9
10    public SSLConnection(String keystorePath) throws Exception {
11        super(keystorePath);
12    }
13
14    public void connectTo(SSLADM ssladm, String nickname) {
15        try {
16            SSLSocketFactory ssf = getSSLContext().getSocketFactory();
17            SSLSocket socket = (SSLSocket) ssf.createSocket(ADM_HOST, ADM_PORT);
18            socket.startHandshake();
19
20            // Connessione
21            sendData(socket, nickname);
22
23            // Blocco Chiavi
24            handleChiavi(socket, ssladm);
25
26            // Blocco Giocate
27            handleGiocate(socket, ssladm);
28
29            // Blocco Stringhe Casuali
30            handleStringheCasuali(socket, ssladm);
31
32            // Blocco Risultato
33            handleRisultato(socket, ssladm);
34
35            socket.close();
36
37        } catch (Exception ex) {
38            ex.printStackTrace();
39        }
40    }
41
42    protected abstract void handleChiavi(SSLocket socket, SSLADM ssladm) throws Exception;
43    protected abstract void handleGiocate(SSLocket socket, SSLADM ssladm) throws Exception;
44    protected abstract void handleStringheCasuali(SSLocket socket, SSLADM ssladm) throws Exception;
45    protected abstract void handleRisultato(SSLocket socket, SSLADM ssladm) throws Exception;
46 }
```

SSLConnection.java

**Listing 4.8** – Classe SSLConnection.java.

```

1  public class Giocata {
2
3      private String numero;
4      private String colore;
5      private String pariDispari;
6      private String bit;
7
8      public Giocata(String numero, String colore, String pariDispari) {
9          this.numero = numero;
10         this.colore = colore;
11         this.pariDispari = pariDispari;
12         this.bit = this.toBit();
13     }
14
15     public String toBit(){
16         String sogliaNumero = (this.numero.isEmpty() == false)? "1" : "0";
17         String numero = (sogliaNumero == "1") ? Integer.toBinaryString(Integer.valueOf(this.numero)) : "000000";
18         String sogliaColore = (this.colore.isEmpty() == false)? "1" : "0";
19         String colore = (sogliaColore == "1") ? (this.colore.equals("rosso")? "1" : (this.colore.equals("nero")? "0" :
20             null)) : "0";
21         String sogliaPariDispari = (this.pariDispari.isEmpty() == false)? "1" : "0";
22         String pariDispari = (sogliaPariDispari == "1") ? ((this.pariDispari.equals("pari"))? "1" :
23             (this.pariDispari.equals("dispari")? "0" : null)) : "0";
24
25         while(numero.length() != 6)
26             numero = "0" + numero;
27
28         return sogliaNumero + numero + sogliaColore + colore + sogliaPariDispari + pariDispari;
29     }
30
31     public static String fromString(String g){
32         String numero, colore, pariDispari;
33         if(String.valueOf(g.charAt(0)).equals("0"))
34             numero = null;
35         else
36             numero = g.substring(1, 7);
37         if(String.valueOf(g.charAt(7)).equals("0"))
38             colore = null;
39         else
40             colore = String.valueOf(g.charAt(8));
41         if(String.valueOf(g.charAt(9)).equals("0"))
42             pariDispari = null;
43         else
44             pariDispari = String.valueOf(g.charAt(10));
45
46         return numero + "," + colore + "," + pariDispari;
47     }
48
49     class Roulette{
50
51         static Hashtable<String, String> hashtable = new Hashtable<String, String>();
52
53         public String esito(String risultato, String g){
54             // Giocata
55             String[] gString = Giocata.fromString(g).split(",");
56             // Risultato
57             String colore = hashtable.get(risultato);
58             int value = Integer.parseInt(risultato) % 2;
59             String pariDispari = (value == 0)? "pari" : "dispari";
60             String rGiocata = new Giocata(risultato, colore, pariDispari).toBit();
61             String[] rString = Giocata.fromString(rGiocata).split(",");
62
63             if(gString[0].equals(rString[0]) || gString[1].equals(rString[1]) || gString[2].equals(rString[2]))
64                 return "Vincente!";
65             else
66                 return "Perdente!";
67
68         }
69     }
}

```

Listing 4.9 – Classe Giocata.java.

```

1  public class SSLADM extends SSLBase{
2
3      private static final int PORT = 4002;
4      private SSLContext sslContext;
5      private JokerChain jokerChain;
6      private KeyStore ks;
7      private CountDownLatch countDownLatch;
8      private static final Semaphore inputSemaphore = new Semaphore(1);
9
10     public SSLADM(String keystorePath, JokerChain jokerChain) throws Exception {
11         super(keystorePath);
12         this.sslContext = getSSLContext();
13         this.jokerChain = jokerChain;
14         this.ks = getKS();
15         this.countDownLatch = new CountDownLatch(4);
16     }
17
18     public void startConnection() {
19         try {
20             SSLServerSocketFactory ssf = this.sslContext.getServerSocketFactory();
21             SSLServerSocket serverSocket = (SSLServerSocket) ssf.createServerSocket(PORT);
22             serverSocket.setWantClientAuth(true);
23
24             System.out.println("\u001B[33m(SSLADM) In attesa di connessioni ... \u001B[0m");
25             int count = 0;
26             while(count < 4){
27                 final SSLSocket socket = (SSLSocket) serverSocket.accept();
28                 new Thread(() -> handleClientConnection(socket)).start();
29                 count++;
30             }
31             serverSocket.close();
32         } catch (Exception ex) {
33             ex.printStackTrace();
34         }
35     }
36
37     public void handleClientConnection(SSLSocket socket) {
38         try{
39             String client = (String) SSLConnection.receiveData(socket);
40             System.out.println("\u001B[33m(SSLADM) Connesso con " + client + "\u001B[0m");
41
42             SSLSession session = socket.getSession();
43             if(this.jokerChain.getTransactions().isEmpty()){
44                 X509Certificate admCert = (X509Certificate) this.ks.getCertificate("adm");
45                 PublicKey admPublicKey = admCert.getPublicKey();
46
47                 X509Certificate jokerCert = (X509Certificate) session.getPeerCertificates()[0];
48                 PublicKey jokerPublicKey = jokerCert.getPublicKey();
49
50                 System.out.println("\u001B[33m(SSLADM) Creazione Blocco Genesi ... \u001B[0m");
51                 this.jokerChain.createGenesisBlock(admPublicKey, jokerPublicKey);
52             }
53
54             // Blocco Chiavi
55             Thread.sleep(500);
56             PublicKey publicKey = (PublicKey) SSLConnection.receiveData(socket);
57             System.out.println("\u001B[33m(SSLADM) Ricevuta chiave pubblica da: " + client + "\u001B[0m");
58             this.jokerChain.addTransaction(new Transaction(2, Base64.getEncoder().encodeToString(publicKey.getEncoded()), client));
59             this.countDownLatch.countDown();
60             if(this.countDownLatch.getCount() == 0)
61                 this.resetLatch(4);
62
63             // Blocco Giocate
64             Thread.sleep(500);
65             String g = (String) SSLConnection.receiveData(socket);
66             if(g != null){
67                 byte[] signedG = (byte[]) SSLConnection.receiveData(socket);
68                 System.out.println("\u001B[33m(SSLADM) Ricevuta giocata da: " + client + "\u001B[0m");
69                 if (!verifySignature(g.getBytes(), signedG, publicKey))
70                     System.out.println("\u001B[33m(SSLADM) Firma non valida. Giocata potenzialmente compromessa!\u001B[0m");
71                 else{
72                     System.out.println("\u001B[33m(SSLADM) Firma valida. Inserimento della giocata nella JokerChain in corso
73                         ... \u001B[0m");
74                     this.jokerChain.addTransaction(new Transaction(3, g, client));
75                 }
76             }
77             this.countDownLatch.countDown();
78             if(this.countDownLatch.getCount() == 0)
79                 this.resetLatch(4);
}

```

```

79      // Blocco Stringhe Casuali
80      Thread.sleep(500);
81      inputSemaphore.acquire();
82      byte[] prng = (byte[]) SSLConnection.receiveData(socket);
83      if(prng != null){
84          byte[] signedPrng = (byte[]) SSLConnection.receiveData(socket);
85          System.out.println("\u001B[33m(SSLADM) Ricevuta stringa casuale da: " + client + "\u001B[0m");
86          if (!verifySignature(prng, signedPrng, publicKey))
87              System.out.println("\u001B[33m(SSLADM) Firma non valida. Stringa potenzialmente compromessa!\u001B[0m");
88          else{
89              System.out.println("\u001B[33m(SSLADM) Firma valida. Inserimento della stringa nella JokerChain in corso
... \u001B[0m");
90              this.jokerChain.addTransaction(new Transaction(4, Base64.getEncoder().encodeToString(prng), client));
91          }
92      }
93      inputSemaphore.release();
94      this.countDownLatch.countDown();
95      if(this.countDownLatch.getCount() == 0)
96          this.resetLatch(4);
97
98      // Blocco Risultato
99      Thread.sleep(500);
100     inputSemaphore.acquire();
101     String risultato = (String) SSLConnection.receiveData(socket);
102     if(risultato != null){
103         byte[] signedRisultato = (byte[]) SSLConnection.receiveData(socket);
104         System.out.println("\u001B[33m(SSLADM) Ricevuto risultato da: " + client + "\u001B[0m");
105         if (!verifySignature(risultato.getBytes(), signedRisultato, publicKey))
106             System.out.println("\u001B[33m(SSLADM) Firma non valida. Risultato potenzialmente
compromessa!\u001B[0m");
107         else{
108             System.out.println("\u001B[33m(SSLADM) Firma valida. Inserimento del risultato nella JokerChain in corso
... \u001B[0m");
109             this.jokerChain.addTransaction(new Transaction(5, risultato, client));
110             System.out.println(" --- \n\u001B[33m(SSLADM) Il numero estratto è: \u001B[1m" + risultato +
"\u001B[0m\n---");
111         }
112     }
113     inputSemaphore.release();
114     if(risultato == null && g != null && !g.equals("000000000000")){
115         Roulette roulette = new Roulette();
116         String numero = jokerChain.filterTransaction(5, jokerChain.getCurrentID() - 1).get(0).getData();
117         System.out.println("\u001B[33m(SSLADM) La giocata di " + client + " è \u001B[1m" + roulette.esito(numero, g) +
"\u001B[0m");
118     }
119
120     } catch(Exception e){
121         e.printStackTrace();
122     }
123 }

```

Listing 4.10 – Classe SSLADM.java.

```
1 public class SSLBanco extends SSLConnection{
2
3     private Player banco;
4     private JokerChain jokerChain;
5
6     public SSLBanco(String keystorePath, Player banco, JokerChain jokerChain) throws Exception {
7         super(keystorePath);
8         this.banco = banco;
9         this.jokerChain = jokerChain;
10    }
11
12    @Override
13    protected void handleChiavi(SSLocket socket, SSLADM ssladm) throws Exception {
14        KeyPair keyPair = Generator.generateKeyPair();
15        this.banco.setKeyPair(keyPair);
16        SSLConnection.sendData(socket, keyPair.getPublic());
17        ssladm.awaitCompletion();
18    }
19
20    @Override
21    protected void handleGiocate(SSLocket socket, SSLADM ssladm) throws Exception {
22        SSLConnection.sendData(socket, null);
23        ssladm.awaitCompletion();
24    }
25
26    @Override
27    protected void handleStringheCasuali(SSLocket socket, SSLADM ssladm) throws Exception {
28        byte[] prng = Generator.generatePRG();
29        byte[] signedPrng = signData(prng, this.banco.getKeyPair().getPrivate());
30        SSLConnection.sendData(socket, prng);
31        SSLConnection.sendData(socket, signedPrng);
32        ssladm.awaitCompletion();
33    }
34
35    @Override
36    protected void handleRisultato(SSLocket socket, SSLADM ssladm) throws Exception {
37        String merkleRoot = this.jokerChain.calculateMerkleRootForType(4);
38        String risultato = String.valueOf(new BigInteger(merkleRoot, 16).mod(new BigInteger("37")).intValue());
39        byte[] signedRisultato = signData(risultato.getBytes(), this.banco.getKeyPair().getPrivate());
40        SSLConnection.sendData(socket, risultato);
41        SSLConnection.sendData(socket, signedRisultato);
42    }
43
44 }
```

SSLBanco.java

**Listing 4.11** – Classe SSLBanco.java.

```

1  public class SSLPlayer extends SSLConnection{
2
3      // private static final int PORT = 4001;
4      private Player p;
5      private static final Semaphore inputSemaphore = new Semaphore(1);
6
7      public SSLPlayer(Player p) throws Exception {
8          super();
9          this.p = p;
10     }
11
12     @Override
13     protected void handleChiavi(SSLocket socket, SSLADM ssladm) throws Exception {
14         KeyPair keyPair = Generator.generateKeyPair();
15         this.p.setKeyPair(keyPair);
16         SSLConnection.sendData(socket, keyPair.getPublic());
17         ssladm.awaitCompletion();
18     }
19
20     @Override
21     protected void handleGiocate(SSLocket socket, SSLADM ssladm) throws Exception {
22         try(Scanner scanner = new Scanner(System.in)){
23             inputSemaphore.acquire();
24             Thread.sleep(500);
25             boolean invalid = false;
26             // Numero
27             System.out.print("(" + this.p.getNickname() + ") Scegli un numero (0-36 | \": \"");
28             String numero = scanner.nextLine();
29             if(numero != "" && !(Integer.parseInt(numero) >= 0 && Integer.parseInt(numero) <= 36))
30                 invalid = true;
31             // Colore
32             System.out.print("(" + this.p.getNickname() + ") Scegli un colore (nero | rosso | \": \"");
33             String colore = scanner.nextLine();
34             if(!colore.contains("nero") && !colore.contains("rosso") && colore != "")
35                 invalid = true;
36             // Pari o Dispari
37             System.out.print("(" + this.p.getNickname() + ") Scegli pari o dispari (pari | dispari | \": \"");
38             String parita = scanner.nextLine();
39             if(!parita.contains("pari") && !parita.contains("dispari") && parita != "")
40                 invalid = true;
41             if(numero == "" && colore == "" && parita == "")
42                 invalid = true;
43
44             if (!invalid)
45                 this.p.setGiocata(new Giocata(numero, colore, parita));
46             else
47                 this.p.setGiocata(new Giocata("", "", ""));
48
49             byte[] signedG = signData(this.p.getGiocata().getBit().getBytes(), this.p.getKeyPair().getPrivate());
50             SSLConnection.sendData(socket, this.p.getGiocata().getBit());
51             SSLConnection.sendData(socket, signedG);
52             inputSemaphore.release();
53             ssladm.awaitCompletion();
54         }catch(Exception e){
55             e.printStackTrace();
56         }
57     }
58
59     @Override
60     protected void handleStringheCasuali(SSLocket socket, SSLADM ssladm) throws Exception {
61         if(!this.p.getGiocata().getBit().equals("000000000000")){
62             byte[] prng = Generator.generatePRNG();
63             byte[] signedPrng = signData(prng, this.p.getKeyPair().getPrivate());
64             SSLConnection.sendData(socket, prng);
65             SSLConnection.sendData(socket, signedPrng);
66         }
67         else
68             SSLConnection.sendData(socket, null);
69             ssladm.awaitCompletion();
70     }
71
72     @Override
73     protected void handleRisultato(SSLocket socket, SSLADM ssladm) throws Exception {
74         SSLConnection.sendData(socket, null);
75     }
76 }
77 }
```

Listing 4.12 – Classe SSLPlayer.java.

```

1 public class JokerChain {
2
3     private static File blockchainFile = new File("jokerchain.txt");
4     private List<Transaction> transactions = new ArrayList<>();
5     private int currentID;
6
7     public JokerChain(boolean reset) throws Exception {
8         if (blockchainFile.exists() && blockchainFile.length() != 0 && !reset) {
9             try (BufferedReader reader = new BufferedReader(new FileReader(blockchainFile))) {
10                 String line;
11                 while ((line = reader.readLine()) != null) {
12                     String[] parts = line.split(",");
13                     int id = Integer.parseInt(parts[0]);
14                     int type = Integer.parseInt(parts[1]);
15                     String data = parts[2];
16                     String hash = parts[3];
17                     String sender = parts[4];
18                     transactions.add(new Transaction(id, type, data, hash, sender));
19                 }
20                 currentID = transactions.get(transactions.size() - 1).getId() + 1;
21                 if(!this.checkGenesis())
22                     throw new Exception("Errore: è presente più di un blocco genesi!");
23             }
24         } else {
25             currentID = 0;
26             if (reset)
27                 blockchainFile.delete();
28             if (!blockchainFile.exists())
29                 blockchainFile.createNewFile();
30         }
31     }
32
33     public void addTransaction(Transaction transaction) throws IOException {
34         transaction.setId(currentID);
35         transactions.add(transaction);
36         if (transaction.getType() == 5)
37             currentID++;
38         try (BufferedWriter writer = new BufferedWriter(new FileWriter(blockchainFile, true))) {
39             writer.write(transaction.toCSV());
40             writer.newLine();
41         }
42     }
43
44     public String calculateMerkleRootForType(int type) throws Exception {
45         List<Transaction> transactionsOfType = filterTransaction(type, this.currentID);
46         List<String> hashes = transactionsOfType.stream()
47             .map(Transaction::getHash)
48             .collect(Collectors.toList());
49
50         return computeMerkleRoot(hashes);
51     }
52
53     public void createGenesisBlock(PublicKey admPublicKey, PublicKey jokerPublicKey) throws IOException {
54         String admPK = Base64.getEncoder().encodeToString(admPublicKey.getEncoded());
55         String jokerPK = Base64.getEncoder().encodeToString(jokerPublicKey.getEncoded());
56         this.addTransaction(new Transaction(1, admPK, "ADM"));
57         this.addTransaction(new Transaction(1, jokerPK, "Joker"));
58         this.currentID++;
59     }
60
61     public List<Transaction> filterTransaction(int type, int id){
62         List<Transaction> transactionsOfType = new ArrayList<Transaction>();
63         transactionsOfType = this.transactions.stream()
64             .filter(t -> t.getType() == type && t.getId() == id)
65             .collect(Collectors.toList());
66         return transactionsOfType;
67     }
68
69     private boolean checkGenesis() {
70         int[] totalID = IntStream.range(0, this.currentID).toArray();
71         List<Transaction> genesisTransactions = new ArrayList<Transaction>();
72         for(int id: totalID){
73             genesisTransactions = filterTransaction(1, id);
74             if(id != 0 && !genesisTransactions.isEmpty())
75                 return false;
76         }
77         return true;
78     }
79
80 }
```

JokerChain.java

```

81     private String computeMerkleRoot(List<String> hashes) throws Exception {
82         if (hashes.size() == 1)
83             return hashes.get(0);
84
85         List<String> newHashes = new ArrayList<>();
86         for (int i = 0; i < hashes.size(); i += 2) {
87             if (i + 1 < hashes.size())
88                 newHashes.add(Transaction.sha256(hashes.get(i) + hashes.get(i + 1)));
89             else
90                 newHashes.add(Transaction.sha256(hashes.get(i)));
91         }
92         return computeMerkleRoot(newHashes);
93     }
94
95 }
96
97 class Transaction {
98
99     public static String sha256(String input) throws Exception {
100         MessageDigest digest = MessageDigest.getInstance("SHA-256");
101         byte[] hash = digest.digest(input.getBytes("UTF-8"));
102         StringBuilder hexString = new StringBuilder();
103         for (byte b : hash) {
104             String hex = Integer.toHexString(0xff & b);
105             if (hex.length() == 1) hexString.append('0');
106             hexString.append(hex);
107         }
108         return hexString.toString();
109     }
110
111     private int id;
112     private int type;
113     private String data;
114     private String hash;
115     private String sender;
116
117     public Transaction(int id, int type, String data, String hash, String sender) {
118         this.id = id;
119         this.type = type;
120         this.data = data;
121         this.hash = hash;
122         this.sender = sender;
123     }
124
125     public Transaction(int id, int type, String data, String sender) {
126         this.id = id;
127         this.type = type;
128         this.data = data;
129         try {
130             this.hash = sha256(data);
131         } catch (Exception e) {
132             e.printStackTrace();
133         }
134         this.sender = sender;
135     }
136
137     public Transaction(int type, String data, String sender) {
138         this.type = type;
139         this.data = data;
140         try {
141             this.hash = sha256(data);
142         } catch (Exception e) {
143             e.printStackTrace();
144         }
145         this.sender = sender;
146     }
147
148     public String toCSV() {
149         return id + ";" + type + ";" + data + ";" + hash + ";" + sender;
150     }
151
152 }

```

**Listing 4.13** – Classe JokerChain.java.

```

1 public class JokerCasino {
2
3     public static boolean Login(Player p, Scanner scanner) throws Exception{
4         // Lettura Path GP2.0 del Player
5         // ...
6
7         // Procedura di Autenticazione
8         SSLMS sslMS = new SSLMS("Certs/ms_keystore.jks");
9         new Thread(() -> {
10             sslMS.startConnection();
11         }).start();
12
13         SSLJoker sslJoker = new SSLJoker("Certs/joker_keystore.jks", p);
14         new Thread(() -> {
15             sslJoker.startConnection();
16         }).start();
17
18         SSLCitizen sslPlayer = new SSLCitizen(player_cert);
19         Thread.sleep(1000);
20         sslPlayer.connectToMS();                                // Port 4000 + IdP MS
21         new Thread(() -> {
22             sslPlayer.startConnection();
23         }).start();
24
25         Thread.sleep(1000);
26         return sslPlayer.isAuthorized();
27     }
28
29     public static void main(String[] args) throws Exception {
30         System.out.println("\u001B[41m\u001B[1m Mr. Joker's Casino \u001B[0m");
31         Musicplayer.play();
32         Thread.sleep(1000);
33         Scanner scanner = new Scanner(System.in);
34
35         /* LOGIN */
36         Player p = new Player();
37         boolean authorized = Login(p, scanner);
38         if(!authorized)
39             System.exit(0);
40
41         /* GIOCO */
42         // Scelta Nickname ...
43
44         // JokerChain
45         JokerChain jokerChain = new JokerChain(false);
46
47         // Generazione Istanze del nodo ADM, del banco e dei giocatori.
48         SSLADM sslAdm = new SSLADM("Certs/adm_keystore.jks", jokerChain);
49         SSLBanco sslBanco = new SSLBanco("Certs/joker_keystore.jks", new Player("Banco"), jokerChain);
50         SSLPlayer sslPlayer = new SSLPlayer(p);
51         SSLPlayer sslPlayerAlice = new SSLPlayer(new Player("Alice"));
52         SSLPlayer sslPlayerBob = new SSLPlayer(new Player("Bob"));
53
54         new Thread(() -> {
55             sslAdm.startConnection();
56         }).start();
57
58         new Thread(() -> {
59             sslBanco.connectTo(sslAdm, new Player("Banco").getNickname());
60         }).start();
61
62         Thread.sleep(100);
63         new Thread(() -> {
64             sslPlayer.connectTo(sslAdm, p.getNickname());
65         }).start();
66
67         Thread.sleep(100);
68         new Thread(() -> {
69             sslPlayerAlice.connectTo(sslAdm, "Alice");
70         }).start();
71
72         Thread.sleep(100);
73         new Thread(() -> {
74             sslPlayerBob.connectTo(sslAdm, "Bob");
75         }).start();
76
77     }
78 }
79 }
```

JokerCasino.java

**Listing 4.14** – Classe JokerCasino.java.

## 4.2.1 Player: andy

Consideriamo il caso di un giocatore *andy* in possesso di un *GP* 2.0 dalla durata di 180 giorni (durata di default per i certificati rilasciati dal *MS*). Essendo il player *andy* maggiorenne, è ammesso alla sala bingo di *Mr Joker*.

```

1 # Generazione Chiavi Private
2 openssl genkey -paramfile prime256v1.pem -out andy_key.pem
3
4 # Generazione Richiesta GP2.0
5 openssl req -new -key andy_key.pem -out andy_request.pem -config "C:\msys64\mingw64\etc\ssl\openssl.cnf"
6 openssl ca -in andy_request.pem -out andy_cert.pem -policy policy_anything -config "C:\msys64\mingw64\etc\ssl\openssl.cnf"
7
8 # Generazione del Keystore
9 openssl pkcs12 -export -out andy.p12 -inkey andy_key.pem -in andy_cert.pem -name andy -passout pass:aps2023
10 keytool -importkeystore -destkeystore andy_keystore.jks -srckeystore andy.p12 -srcstoretype PKCS12 -alias andy -srcstorepass
aps2023 -deststorepass aps2023
11
12 # Stampa parziale del keystore
13 ...
14 Alias name: andy
15 Creation date: 19 ago 2023
16 Entry type: PrivateKeyEntry
17 Certificate chain length: 1
18 Certificate[1]:
19 Owner: OID.1.2.3.3=Ministero della Salute, OID.1.2.3.2.2=BioNTech-Pfizer, OID.1.2.3.2.1=NULL,
OID.1.2.3.1.7=RCCNRV01C20H703U, OID.1.2.3.1.6=SA, OID.1.2.3.1.5=Salerno, OID.1.2.3.1.4=20-03-2001, OID.1.2.3.1.3=M,
OID.1.2.3.1.2=Ricciardi, OID.1.2.3.1.1=Andrea Vincenzo, CN=localhost, ST=Italia, C=IT
20 Issuer: CN=localhost, OU=MS, O=Ministero della Salute, L=Roma, ST=Italia, C=IT
21 Serial number: 0
22 Valid from: Sat Aug 19 18:04:33 CEST 2023 until: Thu Feb 15 17:04:33 CET 2024
23 Certificate fingerprints:
24     SHA1: 89:7D:88:EA:EB:A3:C5:58:EA:B1:CB:82:2B:79:DE:8C:FE:20:75:BB
25     SHA256: A9:6D:01:89:42:1E:BB:01:42:A8:6D:7D:98:95:97:6C:DC:86:AD:F6:A2:ED:FD:F3:E7:A5:6B:9C:93:E3:74:77
26 Signature algorithm name: SHA256withECDSA
27 Subject Public Key Algorithm: 256-bit EC (secp256r1) key
28 Version: 3
29 ...

```

**Listing 4.15** – Script per la generazione del GP 2.0 e del keystore per il player *andy*.

```

Andy@Andy-VirtualBox:~/Desktop/Università degli Studi di Salerno/Magistrale - AI and IR/1/Anno/Secondo Semestre 2022-23/APL/Project - JokerChain
$ java -enable-preview -jar project.jar
Franco Calafato - Le mie libertà (1984)

Per poter accedere, inserisci il path del suo GP 2.0: Certs/andy_keystore.jks
(SSLAdm) In attesa di connessioni ...
(SSLAdm) Connesso con il Player!
(SSLPlayer) Socket chiuso
(SSLAdm) In attesa di connessioni ...
(SSLAdm) I seguenti dati stanno per essere inviati a Mr.Joker: Data di Nascita e Codice Fiscale
(SSLJoker) Connesso con l'Identity Provier del Ministero della Salute ...
(SSLJoker) Socket chiuso.

(SSLPlayer) Connesso con Mr.Joker!
(SSLAdm) Benvenuto Mr.Joker! Accesso Consentito, Benvenuto RCCNRV01C20H703U
(SSLPlayer) Socket chiuso.

Inserisci un nickname per la sala da gioco:
Benvenuto RCCNRV01C20H703U

(SSLAdm) In attesa di connessioni ...
(SSLAdm) Connesso con Bob
(SSLAdm) Benvenuto RCCNRV01C20H703U
(SSLAdm) Connesso con Alice!
(SSLAdm) Benvenuto RCCNRV01C20H703U
(SSLAdm) Ricevuta chiave pubblica dai Banco
(SSLAdm) Ricevuta chiave pubblica dai Bob
(SSLAdm) Ricevuta chiave pubblica dai Alice
(SSLAdm) Ricevuta chiave pubblica dai Bob
(SSLAdm) Inserisci una stringa (nero | rosso | bianco | dispari | ""): 14
(RCCNRV01C20H703U) Scegli un colore (nero | rosso | bianco | dispari | ""): bianco
(SSLAdm) Ricevuta giocata del RCCNRV01C20H703U
(SSLAdm) Firma valida. Inserimento della giocata nella JokerChain in corso ...
(Alice) Scegli un colore (nero | rosso | bianco | dispari | ""): bianco
(Alice) Scegli pure un dispero (nero | rosso | bianco | dispari | ""): bianco
(SSLAdm) Ricevuta giocata di Alice
(SSLAdm) Inserimento della giocata nella JokerChain in corso ...
(SSLAdm) Ricevuta stringa casuale da: Bob
(SSLAdm) Inserimento della stringa nella JokerChain in corso ...
(SSLAdm) Ricevuto risultato dai Banco
(SSLAdm) Firma valida. Inserimento del risultato nella JokerChain in corso ...
(SSLAdm) Il numero estratto è 14
(RCCNRV01C20H703U è Vincente)
(SSLAdm) La giocata di Alice è Perdente
(SSLAdm) La giocata di Bob è Perdente

```

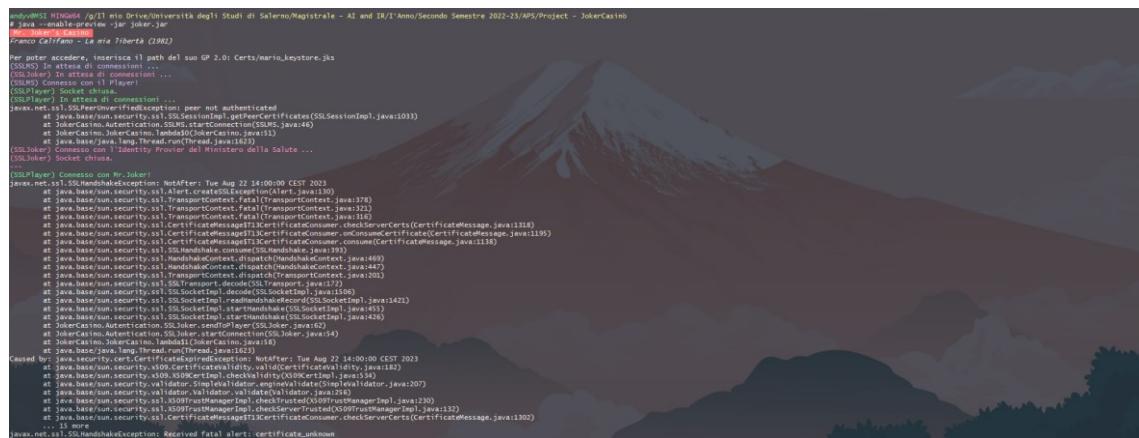
**Figure 4.1** – Esempio di output di gioco. Non avendo specificato un nickname, il giocatore viene identificato nella JokerChain con il suo codice fiscale. In questo esempio, il giocatore è risultato vincitore avendo indovinato il numero estratto dal sistema.

## 4.2.2 Player: mario

Consideriamo il caso di un giocatore *mario* in possesso di un *GP* 2.0 dalla durata di due giorni, rilasciato in data 20 agosto 2023. Essendo la data odierna (25 agosto 2023) oltre alla data di scadenza, 22 agosto 2023, il player *mario* non viene ammesso alla sala bingo di *Mr Joker*.

```
1 # Generazione Chiavi Private
2 openssl genkey -paramfile prime256v1.pem -out mario_key.pem
3
4 # Generazione Richiesta GP2.0
5 openssl req -new -key mario_key.pem -out mario_request.pem -config "C:\msys64\mingw64\etc\ssl\opensslslGP.cnf"
6 openssl ca -in mario_request.pem -out mario_cert.pem -policy policy_anything -config
"\"C:\msys64\mingw64\etc\ssl\opensslslGP.cnf\""
7
8 # Generazione dei Keystore
9 openssl pkcs12 -export -out mario.p12 -inkey mario_key.pem -in mario_cert.pem -name mario -passout pass:aps2023
10 keytool -importkeystore -destkeystore mario_keystore.jks -srckeystore mario.p12 -srcstoretype PKCS12 -alias mario
-srcstorepass aps2023 -deststorepass aps2023
11
12 # Stampa parziale del keystore
13 ...
14 Alias name: mario
15 Creation date: 23 ago 2023
16 Entry type: PrivateKeyEntry
17 Certificate chain length: 1
18 Certificate[1]:
19 Owner: OID.1.2.3.3=Ministero della Salute, OID.1.2.3.2.2=NULL, OID.1.2.3.2.1=Rapido, OID.1.2.3.1.7=RSSMRA80A01H501U,
OID.1.2.3.1.6=RM, OID.1.2.3.1.5=Roma, OID.1.2.3.1.4=01-01-1980, OID.1.2.3.1.3=M, OID.1.2.3.1.2=Rossi, OID.1.2.3.1.1=Mario,
CN=localhost, ST=Italia, C=IT
20 Issuer: CN=localhost, OU=MS, O=Ministero della Salute, L=Roma, ST=Italia, C=IT
21 Serial number: 1
22 Valid from: Sun Aug 20 14:00:00 CEST 2023 until: Tue Aug 22 14:00:00 CEST 2023
23 Certificate fingerprints:
24     SHA1: 3E:66:37:56:A8:07:AD:04:52:9B:FB:AC:77:42:A4:07:80:9E:01:B5
25     SHA256: C1:B1:18:08:FA:89:F1:35:91:F5:E9:53:93:05:1C:8A:AD:79:FF:09:F7:A2:DE:6B:D6:15:C9:40:02:6F:62:57
26 Signature algorithm name: SHA256withECDSA
27 Subject Public Key Algorithm: 256-bit EC (secp256r1) key
28 Version: 3
29 ...
```

**Listing 4.16** – Script per la generazione del GP 2.0 e del keystore per il player *mario*.



**Figure 4.2** – Essendo il certificato scaduto, al giocatore non è permesso accedere ai servizi offerti da Mr. Joker e, quindi, viene escluso dalla generazione casuale di stringhe.

### 4.2.3 Player: infante

Consideriamo il caso di un giocatore *infante* provvisto di un *GP* 2.0 dalla durata di 180 giorni. Essendo il player *infante* minorenne, non è ammesso alla sala bingo di *Mr Joker*.

```
1 # Generazione Chiavi Private
2 openssl genpkey -paramfile prime256v1.pem -out infante_key.pem
3
4 # Generazione Richiesta GP2.0
5 openssl req -new -key infante_key.pem -out infante_request.pem -config "C:\msys64\mingw64\etc\ssl\opensslGP.cnf"
6 openssl ca -in infante_request.pem -out infante_cert.pem -policy policy_anything -config
7 "C:\msys64\mingw64\etc\ssl\opensslGP.cnf"
8
9 # Generazione dei Keystore
10 openssl pkcs12 -export -out infante.p12 -inkey infante_key.pem -in infante_cert.pem -name infante -passout pass:aps2023
11 keytool -importkeystore -destkeystore infante_keystore.jks -srckeystore infante.p12 -srcstoretype PKCS12 -alias infante
12 -srcstorepass aps2023 -deststorepass aps2023
13
14 # Stampa parziale del keystore
15 ...
16 Alias name: infante
17 Creation date: 23 ago 2023
18 Entry type: PrivateKeyEntry
19 Certificate chain length: 1
20 Certificate[1]:
21 Owner: OID.1.2.3.3=Ministero della Salute, OID.1.2.3.2.2=BioNTech-Pfizer, OID.1.2.3.2.1=NULL,
22 OID.1.2.3.1.7=RSSVTR23C41H703I, OID.1.2.3.1.6=SA, OID.1.2.3.1.5=Salerno, OID.1.2.3.1.4=04-03-2023, OID.1.2.3.1.3=F,
23 OID.1.2.3.1.2=Rossi, OID.1.2.3.1.1=Victoria, CN=localhost, ST=Italia, C=IT
24 Issuer: CN=localhost, OU=MS, O=Ministero della Salute, L=Roma, ST=Italia, C=IT
25 Serial number: 2
26 Valid from: Wed Aug 23 23:37:40 CEST 2023 until: Mon Feb 19 22:37:40 CET 2024
27 Certificate fingerprints:
28     SHA1: A2:27:D0:27:14:62:E3:03:41:F3:5C:8A:9E:CE:8C:C9:BF:51:A0:E9
29     SHA256: FC:70:0E:A1:5B:20:18:63:47:64:BD:DB:8F:6B:C6:2D:CF:12:1F:5F:85:9B:60:A7:03:D1:56:85:2C:AF:08:96
30 Signature algorithm name: SHA256withECDSA
31 Subject Public Key Algorithm: 256-bit EC (secp256r1) key
32 Version: 3
33 ...
```

**Listing 4.17** – Script per la generazione del GP 2.0 e del keystore per il player *infante*.

**JokerCasino** Minicard /g/1 mio Drive/Università degli Studi di Salerno/Magistrale - AI e IR/1 Anno/Secondo Semestre 2022-23/AP/Projeto - JokerCasino  
e la licenza è stata rinnovata per il giorno 18/01/2024.  
**Primo Confronto** - Prima Torna (1862)  
JokerCasino - In attesa di connessioni ...  
(SSLServer) In attesa di connessioni ...  
(SSLJoker) In attesa di connessioni ...  
(SSLPlayer) Scegli utente di gioco ...  
(SSLPlayer) Scegli chiudere connessioni ...  
(SSRSMS) I seguenti dati stanno per essere inviati a Mr.Joker: Data di Nascita e Codice Fiscale!  
(SSRSMS) Sodet chiusa.  
(SSRSMS) Sodet chiusa con "Identity Provider del Ministero della Salute ...  
(SSLJoker) Sodet chiusa.  
(SSLPlayer) Connesso con Mr.Joker!  
(SSLPlayer) Riconosciuto da Mr.Joker! Accesso Negato. Il gioco è vietato per i minori di 18 anni!  
(SSLPlayer) Riconosciuto da Mr.Joker! Accesso Negato. Il gioco è vietato per i minori di 18 anni!

**Figure 4.3** – Essendo il player minorenne, al giocatore non è permesso accedere ai servizi offerti da Mr. Joker e, quindi, viene escluso dalla generazione casuale di stringhe, sebbene sia in possesso di un GP 2.0 valido.