

Alma Mater Studiorum - Università di Bologna  
Corso di Laurea in Ingegneria Informatica Magistrale

Attività progettuale di Sicurezza dell' Informazione  
February 2023

# Advanced Face Authentication System: Ensuring Security and Robustness

Patrick Di Fazio

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Passwordless Authentication</b>	<b>3</b>
2.1	Background . . . . .	3
2.2	Challenges . . . . .	4
<b>3</b>	<b>A Survivable Authentication System based on Facial Recognition</b>	<b>5</b>
3.1	Architecture . . . . .	5
3.2	Implementation . . . . .	6
3.3	Signature: List Mode . . . . .	7
3.4	Signature: BLS Mode . . . . .	7
3.5	Implementation choices . . . . .	7
3.6	Interfaccia Utente . . . . .	9
<b>4</b>	<b>Evaluation</b>	<b>10</b>
4.1	List Mode . . . . .	11
4.2	BLS Mode . . . . .	12
<b>5</b>	<b>Conclusioni</b>	<b>13</b>

# 1 Introduzione

Gli applicativi moderni sono utilizzati da migliaia di utenti, connessi contemporaneamente che necessitano di mantenere un'identità sul servizio a cui sono registrati. Per avere la sicurezza che le identità siano univoche ed appartengano ad un solo utente, si utilizzano metodi di accesso come l'inserimento di credenziali. Solo chi dispone delle corrette credenziali può avere l'accesso al servizio che può essere un sito web, un' applicazione o molto altro.

Il caso più comune di autenticazione ad un servizio è quindi l'inserimento testuale di una stringa per l' username e di una per la password. Questo viene utilizzato nei *Login*, per accedere al servizio e durante la *registrazione*, per salvare le credenziali sui server. Con queste modalità, se non sono utilizzate le giuste protezioni, possiamo subire attacchi informatici mirati a rubare credenziali.

Un attaccante malevolo potrebbe infatti cercare di penetrare in un sistema con le credenziali di un altro utente e ne acquisirebbe automaticamente l'identità. Agli occhi del sistema chi possiede la coppia username-password è pienamente riconosciuto come il proprietario dell'account.

Le possibilità di attacco sono molteplici e di diversa natura. Uno dei metodi utilizzati per rubare le password è l'ingegneria sociale, ovvero grazie all'utilizzo di email e messaggi di phishing, si riesce a convincere l'utente a fornire le credenziali o farle inserire in un sito web non sicuro, rendendole facilmente leggibili dall'attaccante. Molto comune è anche l'attacco a forza bruta, o bruteforce, che consiste nel provare molte combinazioni di username o password, al fine di trovare, per probabilità, le password più comuni. Questo attacco sfrutta dei dizionari di parole che di solito sono delle liste di password più utilizzate dagli utenti. Un'altra tecnica sfruttata consiste nel recuperare la lista delle password rubate durante un *data breach* e utilizzarle in altri servizi. In questo caso, come nell'ingegneria sociale, il punto debole è il fattore umano, che si preoccupa molto poco della propria sicurezza online, utilizzando la stessa password su molteplici account. Eliminando il fattore dell'inserimento delle password, sarebbe possibile fornire un layer di protezione aggiuntivo all'account dell'utente finale.

Il progetto ha l'obiettivo di realizzare di un sistema di autenticazione che sfrutta l'approccio passwordless e che permetterà ad un utente registrato ad un' applicazione web di accedere al servizio fornito dal sito utilizzando la biometria facciale, tramite browser. Una volta che l'utente ha raggiunto il servizio, infatti, verrà aperta una schermata in cui è possibile scattare una foto che viene poi elaborata e validata da una catena di server. L'utilizzo di più server aumenta la sicurezza del sistema, perché l'identità dell'utente non è validata da un singolo elemento, ma da molteplici. Questo evita il *Single point of failure* legato alla sicurezza di un unico servizio che si occupa di rilasciare informazioni. Avendo una lista di server possiamo infatti affidare la sicurezza del nostro account ad una lista di firme. Se una di queste dovesse rivelarsi non valida, a causa di una vulnerabilità trovata sul server, o a causa di una manomissione, ci ritroveremmo con le altre N firme che garantiscono l'identità dell'utente e che, in seguito, possono essere verificate.

## 2 Passwordless Authentication

L'autenticazione senza password è vantaggiosa per molte ragioni, ma l'impatto più significativo è sulla user experience e sulla sicurezza. I benefici di questo sistema variano a seconda del caso d'uso. Ad esempio, in ambito aziendale si può adottare l'autenticazione passwordless utilizzando grandi server che permettono l'accesso centralizzato degli utenti, mentre per gli utenti singoli di un servizio si potrebbe utilizzare l'autenticazione passwordless per garantire una maggiore semplicità di accesso, rendendo l'applicativo disponibile per una grande fascia di utenti.

### 2.1 Background

Nei sistemi di autenticazione passwordless, l'utente presentandosi all'applicativo, non ha più bisogno di ricordare e utilizzare le credenziali di accesso. La password in questo caso, sarebbe inesistente e quindi impossibile da rubare. L'unica modalità di accesso risulterebbe infatti un token generato a partire dall'identificazione passwordless. Inoltre, non avendo una password, si eliminerebbero tutti i problemi relativi al mantenimento di questa, come ad esempio l'aggiornamento o il reset della password dovuto allo smarrimento.

L'utente finale risulta avvantaggiato da questa semplicità, potendo utilizzare questo sistema come *Single-Sign-On* o come metodo sicuro di login ad un servizio. Nonostante si utilizzi un sistema di login non tradizionale, a livello applicativo si avrà la completa trasparenza rispetto al sistema di autenticazione utilizzato. Grazie all'utilizzo delle apposite API è infatti possibile garantire l'accesso allo stesso sistema sia tramite password sia tramite autenticazione passwordless. [1]

Quando l'utente si autentica in modo passwordless, non fornisce una stringa di lettere, ma deve comunque fornire qualcosa all'identificatore per avere conferma sulla propria identità. In questo caso l'utente si può autenticare attraverso qualcosa che ha, oppure attraverso qualcosa che è. Esistono infatti varie modalità di autenticazione passwordless, come ad esempio l'autenticazione tramite QR-Code (ovvero scansionando un codice QR), Magic Links (accedendo a dei link statici prefatti) oppure tramite Biometria.

Questo ultimo tipo di autenticazione è ciò che si utilizzerà in questo progetto. In particolare, si studierà l'accesso passwordless tramite biometria facciale, anche se è possibile estendere il progetto ad un qualunque tipo di biometria, come ad esempio quella basata su impronta digitale.



Figura 1: Passwordless Authentication  
[12]

## 2.2 Challenges

L'autenticazione passwordless ha numerosi benefici, ma ha anche molti punti critici da analizzare e sui quali dobbiamo porre attenzione, per evitare di realizzare un sistema insicuro. L'implementazione di tale approccio deve essere fatta a seguito di un'analisi di costi/benefici.

In particolare, con l'autenticazione facciale come sistema di accesso deve mantenere dei vincoli stringenti per garantire il livello minimo di sicurezza.

1. **Accuratezza:** L'accesso al sistema con l'identità di un utente, da parte di altri utenti, non deve essere possibile, quindi si deve avere un alto grado di affidabilità dell'accesso. Si deve quindi ridurre al minimo la quantità di falsi positivi per garantire sicurezza, ma si deve anche evitare di avere falsi negativi che andrebbero a creare un disservizio e quindi peggiorerebbero l'esperienza utente.
2. **Sicurezza:** I sistemi di riconoscimento facciale possono essere bypassati da un attaccante che utilizza delle false immagini o video. Se non si utilizza un sistema con una bassa confidenza di riconoscimento o un sistema che non è capace di compiere scansioni tridimensionali (come nel caso di Windows Hello o lo sblocco facciale di Android) si rischia di compromettere la sicurezza del sistema. Nel caso di una implementazione tramite biometria tridimensionale, generalmente il costo del sistema e la fattibilità diminuisce, dato che ogni utente dovrebbe disporre di un hardware adeguato a compiere la scansione.
3. **Privacy:** Oltre alle questioni di sicurezza, un approccio basato su sblocco facciale deve mantenere il numero minimo possibile di informazioni riguardo alla persona. Se il sistema genera una stringa corrispondente alla faccia dell'utente, non deve essere possibile riuscire a riprodurre o ricostruire una faccia partendo dalla stringa nota, ovvero l'encoding deve essere il meno invertibile possibile. Inoltre per evitare di diffondere l'immagine durante i *data breach*, che recentemente stanno diventando sempre più frequenti, un'ulteriore previdenza per aumentare la privacy è non salvare le immagini dell'utente lato server.
4. **Accessibilità:** Per utilizzare correttamente l'applicativo, nel caso del sistema biometrico facciale l'utente finale deve fare uso di una webcam. Per quanto questo oggetto può essere comune e facilmente reperibile, non è sufficiente per tutte le tipologie di questo tipo di autenticazione. Nel caso di sistemi come Face ID di Apple [6] o Windows Hello di Microsoft [14] si utilizzano infatti particolari componenti hardware che proiettano sulla faccia dell'utente una matrice di luci che possono essere poi analizzate da una speciale fotocamera per produrre un *fingerprint* dell'utente. Non è detto quindi che tutti gli utenti possano disporre dell'hardware e degli strumenti adeguati a riprodurre questo tipo di autenticazione. Utilizzando solo la webcam, invece, si garantisce una maggiore disponibilità del servizio a discapito della sicurezza.

### 3 A Survivable Authentication System based on Facial Recognition

Per definire la costruzione dell'ecosistema di autenticazione passwordless, si deve fare prima un'analisi e uno schema delle principali componenti architetturali.

Il servizio sarà erogato su un server principale che conterrà la logica applicativa e il token di sessione per ogni utente, poi sarà presente il gestore delle identità degli utenti ed un controller che gestirà le richieste ai server di firma. Infine partecipano alla firma N trust server configurabili a runtime dell'applicazione. Di seguito sono presentate l'architettura e l'implementazione del progetto.

#### 3.1 Architecture

Il progetto si divide in tre macrocategorie, in cui ogni categoria ha un compito e un dominio ridotto, in modo da gestire una netta divisione di compiti e quindi avere una maggiore sicurezza sugli elementi più sensibili che verranno posti in sottoreti non raggiungibili dall'esterno.

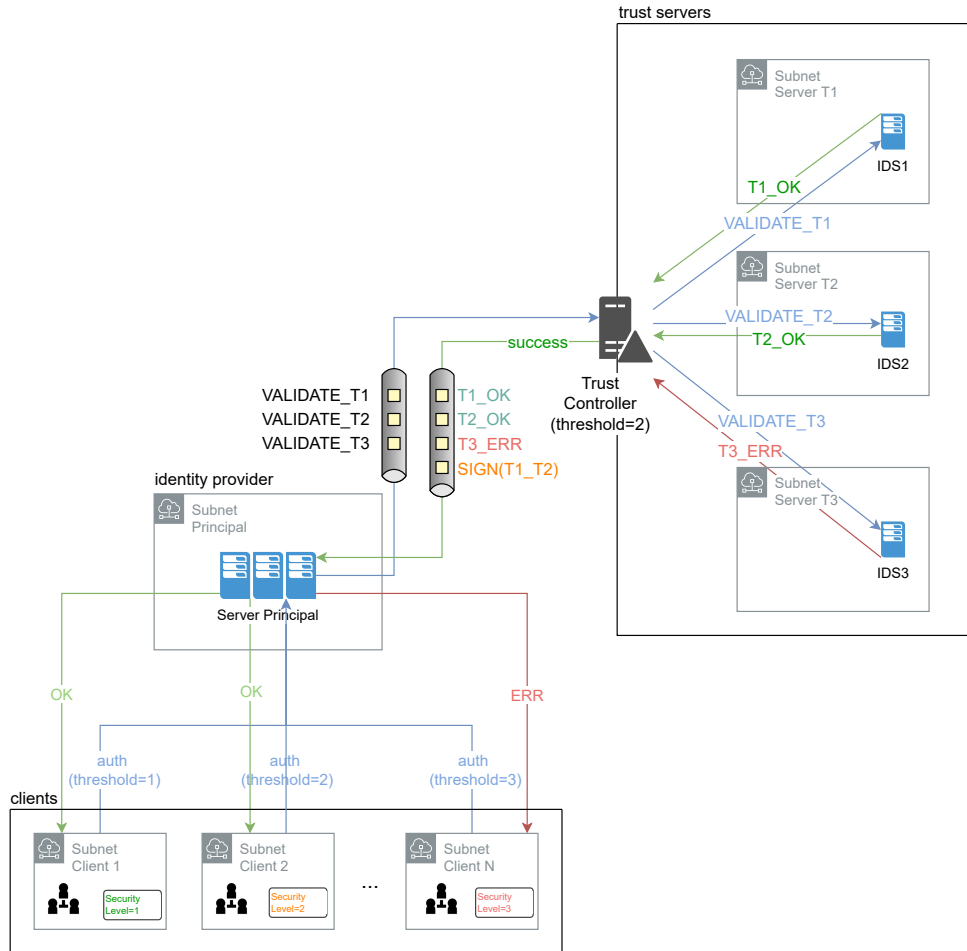


Figura 2: Architecture Diagram

- **Client:** Prevede l'accesso al servizio dei client, a partire dal singolo utente fino ad arrivare all'organizzazione che sceglie di adottare la metodologia passwordless. Qua i client si connettono al servizio in base al livello di sicurezza che vogliono mantenere. Un client che necessita di numerosi server che firmino la sua faccia, sceglierà un livello di sicurezza maggiore. Per accedere a servizi meno critici, sarà altresì possibile che il client scelga un livello di sicurezza minore, a vantaggio di una maggiore velocità di firma e meno occupazione di banda. Il client si connette tramite il browser all'indirizzo del Service Provider, inserisce il proprio Username e scatta la foto. Dopo pochi millisecondi riceve una risposta dal server: se la foto è valida e l'encoding corrisponde con quello salvato sull' Identity Provider, l'utente accede alla Homepage del servizio, altrimenti riceverà l'errore HTTP 401 (Unauthorized).
- **Server:** Questa parte si occupa di ricevere la richiesta dell'utente, generare l'encoding della faccia e spedirlo ad un controller che gestirà la richiesta di firma inoltrando l'encoding ai trust servers. Qui è dove erogiamo il servizio all'utente e generiamo il token che verrà poi utilizzato all'interno del servizio e che identificherà univocamente l'utente nel sito web. Il primo passo che fa il Service Provider è ricevere la richiesta dai Client ed elaborare l'encoding della faccia in locale. Al termine dell'elaborazione gira la richiesta effettuata all'Identity Provider che prepara la richiesta per il Trust Controller, se l'utente è registrato nel sistema e aspetta risposta. Il Trust controller a sua volta, conoscendo la lista di server di firma, ha la logica per fare una richiesta ad ogni server ed unire le firme.
- **Signature:** In questo livello si presentano i server di firma che si occupano di confrontare l'encoding della faccia inviata precedentemente con quello salvato dal gestore di identità degli utenti. I Trust Server fanno un'operazione di confronto di array che restituirà un risultato positivo solo se l'encoding generato rientra in un intervallo di confidenza specificato a runtime. In altre parole, se l'encoding è *abbastanza simile* a quello salvato, firmeranno il pacchetto e lo restituiranno al client. I Trust Server, in base al tipo di firma che avrà il sistema, differiranno fortemente. Per firmare e generare i token il servizio può infatti utilizzare due differenti sistemi: uno è basato sull'algoritmo RSA e uno utilizza le signature BLS, meglio descritte in seguito. Il deploy è effettuato in modo parametrico, utilizzando lo stesso container ma cambiando le variabili d'ambiente per ogni nuovo server. Il Trust Controller deve essere a conoscenza di tutti i server che firmano la foto, per gestire al meglio la sicurezza del sistema.

## 3.2 Implementation

Per ogni macrocategoria, è possibile analizzare i singoli componenti, in particolare i servers, per poi scendere nel dettaglio implementativo. Successivamente si porrà nota sulla metodologia di firma, sull'architettura di rete e sul metodo di deploy degli elementi.

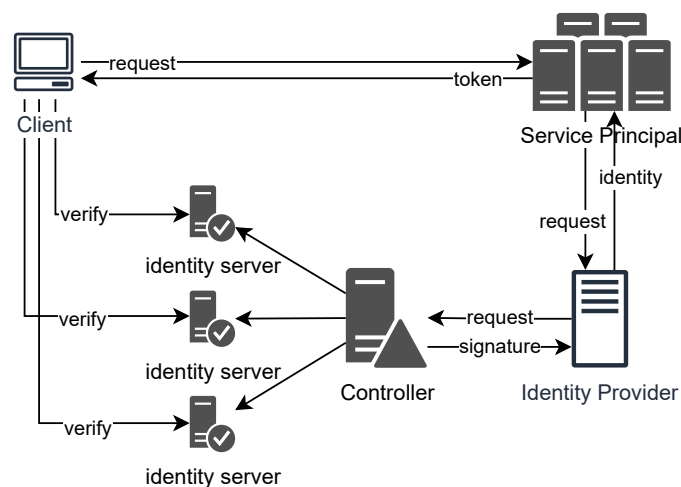


Figura 3: Dettagli diagramma

Nel diagramma in figura possiamo vedere come il primo passo per richiedere un'identità sia la

richiesta da parte del client al server o fornitore del servizio. Qui si contatta quindi il **Service Provider** specificando l'username dell'utente. È proprio nel Service Provider che è effettuato il deploy del webserver raggiungibile dai client ed è lui che si occupa di elaborare la faccia del client per generare l'encoding.

Una volta completata la generazione dell'encoding viene inviata all'**Identity Provider** una richiesta di identità che contiene come parametri l'encoding precedentemente generato e l'username dell'utente. Se l'username corrisponde ad uno esistente, si inoltra la richiesta formata dall'encoding appena ricevuto e dall'encoding salvato per quell'utente, concatenati all'username. Questo pacchetto viene inviato al **Trust Controller**.

Il Controller recupera il numero di server configurato a runtime, secondo il livello di sicurezza richiesto e prepara una richiesta ciclica per ogni **Identity Server** o **Trust Server** presente, che comprende i due encoding ricevuti e l'username dell'utente. Ogni server si occupa di produrre un booleano che indica se i due array sono simili. In caso il risultato sia positivo, ogni server produce un pezzo del *token* che viene poi inviato al Trust Controller come risposta. Se uno dei server non è raggiungibile viene saltato.

La risposta generata, che comprende tutti i pezzi dei token creati, viene poi rigirata all'Identity Provider che fornisce l'identità e il JSON Web Token [9] al client che sarà identificato sul sito web fino al suo logout. Infine, l'applicativo può utilizzare due modalità diverse di signature: la modalità a **Lista** e la modalità con **BLS**. Indipendentemente dal tipo di firma, il funzionamento del servizio apparirà trasparente all'utente, che verrà validato e identificato sempre tramite un **JWT**.

### 3.3 Signature: List Mode

In questa modalità, ogni trust server firma la risposta con la propria chiave pubblica e invia il pacchetto al Trust Controller che collezionerà le firme e le invierà al Service Provider tramite l'Identity Provider. A questo punto il Server potrà cifrare una chiave di sessione con la chiave pubblica recuperata dalle firme dei Trust server e inviarla ai vari Trust Server. Se ognuno di loro risponde con la corretta decifrazione della chiave di sessione, attraverso la propria chiave privata, allora si registra quel server come valido e si incrementa di uno il numero di server che hanno firmato il token di sessione. Quando il numero di firme è superiore al limite di sicurezza desiderato per lo specifico Service Provider, si considera il token di sessione valido e l'utente è libero di navigare nel sito web fino al suo logout, dove le firme vengono invalidate.

Considerando il lato implementativo, per questo tipo di firma si utilizza una modalità di cifratura asimmetrica, utilizzando una libreria *RSA* su Python.

### 3.4 Signature: BLS Mode

Questa modalità di firma, implementa una *multi-signature* [2] ampiamente utilizzata nella Blockchain per ridurre la dimensione delle transazioni, dove per verificare un insieme di firme, il verificatore deve avere una funzione *aggregate()* delle firme, ovvero deve mantenere solo una ridotta aggregazione di tutte le chiavi pubbliche dei firmanti. Così facendo si riduce il numero di elementi computati che quindi risulta in una maggiore velocità nella verifica della firma, avendo un minore sforzo computazionale.

Nel caso dei Trust Server, ogni server genera il proprio token attraverso la funzione *bls.signature()*, cifrando l'encoding e l'username. Queste informazioni saranno concatenate alla *Proof of Possession* della chiave privata e inviate al Trust Controller. Qui si crea la *aggregate()* che unirà tutte le firme dei server e le invierà al Service Provider attraverso l'Identity Provider. Quando si riceve il pacchetto da validare, il Server Principal verificherà la signature con *verify.aggregate()* che confronta l'insieme di firme generate con la lista di *Proof of Possession* delle chiavi pubblica recuperate da ogni Trust Server.

### 3.5 Implementation choices

Durante lo sviluppo del progetto è stato utilizzato **Flask** [8] come web server **Python**, nel Service Provider. È stato scelto questo linguaggio per sfruttare le funzioni di RSA, BLS e le numerose librerie di **face\_recognition** già implementate [7]. Per questo motivo Python è utilizzato anche dal Trust Controller e dai Trust Server. Nell'Identity Provider è stato invece utilizzato **NodeJS**



perchè ha come scopo primario l'inoltro delle richieste e perché possiede librerie per la produzione del JWT.

L'unico servizio esposto è il Server Principal attraverso il web server che è raggiungibile da qualunque client. Ogni altro elemento può essere posto in una unica sottorete o in una sottorete diversa. Di solito per questioni di sicurezza si adotta la seconda scelta. A monte del servizio, dovrebbe comunque essere adottato un **Firewall** per gestire le richieste all'applicativo.

Ognuna delle componenti per velocità di testing e di deploy, è stata containerizzata su **Docker** [5] attraverso la scrittura degli appositi Dockerfile. Per utilizzare invece un container prefatto si può consultare il registry presente nel Docker Hub, dove sono presenti due versioni distinte del servizio, separate attraverso tag *container:list* o *container:bls*. L'utilizzo di Docker o della containerizzazione in generale è perfettamente integrabile con Kubernetes.

Grazie a **Kubernetes** [10] è possibile orchestrare il deploy dei vari container in un unico file **YAML** [15], definendo politiche di rete, di Ingress ed Egress dai pod (un raggruppamento di uno o più container). Il vantaggio di questo approccio è che è possibile avviare l'infrastruttura di rete e i server parametrizzati direttamente da un unico file opportunamente modificato, usando un solo comando:

```
kubectl apply -f deployment.yaml
```

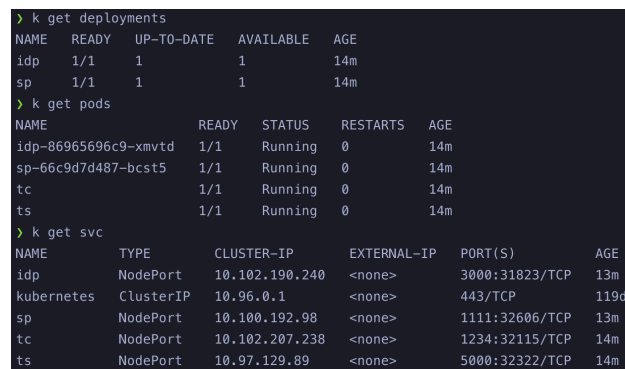
Il file di deploy contiene numerose istruzioni e definisce il Service Provider e l'Identity Provider come dei deployment, il Trust Controller e i Trust Server come pod. Ogni servizio comunica attraverso i *services* definiti su Kubernetes, del tipo **ClusterIP** [13] che espongono una porta e rendono raggiungibile i container attraverso il DNS interno.

Le variabili sono passate ai container tramite una **ConfigMap** che le rende variabili d'ambiente. Questa scelta rende altamente parametrico il deployment [3]. Nella ConfigMap sottostante si parametrizzano infatti i Trust Servers che vengono passati come lista al Trust Controller, insieme al threshold di sicurezza del sistema.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configuration
data:
  trusted-servers: |
    - http://trust_server_1:5000
    - http://trust_server_2:6000
    - http://trust_server_3:7000
  threshold: "2"
```

Kubernetes è un ottimo strumento che porta numerosi vantaggi al sistema. Può sfruttare efficienti sistemi di logging, può definire politiche di sicurezza per fare enforcement del sistema e permette di scalare dinamicamente i deployment quando il carico su un server diventa troppo alto, creando così un ambiente ad alta affidabilità.

Nell'immagine sottostante è possibile vedere un deployment online e funzionante del progetto.



```
> k get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
idp     1/1     1             1           14m
sp      1/1     1             1           14m

> k get pods
NAME                                READY   STATUS    RESTARTS   AGE
idp-86965696c9-xmvtb               1/1     Running   0           14m
sp-66c9d7d487-bcst5                 1/1     Running   0           14m
tc                                    1/1     Running   0           14m
ts                                    1/1     Running   0           14m

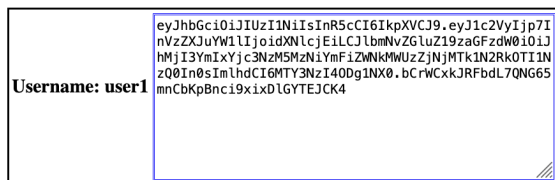
> k get svc
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
idp       NodePort    10.102.190.240 <none>        3000:31823/TCP   13m
kubernetes ClusterIP    10.96.0.1      <none>        443/TCP          119d
sp        NodePort    10.100.192.98  <none>        1111:32606/TCP   13m
tc        NodePort    10.102.207.238 <none>        1234:32115/TCP   14m
ts        NodePort    10.97.129.89   <none>        5000:32322/TCP   14m
```

Figura 4: Risorse create con Kubernetes

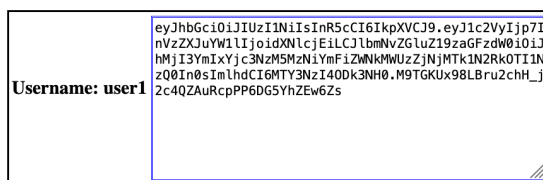
### 3.6 Interfaccia Utente

Nelle due immagini presenti si distingue l'interfaccia dell'utente in base alla modalità di firma con lista o con BLS. Con la modalità con lista infatti è possibile verificare i server che hanno firmato, potendo accedere alle chiavi pubbliche in formato PEM. Firmando con BLS, all'utente appare oltre che la chiave pubblica, anche la Proof of Possession. A fianco ad ogni chiave è possibile inoltre, per entrambi i casi, vedere l'URL del server di firma con la rispettiva porta.

JWT Token



JWT Token



Trust Servers



Trust Servers



Figura 5: Firma con List

Figura 6: Firma con BLS

L'utente verrà identificato nel sito grazie al JWT che sarà formato da una struttura dati contenente l'username dell'utente e l'hash dell'encoding. Il token viene conservato fino al logout ed è univoco per ogni identità.

## 4 Evaluation

In questa sezione distinguiamo il differente tipo di firma dal punto di vista delle prestazioni. Per produrre i grafici è stato utilizzato Wireshark [4]. I test effettuati prendono in considerazione un singolo login tramite registrazione facciale (e quindi invio della foto, generazione dell'encoding e firma dei server) seguito dal logout dell'utente.

Per analizzare il traffico di rete con Wireshark, si può agganciare una sonda alle interfaccia di *loopback* del sistema. Questo è possibile perché è stato utilizzato Minikube [11], ovvero un metodo per fare il run di Kubernetes in locale, sulle interfacce localhost e abilitando il port forwarding per raggiungere con il browser il servizio del Service Provider sulla porta 1111. In questo modo si può raggiungere il webserver e si possono leggere le comunicazioni tra i server interni. Grazie alla sezione statistiche di Wireshark è poi possibile creare grafici significativi per l'utilizzo di banda, per la size dei pacchetti e sulle statistiche TCP della connessione.

Possiamo vedere il primo grafico che rappresenta il numero totale di pacchetti computati e quindi inoltrati tra i vari server per compiere una firma, con un'analisi sulla banda media consumata. Nel secondo e nel terzo si evidenziano il throughput totale della rete e la percentuale di pacchetti in uscita e in entrata nei server, nei 5 secondi di utilizzo. In Figura 8 e in Figura 11 si vede come attorno ai 2 secondi ci sia un picco di utilizzo di rete: questo è dovuto all'invio, al processamento della foto e alla richiesta e quindi conseguente invio delle signature da parte dei trust server. Il grafico prosegue con un picco più contenuto che indica il logout dell'utente. I grafici di I/O in Figura 10 e Figura 11 rappresentano la stessa situazione, ma comprendono entrambi i dati in input e in output dei server.

Durante l' utilizzo del sistema, un utente eseguendo le stesse azioni, ovvero un login e un logout, ha un differente utilizzo di banda. Questo è perché le due modalità di firma impiegano risorse diverse, sia in termini di utilizzo di rete, sia per quanto riguarda le risorse di calcolo. In modalità lista notiamo che per le stesse azioni si registrano **342** pacchetti utilizzati, contro i **340** della modalità BLS. Analizzando invece il bitrate di entrambi, risaltano i **163 Kbit/s** della modalità a lista, contro i **152Kbit/s** della modalità con BLS: un risultato ragionevole dato che la Average packet size di un pacchetto BLS è **316** Byte, contro i **342** della modalità con lista. Notiamo quindi che BLS ha circa lo **0.6%** di pacchetti in meno utilizzati, circa un **6.7%** in meno di Kbit/s scambiati durante le firma e una Average packet size minore del **7.6%** rispetto alla modalità con lista.

### 4.1 List Mode

Statistics		
Measurement	Captured	Displayed
Packets	342	342 (100.0%)
Time span, s	5.740	5.740
Average pps	59.6	59.6
Average packet size, B	342	342
Bytes	117042	117042 (100.0%)
Average bytes/s	20 k	20 k
Average bits/s	163 k	163 k

Figura 7: Pacchetti registrati (LIST)

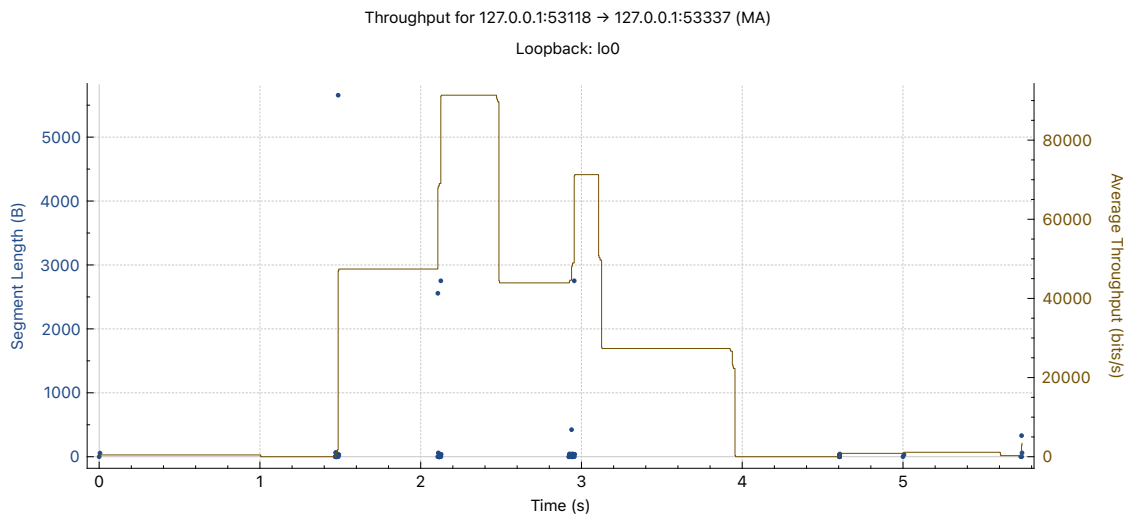


Figura 8: Throughput (LIST)

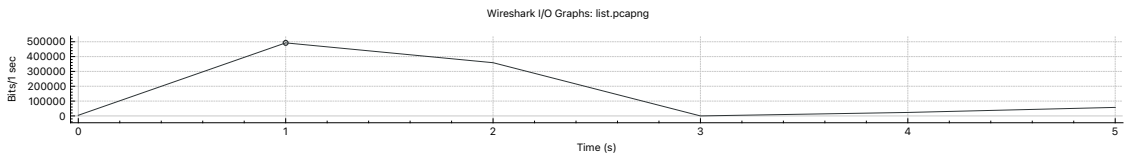


Figura 9: Grafico I/O (LIST)

## 4.2 BLS Mode

Statistics		
Measurement	Captured	Displayed
Packets	340	340 (100.0%)
Time span, s	5.630	5.630
Average pps	60.4	60.4
Average packet size, B	316	316
Bytes	107300	107300 (100.0%)
Average bytes/s	19 k	19 k
Average bits/s	152 k	152 k

Figura 10: Pacchetti registrati (BLS)

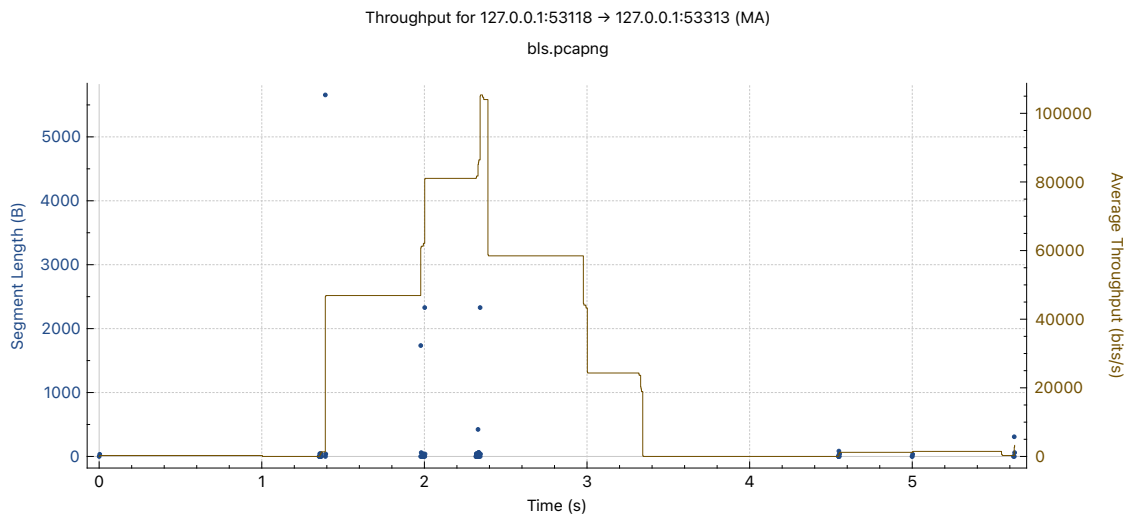


Figura 11: Throughput (BLS)

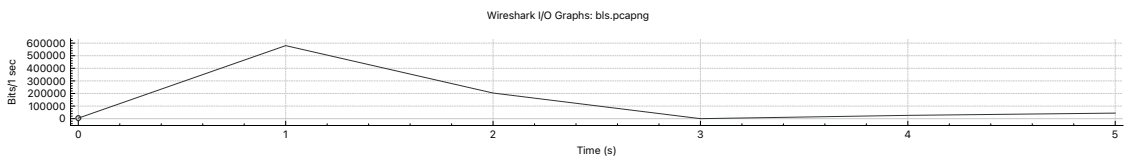


Figura 12: Grafico I/O (BLS)

## 5 Conclusioni

Il progetto **Trust Identity Chain** fornisce un sito web funzionante al quale gli utenti possono autenticarsi utilizzando la biometria facciale e quindi con un approccio passwordless. In questo progetto, infatti, l'utente si collega e accede solamente attraverso l'username e la propria faccia. Così facendo avrà tutta una serie di vantaggi precedentemente elencati e non dovrà ricordarsi una stringa alfanumerica per fare il Login. In uno scenario in cui molti servizi adottano questa scelta, l'utente potrà accedere ad ogni applicazione secondo una modalità di SSO in cui la propria faccia è l'unico elemento che serve per autenticarsi, oltre all'username. In questo modo si ridurrebbe di molto il pericolo di fishing, dato che non ci sarebbero password da rubare. Sarebbe inoltre ridotto anche un potenziale danno in seguito ad un data breach, perché data la unidirezionalità dell'encoding, sarebbe difficile risalire ad una faccia partendo da questo.

Questo come abbiamo detto precedentemente porta numerosi vantaggi, ma apre anche a possibili punti critici. Una criticità del sistema è infatti data dall'utilizzo di solo una fotocamera: lo sblocco facciale non avviene su uno studio tridimensionale del soggetto, ma utilizzando solo due dimensioni e quindi è possibile autenticarsi ad un'utenza utilizzando anche solo una foto di un altro utente. Tuttavia il progetto è estendibile ad altri tipi di biometria, perché agendo sulle API di autenticazione del webserver, si può cambiare metodo di identificazione, mantenendo trasparente il comportamento dell'identity provider, del trust controller e dei server di firma.

In un possibile sviluppo futuro, sarebbe opportuno implementare le funzioni di encoding direttamente lato client, e quindi su Javascript. Questo incrementerebbe notevolmente la privacy dell'utente, il quale non avrà la propria foto salvata (anche periodi brevi di storing sono da evitare) ed elaborata dal server. Per implementare sul client un encoding, sono necessarie però delle librerie che devono produrre una struttura dati compatibile con quella che viene elaborata dai server, quindi sarebbe richiesto un tempo di sviluppo maggiore.

Per concludere, potendo analizzare le performance dei due tipi di firma, si nota che con BLS si ha un leggero incremento dell'efficienza e una riduzione della banda utilizzata. In un utilizzo massivo di questo metodo si risparmierebbero quindi notevoli quantità di dati scambiati tra i server interni. L'incremento prestazionale è dato dalla minor quantità di dati processati, infatti BLS utilizza una versione ridotta delle chiavi e sfrutta un processo di verifica più veloce.

## Elenco delle figure

1	Passwordless Authentication . . . . .	3
2	Architecture Diagram . . . . .	5
3	Dettagli diagramma . . . . .	6
4	Risorse create con Kubernetes . . . . .	8
5	Firma con List . . . . .	9
6	Firma con BLS . . . . .	9
7	Pacchetti registrati (LIST) . . . . .	11
8	Throughput (LIST) . . . . .	11
9	Grafico I/O (LIST) . . . . .	11
10	Pacchetti registrati (BLS) . . . . .	12
11	Throughput (BLS) . . . . .	12
12	Grafico I/O (BLS) . . . . .	12

## Riferimenti bibliografici

- [1] *Autenticazione Passwordless*. URL: <https://www.beyondidentity.com/resources/passwordless-authentication>.
- [2] *BLS Signature*. URL: <https://eprint.iacr.org/2018/483.pdf>.
- [3] *Configmaps*. URL: <https://kubernetes.io/docs/concepts/configuration/configmap/>.
- [4] *Configmaps*. URL: <https://www.wireshark.org/docs/>.
- [5] *Docker Documentation*. URL: <https://docs.docker.com>.
- [6] *Face ID*. URL: <https://support.apple.com/en-us/HT208108>.
- [7] *Face Recognition*. URL: [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition).
- [8] *Flask Documentation*. URL: <https://flask.palletsprojects.com/en/2.2.x/>.
- [9] *JSON Web Token*. URL: <https://jwt.io/>.
- [10] *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/home/>.
- [11] *Minikube*. URL: <https://minikube.sigs.k8s.io/docs/start/>.
- [12] *Passwordless Scheme*. URL: <https://www.transmitsecurity.com/blog/passwordless-authentication-guide>.
- [13] *Service*. URL: <https://kubernetes.io/docs/concepts/services-networking/service/>.
- [14] *Windows Hello*. URL: <https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/hello-biometrics-in-enterprise>.
- [15] *YAML*. URL: <https://yaml.org/>.