Cryptographie & Signal

Anciennement TextSecure

Daniel ARIAN

Cours de Cryptographie, M. CHENEVERT, 2021

JUNIA ISEN

Table des matières

Qu'est-ce que Signal ?

Ce tweet d'Elon Musk invitant à utiliser Signal a semblé énigmatique pour toute personne ne connaissant pas l'application : « Mais qu'est-ce que Signal ? »

Signal est une application de messagerie instantanée open-source et proposant le chiffrement de bout-en-bout par défaut. C'est-à-dire que seuls les utilisateurs finaux sont capables de voir les données en clair. Il y est possible de discuter par message avec un individu ou un groupe ou encore d'effectuer des appels vocaux et vidéo.

A l'heure actuelle, l'application est considérée comme la plus sûre et la plus respectueuse de la confidentialité parmi les applications de messagerie instantanée. Elle est même utilisée et préconisée par le lanceur d'alerte Edward Snowden pour échapper à la « surveillance de masse ».

L'histoire de Signal

En 2010, Whisper Systems lance deux applications Android : TextSecure, permettant le chiffrement des messages textuels, et RedPhone, pour le chiffrement des appels vocaux.

En 2011, Twitter acquiert Whisper Systems. Les deux applications sont alors relaxées sous forme de logiciel open source gratuit.

En 2013, l'un des fondateurs de Whisper Systems, Moxie Marlinspike, quitte Twitter. Il fonde Open Whisper Systems, startup qui continue à développer TextSecure et RedPhone.

En 2014, les deux applications fusionnent pour former Signal.

En 2018, suite à l'acquisition de WhatsApp par Facebook en 2014, le co-fondateur Brian Acton estime que la firme américaine nuit à la confidentialité de son application. Il quitte l'entreprise, s'associe à Moxie Marlinspike, et donne 50 millions de dollars pour créer l'organisation non lucrative Signal Foundation dont le but est de poursuivre le développement de l'application et de préserver sa gratuité et son caractère open-source.

Le protocole

Dans cette partie nous allons détailler les principaux protocoles et notions cryptographiques utilisées dans Signal pour garantir une conversation sécurisée entre deux agents : Alice et Bob.

1. Une vue d'ensemble des étapes du protocole

Publication des clés publiques – A l'installation de Signal, puis périodiquement, Bob et Alice s'enregistrent chacun de leur côté sur un serveur de distribution de clés et y téléversent chacun des clés publiques ayant une durée de vie courte, moyenne et longue.

Configuration de la session – Alice demande et reçoit un ensemble de clés publiques de Bob du serveur de distribution. Alice utilise ces clés pour configurer une session de messagerie durable avec Bob et établir des clés de chiffrement symétrique. C'est le *Extended Triple Diffie-Hellman* ou *X3DH protocol*.

Les deux étapes décrites ci-dessus sont traitées plus en détail dans la partie 2.

Conversation synchrone – Quand Alice veut envoyer un message à Bob (ou vice-versa) ou vient de recevoir un message de Bob, elle échange avec lui une nouvelle clé secrète (Diffie-Hellman) et l'utilise pour commencer à envoyer de nouveaux messages. Chaque nouvelle génération de clé secrète constitue une étape du « *Rachet* asymétrique » ou du « *Diffie-Hellman rachet* ».

Conversation asynchrone – Quand Alice veut envoyer un message à Bob (ou vice-versa) mais qu'elle n'a pas reçu de message de Bob depuis son dernier message à Bob, elle dérive une nouvelle clé de chiffrement symétrique pour son message depuis son état précédent en utilisant une PRF (*Pseudo-Random Function*). On dit que chaque utilisation de cette PRF est une étape du « *Rachet* symétrique ».

Ces deux étapes du protocole constituent ce qu'on appelle l'algorithme du *Double Rachet*

Les deux étapes décrites ci-dessus sont traitées plus en détail dans la partie 3.

2. Extended Triple Diffie-Hellman protocol (X3DH)

Le protocole X3DH permet d'établir une clé secrète partagée entre deux personnes qui s'authentifient mutuellement via l'utilisation clés publiques.

Ce protocole assure les principes de la **confidentialité persistance** (*forward secrecy*) et du **déni plausible** (*cryptographic deniability*).

2.1. Contexte d'utilisation

- **Alice** souhaite envoyer de premières données à Bob par le biais du cryptage et établir une clé secrète partagée avec Bob.

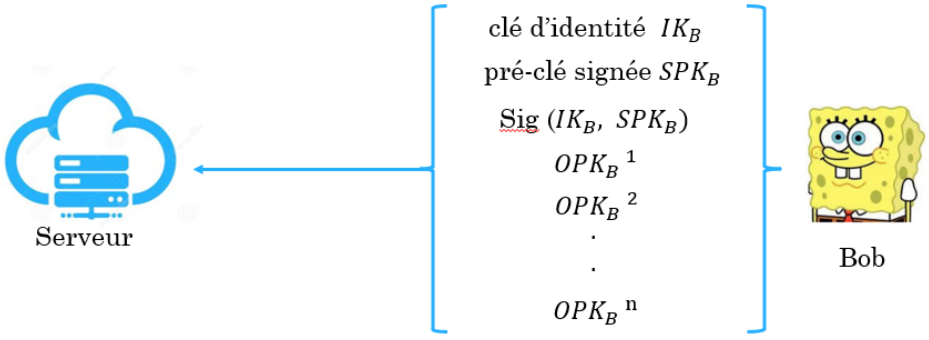
- **Bob** souhaite qu'Alice puisse établir avec lui une clé secrète partagée et lui envoyer des données chiffrées. Cependant, Bob peut être hors ligne quand Alice tente de faire cela. C'est pourquoi Bob est en relation avec un serveur.
- Le **serveur** peut stocker les messages d'Alice à Bob. Ce dernier pourra les récupérer plus tard. Le serveur donne la possibilité à Bob de publier un ensemble de clés et informations qui seront disponibles aux utilisateurs comme Alice.

2.2. Description des clés utilisées

Les clés publiques ci-dessous sont obtenues à l'aide d'une courbe elliptique (X25519 ou X448)

IK_A	Clé d'identité d'Alice (*Identity Key*) Clé ayant une longue durée de vie.
EK_A	Clé éphémère d'Alice (*Ephemeral Key*) Une nouvelle clé est générée à chaque fois que le protocole est lancé.
IK_B	Clé d'identité de Bob (*Identity Key*) Clé ayant une longue durée de vie.
SPK_B	Pré-clé signée de Bob (*Bob's Signed PreKey*) Change de manière périodique.
OPK_B	Clé jetable de Bob (*Bob's One-time PreKey*)

2.3. Publication des clés de Bob sur le serveur



Sig(IK_B, SPK_B) correspond à une signature XEdDSA obtenue en signant SPK_B avec la clé privée correspondant à IK_B.

Bob n'a besoin de publier qu'une seule fois sa clé d'identité sur le serveur. Cependant, il devra envoyer de nouvelles clés jetables (OPK_B) à l'avenir (notamment quand le serveur l'informera que leur nombre y est faible).

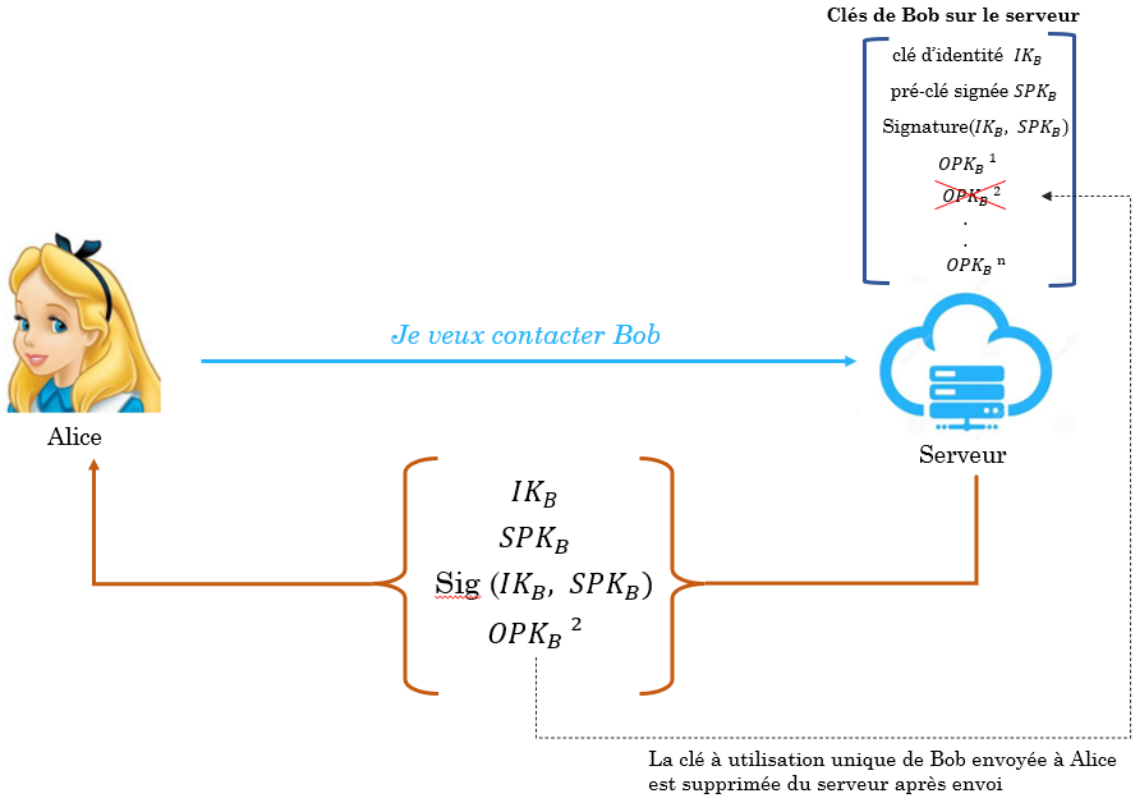
De plus, Bob doit renvoyer à intervalle de temps régulier (toutes les semaines par exemple) une nouvelle pré-clé signée et signature Signature(IK_B, SPK_B) correspondante.

Ces clés remplaceront les anciennes valeurs, qui seront supprimées afin d'assurer le principe de confidentialité persistante.

2.4. Envoi du message initial

2.4.1. Récupération des clés publiques de Bob

Afin d'établir une clé secrète partagée avec Bob selon le protocole X3DH, Alice commence par contacter le serveur pour recevoir un ensemble de clé.



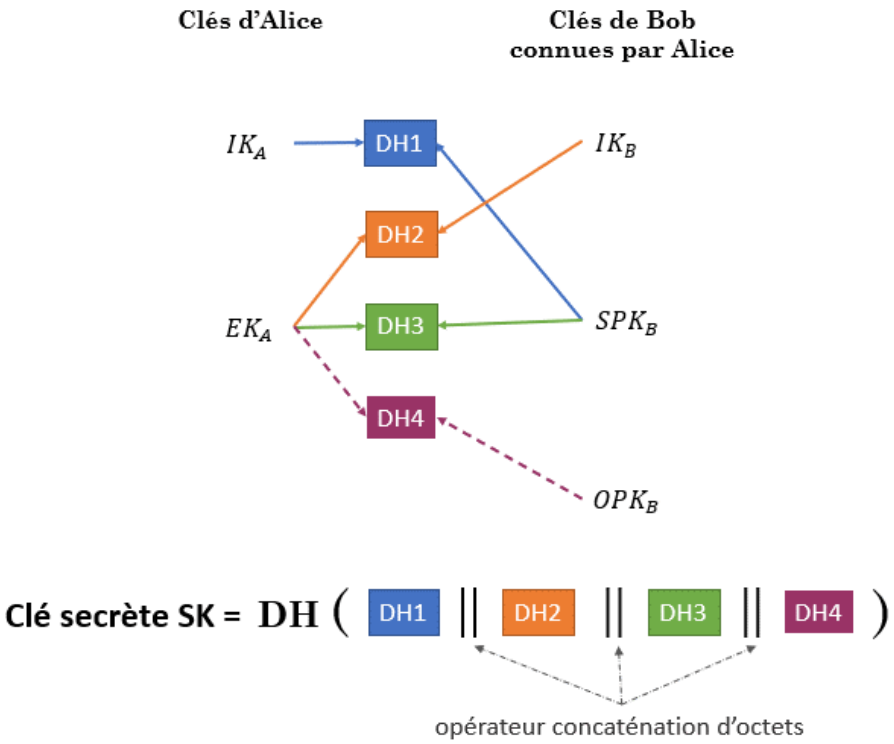
A noter : il est possible qu'il n'y ait plus de clé à utilisation unique (ou clé jetable) pour Bob sur le serveur. Dans ce cas, Alice reçoit juste la clé d'identité de Bob, sa pré-clé signée et la signature.

Alice vérifie ensuite la signature Sig(IK_B, SPK_B). Le protocole s'arrête là en cas d'erreur.

Sinon, elle génère ainsi une paire de clés éphémères dont la clé publique est EK_A qui va être utilisée pour calculer la clé secrète partagée dans la partie suivante.

2.4.2. Calcul de la Clé Secrète (Triple Diffie-Hellman)

Alice a désormais toutes les informations pour calculer la clé secrète *SK* (*Secret Key)* qui sera partagée avec Bob. Elle calcule les Diffie-Hellman (DH) suivants :

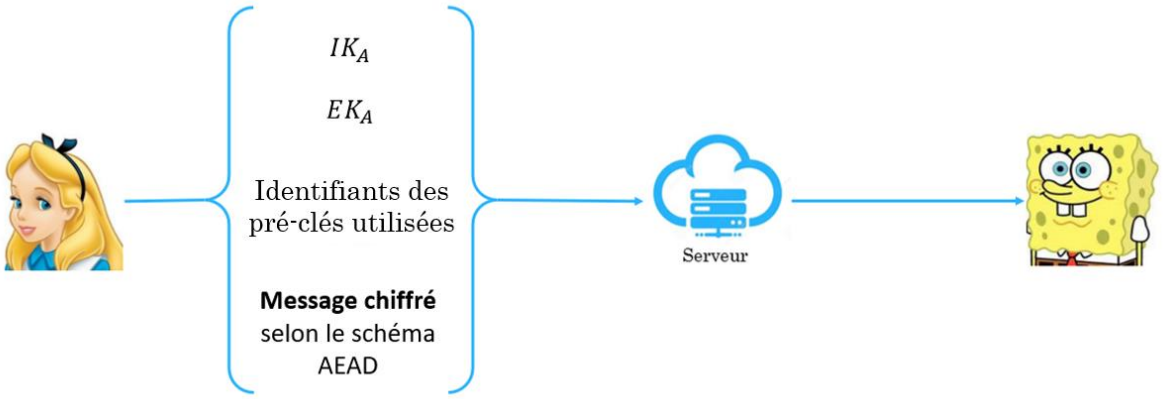


Pour lever toute ambiguïté de notation, on a par exemple DH1 = $DH(IK_A,\ SPK_B)$.

On notera également que si une clé jetable de Bob n'est pas fournie à Alice par le serveur, le calcul de SH reste le même, il suffit juste d'ignorer le terme DH4 à la concaténation.

Afin d'empêcher la reconstruction de SK, Alice supprime sa clé éphémère EK_A ainsi que l'ensemble des DH intermédiaires calculés.

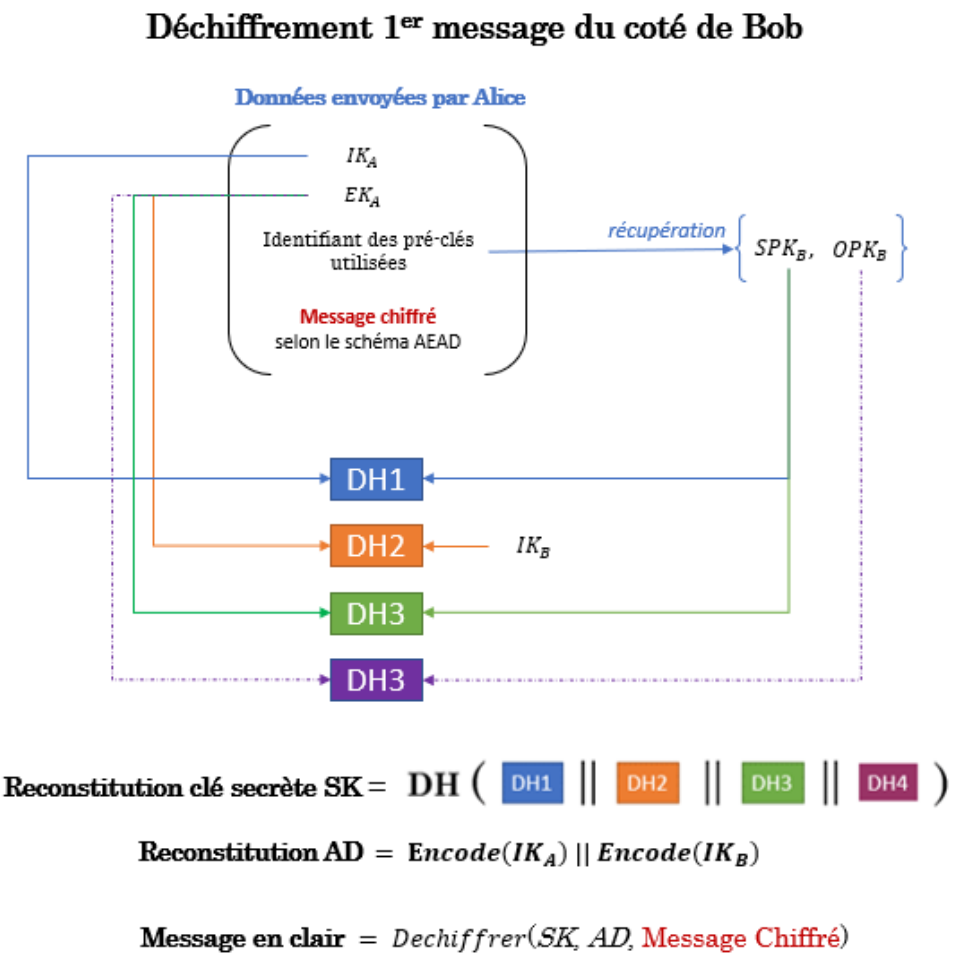
Enfin, Alice envoie le premier message à Bob en incluant les informations mentionnées sur le schéma ci-dessous :



AEAD = *Authenticated Encryption with Associated Data* ou *chiffrement authentifié avec les données associées*. Le but des données associées (AD) est de lier un texte chiffré au contexte où il est supposé apparaître, de sorte que les tentatives de « couper-et- coller » un texte chiffré valide dans un contexte différent peut être détecté et rejeté, ce qui rajoute

une sécurité. Dans notre cas, la clé de chiffrement est $\boldsymbol{SK}$ et les données associées $\boldsymbol{AD} = Encode(IK_A) \,||\, Encode(IK_B)$.

2.5. Reception par Bob du message initial d'Alice

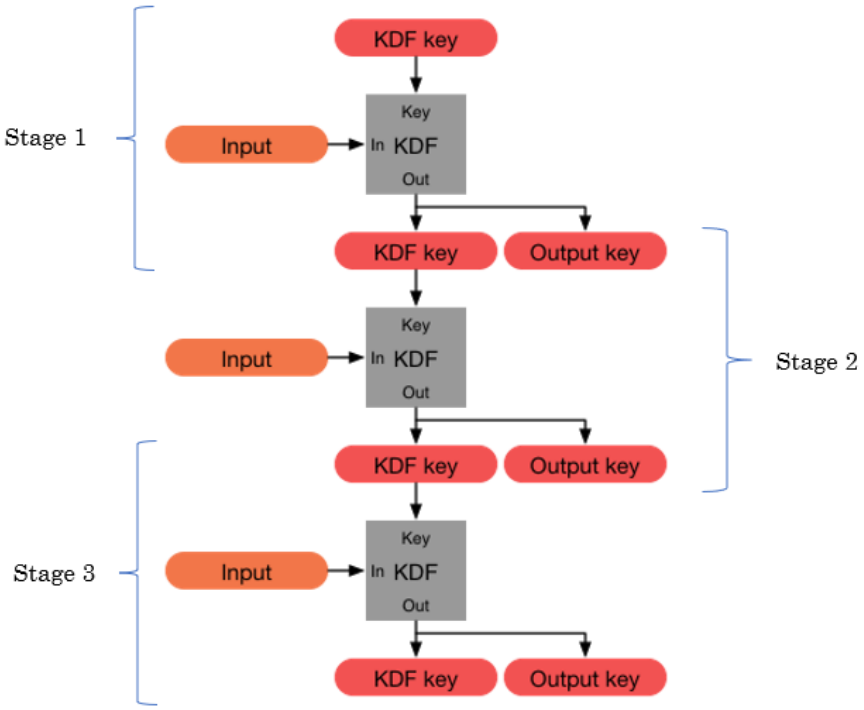


Bob supprime ensuite toutes les pré-clés utilisées ainsi que les DH intermédiaires. La clé *SK* pourra être ré-utilisée ou bien on pourra utiliser une clé dérivée de *SK*.

3. L'algorithme du Double *Rachet*

Alice et Bob ont pu se mettre d'accord sur une clé secrète grâce au protocole X3DH. Ils vont chacun pouvoir maintenant envoyer et recevoir des messages chiffrés en utilisant le *Double Rachet Algorithm*.

Pour comprendre cet algorithme, il est nécessaire d'introduire la notion de *KDF chain* (chaine de fonctions de dérivation de clé). Ci-dessous un exemple de *KDF chain* avec 3 entrées et 3 sorties :



Dans une session Double Rachet entre Alice et Bob, chacun conserve une clé en entrée d'une KDF pour trois chaines : la chaine racine *(root chain)*, la chaine d'envoi (*sending chain*) et la chaine de réception *(receving chain)*.

Au fur et à mesure qu'Alice et Bob s'échangent des messages, ils s'échangent aussi de nouvelles clés publiques DH, et les clés secrètes DH obtenues deviennent les nouvelles entrées de la chaine racine. Les clés en sortie de la chaine racine deviennent les nouvelles clés KDF pour l'envoi et la réception de messages. C'est le ***Diffie-Hellman rachet***.

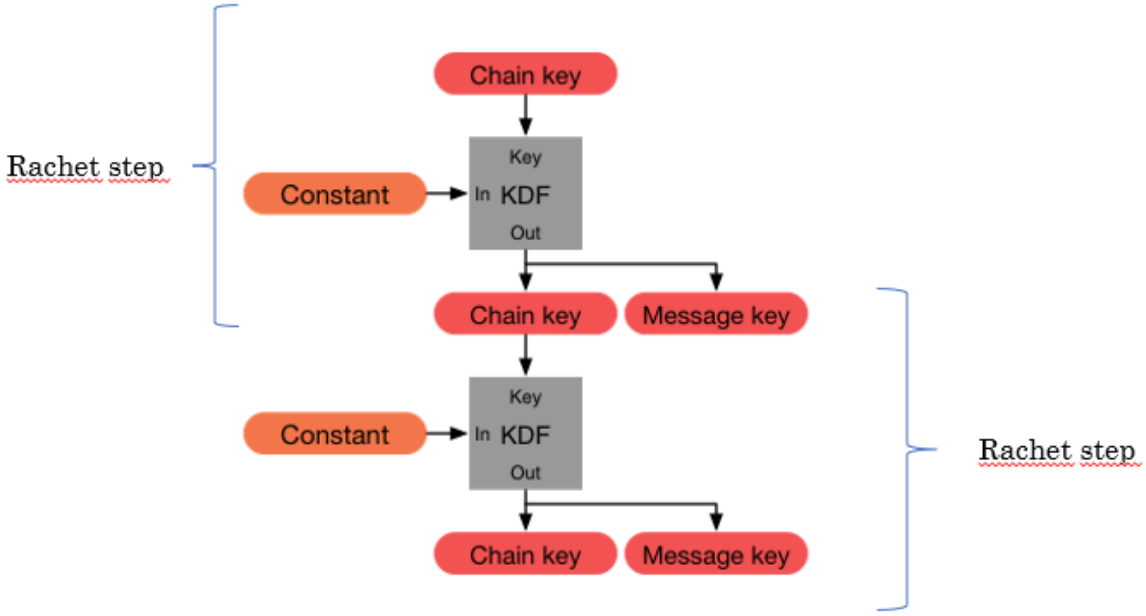
Les chaines d'envoi et de réception avancent à chaque message entrant ou sortant. Leurs clés de sortie sont utilisées pour chiffrer et déchiffrer les messages. C'est le ***rachet à clé symétrique***.

Le Double Rachet est la combinaison du *Diffie-Hellman rachet* et du *rachet à clé symétrique*.

Dans les sections suivantes, nous allons détailler comment ces deux *rachet* fonctionnent et comment ils s'assemblent pour former le *Double Rachet*.

3.1. Rachet à clé-symétrique

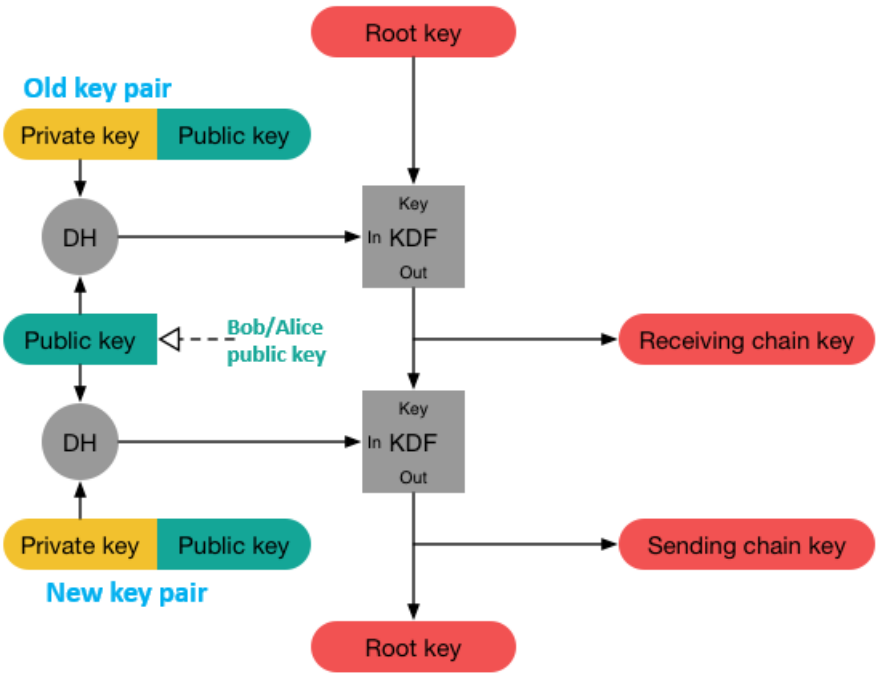
Chaque message reçu ou envoyé a été chiffré avec une clé de message jetable (*message key*). Ces clés jetables sont obtenues en sortie des chaines de fonctions de dérivation de clé d'envoi et de réception. Le calcul de la prochaine clé jetable se fait en une seule étape de rachet. Le diagramme ci-dessous illustre deux étapes :



3.2. Diffie-Hellman Rachet (*DH rachet*)

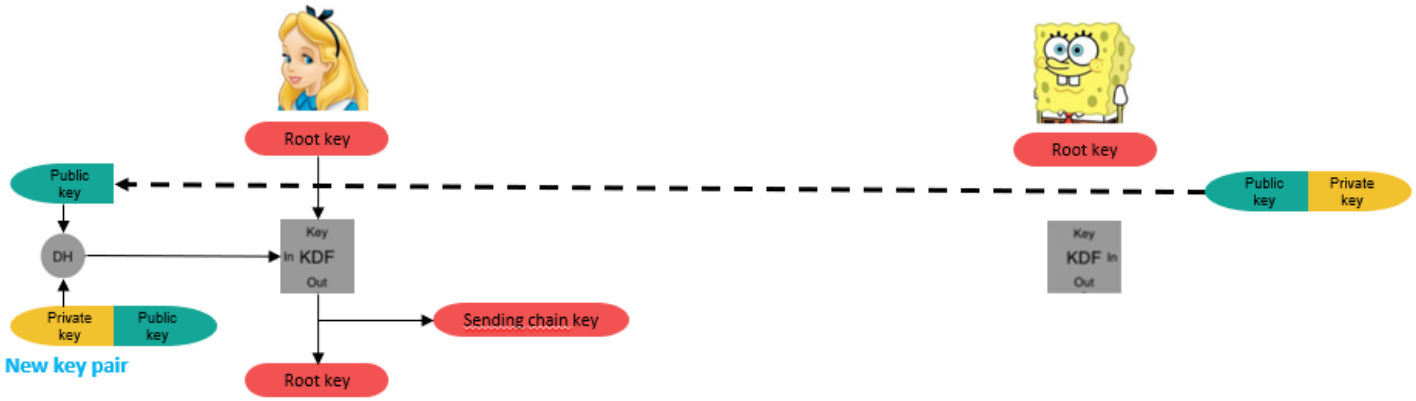
Le Rachet à clé-symétrique ne suffit pas à lui seul pour être certain d'assurer une confidentialité d'une future conversation. En effet, si un attaquant parvient à récupérer la clé de chaine (*chain key*) d'envoi et de réception de Bob ou Alice, il est capable de calculer toutes les futures clés de messages et donc de déchiffrer tous les prochains messages. Pour pallier ce problème, on combine à ce *rachet* le *DH rachet* qui se charge de mettre à jour les clés de chaines.

Pour implémenter le DH rachet, Alice et Bob génèrent une paire de clé privée et publique Diffie-Hellman. Chaque message envoyé par Alice (ou Bob) contient en en-tête la clé publique d'Alice (ou de Bob). A chaque fois qu'une nouvelle clé publique est reçue par un des parties, une étape du DH rachet est réalisée. De ce fait, la paire de clé du partie est remplacée par une nouvelle paire de clés. Ci-dessous l'illustration d' une étape du DH rachet :

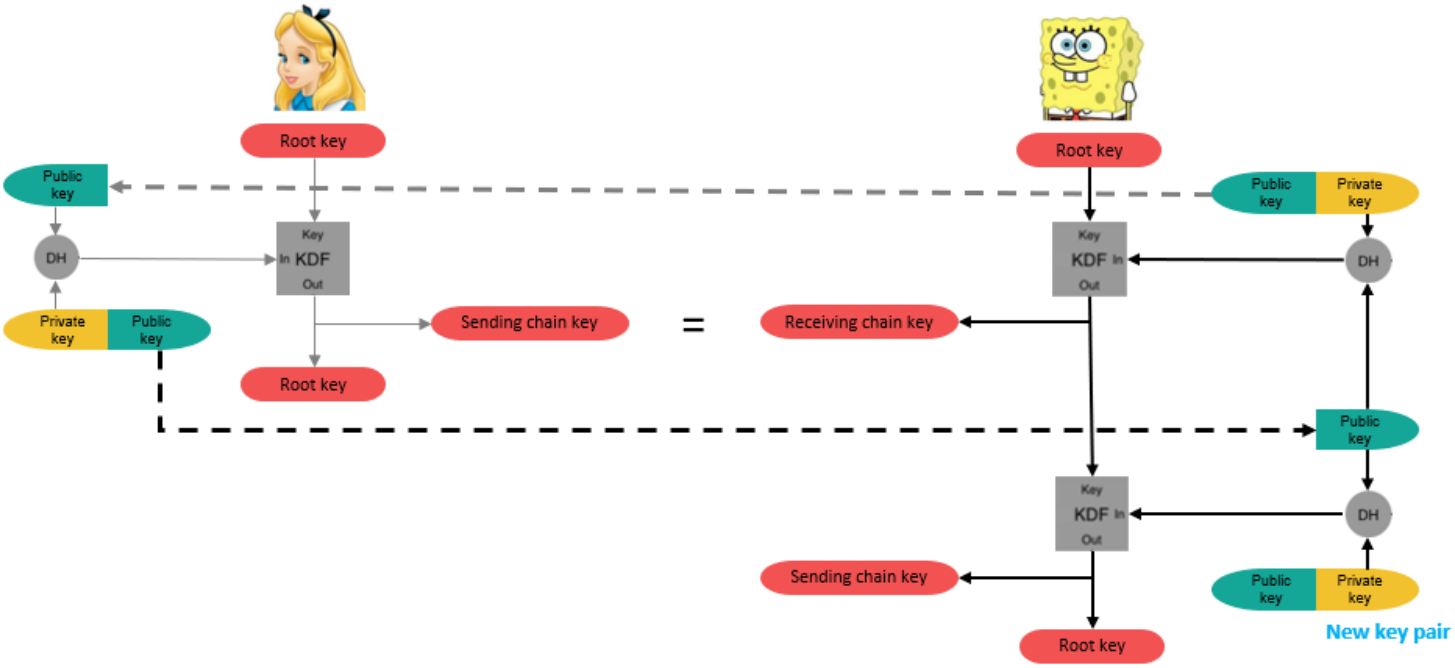


Ainsi, nous allons voir dans les diagrammes ci-dessous comment se déroule une suite de *DH rachet* dans une conversation entre Alice et Bob.

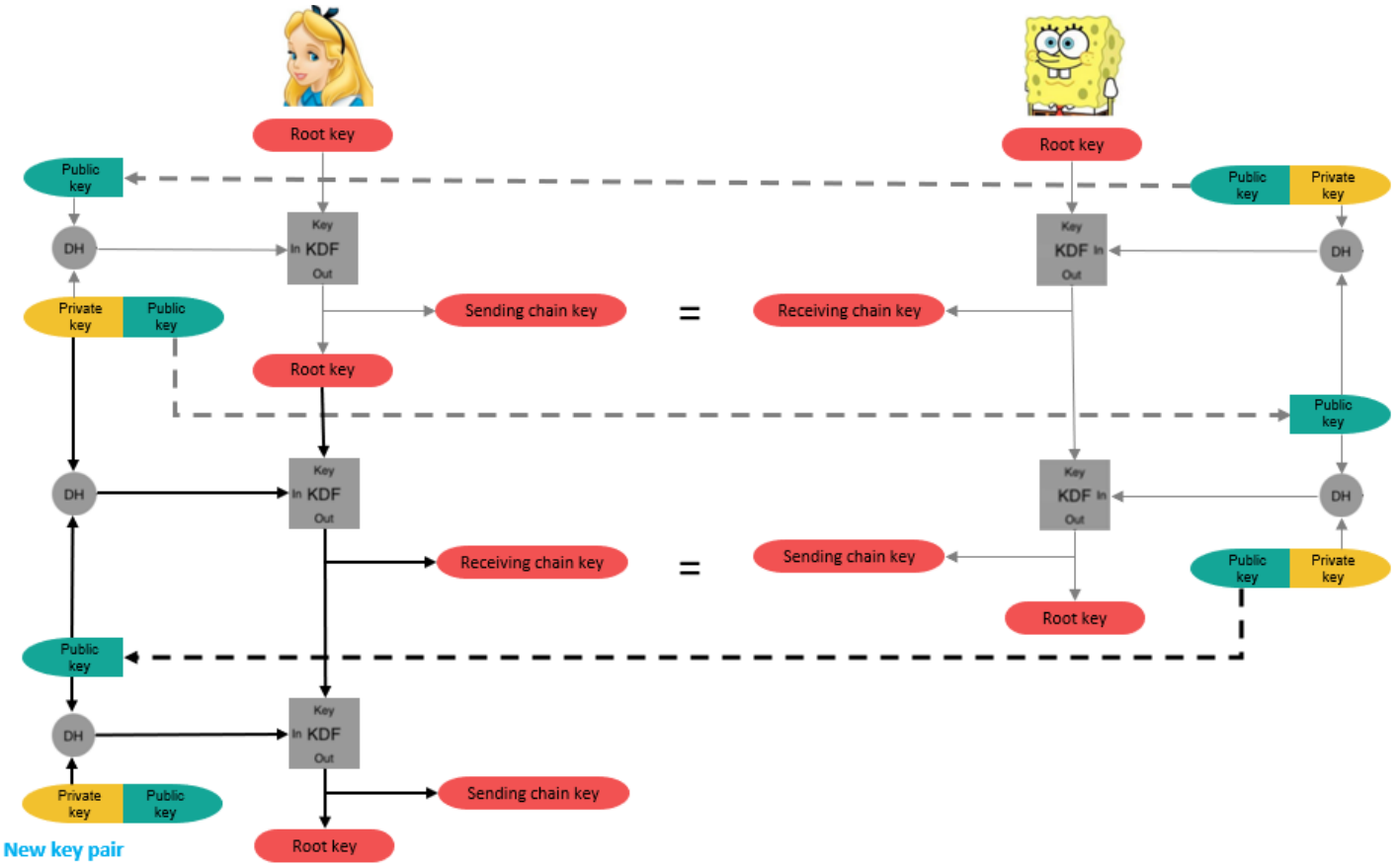
Tout d'abord, Alice prend connaissance de la clé publique de Bob. A ce stade, Bob ne connait par la clé publique d'Alice :



Alice envoie ensuite son message initial à Bob en envoyant sa nouvelle clé publique. Bob réalise alors une étape du DH rachet :



Eventuellement, Bob répond à Alice, il lui envoie alors sa nouvelle clé publique et Alice réalise une étape du DH rachet :



Le schéma continue ainsi de suite comme une partie de ping-pong…

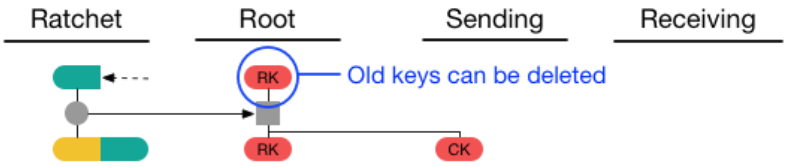
3.3. Double Rachet

La combinaison du rachet à clé symétrique et du *DH rachet* donne le Double Rachet :

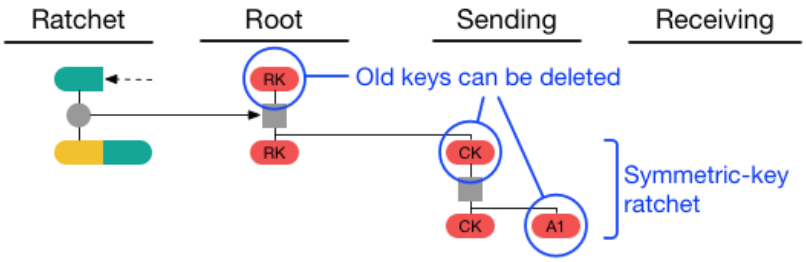
- Quand un message est reçu ou envoyé, une étape du rachet à clé symétrique est réalisée à la chaine d'envoi ou de réception pour dériver une clé de message.
- Quand une nouvelle clé publique du *DH rachet* est reçue, une étape du *DH rachet* est réalisée avant que le rachet à clé symétrique ne remplace les clés de chaines.

In the below diagram Alice has been initialized with Bob's ratchet public key and a shared secret which is the initial root key. As part of initialization Alice generates a new ratchet key pair, and feeds the DH output to the root KDF to calculate a new root key (*RK*) and sending chain key (*CK*):

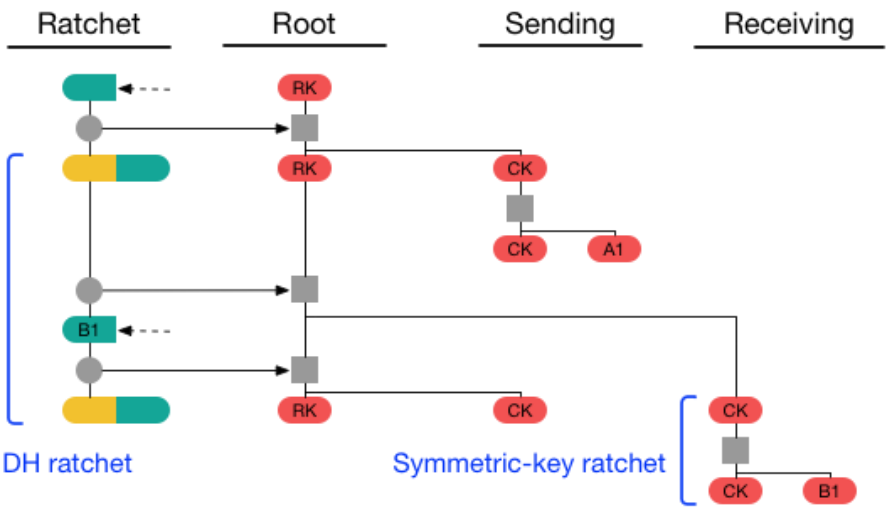
Dans le diagramme ci-dessous, c'est l'initialisation. Alice a reçu la clé publique du DH rachet de Bob ainsi que la clé initiale de la chaine de racine (*Root Key*, RK). Lors de cette initialisation, Alice génère une nouvelle paire de clé et calcule une nouvelle clé racine (RK) et la clé de chaine d'envoi (*sending Chain Key, CK).*



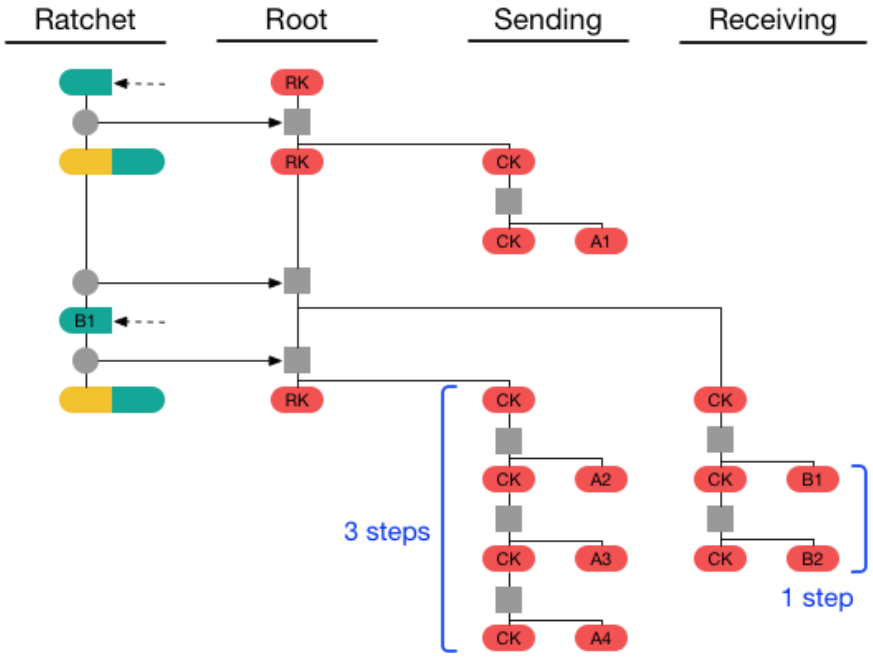
Quand Alice envoie son premier message A1, elle applique un rachet à clé symétrique à sa clé de chaine d'envoi (CK). Elle obtient ainsi une nouvelle clé de message qu'on note A1 et une nouvelle clé de chaine CK.



Si Alice reçoit ensuite une réponse B1 de Bob qui contient une nouvelle clé publique pour Bob qu'on note B1, Alice applique une étape du DH rachet pour obtenir une nouvelle clé de chaine pour la chaine d'envoi et une nouvelle clé de chaine pour la chaine de réception. Elle applique ensuite un rachet à clé symétrique à sa chaine de réception pour calculer la clé de message du message reçu.

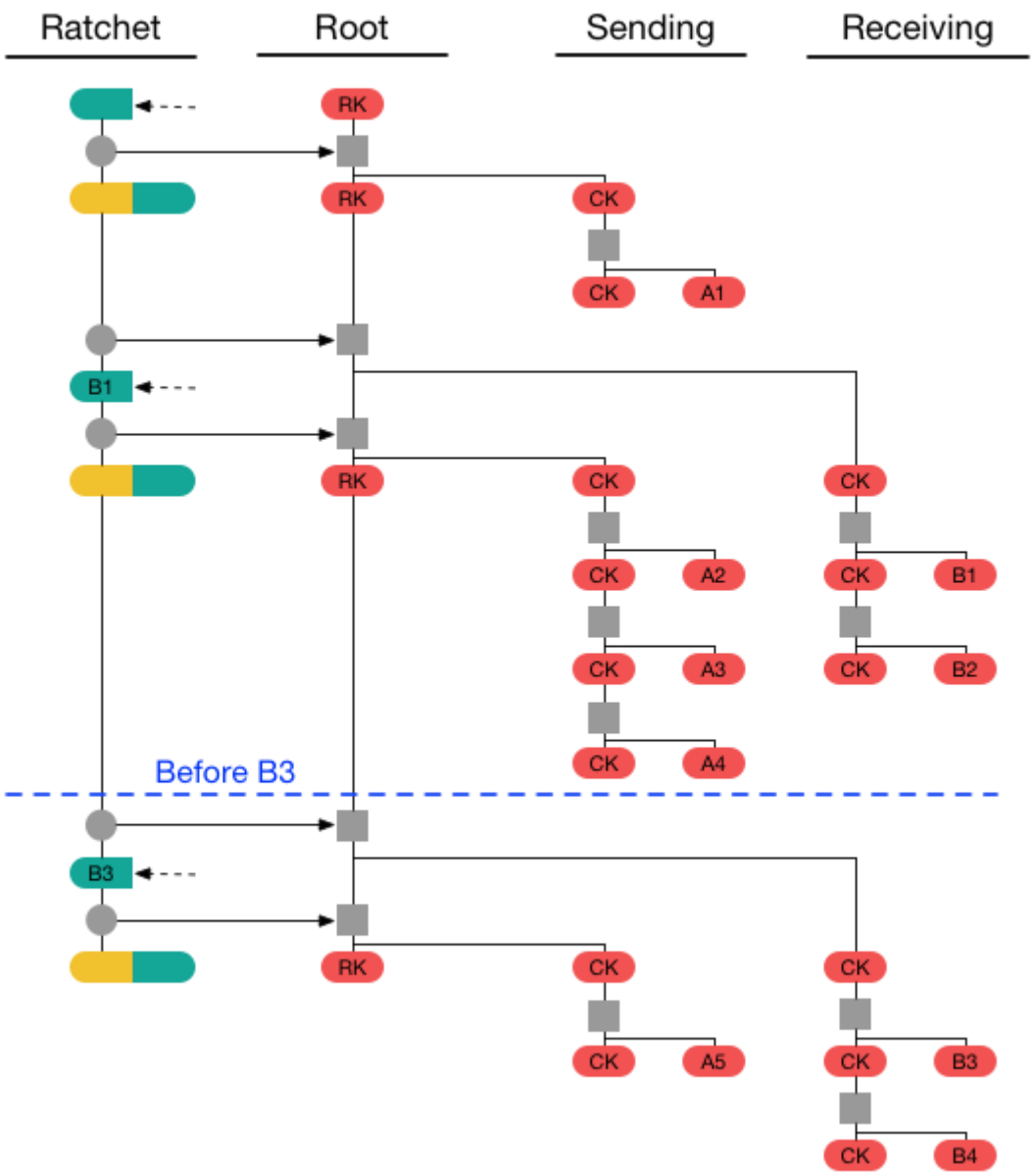


Alice envoie ensuite un message A2, elle reçoit un message B2 avec l'ancienne clé publique de Bob puis elle envoie les messages A3 et A4. Dans ce cas, Alice va effectuer trois étapes du rachet à clé symétrique sur sa chaine d'envoi et seulement une étape du rachet sur sa chaine de réception.



Suppose Alice then receives messages B3 and B4 with Bob's next ratchet key, then sends a message A5. Alice's final state will be as follows:

Enfin, Alice recoit les messages B3 et B4 avec la nouvelle clé publique de Bob issue de son DH rachet puis elle envoie le message A5. Le dernier état d'Alice sera :



Il existe bien sur d'autres cas possibles (comme par exemple si les messages ne sont pas reçus dans le bon ordre) mais ils ne seront pas traités ici.

Signal : sécurisé ?

Selon une étude menée en 2016 [2], le protocole de Signal n'apporte pas dans son implémentation de nouvelles vulnérabilités, seules celles propres aux blocs de base utilisés (KDF, DH) mais qui sont déjà considérées comme plutôt robustes.

La force Signal réside dans son *Double Rachet Algorithm* qui vient changer la clé publique de Bob et Alice à chaque nouveau message envoyé. De plus, chaque message envoyé est chiffré avec une clé jetable. Ainsi, même si un attaquant parvenait à récupérer la clé d'un des messages, il ne pourrait pas lire les précédents messages ni lire les futurs messages. Il en vient de même pour les clés publiques. Le protocole assure donc une confidentialité persistance.

Nous pouvons ainsi dire qu'à l'heure actuelle, Signal semble être un protocole robuste et on comprend mieux pourquoi il est recommandé par des experts en cybersécurité.

Aussi, il est important de noter qu'aucun message n'est enregistré sur des serveurs, de ce fait, les messages déchiffrés sont uniquement stockés sur les téléphones d'Alice et Bob. Mais faut-il encore que leur smartphone ne soit pas infecté par un malware comme Pegasus, mais ce n'est plus du ressort de la cryptographie…

Sources

[1] Documentation officielle :

- X3DH agreement protocol
- Double Rachet Algorithm

[2] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt and D. Stebila, "A Formal Security Analysis of the Signal Messaging Protocol," *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 451-466, doi: 10.1109/EuroSP.2017.27.

[3] Double Ratchet Messaging Encryption – Computerphile
https://www.youtube.com/watch?v=9sO2qdTci-s