



UNIVERSITÀ
DI PISA

University of Pisa

Department of Information Engineering

FOC project

Nicolò Mariano Fragale, Federico Rossetti
May 2025

Contents

1	Analisi DSS	2
2	Digital Signature Server	5
2.1	__init__	5
3	Autenticazione	6
3.1	Ed25519	6
3.2	Identificatore di protocollo e prevenzione cross-protocol	7
3.3	Sintesi	8
4	Flusso della comunicazione: descrizione lineare e schemi	9
4.1	Obiettivo e precondizioni	9
4.2	Handshake: negoziazione, autenticazione del server, derivazione di K	9
4.3	Fase applicativa: confidenzialità, integrità e anti-replay	10
4.4	Mappa concettuale delle operazioni	10
4.5	Esempi di uso nella fase applicativa	11

1 Analisi DSS

Obiettivo del sistema Un'organizzazione utilizza un servizio di firma digitale (Digital Signature Server, DSS) che agisce come terza parte fidata: genera coppie di chiavi per conto dei dipendenti, le conserva e produce firme digitali su richiesta dell'utente.

Registrazione iniziale (off-line) e credenziali Gli utenti/dipendenti sono registrati *off-line*. In fase di registrazione ricevono:

- la **chiave pubblica del DSS** (da conservare);
- una **password iniziale**, che *deve essere cambiata al primo accesso*.

Canale sicuro e autenticazione Prima di invocare qualsiasi operazione, l'utente deve stabilire un **canale sicuro** verso il DSS, che soddisfi i requisiti di:

- **Perfect Forward Secrecy (PFS)**;
- **integrità dei messaggi**;
- **protezione dal replay** (no-replay);
- **non-malleabilità**.

L'autenticazione avviene come segue:

- **autenticazione del server** tramite la sua chiave pubblica (nota all'utente);
- **autenticazione dell'utente** tramite la sua password.

Operazioni esposte dal servizio Dopo la connessione sicura e l'autenticazione, il DSS espone le seguenti operazioni di livello applicativo:

1. **CreateKeys**: crea e memorizza una coppia (*privata, pubblica*) per l'utente invocante. Se la coppia *esiste già*, l'operazione *non ha effetto* (idempotenza).

Precondizione: sessione sicura attiva; utente autenticato.

Postcondizione: esiste una coppia di chiavi associata all'utente (invariata se già presente).

2. **SignDoc(documento)**: restituisce la **firma digitale** del documento passato come argomento; il DSS firma *per conto dell'utente invocante*.

Precondizione: sessione sicura; utente autenticato; *coppia di chiavi esistente*.

Postcondizione: ottenuta e restituita la firma digitale sul documento.

3. **GetPublicKey(utente)**: restituisce la **chiave pubblica** dell'utente indicato.

Precondizione: sessione sicura; utente autenticato, coppia di chiavi esistente.

Postcondizione: consegna della chiave pubblica richiesta.

4. **DeleteKeys:** elimina la coppia di chiavi dell'utente invocante. *Dopo l'eliminazione, l'utente non può crearne una nuova a meno di una nuova registrazione off-line.*

Precondizione: sessione sicura; utente autenticato; coppia di chiavi esistente.

Postcondizione: nessuna coppia di chiavi associata all'utente; blocco della ricreazione fino a nuova registrazione.

Gestione e protezione delle chiavi Il server **memorizza le chiavi private degli utenti in forma cifrata.**

Ordine operativo (flusso tipico coerente con la consegna)

1. **Registrazione off-line:** consegna all'utente della chiave pubblica del DSS e della password iniziale.
2. **Stabilire il canale sicuro** con il DSS (proprietà PFS, integrità, no-replay, non-malleabilità).
3. **Autenticarsi:** validare il server tramite la sua chiave pubblica; autenticare l'utente via password.
4. **Cambio password al primo accesso.**
5. **Creazione chiavi (una tantum):** invocare **CreateKeys** se l'utente non ha ancora una coppia.
6. **Uso ordinario:**
 - per firmare un documento: **SignDoc(documento)**;
 - per ottenere la chiave pubblica di un utente: **GetPublicKey(utente)**.
7. **Cessazione:** se l'utente vuole dismettere le proprie chiavi, invoca **DeleteKeys**. Per poterle avere di nuovo, sarà necessaria **una nuova registrazione off-line.**

Vincoli e requisiti da rispettare

- Tutte le operazioni applicative avvengono **dopo** l'instaurazione del canale sicuro.
- **CreateKeys** è **idempotente** se la coppia esiste già.
- **DeleteKeys** ha effetto **vincolante**: impedisce la creazione di nuove chiavi fino a nuova registrazione.

- Le chiavi private sono **sempre archiviate cifrate** lato server.

Contenuti obbligatori della relazione La relazione del progetto deve includere:

1. **Specifiche e scelte progettuali**, con particolare attenzione al **protocollo di autenticazione** tra utente e servizio.
2. **Formato di tutti i messaggi** scambiati (livello applicativo).
3. **Diagrammi di sequenza** di ogni protocollo di comunicazione utilizzato (livello applicativo).

2 Digital Signature Server

2.1 __init__ Questa è la funzione costruttore della classe.

- **host='0.0.0.0'**: L'indirizzo IP su cui il server ascolterà le connessioni. Il valore '0.0.0.0' indica che il server sarà accessibile su tutte le interfacce di rete disponibili del computer (locale, LAN, Internet).
- **port=5000**: La porta su cui il server ascolterà le connessioni in ingresso.
- **certfile='server-cert.pem'**
- **keyfile='server-key.pem'**
- **self.server_socket = socket.socket(...)**: Crea un nuovo socket TCP/IP.
 - **socket.AF_INET**: Specifica che il server utilizzerà l'IPv4 per la comunicazione di rete.
 - **socket.SOCK_STREAM**: Indica che il socket è di tipo stream (flusso)
- **self.server_socket.bind(...)**: Associa il socket a una porta e un indirizzo specifici sul server, (0.0.0.0 e 5000).
- **self.server_socket.listen(5)**: lunghezza massima della coda di connessioni in attesa.

3 Autenticazione

L'autenticazione del server costituisce un punto cardine del protocollo: è il meccanismo che permette all'utente di accertarsi di comunicare effettivamente con il *Digital Signature Server* (DSS) e non con un attaccante (*man-in-the-middle*). A tal fine, il progetto adotta due scelte fondamentali:

1. L'utilizzo dello schema di firma digitale **Ed25519**.
2. L'inclusione di un identificatore di protocollo/versione (`PROTO = "DSS/1"`) nel *transcript* dell'handshake.

3.1 Ed25519 è uno schema di firma digitale basato su curve ellittiche, che utilizza l'algoritmo EdDSA (*Edwards-curve Digital Signature Algorithm*) sulla curva Curve25519.

Motivazioni della scelta

- **Sicurezza elevata:** offre un livello di sicurezza paragonabile a RSA-3072, ma con chiavi e firme molto più corte.
- **Efficienza:** genera e verifica firme in tempi ridottissimi, riducendo il carico computazionale.
- **Compattezza:** le chiavi (32 byte) e le firme (64 byte) sono estremamente leggere e adatte a protocolli di rete.
- **Standard consolidato:** ampiamente utilizzato in progetti reali (OpenSSH, GnuPG, Tor), quindi ben testato e supportato.

Funzionamento nel protocollo Il server possiede una coppia di chiavi Ed25519 *a lungo termine* (generata una volta sola e fornita agli utenti in fase di registrazione offline). Durante l'handshake:

1. Viene costruito un *transcript* che lega i parametri effimeri di Diffie–Hellman, i nonces e l'identificatore di protocollo.
2. Il server calcola l'hash del transcript e lo firma con la propria chiave privata Ed25519.
3. L'utente, in possesso della chiave pubblica del server (ricevuta offline), verifica la firma.

In questo modo si garantisce che la chiave effimera e i parametri di sessione provengano realmente dal server, e non da un attaccante.

Applicazioni reali Ed25519 è usato in contesti dove sono fondamentali autenticità e prestazioni:

- **OpenSSH** come formato di chiave per autenticazione sicura dei server.
- **Protocolli di messaggistica sicura** (Signal, Wire) per firme veloci e leggere.

- **Blockchain e criptovalute** (es. Monero) per firme compatte ed efficienti.

Punti di forza

- Algoritmo moderno, con parametri ben scelti per evitare debolezze note in curve ellittiche.
- Non richiede infrastruttura PKI esterna: basta distribuire la chiave pubblica del server offline.
- Alta resistenza a implementazioni insicure (side-channel).

Possibili migliorie

- Proteggere ulteriormente la chiave privata a lungo termine con *Hardware Security Modules* (HSM) o enclave sicure.
- Introdurre un meccanismo di *certificate pinning* o catena di fiducia per evitare distribuzione manuale della chiave.

3.2 Identificatore di protocollo e prevenzione cross-protocol Durante la costruzione del transcript viene inserito un campo costante:

PROTO = "DSS/1"

Tale campo identifica in modo univoco il protocollo e la sua versione.

Motivazioni della scelta

- **Versioning**: consente di distinguere handshake di versioni differenti del protocollo (es. DSS/2) e gestire la compatibilità.
- **Difesa da attacchi cross-protocol**: impedisce che firme valide nel contesto DSS possano essere riutilizzate in altri protocolli (ad es. SSH, TLS) o in versioni diverse.

Funzionamento Poiché il transcript firmato include l'identificatore di protocollo:

- Una firma prodotta per DSS/1 non sarà valida per TLS/1.3 o per DSS/2.
- Questo vincola crittograficamente le chiavi effimere e i nonces al protocollo specifico.

Applicazioni reali

- Protocolli moderni come TLS 1.3 includono un campo *context string* nelle funzioni di derivazione delle chiavi, con finalità analoghe.
- Signal e Noise Protocol Framework usano identificatori simili per legare le chiavi a un determinato protocollo e versione.

Punti di forza

- Semplicità: l'aggiunta di un identificatore non complica l'implementazione.
- Robustezza: previene classi di attacchi sottili e difficili da rilevare (firma riutilizzata in altro contesto).

Possibili migliorie

- Utilizzare identificatori strutturati (DSS/1.0, DSS/1.1, ...) per gestire release incrementali.
- Includere anche l'elenco delle *ciphersuite* supportate per rafforzare la negoziazione.

3.3 Sintesi L'autenticazione nel DSS si fonda su:

1. Uno schema di firma digitale moderno (Ed25519) che garantisce autenticità e prestazioni elevate.
2. L'uso di un identificatore di protocollo/versione nel transcript per assicurare il corretto contesto delle firme digitali e prevenire riutilizzi illeciti.

Queste scelte rendono il protocollo robusto, sicuro e allineato alle migliori pratiche dei protocolli crittografici moderni.

4 Flusso della comunicazione: descrizione lineare e schemi

Questa sezione descrive in modo lineare come client e server instaurano il canale sicuro e come avviene lo scambio dei messaggi applicativi, senza riportare codice. I riferimenti ai “componenti logici” (es. *ClientHello*, *ServerHello*, *Transcript*, *HKDF*, *AES-GCM*) corrispondono alle parti omonime implementate nel progetto.

4.1 Obiettivo e precondizioni

- **Obiettivo:** stabilire un canale autenticato e confidenziale con *Perfect Forward Secrecy* (PFS) tra utente e DSS, per poi inviare/rispondere a richieste applicative (es. *SignDoc*).
- **Precondizioni:** l'utente possiede la **chiave pubblica Ed25519 del server** (distribuita off-line in fase di registrazione).

4.2 Handshake: negoziazione, autenticazione del server, derivazione di K

1. **Caricamento chiave del server (lato client).** Il client carica la chiave pubblica Ed25519 del DSS (ottenuta off-line). Servirà a verificare la firma del server.
2. **Connessione TCP.** Il client apre una connessione verso il DSS; il server accetta.
3. **ClientHello.** Il client genera una chiave *effimera* X25519 per la sessione e un nonce N_c , quindi invia al server la propria chiave pubblica effimera e N_c .
4. **ServerHello con firma.** Il server genera a sua volta una chiave effimera X25519 e un nonce N_s , calcola lo *shared secret* ECDH e costruisce il **transcript** (Tab. 1). Il server firma l'hash del transcript con la propria chiave privata Ed25519 e invia al client: la sua chiave pubblica effimera, N_s e la **firma** sul transcript.
5. **Verifica autenticità del server (lato client).** Il client ricostruisce lo stesso transcript e ne verifica la firma con la *pubblica* Ed25519 del server. Se la verifica fallisce, la sessione viene interrotta.
6. **Derivazione condivisa della chiave di sessione K .** Entrambe le parti calcolano lo stesso *shared secret* ECDH. Da questo estraggono e *legano al contesto* la chiave di sessione K tramite **HKDF-SHA256**, usando i parametri di Tab. 2. In questo modo K è univoca per la sessione e vincolata al transcript (protocollo, chiavi effimere, nonces).
7. **Key-confirmation bidirezionale.** Client e server si scambiano due brevi messaggi autenticati (*ClientFinish* e *ServerFinish*) cifrati con K . Ogni lato verifica di poter decifrare e convalidare il tag di autenticazione, dimostrando di conoscere K .

Table 1: Transcript di handshake (bytes in ordine canonico).

Campo	Significato
PROTO	Identificatore protocollo/versione (es. DSS/1); previene riusi cross-protocol.
client_pub	Chiave pubblica <i>effimera</i> X25519 del client (PFS).
server_pub	Chiave pubblica <i>effimera</i> X25519 del server (PFS).
N_c	Nonce lato client (anti-replay; contribuisce a unicità sessione).
N_s	Nonce lato server (anti-replay; contribuisce a unicità sessione).

Table 2: Parametri di derivazione HKDF per la chiave di sessione K .

Parametro	Valore/ruolo
input keying material	<i>shared secret</i> da ECDH X25519.
salt	Concatenazione $N_c N_s$; garantisce unicità tra sessioni.
info	$\text{hash}(\text{transcript})$; lega K al contesto (protocollo, chiavi, nonces).
length	32 byte (idoneo ad AES-256-GCM).

Esito dell'handshake. Se le verifiche vanno a buon fine, client e server condividono la stessa chiave K e l'hanno dimostrato (key-confirmation). Il canale sicuro è *attivo*.

4.3 Fase applicativa: confidenzialità, integrità e anti-replay Una volta attivo il canale:

- **Cifratura/autenticazione:** tutti i messaggi applicativi sono protetti con **AES-GCM** sotto la chiave K , ottenendo confidenzialità, integrità e autenticità (tramite il tag AEAD).
- **Anti-replay:** ogni messaggio include un contatore di sequenza *crescente* e un nonce applicativo *unico*; inoltre, una *AAD* (dati autenticati aggiuntivi) lega ogni messaggio al contesto di sessione (tramite l'hash del transcript) e ai parametri anti-replay (seq, nonce).
- **Vincolo richiesta-risposta:** la risposta del server è crittograficamente legata alla richiesta (riutilizzando lo stesso contesto AAD) così da impedire riordinamenti o riusi fuori contesto.

4.4 Mappa concettuale delle operazioni

Autenticazione del server. Realizzata firmando l'hash del transcript con Ed25519; verificata dal client con la chiave pubblica distribuita off-line.

PFS (Perfect Forward Secrecy). Ottenuta grazie alle chiavi X25519 *effimere* per sessione su entrambe le parti.

Derivazione K . HKDF-SHA256 con `salt = $N_c || N_s$` e `info = hash(transcript)` garantisce una chiave unica e contestualizzata.

Integrità e non-malleabilità. AES-GCM (AEAD) rifiuta qualsiasi modifica ai messaggi (tag non valido).

Protezione dal replay. Combinazione di *seq* crescente, *nonce* unici e AAD che incorpora contesto, seq e nonce.

4.5 Esempi di uso nella fase applicativa Una volta attivo il canale, le operazioni applicative (`Login`, `CreateKeys`, `SignDoc`, `GetPublicKey`, `DeleteKeys`) viaggiano all'interno del tunnel cifrato:

- **Richiesta:** il client invia un messaggio cifrato/autenticato con K , includendo *seq*, *nonce* e AAD vincolata al transcript.
- **Risposta:** il server elabora, quindi invia la risposta cifrata/autenticata con K , riutilizzando lo stesso contesto AAD per legare la risposta alla richiesta.