UNIVERSITÀ
DI PARMA

Report sul progetto per "High Performance Computing"

Corso a cura del professore Michele Amoretti

Giuseppe Ricciardi 355695, Danilo Paglialunga 345142

LM Ingegneria Informatica, ottobre 2023

Indice

1 Introduzione

Lo scopo di questa relazione è quello di illustrare e analizzare il processo di realizzazione del progetto per il corso di *High Performance Computing*, che consiste nella simulazione del modello proposto da Peter Schiansky [1] per l'utilizzo di token quantistici nell'ambito di pagamenti digitali.
Attualmente, la sicurezza dei sistemi crittografici si basa su problemi matematici computazionalmente complessi, ma questi sistemi sono vulnerabili agli attacchi computazionali quantistici. La ricerca di soluzioni resistenti ai computer quantistici è in corso, ma alcuni di questi sistemi sono già stati compromessi da attacchi computazionali. I principi della meccanica quantistica possono offrire sicurezza contro avversari con un potere computazionale illimitato per alcune applicazioni, conosciuta come sicurezza informativa. Quest'ultima è una delle principali motivazioni che spinge la ricerca nell'ambito quantistico per la nascita di una "quantum internet".
Attualmente, la tecnologia più matura e ampiamente implementata per questo scopo è la *Quantum Key Distribution (QKD)*, che consente a due parti di comunicare e di scambiarsi delle chiavi per comunicare in modo sicuro su un canale pubblico. Il sistema proposto dal paper genera e verifica criptogrammi

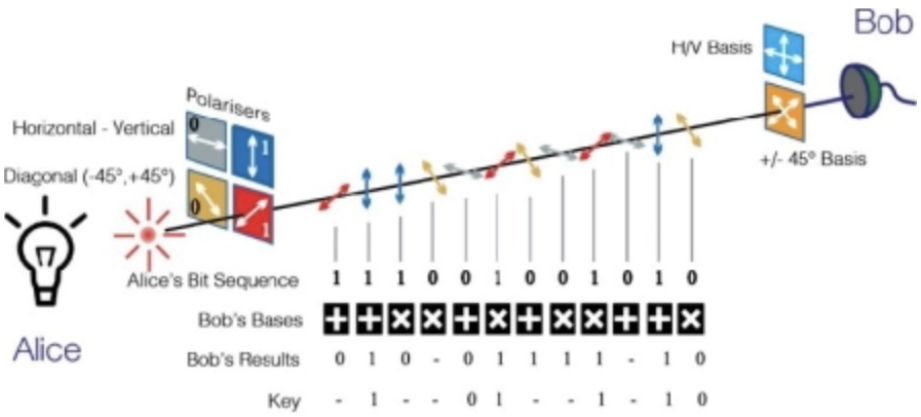
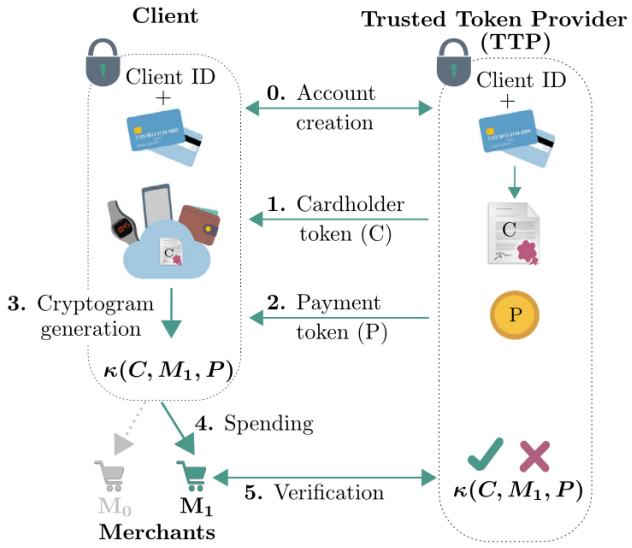


Figura 1: Funzionamento del QKD

quantistici a sicurezza informativa, garantendo la validità delle transazioni senza richiedere fiducia in canali intermedi o reti no trust. È necessaria solo una comunicazione autenticata tra il cliente e il fornitore dei token pagamento in un momento precedente.Il fornitore dei token di pagamento nella realtà viene rappresentata da un istituto considerato sicuro, ad esempio una banca, la quale all'interno di questo progetto prende il nome di TTP (Trusted Token Provider). La protezione delle informazioni sensibili del cliente è garantita da una funzione a sicurezza informativa, mentre l'impegno per l'acquisto è garantito dalle leggi della meccanica quantistica. Inoltre, in caso di filiali di verifica multiple, non è richiesta alcuna comunicazione incrociata per convalidare la transazione.

1.1 Funzionamento dei pagamenti digitali

Prima di descrivere il modello proposto dal paper è utile illustrare il funzionamento classico dei pagamenti digitali.



Il funzionamento del modello in figura 1.1 può essere riassunto nei seguenti passaggi:

1. Il **Cliente** crea un account presso un **Trusted Token Provider (TTP)** concordando e condividendo in maniera sicura con esso il proprio codice cliente e le informazioni sensibili della carta di credito attraverso un canale autenticato ed encryptato.

2. Il TTP, una volta siglato l'accordo con il Client, invia il token relativo al titolare della carta, denominato C, attraverso un canale sicuro.

3. Dopo che il Client ne fa richiesta, il TTP genera casualmente un token di pagamento monouso, denominato P, e lo invia al Cliente tramite un canale sicuro.

4. Il dispositivo del Cliente utilizza il token segreto C memorizzato, l'ID pubblico del **Mercante** scelto presso cui fare l'acquisto M_i e il token di pagamento P per calcolare un criptogramma $k(C, M_i, P)$ mediante l'uso della funzione HMAC.

5. Il Cliente invia il crittogramma trovato aggiungendo al messaggio il proprio token C. Il Mercante riceverà il messaggio e si occuperà di aggiungere ad esso il proprio codice ID pubblico per procedere successivamente alla verifica del token di Pagamento.

6. Il Commerciante richiede al TTP di verificare il crittogramma inviando il messaggio che contiene il crittogramma, il token C del cliente e il proprio ID pubblico. Il TTP calcolerà l'HMAC sulla base delle informazioni contenute nel messaggio ricevuto dal Commerciante. Se il crittogramma trovato coincide con quello inviato dal Cliente **accetta la transazione**, in tutti gli altri casi il TTP respinge la transazione.

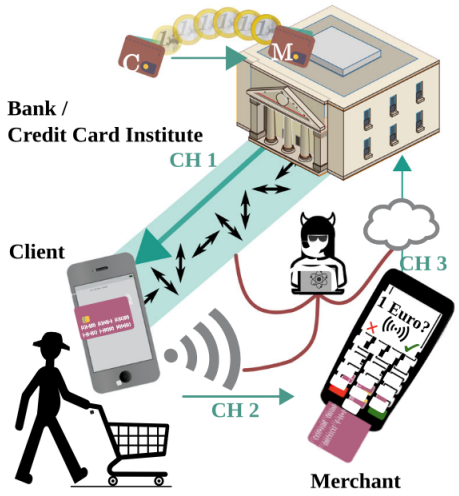


Figura 2: Rappresentazione del modello di pagamenti digitali quantistici

Lo schema in figura 2 illustra il funzionamento del modello proposto da [1]. Così come per i pagamenti classici, anche in questo caso sono considerate tre parti: un Cliente, un Commerciante e un Istituto Bancario/Emittente di carte di credito. I canali quantistici CH 1, CH 2 e CH 3 sono insicuri.
Tutte le parti coinvolte, tranne la Banca, possono anche agire in modo malevolo. Durante un pagamento, la Banca invia un insieme di stati quantistici al dispositivo del Cliente (ad esempio, telefono, computer, ecc.), che li misura e li trasforma in un token di pagamento sicuro quantisticamente - un *criptogramma* - che mostriamo qui come una carta di credito monouso. Il Cliente utilizza questo token classico per pagare il Commerciante, il quale successivamente contatta la Banca per la verifica del pagamento.
La Banca (TTP) crea coppie di fotoni intrinsecamente correlati utilizzando una sorgente di Conversione Parametrica Spontanea (SPDC). La polarizzazione di uno dei fotoni viene misurata casualmente dal TTP in una base lineare o diagonale, creando la descrizione classica (b, B), che prepara in remoto il token quantistico $|P\rangle$ sul secondo fotone. Quest'ultimo viene inviato al Cliente che misura la sua polarizzazione in una base $m_i = MAC(C, M_i)$ specificata da un Codice di Autenticazione del Messaggio (MAC) dell'ID del Commerciante M_i e del token privato del Cliente C, ottenendo così $|P\rangle$. La comunicazione clas-

sica tra il TTP, il Cliente e il Commerciante viene utilizzata per verificare la compatibilità tra k, M_i e C con (b, B).

2 Realizzazione

Per implementare il modello precedentemente descritto, è stato utilizzato il linguaggio di programmazione *Python* insieme alla libreria *netqasm*, che fa uso del simulatore di rete quantistica *squidASM*. Questa sezione fornisce dettagli sull'implementazione, iniziando dal ruolo del TTP e procedendo fino al Mercante. Alla fine di questa sezione, vengono presentate le fasi di testing e un'analisi dei risultati ottenuti.

2.1 Struttura del progetto

La figura 3 mostra l'architettura del progetto.

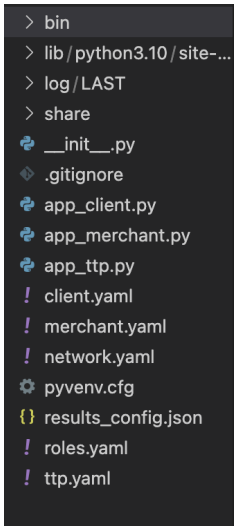


Figura 3: Struttura del progetto

I file denominati "app_client.py," "app_merchant.py," e "app_ttp.py" costituiscono la base per la logica e il funzionamento dei diversi elementi del progetto, mentre i file con estensione ".yaml" vengono utilizzati per specificare i parametri da impiegare nelle simulazioni.

2.2 TTP

Il **TTP** (Trusted Token Provider) svolge un ruolo centrale nel sistema di pagamento digitale proposto, in particolare si occupa di:

1. **Generare e distribuire gli stati quantistici**. Il TTP genera una serie casuale di bit e una base coniugata corrispondente per rappresentare il token di pagamento quantistico $|P\rangle$. Il Token di pagamento quantistico $|P\rangle$ viene distribuito mediante l'uso di una comunicazione quantistica al Cliente. Il metodo *distribute_states* definito in "app_ttp.py" si occupa di realizzare questo passaggio.

```python
def distribute_states(conn, epr_socket, socket, target, n):
    bit_flips = [None for _ in range(n)]
    basis_flips = [random.randint(0, 1) for _ in range(n)]

    for i in range(n):
        q = epr_socket.create_keep(1)[0]
        if basis_flips[i]:
            q.H()
        m = q.measure()
        conn.flush()
        bit_flips[i] = int(m)
    return bit_flips, basis_flips
```

Figura 4: Metodo *distribute_states* in "app_ttp.py"

2. **Calcolare l'HMAC**. Dopo aver distribuito gli stati quantistici, il TTP è pronto a ricevere un messaggio da un Commerciante per effettuare il calcolo dell'HMAC (Hash-based Message Authentication Code). L'obiettivo di questo calcolo è garantire l'integrità e l'autenticità del messaggio ricevuto. Per eseguire questa operazione, il TTP utilizza l'algoritmo di hash SHA-256, facendo uso della libreria HMAC, e la chiave necessaria per l'HMAC è estratta dal messaggio stesso fornito dal Commerciante.

```python
def compute_hmac(key, message):
    # Usa SHA-256 come funzione hash di base
    hash_algorithm = hashlib.sha256

    # Calcola l'HMAC utilizzando la chiave e il messaggio
    hmac_result = hmac.new(key, message, hash_algorithm)

    # Converte l'HMAC in una stringa binaria
    hmac_binary = binascii.hexlify(hmac_result.digest())
    return hmac_binary
```

Figura 5: Metodo *compute_hmac* in "app_ttp.py"

3. **Verificare il token di pagamento**. Dopo aver ricevuto il messaggio criptogramma dal Commerciante, il TTP avvia il processo di verifica del token di pagamento. Il TTP confronta i risultati delle misurazioni effettuate dal Cliente con quelle della propria misurazione, eseguendo

una verifica basata sul confronto delle basi e dei risultati delle misurazioni. L'implementazione utilizza una tolleranza del 75% per accettare il pagamento.

4. **Gestire le comunicazioni con Clienti e Commercianti**. Il TTP gestisce la comunicazione con i Clienti e i Commercianti attraverso socket dedicati, come indicato nella tua implementazione. I messaggi strutturati sono scambiati tra il TTP, i Clienti e i Commercianti per coordinare le operazioni e inviare conferme.

5. **Registrazione e Log**. Il TTP registra dettagli sulle operazioni effettuate e sugli eventi importanti, come dimostrato nella tua implementazione. Il modulo di log di NetQASM viene utilizzato per tenere traccia degli eventi e delle informazioni significative durante il processo.

2.3 Il cliente

Il **Cliente** si occupa della fase di inizializzazione della transazione, della comunicazione con il TTP e con il Commerciante, della gestione dei dati di pagamento e della creazione del crittogramma **k**.

1. **Inizializzazione**. Il Cliente si inizializza e apre un socket di comunicazione con il TTP per avviare il processo di richiesta di un token di pagamento. Il Cliente specifica il proprio codice cliente (id_client_bank) per l'identificazione.

2. **Scelta del Commerciante**. Il Cliente seleziona un Commerciante con cui desidera effettuare una transazione dalla lista fornita e precondivisa con il TTP. Viene calcolato un HMAC per garantire l'autenticità della richiesta. La chiave HMAC è generata utilizzando il proprio codice cliente e il codice del Commerciante selezionato.

3. **Comunicazione con il TTP**. Il Cliente comunica con il TTP per richiedere lo stato quantistico necessario per la transazione. In questo processo, il TTP distribuisce gli stati quantistici al Cliente in base alle basi coniugate precedentemente calcolate. Il Cliente riceve gli stati quantistici e li misura in base al risultato dell'output restituito dall'operazione di HMAC effettuata. Questa operazione viene eseguita nella funzione *receive_states*. Il risultato della misurazione sarà il crittogramma **k** da inviare successivamente al Commerciante.

4. **Comunicazione con il Commerciante**. Il Cliente invia un messaggio al Commerciante per iniziare il processo di pagamento. Il messaggio include i risultati delle misurazioni che costituiscono il crittogramma **k** e il proprio **codice cliente**.

```
"""Socket nel quale il Client riceve il token di pagamento"""
client = NetQASMConnection(
    app_name=app_config.app_name,
    log_config=app_config.log_config,
    epr_sockets=[epr_socket],
)
with client:
    bit_flips, basis_flips = receive_states(
        client, epr_socket, socket, "ttp", num_bits, bases_token
    )
outcomes = [int(b) for b in bit_flips]
basis = [int(b) for b in basis_flips]
print(f"Client raw key: {outcomes}")
print("Mando al commerciante il token di pagamento")
print("\n ------------------------------------ \n")
msgMerchant = StructuredMessage(header="InfoToken", payload=[outcomes,id_client_bank])
"""INVIO DELLA PROPRIA BASE"""
socket2.send_structured(msgMerchant)
return{"raw_key": outcomes,}
```

Figura 6: Ricezione del token di pagamento e comunicazione con il commerciante da parte del cliente

2.4 Il mercante

Il **Mercante** svolge un ruolo marginale nella verifica delle transazioni in quanto si occupa semplicemente di aggiungere al messaggio ricevuto dal Cliente il proprio identificativo. Successivamente, inoltra il nuovo messaggio al TTP che si occuperà di verificare la validità del token e della transazione. Infine, attende un messaggio dal TTP che lo informerà sull'esito della transazione.

1. **Inizializzazione e Comunicazione con il Cliente**. Il Mercante si inizializza e apre un socket di comunicazione con il Cliente per avviare il processo di pagamento.

2. **Ricezione del Criptogramma dal Cliente**. Il Mercante riceve il criptogramma dal Cliente tramite il socket di comunicazione dedicato. Il codice del Mercante viene aggiunto al criptogramma per ulteriori operazioni di verifiche ed identificazione.

3. **Comunicazione con il TTP**. Il Mercante apre un socket di comunicazione con il Trusted Token Provider (TTP) per inviare il criptogramma e richiedere la verifica della transazione. Il criptogramma viene inviato al TTP tramite un messaggio strutturato per la verifica. Il TTP riceve il criptogramma dal Mercante e avvia il processo di verifica analizzato in precedenza.

4. **Conclusione della Transazione**. Il Mercante riceve la risposta dal TTP, che può essere una conferma o un rifiuto della transazione. Il Mercante completa la transazione in base alla risposta ricevuta dal TTP. Se la transazione è stata confermata, il Mercante registra il pagamento come

avvenuto con successo. In caso di rifiuto, il Mercante informa il Cliente della transazione non riuscita.

```python
def main(app_config=None, num_bits=128, id_merchant=None):
    print("\n ------------------------------------ \n")
    print("Ciao sono Merchant, apro il Socket con il Cliente")
    # Socket for classical communication
    merchant = NetQASMConnection(
        app_name=app_config.app_name,
        log_config=app_config.log_config,
    )
    socket = Socket("merchant", "client", log_config=app_config.log_config)
    socket2 = Socket("merchant", "ttp", log_config=app_config.log_config)
    criptogram = socket.recv_structured().payload
    criptogram.append(id_merchant)
    print(f"Client criptogram: {criptogram}")
    print(f"Mando il messaggio completo ricevuto dal Client per verificare la correttezza")
    print("\n ------------------------------------ \n")
    msgTTP = StructuredMessage(header="InfoToken", payload=criptogram)
    """INVIO DEL TOKEN di Pagamento al TTP"""
    socket2.send_structured(msgTTP)
    msgVerifica = socket2.recv_structured().payload
    print(msgVerifica[0])
```

Figura 7: Codice sorgente del mercante

2.5 Testing e risultati

Nella fase di testing la lunghezza della chiave è pari a 128bit.

2.5.1 Simulazione 1: Token di Pagamento Rifiutato

```
 ------------------------------------

Ciao sono Client, apro il Socket con TTP
Il mio codice cliente: 10003400

 ------------------------------------

Ciao sono Merchant, apro il Socket con il Cliente

 ------------------------------------

 ------------------------------------

Ciao sono TTP! Apro il mio Socket per comunicare con Client
Chiave da N bit  128
Lista clienti banca:
[10003400]

Scegli il commerciante da cui effettuare l'acquisto
1. Esselunga
2. Coop
```

```
22 Inserisci il numero del commerciante da contattare: 1
23 Hai scelto di effettuare la transazione presso: Esselunga
24 Message: Esselunga
25 HMAC (binary): b'41390
       a20db67ee9129a74d7acb08468935c5d097c09b7e8c9b0714d3c42c2c3e'
26 Base misurazione token di pagamento[1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
27
28  -------------------------------------
29
30 Risultati chiave TTP: {outcomes}
31 Risultati base TTP: {theta}
32 Client raw key: [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
        1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
        0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
        1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0,
        1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
        1, 0, 0, 1, 0, 0]
33 Mando al commerciante il token di pagamento
34
35  -------------------------------------
36
37 Client criptogram: [[0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 0], 10003400, 'Esselunga']
38 Mando il messaggio completo ricevuto dal Client per verificare la
       correttezza
39
40  -------------------------------------
41
42 TTP ho ricevuto il criptogramma: [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0]
43 Message: Esselunga
44 HMAC (binary): b'41390
       a20db67ee9129a74d7acb08468935c5d097c09b7e8c9b0714d3c42c2c3e'
45 Base misurazione token di pagamento[1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
```

```
Verifica finale della validit\'a del token. Tolleranza 75\%

 -------------------------------------

Token di pagamento non accettato
outcomes: [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
     0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
     1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1,
     1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0,
     0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1,
     0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0,
     0, 0, 1, 0] theta: [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
    1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
    0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
    1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1,
    1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
    0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
    0, 1, 1, 1, 0, 0, 1, 0, 0]
key_token: [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1,
    1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0,
    1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
    1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
    0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
    1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
    0, 0, 1, 0, 0]
bases_token:[1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
    0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1,
    0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0,
    0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
    0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
    1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
    1, 0, 0, 1, 0]
Pagamento non approvato
```

Nella prima simulazione, il token di pagamento è stato rifiutato dal TTP. Questo è avvenuto a causa delle differenze tra il token inviato dal Cliente e quello atteso dal TTP, con una discrepanza superiore alla tolleranza del 75%.

Analisi: Il token generato dal Cliente non corrispondeva al token atteso dal TTP. Questa discrepanza potrebbe essere dovuta a errori durante la generazione del token o a disturbi nel canale quantistico.

2.5.2 Simulazione 2: Token di Pagamento Accettato

```
(env) danilopag@danilopag-SVE1513C1EW:~/Scrivania/bb84/env$ netqasm
     simulate

--------------------------------------

Ciao sono Client, apro il Socket con TTP
Il mio codice cliente: 10003400

 --------------------------------------


 --------------------------------------
```

```
12
13
14  ------------------------------------
15
16 Ciao sono Merchant, apro il Socket con il Cliente
17 Ciao sono TTP! Apro il mio Socket per comunicare con Client
18 Chiave da N bit  128
19 Lista clienti banca:
20 [10003400]
21 Scegli il commerciante da cui effettuare l'acquisto
22 1. Esselunga
23 2. Coop
24 Inserisci il numero del commerciante da contattare: 1
25 Hai scelto di effettuare la transazione presso: Esselunga
26 Message: Esselunga
27 HMAC (binary): b'41390
       a20db67ee9129a74d7acb08468935c5d097c09b7e8c9b0714d3c42c2c3e'
28 Base misurazione token di pagamento[1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
29 Client raw key: [1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
        0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
        1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
        0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0,
        1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
        0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
        1, 0, 1, 0, 0, 0]
30 Mando al commerciante il token di pagamento
31
32  ------------------------------------
33
34
35  ------------------------------------
36
37 Risultati chiave TTP: {outcomes}
38 Risultati base TTP: {theta}
39 Client criptogram: [[1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0], 10003400, 'Esselunga']
40 Mando il messaggio completo ricevuto dal Client per verificare la
       correttezza
41
42  ------------------------------------
43
44 TTP ho ricevuto il criptogramma: [1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
```

```
    1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
    0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0]
Message: Esselunga
HMAC (binary): b'41390
    a20db67ee9129a74d7acb08468935c5d097c09b7e8c9b0714d3c42c2c3e'
Base misurazione token di pagamento[1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
    1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
    1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
    1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
    1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
    0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
    0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
Verifica finale della validit\'a del token. Tolleranza 75%

 -------------------------------------

Token di pagamento accettato
Pagamento approvato
```

Nella seconda simulazione, il token di pagamento è stato accettato dal TTP, consentendo il completamento della transazione.

Analisi: Il token generato dal Cliente corrispondeva al token atteso dal TTP entro la tolleranza del 75%. La transazione è stata quindi approvata e completata con successo.

2.5.3 Risultati Ottenuti dal Paper di Riferimento del Protocollo

I risultati presentati nel paper di riferimento del protocollo evidenziano in modo inequivocabile l'efficacia del sistema proposto nel garantire la sicurezza delle transazioni e nell'assicurare l'autenticità dei token di pagamento. Un aspetto fondamentale emerso dalle simulazioni è l'escalation della sicurezza in relazione alla lunghezza della chiave quantistica utilizzata.

Con l'aumentare della lunghezza della chiave quantistica, diventa sempre più improbabile che un utente malintenzionato possa effettuare un attacco di Double Spending. Questo fenomeno è il risultato diretto dell'intrinseca natura della crittografia quantistica, che sfrutta i principi della fisica quantistica per garantire la sicurezza delle informazioni trasmesse. Mentre nei sistemi classici la sicurezza si basa su complesse operazioni matematiche e algoritmi crittografici, la crittografia quantistica si avvale delle leggi della meccanica quantistica, rendendo l'intercettazione delle informazioni una violazione fisica delle leggi della natura stessa.

In questo contesto, la lunghezza della chiave quantistica agisce come una sorta di barriera invalicabile per gli attaccanti. Aumentando esponenzialmente le dimensioni della chiave, si rende praticamente impossibile la decrittazione del messaggio da parte di un utente malevolo, garantendo così la sicurezza delle transazioni e prevenendo qualsiasi tentativo di frode.

Questi risultati confermano che il protocollo proposto rappresenta un passo avanti significativo nella sicurezza delle transazioni digitali, aprendo la strada a nuove opportunità nel mondo dei pagamenti online sicuri e affidabili. La combinazione di tecnologie avanzate della crittografia quantistica con l'analisi

rigorosa svolta nel paper di riferimento offre una base solida per lo sviluppo di sistemi di pagamento futuri, dove l'integrità delle transazioni è garantita da principi scientifici fondamentali.

3 Conclusioni

In conclusione, il sistema di pagamento digitale basato sulla crittografia quantistica rappresenta un notevole progresso nella sicurezza delle transazioni online. Attraverso un'implementazione basata su Python e l'uso della libreria netqasm in combinazione con il simulatore di rete quantistica squidASM, il sistema è stato progettato con cura per garantire la massima sicurezza e autenticità nelle transazioni finanziarie

Il ruolo centrale del Trusted Token Provider (TTP) è fondamentale in questo contesto, poiché gestisce la generazione e la distribuzione degli stati quantistici, calcola l'HMAC per proteggere l'integrità dei messaggi e verifica l'autenticità dei token di pagamento. La capacità del TTP di mantenere un registro delle operazioni svolte aggiunge un ulteriore livello di sicurezza e trasparenza al sistema.

I Clienti e i Commercianti giocano ruoli complementari nel processo di transazione, facilitando la comunicazione con il TTP e assicurando che le operazioni si svolgano senza intoppi. Le simulazioni eseguite dimostrano l'efficacia del sistema nell'accettare o respingere i token di pagamento in base alla loro validità, confermando la solidità della struttura progettata.

Inoltre, i risultati ottenuti dal paper di riferimento del protocollo evidenziano chiaramente il vantaggio di utilizzare chiavi quantistiche più lunghe, che rendono praticamente impossibile qualsiasi tentativo di frode, inclusi gli attacchi di Double Spending.

In un'epoca in cui la sicurezza delle transazioni digitali finanziarie è fondamentale, il sistema di pagamento digitale basato sulla crittografia quantistica offre una soluzione avanzata e affidabile. La combinazione di tecnologie di crittografia quantistica con un'analisi rigorosa fornisce una solida base per lo sviluppo di sistemi di pagamento futuri, in cui l'integrità delle transazioni è garantita da principi scientifici e matematici fondamentali. Questo rappresenta un passo avanti significativo nel mondo dei pagamenti online sicuri e affidabili. Tutto il codice scritto per la realizzazione di questo progetto è open-source con licenza General Public License (GPLv3) ed è consultabile sulla piattaforma Github al link https://github.com/danilopag/Quantum-Internet-Payment.

Riferimenti bibliografici

[1] Peter Schiansky, Julia Kalb, Esther Sztatecsny, Marie-Christine Roehsner, Tobias Guggemos, Alessandro Trenti, Mathieu Bozzio, and Philip Walther. Demonstration of quantum-digital payments. *Nature Communications*, 14(1), jun 2023.