Vulnerabilità nell'uso dello scambio di chiavi di Diffie-Hellman

Corso di Crittografia

AA 2023/2024

Simonazzi Gian Marco

Materiale e strumenti

La presentazione mostrerà alcuni casi di attacco ad uno scambio di chiavi in modo teorico e poi tramite un esempio in codice.

Verrano mostrati come esempi degli esercizi tratti dalle piattaforme CryptoHack e OliCyber (Olimpiadi italiane di cybersecurity).

Il codice mostrato è in linguaggio Python, eseguito in Jupyter Notebooks. Per alcuni esempi verrà utilizzato l'interprete SageMath.

Crittografia simmetrica e asimmetrica

Il limite principale della crittografia simmetrica è la condivisione della chiave di cifratura. Questo limite diventa importante se la comunicazione tra due attori avviene attraverso un canale di comunicazione non privato (es. radio, ethernet, ecc...).

La crittografia asimmetrica risolve questo problema introducendo meccanismi come le coppie di chiavi pubbliche e private (es. RSA). Tuttavia questi algoritmi asimmetrici sono spesso molto più onerosi in termini di risorse di computazione e banda rispetto alle controparti simmetriche.

Lo scambio di chiavi di Diffie-Hellman permette di ottenere i vantaggi di entrambi i mondi:

- Scambio asimmetrico di una chiave di cifratura
- Comunicazione cifrata tramite cifratura simmetrica

Lo scambio di chiavi

1. Alice e Bob scelgono di comune accordo un numero primo grande p ed un generatore $g \in \mathbb{Z}_p^*$
2. Alice sceglie casualmente la sua chiave privata $a \in \mathbb{Z}_{p-1} \setminus \{0, 1\}$ e trasmette a Bob il valore

$$A = g^a \mod p$$

3. Bob sceglie casualmente la sua chiave privata $b \in \mathbb{Z}_{p-1} \setminus \{0, 1\}$ e trasmette ad Alice il valore

$$B = g^b \mod p$$

4. Alice e Bob calcolano rispettivamente

$$g^{ab} = (B)^a \mod p = (A)^b \mod p$$

Il valore comune ai due attori $g^{ab} \mod p$ è la chiave condivisa.

La sicurezza dello scambio

Si noti che

$$a = \log_g A \mod p$$

La sicurezza del sistema dipende dalla difficoltà del problema del logaritmo discreto

Saper risolvere il logaritmo discreto implica saper risolvere Diffie-Hellman

La sicurezza dipende anche da alcuni requisiti:

- p deve essere un **primo sicuro**
- a, b, p devono essere **sufficientemente grandi**

Il problema del logaritmo discreto

Sebbene il problema sia normalmente difficile, esistono comunque degli algoritmi in grado di calcolare il logaritmo discreto. La loro efficienza dipende fortemente dalla scelta dei parametri durante lo scambio.

Alcuni di questi algoritmi sono:

- Algoritmo di Pohlig-Hellman
- Algoritmo di Shanks
- Index Calculus

Caso 1: Parametri a,b non sufficientemente grandi

Assumiamo che Alice prenda a troppo piccolo

Dati A, p, per trovarlo è sufficiente un semplice brute force

```
a = 2
while(pow(g,a,p) != A):
    a += 1
```

In alternativa, se g risultasse abbastanza piccolo, è possibile che

$$A = g^a < p$$

rendendo così possibile calcolare $a = \log_g(A)$

Caso 2: p primo non sicuro

Si tratta di numeri primi che favoriscono il calcolo del logaritmo discreto.

Ad esempio, si può guardare a casi in cui $p - 1$ è facilmente fattorizzabile in fattori primi piccoli.

Questa caratteristica favorisce l'algoritmo di Pohlig-Hellman.

Caso 3: modulo non primo

Assumiamo che il modulo scelto da Alice e Bob non sia non primo

In particolare, il modulo scelto n è della forma

$$n = p \cdot q$$

Assumiamo inoltre di conoscere i fattori primi di n

La difficoltà di questo attacco dipende direttamente dalla difficoltà di fattorizzazione di n

Dati A, n e la fattorizzazione di n, possiamo applicare il Teorema Cinese del Resto per dividere il logaritmo discreto

$$A = g^a \mod pq$$

nel seguente sistema di logaritmi discreti

$$\begin{cases} A \equiv g^a \mod p \\ A \equiv g^a \mod q \end{cases}$$

Risolvendo i due logaritmi discreti, possiamo poi rimettere insieme la chiave originale a

$$\begin{cases} a \equiv a_p \mod p-1 \\ a \equiv a_q \mod q-1 \end{cases}$$

Caso reale

Nel 2016 si è scoperto che nel codice sorgente della utility socat era presente (hard-coded) un modulo di 1024 bit per Diffie-Hellmann non primo.

Sebbene non vi siano prove che possano dimostrarlo, è stata mossa l'ipotesi che l'inserimento di questa vulnerabilità crittografica avesse come scopo quello di creare una "backdoor" per poter decifrare le comunicazioni degli utenti della utility.

[4, 5]

Best practice: openssl

I casi precedenti presuppongono l'uso di parametri impropri (volontariamente o no) nel contesto dello scambio di chiavi di Diffie-Hellman.

Nello sviluppo di software la scelta migliore è spesso affidarsi a librerie standard di crittografia come openssl. Questa offre interfacce per generare parametri sicuri per lo scambio di Diffie-Hellman.

```
$ openssl dhparam -out dhparam.pem 2048
$ openssl genpkey -paramfile dhparam.pem -out dhkey.pem
```

Caso 4: Man-in-the-middle

Questo tipo di attacco riguarda la possibile presenza di attori malevoli (Carol) su un canale di comunicazione intercettabile.

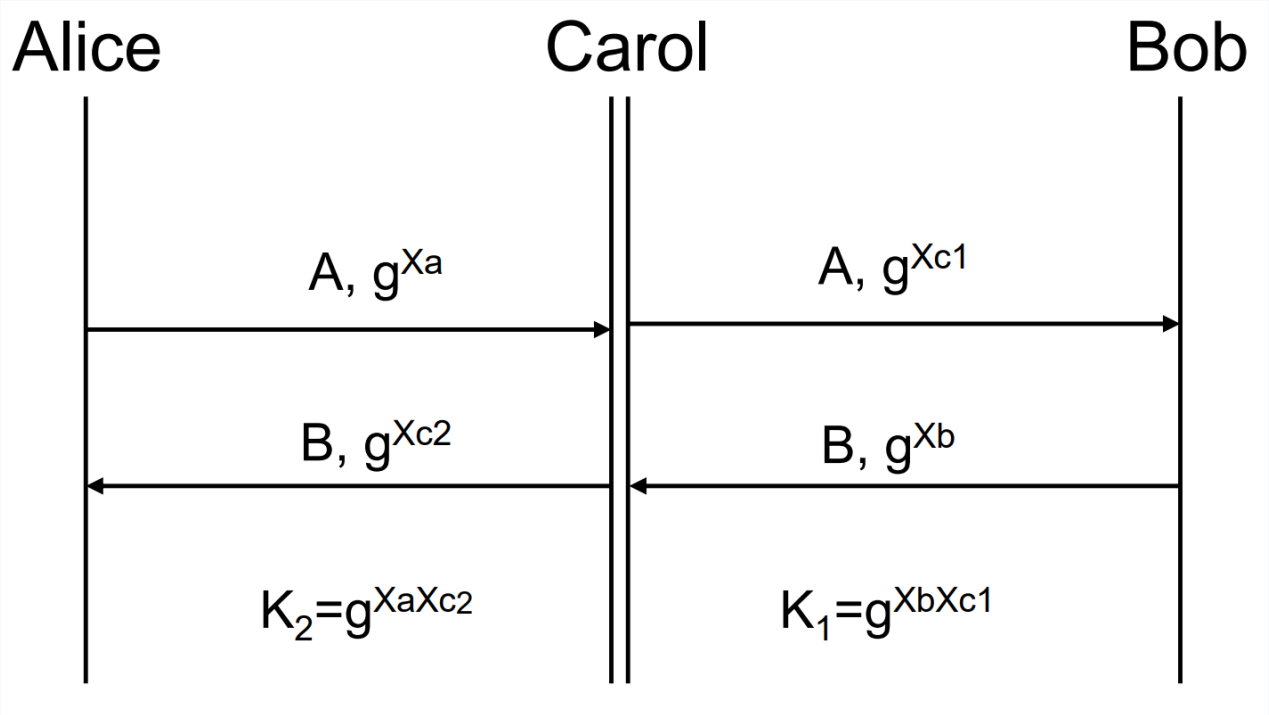
Il procedimento è il seguente:

- Alice invia p, g e g^a
- Carol intercetta la comunicazione ed invia a Bob p, g e g^c
- Bob risponde con g^b
- Carol intercetta la comunicazione ed invia ad Alice g^c

A questo punto Carol sta comunicando contemporaneamente con Alice e Bob con rispettivamente le chiavi

$$K_1 = g^{ac} \quad \text{e} \quad K_2 = g^{bc}$$

Carol può quindi replicare i messaggi in arrivo da un attore verso l'altro e risultare quindi "trasparente". Tuttavia Carol è in grado di leggere in chiaro i messaggi scambiati tra Alice e Bob.



Autenticazione

È chiaro che allo scambio manca un meccanismo di autenticazione per verificare l'identità dell'interlocutore.

Per ovviare a questo problema si può ricorrere alla crittografia a chiave pubblica. Alice e Bob firmano con una chiave privata un messaggio contenente g^a, g^b e rispettivamente le identità A, B.

$$A \rightarrow B: \quad A, g^{Xa}$$

$$A \leftarrow B: \quad B, g^{Xb}, \mathrm{Sign}_B(g^{Xa} \,||\, g^{Xb} \,||\, A)$$

$$A \rightarrow B: \quad \mathrm{Sign}_A(g^{Xa} \,||\, g^{Xb} \,||\, B)$$

Caso reale: TLS

In questa sezione si analizza il protocollo più importante che utilizza lo scambio di Diffie-Hellman, ossia TLS.

TLS è un protocollo che si posizione al livello Presentation dello stack ISO/OSI ed è responsabile della trasmissione sicura di dati tra client e server nelle comunicazioni TCP. TLS è infatti alla base del protocollo HTTPS.

TLS offre anche un meccanismo di autenticazione attraverso certificati distribuiti da Certificate Authorities (CA).

TLS Handshake (DH)

1. **Client hello**: Il client manda al server la versione di TLS utilizzata, una lista di famiglie di algoritmi di cifratura compatibili e un random.
2. **Server Hello**: Il server sceglie un'algoritmo tra quelli proposti dal client. Oltre a questo comunica anche il certificato SSL e i parametri per DH.
3. **Authentication**: Il client verifica il certificato tramite la CA. Se l'identità è verificata, il client risponde con i suoi parametri DH.
4. **Pre-master secret**: Entrambi calcolano la chiave condivisa per l'algoritmo di cifratura scelto.
5. **Ready**: Entrambi mandano un messaggio cifrato per segnalare la riuscita dell'handshake.

Perfect forward secrecy

Dalla versione 1.3, ogni comunicazione TLS deve implementare il meccanismo di perfect forward secrecy.

Questa proprietà richiede la generazione di nuova chiave privata per ogni sessione. Facendo così, la compromissione di una chiave durante la sessione non implicherebbe la compromissione di sessioni nel futuro.

Per questo motivo TLS 1.3 permette solamente lo scambio di chiavi tramite Diffie-Hellman.

Heartbleed

Nel 2014 è stata resa pubblica la presenza di una vulnerabilità software, detta Heartbleed, all'interno della popolare libreria di crittografia OpenSSL.

Questa vulnerabilità ha reso possibile l'esfiltrazione di alcune zone di memoria della macchina (sia client che server) che potevano potenzialmente contenere la chiave di sessione.

All'epoca la forward secrecy non era obbligatoria in TLS 1.2 e molti server facevano uso di RSA per effettuare l'handshake. Questo incidente ha spinto l'obbligatorietà del forward secrecy in TLS 1.3.

References I

 Olicyber.

https://training.olicyber.it.

 CryptoHack.

https://cryptohack.org.

 Heartbleed.

https://heartbleed.com/.

References II

Wong, D.

How to Backdoor Diffie-Hellman.

NCC Group, 2016.

Github Repository.

How to Backdoor Diffie-Hellman (Github)..

References III

Ferguson, N. and Schneier, B. and Kohno, T.

Il manuale della Crittografia: applicazioni pratiche dei protocolli crittografici

Apogeo, 2011.

Languasco, A. and Zaccagnini, A.

Manuale di crittografia: Teoria, algoritmi e protocolli

Hoepli, 2020.

Grazie dell’attenzione

Appendici

Implementazione Pohlig-Hellman I

L'algoritmo permette di dividere il calcolo del logaritmo discreto in $\mathbb{Z}_p^*$ nei suoi sottogruppi di ordine primo.

Assumiamo di conoscere i fattori di $p-1$ tali che

$$p - 1 = q_1^{n_1} \cdot q_2^{n_2} \cdot \dots \cdot q_k^{n_k}$$

Avremo quindi che esistono k sottogruppi di $\mathbb{Z}_p^*$ di ordine $q_i^{n_i}$.

Possiamo prendere il logaritmo discreto originale

$$y = g^x \mod p$$

ed elevare entrambi i membri della equazione

$$y^{\frac{p-1}{q_i^{n_i}}} = \left(g^x\right)^{\frac{p-1}{q_i^{n_i}}} \mod p$$

Implementazione Pohlig-Hellman II

Definiamo

$$g_i = g^{\frac{p-1}{q_i^{n_i}}} \qquad y_i = y^{\frac{p-1}{q_i^{n_i}}}$$

Siccome l'ordine di g_i è $q_i^{n_i}$, allora $x_i < q_i^{n_i}$ e quindi possiamo scrivere

$$y_i = g_i^{x_i} \mod p$$

con

$$x_i = x \mod q_i^{n_i}$$

Possiamo quindi risolvere tutti i logaritmi discreti in modo tale da ottenere tutti gli x_i per $i \in [1, \ldots, k]$

Implementazione Pohlig-Hellman III

Infine, usando il Teorema Cinese del Resto è possibile risolvere il sistema di congruenze

$$\begin{cases} x = x_1 & \mod p_1^{n_1} \\ x = x_2 & \mod p_2^{n_2} \\ \dots & \\ x = x_k & \mod p_1^{n_k} \end{cases}$$

per ottenere il valore di x.