**DOMANDE DA FARE AL PROF – PARTE 1**

1. **N.B. questi comandi li mettiamo nella relazione ma non nello script, in quanto necessita di una conferma manuale da parte del prof e non so se gli va bene oppure vuole tutto automatico lanciando un semplice comando**
   1. **Possiamo consegnare già script belli e fatti con i certificati già pronti, basta che poi mostriamo i comandi nella relazione**
2. **Dobbiamo scrivere anche i comandi relativi all’implementazione della parte di handshake di TLS? Ossia scambio di chiavi DH, firma ECDSA?  
   Oppure semplicemente avviare la connessione tramite s\_client e s\_server?  
   Oppure ancora, far partire la connessione TLS tramite s\_client e s\_server ma scrivere semplicemente i comandi che TLS usa durante la fase di handshake (comandi relativi allo scambio di chiavi DH e firma della stringa casuale).**
   1. **Non c’è bisogno**
3. **La firma della richiesta di ottenimento delle credenziali abbiamo detto che viene fatta tramite una chiave segreta interna alla CIE alla quale nessuno può accedervi e questa necessita dell’inserimento successivo del PIN da parte dell’utente per “confermare” la firma, ossia confermiamo noi, possessori del PIN associato che stiamo effettuando noi la richiesta dell’ottenimento di certe credenziali.  
   Su openssl, la firma tramite PIN la effettuiamo tramite una semplice chiave segreta generata da parametri di una curva ellittica nonostante sia specificato che il pin sia di 10 cifre decimali?** 
   1. **Basta fare uno script che simula la carta, questo ha al proprio interno il pin per permettere la firma e conosce dove si trova il file che ha la chiave segreta di firma. Quando viene lanciato lo script e come argomento ha sia il PIN che l’hash del messaggio, ((PIN,hash)) controlla se il pin immesso dall’utente è quello presente all’interno dello script: se è corretto effettua la procedura di firma ECDSA Sign(PIN,hash) usando come chiave di firma la chiave segreta presente nella CIE (presente nello script) e appunto firma il messaggio hash = sha256(m), se invece non è corretto abortisce la connessione.**
4. **Come simuliamo il fatto che l’autorità ci rilasci le credenziali? Tramite semplice file che contiene “una chiave segreta” che rappresenta le credenziali?**
   1. **Si tramite semplice file**
5. **Dobbiamo rappresentare l’intero flusso di esecuzione, ossia effettuare passo dopo passo come avviene l’intera procedura di comunicazione client🡪autorità e client🡪server oppure possiamo fare che mostriamo i comandi una sola volta e poi per le altre volte non li riscriviamo?  
   Ad esempio, per dare il certificato digitale della CIE al client c’è tutta la procedura di: generazione della richiesta da parte del client, creazione di una certa autorità fidata che autocertifica i certificati e rilascio effettivo del certificato.  
   Dobbiamo fare questa procedura pure per l’autorità di rilascio delle credenziali (analogamente per il server) oppure no?**
   1. **Si, tutto**
6. **E’ corretto far si che ci sia una sola autorità affidabile che rilascia i certificati oppure no? (nel mio caso ho creato un’unica autorità affidabile (IPZS) che rilascia i certificati sia al client (tramite la CIE) e all’autorità per dimostrare di essere affidabile quando il client la contatta).**
   1. **Possiamo fare questa assunzione**
7. **Dobbiamo interfacciarci col browser oppure il tutto direttamente da terminale?** 
   1. **Possiamo fa anche solo da terminale**
8. **Dobbiamo simularlo l’invio dei file? Oppure semplicemente possiamo simulare il tutto tramite dei semplici file di testo cifrati e poi da decifrare per ottenerne il contenuto?**
9. **In TLS le chiavi che vengono usate per DH sono diverse da quelle che client e server hanno nei loro certificati? Ossia la PK della CIE e la PK dell’autorità di rilascio delle credenziali**
   1. **Sono chiavi diverse (effimere)**

**DOMANDE DA FARE AL PROF – PARTE 2**

1. Dobbiamo tenere in considerazione la presenza di eventuali avversari?

**COMUNICAZIONE CLIENT – AUTORITA’ RILASCIO CREDENZIALI**

1. **Generazione certificati CLIENT e AUTORITA’**
   1. **Il certificato del client glielo firma l’IPZS e quindi devo creare questa autorità suprema affidabile che si autofirma i certificati.**
      1. **E’ necessario che il client effettui prima la richiesta di ottenimento del certificato all’IPZS (pre-game).**
   2. **Il certificato dell’autorità glielo firma sempre la stessa autorità che ha firmato il certificato al client ossia la IPZS.**

**a.** Dobbiamo fare le seguenti cose perché prima di tutto dobbiamo far si che l’IPZS possa rilasciarci il certificato. Per farlo, dobbiamo necessariamente, lato client, effettuare una richiesta di ottenimento del certificato all’IPZS (tipo noi che andiamo al comune e diamo tutti i nostri dati per farci rilasciare la CIE e questi dati saranno inseriti nel certificato digitale della CIE).

openssl ecparam -name prime256v1 -out prime256v1.pem //per generare i parametri con curva ellittica

openssl genpkey -paramfile prime256v1.pem -out clientsec\_key.pem //per generare la chiave pubblica e privata

openssl pkey -in clientsec\_key.pem -pubout -out clientpub\_key.pem //per estrarre la chiave pubblica dal precedente file

Necessitiamo di un file di configurazione **openssl.cnf**, utile per la generazione dei certificati (usiamo quello di default ma avremmo potuto crearne anche uno nostro).  
Nel nostro caso, mostreremo l’utilizzo di questo comando ma avremo il file di richiesta già compilato per questioni di praticità

openssl req -new -key clientsec\_key.pem -out clientrequest.pem -config openssl.cnf //per effettuare la richiesta di ottenimento di un certificato

openssl req -in clientrequest.pem -text //se vogliamo leggerlo

--- Dopodiché viene la parte di autogenerazione e approvazione dei certificati da parte dell’IPZS.

Necessitiamo prima di generare una chiave privata dell’IPZS usando sempre il comando:

openssl genpkey -paramfile prime256v1.pem -out IPZSsec\_key.pem //generazione SK IPZS

openssl req -new -x509 -days 365 -key IPZSsec\_key.pem -out IPZScert.pem -config openssl.cnf

openssl x509 -in IPZScert.pem -text //per leggere il file

-Per una migliore organizzazione dei file creiamo diverse cartelle:

mkdir demoCA //contiene la configurazione dell’IPZS

mkdir demoCA/private //contiene la chiave privata

mkdir demoCA/newcerts //contiene i certificati emessi da IPZS

dobbiamo adesso creare due file vuoti che servono per inizializzare il database dei certificati emessi:

touch demoCA/index.txt demoCA/serial

Adesso dobbiamo inizializzare adesso il file serial inizializzando quello che è il contatore dei certificati:

echo "00" >> demoCA/serial

quindi in demoCA devo metterci il certificato autofirmato dell’autorità:

cp IPZScert.pem demoCA

e in demoCA/private metterci la chiave privata dell’autorità:

cp IPZSsec\_key.pem demoCA/private

- Adesso dobbiamo passare al rilascio vero e proprio del certificato da parte dell’IPZS al client che ne ha fatto la richiesta.

Prima di tutto, cambiamo il nome della chiave privata dell’ipzs in cakey.pem con:

mv demoCA/private/IPZSsec\_key.pem demoCA/private/cakey.pem //obbligatorio avere come nome cakey.pem

mv demoCA/private/IPZScert.pem demoCA/private/cacert.pem //obbligatorio avere come nome cacert.pem

dopodiché:

Set-Content .\demoCA\serial "00"

Get-Content .\demoCA\serial

New-Item -Path .\demoCA\index.txt -ItemType File -Force

e infine il seguente comando

openssl ca -in clientrequest.pem -out clientcert.pem -policy policy\_anything -config openssl.cnf

Adesso quindi è stato generato il certificato della CIE al client da parte dell’IPZS.

**b.** possiamo assumere che gli stessi comandi li faccia anche l’autorità di rilascio delle credenziali per ottenere un certificato affidabile fornito da qualcuno di affidabile (un’altra autorità affidabile che eroga i certificati (Da chiedere al prof)

Per adesso lo faccio lo stesso perché se qualcosa già me lo trovo fatto.  
Supponiamo che i certificati dell’autorità di rilascio delle credenziali li dia sempre l’IPZS per evitare di dover fare un’altra autorità sicura (che in realtà potrebbe non esserlo data la struttura del progetto).  
Questa volta è quindi l’autorità di rilascio delle credenziali a richiedere il certificato per dimostrare di essere affidabile quando il client la contatta.  
Di conseguenza i passaggi da fare sono gli stessi di prima, ossia l’autorità deve effettuare una richiesta di ottenimento del certificato all’IPZS e quindi:

openssl genpkey -paramfile prime256v1.pem -out credcasec\_key.pem //per generare la SK dell’autorità di rilascio delle credenziali

openssl pkey -in credcasec\_key.pem -pubout -out credcapub\_key.pem //per estrarre la PK dal precedente file

Questa volta andiamo ad usare un file di configurazione diverso, settato appositamente per l’autorità di rilascio delle credenziali chiamato **caconfig.cnf** e adesso l’autorità di rilascio delle credenziali effettuerà la richiesta di ottenimento del certificato che verrà poi firmato da parte dell’IPZS (assunzione).

openssl req -new -key credcasec\_key.pem -out credcarequest.pem -config caconfig.cnf

openssl req -in credcarequest.pem -text //se vogliamo leggerlo

Questa richiesta dobbiamo adesso farla approvare dall’autorità affidabile, ossia IPZS tramite:

openssl ca -in credcarequest.pem -out credcacert.pem -policy policy\_anything -config openssl.cnf

Adesso quindi, sia client che autorità di rilascio delle credenziali hanno il proprio certificato firmato dall’autorità affidabile IPZS e da qui può partire la connessione (scambio di dati) tra loro due.

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

1. **Effettuare quindi la prima parte di TLS in cui avviene lo scambio di chiavi Diffie-Hellman andando ad usare il setting delle curve ellittiche**

Prima di tutto dobbiamo stabilire una connessione TLS tra client e autorità che funzionerà da server.  
Ora, il modo più veloce sarebbe stabilire una connessione con gli appositi comandi della libreria OpenSSL **s\_client** e **s\_server**, che Q rendono la connessione affidabile tramite proprio i certificati che abbiamo creato prima.   
D’altro canto però poiché TLS ha una prima fase di handshake in cui viene effettuato lo scambio di chiavi DH con annessa firma della stringa randomica tramite un qualsiasi protocollo di firma (usiamo ECDSA) secondo me dobbiamo lo stesso mostrare i comandi ma poi la connessione vera e propria la stabiliamo direttamente con **s\_client e s\_server**.

**Connessione client tramite s\_client e server tramite s\_server**

Prima di tutto è necessario che il client si colleghi al server, ossia all’autorità di rilascio delle credenziali per poter ottenere certe credenziali in relazione a quanto richiesto e presente sul proprio certificato digitale.  
Quindi il server si identificherà e stessa cosa farà il client inviandogli il proprio certificato ossia il certificato digitale della CIE.

Prima di tutto stabiliamo che l’autorità di rilascio delle credenziali sia il server che comunicherà con chi effettua certe richieste inviandogli il proprio certificato digitale (credcacert.pem) ritenuto affidabile in quanto proviene dall’IPZS (ritenuto affidabile).

Necessitiamo però che sia anche il client ad identificarsi e per questo dobbiamo creare una nuova cartella “**Trusted**” in cui all’interno andranno tutti i certificati affidabili del client.  
Per fare ciò, dobbiamo creare un file unico che ha tutti i certificati affidabili (perché windows è stronzo ma funziona anche su linux) ossia il proprio (quello della CIE rilasciato dall’IPZS) e quello dell’IPZS.  
Questo file lo chiamiamo “**client\_certbundle.pem**”.  
Avviando il server, gli diciamo che tutti i documenti presenti in trusted sono affidabili e quindi può dormire sogni tranquilli quando il client si connette al server.  
Per avviare il server:

openssl s\_server -cert credcacert.pem -key credcasec\_key.pem -port 8899 -Verify 5 -CAfile Trusted/ client\_certbundle.pem

Dopodiché, per far connettere il client al server eseguiamo il seguente comando:

openssl s\_client -cert clientcert.pem -key clientsec\_key.pem -connect 127.0.0.1:8899 -CAfile Trusted/client\_certbundle.pem -verify\_return\_error

**Simulazione firma digitale tramite CIE**

Dopo che l’utente si è connesso al server correttamente stabilendo una connessione TLS, quello che adesso deve fare è richiedere certe credenziali.  
Ricordiamo che per effettuare la richiesta di ottenimento delle credenziali, è necessario che l’utente sia in possesso della propria CIE e del proprio PIN per effettuare la firma ECDSA del messaggio hash tramite la funzione Sign(PIN,hash) usando una chiave segreta di firma presente all’interno della CIE (e che nessuno può vedere o toccare).  
Supponiamo quindi che all’utente, dopo aver poggiato la propria CIE su un lettore NFC, gli venga chiesto di inserire il proprio PIN per permettere la firma ECDSA del messaggio hash = sha256(m) (messaggio in cui è semplicemente scritto “Richiesta ottenimento credenziali”).  
L’utente quindi, digitando il PIN, se PIN corrisponde a uno specifico PIN memorizzato nella CIE, restituisce una firma ECDSA del messaggio m di cui hash ne rappresenta lo SHA256, e restituisce un errore altrimenti.

Immagine che contiene testo, schermata

Descrizione generata automaticamenteIl tutto è stato effettuato producendo uno script in c chiamato “**pin\_check.c**” eseguibile su windows effettuando prima la compilazione con **gcc -o pin\_check pin\_check.c** e successivamente eseguendolo con **./pin\_check PIN (di dieci cifre)** (nell’esempio **PIN = 1234567890** (il tutto sarà presente in un unico script che chiama questi due comandi).  
Lo script è strutturato nel seguente modo: PIN equivale proprio al pin che la CIE si aspetta che l’utente digiti.  
  
  
Dopo aver effettuato alcuni controlli di routine, salviamo in una variabile il pin digitato dall’utente e tramite una strcmp (in questa fase assumiamo che la comunicazione sia sicura e quindi senza la presenza di avversari che potrebbero esibire dei timing attack) confrontiamo se il pin presente nella CIE corrisponde al pin inserito dall’utente.  
Se affermativo, allora procede con l’effettuare l’hash del messaggio m salvandolo in un file a parte con il seguente comando:

openssl dgst -sha256 test.txt > outputsha256.txt

e infine effettua la firma ECDSA, sfruttando la chiave segreta di firma presente in **clientsec\_key.pem** del messaggio hashato contenuto in **outputsha256.txt** producendo un file .bin in cui è contenuta la firma con il seguente comando:

openssl dgst -sign clientsec\_key.pem -out signature.bin outputsha256.txt

Se invece il pin inserito dall’utente NON corrisponde a quello presente nella CIE allora viene lanciato un messaggio di errore invitando l’utente a riprovare l’inserimento del PIN.

A questo punto quindi, l’autorità di rilascio delle credenziali, una volta che ha ricevuto questa richiesta firmata, effettuerà la verifica della firma col seguente comando:

openssl dgst -verify clientpub\_key.pem -signature signature.bin outputsha256.txt

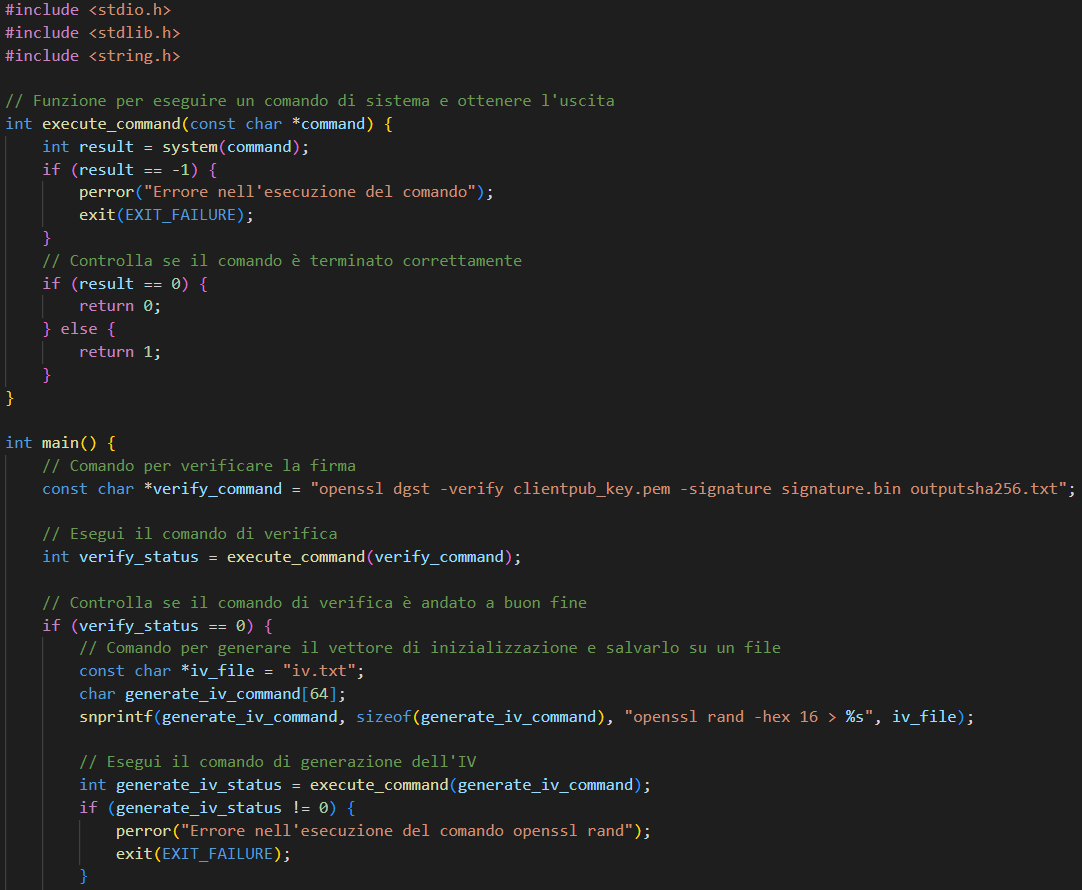
Se la verifica va a buon fine allora l’autorità di rilascio delle credenziali può rilasciare le credenziali.

Innanzitutto, possiamo supporre che l’interfacciamento col db, per controllare che i dati arrivati all’autorità siano validi e non falsificati, non avvenga in quanto non è un’operazione semplice da effettuare.  
L’idea però potrebbe essere quella di confrontare i dati presenti nel file “**index.txt**” ovvero il file “database” che riempie l’autorità che rilascia i certificati (ossia l’IPZS) con i dati presenti all’interno del certificato digitale del client (**clientcert.pem**) e in particolare il campo subject.  
Se, tutti i dati corrispondono, allora significa che quel certificato digitale è stato rilasciato da un’autorità che riteniamo essere affidabile e quindi adesso l’autorità di rilascio delle credenziali può proseguire con la generazione dei dati sensibili per il client.

Per quanto riguarda il rilascio delle credenziali, è sufficiente che l’autorità rilasci all’utente un file cifrato in quanto essendo su TLS e la connessione è sicura, non è un problema utilizzare la cifratura simmetrica.  
E’ stato quindi creato un file che contiene le credenziali chiamato “**credentials.txt**” che verrà cifrato e inviato al client tramite il seguente comando:

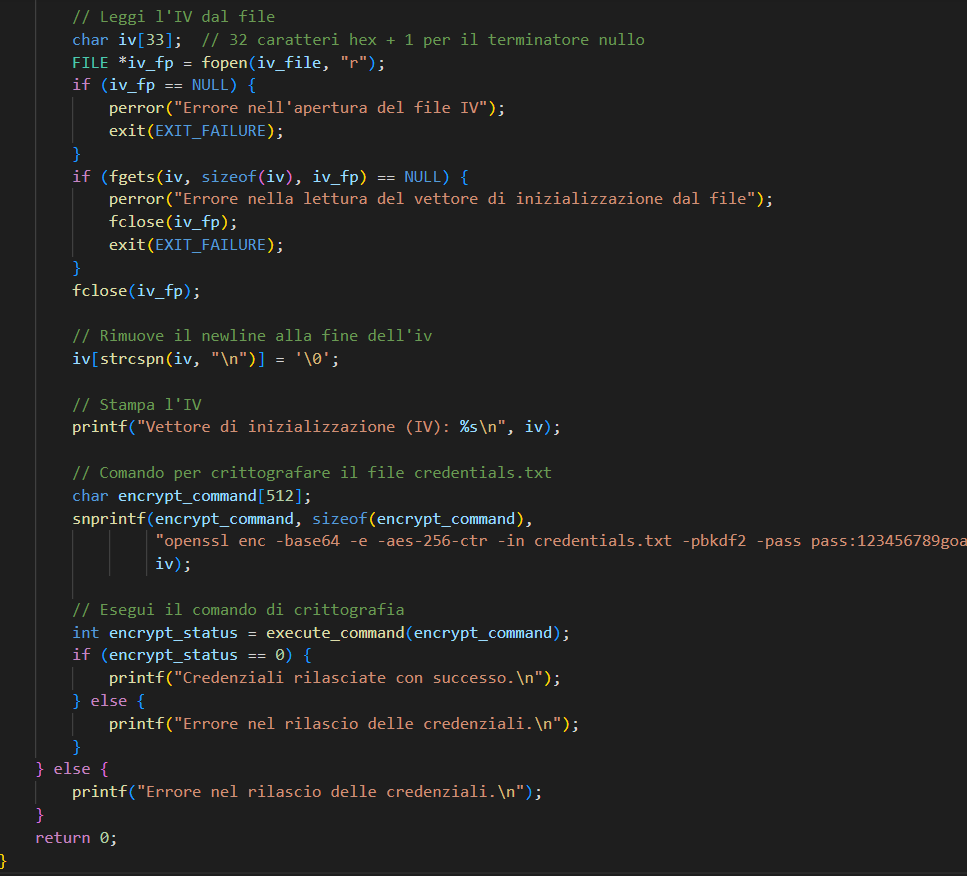
openssl enc -base64 -e -aes-256-ctr -in text.txt -pbkdf2 -pass pass:123456789goabcdef123456789goabcdef123456789goabcdef123456789goabcdef -iv <result\_from openssl rand -hex 16> -out ciphertext.bin

La verifica e il rilascio delle credenziali sono state inserite in uno script.c chiamato “**credentials\_release.c**” che, se la verifica va a buon fine allora l’autorità di rilascio delle credenziali rilascerà le credenziali, altrimenti farà apparire un messaggio di errore “Errore nel rilascio delle credenziali”.

Di particolare rilevanza è il salvataggio del vettore di inizializzazione su un file a parte in quanto questo, generato da:

openssl rand -hex 16

ad ogni esecuzione sarà sempre differente e questo ci serve per poter, lato client, effettuare la decifratura delle credenziali.



Proprio per questo è stato creato un altro script.c chiamato “**decrypting.c**” che permette di decifrare le credenziali inviate in maniera cifrata dall’autorità di rilascio delle credenziali tramite il seguente comando:

openssl enc -base64 -d -aes-256-ctr -pbkdf2 -pass pass:123456789goabcdef123456789goabcdef123456789goabcdef123456789goabcdef -iv <result\_from openssl rand -hex 16> -in ciphertext.bin

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata

Descrizione generata automaticamente

La fase di richiesta delle credenziali da parte del client e la fase di rilascio delle credenziali da parte dell’autorità con annessa cifratura e decifratura è stato inserito in un **makefile** eseguibile tramite il comando **make all** in cui verranno compilati i file.c ed eseguiti. E’ possibile modificare il makefile per controllare alcune condizioni di errore tra cui l’inserimento del pin errato.

**COMUNICAZIONE CLIENT – SERVER**

Questa è la fase successiva a quella in cui il client ottiene le credenziali con le quali è in grado di accedere ad un server S per ottenere una certa risorsa.  
Adesso il client è in possesso del proprio certificato digitale (quello della CIE) “**clientcert.pem**” e delle credenziali “**decrypted\_credentials.txt**” rilasciate dall’autorità di rilascio delle credenziali.

Innanzitutto, come avviene per la connessione client-autorità rilascio credenziali, anche qui necessitiamo che il server, per essere ritenuto affidabile, debba ottenere un proprio certificato rilasciato da un’autorità affidabile.  
Anche in questa situazione, si considera come autorità affidabile l’IPZS che rilascerà un certificato valido per il server S che per semplicità verrà chiamato **serverchevoglioio.it**.

I comandi sono sempre gli stessi:  
prima di tutto è necessario che il server ottenga la sua chiave segreta SK e la sua chiave pubblica PK:

openssl genpkey -paramfile prime256v1.pem -out serversec\_key.pem //per generare la chiave pubblica e privata

openssl pkey -in clientsec\_key.pem -pubout -out serverpub\_key.pem //per estrarre la chiave pubblica dal precedente file

Successivamente il server effettua la richiesta di ottenimento di un certificato tramite il comando:

openssl req -new -key serversec\_key.pem -out serverrequest.pem -config openssl.cnf

e infine il comando che permette all’IPZS di poter accettare la richiesta e rilasciare il certificato al server:

openssl ca -in serverrequest.pem -out servercert.pem -policy policy\_anything -config openssl.cnf

Anche in questo caso il database è stato aggiornato con il nuovo certificato rilasciato.

Adesso che il server ha tutto il necessario per dimostrare di essere affidabile, il client effettua una prova di connessione tramite TLS sempre con i comandi openssl s\_client e s\_server.

In particolare per settare il server usiamo il seguente comando:

openssl s\_server -cert servercert.pem -key serversec\_key.pem -port 8899 -Verify 5 -CAfile Trusted/ client\_certbundle.pem -WWW

in cui anch’esso necessita di controllare i certificati affidabili presenti in **client\_certbundle.pem**.  
Questo comando, a differenza di quello usato dall’autorità di rilascio delle credenziali, ha un’opzione in più ovvero **-WWW** in quanto il server deve rilasciare una certa risorsa che il client richiede.

Il client quindi, per connettersi al server eseguirà il seguente comando:

openssl s\_client -cert clientcert.pem -key clientsec\_key.pem -connect 127.0.0.1:8899 -CAfile Trusted/client\_certbundle.pem -verify\_return\_error

Immagine che contiene testo, schermata, Carattere, informazione

Descrizione generata automaticamenteAdesso quindi, il client, per ottenere la risorsa, ossia effettivamente ottenere ciò che il server offre, è necessario che si colleghi ad un qualsiasi browser (per gli esempi è stato usato chrome) e digitale il seguente url: [**https://127.0.0.1/resource.html**](https://127.0.0.1/resource.html)che gli mostrerà a video il sito al quale egli voleva collegarsi.

Quando si collega però, ci sono problemi di certificati che vanno inseriti manualmente nel browser per avvisare a quest’ultimo che la risorsa che stiamo richiedendo è una risorsa proveniente da un server sicuro!

L’unica differenza che c’è con la progettazione della funzionalità nel WP2 è che non vengono richieste le credenziali all’utente, funzione che non è di semplice implementazione tramite i metodi classici elencati fino ad adesso.  
L’idea di base sarebbe quella di, da parte del server, filtrare le richieste ai soli possessori di credenziali valide per accedere a quella specifica risorsa.  
Proprio per questo, se il possiede le credenziali valide allora il server permetterà l’accesso, altrimenti abortisce la connessione lanciando un messaggio di errore.