Seguretat a la Missatgeria Instantània

Marc Cosgaya Capel

Seguretat i Protecció de Dades

Universitat de Girona

Índex

1 Introducció

Avui en dia la missatgeria instantània és cabdal. WhatsApp, per exemple, actualment té 2 mil milions d'usuaris[1]. D'aquí la importància de què aquesta sigui segura. És per aquest motiu que m'he decantat per aquest tema. En aquest treball es parlarà de les tècniques de seguretat en aquest àmbit.

El protocol SMTP i l'SMS han sigut clau per a la missatgeria actual. Però ràpidament han sigut superats pel gran ús de la missatgeria instantània. Podem comparar el nombre de correus enviats en total cada any[2] i ràpidament veurem que el volum de missatges SMTP no s'allunya gaire del volum de missatges instantanis.

D'entre les aplicacions de missatgeria instantània tenim[3]:

- WhatsApp amb 2000M usuaris mensuals actius.
- Facebook Messenger amb 1300M usuaris mensuals actius.
- WeChat amb 1251M usuaris mensuals actius.
- QQ amb 591M usuaris mensuals actius.
- Telegram amb 55M usuaris mensuals actius.
- Snapchat amb 538M usuaris mensuals actius.

Es pot veure que, en total, la suma dels usuaris actius en les aplicacions és més gran que el nombre d'usuaris mensuals actius del correu electrònic (3900M el 2019[4]).

És clar, totes aquestes aplicacions han hagut d'implementar la seguretat en un menor o major grau. Tècniques com l'*End-to-End Encryption* o la comunicació *Peer-to-Peer* han sigut clau per garantir-la. En aquest treball em centraré en la primera.

2 End-to-End Encryption

End-to-End Encryption (E2EE), tal com indica el nom, és el sistema de comunicació encriptada on només les persones implicades poden llegir el contingut dels missatges[5]. Això vol dir que ni l'empresa que proveeix la infraestructura per garantir la comunicació ni agents amb intencions malicioses poden escoltar els missatges i llegir-ne el continguts. També s'elimina la possibilitat d'una filtració de tots els missatges entre usuaris si el servidor central és accedit il·lícitament.

Això s'aconsegueix amb encriptació dels missatges. Aquests només poden ser desxifrats per les persones que s'envien els missatges. Per tant, es necessita un element secret, claus, que només aquestes tenen. Hi ha, doncs, mètodes de compartició segura de claus per després iniciar una conversa segura.

Cal tenir en compte que E2EE no és el mateix que la comunicació *Peer-to-Peer* (P2P). En P2P no hi ha presència de servidors. Això sí, E2EE i P2P es poden complementar.

3 Algorismes i Tècniques

3.1 Clau Simètrica vs Clau Asimètrica

En l'encriptació es pot fer la distinció entre algorismes de clau simètrica, o de clau privada, i algorismes de clau asimètrica, o de clau pública[6]. En els simètrics existeix una clau secreta per a cada parella d'agents, compartint la clau amb un mètode alternatiu prèviament a la comunicació. En els asimètrics existeix una parella de claus per a cada agent, on una clau pública encripta els missatges i una clau privada els desencripta.

Els algorismes asimètrics són més eficients a l'hora d'estalviar el nombre de claus necessàries per a un nombre elevat d'usuaris que els volen fer servir. Aquests algorismes, però, tenen l'inconvenient de què són més lents. Per tant, el què es fa normalment és enviar la clau simètrica amb un *handshake* asimètric. Això se'n diu *key exchange*. Un exemple d'aquesta aplicació és SSL/TLS.

Exemples d'algorismes de clau simètrica[7]:

- AES
- Salsa20/ChaCha20
- Serpent
- Twofish
- Camellia

Exemples d'algorismes de clau asimètrica[8]:

- RSA
- ECC
- El Gamal
- DSA

Exemples d'algorismes de *key exchange*[9]:

- SSL/TLS Handshake
- **Diffie-Hellman**
- **Elliptic Curve Diffie-Hellman**
- **Diffie-Hellman Ephemeral**
- **Elliptic Curve Diffie-Hellman Ephemeral**

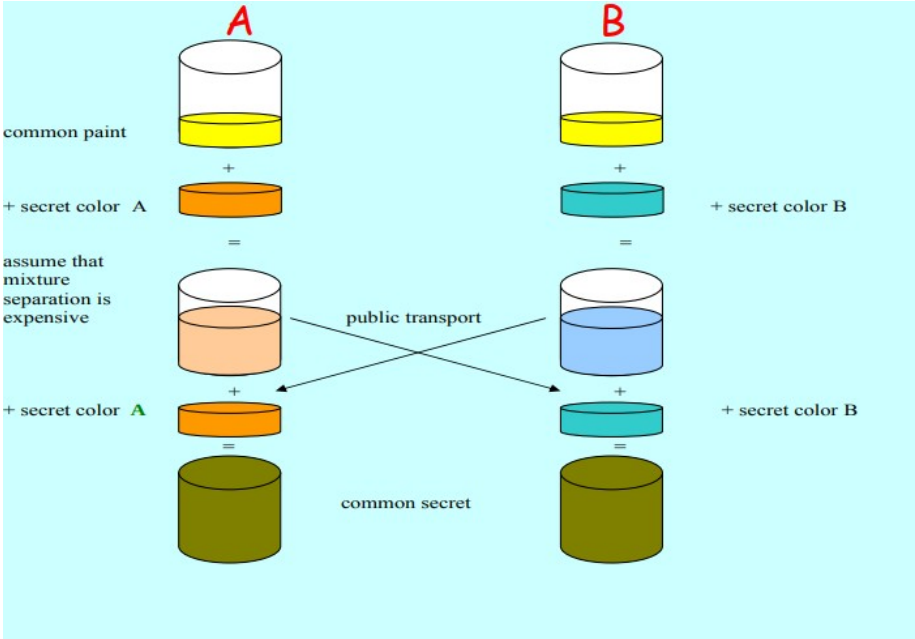
3.2 Key Exchanges

3.2.1 Diffie-Hellman (DH)

També conegut com a Diffie-Hellman-Merkle. Es fan servir dos nombre públics[10]: un mòdul *p* i una base *g* generadora. En aquest cas p ha de ser primer i g ha de ser l'arrel primitiva del mòdul p.

La clau pública *A* es genera amb $A = g^a \ mod \ p$ on *a* és un nombre a l'atzar entre *1* i *p-1*. Després s'envia a l'altre persona pel domini públic. Aquesta també haurà fet el càlcul de la seva $B = g^b \ mod \ p$ amb una *b* privada i l'haurà enviada.

Per calcular la clau simètrica *K* es genera amb $K = B^a \ mod \ p = g^{ba} \ mod \ p = g^{ab} \ mod \ p = A^b \ mod \ p$ (en vermell l'usuari A i en blau l'usuari B). Es pot veure que arriben a una mateixa *K* sense haver hagut de compartir la seva clau privada pel domini públic.

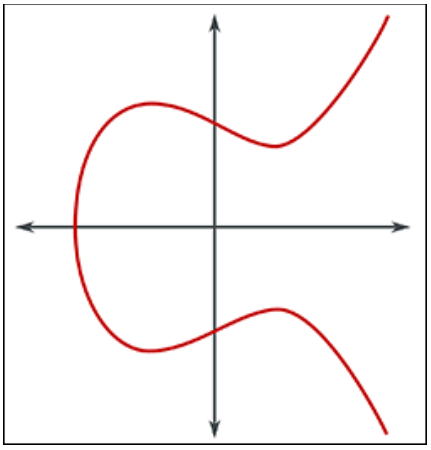


Una visualització d'aquest algorisme. Font: A.J. Han Vinck, University of Duisburg-Essen[11].

Aquest algorisme s'aprofita de la dificultat del problema del logaritme discret, que fa poc pràctic intentar trencar les claus. És molt difícil descobrir la clau privada *a* a partir de $A = g^a \ mod \ p$.

3.2.2 Elliptic Curve Diffie-Hellman (ECDH)

És una metodologia diferent de DH, però l'essència és la mateixa[12]. En aquest cas treballem amb Corbes el·líptiques. Funcions que tenen la forma $y^2 = x^3 + ax + b$ on a i b formen part del domini públic. També s'ha d'escollir un punt G públic dins la corba el·líptica com a base.



Corba el·líptica bàsica.

En aquest algorisme es fan multiplicacions escalars de punts en la corba el·líptica. És una operació que permet obtenir un nou punt dins la corba a partir d'un anterior. Per calcular la clau pública A, l'usuari A ha de fer $A = a \times G$ on a és la clau privada i x és el producte en la corba el·líptica. El mateix ha de fer B per calcular la seva clau pública amb $B = b \times G$. En aquest cas la clau simètrica K s'aconsegueix amb $K = a \times B = a \times b \times G = b \times a \times G = b \times A$.

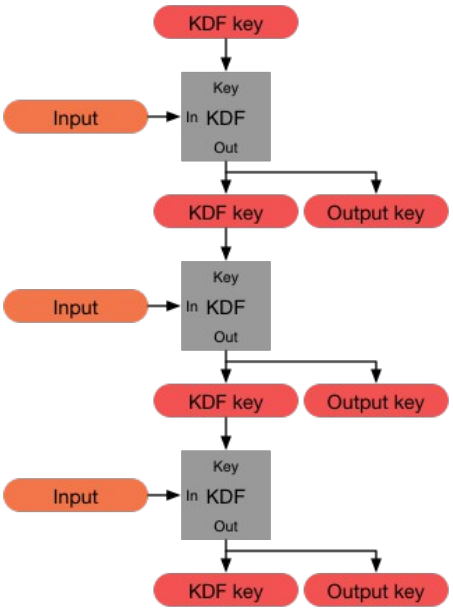
El càlcul de K és menys costós computacionalment, ja que no ha de fer servir grans nombres primers p. És per això que no necessita claus privades tan grans. L'algorisme s'aprofita de la dificultat del problema del logaritme discret per la corba el·líptica, que és més costós que el del logaritme discret.

3.2.3 Ephemeral DH i ECDH

Tant el DH com l'ECDH poden ser efímers o *ephemeral*[13] (DHE i ECDHE). Això vol dir que es fa un intercanvi de claus a l'inici de cada sessió de comunicació. Aquesta tècnica fa que no sigui pràctic intentar trencar les claus, ja que la sessió ja haurà acabat molt abans de descobrir-les.

3.3 Ratchets

És una tècnica que serveix per encriptar cada missatge amb una clau diferent. S'estableix una clau inicial al atzar i a partir d'aquí es fa servir una funció per generar la nova clau a partir de l'anterior. En cada iteració, una part de la nova clau es fa servir per encriptar el missatge.



Font: Documentació de Signal[14].

A cada iteració es genera una nova clau KDF (de *Key Derivation Function*) i una clau d'Output, que encriptarà el missatge, a partir de la clau KDF anterior i un Input. A l'Input es pot posar, per exemple, una constant del protocol o, per exemple, el resultat d'aplicar DH. L'Output es fa servir per encriptar cada missatge.

Es diu *Ratchet* perquè es fa l'analogia amb una roda dentada que només permet la rotació cap a un sentit, ja que la *Key Derivation Function* no és reversible. Fins i tot sabent el resultat d'aquesta i l'Input corresponent no és possible descobrir la clau KDF anterior.

3.3.1 Symmetric-key Ratchet

Aquesta aplicació de KDF fa servir sempre un Input constant (clau simètrica). Primer s'obté l'Input en fer DH al principi entre els clients (dos pels dos sentits). Segon, els clients fan servir aquest Input sincronitzant els Ratchets locals que tenen per anar encriptant i desencriptant cada missatge simètricament.

Encara que no es pugui descobrir la clau KDF anterior i, com a conseqüència, els Outputs anteriors, sí que es poden calcular totes les claus KDF posteriors si una és trencada per un agent maliciós. És per aquest motiu que no és recomanable fer servir aquesta aplicació sola.

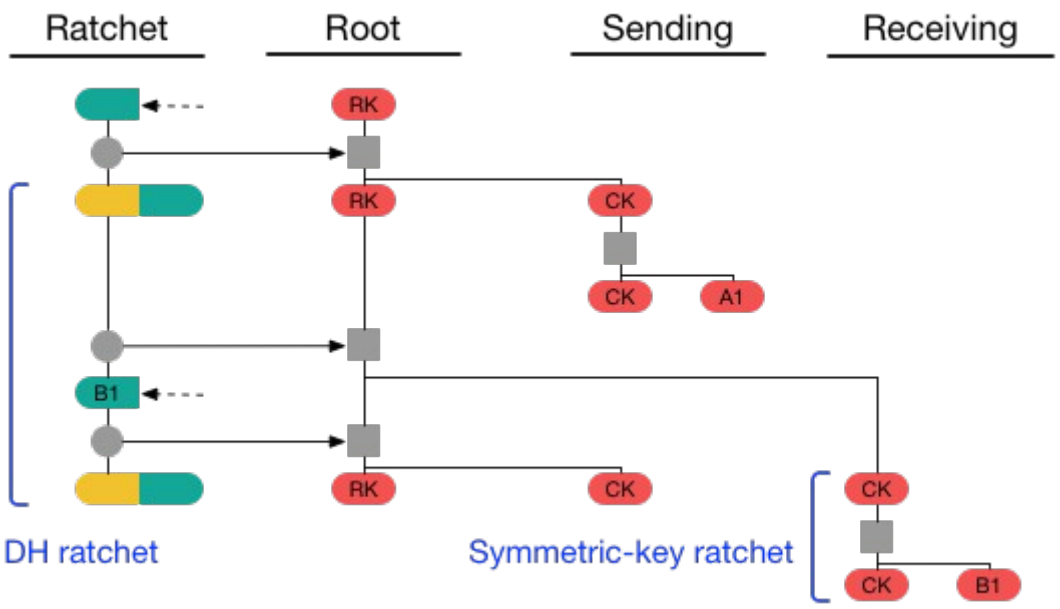
3.3.2 Diffie-Hellman Ratchet

En DH Ratchet un dels clients envia la seva clau pública i l'altre en calcula la clau simètrica. El segon envia la seva clau pública i el primer també calcula la simètrica amb la seva privada. Fins aquí és igual que Diffie-Hellman.

La diferència rau en el fet que cada cert temps un dels clients calcularà amb KDF la nova parella de claus i tornarà a haver-hi un DH entre els parts.

3.3.3 Double Ratchet

Ara, si entre cadascun d'aquests acords DH es fa servir la Symmetric-key Ratchet amb el resultat del DH anterior com a Input tenim el que anomenem **Double Ratchet** (antigament Axolotl Ratchet). Es diu *Double* perquè combina les dues aplicacions anteriors.



Font: Documentació de Signal.

Per tant, en un xat hi haurà una cadena per enviar i una per rebre per a cada client on cada cert temps s'actualitzarà els Inputs de KDF amb la sortida del DH Ratchet. Doncs, en aquesta combinació hi ha una *root* DH Ratchet i dues *sending* i *receiving* Symmetric-key Ratchets per a cada client.

Idealment, només es fa una iteració simètrica per a cada iteració DH. A més a més, les claus de les cadenes es guarden en local durant un cert temps per poder mantenir el sincronisme de les cadenes quan la comunicació no és en temps real.

4 Protocols de Missatgeria Instantània

Una llista d'alguns protocols i com implementen E2EE:

- **Signal Protocol**: Inicialment TextSecure Protocol, fa servir l'algorisme Double Ratchet amb triple Diffie Hellman (X3DH) i un sistema de pre-claus[15]. L'algorisme de clau simètrica és AES-256. Està publicat sota la llicència GPLv3[16].
- **MTProto**: Fa servir KDF que genera una AES-256 com a clau simètrica per a cada missatge[17]. La KDF té com a entrada inicial una clau establerta per DH. Per a l'Input de cada iteració es concatena el missatge en clar i part de la clau del DH inicial i se'n fa el hash. També envia l'Input juntament amb el missatge encriptat per comprovar que el hash generat sigui el mateix.
- **XMPP**: Mitjançant l'extensió OMEMO[18]. Fa servir Double Ratchet amb una versió modificada del X3DH com a *key exchange*. L'algorisme de clau simètrica és AES-256. La clau privada també s'envia encriptada a la capçalera dels missatges.
- **WebRTC**: Protocol P2P d'àudio, vídeo i missatgeria instantània que fa servir Secure Real Time Protocol (SRTP)[19] per enviar les dades i DTLS-SRTP com a *key exchange*[20]. SRTP fa servir AES. DTLS és la versió de TLS sobre UDP en comptes de TCP. Els clients mitjançant el *handshake* DTLS s'intercanvien les claus AES. El servidor de senyalització, necessari en P2P, fa servir HTTPS (HTTP amb SSL/TLS). Cal tenir en compte que la missatgeria instantània no necessàriament ha de ser en temps real.

5 Protocols en les Aplicacions

Aplicacions de missatgeria instantània i els protocols que fan servir per implementar E2EE:

- **WhatsApp**: Fa servir XMPP[21] modificat juntament amb Signal Protocol[22].
- **Skype**: Les conversacions privades fan servir Signal Protocol[23].
- **Facebook Messenger**: Les conversacions secretes fan servir Signal Protocol[24].
- **Telegram**: Les videotrucades, trucades i conversacions secretes fan servir MTProto[25].
- **Discord**: Fa servir WebRTC[26].
- **Google Meet**: Fa servir WebRTC.
- **Signal**: Fa servir Signal Protocol.

6 Fonts

1) https://techcrunch.com/2020/10/29/whatsapp-is-now-delivering-roughly-100-billion-messages-a-day/
2) https://www.statista.com/statistics/456500/daily-number-of-e-mails-worldwide/
3) https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/
4) https://financesonline.com/number-of-email-users/
5) https://web.archive.org/web/20151223002912/http://www.wired.com/2014/11/hacker-lexicon-end-to-end-encryption/
6) https://www.javatpoint.com/symmetric-encryption-vs-asymmetric-encryption
7) https://cryptobook.nakov.com/symmetric-key-ciphers/popular-symmetric-algorithms
8) https://www.omnisecu.com/security/public-key-infrastructure/asymmetric-encryption-algorithms.php
9) https://www.jscape.com/blog/key-exchange
10) https://ee.stanford.edu/~hellman/publications/24.pdf
11) https://web.archive.org/web/20140708200306/http://www.exp-math.uni-essen.de/~vinck/crypto/script-crypto-pdf/add-to-3.pdf
12) https://medium.com/swlh/understanding-ec-diffie-hellman-9c07be338d4a
13) https://www.ecdhe.com
14) https://signal.org/docs/specifications/doubleratchet/
15) https://signal.org/docs/specifications/x3dh/
16) https://web.archive.org/web/20161228221248/https://github.com/whispersystems/libsignal-protocol-java
17) https://core.telegram.org/mtproto
18) https://xmpp.org/extensions/xep-0384.html
19) https://telnyx.com/resources/webrtc-encryption-and-security
20) https://datatracker.ietf.org/doc/html/rfc5764
21) https://web.archive.org/web/20150626111834/http://shakal.blog.de/2011/03/22/whatsapp-risiken-10872342/
22) https://signal.org/blog/whatsapp-complete/
23) https://signal.org/blog/skype-partnership/
24) https://signal.org/blog/facebook-messenger/
25) https://core.telegram.org/techfaq
26) https://bloggeek.me/massive-applications-using-webrtc/