Documentazione Progetto Sicurezza dell'Informazione

Riccardo Aldrovandi

June 2025

1 Scopo del Progetto

Questo progetto implementa un sistema di comunicazione sicura tra un client e un server, in particolare grazie a:

- **ECDH**, Una versione di Diffie-Hellman basata su curve ellittiche, che offre la stessa sicurezza con chiavi molto più piccole. Questo permette di cifrare la chiave pubblica ECDH con RSA senza incorrere nei limiti di dimensione del blocco, a differenza della normale versione DH che cifrata con RSA (2048 bit) riscontrava dei problemi.
- **RSA** per proteggere lo scambio delle chiavi pubbliche ECDH.
- **Firma digitale**, per garantire l'autenticità delle due parti.
- **Cifratura simmetrica** con meccanismo di integrità tramite il protocollo *ChaCha20-Poly1305*, ampiamente usato nei dispositivi di tutti i giorni specialmente in quelli mobile, grazie alla sua efficienza e robustezza.
- **Nonce univoci** per evitare attacchi di tipo replay.

2 Architettura del progetto

Il progetto è strutturato nel seguente modo:

- **Server.java** accetta connessioni client e stabilisce canali sicuri..
- **Client.java** si connette al server e comunica attraverso un canale cifrato.
- **CryptoUtils.java**, classe che si occupa di gestire le chiavi asimettriche, dal caricamento da memoria a cifratura, decifratura, firma e verifica.
- **NonceGenerator.java**, classe il cui scopo è di garantire nonce univoci, grazie ad un contatore interno.
- **ReplayAttackException.java** usata per disgtinguere eccezioni generate da replay attack.

3 Autenticazione e scambio di chiavi

Per prima cosa il server invia al client un nonce generato da un PRNG che verrà usato assieme alla chiave concordata successivamente per creare la chiave simmetrica. Il numero random è necessario onde evitare che un intrusore esegui un attacco di tipo replay, infatti nel caso che la chiave concordata abbia lo stesso parametro per due sessioni diverse, comunque essa risulterà diversa. Supponendo, per comdoità, che il server abbia le chiavi pubbliche di tutti i client salvate in locale (e che i client abbiano la chiave pubblica del server), il client invia il suo id al server grazie al quela caricherà in memoria la pub key del client. Gli host generano una coppia di chiavi ECDH che verranno cifrate con RSA e firmate digitalmente, in questo modo è garantita sia l'autenticità dei due host ed è rispettata la confidenzialità. La firma viene verificata all'arrivo e la verifica fallisce la socket si chiude. Per gestire le varie fasi di utilizzo delle chiavi asimmetriche, si utilizza una classe ausiliaria chiamata CryptoUtils. In particolare le chiavi usate sono di tipo DER, con la pubblica in formato X.509 e la privata PKCS8. Le due parti ora sono pronte a derivare una nuova chiave simmetrica, usando il nonce e quella appena ottenuta, in particolare tramite SHA-256 si genera una chiave di 256 bit usata dall'algorimto ChaCha20.

4 Comunicazione sicura

Ogni messaggio è inviato assieme alla sua lunghezza, che lo precede, questa potrebbe essere una vulnerabilità che un attaccante potrebbe sfruttare. Pertanto pur mantenendo questo protocollo, viene inviato un messaggio unico cifrato che contiene sia la lunghezza che il messaggio, a sua volta preceduto dalla lunghezza del cifrato totale inviata in chiaro. La del messaggio originario è contenuta nel payload del cifrato. ChaCha20-Poly1305 ha sia una parte di cifratura e una di MAC in tal modo oltre a garantire la confidenzialità cercata, se un messaggio viene manomesso viene subito rilevato, come si può vedere eseguendo il programma e scegliendo di manomettere un messaggio. Ogni messaggio è inviato assieme un nonce univoco (Vettore di inizializzazione, IV), questo passaggio è fondamentale ed è forse la parte più importante del progetto, infatti se nel giro di una sessione due nonce si ripetono, la chiave di cifratura è compromessa e di conseguenza tutti i messaggi scambiati. La classe NonceGenerator si occupa di generare nonce univoci che non si ripetono, grazie ad un contatore di 64 bit (long, evitando casi di overflow). Nonostante la parte randomica del nonce rimane la stessa per tutta la sessione, ogni messaggio sarà cifrato in modo diverso, grazie appunto al contatore. In particolare il nonce nei 4 Byte MSB ha contiene il numero random e nei restanti 8 Byte LSB è definito dal contatore. In fase di decifratura si verifica inizialmente che il nonce non sia mai stato usato, mantenendo uno storico(Set) di quelli usati in precedenza, mitigando così replay attack. Se il nonce è valido il messaggio viene decifrato e infine viene controllato grazie al MAC Poly1305 se l'integrità del messaggio è stata compromessa, avvertendo l'host in caso di evenienza.

Un utente Client può simulare una compromissione del messaggio.

Meccanismo	**Descrizione**
ECDH	Scambio sicuro di chiavi
RSA	Protezione delle chiavi pubbliche ECDH
Firma digitale	Protezione contro man-in-the-middle
ChaCha20-Poly1305	Cifratura simmetrica autenticata
Nonce + replay check	Protezione contro replay attack
Messaggio “COMPROMISED”	Notifica in caso di corruzione/alterazione

Table 1: Meccanismi di sicurezza adottati

5 Miglioramenti ed Estensioni

Questo progetto sicuramente ha delle vulnerabilità, come ad esempio il caso in cui un attaccante riesca a rubare la chiave privata e pubblica di un client, riuscirebbe ad impersonarsi in quella persona. Per il primo sarebbe necessario creare una catena di fiducia con una Certification Authority, in questo modo ogni utente avrebbe il suo certificato che garantisce la sua identità e nel caso verrebe gestita anche la revoca. Sarebbe buona prassi utilizzare TLS per avere una sicurezza a livello implementativo e IPSec invece per il Layer 3 TCP/IP.Infine si potrebbe aggiungere un timestamp incluso nei messaggi e un meccanismo di identificazione attiva fra Client e Server.