UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Algoritmi e Protocolli per la Sicurezza

Project Work

Group members:

Adinolfi Teodoro	0622701902	t.adinolfi2@studenti.unisa.it	WP2
Amato Emilio	0622701903	e.amato16@studenti.unisa.it	WP3
Bove Antonio	0622701898	a.bove57@studenti.unisa.it	WP4
Corcione Marta	0622702020	m.corcione5@studenti.unisa.it	WP1

ANNO ACCADEMICO 2022/2023

CONTENTS

CHAPTER 1

WP1: MODELLO

In questo primo capitolo, ci concentreremo sulla definizione degli attori onesti coinvolti nel sistema, analizzando i loro obiettivi e le funzionalità che si intendono realizzare. Gli attori onesti sono quegli individui o entità che agiscono in conformità con le regole e le politiche stabilite, cercando di raggiungere i loro obiettivi senza compromettere l'integrità del sistema.
Successivamente, esamineremo i possibili avversari (o threat models) che potrebbero essere interessati a compromettere il sistema, analizzando le loro risorse e le motivazioni che li spingono ad agire. Questa analisi ci permetterà di identificare gli attacchi che il sistema potrebbe subire e di comprendere quali misure di sicurezza dovranno essere adottate per contrastarli.
Una volta compreso il contesto in cui il sistema opera, identificheremo le proprietà di sicurezza che si vorrebbe preservare in presenza di attacchi. Queste proprietà sono fondamentali per garantire il corretto funzionamento del sistema e la protezione delle informazioni sensibili.

1.1 Attori del sistema

In questo scenario, ci sono diversi attori coinvolti nel sistema, ognuno con obiettivi e funzionalità specifiche. Di seguito un elenco dettagliato di tali attori e delle loro responsabilità:

- **Mister Joker**, il proprietario della sala Bingo, ha un ruolo fondamentale nella creazione e gestione del suo ambiente di gioco online. Il suo obiettivo è offrire ai partecipanti un'esperienza di gioco sicura, trasparente e divertente. Per farlo, richiede agli studenti dell'Università degli Studi di Salerno di implementare un meccanismo per la generazione continua di stringhe casuali e l'implementazione del sistema di accesso basato sul Green Pass 2.0. Inoltre, deve comunicare efficacemente i vantaggi del sistema proposto, contrastando le argomentazioni dei no-fox, ovvero gli oppositori dell'innovazione

- **Server**, è responsabile di gestire le connessioni con i giocatori, di controllare l'accesso alle stanze virtuali e di coordinare il processo di generazione continua di stringhe casuali. Il server agisce anche come Banco in molti giochi, gestendo il progredire della partita

- **Partecipanti alla sala Bingo**, sono gli utenti che desiderano giocare ai giochi di fortuna offerti da Mister Joker; tale partecipazione richiede l'utilizzo del Green Pass 2.0. Inoltre, sono coinvolti nel processo di generazione continua nel tempo di stringhe casuali e al tempo stesso sono responsabili della gestione del loro profilo

- **Ministero della Salute**, è responsabile dell'emissione del Green Pass 2.0 ed è incaricato di rilasciare la firma digitale ad esso associata, garantendo l'autenticità e la validità delle informazioni contenute nel documento. È inoltre responsabile della revoca del Green Pass 2.0 qualora ve ne dovesse essere bisogno

- **Data Protection Authority (DPA)**, è responsabile di imporre le politiche sulla privacy e di stabilire quali informazioni del Green Pass 2.0 possono essere esibite per accedere ai servizi, come la sala Bingo virtuale

1.2 Possibili attaccanti del sistema

Nel contesto del sistema della sala Bingo proposto da Mister Joker è fondamentale analizzare e comprendere i potenziali attaccanti che potrebbero cercare di comprometterlo, considerando le loro motivazioni e le risorse a loro disposizione.

- **Bob the Trickster**

- **Tipologia:** attivo
- **Descrizione:** singolo partecipante alla sala Bingo che cerca di manipolare il meccanismo di generazione delle stringhe casuali, fornendo un contributo personale costruito appositamente per ottenere un vantaggio nel gioco
- **Risorse:** ha accesso limitato alle risorse computazionali

- **The Missing Max**
  - **Tipologia:** attivo
  - **Descrizione:** singolo partecipante alla sala Bingo il cui scopo è quello di compromettere la reputazione del servizio offerto da Mister Joker rifiutandosi di inviare il suo contributo ad ogni round e, di conseguenza, causando un attacco di tipo DoS, non consentendo lo svolgersi della partita
  - **Risorse:** non dispone di potenza di calcolo rilevante, data anche l'inutilità, e non ha accesso alle linee di comunicazione

- **The Trickster's Guild**
  - **Tipologia:** attivo
  - **Descrizione:** gruppo di partecipanti alla sala Bingo che potrebbe lavorare insieme per compromettere il meccanismo di generazione delle stringhe casuali. Questi avversari potrebbero coordinarsi per influenzare il processo di generazione delle stringhe in modo da favorire il gruppo
  - **Risorse:** maggiori risorse computazionali rispetto a quelle del singolo avversario (Bob the Trickster)

- **Cospiracy Conclave**
  - **Tipologia:** attivo
  - **Descrizione:** gruppo formato da partecipanti disonesti alla sala Bingo in combutta con individui in grado di manipolare il server della sala. Lavorano insieme per compromettere il meccanismo di generazione delle stringhe casuali con l'obbiettivo di derubare il solo giocatore onesto al tavolo
  - **Risorse:** maggiori risorse computazionali rispetto a quelle della coalizione (Trickster's Guild)

- **Jack the Manipulator**
  - **Tipologia:** attivo
  - **Descrizione:** giocatore al tavolo, si posiziona sui canali di comunicazione tra server e giocatori e il suo intento è quello di manipolare la stringa casuale calcolata dal server, in modo da trarne un vantaggio personale
  - **Risorse:** possiede una discreta potenza di calcolo e ha accesso ai canali di comunicazione tra utenti e sistema della sala Bingo
- **Carol the Mole**
  - **Tipologia:** attivo
  - **Descrizione:** un'insider che lavora per la sala Bingo. Potrebbe compromettere il sistema dall'interno per vantaggio personale, come truffe finanziarie. Nel contesto della generazione delle stringhe casuali, Carol potrebbe sfruttare il suo ruolo di dipendente per alterare il processo di generazione di tali stringhe ed aiutare uno dei partecipanti per ottenere vincite illecite
  - **Risorse:** ha accesso alle macchine che gestiscono la sala Bingo
- **Frank the Rogue Server**
  - **Tipologia:** attivo
  - **Descrizione:** è un server che si comporta in modo malevolo. Nonostante dovrebbe garantire la gestione corretta delle connessioni, dei controlli di accesso e del coordinamento del gioco, usa le informazioni del Green Pass 2.0 ricevute dagli utenti per scopi illeciti. Questo può includere l'uso improprio di informazioni personali, la vendita di queste informazioni a terzi o qualsiasi altro atto che comprometta la privacy e la sicurezza degli utenti
  - **Risorse:** ha accesso a tutte le risorse del server, compresi i dati degli utenti e i Green Pass 2.0 ricevuti per l'accesso alla sala Bingo
- **John the Fraudster**
  - **Tipologia:** attivo

- **Descrizione:** avversario che, per vie traverse, è venuto a conoscenza di tutte le informazioni personali di un individuo e tenta di impersonare quest'ultimo. Il suo obiettivo è richiedere un Green Pass 2.0 all'autorità sanitaria a nome dell'individuo di cui conosce l'identità
- **Risorse:** possiede una discreta potenza di calcolo e ha accesso a varie informazioni personali dettagliate, permettendogli di convincere l'autorità sanitaria che è l'individuo che sta cercando di impersonare

- **Alice the Eavesdropper**
  - **Tipologia:** passivo
  - **Descrizione:** avversario interessato a intercettare tutte le operazioni del sistema (autenticazione, dati del Green Pass 2.0, stringhe casuali generate, etc.). Questa tipologia di avversario può essere sia interno all'organizzazione che esterno, e non ha come principale obiettivo quello di compromettere il normale funzionamento del sistema, ma semplicemente è interessato a raccogliere quante più informazioni possibili
  - **Risorse:** la potenza computazionale a sua disposizione non è trascurabile, ed ha accesso alle varie linee di comunicazione tra le parti del sistema

- **Eve the Impersonator**
  - **Tipologia:** attivo
  - **Descrizione:** avversaria interessata ad attaccare la fase di emissione del Green Pass 2.0 o di accesso alla sala Bingo, in modo da rubare il Green Pass dell'utente. Di conseguenza, potrebbe impersonare l'utente di cui ha rubato il Green Pass, al fine di prosciugare il suo conto o giocare al posto suo.
  - **Risorse:** possiede una discreta potenza di calcolo e ha accesso ai canali di comunicazione sia tra gli utenti e la sala Bingo sia tra gli utenti e il Ministero della Salute

- **Simon the Forger**
  - **Tipologia:** attivo

- **Descrizione:** affermato NoVax il cui scopo è creare dei Green Pass falsi che possano essere utilizzati dai membri della sua comunità
- **Risorse:** ha accesso a risorse che gli permettono di creare dei Green Pass pur non essendo una autorità competente

- **Larry the No-Fox**
  - **Tipologia:** passivo/attivo
  - **Descrizione:** un avversario che si oppone all'innovazione e alle nuove tecnologie, sostenendo che queste soluzioni sono vulnerabili agli attacchi informatici e che compromettono la privacy degli utenti. Ha come scopo quello di amplificare notizie di violazioni di sicurezza o problemi di privacy del sistema per influenzare l'opinione pubblica
  - **Risorse:** spiccate abilità comunicative

1.2.1 Collocazione degli attaccanti all'interno del sistema

Di seguito sono riportati i vari scenari che mostrano l'interazione tra gli utenti e i server del Ministero della Salute e della sala Bingo, con i relativi messaggi scambiati per ottenere le funzionalità richieste per il sistema e il possibile collocamento degli avversari precedentemente descritti.

Ottenimento Green Pass 2.0

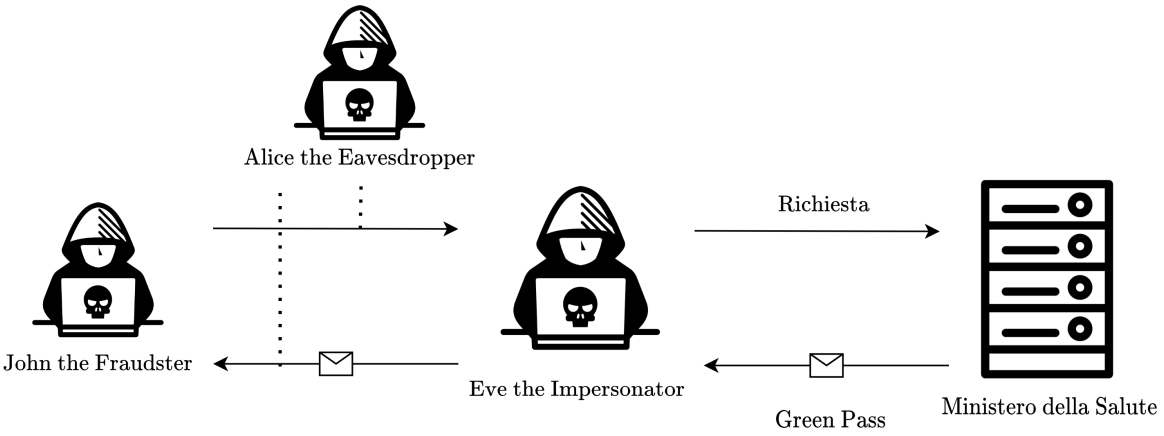

Figure 1.1

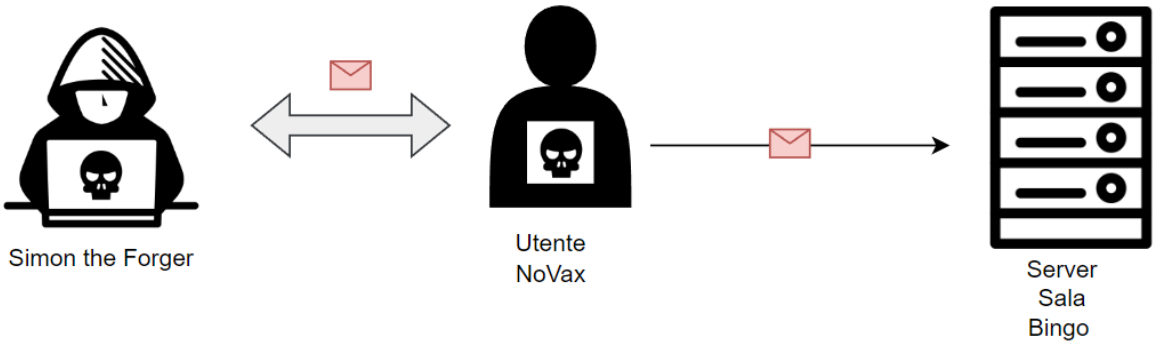

Figure 1.2

Identificazione al proprio profilo mediante Green Pass 2.0

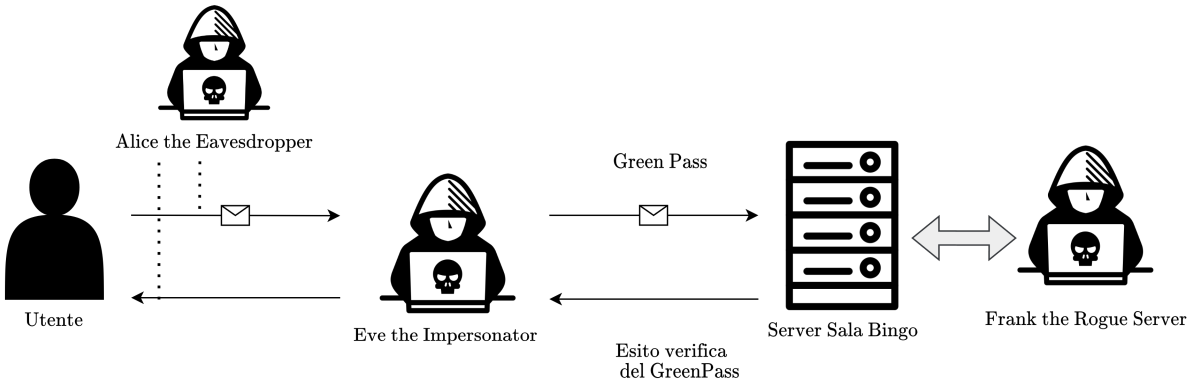

Figure 1.3

Accesso al tavolo di gioco

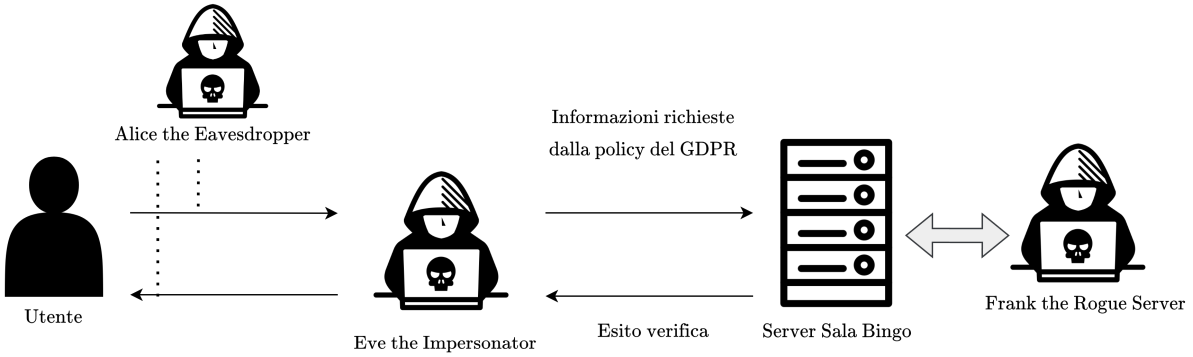

Figure 1.4

Generazione continua di stringhe casuali

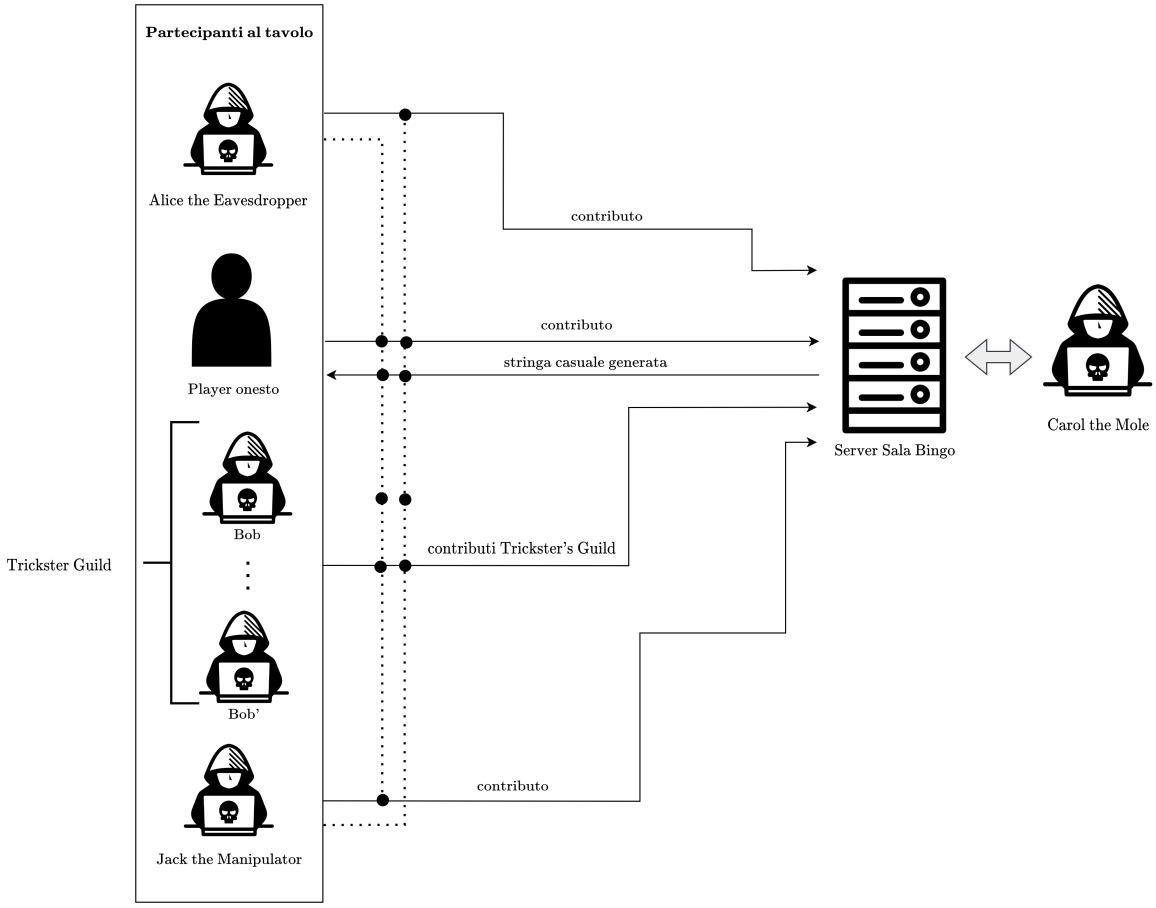

Figure 1.5

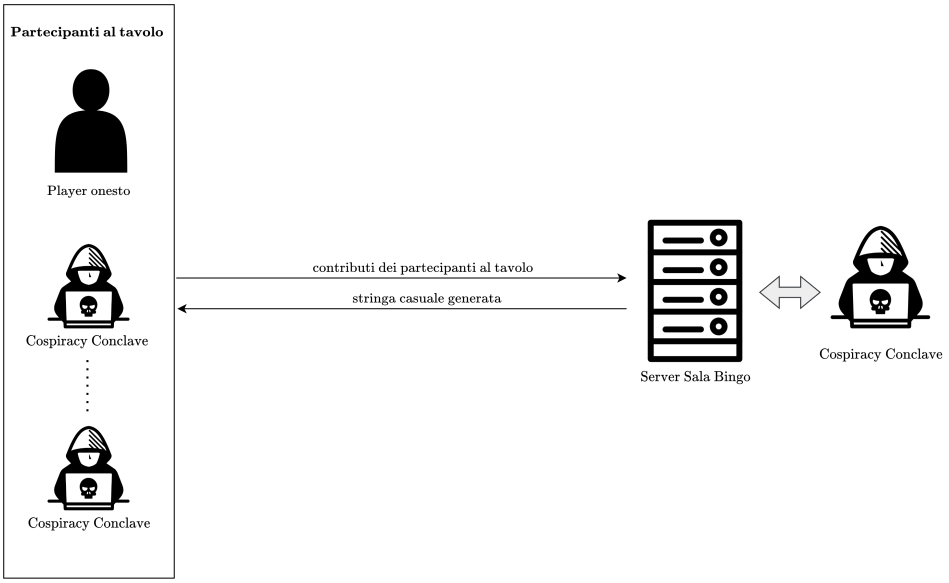

Figure 1.6

1.3 Proprietà

Nel presente sottoparagrafo, verranno esaminate le principali proprietà che il sistema di stanze virtuali per giochi di fortuna ideato da Mister Joker deve possedere per garantire un ambiente sicuro, affidabile e performante. Le proprietà saranno analizzate in base ai quattro pilastri fondamentali: *confidenzialità*, *integrità*, *trasparenza* ed *efficienza*.

1. **Confidenzialità**

   - **C.1**: i dati degli utenti presenti nel Green Pass 2.0 devono essere protetti, ovvero accessibili solo dalle parti autorizzate in maniera controllata
   - **C.2**: garantire la privacy delle comunicazioni tra gli utenti e il server della sala Bingo, in modo che non sia possibile carpire informazioni utili al processo di generazione delle stringe casuali
   - **C.3**: garantire la privacy delle comunicazioni tra gli utenti e il Ministero della Salute, in modo che non sia possibile carpire dati personali dell'utente che richiede il Green Pass e/o rubarlo
   - **C.4**: garantire la privacy delle comunicazioni tra gli utenti e il server della sala Bingo, in modo che non sia possibile carpire dati personali dell'utente che prova ad accedere ad essa
   - **C.5**: garantire che un utente malintenzionato non sia in grado di identificarsi in un altro utente per l'accesso ad un servizio sfruttando il Green Pass 2.0
   - **C.6**: il Green Pass 2.0 deve poter essere mostrato in maniera sicura e verificabile, garantendo che tali informazioni, una volta verificate per l'autenticazione, non vengano conservate o memorizzate dal server per qualsiasi scopo
   - **C.7**: garantire che un utente non sia in grado di richiedere un Green Pass valido che contenga informazioni non correlate alla sua persona

2. **Integrità**

   - **I.1**: preservare l'integrità della stringa casuale estratta a partire dai contributi degli utenti e del server, impedendo la manipolazione o la

sostituzione dei contributi o della stessa stringa con valori predeterminati da parte di utenti malevoli

- **I.2**: il Green Pass 2.0 deve essere verificabile, ovvero deve essere possibile verificare da tutti che sia stato rilasciato dall'autorità competente
- **I.3**: deve essere possibile verificare che i dati esibiti da un utente appartengano effettivamente al Green Pass 2.0 in suo possesso
- **I.4**: la stringa estratta, generata a partire dai contributi dei partecipanti alla sala deve essere impredicibile
- **I.5**: garantire che il Green Pass esibito sia effettivamente riconducibile all'utente che lo mostra
- **I.6**: non deve essere possibile per un partecipante astenersi al processo di generazione delle stringhe casuali
- **I.7**: garantire che ogni azione all'interno delle sale virtuali non possa essere contestata senza essere corredata da valide motivazioni digitalmente documentate

3. **Trasparenza**

- **T.1**: il sistema di generazione di stringhe casuali deve utilizzare algoritmi e protocolli pubblicamente noti e disponibili, in modo che gli utenti possano verificare l'integrità della stringa estratta, rendendo così difficile, per un utente malintenzionato, influenzare il processo di generazione senza essere scoperto
- **T.2**: il processo di generazione e verifica del Green Pass 2.0 deve essere noto e verificabile da tutti
- **T.3**: il sistema deve garantire che il processo di generazione delle stringhe sia imparziale

4. **Efficienza**

- **E.1**: garantire che la generazione di stringhe casuali sia eseguita in modo rapido ed efficiente, riducendo al minimo l'impatto sulle prestazioni del sistema e l'esperienza degli utenti

- **E.2**: garantire che la verifica del Green Pass 2.0 sia eseguita in modo rapido ed efficiente, riducendo al minimo l'impatto sulle prestazioni del sistema e l'esperienza degli utenti
- **E.3**: l'identificazione presso un servizio digitale tramite Green Pass 2.0 deve avvenire in modo semplice e rapido

1.4 Completeness

Definiamo ora la *completeness*, ovvero il comportamento del sistema nel caso in cui tutte le parti in gioco si comportino in maniera onesta. Sotto queste ipotesi, quando tutte le parti del sistema operano come stabilito, senza discostarsi dal comportamento standard, il sistema svolge efficacemente la sua funzione. In particolare:

- se l'utente ha effettivamente ricevuto il vaccino, facendo richiesta al Ministero, con i propri dati, riceverà un Green Pass 2.0 valido
- se l'utente è in possesso di un Green Pass 2.0 valido può accedere alla sala Bingo di Mister Joker
- se tutti i partecipanti al tavolo di gioco forniscono il proprio contributo al server questo genererà una stringa casuale a partire da questi ultimi

CHAPTER 2

WP2: SOLUZIONE

In questo capitolo ci concentreremo sulla presentazione di una soluzione che risponde al modello identificato nel *Work Package 1 (WP1)*. L'obiettivo è quello di proporre un sistema che riesca a raggiungere un ragionevole compromesso tra efficienza, trasparenza, confidenzialità e sicurezza.
Concentreremo la nostra attenzione sui seguenti problemi chiave:

- creazione e ottenimento del Green Pass 2.0
- identificazione e accesso alla sala Bingo
- generazione continua di stringhe casuali

2.1 Panoramica generale di funzionamento

Il processo inizia con la richiesta e la successiva creazione del Green Pass 2.0. L'autorità sanitaria riconosciuta - il Ministero della Salute - emette il Green Pass 2.0 a un individuo compilato con i dati anagrafici e sanitari a lui appartenenti, e raccolti dal ministero in sede di vaccinazione (per le informazioni sul vaccino) o già in suo possesso (generalità). L'operazione di emissione avviene una sola volta, dopodiché l'autorità sanitaria non è più coinvolta nel processo, salvo per eventuali revoche del Green Pass.

Una volta che l'individuo è in possesso del Green Pass 2.0, può utilizzarlo per

identificarsi e accedere alla sala Bingo di Mister Joker. Questo passaggio consente all'individuo di accedere al suo profilo personale, dove può gestire il suo saldo online.

Una volta identificato con successo, l'individuo può richiedere l'accesso alla funzionalità di generazione continua di stringhe casuali, a condizione che le informazioni contenute nel suo Green Pass 2.0 soddisfino le politiche imposte dal Garante per la Protezione dei Dati Personali (DPA).

2.2 Generazione e utilizzo del Green Pass 2.0

Nella presente sezione, svilupperemo un protocollo per la generazione di un Green Pass 2.0 e l'impiego di tale documento per l'accesso a servizi digitali. Questo processo coinvolge come attori chiave: l'utente, il Ministero della Salute e il server della sala Bingo.

In questo protocollo, i partecipanti interagiscono con il Ministero della Salute per la richiesta e generazione del Green Pass e con il server della sala Bingo per usufruire dei servizi da essa offerti. Andremo ad analizzare dapprima il modo con cui un utente interessato a ricevere il Green Pass 2.0 possa farne richiesta al Ministero, chiarendo anche quello che è il processo di rilascio per via telematica di questo ultimo.
Andremo a delineare il contenuto e la struttura del Green Pass 2.0 affinché questo possa essere utilizzato sia come strumento digitale valido per l'identificazione e l'accesso ad un servizio digitale, sia in modo che le informazioni in esso contenute possano essere mostrate su richiesta cercando di mantenere la maggiore riservatezza possibile.
Analizzeremo, infine, un possibile processo di identificazione ed accesso ad un servizio digitale, in particolare, quello offerto dalla sala Bingo di Mister Joker.

2.2.1 Supposizioni

Alla base del funzionamento del protocollo proposto, si suppone che l'utente si sia recato personalmente in un hub vaccinale, dove, a monte della somministrazione del vaccino, sia stato identificato attraverso i propri documenti in corso di validità e che in tale occasione abbia fornito agli operatori **il suo numero di telefono**.
Gli operatori, a questo punto, hanno provveduto a far confluire verso i database del Ministero della Salute le informazioni circa il vaccino somministrato all'utente ed il suo

numero di telefono, previa firma del documento informativo sul trattamento dei dati personali.
Supponiamo inoltre che:

- il server del Ministero della Salute sia una parte fidata: questo ci permette di supporre che i Green Pass da lui emessi contengano informazioni corrette e che il procedimento di rilascio e la struttura del Green Pass 2.0 siano perfettamente in accordo al protocollo di seguito esposto
- il ministero abbia reso pubbliche tutte le informazioni necessarie alla verifica del Green Pass, in particolare gli algoritmi ed i protocolli utilizzati: questo presuppone che la struttura del Green Pass 2.0, il procedimento di rilascio ed anche di identificazione tramite quest'ultimo siano disponibili a chiunque
- affidandoci alla PKI già esistente supponiamo che il server del Ministero della Salute, al quale l'utente si collega per richiedere il rilascio del Green Pass 2.0, possegga un certificato valido rilasciato da una *certification autority* utilizzato appositamente per firmare i Green Pass 2.0 da lui rilasciati e possegga un certificato valido da utilizzare per l'esecuzione del protocollo TLS necessario per instaurare una comunicazione sicura con l'utente

Procediamo adesso con l'analisi dettagliata di quanto delineato fino a questo momento.

2.2.2 Identificazione dell'utente

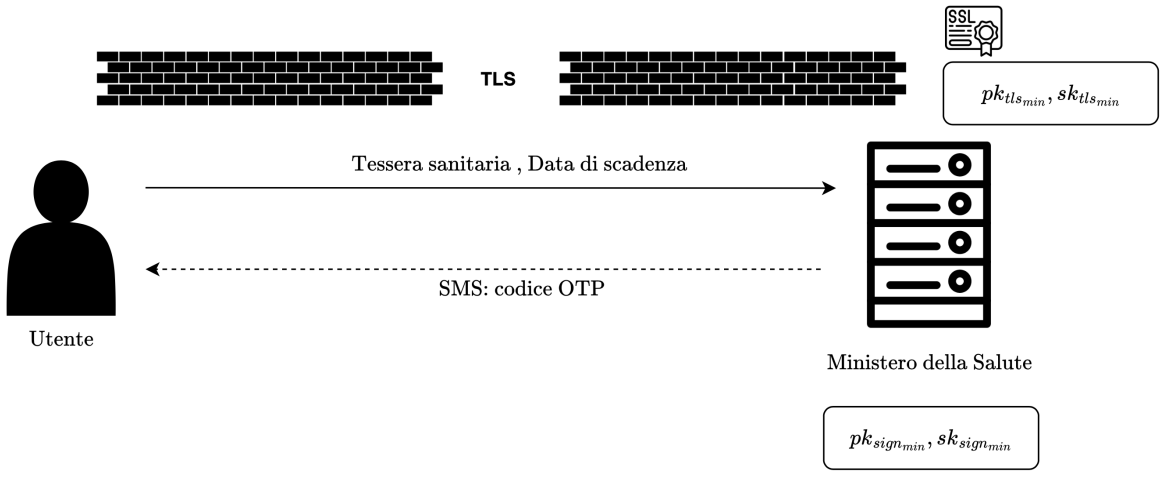

Figure 2.1

Quando l'utente decide di collegarsi al sito del Ministero della Salute per richiedere il Green Pass 2.0 viene instaurata una connessione TLS tra quest'ultimo ed il server. Viene a questo punto eseguito il seguente protocollo:

1. l'utente invia al server il numero di identificazione della tessera sanitaria, in particolare le ultime 8 cifre, in quanto rappresentano un identificativo univoco per la singola tessera sanitaria e la data di scadenza della stessa
2. il server invia un *sms* all'utente con un codice OTP

In particolare, il codice OTP viene generato come una stringa random esadecimale di lunghezza 24 bit (6 caratteri). Il server associa a tale codice un counter per limitare il numero di tentativi che l'utente può effettuare posto uguale a 3 prima di doverne richiedere uno nuovo. Inoltre, il tempo necessario per richiedere un nuovo OTP aumenta progressivamente con ogni nuova richiesta con una politica esponenziale nel tempo del tipo $5 \cdot 2^n$ minuti con $n \in \mathbb{N}$. Tale scelta serve a prevenire tentativi di accesso fraudolenti attraverso attacchi di tipo brute-force. Una volta entrato in possesso del codice OTP è possibile procedere con la prossima fase, ovvero, la richiesta del Green Pass 2.0

2.2.3 Richiesta del Green Pass 2.0

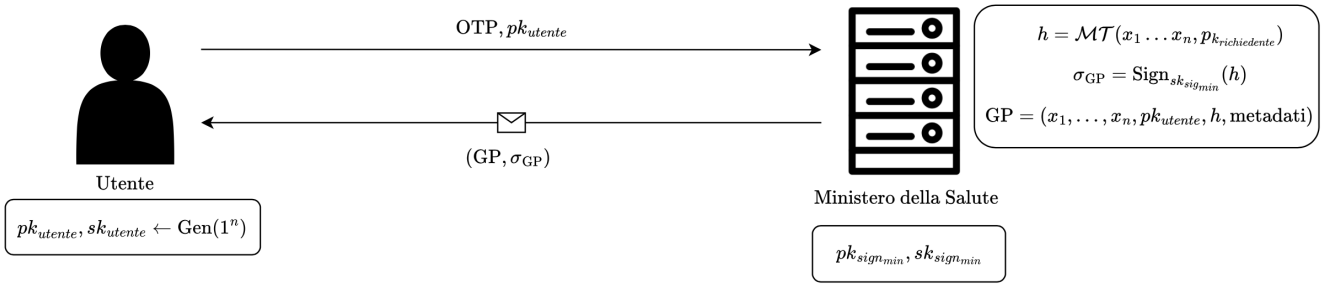

Figure 2.2

1. L'utente procede con l'effettiva richiesta del Green Pass 2.0 inviando:
    - il codice OTP ricevuto dal Ministero al passo precedente
    - una chiave pubblica pk_{utente} generata in accordo all'algoritmo $\mathrm{Gen}(1^n)$

2. Il Ministero della Salute riceve tali informazioni e:

   - verifica che l'OTP sia valido (in caso negativo, il processo termina)
   - procede con la generazione del Green Pass (il cui formato sarà discusso a breve) che invia all'utente, corredato da una firma σ_{GP} utilizzata per garantire l'autenticità del documento

Contributo segreto dell'utente La chiave pubblica, inviata dall'utente al Ministero, permette di generare un documento digitale dipendente da un informazione segreta (la chiave privata) in possesso dell'utente e solo di quest'ultimo.

Generazione del Green Pass 2.0

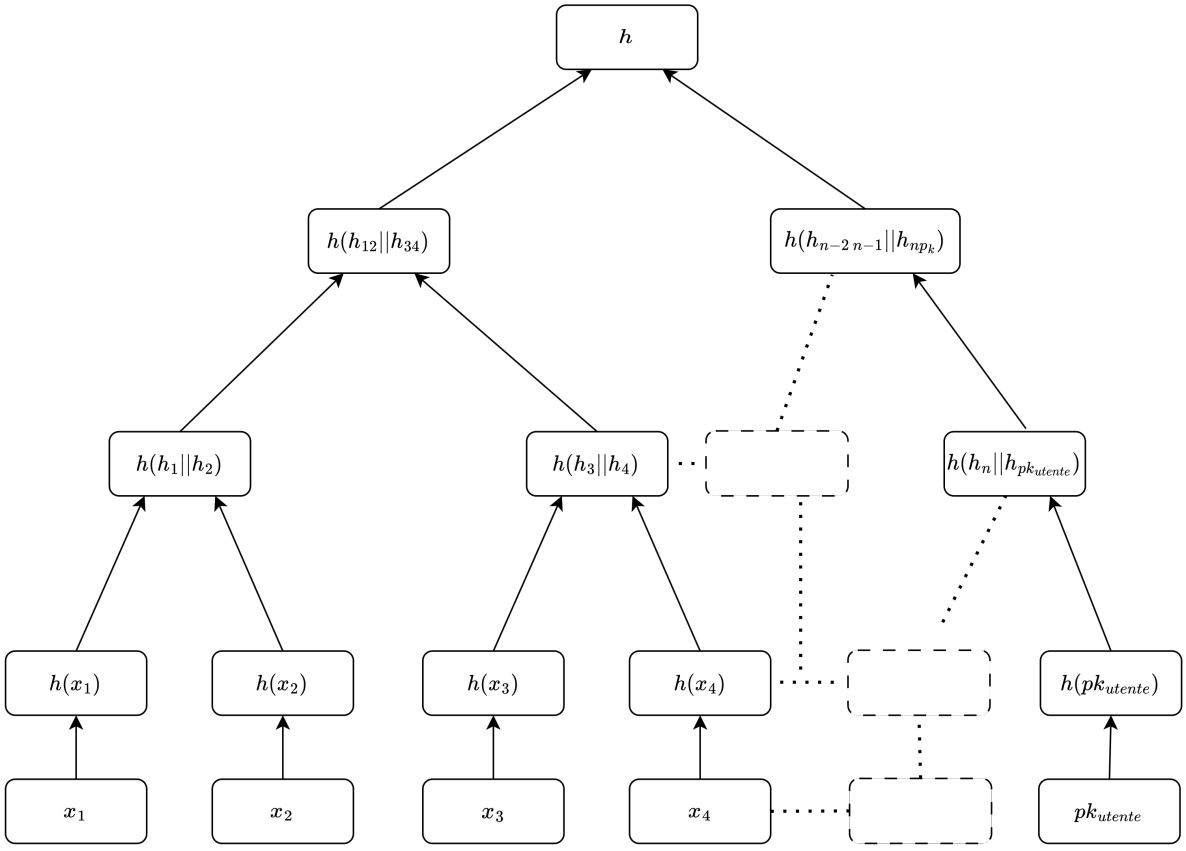

Figure 2.3: Merkle Tree Green Pass 2.0

Il contenuto del documento Il Green Pass 2.0 include dati e generalità dell'utente che, al momento del rilascio, sono già in possesso del Ministero della Salute

(analogamente al Green Pass 1.0) oltre che la chiave pubblica fornita dall'utente al passo precedente; tali dati saranno indicati con $x_1 \dots x_n$.
Per ottenere una traccia digitale unica del documento, ma che al tempo stesso permetta all'utente finale di mostrarne solo una parte, il Ministero utilizza l'algoritmo $\mathcal{MT}$ alimentato tramite i dati sopra citati insieme al contributo personale dell'utente, ovvero la chiave pubblica pk_{utente}.
All'interno della Figura 2.2, quanto appena affermato viene indicato come:

$$\mathcal{MT}(x_1 \dots x_n, pk_{utente})$$

Una volta ottenuto il valore h rappresentativo dell'intero Green Pass 2.0 il Ministero procederà con l'applicarvi una firma digitale σ_{GP} tramite l'algoritmo $\text{Sign}_{sk_{sig_{min}}}(h)$. Questo assicura che il documento sia stato rilasciato effettivamente dal Ministero e soltanto da quest'ultimo.
Si noti che a questo punto, per verificare che un certo dato x_i sia parte del Green Pass 2.0 identificato da h, sarà sufficiente corredare x_i con gli hash necessari al calcolo di h a partire dalla foglia x_i.
Una volta calcolata e firmata h insieme a tutti gli *hash* intermedi che vanno a costruire il *Merkle Tree* vengono salvati all'interno del documento ed entrano a far parte di quella categoria di informazioni che, da questo momento in poi, considereremo come **metadati**. Con il simbolo GP in Figura 2.2 facciamo dunque riferimento al Green Pass 2.0 nella sua totalità, ovvero:

$$\text{GP} = (\underbrace{x_1 \dots x_n, pk_{utente}}_{\text{dati}}, \text{metadati})$$

La coppia di informazioni GP, σ_{GP} a questo punto sarà inviata tramite il canale di comunicazione sicuro instaurato tra il server del Ministero della Salute e l'utente, aperto in precedenza.

2.2.4 Utilizzo del Green Pass 2.0 per l'accesso al profilo

A questo punto si suppone che l'utente abbia con sé un Green Pass 2.0 valido e che desideri accedere ad un servizio online. In particolare, descriveremo il meccanismo con cui l'utente può accedere/registrarsi al sito di Mister Joker e la procedura di verifica dei dati del Green Pass 2.0 che gli permetteranno l'accesso alle sale virtuali.

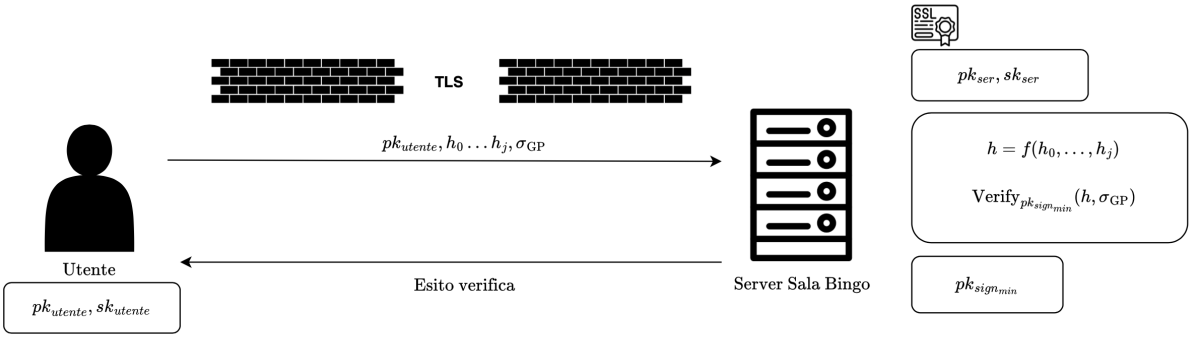

Figure 2.4: Esempio di connessione ai servizi online offerti da Mister Joker

Supponiamo che il server di Mister Joker possieda un certificato valido, rilasciato da una *certification autority* tramite cui è possibile per un utente avviare una sessione TLS. Per fornire la possibilità ad un utente di accedere ai servizi tramite l'utilizzo del Green Pass 2.0 devono essere analizzati due punti chiave che andremo ora ad esporre:

1. deve essere fornito dall'utente un Green Pass 2.0 in corso di validità che sia dunque stato rilasciato (e firmato) dal Ministero della Salute
2. una volta verificata l'autenticità del documento, bisogna **autenticare** l'utente e quindi il server deve essere in grado di verificare che, colui che ha presentato il Green Pass 2.0 considerato valido sia effettivamente il proprietario di tale documento

Verifica della validità del documento Una volta stabilita la connessione sicura, l'utente procede con l'invio delle seguenti informazioni:

1. la sua chiave pubblica pk_{utente} che ricordiamo essere un campo del Green Pass 2.0
2. tutti i **metadati necessari alla verifica di** h (radice del *Merkle Tree*). In particolare l'utente invierà la *Merkle proof* associata alla chiave pubblica pk_{utente}

A questo punto il server, che ha accesso alla chiave pubblica utilizzata dal Ministero per la firma dei Green Pass 2.0 ($pk_{sig_{min}}$), seguirà il seguente protocollo:

1. calcola h tramite l'algoritmo che in Figura 2.4 è indicato con f. Tale algoritmo procede nel calcolo di h a partire dal dato in chiaro pk_{utente} e la *Merkle Proof* costituita dagli j hash intermedi $h_0, \dots, h_j$

2. tramite l'algoritmo $\text{Vrfy}_{pk_{sign_{min}}}(h, \sigma_{GP})$ il server può verificare l'autenticità del documento

Si noti che, tramite questa sequenza di operazioni, il server della sala Bingo è ora in possesso della chiave pubblica pk_{utente} e sa che questa appartiene ad un Green Pass 2.0 valido; può quindi fare affidamento al fatto che l'utente conosca un'informazione segreta correlata a tale chiave (sk_{utente}). L'esito della verifica viene comunicato all'utente e in caso questo sia positivo, l'utente ed il server procedono con la fase di identificazione.

Identificazione dell'utente L'identificazione dell'utente può essere gestita tramite l'applicazione del *protocollo di identificazione di Schnorr*. In particolare l'utente può essere identificato come il **prover**, mentre il server della sala Bingo vestirà i panni del **verifier**, infatti, affinché il server sia convinto che l'utente sia il vero possessore del Green Pass 2.0 è necessario che egli sia in grado di dimostrargli di essere a conoscenza di un segreto (sk_{utente}) a partire da pk_{utente}. Il protocollo si articola nel modo seguente:

- l'utente seleziona un valore casuale $r \in \mathbb{Z}_q$ ed invia al server $a = g^r$
- il server sceglie un valore $c \in \mathbb{Z}_q$ e lo invia all'utente
- l'utente invia al server il valore $z = r + cx \in \mathbb{Z}_q$

Il server può verificare tramite questo scambio che l'utente sia a conoscenza del segreto $x \Leftrightarrow sk_{utente}$. Il server può infatti verificare che $g^z \stackrel{?}{=} a(pk_{utente})^c$ concludendo che deve necessariamente trattarsi della persona a cui il Green Pass 2.0 è stato rilasciato.
Una volta identificato l'utente, il server verifica se ad essersi identificato è un nuovo utente oppure uno già registrato al sito, ciò può avvenire ad esempio grazie al fatto che il server memorizza pk_{utente} insieme ai dati di gioco a lui associati.

Green Pass 2.0 valido ma revocato Essendo il Green Pass 2.0 un documento revocabile, un'ulteriore verifica che viene eseguita dal server della Sala Bingo, consiste oltre che nella verifica della validità del documento sulla base di σ_{GP}, anche nel controllo che tale Green Pass 2.0, non sia incluso nella lista dei documenti digitali revocati.
Si suppone che una lista contenente queste informazioni sia rilasciata e manutenuta dal Ministero della Salute e che sia consultabile sia dall'applicativo client sia da quello server. Nel caso in cui l'utente abbia sottomesso un Green Pass 2.0 revocato, quello che in Figura 2.4 è indicato come esito verifica, sarà negativo, segnando la fine del protocollo.

2.2.5 Utilizzo del Green Pass 2.0 per l'accesso al tavolo di gioco

A seguito dell'identificazione, l'utente avrà accesso alla pagina principale del sito di Mister Joker dove potrà verificare il suo saldo e gestire il proprio profilo, tuttavia, l'accesso ad una delle sale virtuali richiede che l'utente invii una serie di informazioni del suo Green Pass 2.0 necessarie in quel momento, decise dal *Garante della Protezione dei Dati Personali*.

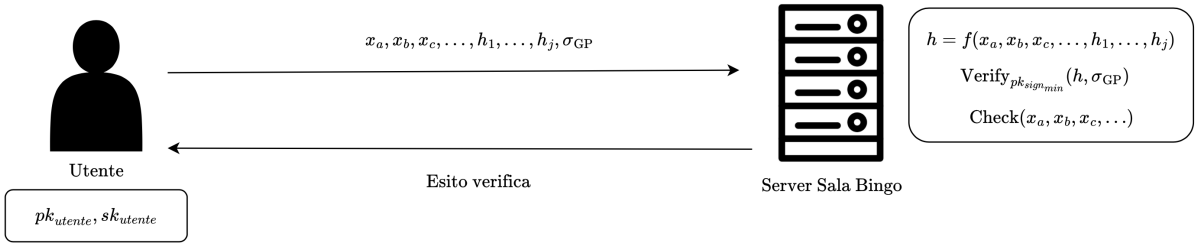

Figure 2.5: Ingresso alla sala virtuale

1. L'utente che decide di partecipare ad un gioco invia le informazioni richieste per l'accesso alla sala virtuale in accordo alle politiche giornaliere del GDPR. Tali informazioni in Figura 2.5 sono evidenziate tramite $x_a, x_b, x_c, \ldots$ dove $a, b, c \in \mathbb{N}$ e appartenenti all'intervallo $[0, n]$, con n numero dei campi del Green Pass 2.0.
   Le informazioni sono inoltre corredate di *Merkle Proof* in modo che il server possa verificare la correttezza e l'appartenenza al Green Pass di queste ultime.

2. Il server, dopo aver ricevuto le informazioni dell'utente può procedere con le seguenti operazioni:

   - calcola h andando ad utilizzare nuovamente l'algoritmo f a partire dai dati $x_a, x_b, x_c \ldots$ e la *Merkle Proof*
   - verifica che la firma di h (il valore σ_{GP} inviato dall'utente) sia effettivamente stata rilasciata dal Ministero, accertandosi che i dati inviati siano autentici
   - controlla che la politica del giorno imposta dal GDPR sia verificata e che quindi i dati $x_a, x_b, x_c \ldots$ soddisfino i requisiti

A questo punto l'utente ha il permesso di entrare all'interno di una sala virtuale in cui

potrà prendere parte al protocollo di generazione delle stringhe causali continue nel tempo descritto successivamente.

2.3 Generazione continua delle stringhe casuali

Nella presente sezione, svilupperemo un protocollo per la generazione continua di stringhe casuali, come richiesto da Mister Joker, che garantisca imparzialità e non manipolabilità. Questo processo coinvolge sia i partecipanti alla stanza virtuale che il server della sala Bingo.

In particolare, i players interagiscono con il server della sala Bingo partecipando al gioco, contribuendo al processo di generazione delle stringhe casuali ed analizzando e verificando il risultato di tale processo.

2.3.1 Soluzione proposta

Si assume che alla stanza virtuale partecipino p giocatori e che ognuno di essi e il server della sala Bingo abbiano rispettivamente una coppia di chiavi pubblica/privata (pk_i, sk_i) e (pk_s, sk_s).

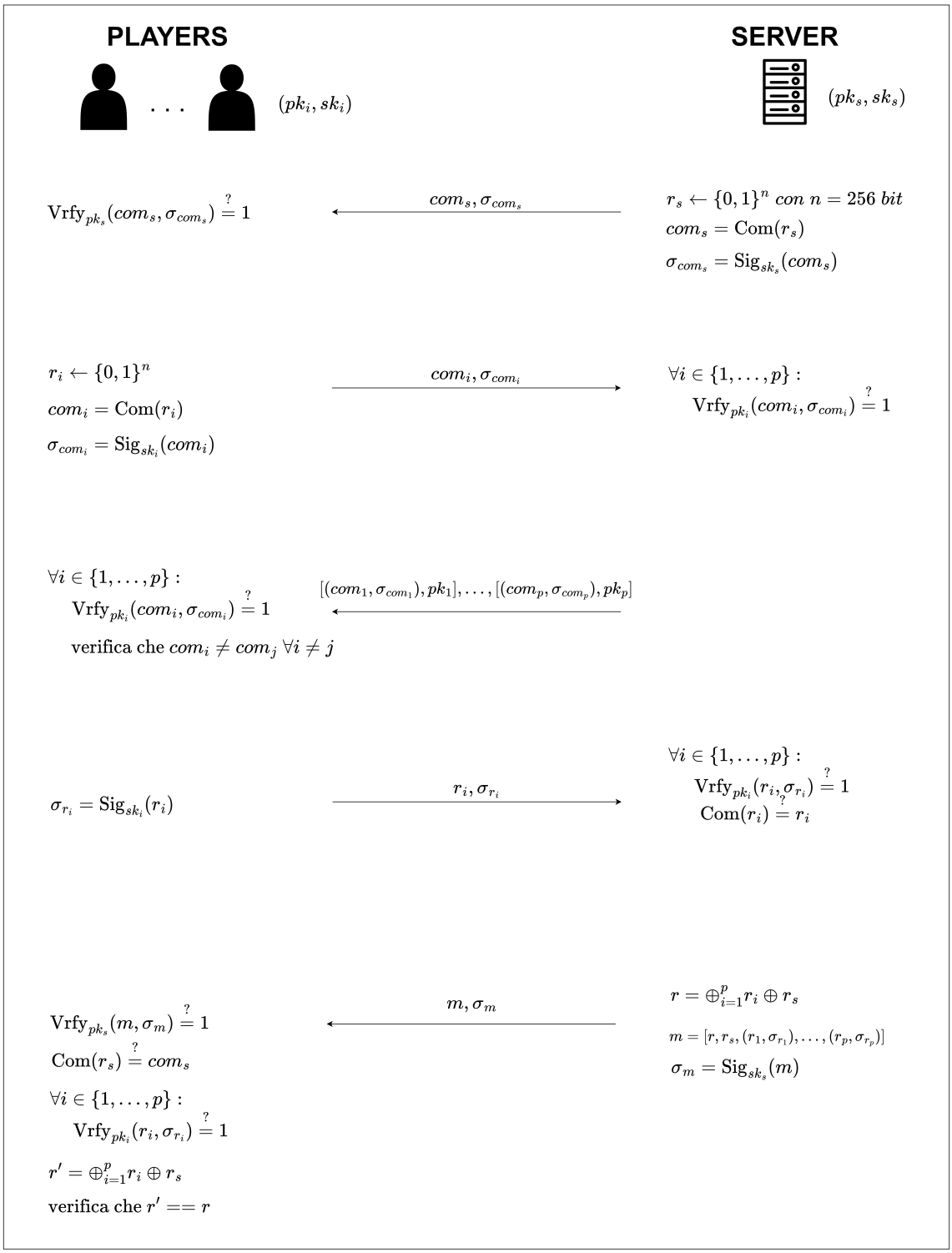

Figure 2.6: Protocollo associato alla soluzione proposta

Gli step dettagliati del protocollo proposto sono di seguito riportati:

1. ad inizio round, il server genera le seguenti informazioni:

   - un contributo casuale

$$r_s \leftarrow \{0,1\}^n$$

con $n = 256$ *bit* di cui, tramite uno *schema di commitment con random oracle* calcola l'*impegno*

$$com_s = Com(r_s)$$

- ed il commitment firmato, ovvero

$$\sigma_{com_s} = Sig_{sk_s}(com_s)$$

Tali informazioni generate, (com_s, σ_{com_s}), saranno poi inviate ad ogni player

2. ogni player i verifica la firma associata al commitment inviato dal server

$$\text{Vrfy}_{pk_s}(com_s, \sigma_{com_s}) \stackrel{?}{=} 1$$

genera un contributo casuale

$$r_i \leftarrow \{0,1\}^n$$

ne calcola il commitment

$$com_i = \text{Com}(r_i)$$

e gli associa la firma

$$\sigma_{com_i} = Sig_{sk_i}(com_i)$$

Una volta recepite tali informazioni, il server, per ogni player, ne verifica la firma

$$Vrfy_{pk_i}(com_i, \sigma_{com_i}) \stackrel{?}{=} 1$$

per poi collezionare per ognuno di essi la coppia (com_i, σ_{com_i})

3. il server invia a ciascun player i commitment di tutti gli altri giocatori, corredati di firma e relativa chiave pubblica

$$[(com_1, \sigma_{com_1}), pk_1], ..., [(com_p, \sigma_{com_p}), pk_p]$$

Alla ricezione, ciascun player verifica le firme associate ai commitment

$$\text{Vrfy}_{pk_i}(com_i, \sigma_{com_i}) \stackrel{?}{=} 1$$

e controlla che non vi siano commitment uguali. In caso negativo, ciò può essere sintomo di un *replay attack* dato dalla collaborazione disonesta tra un insieme di partecipanti e il server. Pertanto, se ciò dovesse accadere, tutti i players onesti, insospettiti di un possibile imbroglio, abbandonano la stanza virtuale, lasciando al tavolo di gioco i soli disonesti

4. a questo punto, se la verifica è andata a buon fine, vi è la fase di *reveal* in cui ogni player firma il proprio contributo

$$\sigma_{r_i} = \mathrm{Sig}_{sk_i}(r_i)$$

e invia la coppia r_i, σ_{r_i} al server, il quale verifica la firma

$$\mathrm{Vrfy}_{pk_i}(r_i, \sigma_{r_i}) \stackrel{?}{=} 1$$

e che i contributi inviati dai players "aprano" correttamente il commitiment invato in precedenza

$$\mathrm{Com}(r_i) \stackrel{?}{=} r_i$$

5. successivamente, avendo collezionato tutti i contributi, il server può calcolare la stringa complessiva generata come

$$r = \oplus_{i=1}^{p} r_i \oplus r_s$$

e invia ad ogni player r, il suo contributo r_s per l' "apertura" del commitment e tutte le coppie (r_i, σ_{r_i}) relative al contributo di ciascun player

$$m = [r, r_s, (r_1, \sigma_{r_1}), ..., (r_p, \sigma_{r_p})]$$

firmando tale messaggio con

$$\sigma_m = Sig_{sk_s}(m)$$

Questa fase è gestita tramite un *timeout*, entro il quale il server è incaricato dell'invio delle informazioni precedentemente esposte ed oltre il quale, se tali informazioni non vengono recapitate a tutti i players, la partita viene considerata come conclusa. Questo viene fatto perché la stringa casuale finale calcolata dal server potrebbe portarlo, ad esempio, in una situazione di forte svantaggio, e di

conseguenza questo potrebbe rifiutarsi di inviare tale stringa insinuando che un player non abbia rivelato il proprio commitment. Pertanto, l'utilizzo di tale meccanismo, oltre alla possibilità di eventuali penalizzazioni, dovrebbe scoraggiare tale comportamento da parte del server.

Ricevuta la coppia (m, σ_m), ogni player verifica la firma associata al messaggio m

$$Vrfy_{pk_s}(m, \sigma_m) \stackrel{?}{=} 1$$

verifica che il contributo r_s "apra" il commitment com_s

$$\mathrm{Com}(r_s) \stackrel{?}{=} com_s$$

controlla la firma associata ad ogni contributo r_i degli altri player

$$Vrfy_{pk_i}(r_i, \sigma_{r_i}) \stackrel{?}{=} 1$$

per poi calcolare

$$r' = \oplus_{i=1}^{p} r_i \oplus r_s$$

e assicurarsi che sia uguale a quello ricevuto dal server

$$r' \stackrel{?}{=} r$$

Come per lo *step 5*, anche negli *step 2* e *4* è presente un meccanismo basato su *timeout* da parte del server per cui se un partecipante non invia inizialmente il commitment del proprio contributo o il contributo stesso in fase di *reveal* entro un intervallo di tempo stabilito, questo viene escluso dalla partita in corso.

2.3.2 Possibile vulnerabilità riscontrata nella soluzione proposta

Il protocollo precedentemente proposto non offre ai partecipanti alla stanza virtuale un mezzo autonomo per verificare le operazioni eseguite dal server. Di conseguenza, essi non possiedono gli strumenti necessari per identificare un'eventuale condotta disonesta da parte di quest'ultimo. In particolare, un giocatore x, potrebbe essere erroneamente incolpato di non aver contribuito come previsto, questo perché il server potrebbe trarre

dei vantaggi non includendo il suo contributo, senza che gli altri partecipanti siano consapevoli, come già descritto nella sezione precedente.

Si tratta quindi di un punto debole del protocollo, in quanto non consente ai giocatori di rilevare e segnalare comportamenti disonesti.

2.3.3 Proposta di soluzione

In questa sezione si presenterà una proposta di soluzione che, in aggiunta a quanto esposto nella prima, risolve la possibile vulnerabilità analizzata in precedenza. Gli attori principali di questa fase coinvolti sono: i players, il server della sala Bingo e la blockchain.
In particolare, si è deciso di utilizzare una **blockchain permissioned**, una soluzione che assicura trasparenza, sicurezza e fiducia tra le parti coinvolte. In questo tipo di blockchain, l'accesso è limitato e controllato: solo gli utenti che hanno ottenuto l'approvazione possono partecipare alla rete e alle sue attività. La *governance* della blockchain è affidata ai proprietari delle varie sale Bingo, creando un ambiente in cui ogni sala ha un ruolo attivo nel mantenere l'integrità del sistema. Questo approccio alla governance non solo distribuisce il controllo e l'autorità tra più partecipanti, ma garantisce anche che le decisioni siano prese da coloro che hanno un interesse diretto nel successo del sistema. Inoltre, i partecipanti alla sala Bingo e il server sono gli unici ad avere accesso a questa blockchain.

L'idea di base è che i giocatori utilizzino la blockchain per immettere i loro contributi e renderli pubblici a tutti i partecipanti alla stanza virtuale, compreso il server, offrendo trasparenza nel processo di pubblicazione e recupero dei contributi dei giocatori, proprietà che nella prima soluzione proposta non potevano essere coperte in quanto presente come intermediario tra i players il solo server.

Nel seguito viene riportato lo schema associato alla soluzione proposta:

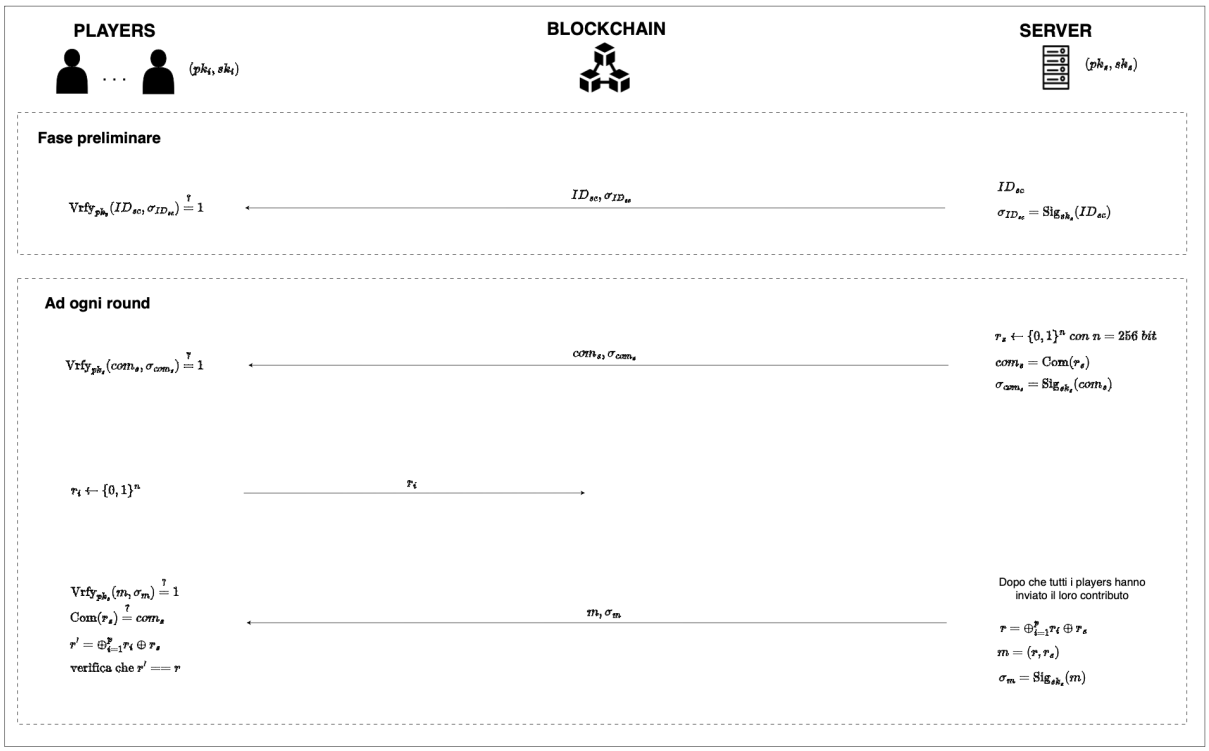

Figure 2.7: Protocollo associato alla seconda soluzione proposta

Procediamo con la descrizione step-by-step del protocollo introdotto:

1. inizialmente, il server crea lo *smart contract* e comunica a tutti i partecipanti al tavolo l'id ad esso associato ID_{sc}, con la rispettiva firma

   $$\sigma_{ID_{sc}} = Sig_{sk_s}(ID_{sc})$$

   che funge da garanzia per i giocatori del tavolo, i quali sono in questo modo certi che il server ha avviato la partita sullo smart contract avente tale ID.

   Tale informazione verrà comunicata solo una volta ad inizio round poiché lo stesso *smart contract* sarà utilizzato per i successivi round fino alla conclusione della partita.

2. a partire dal primo e per ogni round, il server genera il proprio contributo casuale r_s come

   $$r_s \leftarrow \{0,1\}^n, \text{ con n = 256 bit}$$

e tramite uno *schema di commitment con random oracle* calcola l'*impegno*

$$com_s = Com(r_s)$$

ed il commitment firmato, ovvero

$$\sigma_{com_s} = Sig_{sk_s}(com_s)$$

Il server invierà dunque, ad ogni player appartenente alla sala, la coppia di informazioni (com_s, σ_{com_s}), il quale, una volta ricevuta, ne verificherà la firma

$$Vrfy_{pk_s}(com_s, \sigma_{com_s}) \stackrel{?}{=} 1$$

3. a questo punto, ogni player i genera il proprio contributo casuale

$$r_i \leftarrow \{0,1\}^n, \text{ con n = 256 bit}$$

e lo invia tramite una transazione sulla blockchain, che sarà successivamente immagazzinato nello *smart contract* associato all'ID fornito dal server

4. nel momento in cui tutti i contributi dei players sono stati caricati, un evento generato dallo *smart contract* avvisa il server, il quale può quindi prelevarli e calcolare la stringa casuale complessiva come

$$r = \oplus_{i=1}^{p} r_i \oplus r_s$$

A questo punto, il server può rivelare il suo contributo r_s, il quale sarà inviato ai players insieme al valore della stringa casuale complessiva, attraverso il messaggio $m = (r, r_s)$, al quale sarà associata la rispettiva firma

$$\sigma_m = Sig_{sk_s}(m)$$

Infine, ogni player verifica innanzitutto la firma associata al messaggio m

$$Vrfy_{pk_s}(m, \sigma_m) \stackrel{?}{=} 1$$

verifica che il contributo r_s "apra" il commitment com_s

$$Com(r_s) \stackrel{?}{=} com_s$$

per poi calcolare la stringa casuale complessiva sulla base dei contributi associati agli altri players, noti in quanto presenti sulla blockchain

$$r' = \oplus_{i=1}^{p} r_i \oplus r_s$$

e verificando la corrispondenza tra quanto calcolato e quanto generato dal server

$$r' \stackrel{?}{=} r$$

2.3.4 Smart Contract

```solidity
pragma solidity >=0.7.0 <0.9.0;

// @title PlayersContributions.
contract PlayersContributions {

    // The number of players participating in the game.
    uint public numberOfPlayers;

    // The current round number.
    uint public roundNumber = 1;

    // This struct will represent a single contribution from a player.
    struct Contribution {
        string contribution;    // The string contributed by the player.
        bool contributed;       // If true, the player has contributed for the current round.
    }

    // This will associate a round number and a player's address with their contribution.
    mapping(uint => mapping(address => Contribution)) public contributions;

    // This will store the unique contributions for a round to prevent duplicate contributions.
    mapping(uint => mapping(contribution => bool)) uniqueContributions;

    // Event that will be emitted when all players have made their contributions.
    event AllPlayersContributed(uint round);

    // Create a new PlayersContributions with a specific number of players.
    constructor(uint _numberOfPlayers) {
        numberOfPlayers = _numberOfPlayers;
    }

    // This function will be called by players to contribute to the game.
    function contribute(string _contribution) external {
        // Ensure that the player has not already contributed for the current round.
        require(!contributions[roundNumber][msg.sender].contributed, "Player has already contributed for this round.");

        // Ensure that the contribution is unique for the current round.
        require(!uniqueContributions[roundNumber][_contribution], "Contribution must be unique.");

        // Record the player's contribution.
        contributions[roundNumber][msg.sender] = Contribution(_contribution, true);

        // Mark the contribution as used for the current round.
        uniqueContributions[roundNumber][_contribution] = true;

        // Increment the round number if all players have contributed.
        if (allPlayersHaveContributed()) {
            emit AllPlayersContributed(roundNumber);
            roundNumber++;
        }
    }

    // This function will be used to get the contributions of all players for a specific round.
    function getContributions(uint _roundNumber) external view returns (string[] memory) {
        string[] memory players_contributions = new string[](numberOfPlayers);
        uint index = 0;
        for (uint i = 0; i < numberOfPlayers; i++) {
            players_contributions[index] = contributions[_roundNumber][msg.sender].contribution;
            index++;
        }
        return players_contributions;
    }

    // This function will check if all players have contributed for the current round.
    function allPlayersHaveContributed() internal view returns (bool) {
        for (uint i = 0; i < numberOfPlayers; i++) {
            if (!contributions[roundNumber][msg.sender].contributed) {
                return false;
            }
        }
        return true;
    }
}
```

Figure 2.8: Smart contract

Lo *smart contract* descritto, denominato **PlayersContributions**, è progettato per gestire i contributi casuali dei players durante ciascun round di gioco. In particolare, mantiene il numero di giocatori presenti nella stanza virtuale, il numero del round corrente, e tiene traccia dei contributi di ciascun giocatore per ciascun round. Ciascun contributo è rappresentato da una struttura dati *Contribution*, che contiene il contributo casuale del giocatore e un indicatore booleano che segnala se il giocatore ha già contribuito per il round corrente.
Lo *smart contract* è inizializzato con il numero di giocatori partecipanti. I giocatori poi interagiscono con esso attraverso la funzione *contribute*, che accetta il contributo casuale del giocatore come argomento. Questa funzione verifica prima di tutto che il giocatore non abbia già inviato un contributo per il round corrente, e poi che il contributo non sia duplicato, ossia non sia già stato inviato da un altro giocatore nello stesso round. Una volta superate queste verifiche, il contributo del giocatore viene registrato, e se tutti i giocatori hanno inviato i loro contributi, viene emesso un evento *AllPlayersContributed* e il numero del round viene incrementato. Il server può quindi richiamare la funzione *getContributions* per ottenere tutti i contributi di un round specifico. Infine, la funzione *allPlayersHaveContributed* verifica se tutti i giocatori hanno inviato un contributo per il round corrente, iterando su su di essi e controllando l'indicatore *contributed* del loro oggetto *Contribution*.

2.4 Specifiche degli algoritmi utilizzati

2.4.1 Gen

L'algoritmo Gen(1^n) viene utilizzato dall'utente per la generazione della coppia di chiavi pubbliche e private. In particolare tramite questo algoritmo andiamo ad istanziare il problema del **logaritmo discreto** ottenendo le seguenti informazioni, $\mathbb{G}, q, g$ dove:

- $\mathbb{G}$ è un gruppo ciclico di ordine q
- q rappresenta il numero di elementi del gruppo
- g è un generatore del gruppo

A questo punto è possibile selezionare un certo valore $x \in \mathbb{Z}_q$ e calcolare $y = g^x$.

La **chiave pubblica**, indicata con pk è:

$$\mathbb{G}, q, g, y$$

La **chiave privata**, indicata con sk è:

$$\mathbb{G}, q, g, x$$

Si noti che la dimensione in termini di bit degli elementi dipende dal parametro di sicurezza 1^n.

2.4.2 $\mathcal{MT}$

Funzione che a partire da una serie di dati in chiaro restituisce l'intera struttura del Merkle Tree ad essi associato. Ogni *hash* che va a rappresentare i nodi dell'albero è ottenuto tramite l'applicazione della funzione *SHA-256*.

2.4.3 $\mathbf{Sign}_{sk}$

$\text{Sign}_{sk}(m)$ è un algoritmo di firma digitale, in particolare, facciamo riferimento allo **schema di firma di Schnorr**. Questo, a partire da un messaggio m ed una chiave privata sk restituisce una coppia (z, a) che è stata generalmente indicata con σ in cui:

- a rappresenta un contributo casuale del gruppo, tale che $a = g^r$, con $r \in \mathbb{Z}_q$
- $z = r + H(y||a||m)\, x$, si noti che $H(\cdot)$ è un *random oracle* ed è dunque implementato nella pratica con *SHA-256*

2.4.4 $\mathbf{Verify}_{pk}$

La funzione $\text{Verify}_{pk}(m, \sigma) : \{0, 1\}$ a partire da una firma $\sigma = (z, a)$ ed un messaggio m (quello che è stato firmato) verifica, sulla base della conoscenza della chiave pubblica del firmatario, che σ sia una firma valida per m, ciò avviene verificando che $g^z = ay^c$.

2.4.5 f

Con f si è indicato l'algoritmo che, a partire da una certa *Merkle Proof* $h_1 \dots h_j$ restituisce il valore genericamente indicato con h, ovvero la **root del Merkle Tree**.

Tale informazione rappresenta una traccia digitale dell'intero documento nonostante possa essere calcolata a partire da una quantità molto ridotta di informazioni in chiaro.

2.4.6 Check

Con *Check* si è indicato l'algoritmo che, in accordo alla *policy* del GDPR verifica la validità delle informazioni fornite dall'utente.

2.4.7 Commitment scheme

Lo *schema di commitment con random oracle* utilizzato è definito come segue: pur formalmente basato su un algoritmo randomizzato Gen che emette parametri pubblici $params$, dal momento in cui il modello adottato è basato su *random oracle*, questi ultimi non sono necessari poichè H funge da parametro pubblico dello schema.

Quest'ultimo prevede che l'"impegno" di un messaggio m venga calcolato come $com = H(m||r)$, con $r \leftarrow \{0,1\}^n$, tuttavia essendo nel contesto dei protocolli presentati i messaggi m già scelti randomicamente su $\{0,1\}^n$, con $n = 256$ bit, ciò giustifica il perché la concatenazione a tale messaggio m di un *salt* r non risulti necessaria in quanto la probabilità con cui un utente malintenzionato interroghi H su input m è trascurabile, motivo per cui viene calcolato come

$$com = H(m)$$

Infine, la verifica prevede che, "disimpegnato" m, venga controllato

$$H(m) \stackrel{?}{=} com$$

In particolare, la *collision resistant hash function* H utilizzata è *SHA-256*.

CHAPTER 3

WP3: ANALISI

Lo scopo del seguente capitolo è quello di analizzare la soluzione evidenziata all'interno del WP2 alla luce delle proprietà espresse in termini di **confidenzialità**, **integrità**, **trasparenza** ed **efficienza** presentate all'interno del WP1.

3.1 Confidenzialità

Di seguito viene eseguita un'analisi degli avversari che compromettono la confidenzialità del sistema:

- **Jack the Manipulator**: questo avversario potrebbe tentare di manipolare il contributo dei giocatori e del server durante il processo di generazione delle stringhe per trarne vantaggio poiché ha accesso ai canali di comunicazione. Il suo attacco non va a buon fine in quanto la connessione dei giocatori al server avviene tramite il protocollo TLS il quale è resiliente ad attacchi di tipo *human in the middle*
- **Frank the Rouge Server**: questo avversario attacca la confidenzialità delle informazioni presenti all'interno del Green Pass 2.0, essendo egli in grado di accedere ai dati presenti all'interno del server della Sala Bingo in fase di accesso di un utente ad una sala virtuale, Frank potrebbe collezionare i dati inviati dall'utente per venderli con scopi illeciti. Tali dati sono inviati sì su un canale

sicuro ma ad un server fraudolento; in particolare, questi saranno visibili in chiaro da Frank e possono dunque essere da lui collezionati

- **John the Fraudster**: questo avversario attacca il Ministero della Salute richiedendo sotto falsa identità un Green Pass 2.0, poiché egli è a conoscenza di tutte le informazioni della persona attaccata ed ha accesso al telefono cellulare di quest'ultima, essendo quindi in grado di richiedere un Green Pass 2.0 sotto falsa identità, portando a buon fine il suo attacco

- **Alice the Eavesdropper**: questo avversario è in ascolto sul canale di comunicazione nella speranza di ottenere informazioni utili che possano essere raccolte e/o vendute. Essendo i canali di comunicazione fra utente e Ministero e fra giocatori e servizi di Mister Joker protetti da un canale sicuro TLS, questo avversario non è in grado di raccogliere alcuna informazione utile

- **Eve the Impersonator**: questo avversario attacca sia il processo di generazione del Green Pass 2.0 sia l'accesso da parte di un utente ai servizi digitali offerti da Mister Joker tramite un attacco attivo di tipo *human in the middle* con l'obbiettivo di rubare un documento valido ed utilizzarlo per i propri scopi. Tuttavia, come ribadito per i precedenti avversari, la presenza di un canale TLS rende impossibile tale attacco. Anche nel caso in cui Eve fosse entrato in possesso per altre vie di tale documento digitale questo non riuscirebbe comunque ad utilizzarlo per i propri scopi non essendo in grado di dimostrare ad un *Verifier* di essere il possessore del documento

Per quanto concerne le proprietà espresse nel *WP1*:

- **C.1**: questa proprietà viene garantita in quanto, grazie alla struttura con cui sono rappresentati i dati del Green Pass 2.0, l'utente può fornire in chiaro solo alcune informazioni selezionate e dunque in maniera controllata ed essere certo che tali informazioni siano accessibili solo alla parte autorizzata a cui sono destinate

- **C.2**, **C.3**,**C.4**: queste proprietà vengono garantite grazie alla presenza di un canale di comunicazione basato su TLS sia fra l'utente ed il Ministero che fra l'utente e la sala Bingo.

- **C.5**: questa proprietà è garantita a patto che **l'utente non ceda volontariamente o venda** la chiave privata, associata alla chiave pubblica presente sul Green Pass 2.0. Tuttavia, in questi casi, i meccanismi digitali possono intervenire parzialmente poiché
  - se l'utente non cede volontariamente la propria chiave privata, può richiedere una revoca del Green Pass 2.0 in modo tale da associare a quest'ultimo una nuova chiave segreta
  - altrimenti se l'utente cede volontariamente questa informazione, non è possibile in alcun modo intervenire
- **C.6**: una volta che le informazioni richieste dal GDPR appartenenti al Green Pass 2.0 sono state inviate e corredate di prova, non è possibile per l'utente assicurarsi che queste non vengano collezionate. L'unica informazione che, a seguito dell'identificazione resta totalmente ignota anche ad un server malevolo è quella legata alla chiave segreta in possesso dell'utente in quanto questa è totalmente anonima. Quanto affermato è garantito dall'esistenza di una *zero knowledge proof* utilizzata nell'ambito dello *schema di identificazione di Schnorr* adottato per lo svolgimento della fase di identificazione
- **C.7**: questa proprietà è garantita in maniera parziale, in quanto il Ministero non è in grado di verificare l'identità del richiedente direttamente, essendo basata interamente sull'identificazione digitale. La prova dell'identità avviene attraverso uno scambio di informazioni, ritenuto ragionevolmente correlato alla richiesta di dati conosciuti solo dall'utente interessato. L'identificazione fisica presso l'hub vaccinale, che richiede anche l'indicazione del proprio numero di telefono per ricevere il codice OTP, presuppone che l'utente sia stato effettivamente vaccinato e abbia fornito il proprio numero di cellulare. Tuttavia, un utente malevolo, se in possesso dei dati personali dell'utente (come la tessera sanitaria) e del suo cellulare (per visualizzare il codice OTP rilasciato dal Ministero), potrebbe ottenere un documento originale associato a una chiave privata da lui scelta e farsi "scambiare" per quella persona dal Ministero, ricevendo così un documento originale e valido

Di seguito è riportato in forma tabellare un indice sintetico del grado di soddisfacibilità da parte del sistema delle diverse proprietà:

Proprietà	Soddisfacimento senza blockchain	Soddisfacimento con blockchain
C.1	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
C.2	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
C.3	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
C.4	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
C.5	⋆⋆	⋆⋆
C.6		
C.7	⋆ ⋆ ⋆	⋆ ⋆ ⋆

Table 3.1: Grado di soddisfacimento per le proprietà di confidenzialità. Si è stabilito che ogni proprietà può essere gradata con al più 4⋆.

3.2 Integrità

Di seguito viene eseguita un'analisi degli avversari che compromettono l'integrità del sistema:

- **Bob the Trickster**: l'unico modo che ha questo avversario di minare l'integrità del processo di generazione delle stringhe casuali è quello di poter accedere ai contributi di tutti gli utenti e del server e costruire appositamente il suo per ottenere un vantaggio. Tuttavia, Bob non ha accesso ai canali di comunicazione tra i players e il server e, di conseguenza, anche non inviando un contributo casuale non riuscirà ad ottenere un vantaggio dal momento in cui il suo contributo verrà messo in $\oplus$ con tutti gli altri contributi casuali dei giocatori onesti. Pertanto, se l'utente non ha controllo sugli input casuali, come nel caso in esame, allora non ha controllo sull'output

- **The Trickster's Guild**: questo gruppo di avversari può provare a minare l'integrità del processo di generazione delle stringhe casuali costruendo contributi strutturati per ottenere un vantaggio rispetto alla corrispettiva parte di players onesti. Nel caso peggiore, tutti i partecipanti alla partita appartengono a questa categoria e sono tutti in combutta contro il server (che in tale scenario si suppone onesto in quanto una partita di soli avversari non rappresenta un caso di interesse). Così come il singolo avversario Bob, tuttavia, si suppone non abbiano accesso ai canali di comunicazione e non sono in grado di prevedere il contributo o

i contributi casuali delle parti oneste. Poiché il loro contributo costruito in maniera malevola andrà in $\oplus$ con quello delle parti oneste, non saranno in grado di prendere in alcun modo il controllo dell'output.
Nello scenario in cui tutti i partecipanti alla partita sono disonesti, la garanzia che questi non saranno in alcun modo in grado di influenzare l'esito della partita arriva dal contributo casuale del server. Sebbene questo invii per primo ai player una prova del proprio impegno, essendo questa un commitment del proprio contributo, i giocatori non saranno in grado di violarlo e ciò garantisce che vi sia almeno un contributo casuale che prende parte al processo di generazione, rendendo vano l'attacco

- **The Missing Max**: questo tipo di avversario potrebbe minare l'integrità del sistema decidendo volontariamente di non partecipare al processo di generazione delle stringhe casuali non inviando il proprio contributo. Questo attacco, sebbene non faccia terminare la partita richiede che il server, scaduto il timeout relativo a tale player interrompa la connessione con quest'ultimo. Per tale motivo, l'attaccante è scoraggiato dall'esibire questo comportamento e, di conseguenza, la possibilità di un tale attacco è pressoché nulla

- **Carol the Mole**: questo avversario ha pieno controllo sullo svolgimento della partita in quanto l'architettura proposta è di tipo client-server e quindi tutti i partecipanti inviano e ricevono messaggi dal server. Essendo quest'ultimo l'unico a ricevere il contributo di tutti i giocatori, al tempo stesso è anche il primo a poter conoscere la stringa casuale estratta. Poiché anche il server partecipa con un proprio contributo alla generazione, questo potrebbe scegliere volontariamente un valore calcolato ad arte per influenzare il processo di generazione. Tuttavia il protocollo prevede che, come "prova di onestà", il server invii un impegno del proprio contributo ai giocatori in apertura della partita rassicurandoli sul fatto che non potrà in alcun modo barare al momento dell'estrazione della stringa in quanto ci sarà un solo contributo verificabile in grado di aprire il commitment.
  Una ulteriore situazione in cui il server potrebbe agire in maniera malevola consiste nel tentare di mentire a tutte le parti oneste istanziando una "versione diversa" della partita con ognuno di loro. In tale ambiente simulato, ogni player vedrebbe una versione del protocollo eseguita in maniera corretta, tuttavia, il

server avrebbe modo di agire in quanto potrebbe manipolare il contributo di tutti i partecipanti ad eccezione di quello onesto. Tale tentativo di attacco fallirebbe in quanto in un primo momento, i giocatori inviano un commitment del proprio contributo al server e si aspettano che questo lo inoltri in broadcast. Se il server volesse mentire ad un giocatore, dovrebbe costruire $p - 1$ stringhe ad hoc sulla base di quella fornita dal player attaccato per influenzare la stringa finale estratta e calcolarne il commitment. Tuttavia, questa strategia non è in grado di compromettere l'impegno di nessun giocatore in quanto "inviolabile" e, di conseguenza, non può essere attuata con successo.
Un ulteriore comportamento disonesto del server potrebbe essere il seguente: una volta estratta una stringa, prima di inviarla alla sala, accorgendosi che questa risulta essere sfavorevole, il server potrebbe interrompere il protocollo rifiutandosi di inviarla ai giocatori ed accusando uno di questi di non aver rivelato il proprio contributo entro lo scadere del timeout, continuando a giocare il round corrente senza il player accusato. Nella versione del protocollo priva di blockchain non vi è modo di contrastare questa situazione in quanto il player accusato non ha alcuna evidenza digitale dell'effettivo invio. Un commento a margine circa questo possibile attacco è che un comportamento ripetuto di questo tipo potrebbe allontanare i giocatori onesti dalle sale di Mister Joker; un danno di questo tipo potrebbe rappresentare un deterrente a comportamenti di questo genere in quanto non farebbe gli interessi del server disonesto. Tuttavia, tale escamotage potrebbe essere utilizzato per evitare siano ottenute grosse vincite in denaro da parte di giocatori onesti. La seconda soluzione proposta invece, che adotta la blockchain, risolve questo problema, dal momento in cui i giocatori non inviano i loro contributi direttamente al server, ma ad uno smart contract inizializzato all'inizio della partita, che avviserà il server nel momento in cui tutti i players avranno inviato il loro contributo. In questo modo, il server non potrà più mentire ai giocatori dal momento in cui tutti possono verificare le informazioni del round in corso e non.

- **Cospiracy Conclave**: questo raggruppamento di avversari è sicuramente uno dei più forti che è possibile analizzare nel contesto della generazione delle stringhe casuali. Tale gruppo prevede infatti la partecipazione di un certo numero di players disonesti (avversario **The Trickster's Guild**), che nel caso peggiore

possono essere tutti i partecipanti al tavolo ad eccezione di un player onesto, e l'avversario **Carol the Mole** che cooperando con i players disonesti vuole portare il gruppo alla vittoria per poi dividere la vincita. In questo scenario, un possibile attacco potrebbe essere il seguente. Supponiamo che al tavolo vi siano 4 players (A, B, C, D), di cui uno onesto (D) e tre disonesti che si sono accordati con il server nel seguente modo:

- A e B non dovranno inviare subito il loro contributo, ma attendere che il server gli invii la stringa da replicare - ad esempio, A riceverà il contributo del server e B riceverà il contributo del player onesto, in maniera tale da annullarli
- C, invece, dovrà inviare il contributo che in quel momento lo porterebbe in vantaggio, fino alla vittoria

In questa maniera, la stringa finale calcolata, data dallo $\oplus$ di tutti i contributi, coinciderà proprio con quella inviata da C, dal momento in cui quella di A si annullerà con quella del server e quella di B si annullerà con quella di D. Ora, l'esecuzione di questo avversario, nel protocollo proposto, fallirebbe, dal momento in cui ogni player onesto, quando riceve i commitment dal server di tutti gli altri players, verifica che non ve ne siano due uguali, caso in cui capirebbe che vi è un imbroglio in corso e uscirebbe dalla partita, lasciando al tavolo i soli players disonesti

- **Jack the Manipulator**: poiché TLS garantisce un *tunnel sicuro* all'interno della rete, Jack non è in grado di minare l'integrità del sistema di generazione delle stringhe causali non potendo in alcun modo influenzare i messaggi inviati sul canale
- **Simon the Forger**: questo avversario è sconfitto a priori in quanto non vi è alcun modo per un avversario di ottenere una firma valida senza conoscere la chiave segreta del ministero (che si considera in questo scenario parte onesta)
- **Eve the impersonator**: questo avversario è sconfitto in quanto, a fronte della richiesta di una *Zero Knowledge Proof* per il Green Pass 2.0 mostrato, non è in

grado di completare il protocollo non essendo a conoscenza della chiave privata generata inisme alla chiave pubblica presente all'interno del documento rubato

Per quanto concerne le proprietà espresse nel *WP1*:

- **I.1**: questa proprietà è garantita dal fatto che l'utente si sia connesso alla sala Bingo aprendo un canale di comunicazione sicuro cifrato tramite TLS , nell'ambito della partita dunque il suo contributo non può essere letto o modificato da utenti malevoli in ascolto sul canale. Tuttavia, anche nel momento in cui il server accede ai contributi prodotti dagli utenti non ha modo di manipolarli, dal momento in cui ogni utente riesce a verificare come quanto comunicato risulti differente da quanto incluso nel processo di generazione da parte del server

- **I.2**: questa proprietà è garantita dalla presenza di un'unica firma digitale apposta dal Ministero della Salute sfruttando una specifica chiave pubblica, pubblicamente nota. Si è inoltre supposto che il ministero (che è una parte fidata) abbia rilasciato pubblicamente la procedura con cui il Green Pass 2.0 viene costruito e come viene firmato

- **I.3**: questa proprietà è sempre verificata, ciò avviene perché all'utente è sempre richiesto di corredare con una Merkle Proof ogni informazione inviata, tale prova è fondamentale per il calcolo di quella che è la radice dell'albero (h). Poiché la radice del Merkle Tree è stata firmata in fase di generazione del documento dal Ministero, verificando la firma di quest'ultimo sulla base di h, è possibile essere certi che:

  - l'informazione inviata è valida
  - l'informazione inviata appartiene a quel Green Pass
  - l'informazione inviata appartiene all'utente identificato in precedenza, in quanto la h calcolata durante la fase di identificazione deve coincidere con quella ottenuta a partire dai ogni nuovo dato inviato

- **I.4**: l'impredicibilità della stringa estratta getta le radici nella scelta dell'operazione di XOR per unire i contributi dei diversi partecipanti

- **I.5**: questa proprietà viene garantita dalla presenza di un contributo fornito dall'utente in fase di rilascio del Green Pass 2.0, ovvero la chiave pubblica da lui

generata. Nel caso in cui il documento fosse stato rubato in qualche modo da un malintenzionato, sarebbe verificabile che il documento sia valido ma che non appartenga a quest'ultimo poiché a fronte di una *Zero Knowledge Proof* non sarebbe in grado di provare ad un Verifier che è a conoscenza dell'informazione segreta legata alla chiave pubblica presente all'interno del documento

- **I.6**: questa proprietà è garantita in parte in quanto, un player può comunque decidere di non inviare alcun contributo, tuttavia allo scadere del timeout legato a tale giocatore, il server (se onesto), lo espellerà dalla partita, proseguendo il round senza il player accusato

- **I.7**: questa proprietà, nella soluzione senza blockchain, non viene garantita in quanto i partecipanti si affidano al server come unico intermediario per comunicare tra di loro e ottenere le informazioni relative allo svolgimento della partita. Tale dipendenza dal server limita la capacità dei partecipanti di accertarsi reciprocamente della validità delle informazioni fornite da quest'ultimo, impedendo in alcune situazioni di avere evidenze digitali di comportamenti scorretti. Nel caso in cui il server dovesse ad esempio affermare che il timer associato al contributo di un giocatore sia scaduto, questo non ha alcuna prova per dimostrare di aver partecipato, cosa che invece può essere effettuata nella soluzione con blockchain.

Come avvenuto per il caso precedente è di seguito riportato in forma tabellare un indice sintetico del grado di soddisfacibilità da parte del sistema delle diverse proprietà:

Proprietà	**Soddisfacimento senza blockchain**	**Soddisfacimento con blockchain**
I.1	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
I.2	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
I.3	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
I.4	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
I.5	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
I.6	⋆ ⋆ ⋆	⋆ ⋆ ⋆
I.7	⋆	⋆ ⋆ ⋆⋆

Table 3.2: percentuale di soddisfacimento per le proprietà di integrità, si è stabilito che ogni proprietà può essere gradata con al più 4 ⋆

3.3 Trasparenza

Il sistema, per come è stato progettato (con particolare riferimento alla soluzione con blockchain) è fortemente orientato alla trasparenza. Il motivo risiede nel fatto che ogni player può facilmente verificare i contributi di tutti gli altri giocatori e che l'intero processo di generazione della stringa casuale sia corretto, potendo, a fine round, calcolare la stringa estratta e verificare che questa corrisponda effettivamente a quella inviata dal server. In particolar modo, si ha ancora più trasparenza dal momento in cui sono i giocatori stessi a inviare i loro contributi allo smart contract e quindi il server (disonesto) non potrà accusare i player senza motivo, come discusso in precedenza, in quanto non assume più il ruolo di interlocutore tra i partecipanti, perdendo il ruolo centrale ricoperto nella versione precedente del protocollo.
Per quanto concerne le proprietà espresse nel *WP1*:

- **T.1**: questa proprietà viene garantita dal momento in cui il protocollo proposto non fa affidamento a terze parti o strumenti poco chiari, il calcolo effettuato dal server può essere ripetuto da tutti i giocatori che, nel corso del protocollo, raccolgono tutte le informazioni necessarie alla verifica della stringa estratta
- **T.2**: il processo di generazione e verifica del Green Pass 2.0 viene reso pubblicamente noto a tutti dal Ministero, garantendo di fatto questa proprietà.
- **T.3**: questa proprietà viene garantita grazie all'impiego delle firme digitali, schemi di commitment e verifiche incrociate facendo sì che il sistema garantisca imparzialità nel processo di generazione poiché:
  - l'uso dei commitment, sia lato server all'inizio di ogni round che lato utente in fase di generazione dei propri contributi, assicura che ogni giocatore sia vincolato al proprio impegno e non possa cambiarlo successivamente
  - l'uso delle firme digitali, sia lato server che lato utente per ogni comunicazione effettuata, garantisce che tutti i partecipanti siano in grado di verificare l'autenticità delle informazioni scambiate
  - la verifica delle firme ad ogni fase, la rilevazione di duplicati quando vengono comunicati i commitment e la loro apertura assicurano che nessun giocatore possa manipolare i propri numeri o duplicare i commitment di altri giocatori

Come avvenuto per il caso precedente è di seguito riportato in forma tabellare un indice sintetico del grado di soddisfacibilità da parte del sistema delle diverse proprietà:

Proprietà	Soddisfacimento senza blockchain	Soddisfacimento con blockchain
T.1	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
T.2	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
T.3	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆

Table 3.3: percentuale di soddisfacimento per le proprietà di trasparenza, si è stabilito che ogni proprietà può essere gradata con al più 4 ⋆

3.4 Efficienza

Per quanto concerne le proprietà espresse nel *WP1*:

- **E.1**: il protocollo proposto per la generazione di stringhe casuali permette tramite lo scambio di 5 messaggi fra ogni utente ed il server di generare una stringa casuale il cui valore dipende dai contributi di tutti i partecipanti alla comunicazione. Sebbene il numero di messaggi sia contenuto, si noti che gran parte di questi devono essere firmati e la verifica della correttezza degli scambi può richiedere fino a p verifiche di firme digitali. Nel caso in cui bisogni accertarsi che non vi siano duplicati, sono richieste misure adeguate per il contenimento della complessità computazionale dell'operazione, come la scelta di strutture dati adeguate. Con particolare riferimento alla soluzione che vede l'introduzione della blockchain, questa viene utilizzata il minimo possibile, al fine di garantire il rispetto di alcune proprietà, senza la quale non sarebbero ottemperate. Trovandoci, inoltre, in un contesto permissioned, le transazioni sono elaborate più velocemente rispetto a una blockchain permissionless

- **E.2**: dato un Green Pass 2.0 la sua verifica richiede il controllo di una sola firma ed è dunque molto efficiente. Nel caso in cui bisogni verificare il documento a partire da uno dei dati ad esso appartenente corredato di *Merkle Proof*, è possibile portare a termine tale operazione in un numero di *hash* $O(\log n)$ con l'aggiunta della verifica di una sola firma digitale come nel caso precedente

- **E.3**: l'identificazione di un utente presso un qualsiasi fornitore di servizi digitali richiede che questo condivida dapprima la propria chiave pubblica con il server fornitore del servizio (che è parte del Green Pass 2.0 e deve dunque essere corredata di *Merkle Proof*), la cui verifica di validità ricade nell'analisi effettuata discutendo la proprietà **E.2**. Il processo di identificazione vero e proprio può essere portato a termine tramite lo scambio di 3 messaggi con il server e qualche esponenziazione modulare. Essendo inoltre un procedimento automatizzabile che non richiede l'intervento dell'utente, tale scambio di informazioni può essere totalmente nascosto, rendendo di fatto semplice a quest'ultimo l'accesso a qualsiasi servizio tramite Green Pass 2.0

Come avvenuto per il caso precedente è di seguito riportato in forma tabellare un indice sintetico del grado di soddisfacibilità da parte del sistema delle diverse proprietà:

Proprietà	**Soddisfacimento senza blockchain**	**Soddisfacimento con blockchain**
E.1	⋆ ⋆ ⋆	⋆ ⋆
E.2	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆
E.3	⋆ ⋆ ⋆	⋆ ⋆ ⋆

Table 3.4: percentuale di soddisfacimento per le proprietà di efficienza. Si è stabilito che ogni proprietà può essere gradata con al più 4 ⋆

3.5 Considerazioni finali

Vengono di seguito riportati i grafici radar dove è sinteticamente rappresentato il sistema progettato, con e senza blockchain rispettivamente. La realizzazione di tali grafici è stata fatta tenendo in considerazione i gradi di soddisfacimento delle proprietà del sistema, usando le tabelle sovrastanti.

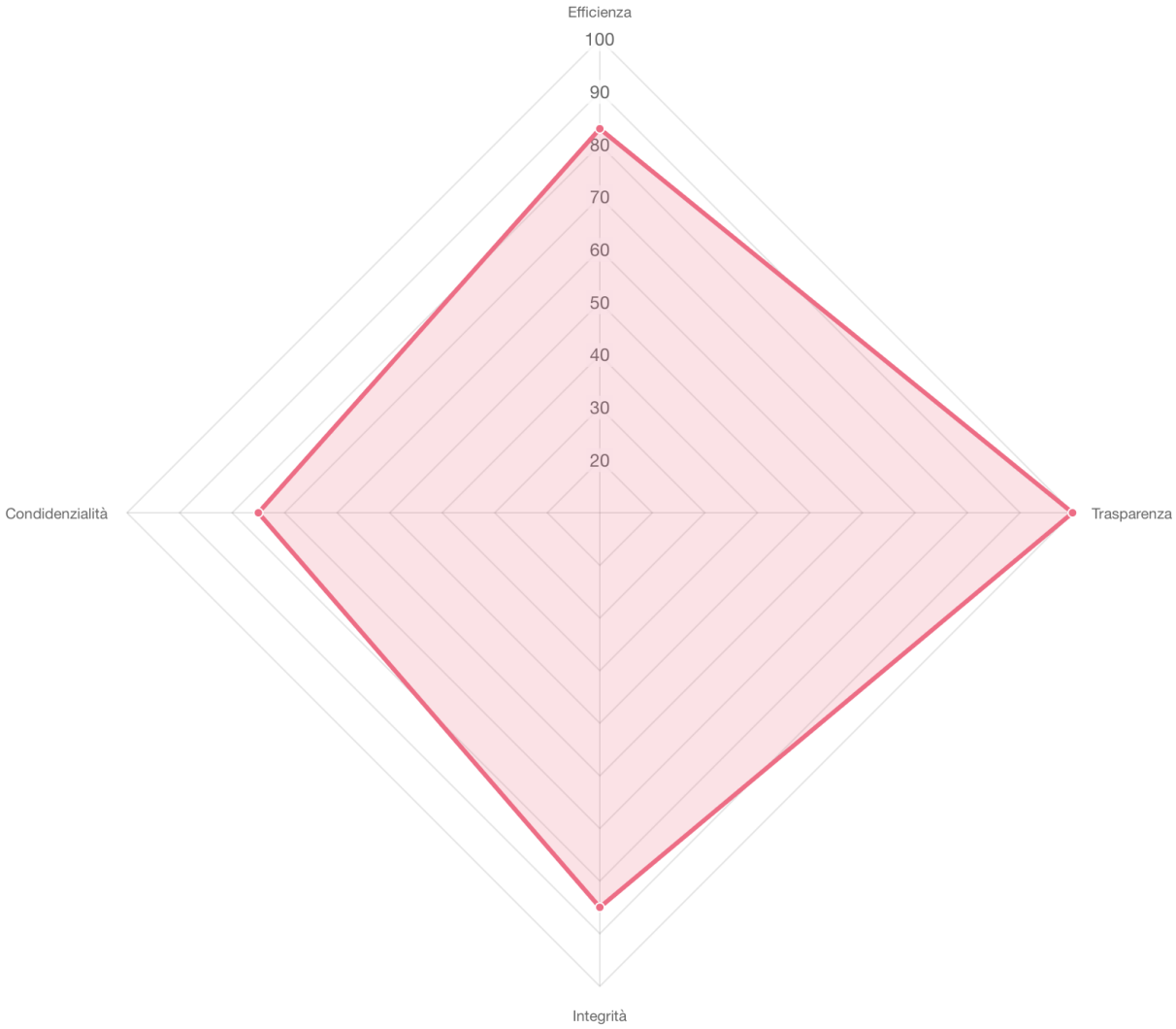

Figure 3.1: Grafico a radar rappresentante le prestazione del sistema considerando il protocollo per la generazione di stringhe casuali senza la blockchain

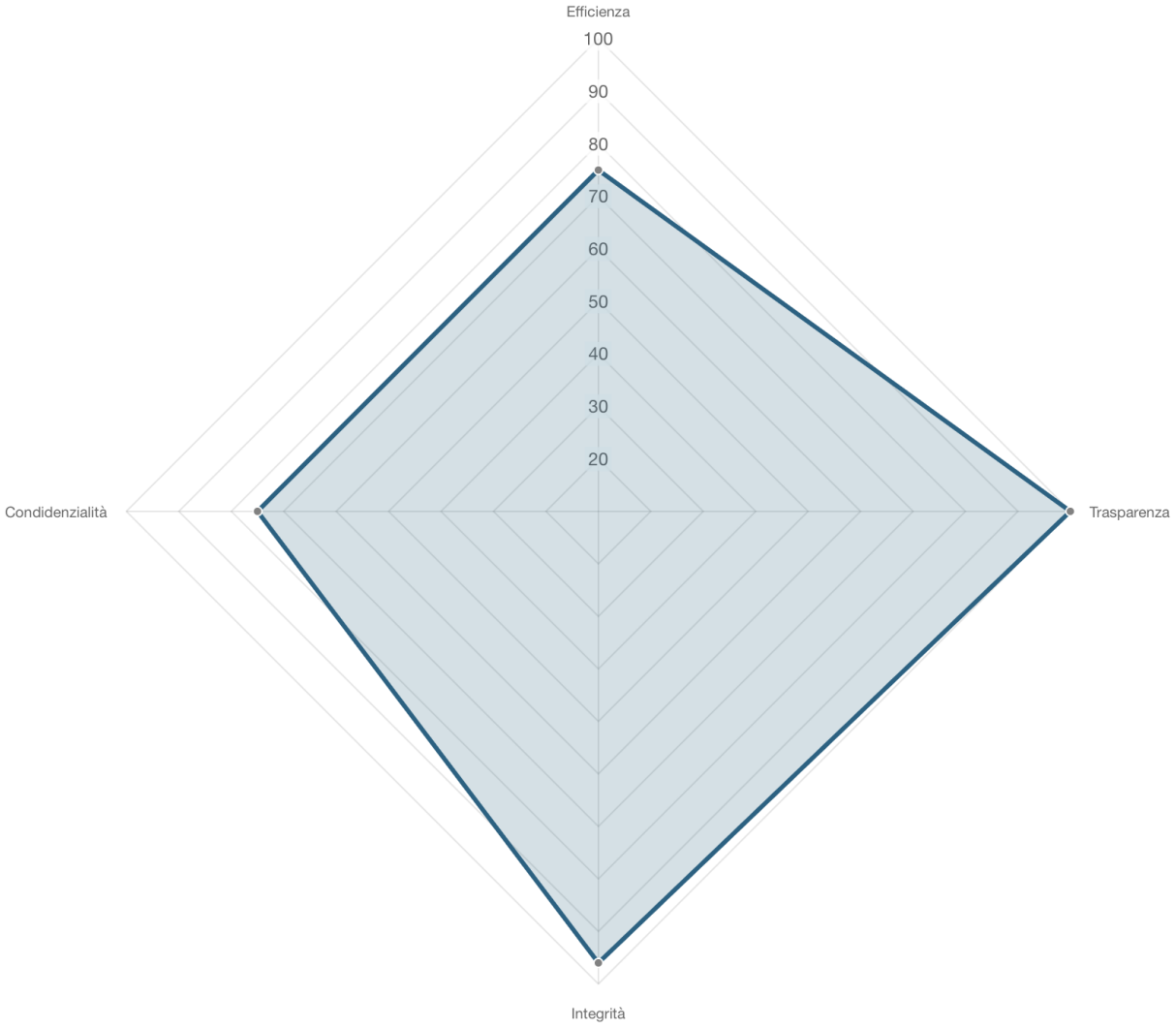

Figure 3.2: Grafico a radar rappresentante le prestazione del sistema considerando il protocollo per la generazione di stringhe casuali con la blockchain

CHAPTER 4

WP4: IMPLEMENTAZIONE E PRESTAZIONI

L'implementazione pratica di quanto discusso nel Capitolo 2 è stata realizzata interamente in Python. Per la manipolazione di stringhe e/o strutture dati, è stato utilizzato il linguaggio Python, mentre per le operazioni di generazione di stringhe casuali e l'utilizzo di protocolli crittografici noti, si è fatto uso delle chiamate alla libreria OpenSSL.

4.1 Elementi principali

Nel seguito, presenteremo gli elementi principali del software evidenziando in che modo essi interagiscono.

4.1.1 Approccio base

Per realizzare le comunicazioni tra utenti e i diversi server coinvolti nei protocolli proposti, abbiamo scelto di utilizzare una struttura basata su oggetti. In questo contesto, le parti coinvolte sono rappresentate mediante istanze di oggetti di classi specifiche. Tramite la chiamata dei metodi messi a disposizione dalle interfacce degli oggetti coinvolti nelle comunicazioni, si è simulato uno scambio di messaggi, emulando

così l'interazione tra due applicativi in esecuzione su macchine diverse e connesse tramite la rete internet. A tal fine, abbiamo sviluppato due classi principali:

- *User*: questa classe rappresenta gli utenti che desiderano collegarsi ad un server. Ogni utente possiede e gestisce tutte le informazioni necessarie per stabilire una connessione sicura con il server. In particolare, una classe derivata è *Player* che presenta tutta una serie di metodi utili al fine di partecipare all'estrazione delle stringhe casuali

- *Server*: questa classe costituisce una base per le classi derivate *ServerMinistero* e *ServerMJ*. La classe *ServerMinistero* modella il comportamento del server del Ministero, mentre *ServerMJ* gestisce le operazioni relative alle sale virtuali di Mister Joker

4.1.2 GreenPass

La classe *GreenPass* rappresenta e implementa il Green Pass a partire dalla descrizione fornita nei capitoli precedenti. I dati forniti dall'utente sono organizzati all'interno di una apposita struttura dati realizzata tramite la classe *Merkle Tree*. L'interfaccia di un Green Pass 2.0 è composta da:

- **Costruttore**: il costruttore della classe prende in input un dizionario *data* contenente le informazioni da inserire nel Merkle Tree.

- **Accesso ai dati**: la classe fornisce un metodo che consente di ottenere i dati associati a un determinato ID all'interno del Merkle Tree. Inoltre, offre un metodo che restituisce la prova di autenticità per i dati specificati che consiste in una lista di elementi che consentono di verificare che i dati facciano effettivamente parte del Merkle Tree.

- **Recupero della firma**: la classe contiene un attributo che rappresenta la firma del Green Pass. Ciò consente di garantire l'autenticità e verificare se è stato emesso da una fonte attendibile, quale il Ministero della Salute.

Rappresenta quindi la coppia di informazioni che nel Capitolo 2 è stata indicata come $(\mathrm{GP}, \sigma_{\mathrm{GP}})$

4.1.3 Merkle Tree

Per la realizzazione del *Green Pass 2.0* ed in particolare per la realizzazione di quello che, nel Capitolo 2 è stato definito come *algoritmo* $\mathcal{MT}$ è stato necessario definire la classe *MerkleTree*, che implementa un albero binario gestito in accordo con le specifiche di un Merkle Tree. In particolare, questa classe presenta le seguenti caratteristiche:

- **Foglie**: rappresentano i diversi dati memorizzati, e ad ogni foglia è associato un hash ottenuto applicando la funzione crittografica SHA256
- **Radice (Root)**: rappresenta l'intero Merkle Tree, ossia l'hash della radice dell'albero, ottenuto combinando gli hash delle foglie e dei nodi intermedi secondo un algoritmo di concatenazione sicuro
- **Nodi interni**: rappresentano gli hash intermedi, calcolati combinando gli hash delle loro foglie figlie

Tramite le relazioni fra i diversi nodi dell'albero è stata resa possibile l'implementazione di una serie di metodi tramite i quali è possibile:

- costruire a partire da una lista di dati la struttura del *Merkle Tree*, come specificato nel Capitolo 2 tramite l'algoritmo $\mathcal{MT}$
- ricevere per ogni dato una lista di hash (*Merkle Proof*) tramite cui è possibile risalire alla radice dell'albero
- ricavare il valore della radice dell'albero h a partire da una generica *Merkle Proof*, indicata con f all'interno del Capitolo 2

4.1.4 Libreria Python Cryptography

Per lavorare in modo pratico e veloce con i certificati e/o le chiavi pubbliche e private dei server e degli utenti, abbiamo fatto uso della libreria *cryptography*. Questa libreria mette a disposizione una serie di metodi specificamente pensati per gestire le primitive crittografiche, il che ci ha permesso di utilizzare, ad esempio, i parametri su cui sono stati definiti i gruppi crittografici, nonché il generatore associato. La scelta di utilizzare *cryptography* ci ha permesso di concentrarci sulla progettazione e l'implementazione dei

protocolli di sicurezza, senza dover affrontare complessità aggiuntive riguardanti la gestione delle primitive crittografiche.

KeyManagement

Unendo le primitive di *Cryptography* ai comandi *OpenSSL*, abbiamo creato questa classe per la gestione delle chiavi e delle operazioni che è possibile fare con esse.
Si noti che la libreria *Cryptography* implementa già molti dei metodi qui presentati, tuttavia non lo fa utilizzando OpenSSL, strumento richiesto ai fini del progetto.
Andiamo ora a descrivere brevemente quali sono le operazioni che possiamo eseguire tramite l'utilizzo di questa classe:

- **Generazione delle chiavi**: la classe permette di generare una coppia di chiavi crittografiche ECDSA (chiave privata e chiave pubblica) per un utente. Le chiavi vengono generate utilizzando la curva ellittica "prime256v1" e salvate in file con estensione ".pem" nella directory specificata
- **Recupero della chiave pubblica**: la classe fornisce un metodo per recuperare la chiave pubblica associata a una chiave privata. Questo è utile, ad esempio, per inviare la chiave pubblica agli altri utenti o al server per l'autenticazione
- **Firma di messaggi**: gli utenti possono firmare i messaggi utilizzando la loro chiave privata. La firma del messaggio viene salvata in un file ".bin"
- **Verifica della firma**: la classe fornisce un metodo per verificare la firma di un messaggio. La firma e il messaggio vengono verificati tramite OpenSSL e il risultato è restituito come valore booleano

TLS

Presentiamo ora una serie di classi che sono state utilizzate per la realizzazione del protocollo TLS.

TLSClientHandler La classe *TLSClientHandler* gestisce la fase di handshake TLS (Transport Layer Security) con il server. La sua funzione principale è stabilire una connessione sicura tra l'utente e il server. Le caratteristiche principali della classe TLSClientHandler includono:

- **Gestione del protocollo TLS**: la classe implementa le tre fasi principali del protocollo TLS per stabilire una connessione sicura tra l'utente e il server
- **Cifratura e autenticazione**: vengono utilizzati algoritmi di cifratura simmetrica (AES-256-CTR) per proteggere i messaggi durante la trasmissione e HMAC-SHA256 per autenticare i messaggi ricevuti

TLSServerHandler La classe *TLSServerHandler* gestisce il lato server della fase di handshake TLS (Transport Layer Security). Questa classe è responsabile di stabilire una connessione sicura con l'utente (client). Le caratteristiche principali della classe TLSServerHandler includono:

- **Gestione del protocollo TLS**: la classe implementa le tre fasi del protocollo TLS per consentire una connessione sicura tra il server e l'utente
- **Diffie-Hellman key exchange**: utilizza l'algoritmo di scambio delle chiavi Diffie-Hellman per generare una chiave di sessione con l'utente, permettendo una comunicazione segreta
- **Cifratura e autenticazione**: utilizza AES-256-CTR per cifrare i messaggi durante la trasmissione e HMAC-SHA256 per autenticare i messaggi ricevuti
- **Gestione delle chiavi**: conserva una struttura dati (dictionary) *connections* per memorizzare le chiavi di sessione associate a ciascun utente, consentendo una comunicazione sicura e separata tra il server e gli utenti

4.1.5 Blockchain

La classe *Blockchain* rappresenta la nostra astrazione di una *permissioned blockchain*. Tale oggetto è in grado di memorizzare gli smart contracts in un dizionario utilizzando un ID univoco. La blockchain è implementata come un singleton, il che significa che può essere istanziata una sola volta e la stessa istanza è condivisa da tutte le altre parti del codice. La classe offre metodi per aggiungere nuovi smart contract alla blockchain e per ottenere uno specifico smart contract dato il suo ID.

ContributionGame

La classe *ContributionGame* rappresenta un'implementazione dello smart contract così come presentato nel Capitolo 2. In particolare, ogni player, dopo aver ottenuto lo smart contract dal server che avvia la partita, utilizza uno specifico metodo dello smart contract stesso per inviare il proprio contributo alla generazione della stringa casuale finale, che viene quindi salvato sulla blockchain. Successivamente, dopo che tutti i players hanno inviato il loro contributo, lo smart contract avviserà il server attraverso un apposito evento, momento in cui quest'ultimo recupererà i contributi di tutti i players dalla blockchain e li userà, insieme a quello da lui generato, per calcolare la stringa casuale finale che sarà infine inviata a tutti i players insieme a delle informazioni aggiuntive per consentire loro la verifica dell'intero processo in modo trasparente.

4.2 Esempio di esecuzione

Al fine di poter simulare l'intero flusso di esecuzione, sono stati inseriti all'interno della classe *Database* i dati di utenti fittizi utilizzando un *dizionario*, dove la chiave rappresenta le ultime 8 cifre della tessera sanitaria e il valore è a sua volta un *dizionario* che mantiene tutte le informazioni associate allo specifico utente per la creazione del Green Pass. Ciò presuppone, ovviamente, che gli utenti si siano recati in precedenza presso un hub vaccinale e che il Ministero della Salute, a seguito della somministrazione del vaccino, abbia inserito nel database tali informazioni.
Inoltre, per quanto concerne invece il server del Ministero della Salute e quello della Sala Bingo di Mister Joker, questi sono stati precedentemente creati e sono rappresentati dalle cartelle *Server_193.205.162.73* e *Server_214.23.65.7* rispettivamente. All'interno di ogni cartella troviamo, in particolare, i file *cert.pem*, *ecdsa_key.pem* e *ecdsa_pub.pem* che rappresentano il certificato digitale, la chiave privata e la chiave pubblica del server. In particolare, i certificati digitali dei due server sono stati rilasciati da una CA root da noi definita, il cui certificato è presente all'interno della cartella *CA*.
In aggiunta, è presente un'ulteriore cartella, chiamata *PublicInfo*, che, come indica il nome, contiene tutta una serie di informazioni pubblicamente note:

- *public_key_MS.pem*, rappresenta la chiave pubblica del Ministero della Salute
- *revokedGP.txt*, rappresenta la lista degli *Unique certificate identifier* revocati

- *policyGDPR.txt*, rappresenta l'insieme delle policy richieste dal GDPR in uno specifico momento

All'avvio della partita viene anche creato un ulteriore file, chiamato *contribution_info.txt*, che vuole in qualche modo rappresentare la storia delle informazioni di ciascun round presenti sulla blockchain.
Di seguito si riporta un possibile flusso di esecuzione considerando 2 players e 5 round.

```
#####################################################

#                   WELCOME                         #

#####################################################

How many users do you want to simulate? 2
Insert the IP address of the user: 192.168.1.1
Insert the IP address of the user: 192.168.1.2

#####################################################

#               Green Pass request                  #

#####################################################

## User 192.168.1.1 is requesting the Green Pass... ##

[User 192.168.1.1]:  connecting to Ministero della Salute...
[Server Ministero della Salute]:  new connection from 192.168.1.1

[User 192.168.1.1]:  starting TLS handshake...
[User 192.168.1.1]:  generating DH parameters...
[User 192.168.1.1]:  sending DH contribution...
[Server Ministero della Salute]:  sending DH contribution...
[Server Ministero della Salute]:  generating DH key...
[Server Ministero della Salute]:  creating TLS keys...
[User 192.168.1.1]:  generating DH key...
[User 192.168.1.1]:  creating TLS keys...
[Server Ministero della Salute]:  sending public key and certificate...
[User 192.168.1.1]:  verifing certificate...
[User 192.168.1.1]:  certificate verified!
[User 192.168.1.1]:  TLS handshake completed

Insert the last 8 digits of your HealthCard:
Insert the HealthCard expiration date (format YYYY-mm-dd):
[User 192.168.1.1]:  sending sanity information...
[Server Ministero della Salute]:  sanity info received
[Server Ministero della Salute]:  checking info...
[Server Ministero della Salute]:  user info correct
[Server Ministero della Salute]:  sending OTP...
[User 192.168.1.1]:  OTP received ed421d

Insert the OTP received on your phone:
[User 192.168.1.1]:  sending OTP and public key...
[Server Ministero della Salute]:  OTP and public key received
[Server Ministero della Salute]:  OTP correctly verified
[Server Ministero della Salute]:  sending green pass...
[User 192.168.1.1]:  green pass received
[User 192.168.1.1]:  closing connection...
[Server Ministero della Salute]:  closing connection with 192.168.1.1...
[Server Ministero della Salute]:  connection closed
```

```
## User 192.168.1.2 is requesting the Green Pass... ##

[User 192.168.1.2]:  connecting to Ministero della Salute...
[Server Ministero della Salute]:  new connection from 192.168.1.2

[User 192.168.1.2]:  starting TLS handshake...
[User 192.168.1.2]:  generating DH parameters...
[User 192.168.1.2]:  sending DH contribution...
[Server Ministero della Salute]:  sending DH contribution...
[Server Ministero della Salute]:  generating DH key...
[Server Ministero della Salute]:  creating TLS keys...
[User 192.168.1.2]:  generating DH key...
[User 192.168.1.2]:  creating TLS keys...
[Server Ministero della Salute]:  sending public key and certificate...
[User 192.168.1.2]:  verifing certificate...
[User 192.168.1.2]:  certificate verified!
[User 192.168.1.2]:  TLS handshake completed

Insert the last 8 digits of your HealthCard:
Insert the HealthCard expiration date (format YYYY-mm-dd):
[User 192.168.1.2]:  sending sanity information...
[Server Ministero della Salute]:  sanity info received
[Server Ministero della Salute]:  checking info...
[Server Ministero della Salute]:  user info correct
[Server Ministero della Salute]:  sending OTP...
[User 192.168.1.2]:  OTP received 5975c1

Insert the OTP received on your phone:
[User 192.168.1.2]:  sending OTP and public key...
[Server Ministero della Salute]:  OTP and public key received
[Server Ministero della Salute]:  OTP correctly verified
[Server Ministero della Salute]:  sending green pass...
[User 192.168.1.2]:  green pass received
[User 192.168.1.2]:  closing connection...
[Server Ministero della Salute]:  closing connection with 192.168.1.2...
[Server Ministero della Salute]:  connection closed


##################################################

#              Sala Bingo Mister Joker             #

##################################################

## User 192.168.1.1 is accessing to Sala Bingo Mister Joker... ##

[User 192.168.1.1]:  connecting to Sala Bingo Mister Joker...
[Server Sala Bingo Mister Joker]:  new connection from 192.168.1.1

[User 192.168.1.1]:  starting TLS handshake...
[User 192.168.1.1]:  generating DH parameters...
[User 192.168.1.1]:  sending DH contribution...
[Server Sala Bingo Mister Joker]:  sending DH contribution...
[Server Sala Bingo Mister Joker]:  generating DH key...
[Server Sala Bingo Mister Joker]:  creating TLS keys...
[User 192.168.1.1]:  generating DH key...
[User 192.168.1.1]:  creating TLS keys...
[Server Sala Bingo Mister Joker]:  sending public key and certificate...
[User 192.168.1.1]:  verifing certificate...
[User 192.168.1.1]:  certificate verified!
[User 192.168.1.1]:  TLS handshake completed

[User 192.168.1.1]:  sending green pass footprint...
[Server Sala Bingo Mister Joker]:  green pass correctly verified

[User 192.168.1.1]:  starting identification...
```

```
[Server Sala Bingo Mister Joker]:  user identified
[Server Sala Bingo Mister Joker]:  guaranteed profile access

[User 192.168.1.1]:  sending policy GDPR information...
[Server Sala Bingo Mister Joker]:  policy GDPR information received
[Server Sala Bingo Mister Joker]:  verifying policy GDPR information...
[Server Sala Bingo Mister Joker]:  policy accepted
[Server Sala Bingo Mister Joker]:  user enabled to play

## User 192.168.1.2 is accessing to Sala Bingo Mister Joker... ##

[User 192.168.1.2]:  connecting to Sala Bingo Mister Joker...
[Server Sala Bingo Mister Joker]:  new connection from 192.168.1.2

[User 192.168.1.2]:  starting TLS handshake...
[User 192.168.1.2]:  generating DH parameters...
[User 192.168.1.2]:  sending DH contribution...
[Server Sala Bingo Mister Joker]:  sending DH contribution...
[Server Sala Bingo Mister Joker]:  generating DH key...
[Server Sala Bingo Mister Joker]:  creating TLS keys...
[User 192.168.1.2]:  generating DH key...
[User 192.168.1.2]:  creating TLS keys...
[Server Sala Bingo Mister Joker]:  sending public key and certificate...
[User 192.168.1.2]:  verifing certificate...
[User 192.168.1.2]:  certificate verified!
[User 192.168.1.2]:  TLS handshake completed

[User 192.168.1.2]:  sending green pass footprint...
[Server Sala Bingo Mister Joker]:  green pass correctly verified

[User 192.168.1.2]:  starting identification...
[Server Sala Bingo Mister Joker]:  user identified
[Server Sala Bingo Mister Joker]:  guaranteed profile access

[User 192.168.1.2]:  sending policy GDPR information...
[Server Sala Bingo Mister Joker]:  policy GDPR information received
[Server Sala Bingo Mister Joker]:  verifying policy GDPR information...
[Server Sala Bingo Mister Joker]:  policy accepted
[Server Sala Bingo Mister Joker]:  user enabled to play

##################################################

#               Game initialization               #

##################################################

[Server Sala Bingo Mister Joker]:  starting game...
[Server Sala Bingo Mister Joker]:  creating smart contract...
[Server Sala Bingo Mister Joker]:  sending smart contract to the blockchain...

[Server Sala Bingo Mister Joker]:  sending smart contract to player 192.168.1.1...
[User 192.168.1.1]:  smart contract received

[Server Sala Bingo Mister Joker]:  sending smart contract to player 192.168.1.2...
[User 192.168.1.2]:  smart contract received

## Starting round 1... ##

[Server Sala Bingo Mister Joker]:  generating contribution...
[Server Sala Bingo Mister Joker]:  creating commitment...

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.1...
[User 192.168.1.1]:  server commitment received

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.2...
[User 192.168.1.2]:  server commitment received
```

```
[User 192.168.1.1]:  sending contribution to the smart contract...

[User 192.168.1.2]:  sending contribution to the smart contract...

[SmartContract Contribution Game]:  all players have contributed

[Server Sala Bingo Mister Joker]:  retreiving players contributions...
[Server Sala Bingo Mister Joker]:  computing final random string...
[Server Sala Bingo Mister Joker]:  Random string: 67f0d2a82322ca7689b4014cc1044d1c55f1ea5c5e7cef8b09783a62a7d742b0

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.1...
[User 192.168.1.1]:  final random string and server contribution received
[User 192.168.1.1]:  verifying server commitment...
[User 192.168.1.1]:  server commitment verified
[User 192.168.1.1]:  verifying random string...
[User 192.168.1.1]:  random string verified

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.2...
[User 192.168.1.2]:  final random string and server contribution received
[User 192.168.1.2]:  verifying server commitment...
[User 192.168.1.2]:  server commitment verified
[User 192.168.1.2]:  verifying random string...
[User 192.168.1.2]:  random string verified

## Round 1 completed. ##


## Starting round 2... ##

[Server Sala Bingo Mister Joker]:  generating contribution...
[Server Sala Bingo Mister Joker]:  creating commitment...

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.1...
[User 192.168.1.1]:  server commitment received

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.2...
[User 192.168.1.2]:  server commitment received

[User 192.168.1.1]:  sending contribution to the smart contract...

[User 192.168.1.2]:  sending contribution to the smart contract...

[SmartContract Contribution Game]:  all players have contributed

[Server Sala Bingo Mister Joker]:  retreiving players contributions...
[Server Sala Bingo Mister Joker]:  computing final random string...
[Server Sala Bingo Mister Joker]:  Random string: 2e6f787accb21063f7c866e1eedfdce0e9bee2852bacf0c2b6c5eec2745ff5c1

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.1...
[User 192.168.1.1]:  final random string and server contribution received
[User 192.168.1.1]:  verifying server commitment...
[User 192.168.1.1]:  server commitment verified
[User 192.168.1.1]:  verifying random string...
[User 192.168.1.1]:  random string verified

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.2...
[User 192.168.1.2]:  final random string and server contribution received
[User 192.168.1.2]:  verifying server commitment...
[User 192.168.1.2]:  server commitment verified
[User 192.168.1.2]:  verifying random string...
[User 192.168.1.2]:  random string verified

## Round 2 completed. ##
```

```
## Starting round 3... ##

[Server Sala Bingo Mister Joker]:  generating contribution...
[Server Sala Bingo Mister Joker]:  creating commitment...

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.1...
[User 192.168.1.1]:  server commitment received

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.2...
[User 192.168.1.2]:  server commitment received

[User 192.168.1.1]:  sending contribution to the smart contract...

[User 192.168.1.2]:  sending contribution to the smart contract...

[SmartContract Contribution Game]:  all players have contributed

[Server Sala Bingo Mister Joker]:  retreiving players contributions...
[Server Sala Bingo Mister Joker]:  computing final random string...
[Server Sala Bingo Mister Joker]:  Random string: 6943a7bc34cc5e6373d0e55b6ca95b3e64c9e2060084abb8a2d70fc189dd1d42

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.1...
[User 192.168.1.1]:  final random string and server contribution received
[User 192.168.1.1]:  verifying server commitment...
[User 192.168.1.1]:  server commitment verified
[User 192.168.1.1]:  verifying random string...
[User 192.168.1.1]:  random string verified

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.2...
[User 192.168.1.2]:  final random string and server contribution received
[User 192.168.1.2]:  verifying server commitment...
[User 192.168.1.2]:  server commitment verified
[User 192.168.1.2]:  verifying random string...
[User 192.168.1.2]:  random string verified

## Round 3 completed. ##


## Starting round 4... ##

[Server Sala Bingo Mister Joker]:  generating contribution...
[Server Sala Bingo Mister Joker]:  creating commitment...

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.1...
[User 192.168.1.1]:  server commitment received

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.2...
[User 192.168.1.2]:  server commitment received

[User 192.168.1.1]:  sending contribution to the smart contract...

[User 192.168.1.2]:  sending contribution to the smart contract...

[SmartContract Contribution Game]:  all players have contributed

[Server Sala Bingo Mister Joker]:  retreiving players contributions...
[Server Sala Bingo Mister Joker]:  computing final random string...
[Server Sala Bingo Mister Joker]:  Random string: 83a442e80a5894dc95f289804c507aec562884feb4ebf25dee7819547fcf6877

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.1...
[User 192.168.1.1]:  final random string and server contribution received
[User 192.168.1.1]:  verifying server commitment...
[User 192.168.1.1]:  server commitment verified
[User 192.168.1.1]:  verifying random string...
[User 192.168.1.1]:  random string verified
```

```
[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.2...
[User 192.168.1.2]:  final random string and server contribution received
[User 192.168.1.2]:  verifying server commitment...
[User 192.168.1.2]:  server commitment verified
[User 192.168.1.2]:  verifying random string...
[User 192.168.1.2]:  random string verified

## Round 4 completed. ##


## Starting round 5... ##

[Server Sala Bingo Mister Joker]:  generating contribution...
[Server Sala Bingo Mister Joker]:  creating commitment...

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.1...
[User 192.168.1.1]:  server commitment received

[Server Sala Bingo Mister Joker]:  sending contribution to player 192.168.1.2...
[User 192.168.1.2]:  server commitment received

[User 192.168.1.1]:  sending contribution to the smart contract...

[User 192.168.1.2]:  sending contribution to the smart contract...

[SmartContract Contribution Game]:  all players have contributed

[Server Sala Bingo Mister Joker]:  retreiving players contributions...
[Server Sala Bingo Mister Joker]:  computing final random string...
[Server Sala Bingo Mister Joker]:  Random string: 088ef6e2fd5b7c1de2f1e14813376926cbef839d46bf4d209bad64a49793685b

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.1...
[User 192.168.1.1]:  final random string and server contribution received
[User 192.168.1.1]:  verifying server commitment...
[User 192.168.1.1]:  server commitment verified
[User 192.168.1.1]:  verifying random string...
[User 192.168.1.1]:  random string verified

[Server Sala Bingo Mister Joker]:  sending final random string and server contribution to player 192.168.1.2...
[User 192.168.1.2]:  final random string and server contribution received
[User 192.168.1.2]:  verifying server commitment...
[User 192.168.1.2]:  server commitment verified
[User 192.168.1.2]:  verifying random string...
[User 192.168.1.2]:  random string verified

## Round 5 completed. ##


##################################################

#                     GOODBYE                    #

##################################################
```