# Project Work APS

GRUPPO AAPM

09 July
A.Y. 2023/24



| | | | | |
|---|---|---|---|---|
| Antonio | Sessa | 0622702305 | a.sessa108@studenti.unisa.it | WP3 |
| Angelo | Molinario | 0622702311 | a.molinario3@studenti.unisa.it | WP4 |
| Massimiliano | Ranauro | 0622702373 | m.ranauro2@studenti.unisa.it | WP2 |
| Pietro | Martano | 0622702402 | p.martano@studenti.unisa.it | WP1 |


## Indice

# 1 WP1

## 1.1 Attori onesti del sistema

• **Fornitori**, sono le entità che hanno la capacità di rilasciare delle credenziali agli utenti che si identificano mediante una CIE. Il loro obiettivo è quello di riconoscere tramite la CIE gli utenti, ed erogare credenziali agli utenti autorizzati. Dispongono di un certificato digitale valido riconosciuto da utenti e server.
• **Server**, sono i detentori dei servizi che possono essere erogati agli utenti se in possesso delle giuste credenziali. Le credenziali devono essere rilasciate da fornitori riconosciuti come affidabili dal server. Un server per erogare un servizio può richiedere combinazioni di più credenziali, che possono anche essere rilasciati da fornitori differenti. Il loro obiettivo è quello di erogare i servizi agli utenti muniti di credenziali appropriate.
• **Utenti**, sono gli utenti muniti di CIE che intendono ottenere credenziali dai fornitori. Intendono poi usare le credenziali ricevute per accedere ai servizi forniti dai server. All'interno della CIE, oltre alla chiave segreta di firma, è presente un certificato digitale valido per instaurare connessioni TLS 1.3 rilasciato da IPSZ.
• **Ufficio anagrafe e IPZS**, sono gli enti che si occupano di fornire la carta d'identità elettronica (CIE) e il PIN associato agli utenti. Sono anche una certification authority riconosciuta da tutti gli attori onesti del sistema.

## 1.2 Attori disonesti del sistema

### I. Ladro di credenziali

- **Obiettivo**, Rubare le credenziali di un utente onesto al momento della creazione e/o ottenere le credenziali di un utente onesto.
- **Risorse**, Avversario attivo sul canale di comunicazione durante la fase di richiesta di credenziali da parte di un utente onesto verso un fornitore onesto e/o attivo sul canale di comunicazione durante la fase di richiesta di erogazione di un servizio. Risorse computazionali discrete.

### II. Archivista di credenziali

- **Obiettivo**, Ottenere la conoscenza di quali credenziali sono in possesso gli utenti onesti (cioè associare ad una credenziale una CIE). Il suo obiettivo è solo la conoscenza e non l'usufruire delle credenziali per accedere ai servizi.
- **Risorse**, Avversario attivo sui canali di comunicazione tra utenti onesti e fornitori onesti e utenti onesti e server onesti, può avere le conoscenze di uno o più server.

### III. Fornitore malevolo

. **Obiettivo**, Usufruire delle credenziali rilasciate ad utenti onesti per accedere ai servizi dei server.

. **Risorse**, Conoscenza delle credenziali di ogni utente onesto a cui le ha erogate.

**IV. Forgiatore di credenziali**

. **Obiettivo**, Creazione di credenziali valide per accedere ai servizi dei server che le richiedono.

. **Risorse**, Conoscenza di più credenziali valide, erogate da più fornitori onesti. Discrete risorse di calcolo computazionale.

**V. Fornitore Forgiatore**

. **Obiettivo**, Creazione di credenziali valide per accedere ai servizi dei server che le richiedono.

. **Risorse**, Capacità di forgiare credenziali, inoltre ha la capacità di firmare come un fornitore riconosciuto valido dai server.

**VI. Server malevolo**

. **Obiettivo**, Usufruire delle credenziali ricevute dagli utenti onesti che richiedono servizi per accedere a servizi di altri server.

. **Risorse**, Conoscenza di più credenziali di utenti onesti che hanno effettuato richieste di erogazione servizi al server malevolo.

**VII. Insieme di utenti "onesti"**

. **Obiettivo**, Accedere a servizi che richiedono più credenziali, unendo le credenziali di più utenti.

. **Risorse**, Possesso di diverse credenziali valide erogate a diversi utenti da fornitori onesti.

**VIII. Ladro di servizi**

. **Obiettivo**, Ottenere accesso a servizi a cui non è autorizzato ad accedere.

. **Risorse**, Conoscenza di diverse credenziali erogate a diversi utenti da fornitori onesti. Attivo sul canale di comunicazione tra utenti e server.

**IX. Prestatore di credenziali**

. **Obiettivo**, Prestare le proprie credenziali valide ad utenti che non ne hanno diritto.

. **Risorse**, Possesso di uno o più credenziali valide.

**X. Combinazione di avversari**

- **Obiettivo**, Uno o più obiettivi degli avversari descritti precedentemente.
- **Risorse**, Combinazione delle risorse degli avversari precedentemente descritti.

## 1.3 Proprietà

### 1.3.1 Confidenzialità

1. L'identità degli utenti in possesso/avente diritto a credenziali deve essere nota soltanto al fornitore erogatore della credenziale.
2. Garantire la segretezza delle comunicazioni tra utente e server, in modo che un avversario non può ottenere informazioni utili sui servizi erogati e sull'identità del possessore della credenziale.
3. Garantire la segretezza delle comunicazione tra utente e fornitore, in modo che un avversario non possa ottenere informazioni sulla avvenuta o meno erogazione della credenziale e sulla credenziale.

### 1.3.2 Integrità

1. Durante la fase di erogazione di credenziali, garantire il ricevimento della credenziale corretta all'utente onesto e solo all'utente onesto.
2. Durante la fase di verifica delle credenziali, garantire l'erogazione del servizio se la credenziali sono valide e appartengono al singolo utente, la non erogazione del servizio se la credenziali non sono valide e/o appartengono a più utenti diversi.

### 1.3.3 Trasparenza

1. La validità di una credenziale deve essere pubblicamente verificabile da qualsiasi attore del sistema.

### 1.3.4 Efficienza/Usabilità

1. Garantire che un server possa ricevere il numero minimo di credenziali per accedere ad un servizio.
2. Garantire che la verifica delle credenziali di un utente da parte di un server sia eseguita in modo efficiente.
3. Garantire che la creazione di una credenziale sia eseguita in modo efficiente.

### 1.4 Completeness

Definiamo ora il funzionamento del sistema qualora tutte le parti in gioco operino in modo onesto, rispettado tutte le proprietà definite precedentemente. In particolare:

1. Un utente munito di CIE fa richiesta di erogazione di una credenziale ad un fornitore. Se l'utente è autorizzato a ricevere la credenziale, riceverà una credenziale valida.

2. Un utente facente richiesta di un servizio ad un server, riceverà il servizio dopo la verifica del possedimento delle giuste credenziali.

## 2 WP2

Per semplicità, ogni volta che parliamo di firma, intendiamo una firma ECDSA, inoltre ogni volta che viene utilizzato un certificato digitale è assunto che questo segua il formato x509V3 e che sia stato rilasciato da una Certification Authority trusted.

### 2.1 Richiesta Credenziale

L'utente U, che è interessato a ricevere dal fornitore F una credenziale, instaura una connessione TLS con F. In particolare la connessione è TLS 1.3 e prevede una identificazione mutua: F si identificherà con il suo certificato digitale e U si identificherà con il certificato digitale presente nella sua CIE (questo richiederà l'utilizzo dell'algoritmo $Sign(PIN, hash)$ della CIE). F rifiuterà qualsiasi altro certificato digitale non rilasciato dalla IPSZ (l'unica certification authority riconosciuta dal fornitore è IPSZ) e di conseguenza interromperà la connessione[1].

A questo punto l'utente, che è stato identificato correttamente dal fornitore avvia la procedura di richiesta delle credenziali inviando al fornitore una *Credential Request*. L'utente U genera una coppia di chiavi ($Pk_h$,$Sk_h$) se è la prima volta che richiede una credenziale, altrimenti userà la stessa usata precedentemente. Per generare la coppia di chiavi usa l'algoritmo Gen per chiavi per firme ECDSA, con n uguale a 128. La *Credential Request* è così strutturata:

$$[credential; Cert_{cie}; Pk_h; r] \tag{1}$$

**1.** *credential*: la credenziale richiesta al fornitore F, sono delle stringhe con forma standardizzata di varie dimensioni. e.g: "Residenza:Salerno", "Isee:8000"

**2.** $Cert_{cie}$: Il certificato digitale associato alla CIE dell'utente.

**3.** $Pk_h$: La chiave pubblica dell'utente U.

[1] Ovviamente anche U interromperà la connessione nel caso riconosce come non valido il certificato digitale del fornitore.

4. $r$ : 128 random bits.

Il valore di $r$ ha lo scopo di generare un ID. Un utente per poter utilizzare più credenziali insieme ha necessità che l'ID associato ad ogni credenziale sia identico. L'utente quindi genera il valore di $r$ solamente alla richiesta della prima credenziale e, successivamente, utilizzare sempre quel valore.

Ricevuta la *Credential Request* F verificherà che l'utente U, identificato dalla sua CIE, ha diritto a ricevere la credenziale richiesta. Se la fase di verifica non va a buon fine allora F termina la connessione con l'utente altrimenti risponde inviando ad U la credenziale richiesta e alcune informazioni aggiuntive. La risposta inviata da F, *CRED*, è strutturata nel seguente modo:

$$CRED = [(ID, Cert_f, Pk_h, credential), \sigma_f] \tag{2}$$

1. $(ID, Cert_f, PK_h, credential)$: Tupla di dati necessari per poter utilizzare correttamente una credenziale
    1. $ID$: sequenza di 256 bit rappresentate un singolo utente. Viene calcolato come hash del certificato digitale dell'utente e la sequenza di random bit inviati dall'utente.

$$\text{I}D = SHA256(Cert_{cie}||r) \tag{3}$$

    2. $Cert_f$: Certificato digitale del fornitore che ha concesso le credenziali all'utente.
    3. $Pk_h$: Chiave pubblica scelta dall'utente.
    4. $credential$: Credenziale richiesta dall'utente
2. $\sigma_f$: Firma di $(ID, Cert_f, Pk_h, credential)$ utilizzando la chiave segreta di F $Sk_f$

Il fornitore invia all'utente CRED e allo stesso tempo memorizza all'interno del suo database la CRED in questo modo l'utente che smarrisce una credenziale può recuperarla inviando ad F semplicemente il suo certificato digitale e i random bit utilizzati per la generazione dell'ID (dopo una connessione con identificazione mutua TLS).

## 2.2 Erogazione del servizio

L'utente U, che è interssato a ricevere da un server S un servizio, instaura una connessione TLS con S, questa volta l'identificazione non è mutua e quindi l'utente non invia il proprio certificato. L'utente U invia:

$$[(Pk_h, Sign_{Sk_h}(randTLS_U||randTLS_S||Pk_S)] \tag{4}$$

Dove $randTLS$ sono i bit random inviati dal protocollo TLS rispettivamente dall'utente e dal server, e $Pk_S$ è la chiave pubblica del server, nota dal suo

certificato digitale. Se il server riesce a verificare la validità della firma con $Pk_h$ continua la comunicazione, avendo identificato l'utente come proprietario di $Pk_h$, altrimenti la interrompe. Successivamente il server S richiede le credenziali necessarie per erogare il servizio ad U, che risponde inviando ad S una lista contenente il numero minimo di CRED richieste. Il server S verifica per ogni elemento della lista che:

1. Il Certificato presente in CRED sia di un fornitore valido.
2. La funzione di verifica $Ver(Pk_f, CRED[0], CRED[1]) == 1$; $Pk_f$ è recuperabile dal certificato presente in CRED.
3. $Pk_h$ sia la chiave con cui si è identificato precedentemente l'utente, in seguito alla firma di $(randTLS_U||randTLS_S||Pk_S)$ .
4. la *credential* sia una di quelle richieste.

Se una delle condizioni precedenti non è verificata allora il server termina la connessione con l'utente.
Nel caso in cui il server richieda più di una credenziale allora i passi precedenti vengono eseguiti per ogni elemento della lista ricevuta e viene eseguito un'ulteriore controllo sull'ID dell'utente, in particolare viene verificato che tutte le credenziali necessarie abbiano tutte lo stesso ID, che per come è costruito identifica in maniera univoca un utente.

# 3 WP3

## 3.1 Completezza

Se tutte le parti coinvolte seguono il protocollo, allora un utente munito di CIE può richiedere le credenziali ad uno o più fornitori, e può dopo usare le credenziali inviandole al server per ricevere un servizio.

## 3.2 Confidenzialità

Gli avversari che cercano di intaccare le proprietà di confidenzialità sono i seguenti:

- **Archivista di credenziali**, non riesce a raggiungere il suo obiettivo. Quando un utente si collega ad un fornitore onesto o ad un server onesto instaura un canale di comunicazione con TLS 1.3, rendendo impossibile per l'archivista cogliere qualsiasi informazione utile. Quando l'archivista possiede la conoscenza di un server, e quindi conosce delle CRED, comunque non riesce ad identificare la persona (identificata dalla sua CIE) proprietaria della CRED. Nella CRED è presente una chiave pubblica, $Pk_h$, scollegata dalla CIE, e un ID che è ottenuto come SHA256($Cert_{cie}||r$), dove i 128 random bits garantiscono la proprietà di segretezza.

## 3.3 Integrità

Gli avversari che cercano di intaccare le proprietà a di integrità sono i seguenti:

- **Ladro di credenziali**, l'utilizzo di canali di comunicazione TLS 1.3 durante la fase di richiesta credenziale e erogazione servizio rendono impossibile per il ladro ottenere le CRED di un utente onesto.
- **Fornitore malevole/Ladro di servizi**, anche se il fornitore malevolo o il ladro di servizi ottiene le CRED di utenti onesti, non può usufruire dei servizi per via del primo step del protocollo di erogazione servizio che richiede di identificarsi come proprietari della chiave $Pk_h$ presente nel CRED (equazione 4).
- **Forgiatore di credenziali**, seguendo le specifiche del protocollo riesce a generare correttamente parte delle CRED, $(ID, Cert_f, Pk_h, credential)$, ma non è in grado di ottenere una firma ritenuta valida dal server.
- **Fornitore Forgiatore**, riesce a generare una CRED che viene ritenuta valida dai server, anche se l'ID non è generato secondo il protocollo. La debolezza è che i server non hanno modo di sapere se l'ID è creato come sha256($Cert_{cie}||r$), oppure sono solo 256 bit random.
- **Server Malevolo**, potrebbe provare ad attuare un "man in the middle attack", ovvero quando un utente si connette al Server Malevolo, il Server

Malevolo potrebbe provarsi a connettere a un terzo Server Onesto impersonando l'utente, questo fallisce perché il Server Malevolo non riesce ad ottenere $[(Pk_h, Sign_{Sk_h}(randTLS_U||randTLS_S||Pk_{ServerTerzo})]$ valido, e quindi il server terzo interromperebbe la connessione.

- **Insieme di utenti "onesti"**, anche se più utenti si accordano per usare una stessa $Pk_h$ e $Sk_h$, comunque al momento dell'erogazione delle loro rispettive CRED, riceverebbero CRED con campi ID differenti (il server non erogherebbe il servizio se vengono inviate CRED con ID differenti). Dato che la seconda componente della CRED è la firma fatta dal fornitore della prima componente, la prima parte della CRED non può essere modificata, senza invalidare la CRED. Inoltre anche se gli utenti possono scegliere i 128 random bits da concatenare ai propri certificati digitali CIE prima dell'hash, è impossibile ottenere due ID uguali, se così fosse avrebbero trovato una collisione in SHA256.

- **Prestatore di credenziali**, se ha intenzione di prestare anche la sua $Sk_h$, oltre alla CRED, allora riesce a battere il protocollo, altrimenti viene battuto come nel caso Fornitore malevole/Ladro di servizi

- **Combinazione di avversari**, il sistema proposto rimane resistente a combinazione di avversari, ammenoché tra questi non rientri il Fornitore Forgiatore.

## 3.4 Trasparenza

La validità di una CRED è in solo in parte pubblicamente verificabile, infatti non si può pubblicamente verificare che l'ID presente nella CRED sia prodotto seguendo il protocollo.

## 3.5 Efficienza/Usabilità

Il sistema non fa uso di algoritmi computazionalmente costosi per la generazione e verifica di credenziali, inoltre l'utilizzo di una singola coppia di chiavi per tutte le credenziali aumenta l'usabilità del sistema per l'utente. L'efficienza del sistema è migliorata anche grazie al fatto che l'utente invia al server solo le credenziali strettamente necessarie per accedere al servizio grazie all'atomicità delle credenziali.

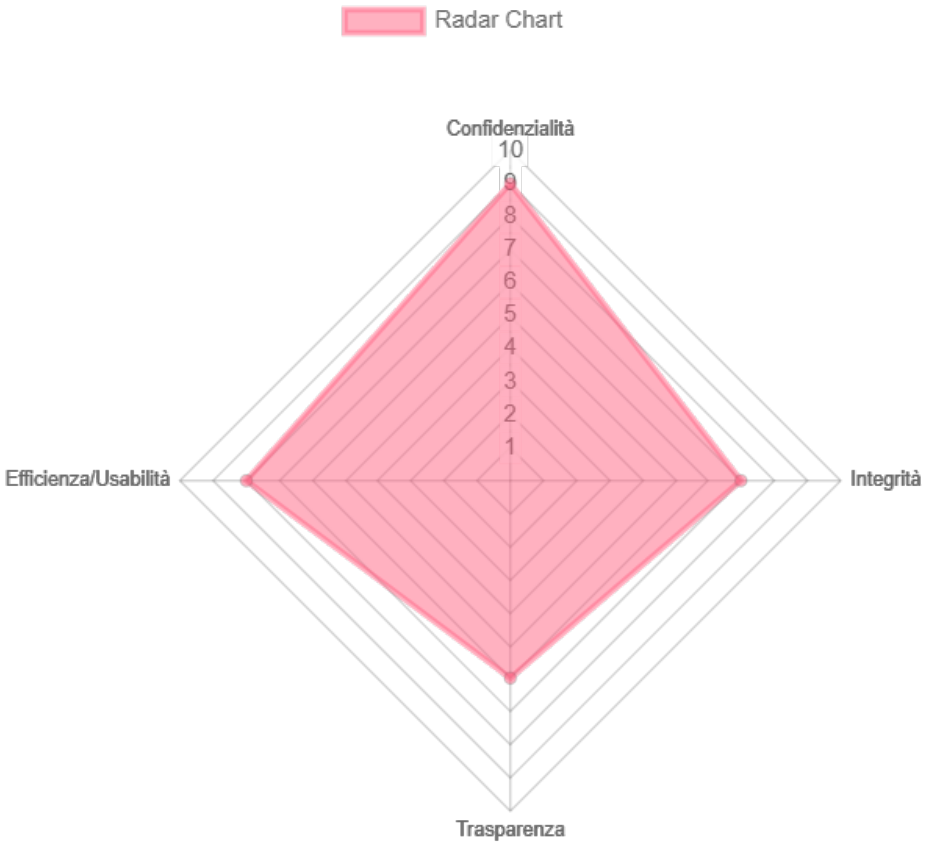


Figura 1: Radar chart rispetto le proprietà di confidenzialità, integrità, trasparenza ed efficienza/usabilità

# 4 Modifiche apportate rispetto alla prima consegna

La prima soluzione consegnata non soddisfaceva il requisito di confidenzialità, dato che il server riconosceva gli utenti che accedono ad un suo servizio, visto che richiedevamo l'identificazione mutua TSL anche per la connessione al server. Ora l'utente si identifica come proprietario di $Pk_h$ che non è riconducibile ad una persona. Inoltre è stato semplificato il protocollo, rimuovendo l'uso del merkle tree all'interno delle credenziali.

# 5 WP4

L'implementazione è stata realizzata usando Python per l'hosting locale dei server, per il parsing e per la chiamata dei comandi di OpenSSL. Per TLS invece, è stata usato il modulo Python SSL.

## 5.1 Inizializzazione dell'ambiente

Prima di poter utilizzare gli script python che implementano il protocollo descritto è necessario creare tutti i player in gioco con le loro relative informazioni. Prima di eseguire gli script bash è necessario assicurarsi che il valore della variabile *OPENSSL* sia impostata al percorso con la versione di Openssl da eseguire, di default viene utilizzata quella memorizzata all'interno del path del sistema.

### 5.1.1 Generazione CA e certificati digitali

Nell'ambiente creato, abbiamo bisogno di un CA trusted, che possiamo identificare come 'IPSZ', che per semplicità viene utilizzata sia per rilasciare la CIE sia per rilasciare i certificati ai fornitori e ai server che erogano un servizio.
Per generare la Certification Authority abbiamo innanzitutto generato una coppia di chiavi per la firma ECDSA utilizzando la curva ellittica prime256v1.

```
#!/bin/bash

CA="IPZS"
PRIVATE_KEY_FILE="CASk.pem"
PUBLIC_KEY_FILE="CAPk.pem"
# Path al file openssl da voler utilizzare
OPENSSL="openssl"

# Creazione della chiave pubblica e privata per la CA
# Genera chiave privata usando la curva prime256v1
$OPENSSL ecparam -name prime256v1 -out Private/prime256v1.pem
$OPENSSL genpkey -paramfile Private/prime256v1.pem -out Private/"
    $PRIVATE_KEY_FILE"
echo "Private key generated and saved to Private/$PRIVATE_KEY_FILE"

# Generate la corrispondente chiave pubblica
$OPENSSL pkey -in Private/"$PRIVATE_KEY_FILE" -pubout -out Private/
    "$PUBLIC_KEY_FILE"
echo "Public key generated and saved to Private/$PUBLIC_KEY_FILE"
```

Abbiamo eseguito il seguente comando per rilasciare un certificato digitale auto firmato, da usare come root trusted all'interno del nostro ambiente, in particolare facendo uso di un file di configurazione, che viene creato al momento dell'esecuzione dello script CA.sh e aggiungendo al certificato le estensioni necessarie, in particolare tramite l'estensione *basicConstraints: critical, CA:true* firmiamo il certificato della CA abilitandola a rilasciare altri certificati.

```
$OPENSSL req -new -x509 -days 365 -key Private/"$PRIVATE_KEY_FILE"
    -out "$CA"cert.pem -config config.cnf -extensions ext
```

Per creare la Certification Authority è necessario eseguire lo script CA.sh.

### 5.1.2 Generazione del Fornitore

Per creare un fornitore è necessario eseguire lo script Fornitore.sh che si occupa della generazione della coppia di chiavi da utilizzare per la firma ECDSA e allo stesso tempo si occupa di generare una richiesta per un certificato digitale contenente l'estensione *DNS.1 = localhost* necessaria per il funzionamento della simulazione del protocollo eseguita in locale e presentata successivamente. La coppia di chiavi viene generata nello stesso modo descritto precedentemente per la CA.
La generazione di una richiesta per un certificato digitale avviane tramite il seguente comando:

```
$OPENSSL req -new -key Private/Sk.pem -out Cert/request.pem -addext
    "subjectAltName = DNS:localhost"
```

Lo stesso script si occupa anche di far firmare la richiesta dalla CA per ottenere il certificato digitale da utilizzare. La firma della richiesta viene eseguita tramite il seguente comando:

```
$OPENSSL x509 -req -in ../$UTENTE/Cert/request.pem -CAserial serial
    -CA IPZScert.pem -CAkey Private/CaSk.pem -out ../"$UTENTE"/
    Cert/Fornitore_cert.pem -days 365 -extfile user_ext.cnf -
    extensions v3_ca
```

### 5.1.3 Generazione del Server

Per creare un server che eroga un servizio è necessario eseguire lo script Server.sh che esegue tutte le operazioni necessarie nello stesso modo descritto per il fornitore.

### 5.1.4 Generazione dell'utente

Per generare un utente è necessario eseguire lo script User.sh. Questo script si occupa di creare la chiave privata e pubblica associate alla CIE e il relativo certificato. Per semplicità le informazioni da inserire all'interno del certificato della CIE vengono richieste ad ogni esecuzione dello script tuttavia questo può essere sostituito utilizzando un file di configurazione contenente le informazioni del singolo utente. Lo script può essere utilizzato per generare un numero arbitrario di utenti semplicemente modificando il valore della variabile UTENTE

presente al suo interno. In accordo con quanto descritto all'interno del WP2 la coppia di chiavi $(Pk_h, Sk_h)$ non viene generata dallo script ma dal file client.py quando l'utente richiede per la prima volta una credenziale ad un fornitore, anche in questo caso per la loro generazione vengono utilizzati i seguenti comandi Openssl:

```
def generateClientKey(savingPath):

    Pk = os.path.join(savingPath,'Pkh.pem')
    Sk = os.path.join(savingPath,'Skh.pem')
    run_openssl(f"{KeyManager.OPENSSL} ecparam -genkey -name
    prime256v1 -out {Sk}")
    run_openssl(f"{KeyManager.OPENSSL} pkey -in {Sk} -pubout -out {
    Pk}")
    return Sk, Pk
```

dove viene utilizzata la curva ellittica prime256v1.

## 5.2 Generazione della Credenziale e verifica

In questa sezione presentiamo pezzi di codice usati per la generazione delle credenziali, in cui vengono utilizzati comandi OpenSSL.
Per generare la richiesta di credenziale sono necessari 128 random bits, per ottenerli usiamo il comando 'rand' di OpenSSL.

```
def generate_random_bits():
    """Genera i 128 bit della credential request usando il comando
    OPENSSL rand -hex 16"""
    command = ['openssl', 'rand', '-hex', '-out', 'random_bits.txt'
    , '16']  # 16 bytes = 128 bits
    subprocess.run(command, capture_output=True, text=True)
    """load file restituisce il contenuto del file"""
    return load_file('USER/user1/random_bits.txt')
```

Dopo aver creato la richiesta di credenziale

```
cred_request = {
        "Credential": credential,
        "certCie": cert_cie,
        "Pkh": pkh,
        "r": r
    }
```

la stessa viene inviata al fornitore di credenziali che procederà a generare una credenziale valida. Prima controlla la validità della CIE, confrontandola con quella dell'utente che ha iniziato la connessione TLS con identificazione mutua. Dopo la verifica del certificato CIE, concateniamo i dati e li firmiamo, con il comando OpenSSL

```
openssl dgst -sign FORNITORE_KEY -out signature.bin temp_file
```

dove FORNITORE_KEY è una variabile contente la chiave privata per la firma ECDSA del server, e temp_file è un file temporaneo da firmare che contiene la concatenazione di ID, certificato del fornitore, chiave pubblica h dell'user e la credenziale.

```
def generate_id(cie, r):
    '''Genera l'ID come sha256(cert_cie||r).'''
    return hashlib.sha256((cie + r).encode()).hexdigest()

def sign_CRED(id, certF, pkh, credential):
    """...ommettiamo il codice per generare temp_file"""
    '''Firma i dati concatenati.'''
    data = concatenate_data(id, certF, pkh, credential)
    print(f"Dati concatenati per la firma:\n{data}")

    with tempfile.NamedTemporaryFile(delete=False, mode='w', suffix
    ='.txt') as temp_file:
        temp_file.write(data)
        temp_file_path = temp_file.name
    try:
        # Firma i dati concatenati
        command = [
            'openssl', 'dgst', '-sign', FORNITORE_KEY, '-out',
            'signature.bin', temp_file_path
        ]
        subprocess.run(command, check=True)
        with open('signature.bin', 'rb') as signature_file:
            signature = signature_file.read()
        return signature.hex()
    finally:
        os.remove(temp_file_path)
        os.remove('signature.bin')
```

La credenziale valida ha quindi la seguente struttura:

```
CRED = {
            "ID": ID,
            "certF": certF,
            "Pkh": pkh,
            "Credential": credential,
            "sign": signature
        }
```

Quando un server riceve una credenziale per verificarne la validità esegue la seguente funzione, che al suo interno esegue il comando:

```
openssl verify -CAfile -TRUSTED_CA_PATH temp_cert_path
```

Per verificare che il certificato presente in CRED sia valido, successivamente esegue:

```
openssl x509 -pubkey -noout -in temp_cert_path
```

Per estrarre la chiave pubblica presente in certF, e successivamente verifica la validità della firma:

```
def verify_CRED(data,certF, signature, pubkey_path):
    """...omettiamo il codice per generare i temp_file, sig_file e
    temp_cert"""
    try:
        # verifica validita certificato
        command = [
                'openssl', 'verify', '-CAfile',
                TRUSTED_CA_PATH, temp_cert_path
        ]
        result = subprocess.run(command)
        if result.returncode !=0:
            return -1

        # estrai chiave pubblica da certF e inseriscilo in un file
        with open('pubkeyf.pem', 'w') as file:
            command_p = [
                'openssl', 'x509', '-pubkey', '-noout', '-in',
                temp_cert_path
            ]
            subprocess.run(command_p, stdout=file)

        # Comando OpenSSL per verificare la firma
        command = [
            'openssl', 'dgst', '-verify', 'pubkeyf.pem',
            '-signature', sig_file_path, temp_file_path
        ]
        # Esegue il comando e cattura l'output
        result = subprocess.run(command, capture_output=True,
                                text=True)

        # Stampa eventuali errori OpenSSL se la verifica fallisce
        if result.returncode != 0:
            print(f"OpenSSL error: {result.stderr}")

        # Restituisce True se la verifica ha successo
        return result.returncode == 0
```

## 5.3 Gestione della connessione TLS tramite python SSL

Per poter eseguire una simulazione del funzionamento del protocollo è stato necessario creare due server, uno che rappresenta il fornitore e l'altro che rappresenta il server da cui vogliamo un servizio, che fanno uso del protocollo TLS per comunicare in modo sicuro. Per realizzare i due server è stata realizzata una classe *TLSServer_base* che prende in ingresso le informazioni necessarie per creare un server, ovvero l'indirizzo IP su cui il server può essere raggiunto e la porta su cui si trova in ascolto. I rimanenti parametri vengono utilizzati dal modulo ssl per la gestione del protocollo tls, in particolare viene fornito il certificato del server e la chiave privata del server, la cartella contenente le Certification Auhtority fidate e un parametro *require_client_cert* tramite il quale è possibile il requisito di mutua autenticazione.

```
    def __init__(self, host, port, server_cert, server_key, ca_dir,
     require_client_cert=False):
        self.host = host
        self.port = port
        self.server_cert = server_cert
        self.server_key = server_key
        self.ca_dir = ca_dir

        self.context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER,
    openssl_executable=self.openssl_path)
        self.context.minimum_version = ssl.TLSVersion.TLSv1_3
        # Carica il certificato e la chiave privata del server
        print(self.server_cert, self.server_key)
        self.context.load_cert_chain(certfile=self.server_cert,
    keyfile=self.server_key)

        # Carica i certificati CA dalla directory
        self.load_ca_certificates(self.context, self.ca_dir)

        # Richiedi la verifica del certificato del client
        if require_client_cert:
            self.context.verify_mode = ssl.CERT_REQUIRED

        # Configurazione del socket server
        self.server_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
        self.server_socket.bind((self.host, self.port))
        self.server_socket.listen(1)
```

Durante la creazione del contesto ssl definiamo che il protocollo è eseguito lato server e richiediamo che il server accetta unicamente connessione TLS 1.3. Tramite l'istruzione *self.context.load_cert_chain(certfile=self.server_cert, keyfile=self.server_key)* carichiamo il certificato del server da inviare al client e la chiave segreta del server con cui firmarlo. Quando il server preso in considerazione è un fornitore il parametro *require_client_cert* è impostato a True questo significa che il client dovrà inviare il certificato digitale associato alla sua CIE e rilasciato da IPZS affinchè il fornitore accetti la richiesta di connessione, nel

caso in cui il client non invia il certificato oppure il certificato è stato rilasciato da un'ente differente da IPZS allora la connessione verrà chiusa. Nel caso in cui il server considerato è quello che deve erogare il servizio allora il parametro *require_client_cert* è impostato a False e di conseguenza sarà solo il client a richiedere il certificato digitale del server. In questo modo è stato realizzato quanto definito nel WP2 per quando riguarda la mutua autenticazione tra Fornitore e Client e l'autenticazione tra Server e Client.
La classe base *TLSServer_base* viene estesa dalle classi *Server* e *Fornitore*. In maniera del tutto analoga alla creazione del server è stato creato un client che utilizza TLS 1.3 per comunicare con il Fornitore quando vuole richiedere delle credenziali e con il server quando vuole utilizzare le credenziali per ottenere un servizio.

## 5.4 Parti del protocollo non implementate

L'implementazione proposta tramite l'utilizzo dei comandi base openssl e tramite l'utilizzo del modulo ssl hanno permesso la realizzazione della maggior parte del protocollo definito, fatta eccezione per l'equazione 4 definita nel WP2 $[(Pk_h, Sign_{Sk_h}(randTLS_U||randTLS_S||Pk_S)]$ che rappresenta lo step più importante per contrastare l'avversario server malevolo. Questa ulteriore operazione è semplicemente realizzabile tramite i comandi base openssl in quanto il tutto si riduce ad una semplice firma del messaggio $(randTLS_U||randTLS_S||Pk_S)$ da parte del client e da una successiva verifica da parte del server, tuttavia non è stato possibile implementare la prima operazione di firma a causa dell'impossibilità nell'ottenere i bit random scambiati da client e server durante la sessione tls.
I comandi da eseguire per eseguire la firma del messaggio nel caso in cui è possibile entrare facilmente in possesso dei bit random sono:

```
cat > message.txt <<EOL
(randTLS_U || randTLS_S || Pk_S)
EOL
# Firma del file message.txt utilizzando la chiave Sk_h
openssl dgst -sign CLIENT_SKH_KEY -out signature.bin message.txt

# Verifica della firma da parte del server
openssl dgst -verify CLIENT_PKH -signature signature.bin message.
    txt
```

## 5.5 Esempio di Esecuzione

Di seguito viene presentato un esempio di esecuzione partendo dalla creazione della CA, del fornitore, del server e dell'utente e viene presentata la sequenza di comandi da eseguire e quali script utilizzare per ottenere lo stesso risultato della simulazione.

### 5.5.1 Inizializzazione dell'ambiente

Di seguito viene presentato l'ordine dei comandi da eseguire per la creazione degli attori del sistema:

```
./CA.sh
./Fornitore.sh
./Server.sh
./User.sh
```

Output del comando CA.sh

```
Creazione del file di configurazione per la creazione della CA
-------------------------config.cnf---------------------------------
[req]
prompt = no
distinguished_name = dq
req_extensions = ext

[dq]
CN = IPZS
emailAddress = info@IPZS.it
O = IPZS
L = Rome
C = IT

[ext]
subjectAltName = DNS:*IPZS.it
basicConstraints = critical, CA:true
---------------------------------------------------------------------
Creazione del file per la gestione delle estensioni user_ext.cnf
-------------------------user_ext.cnf---------------------------------
[ v3_req ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = localhost

[ v3_ca ]
subjectAltName = @alt_names
basicConstraints = CA:FALSE
---------------------------------------------------------------------
Private key generated and saved to Private/CASk.pem
Public key generated and saved to Private/CAPk.pem
---------------------------------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            62:48:35:96:b6:21:54:2d:96:f4:12:65:07:e2:37:98:df:a2:75:6a
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = IPZS, emailAddress = info@IPZS.it, O = IPZS, L = Rome, C = IT
        Validity
            Not Before: Jul  8 15:13:35 2024 GMT
            Not After : Jul  8 15:13:35 2025 GMT
        Subject: CN = IPZS, emailAddress = info@IPZS.it, O = IPZS, L = Rome, C =
     IT
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:87:18:ab:f3:33:52:ee:c4:fa:c4:60:c8:8f:42:
                    5a:a0:05:49:59:ca:51:95:2f:30:be:72:aa:fc:60:
                    7d:30:88:d1:22:a9:c5:09:86:9d:3b:22:50:0c:04:
```

```
                    64:fb:19:41:f7:aa:84:d3:3a:a6:bf:b2:ac:ff:65:
                    0f:10:c9:87:dd
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:*IPZS.it
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                A9:9D:38:64:67:E5:6B:69:9D:6E:FB:F3:F3:DD:13:69:68:B4:20:64
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:20:7f:58:a1:3c:e7:67:3b:34:fd:e8:6f:1c:e8:19:
        23:31:6f:51:ef:f6:c9:81:1d:83:ad:9e:08:5a:9c:5b:eb:1a:
        02:21:00:ac:64:20:97:cf:e3:14:fe:e3:5c:27:76:2d:ab:41:
        f7:14:45:93:e9:0a:70:53:66:71:ab:08:a3:8a:b4:54:81
-----BEGIN CERTIFICATE-----
MIIB9zCCAZ2gAwIBAgIUYkg1lrYhVC2W9BJlB+I3mN+idWowCgYIKoZIzj0EAwIw
VzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w
CwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb21lMQswCQYDVQQGEwJJVDAeFw0yNDA3
MDgxNTEzMzVaFw0yNTA3MDgxNTEzMzVaMFcxDTALBgNVBAMMBElQW1MxGzAZBgkq
hkiG9w0BCQEWDGluZm9ASVBaUy5pdDENMAsGA1UECgwESVBaUzENMAsGA1UEBwwE
Um9tZTELMAkGA1UEBhMCSVQwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAASHGKvz
M1LuxPrEYMiPQlqgBUlZylGVLzC+cqr8YH0wiNEiqcUJhp07IlAMBGT7GUH3qoTT
Oqa/sqz/ZQ8QyYfdo0cwRTATBgNVHREEDDAKgggqSVBaUy5pdDAPBgNVHRMBAf8E
BTADAQH/MB0GA1UdDgQWBBSpnThkZ+VraZ1u+/Pz3RNpaLQgZDAKBggqhkjOPQQD
AgNIADBFAiB/WKE852c7NP3obxzoGSMxb1Hv9smBHYOtnghanFvrGgIhAKxkIJfP
4xT+41wndi2rQfcURZPpCnBTZnGrCKOKtFSB
-----END CERTIFICATE-----
```

Output del comando Fornitore.sh

```
Private key generated and saved to Private/Sk.pem
Public key generated and saved to Private/Pk.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Salerno
Locality Name (eg, city) []:Salerno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Comune di Salerno
Organizational Unit Name (eg, section) []:Anagrafe
Common Name (e.g. server FQDN or YOUR name) []:Anagrafe Salerno
Email Address []:anagrafeSalerno@uffici.it

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
-------------------------richiesta----------------------------------
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IT, ST = Salerno, L = Salerno, O = Comune di Salerno, OU =
     Anagrafe, CN = Anagrafe Salerno, emailAddress = anagrafeSalerno@uffici.it
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:36:df:e4:74:69:57:6a:4a:d4:17:26:df:6b:4c:
                    53:7a:94:fa:f7:eb:60:00:86:47:76:ae:3b:0b:8a:
```

```
                    ff:57:35:7f:7a:04:71:ef:64:b3:b6:8b:af:2b:46:
                    b2:1a:ad:14:ec:8b:bc:77:48:4c:91:71:31:42:6d:
                    8a:ba:de:90:9e
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            Requested Extensions:
                X509v3 Subject Alternative Name:
                    DNS:localhost
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:a2:a8:b1:f4:4c:1b:04:a6:ee:10:58:70:dc:
        c1:b3:aa:8c:e0:bc:df:fe:11:dc:57:34:41:01:35:3a:48:f2:
        cf:02:20:6c:2c:49:4d:82:dc:36:90:52:bb:ed:80:eb:96:eb:
        5c:94:89:05:fe:48:99:8f:f0:e9:43:49:92:f3:66:82:70
-----BEGIN CERTIFICATE REQUEST-----
MIIBiTCCAS8CAQAwgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQIDAdTYWxlcm5vMRAw
DgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkgU2FsZXJubzERMA8G
A1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNhbGVybm8xKDAmBgkq
hkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQwWTATBgcqhkjOPQIB
BggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AAhkd2rjsLiv9XNX96
BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeoCcwJQYJKoZIhvcNAQkOMRgw
FjAUBgNVHREEDTALgglsb2NhbGhvc3QwCgYIKoZIzj0EAwIDSAAwRQIhAKKosfRM
GwSm7hBYcNzBs6qM4Lzf/hHcVzRBATU6SPLPAiBsLElNgtw2kFK77YDrlutclIkF
/kiZj/DpQ0mS82aCcA==
-----END CERTIFICATE REQUEST-----
-----------------------------------------------------------------------
Invio della richiesta alla CA
Certificate request self-signature ok
subject=C = IT, ST = Salerno, L = Salerno, O = Comune di Salerno, OU = Anagrafe,
     CN = Anagrafe Salerno, emailAddress = anagrafeSalerno@uffici.it
Firma della richiesta dalla CA e generazione del certificato digitale completata
-----------------------------------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            42:e8:18:62:fb:20:4b:8d:dd:8b:f3:3f:86:74:f4:72:ae:bc:cd
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = IPZS, emailAddress = info@IPZS.it, O = IPZS, L = Rome, C = IT
        Validity
            Not Before: Jul  8 15:34:23 2024 GMT
            Not After : Jul  6 15:34:23 2034 GMT
        Subject: C = IT, ST = Salerno, L = Salerno, O = Comune di Salerno, OU =
     Anagrafe, CN = Anagrafe Salerno, emailAddress = anagrafeSalerno@uffici.it
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:36:df:e4:74:69:57:6a:4a:d4:17:26:df:6b:4c:
                    53:7a:94:fa:f7:eb:60:00:86:47:76:ae:3b:0b:8a:
                    ff:57:35:7f:7a:04:71:ef:64:b3:b6:8b:af:2b:46:
                    b2:1a:ad:14:ec:8b:bc:77:48:4c:91:71:31:42:6d:
                    8a:ba:de:90:9e
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:localhost
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                18:52:2C:EE:62:2E:D6:EB:5E:C4:BB:44:E4:DD:F5:2C:DA:47:1D:8C
            X509v3 Authority Key Identifier:
                A9:9D:38:64:67:E5:6B:69:9D:6E:FB:F3:F3:DD:13:69:68:B4:20:64
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:c9:50:d9:07:a6:bd:68:73:33:b1:c0:5b:e9:
```

```
        72:4e:86:66:82:c4:33:da:aa:24:ee:03:93:8d:45:fa:0b:55:
        51:02:20:7f:6b:ae:60:95:04:e7:30:c4:20:d1:54:3c:18:31:
        29:10:e1:9e:0d:64:ca:de:1c:da:d3:88:e5:ea:a3:b0:bb
-----BEGIN CERTIFICATE-----
MIICYTCCAgegAwIBAgITQugYYvsgS43di/M/hnT0cq68zTAKBggqhkjOPQQDAjBX
MQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL
BgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw
ODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI
DAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg
U2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh
bGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw
WTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA
hkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU
BgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu
1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw
CgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6
ClVRAiB/a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==
-----END CERTIFICATE-----
```

Output del comando Server.sh

```
Private key generated and saved to Private/Sk.pem
Public key generated and saved to Private/Pk.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Rome
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:At Your Service
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:AYS
Email Address []:info@ays.it

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
-------------------------richiesta----------------------------------
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IT, ST = Rome, L = Rome, O = At Your Service, CN = AYS,
     emailAddress = info@ays.it
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:23:50:4c:2f:fd:87:27:f7:12:e9:93:e8:16:20:
                    52:ee:43:27:af:b4:05:47:1c:b8:a3:66:74:5d:c3:
                    ce:31:24:0b:51:40:ed:da:e2:7c:35:70:2a:af:12:
                    88:15:c5:ef:e9:bc:0b:b8:29:0f:28:ad:4d:4e:e0:
                    6d:2f:a2:c7:24
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            Requested Extensions:
                X509v3 Subject Alternative Name:
                    DNS:localhost
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:ae:74:7b:0d:c3:5c:3d:53:7a:84:5f:ed:99:
```

```
        19:4b:fd:d0:d2:54:7c:c4:3e:2a:10:56:17:31:93:61:d7:4a:
        e6:02:20:72:31:50:13:ae:e2:56:9c:7f:33:3d:b3:80:9b:48:
        45:d0:9c:88:b9:d1:fe:f3:ab:a0:0a:9f:bd:3b:8d:1e:32
-----BEGIN CERTIFICATE REQUEST-----
MIIBUTCB+AIBADBvMQswCQYDVQQGEwJJVDENMAsGA1UECAwEUm9tZTENMAsGA1UE
BwwEUm9tZTEYMBYGA1UECgwPQXQgWW91ciBTZXJ2aWNlMQwwCgYDVQQDDANBWVMx
GjAYBgkqhkiG9w0BCQEWC2luZm9AYXlzLml0MFkwEwYHKoZIzj0CAQYIKoZIzj0D
AQcDQgAEI1BML/2HJ/cS6ZPoFiBS7kMnr7QFRxy4o2Z0XcPOMSQLUUDt2uJ8NXAq
rxKIFcXv6bwLuCkPKK1NTuBtL6LHJKAnMCUGCSqGSIb3DQEJDjEYMBYwFAYDVR0R
BA0wC4IJbG9jYWxob3N0MAoGCCqGSM49BAMCA0gAMEUCIQCudHsNw1w9U3qEX+2Z
GUv90NJUfMQ+KhBWFzGTYddK5gIgcjFQE67iVpx/Mz2zgJtIRdCciLnR/vOroAqf
vTuNHjI=
-----END CERTIFICATE REQUEST-----
-----------------------------------------------------------------------
Invio della richiesta alla CA
Certificate request self-signature ok
subject=C = IT, ST = Rome, L = Rome, O = At Your Service, CN = AYS, emailAddress =
     info@ays.it
Firma della richiesta dalla CA e generazione del certificato digitale completata
-----------------------------------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            77:0e:45:7c:f6:e0:b6:96:08:e6:dc:95:f7:da:fc:96:a1:a6:b3:6d
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = IPZS, emailAddress = info@IPZS.it, O = IPZS, L = Rome, C = IT
        Validity
            Not Before: Jul  8 15:35:35 2024 GMT
            Not After : Jul  6 15:35:35 2034 GMT
        Subject: C = IT, ST = Rome, L = Rome, O = At Your Service, CN = AYS,
     emailAddress = info@ays.it
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:23:50:4c:2f:fd:87:27:f7:12:e9:93:e8:16:20:
                    52:ee:43:27:af:b4:05:47:1c:b8:a3:66:74:5d:c3:
                    ce:31:24:0b:51:40:ed:da:e2:7c:35:70:2a:af:12:
                    88:15:c5:ef:e9:bc:0b:b8:29:0f:28:ad:4d:4e:e0:
                    6d:2f:a2:c7:24
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:localhost
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                21:E2:8C:92:51:F6:19:D0:E8:63:61:08:08:47:45:22:FA:B7:E4:ED
            X509v3 Authority Key Identifier:
                A9:9D:38:64:67:E5:6B:69:9D:6E:FB:F3:F3:DD:13:69:68:B4:20:64
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:f8:cb:b8:d7:73:db:88:cd:56:2f:27:88:ea:
        83:80:1d:db:9e:38:a2:3c:81:14:75:85:aa:f0:ae:af:bb:ae:
        85:02:20:55:d3:e1:dd:1a:de:98:f8:45:a9:fb:4b:1f:a3:bf:
        e4:c6:cc:e9:d6:00:01:45:39:fc:e0:2d:ea:b1:d4:25:c2
-----BEGIN CERTIFICATE-----
MIICKzCCAdGgAwIBAgIUdw5FfPbgtpYI5tyV99r8lqGms20wCgYIKoZIzj0EAwIw
VzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w
CwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb2llMQswCQYDVQQGEwJJVDAeFw0yNDA3
MDgxNTM1MzVaFw0zNDA3MDYxNTM1MzVaMG8xCzAJBgNVBAYTAklUMQ0wCwYDVQQI
DARSb2llMQ0wCwYDVQQHDARSb2llMRgwFgYDVQQKDA9BdCBZb3VyIFNlcnZpY2Ux
DDAKBgNVBAMMA0FZUzEaMBgGCSqGSIb3DQEJARYLaW5mb0BheXMuaXQwWTATBgcq
hkjOPQIBBggqhkjOPQMBBwNCAAQjUEwv/Ycn9xLpk+gWIFLuQyevtAVHHLijZnRd
w84xJAtRQO3a4nwlcCqvEogVxe/pvAu4KQ8orU1O4G0voscko2MwYTAUBgNVHREE
DTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUIeKMklH2GdDoY2EI
```

```
CEdFIvq35O0wHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQwCgYIKoZI
zj0EAwIDSAAwRQIhAPjLuNdz24jNVi8niOqDgB3bnjiiPIEUdYWq8K6vu66FAiBV
0+HdGt6Y+EWp+0sfo7/kxszp1gABRTn84C3qsdQ1wg==
-----END CERTIFICATE-----
```

Output del comando User.sh

```
Private key generated and saved to CIE/SkCIE.pem
Public key generated and saved to CIE/PkCIE.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Salerno
Locality Name (eg, city) []:Salerno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Utente1
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:000001
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
-------------------------richiesta----------------------------------
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IT, ST = Salerno, L = Salerno, O = Utente1, CN = 000001
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:e5:02:cb:74:86:aa:86:d0:5b:26:a6:6a:1a:04:
                    58:61:57:71:d2:e5:d4:0a:ba:46:0c:16:a6:54:eb:
                    57:d6:38:10:60:4e:c4:7d:b2:64:92:f7:23:bf:51:
                    71:a9:53:6f:c8:57:14:61:d1:c3:d3:69:3e:21:6d:
                    25:a2:d6:58:19
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            Requested Extensions:
                X509v3 Subject Alternative Name:
                    DNS:localhost
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:46:02:21:00:b7:eb:2b:d0:4d:8e:bb:c4:fd:32:8b:e9:6e:
        d8:d3:db:01:8a:04:e4:86:69:cc:41:61:a5:e6:76:d1:c3:c4:
        04:02:21:00:a6:c2:a3:5f:98:53:b3:42:f3:3a:ab:50:65:6b:
        67:09:05:50:99:20:d2:ab:32:88:77:11:1a:7d:3a:23:68:2a
-----BEGIN CERTIFICATE REQUEST-----
MIIBNzCB3QIBADBUMQswCQYDVQQGEwJJVDEQMA4GA1UECAwHU2FsZXJubzEQMA4G
A1UEBwwHU2FsZXJubzEQMA4GA1UECgwHVXRlbnRlMTEPMA0GA1UEAwwGMDAwMDAx
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE5QLLdIaqhtBbJqZqGgRYYVdx0uXU
CrpGDBamVOtX1jgQYE7EfbJkkvcjv1FxqVNvyFcUYdHD02k+IW0lotZYGaAnMCUG
CSqGSIb3DQEJDjEYMBYwFAYDVR0RBA0wC4IJbG9jYWxob3N0MAoGCCqGSM49BAMC
A0kAMEYCIQC36yvQTY67xP0yi+lu2NPbAYoE5IZpzEFhpeZ20cPEBAIhAKbCo1+Y
U7NC8zqrUGVrZwkFUJkg0qsyiHcRGn06I2gq
-----END CERTIFICATE REQUEST-----
------------------------------------------------------------------------
Invio della richiesta alla CA
Certificate request self-signature ok
```

```
subject=C = IT, ST = Salerno, L = Salerno, O = Utente1, CN = 000001
Firma della richiesta dalla CA e generazione del certificato digitale completata
-----------------------------------------------------------------------
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            02:12:4d:05:81:01:07:62:0a:74:f1:db:25:cd:c8:7b:b5:20:b3:fd
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = IPZS, emailAddress = info@IPZS.it, O = IPZS, L = Rome, C = IT
        Validity
            Not Before: Jul  8 15:36:45 2024 GMT
            Not After : Jul  6 15:36:45 2034 GMT
        Subject: C = IT, ST = Salerno, L = Salerno, O = Utente1, CN = 000001
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:e5:02:cb:74:86:aa:86:d0:5b:26:a6:6a:1a:04:
                    58:61:57:71:d2:e5:d4:0a:ba:46:0c:16:a6:54:eb:
                    57:d6:38:10:60:4e:c4:7d:b2:64:92:f7:23:bf:51:
                    71:a9:53:6f:c8:57:14:61:d1:c3:d3:69:3e:21:6d:
                    25:a2:d6:58:19
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:localhost
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                BC:01:75:6B:F2:C0:14:AA:64:8D:6A:A7:4A:B5:A2:C9:09:94:11:1F
            X509v3 Authority Key Identifier:
                A9:9D:38:64:67:E5:6B:69:9D:6E:FB:F3:F3:DD:13:69:68:B4:20:64
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:44:02:20:72:63:87:02:3e:32:b3:4e:92:6c:c8:d5:d6:1c:
        4b:c7:58:c2:6e:ea:2a:fb:b4:ea:98:1d:22:6d:14:ab:8a:88:
        02:20:16:f1:b4:d2:4f:cf:de:3b:cc:97:83:a3:72:46:98:56:
        1b:47:64:53:c3:9b:72:77:9d:c3:52:52:35:76:db:13
-----BEGIN CERTIFICATE-----
MIICDzCCAbagAwIBAgIUAhJNBYEBB2IKdPHbJc3Ie7Ugs/0wCgYIKoZIzj0EAwIw
VzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w
CwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb2llMQswCQYDVQQGEwJJVDAeFw0yNDA3
MDgxNTM2NDVaFw0zNDA3MDYxNTM2NDVaMFQxCzAJBgNVBAYTAklUMRAwDgYDVQQI
DAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRAwDgYDVQQKDAdVdGVudGUxMQ8w
DQYDVQQDDAYwMDAwMDEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATlAst0hqqG
0FsmpmoaBFhhV3HS5dQKukYMFqZU61fWOBBgTsR9smSS9yO/UXGpU2/IVxRh0cPT
aT4hbSWillgZo2MwYTAUBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAd
BgNVHQ4EFgQUvAF1a/LAFKpkjWqnSrWiyQmUER8wHwYDVR0jBBgwFoAUqZ04ZGfl
a2mdbvvz890TaWi0IGQwCgYIKoZIzj0EAwIDRwAwRAIgcmOHAj4ys06SbMjV1hxL
xljCbuoq+7TqmB0ibRSriogCIBbxtNJPz947zJeDo3JGmFYbR2RTw5tyd53DUlI1
dtsT
-----END CERTIFICATE-----
```

### 5.5.2 Avvio della simulazione

A questo punto possiamo avviare i due server. I server sono stati configurati in modo tale da essere eseguiti in locale (host=localhost) su due porte differenti, in particolare il Fornitore utilizza la porta 8840 e il server utilizza la porta 8888, per modificare le porte su cui avviare i server è necessario modificare gli script

python Fornitore.py e Server.py e di conseguenza anche il file Client.py specificando le nuove porte utilizzate per i server.
Il primo passo è quello di avviare il fornitore e il client per ottenere una credenziale. Ogni volta che si vuole ottenere una credenziale nuova è necessario avviare lo script python client.py passando come parametri il path alla cartella create con il comando User.sh e la stringa Fornitore per segnalare che siamo l'utente 1 e volgiamo collegarci ad un fornitore per richiedere una credenziale.

```
python3 Fornitore.py
```

```
python3 client.py User1 Fornitore
```

Nel Box sottostante possiamo vedere come il server del Fornitore è stato avviato con successo ed è stata eseguita la mutua autenticazione tramite TLS, in particolare il Fornitore ha estratto alcune delle informazioni dal certificato inviato dall'utente e si è messa in attesa di ricevere la credential request da parte dell'utente.

```
./Fornitore/Cert/Fornitore_cert.pem ./Fornitore/Private/Sk.pem
Caricato certificato CA: ./Fornitore/Trusted/IPZScert.pem
Fornitore in ascolto su localhost:8440...
Connessione accettata da ('127.0.0.1', 47742)
Subject: CN=000001,O=Utente1,L=Salerno,ST=Salerno,C=IT
Valid From: 2024-07-08 15:36:45
Valid Until: 2034-07-06 15:36:45
```

Nel Box sottostante possiamo invece osservare come il client si è connesso con successo al fornitore e sta richiedendo una certa credenziale, e.g: residente salerno.

```
Versione del protocollo: TLSv1.3
Connesso a localhost:8440
Digita la credenziale richiesta: residente salerno
```

Di seguito i messaggi visualizzati sul terminale del fornitore

```
./Fornitore/Cert/Fornitore_cert.pem ./Fornitore/Private/Sk.pem
Caricato certificato CA: ./Fornitore/Trusted/IPZScert.pem
Fornitore in ascolto su localhost:8440...
Connessione accettata da ('127.0.0.1', 45752)
Subject: CN=000001,O=Utente1,L=Salerno,ST=Salerno,C=IT
Valid From: 2024-07-08 15:36:45
Valid Until: 2034-07-06 15:36:45
('127.0.0.1', 45752) ha inviato la seguente richiesta di credenziali:
 {
    "Credential": "residente salerno",
    "certCie": "-----BEGIN CERTIFICATE-----\
     nMIICDzCCAbagAwIBAgIUAhJNBYEBB2IKdPHbJc3Ie7Ugs/0wCgYIKoZIzj0EAwIw\
     nVzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w\
     nCwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb211MQswCQYDVQQGEwJJVDAeFw0yNDA3\
```

```
     nMDgxNTM2NDVaFw0zNDA3MDYxNTM2NDVaMFQxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRAwDgYDVQQKDAdVdGVudGUxMQ8w\
     nDQYDVQQDDAYwMDAwMDEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATlAst0hqqG\
     n0FsmpmoaBFhhV3HS5dQKukYMFqZU61fWOBBgTsR9smSS9yO/UXGpU2/IVxRh0cPT\
     naT4hbSWillgZo2MwYTAUBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAd\
     nBgNVHQ4EFgQUvAF1a/LAFKpkjWqnSrWiyQmUER8wHwYDVR0jBBgwFoAUqZ04ZGfl\
     na2mdbvvz890TaWi0IGQwCgYIKoZIzj0EAwIDRwAwRAIgcmOHAj4ys06SbMjV1hxL\nx1jCbuoq+7
     TqmB0ibRSriogCIBbxtNJPz947zJeDo3JGmFYbR2RTw5tyd53DUlI1\ndtsT\n-----END
     CERTIFICATE-----\n",
12   "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
13   "r": "6d1969bc5eca513c3ec02aaab2a6b0ad\n"
14 }
15 E' stato generata la seguente CRED per l'utente:
16 {
17   "ID": "abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc",
18   "certF": "-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
     hnT0cq68zTAKBggqhkjOPQQDAjBX\
     nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
     nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
     nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
     nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
     nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
     nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA\
     nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
     nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
     nlutexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
     nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
     a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n",
19   "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
20   "Credential": "residente salerno",
21   "sign": "304402206519
     b8cc1ba1c2df93f477ff2be997dad32b284eaa9f0e6a9003e1394ce36c9602200dfefb8040607
      5725b8217488ff98ecb3f0ca6f5e2dcf12a39976b76b989c15c"
22 }
23 Connessione accettata da ('127.0.0.1', 53430)
24 Subject: CN=000001,O=Utente1,L=Salerno,ST=Salerno,C=IT
25 Valid From: 2024-07-08 15:36:45
26 ('127.0.0.1', 53430) ha inviato la seguente richiesta di credenziali:
27  {
28     "Credential": "isee 1800",
29     "certCie": "-----BEGIN CERTIFICATE-----\
     nMIICDzCCAbagAwIBAgIUAhJNBYEBB2IKdPHbJc3Ie7Ugs/0wCgYIKoZIzj0EAwIw\
     nVzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w\
     nCwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb211MQswCQYDVQQGEwJJVDAeFw0yNDA3\
     nMDgxNTM2NDVaFw0zNDA3MDYxNTM2NDVaMFQxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRAwDgYDVQQKDAdVdGVudGUxMQ8w\
     nDQYDVQQDDAYwMDAwMDEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATlAst0hqqG\
     n0FsmpmoaBFhhV3HS5dQKukYMFqZU61fWOBBgTsR9smSS9yO/UXGpU2/IVxRh0cPT\
     naT4hbSWillgZo2MwYTAUBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAd\
     nBgNVHQ4EFgQUvAF1a/LAFKpkjWqnSrWiyQmUER8wHwYDVR0jBBgwFoAUqZ04ZGfl\
     na2mdbvvz890TaWi0IGQwCgYIKoZIzj0EAwIDRwAwRAIgcmOHAj4ys06SbMjV1hxL\nx1jCbuoq+7
     TqmB0ibRSriogCIBbxtNJPz947zJeDo3JGmFYbR2RTw5tyd53DUlI1\ndtsT\n-----END
     CERTIFICATE-----\n",
30     "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
31     "r": "6d1969bc5eca513c3ec02aaab2a6b0ad\n"
32 }
33 E' stato generata la seguente CRED per l'utente:
34 {
```

```
    "ID": "abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc",
    "certF": "-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
     hnT0cq68zTAKBggqhkjOPQQDAjBX\
     nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
     nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
     nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
     nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
     nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
     nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA\
     nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
     nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
     n1utexLtE5N31lLNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
     nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
     a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n",
    "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
    "Credential": "isee 1800",
    "sign": "3045022032
     a6c1963f8ef84b5229e821a3b2f6634254091852de115b63cfaba55b6a184a022100b3d884ff
     5a4e4a60b7ead5a24ca9b101d6b6abe786a72c8794cb538685ab5118"
}
```

Di seguito i messaggi visualizzati sul terminale del client dopo aver richiesto le credenziali "residente salerno" e "isee 1800" richieste dal server per ottenere il servizio

```
Caricato certificato CA: ./User1/Trusted/IPZScert.pem
Versione del protocollo: TLSv1.3
Connesso a localhost:8440
Digita la credenziale richiesta: residente salerno
E' stata inviata al server la seguente richiesta di credenziali:
 {
    "Credential": "residente salerno",
    "certCie": "-----BEGIN CERTIFICATE-----\
     nMIICDzCCAbagAwIBAgIUAhJNBYEBB2IKdPHbJc3Ie7Ugs/0wCgYIKoZIzj0EAwIw\
     nVzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w\
     nCwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb211MQswCQYDVQQGEwJJVDAeFw0yNDA3\
     nMDgxNTM2NDVaFw0zNDA3MDYxNTM2NDVaMFQxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRAwDgYDVQQKDAdVdGVudGUxMQ8w\
     nDQYDVQQDDAYwMDAwMDEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATlAst0hqqG\
     n0FsmpmoaBFhhV3HS5dQKukYMFqZU61fWOBBgTsR9smSS9yO/UXGpU2/IVxRh0cPT\
     naT4hbSWillgZo2MwYTAUBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAd\
     nBgNVHQ4EFgQUvAF1a/LAFKpkjWqnSrWiyQmUER8wHwYDVR0jBBgwFoAUqZ04ZGfl\
     na2mdbvvz890TaWi0IGQwCgYIKoZIzj0EAwIDRwAwRAIgcmOHAj4ys06SbMjV1hxL\nx1jCbuoq+7
     TqmB0ibRSriogCIBbxtNJPz947zJeDo3JGmFYbR2RTw5tyd53DUlI1\ndtsT\n-----END
     CERTIFICATE-----\n",
    "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
    "r": "6d1969bc5eca513c3ec02aaab2a6b0ad\n"
}
Il fornitore ha fornito la seguente credenziale:
{'ID': 'abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc', 'certF
     ': '-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
     hnT0cq68zTAKBggqhkjOPQQDAjBX\
     nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
     nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
     nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
     nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
     nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
```

```
     nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN61Pr362AA\
     nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
     nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
     n1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
     nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
     a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n', '
     Pkh': '-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n', 'Credential': 'residente salerno', 'sign': '304402206519b8cc1ba1
     c2df93f477ff2be997dad32b284eaa9f0e6a9003e1394ce36c9602200dfefb80406075725b
     8217488ff98ecb3f0ca6f5e2dcf12a39976b76b989c15c'}
----------------------------------------------------------------------
Caricato certificato CA: ./User1/Trusted/IPZScert.pem
Versione del protocollo: TLSv1.3
Connesso a localhost:8440
Digita la credenziale richiesta: isee 1800
E' stata inviata al server la seguente richiesta di credenziali:
 {
    "Credential": "isee 1800",
    "certCie": "-----BEGIN CERTIFICATE-----\
     nMIICDzCCAbagAwIBAgIUAhJNBYEBB2IKdPHbJc3Ie7Ugs/0wCgYIKoZIzj0EAwIw\
     nVzENMAsGA1UEAwwESVBaUzEbMBkGCSqGSIb3DQEJARYMaW5mb0BJUFpTLml0MQ0w\
     nCwYDVQQKDARJUFpTMQ0wCwYDVQQHDARSb211MQswCQYDVQQGEwJJVDAeFw0yNDA3\
     nMDgxNTM2NDVaFw0zNDA3MDYxNTM2NDVaMFQxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRAwDgYDVQQKDAdVdGVudGUxMQ8w\
     nDQYDVQQDDAYwMDAwMDEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATlAst0hqqG\
     n0FsmpmoaBFhhV3HS5dQKukYMFqZU61fWOBBgTsR9smSS9yO/UXGpU2/IVxRh0cPT\
     naT4hbSWillgZo2MwYTAUBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAd\
     nBgNVHQ4EFgQUvAFla/LAFKpkjWqnSrWiyQmUER8wHwYDVR0jBBgwFoAUqZ04ZGfl\
     na2mdbvvz890TaWi0IGQwCgYIKoZIzj0EAwIDRwAwRAIgcmOHAj4ys06SbMjV1hxL\nx1jCbuoq+7
     TqmB0ibRSriogCIBbxtNJPz947zJeDo3JGmFYbR2RTw5tyd53DUlI1\ndtsT\n-----END
     CERTIFICATE-----\n",
    "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
    "r": "6d1969bc5eca513c3ec02aaab2a6b0ad\n"
}
Il fornitore ha fornito la seguente credenziale:
{'ID': 'abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc', 'certF
     ': '-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
     hnT0cq68zTAKBggqhkjOPQQDAjBX\
     nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
     nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
     nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
     nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
     nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
     nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN61Pr362AA\
     nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
     nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
     n1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
     nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
     a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n', '
     Pkh': '-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n', 'Credential': 'isee 1800', 'sign': '3045022032
     a6c1963f8ef84b5229e821a3b2f6634254091852de115b63cfaba55b6a184a022100b3d8 84
     ff5a4e4a60b7ead5a24ca9b101d6b6abe786a72c8794cb538685ab5118'}
```

A questo punto l'utente è in possesso delle due credenziali "residente salerno" e "isee 1800" che nella nostra simulazione sono richieste dal server per erogare il servizio. Passiamo quindi ad avviare il Server

```
python3 Server.py
```

e il nostro client per collegarsi al server, tramite il parametro *Server*

```
python3 client.py User1 Server
```

Di seguito l'output del terminale collegato al Server

```
Server/Cert/Server_cert.pem Server/Private/Sk.pem
Caricato certificato CA: Server/Trusted/IPZScert.pem
Server in ascolto su localhost:8888...
Connessione accettata da ('127.0.0.1', 53376)
Credenziali richiesta all'utente ['residente salerno', 'isee 1800']


La firma della seguente credenziale e' valida e il fornitore e' Trusted:
{'ID': 'abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc', 'certF
    ': '-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
    hnT0cq68zTAKBggqhkjOPQQDAjBX\
    nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
    nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
    nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
    nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
    nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
    nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
    nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA\
    nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
    nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
    n1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
    nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
    a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n', '
    Pkh': '-----BEGIN PUBLIC KEY-----\
    nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
    itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
    -----\n', 'Credential': 'residente salerno', 'sign': '304402206519
    b8cc1ba1c2df93f477ff2be997dad32b284eaa9f0e6a9003e1394ce36c9602200dfefb80406
    075725b8217488ff98ecb3f0ca6f5e2dcf12a39976b76b989c15c'}


La firma della seguente credenziale e' valida e il fornitore e' Trusted:
{'ID': 'abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc', 'certF
    ': '-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
    hnT0cq68zTAKBggqhkjOPQQDAjBX\
    nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
    nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
    nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
    nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
    nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
    nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
    nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA\
    nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
    nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
    n1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
    nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
    a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n', '
    Pkh': '-----BEGIN PUBLIC KEY-----\
    nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
    itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
    -----\n', 'Credential': 'isee 1800', 'sign': '3045022032
    a6c1963f8ef84b5229e821a3b2f66342540918 52de115b63cfaba55b6a184a022100b3d 884
    ff5a4e4a60b7ead5a24ca9b101d6b6abe786a72c8794cb538685ab5118'}
Servizio erogato con successo
```

Di seguito l'output del terminale collegato al client

```
Caricato certificato CA: ./User1/Trusted/IPZScert.pem
Versione del protocollo TSL concordata: TLSv1.3
Connesso a localhost:8888
Il server ha richiesto le seguenti credenziali per accedere al servizio ['
    residente salerno', 'isee 1800']
Invio delle seguenti credenziali al server {
    "ID": "abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc",
    "certF": "-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
     hnT0cq68zTAKBggqhkjOPQQDAjBX\
     nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
     nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
     nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
     nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
     nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
     nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA\
     nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
     nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
     n1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
     nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
     a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n",
    "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
    "Credential": "residente salerno",
    "sign": "304402206519
     b8cc1ba1c2df93f477ff2be997dad32b284eaa9f0e6a9003e1394ce36c9602200dfefb804060
     75725b8217488ff98ecb3f0ca6f5e2dcf12a39976b76b989c15c"
}
Invio delle seguenti credenziali al server {
    "ID": "abb589442b621d0bcfa0199f9fba6ff1f334957add62b02a738251c126ef72fc",
    "certF": "-----BEGIN CERTIFICATE-----\nMIICYTCCAgegAwIBAgITQugYYvsgS43di/M/
     hnT0cq68zTAKBggqhkjOPQQDAjBX\
     nMQ0wCwYDVQQDDARJUFpTMRswGQYJKoZIhvcNAQkBFgxpbmZvQElQWlMuaXQxDTAL\
     nBgNVBAoMBElQWlMxDTALBgNVBAcMBFJvbWUxCzAJBgNVBAYTAklUMB4XDTI0MDcw\
     nODE1MzQyM1oXDTM0MDcwNjE1MzQyM1owgaUxCzAJBgNVBAYTAklUMRAwDgYDVQQI\
     nDAdTYWxlcm5vMRAwDgYDVQQHDAdTYWxlcm5vMRowGAYDVQQKDBFDb211bmUgZGkg\
     nU2FsZXJubzERMA8GA1UECwwIQW5hZ3JhZmUxGTAXBgNVBAMMEEFuYWdyYWZlIFNh\
     nbGVybm8xKDAmBgkqhkiG9w0BCQEWGWFuYWdyYWZlU2FsZXJub0B1ZmZpY2kuaXQw\
     nWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAQ23+R0aVdqStQXJt9rTFN6lPr362AA\
     nhkd2rjsLiv9XNX96BHHvZLO2i68rRrIarRTsi7x3SEyRcTFCbYq63pCeo2MwYTAU\
     nBgNVHREEDTALgglsb2NhbGhvc3QwCQYDVR0TBAIwADAdBgNVHQ4EFgQUGFIs7mIu\
     n1utexLtE5N31LNpHHYwwHwYDVR0jBBgwFoAUqZ04ZGfla2mdbvvz890TaWi0IGQw\
     nCgYIKoZIzj0EAwIDSAAwRQIhAMlQ2QemvWhzM7HAW+lyToZmgsQz2qok7gOTjUX6\nC1VRAiB/
     a65glQTnMMQg0VQ8GDEpEOGeDWTK3hza04jl6qOwuw==\n-----END CERTIFICATE-----\n",
    "Pkh": "-----BEGIN PUBLIC KEY-----\
     nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET8zmnDn6Qm6ekfi3es5dXcf0Eqa1\n+Nmt/
     itZJeUg364UPdgOFqakIcEtKpI6FZlsr7UFoJiiAl5svDxqe237hg==\n-----END PUBLIC KEY
     -----\n",
    "Credential": "isee 1800",
    "sign": "3045022032
     a6c1963f8ef84b5229e821a3b2f6634254091852de115b63cfaba55b6a184a022100b3d884ff
     5a4e4a60b7ead5a24ca9b101d6b6abe786a72c8794cb538685ab5118"
}
Hai ottenuto l'accesso al tuo servizio
```

La simulazione eseguita considera il caso in cui tutti gli attori coinvolti sono onesti, è possibile verificare anche il funzionamento del protocollo in alcune

delle situazioni descritte durante l'analisi del protocollo nel WP3, in particolare è stato testata la non erogazione del servizio nel caso in cui una delle credenziali è stata alterata, nel caso in cui due utenti mettono insieme le proprie credenziali per ottenere un servizio a cui il singolo utente non potrebbe avere accesso e infine è stato simulato il caso in cui l'utente non invia il proprio certificato digitale e quindi il Fornitore interrompe la connessione.