Rapport de Stage
Présentation du protocole cryptographique de *Signal* et d'une amélioration (*MARSHAL*)

Ronan Thoraval

7 juillet 2022

Table des matières

1 Introduction

Nous utilisons tous des applications de messagerie instantanée (comme Messenger, WhatsApp ou encore *Signal*) pour communiquer avec notre famille ou nos amis. Mais savons-nous comment sont sécurisées ces applications ? Savons-nous à quel point nous sommes vulnérables en utilisant ces applications de communication ?

Nous nous penchons dans ce rapport sur le cas de l'application *Signal*, réputée comme une des applications les plus sûres (si ce n'est *la* plus sûre) du marché. Nous allons expliquer comment elle sécurise les messages de ses clients en les chiffrant, et en utilisant un protocole bien à elle !

Ce rapport est principalement basé sur le papier [1] d'Olivier Blazy, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Cristina Onete et Léo Robert, qui ont résumé ce protocole et qui en ont proposé une amélioration : *MARSHAL* (pour "Messaging with Asynchronous Ratchets and Signatures for faster HeALing").
Nous pouvons d'ores et déjà conclure que le protocole qu'utilise *Signal* est perfectible, mais il reste tout de même très efficace et sûr.

Nous allons d'abord présenter toutes les notions nécessaires pour pouvoir comprendre ces protocoles, et expliquer toutes les notations utilisées dans ce rapport. Nous vous conseillons de lire attentivement la partie sur les quelques rappels et nouveautés, et de survoler la partie sur les notations sans perdre trop de temps, mais d'y revenir lorsque vous lirez les parties plus concrètes et qu'il y aura des termes ou des notations dont vous ne vous rappelez plus.
Nous présentons ensuite le protocole *X3DH*, qui permet d'initer la communication en créant interactivement la clef nécessaire pour commencer la conversation.
Viennent alors les parties sur ce tant attendu protocole *Signal*, et sur son amélioration.

Nous vous souhaitons une bonne lecture !

2 Préambule

Nous décrivons dans cette section les différentes notions et notations utilisées dans ce rapport et dans le protocole de *Signal*.

Partout dans ce rapport, si vous cliquez sur un "mot-concept" dont vous ne vous rappelez plus la signifiaction ou certains détails, vous serez ramenés à son explication dans cette partie.

Nous ne rappelerons cependant pas toute la théorie des corps, supposée connue pour lire et comprendre ce rapport.

2.1 Quelques rappels et nouveautés

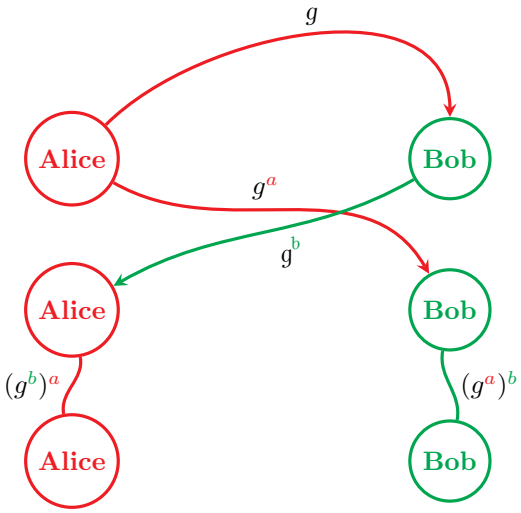
Notons que, dans ce papier, Alice veut parler avec Bob (nous adopterons ce code couleur dans tout le rapport).
C'est donc elle qui initie la communication.

2.1.1 Protocole *Diffie-Hellman*

Le protocole *Diffie-Hellman* permet de construire une clef commune entre deux participants (on peut l'adapter pour prendre en compte un nombre de participants supérieur).
Ce protocole évite de devoir se partager une clef en privé (devoir communiquer dans la vraie vie par exemple) puisqu'il permet de construire une clef commune aux deux participants aux yeux de tous sans en révéler la valeur

Supposons qu'Alice et Bob veulent construire une clef commune non-nulle (connue d'eux seuls) appartenant au corps fini $\mathbb{F}_q$.
Alice ou Bob, peu importe, va déterminer un générateur du groupe $\mathbb{F}_q^\times$, noté g et le partager à l'autre.
Ensuite, chacun élève g à une certaine puissance, disons a pour Alice et b pour Bob. Alice envoie g^a à Bob et Bob envoie g^b à Alice.
Une fois qu'ils ont reçu ces exponentiations, ils ne leur restent plus qu'à les élever à leur propre puissance pour tomber sur la même clef : $(g^a)^b = (g^b)^a$.

Le schéma suivant résume le protocole *Diffie-Hellman* décrit précédemment (en supposant que c'est Alice qui détermine le générateur) :



Nous noterons indifféremment dans ce rapport $DH(a, b)$ ou $DH(g^a, b)$ ou $DH(a, g^b)$ ou $DH(g^a, g^b)$ le résultat de ce protocole ($DH(a, b) = DH(g^a, b) = DH(a, g^b) = DH(g^a, g^b) = g^{ab}$), mais il est important de noter qu'il y a une clef privée et une clef publique à chaque fois, et que seule la clef publique est partagée dans les *Données Additionnelles* que l'on verra dans la partie sur l'algorithme de chiffrement. Remarquons qu'Alice peut calculer ce résultat si et seulement si elle connait b, et réciproquement, Bob peut calculer ce résultat si et seulement si il connait a puisqu'ils connaissent chacun leur exposant respectif et g.

2.1.2 *Key Derivation Function* (KDF)

Une "fonction de dérivation de clef" est une fonction prenant en argument une (ou plusieurs) clef(s) et ce qu'on appelle un (ou des) sel(s), et qui nous retourne une (ou plusieurs) autre(s) clef(s).

Il est possible de faire en sorte que les clefs sortantes aient certaines propriétés, comme une certaine longueur par exemple, mais le principe qui nous intéresse ici est que KDF doit être une fonction pseudo-aléatoire, c'est-à-dire qu'étant donné sa sortie, il est "impossible" de retrouver son entrée.

On peut la voir plus ou moins comme une fonction de hachage.

Dans le cas de *Signal*, nous utiliserons deux telles fonctions. Une qui prend en argument une clef et un sel et qui nous retourne deux clefs, et une qui prend en argument juste une clef et qui nous retourne une paire de clefs. Par soucis de simplification, nous ne différencierons pas ces deux fonctions, et nous laisserons au lecteur la responsabilité de comprendre laquelle est laquelle en vu des entrées et sorties de chacune.

Le paragraphe précédent nous indique donc que, connaissant la sortie de chacune, il est impossible de retrouver les clefs données en paramètres. C'est ce qui nous assurera le principe de *ratchet*, que nous définissons ci-après.

2.1.3 Principe de *ratchet*

Le *ratchet* est un principe cryptologique qui consiste à assurer que, dans un système de chiffrement où plusieurs clefs sont utilisées les unes à la suite des autres, la connaissance d'une clef à un moment donné ne permet pas de retrouver les clefs utilisées précédemment.

Cette condition nous implique que si un message est déchiffré parce que l'on connait la clef de déchiffrement, alors aucun message précédent ne sera déchiffré. C'est une condition indispensable au vu du protocole de *Signal* et *MARSHAL*, qui se servent de certaines clefs pour calculer la suivante.

Comme dit dans la partie précédente, ce principe de *ratchet* sera tenu par la fonction KDF. En effet, puisqu'elle est pseudo-aléatoire, étant donné une sortie (donc une clef à un moment donné), il est très dur de retrouver son entrée (donc les clefs précédentes).

Cette fonction est renforcée par le fait que nous lui "injectons" un sel, qui sera un *Diffie-Hellman* entre deux valeurs particulières. Cela rajoute de l'incertitude et rend l'attaque par exhaustivité très dure à mettre en place.

2.1.4 *Authenticated Encryption with Associated Data* (AEAD)

Le protocole de *Signal* et *MARSHAL* se servent d'une fonction de chiffrement nommé AEAD.

Cette fonction est une fonction symétrique dans le sens où la clef de chiffrement est la même que la clef de déchiffrement. Cette propriété va nous servir pour simplifier les choses dans les protocoles puisqu'Alice et Bob n'auront besoin que de calculer la même clef pour chiffrer/déchiffrer.

Nous mettrons cette clef en indice de la fonction, et le message à chiffrer en argument.

On peut également ajouter un certain nombre d'informations supplémentaires dans cette fonction AEAD, qui nous permettra de déchiffrer plus facilement, ou d'inclure des informations non-chiffrées dans le message chiffré qui serviront à obtenir certaines clefs, pour déchiffrer ledit message, ou chiffrer les messages suivants, ou encore permettre d'authentifier l'envoyeur pour être certain de l'identité de notre interlocuteur (nous verrons cela plus en détails dans la partie traitant des signatures). Nous noterons ces *Données Additionnelles* **AD** dans le reste du document, et nous considèrerons qu'il n'est pas nécéssaire de les différencier entre chaque message, bien que ces *Données Additionnelles* changent tout au long du protocole.

Nous mettrons ces *Données Additionnelles* en argument de la fonction, ce qui veut dire que le chiffré c du message m par la fonction AEAD avec la clef k et les *Données Additionnelles* correspondantes sera noté :

$$c = AEAD_k(m, AD)$$

Nous notons $AEAD^{-1}$ la fonction inverse de $AEAD$. $AEAD^{-1}$ est donc la fonction de déchiffrement.

2.2 Notations

Nous allons voir dans cette section les différentes notations utilisées dans ce rapport.

2.2.1 L'indexation des messages et des chiffrés

Le but du protocole *Signal* et du protocole *MARSHAL* est de sécuriser une communication entre Alice et Bob.

Nous noterons, dans ce rapport, $m_{y,x}$ le $x$ème message de la $y$ème communication. C'est-à-dire que y augmentera à chaque fois que l'interlocuteur changera, et x à chaque fois qu'un même interlocuteur envoie un message. Il est nécessaire de noter que x est remis à zéro à chaque fois que y change de valeur, *i.e.* l'interlocuteur change.

Citons quelques exemples pour mieux comprendre :

- Si l'un des deux envoie $m_{y,x}$ et envoie un autre message immédiatement après le premier, alors ce message sera $m_{y,x+1}$.
- Si le dernier message reçu par Bob (et donc envoyé par Alice) est $m_{y,x}$ et que Bob envoie un message immédiatement, alors ce message sera $m_{y+1,1}$.
- $m_{6,8}$ est le 8ème message envoyé pendant le 6ème échange. Puisqu'Alice commence la conversation, c'est elle qui enverra $m_{1,1}$, Bob enverra $m_{2,1}$, etc, ce qui nous donne que $m_{6,8}$ est le 8ème message envoyé par Bob pendant son 3ème "temps de parole", ce qui est aussi la 6ème fois que l'interlocuteur change (Alice a envoyé 3 fois des messages successifs et Bob aussi).

On remarque que si y est pair, alors c'est Bob qui envoie $m_{y,x}$, et si y est impair, alors c'est Alice qui envoie $m_{y,x}$.

Les chiffrés se comportent exactement de la même manière, et nous notons $c_{y,x}$ le chiffré de $m_{y,x}$.

2.2.2 Les différentes clefs utilisées dans le protocole *X3DH*

Nous définissons ici les différents types de clefs utilisées par le protocole *X3DH*, qui sont au nombre de 3.

Elles seront redéfinies et réexpliquées dans la partie sur le protocole *X3DH*.

2.2.2.a Les clefs d'identité

Les clefs appelées *clefs d'identité* sont des clefs liées au protocole *X3DH* servant à authentifier l'utilisateur. Ce sont des clefs considérées commes les *identifiants* en quelques sortes de l'utilisateur. Elles sont uniques et restent les mêmes durant tout le protocole, et même durant d'autres protocoles.

Elles sont désignées par IK_A pour Alice et par IK_B pour Bob.

2.2.2.b Les clefs éphémères

Les clefs appelées *clefs éphémères* sont des clefs liées au protocole *X3DH* temporaires (comme leur nom l'indique) qu'il faut mettre à jour régulièrement. Les *clefs éphémères* de Bob sont signées, ce qui permet de vérifier qu'elles sont bien à lui (nous parlerons des signatures un peu plus tard, dans la partie dédiée).

Elles sont uniques et désignées par EK_A pour Alice et par SPK_B pour Bob.

2.2.2.c Les clefs optionnelles

Les clefs appelées *clefs optionnelles* sont des clefs liées au protocole *X3DH* qui ne sont pas obligatoires (comme leur nom l'indique). Elles sont uniquement liées à Bob dans le sens où Alice n'en a pas.

Elles ne sont pas uniques, et donc Bob peut décider d'en mettre un nombre arbitraire sur son serveur pour qu'Alice puisse piocher dedans.

Ces clefs sont à usage unique et Bob les supprime dès qu'elles sont utilisées par Alice. De nouvelles peuvent être mises à la place si Bob le désire.

La $k$ème *clef optionnelle* (uniquement liée à Bob, nous le rappelons) est désignée par ${OPK_B}^k$.

Tout ce qui sera de la couleur violette *dans la partie X3DH* sera lié à ces clefs optionnelles, et sera donc optionnel.

2.2.3 Les différentes clefs utilisées dans les protocoles *Signal* et *MARSHAL*

Les deux protocoles que nous allons voir dans ce rapport utilisent des clefs pour chiffrer les messages, mais aussi des clefs pour d'autres utilisations. Nous allons les voir, les lister, les expliquer et mettre au clair leur notation dans cette partie.

2.2.3.a Les clefs de chiffrement et de déchiffrement

Les clefs de chiffrement et de déchiffrement sont les clefs utilisées pour chiffrer et déchiffrer par $AEAD$ les messages envoyés par Alice et Bob.

Comme dit précédemment, une même clef est utilisée pour chiffrer un message et pour déchiffrer le chiffré dudit message.

À chaque message envoyé, cette clef change, ce qui permet de toujours chiffrer un message avec une clef différente.

La clef pour chiffrer $m_{y,x}$ est notée $mk_{y,x}$, et donc, il s'en suit que la clef pour déchiffrer $c_{y,x}$ est également notée $mk_{y,x}$.

2.2.3.b La clef générée grâce à *X3DH*

Le protocole *X3DH* permet à Alice et Bob de se partager une clef, notée SK, qui elle-même permettra de générer les premières clefs du protocole *Signal* (et aussi du protocole *MARSHAL*).

Plus concrètement, cette clef SK permet de calculer la première clef de chiffrement et la première clef de chaîne (que l'on voit dans la partie suivante). Pour cela, nous la donnons en paramètre à la fonction KDF (en plus de la clef de ratchet "different-sender", que l'on voit dans une section suivante).

Nous voyons ci-après comment utiliser les clefs de chaîne.

2.2.3.c Les clefs de chaîne

Une clef de chaîne est une clef utilisée pour calculer la clef de chaîne et la clef de chiffrement suivante d'un même interlocuteur (elle n'est pas valable si la personne qui envoie les messages changent).

Nous verrons plus en détails dans la partie 4.2 comment ces deux dernières clefs sont calculées, mais nous pouvons d'ores et déjà dire que, dans le protocole *Signal*, notre clef de chaîne est donnée en paramètre à la fonction KDF, ce qui nous donne notre clef de chiffrement et notre clef de chaîne suivante, comme annoncé brièvement en 2.1.2.
Cette dernière nous permettra de calculer la clef de chiffrement et la clef de chaîne suivante, et de même pour cette clef de chaîne suivante, ce qui nous permettra d'avoir autant de paire de clefs qu'il y aura de messages à envoyer.
À chaque message envoyé, cette clef change, ce qui permet de toujours avoir une clef de chiffrement et une clef de chaîne différente à l'étape suivante.

La clef de chaîne permettant d'avoir la clef de chiffrement $mk_{y,x}$ est notée $CK_{y,x}$.
Observons donc que $CK_{y,x}$ nous permet d'obtenir $mk_{y,x}$ et $CK_{y,x+1}$ qui nous permet d'obtenir $mk_{y,x+1}$ et $CK_{y,x+2}$.

Nous voyons ci-après comment récupérer $CK_{y+1,1}$.

2.2.3.d Les clefs racines

Une clef racine est une clef utilisée par le protocole *Signal* pour calculer la première clef de chaîne d'Alice lorsqu'elle prend la parole.

Nous verrons plus en détails dans la partie 4.2 comment cette dernière clef est calculée, mais nous pouvons d'ores et déjà dire que notre clef racine est donnée en paramètre à la fonction KDF (en plus de la clef de ratchet "different-sender", que l'on voit dans la section suivante), ce qui nous donne notre première clef de chaîne comme annoncé brièvement en 2.1.2.
À chaque nouvelle communication d'Alice, cette clef change, ce qui permet de toujours avoir une clef racine différente de celle de l'étape suivante, et donc une clef de chaîne et une clef de chiffrement différentes à l'étape suivante.

La clef racine permettant d'avoir la clef de chaîne $CK_{y,1}$ est notée $rk_{\frac{y+1}{2}}$, puisque seule Alice crée les clefs racines (elles sont donc générées une fois sur deux).
Observons donc que $rk_{\frac{y+1}{2}}$ nous permet d'obtenir $CK_{y,1}$ qui nous permet d'avoir toutes les autres clefs nécéssaires à la communication.

Il est important de noter que Bob génère les clefs tmp de la même manière qu'Alice génère ses clefs racines mais le document [1] traite les deux cas comme s'ils étaient différents.
C'est pourquoi, pour éviter de faire une erreur, nous traiterons également ces deux cas différemment, même s'ils ne le paraissent pas.

2.2.3.e Les clefs de ratchet "different sender"

Une clef de ratchet "different sender" (abrégé en rds) est une clef utilisée pour calculer la première clef de chaîne d'un interlocuteur lorsqu'il prend la parole. C'est d'ailleurs lui qui la génère.

Nous verrons plus en détails dans la partie 4.2 comment cette dernière clef est calculée, mais nous pouvons d'ores et déjà dire que notre rds est utilisée dans un protocole DH avec la rds précédente (ou avec une des clefs que Bob a déposées sur le serveur si nous sommes à la genèse de la conversation). Le résultat de ce DH est donné en paramètre à la fonction KDF (en plus de la clef racine vue dans la section précédente), ce qui nous donne notre première clef de chaîne comme annoncé brièvement en 2.1.2.

À chaque nouvelle communication, cette clef change, ce qui permet de toujours avoir une rds différente de celle de l'étape suivante, et donc une clef de chaîne et une clef de chiffrement différentes à l'étape suivante.

La rds permettant d'avoir la clef de chaîne $CK_{y,1}$ est notée rds_y si c'est Alice qui l'a calculée, ou rds_y si c'est Bob qui l'a calculée.

Observons donc que rds_y nous permet d'obtenir $CK_{y,1}$ qui nous permet d'avoir toutes les autres clefs nécéssaires à la communication.

Dans ce cas-là, Alice *et* Bob génèrent des rds chacun leur tour quand ils initient un échange, cette clef n'a donc pas la même indexation que la clef racine.

2.2.3.f Les clefs de ratchet "same sender"

Une clef de ratchet "same sender" (abrégé en rss) est une clef utilisée dans le protocole *MARSHAL* pour calculer la clef de chaîne et la clef de chiffrement suivante d'un même interlocuteur (elle n'est pas valable si la personne qui envoie les messages change, et c'est la personne qui envoie les message qui génère ces clefs).

Nous verrons plus en détails dans la partie 5.2 comment ces deux dernières clefs sont calculées, mais nous pouvons d'ores et déjà dire que notre rss est utilisée dans un protocole DH avec une clef d'identité. Le résultat du DH est concaténé à une signature (que l'on voit dans la section suivante) et l'ensemble est donné en paramètre à la fonction KDF (en plus de la clef de chaîne vue dans une section précédente), ce qui nous donne notre clef de chiffrement et notre clef de chaîne suivantes.

Cette dernière nous permettra de calculer la clef de chiffrement et la clef de chaîne suivantes (en utilisant une nouvelle rss), et de même pour cette clef de chaîne suivante, *etc.*

À chaque message envoyé, cette clef change, ce qui permet de toujours avoir une clef de chiffrement et une clef de chaîne différentes à l'étape suivante.

La rss permettant d'avoir la clef de chiffrement $mk_{y,x}$ et la clef de chaîne $CK_{y,x+1}$ est notée $rss_{y,x}$ si c'est Alice qui l'a calculée, ou $rss_{y,x}$ si c'est Bob qui l'a calculée.

Observons donc que $CK_{y,x}$ a été utilisée avec la rss correspondante, pour obtenir $mk_{y,x}$ et $CK_{y,x+1}$ qui nous permet d'obtenir $mk_{y,x+1}$ et $CK_{y,x+2}$ en utilisant la rss suivante.

2.2.4 Les signatures

2.2.4.a Définition

Une signature, en cryptographie, est un moyen de certifier qu'un message est bien envoyé par une personne précise, qui va "signer" ce message.

L'un des interlocuteurs va donc envoyer un message, et la signature correspondante, pour attester que c'est bien lui qui a envoyé ledit message et non quelqu'un d'autre. Pour ce faire, on suppose que seul cet interlocuteur connait sa clef secrete. Il suffit donc que le message envoyé

atteste que l'envoyeur connaît cette clef (sans rien en divulguer bien sûr) pour assurer que c'est bien lui qui l'a envoyé. Il s'agira donc, pour la personne qui cherche à prouver son identité, de générer une telle signature, dépendante du message.

Les protocoles de signature sont différents suivant le système cryptographique utilisé. Il faut donc se mettre d'accord sur lequel utiliser.

Une fois cet accord fait, la personne qui veut "s'authentifier" génère sa signature suivant le protocole, et l'autre interlocuteur vérifie que la signature est valide, en suivant un protocole prédéfini également.

2.2.4.b Exemple

Nous allons rapidement présenter la *signature de Schnorr* (sans rentrer dans les détails) pour mieux comprendre ce qu'est concrètement une signature.

Alice a le couple clef privée, clef publique (a, g^a) dans un certain système cryptographique (qui utilise pourquoi pas le protocole DH).

Elle veut signer son message m. Elle va générer aléatoirement un $r = g^k$ et calculer $e = H(r\|m)$ et $s = k - a \times e$. (H est ici une fonction de hachage.)

Elle partagera à Bob le triplet (m, e, s), et Bob calculera $g^s \times (g^a)^e = g^{s+a\times e} = r'$ qui sera égal à $g^k = r$ si Alice est bien celle qu'elle prétend être et connait a. Bob calcule ensuite $e' = H(r'\|m)$ et vérifie que $e' = e$. Si c'est la cas, il y a de très fortes chances que la supposée Alice soit la vraie (on ne peut pas exclure le cas où un usurpateur tombe miraculeusement sur a en prenant une valeur aléatoirement).

Si ce ne sont pas les mêmes, alors on peut être sûrs que la supposée Alice n'est pas la vraie.

2.2.4.c L'indexation des signatures

Dans le protocole *MARSHAL*, des signatures sont utilisées. Nous ne nous penchons pas sur lesquelles mais notons qu'elles respectent la définition donnée plus haut. Nous expliquons dans cette partie ce à quoi ces signatures font référence, et comment elles ont été numérotées.

Les signatures utilisées dans ce rapport sont notées $\sigma_{x,y}$, et correspondent à la signature du message $(rds_{y-1}\|rss_{x,y})$. Ces signatures sont utilisées comme entrée pour des fonctions KDF et sont transmises en clair dans les *Données Additionnelles*.

Ces signatures, mélangées aux *Données Additionnelles*, permettent de vérifier que les clefs sont bien envoyées par Alice ou par Bob et non par un tiers.

3 Protocole *X3DH*

Le protocole *X3DH* est ici utilisé pour permettre le partage d'une clef SK qui permettra d'initier une communication sécurisée entre Alice et Bob, et la génération des différentes clefs nécéssaires durant le protocole de *Signal*.

X3DH est un acronyme pour *"Extended Triple Diffie-Hellman"*. Ce protocole se base donc, comme son nom l'indique, sur le protocole *Diffie-Hellman*, noté DH, qu'il utilise trois fois.

Expliquons concrètement en quoi consiste ce protocole :

— Bob transmet trois types de clefs sur un serveur de confiance :

1. IK_B : C'est la clef d'identité de Bob. Il la conservera tout au long de l'échange.
2. SPK_B : C'est la clef éphémère de Bob. Il la changera de "temps en temps", et elle est signée.
3. ${OPK_B}^k$: Ce sont les clefs optionnelles de Bob. Il les transmet une fois, supprime celles utilisées, et en transmet d'autres s'il le souhaite.

— Alice récupère ces clefs et vérifie que SPK_B est bien envoyée par Bob grâce à la signature. Si ce n'est pas le cas, elle arrête tout. (Authentification)

— Alice génère un couple de clefs :

1. IK_A : C'est la clef d'identité d'Alice. Elle la conservera tout au long de l'échange (et même pour d'autres interlocuteurs que Bob).
2. EK_A : C'est la clef éphémère d'Alice.

— Alice calcule ①= $DH(IK_A,SPK_B)$, ②= $DH(EK_A,IK_B)$, ③= $DH(EK_A,SPK_B)$ (et ④= $DH(EK_A,{OPK_B}^k)$ si elle a pu récupérer cette dernière clef).

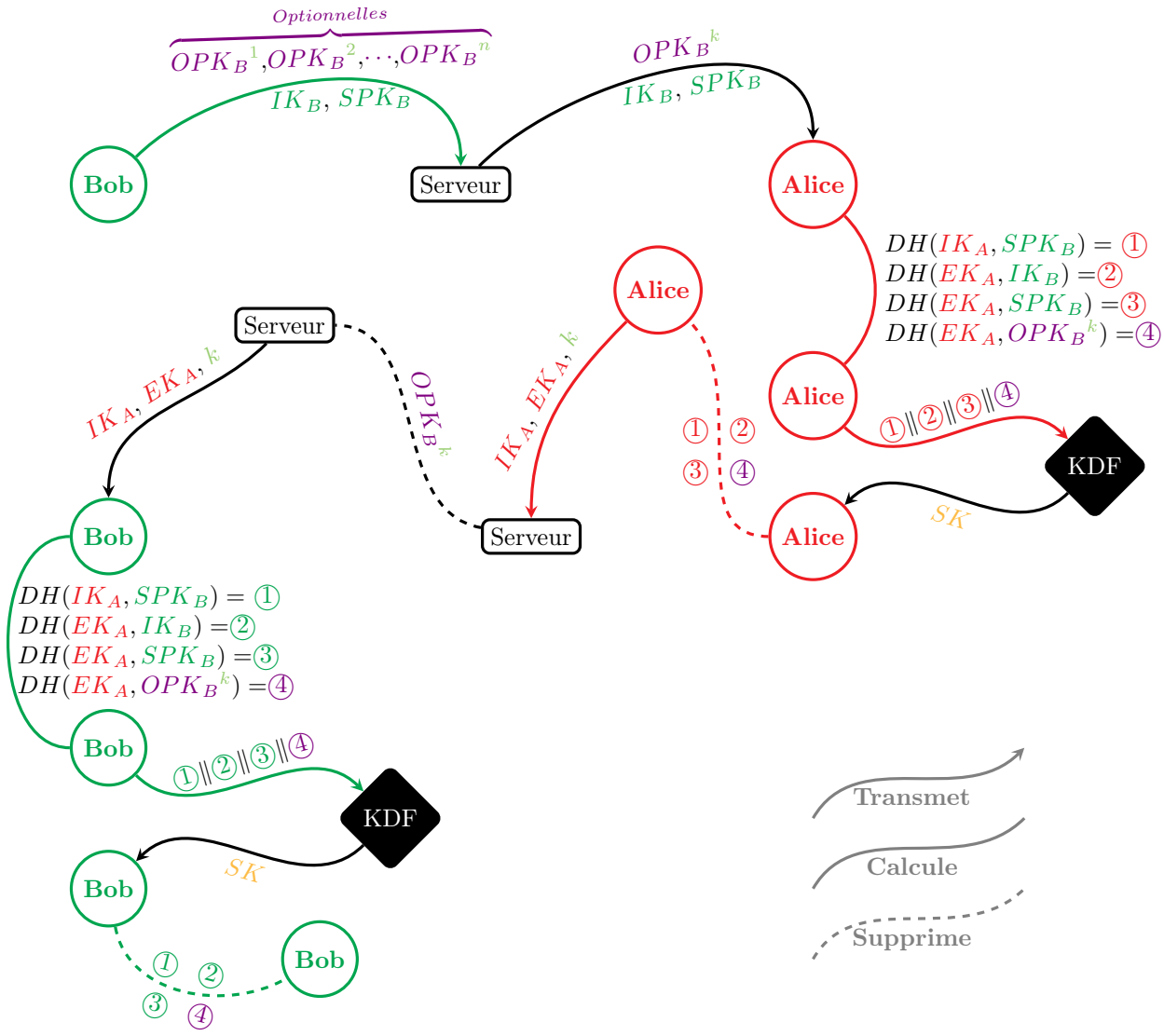
— Alice calcule $SK = KDF(①\|②\|③)$, ou $SK = KDF(①\|②\|③\|④)$.

— Alice supprime les résultats des utilisations du protocole *Diffie-Hellman* et transmet ses clefs (et l'identitifant k de ${OPK_B}^k$ si elle a pu récupérer cette dernière clef) au serveur de confiance.

— Le serveur supprime la clef ${OPK_B}^k$.

— Bob récupère les clefs d'Alice, et fait de même.

Le schéma suivant résume le protocole *X3DH* décrit précédemment :


Optionnelles
$OPK_B{}^1, OPK_B{}^2, \cdots, OPK_B{}^n$
IK_B, SPK_B
Bob
Serveur
$OPK_B{}^k$
IK_B, SPK_B
Alice
$DH(IK_A, SPK_B) = ①$
$DH(EK_A, IK_B) = ②$
$DH(EK_A, SPK_B) = ③$
$DH(EK_A, OPK_B{}^k) = ④$
Alice
IK_A, EK_A, k
Serveur
$OPK_B{}^k$
Alice
① ②
③ ④
Alice
①‖②‖③‖④
KDF
SK
Serveur
IK_A, EK_A, k
Bob
$DH(IK_A, SPK_B) = ①$
$DH(EK_A, IK_B) = ②$
$DH(EK_A, SPK_B) = ③$
$DH(EK_A, OPK_B{}^k) = ④$
Bob
①‖②‖③‖④
KDF
SK
Bob
Bob
① ②
③ ④
Transmet
Calcule
Supprime

4 Protocole *Signal*

Maintenant que l'échange de clef initiale est fait, intéressons-nous au principe général de *Signal*. Ce protocole se base sur le principe de "ratchet", qui consiste à, étant donnée une clef à une certaine étape, empêcher de retrouver les clefs précédentes. Ce rôle sera joué par notre fonction KDF, qui en remplit toutes les conditions.

Il est aussi important de noter que, dans ce protocole, "qui envoie les messages quand" joue un rôle important. C'est-à-dire que le cas où Alice envoie plusieurs messages successifs sera traité différemment du cas où Alice envoie un message juste après en avoir reçu un de Bob par exemple.

4.1 Préliminaires

Le principe du protocole est donc qu'Alice va initialement générer une paire de clefs rk_1, $CK_{1,1}$ que Bob pourra lui aussi générer. La première clef servira à calculer les clefs nécessaires pour déchiffrer les chiffrés envoyés par Bob, et la deuxième servira à calculer les clefs nécessaires pour chiffrer les clairs qu'Alice veut envoyer.

Alice donc, calcule rk_1,$CK_{1,1}$ à partir du DH de rds_1 (qu'elle crée) et SPK_B, et de SK trouvé grâce à $X3DH$. Ensuite, elle calcule $mk_{1,1}$,$CK_{1,2}$ à partir de $CK_{1,1}$ et se sert de $mk_{1,1}$ pour chiffrer son message.
Si elle n'a pas reçu de message de Bob et qu'elle veut envoyer un autre message, elle calcule $mk_{1,2}$,$CK_{1,3}$ à partir de $CK_{1,2}$ et se sert de $mk_{1,2}$ pour chiffrer son message, *etc.*

Passons au point de vue de Bob :
Bob reçoit le premier message d'Alice. Il calcule rk_1,$CK_{1,1}$ de la même manière qu'Alice, pour récupérer, lui aussi, $mk_{1,1}$,$CK_{1,2}$. Il a donc la clef qui a chiffré le message reçu, il peut donc le déchiffrer.
Si Bob reçoit d'autres messages consécutifs d' Alice, il procède de même, en calculant $mk_{1,2}$,$CK_{1,3}$ *etc.*

Supposons maintenant que Bob veuille répondre à Alice. Il calule tmp,$CK_{2,1}$ à partir du DH de rds_2 (qu'il crée) et rds_1, et de rk_1 trouvé au premier calcul de chacun. Ensuite, il calcule $mk_{2,1}$,$CK_{2,2}$ à partir de $CK_{2,1}$ et se sert de $mk_{2,1}$ pour chiffrer son message.
S'il n'a pas reçu de message d'Alice et qu'il veut envoyer un autre message, il calcule $mk_{2,2}$,$CK_{2,3}$ à partir de $CK_{2,2}$ et se sert de $mk_{2,2}$ pour chiffrer son message, *etc.*

Alice, pour déchiffrer, suit le même protocole que Bob, mais en remplaçant rk_1 par tmp.

On voit donc que ce protocole traite différemment les messages envoyés successivement par un même correspondant des messages envoyés successivement par nos deux correspondants distincts. Pour être plus précis, on considère différemment deux messages envoyés successivement par une même personne, et un message envoyé par une personne suivi d'un message envoyé par l'autre personne.

Expliquons concrètement en quoi consiste ce protocole en différenciant plusieurs cas de figure :

- Alice envoie le premier message.
- Alice envoie un message sans avoir reçu de message de Bob après son dernier message envoyé.
- Bob envoie son premier message.
- Bob envoie un message sans avoir reçu de message de Alice après son dernier message envoyé.
- Alice envoie un message en ayant reçu un (ou plusieurs) message(s) de Bob après son dernier message envoyé.
- Bob envoie un message en ayant reçu un (ou plusieurs) message(s) de Alice après son dernier message envoyé.

4.2 Protocole concret

Notons, avant de commencer, qu'il n'y a pas de partie sur le déchiffrement. En effet, chaque interlocuteur déchiffre exactement comme l'autre a chiffré, en remplaçant les clairs par les chiffrés, et $AEAD$ par $AEAD^{-1}$.

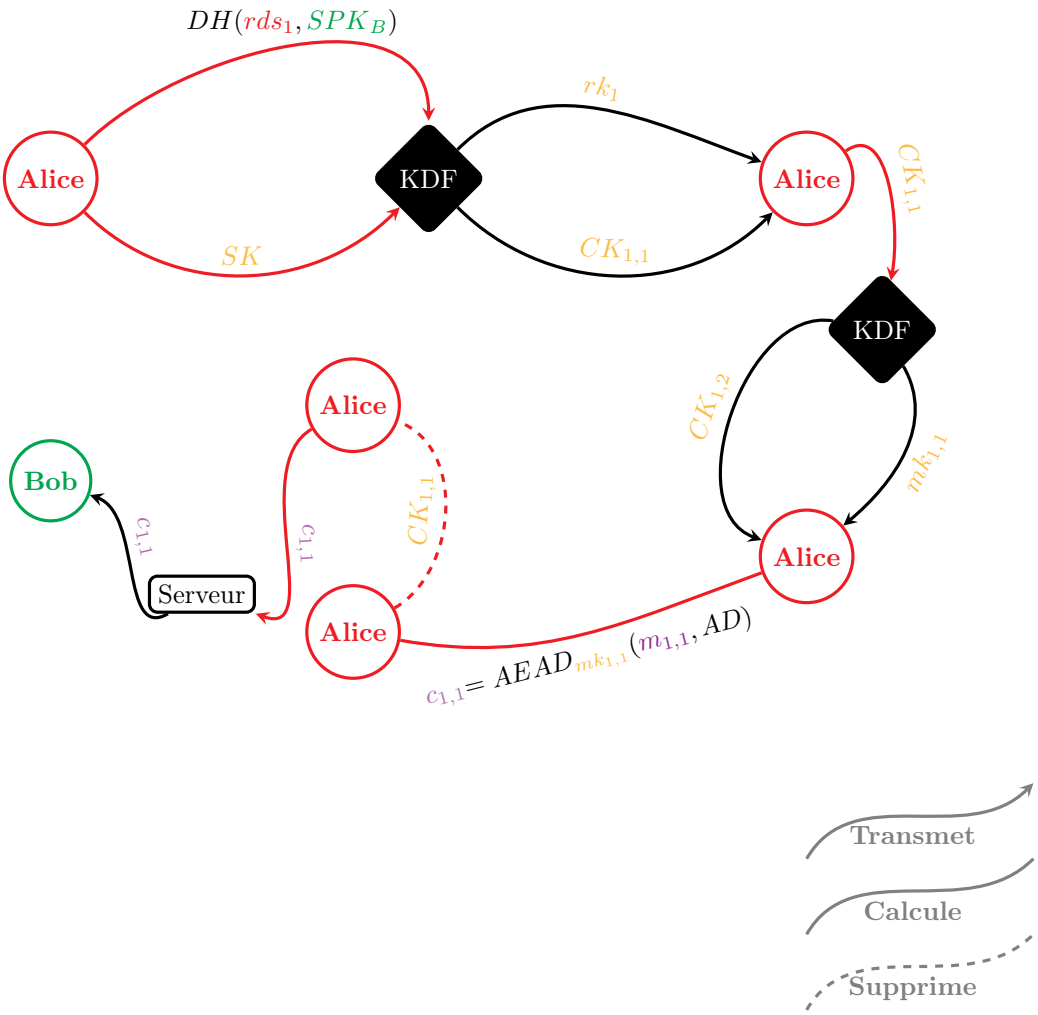
4.2.1 Alice envoie le premier message

Alice souhaite envoyer son premier message $m_{1,1}$ à Bob. Nous sommes donc à la genèse de leur conversation, rien n'a été envoyé pour l'instant.

Alice va donc suivre le protocole suivant :

— Alice calcule rk_1,$CK_{1,1}$= $KDF(DH(rds_1, SPK_B)$,$SK)$.
— Alice calcule $mk_{1,1}$,$CK_{1,2}$= $KDF(CK_{1,1})$.
— Alice calcule $c_{1,1} = AEAD_{mk_{1,1}}(m_{1,1}, AD)$.
— Alice supprime $CK_{1,1}$.
— Alice envoie $c_{1,1}$ au serveur qui l'enverra à Bob.

Le schéma suivant résume le protocole décrit précédemment :



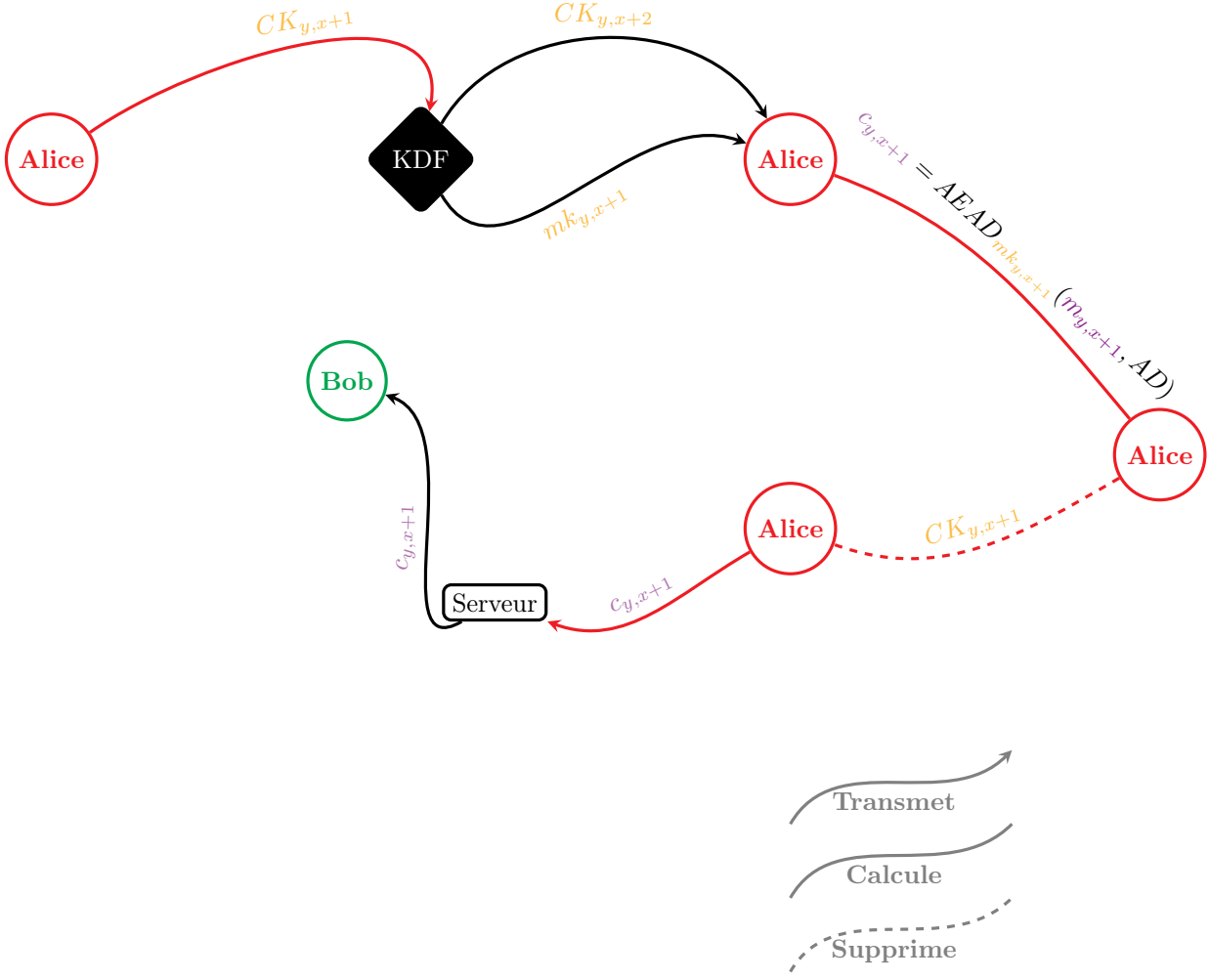
4.2.2 Alice envoie un message sans avoir reçu de message de Bob après son dernier message envoyé

Alice souhaite envoyer un $x+1^{eme}$ message d'affilé $m_{y,x+1}$ à Bob qui ne lui a donc rien envoyé juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Alice va donc suivre le protocole suivant :

— Alice calcule $mk_{y,x+1}$,$CK_{y,x+2}$= $KDF(CK_{y,x+1})$.
— Alice calcule $c_{y,x+1} = AEAD_{mk_{y,x+1}}(m_{y,x+1}, AD)$.
— Alice supprime $CK_{y,x+1}$.
— Alice envoie $c_{y,x+1}$ au serveur qui l'enverra à Bob.

Le schéma suivant résume le protocole décrit précédemment :



4.2.3 Bob envoie son premier message

Bob souhaite envoyer son premier message $m_{2,1}$ à Alice. C'est donc la première fois que Bob envoie un message à Alice durant cette conversation.

Bob va donc suivre le protocole suivant :

— Bob calcule tmp,$CK_{2,1}$= $KDF(DH(rds_2, rds_1), rk_1)$.

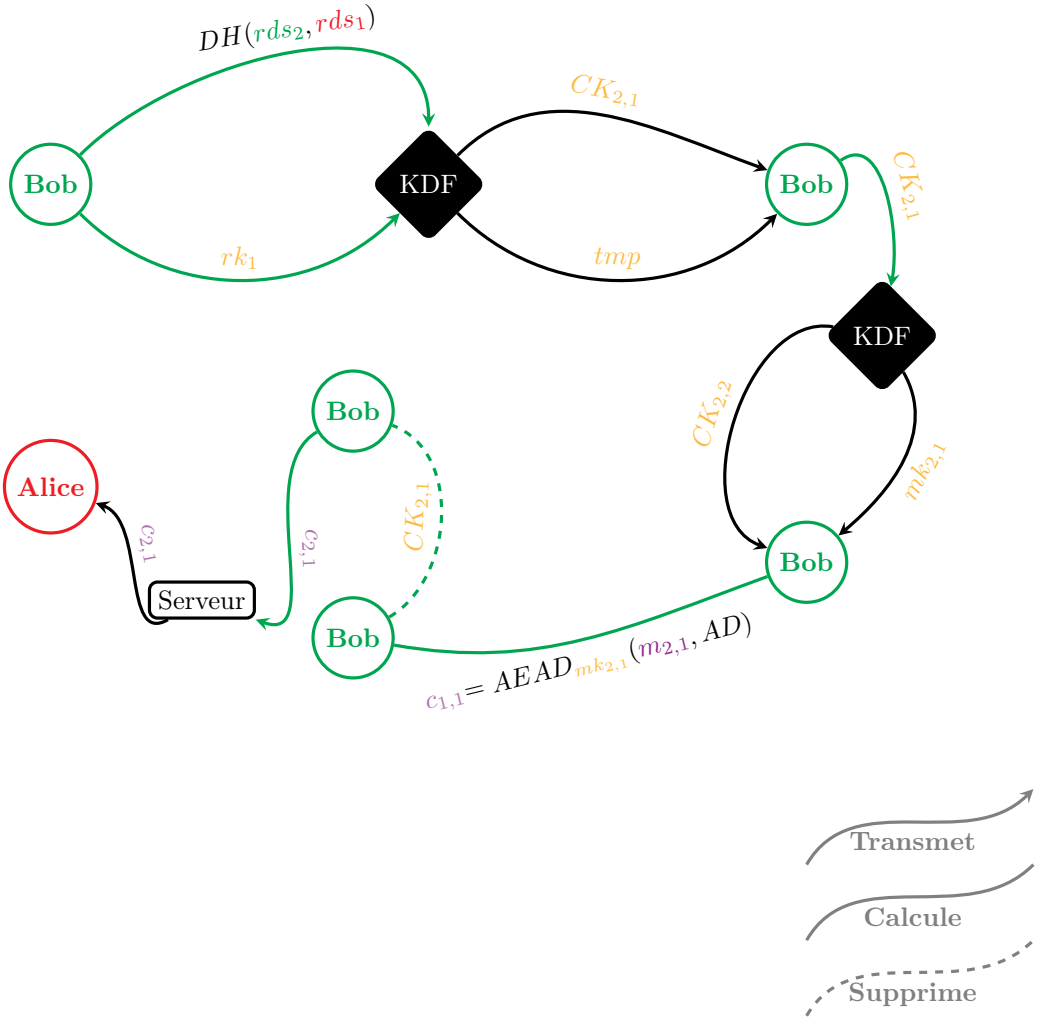
— Bob calcule $mk_{2,1}$,$CK_{2,2}$= $KDF(CK_{2,1})$.

— Bob calcule $c_{2,1} = AEAD_{mk_{2,1}}(m_{2,1}, AD)$.

— Bob supprime $CK_{2,1}$.

— Bob envoie $c_{2,1}$ au serveur qui l'enverra à Alice.

Le schéma suivant résume le protocole décrit précédemment :



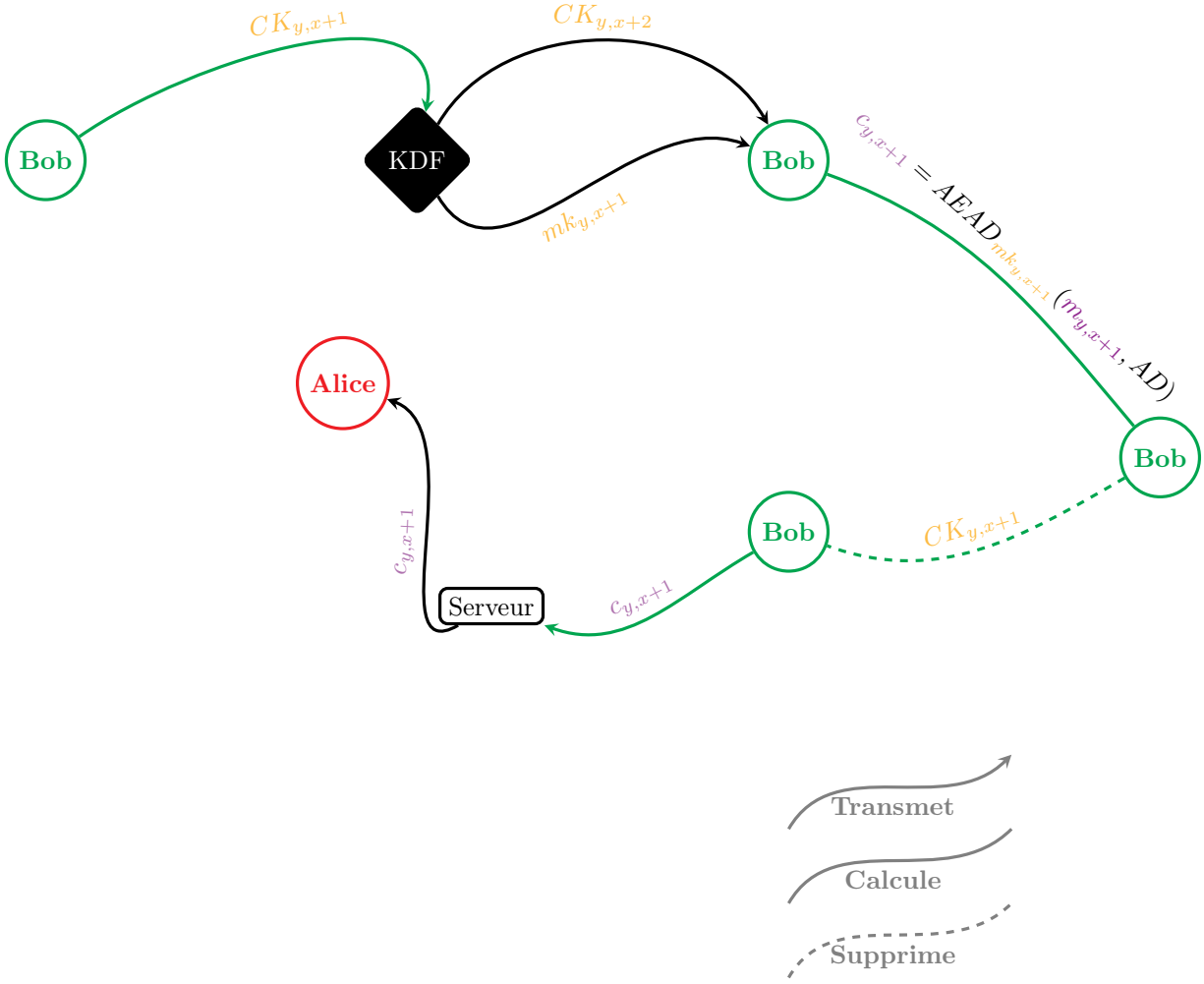
4.2.4 Bob envoie un message sans avoir reçu de message d'Alice après son dernier message envoyé

Bob souhaite envoyer un $x+1^{eme}$ message d'affilé $m_{y,x+1}$ à Alice qui ne lui a donc rien envoyé juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Bob va donc suivre le protocole suivant :

— Bob calcule $mk_{y,x+1}$,$CK_{y,x+2}$= $KDF(CK_{y,x+1})$.
— Bob calcule $c_{y,x+1} = AEAD_{mk_{y,x+1}}(m_{y,x+1}, AD)$.
— Bob supprime $CK_{y,x+1}$.
— Bob envoie $c_{y,x+1}$ au serveur qui l'enverra à Alice.

Le schéma suivant résume le protocole décrit précédemment :



4.2.5 Alice envoie un message en ayant reçu un (ou plusieurs) message(s) de Bob après son dernier message envoyé

Alice souhaite envoyer un message $m_{y,1}$ à Bob qui lui a envoyé un (ou plusieurs) message(s) juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Alice va donc suivre le protocole suivant :

— Alice calcule $rk_{\frac{y+1}{2}}, CK_{y,1} = KDF(DH(rds_y, rds_{y-1}), tmp)$.

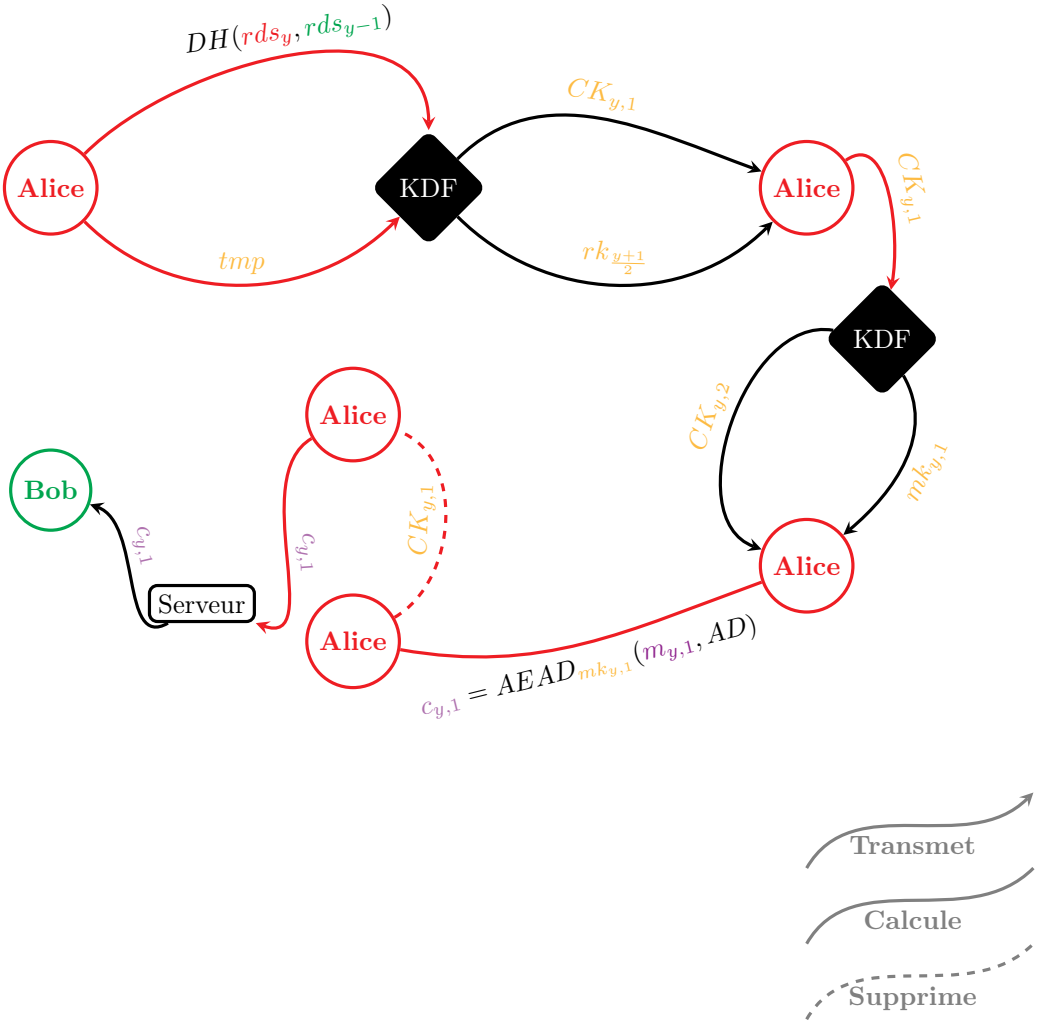
— Alice calcule $mk_{y,1}, CK_{y,2} = KDF(CK_{y,1})$.

— Alice calcule $c_{y,1} = AEAD_{mk_{y,1}}(m_{y,1}, AD)$.

— Alice supprime $CK_{y,1}$.

— Alice envoie $c_{y,1}$ au serveur qui l'enverra à Bob.

Le schéma suivant résume le protocole décrit précédemment :



4.2.6 Bob envoie un message en ayant reçu un (ou plusieurs) message(s) d'Alice après son dernier message envoyé

Bob souhaite envoyer un message $m_{y,1}$ à Alice qui lui a envoyé un (ou plusieurs) message(s) juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Alice va donc suivre le protocole suivant :

— Bob calcule $rk_{\frac{y+1}{2}}, CK_{y,1} = KDF(DH(rds_y, rds_{y-1}), tmp)$.

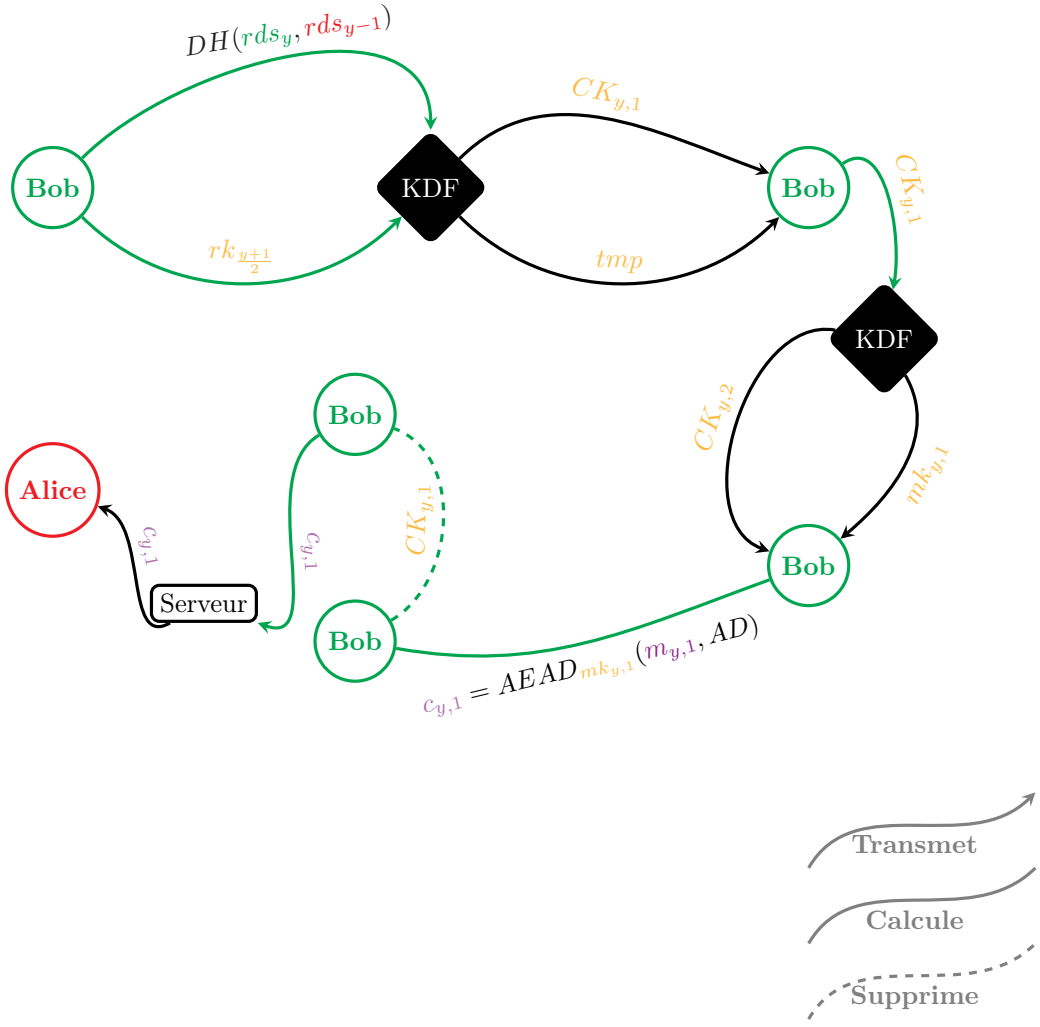
— Bob calcule $mk_{y,1}, CK_{y,2} = KDF(CK_{y,1})$.

— Bob calcule $c_{y,1} = AEAD_{mk_{y,1}}(m_{y,1}, AD)$.

— Bob supprime $CK_{y,1}$.

— Bob envoie $c_{y,1}$ au serveur qui l'enverra à Alice.

Le schéma suivant résume le protocole décrit précédemment :



5 Protocole *MARSHAL*

Le protocole *MARSHAL* est une amélioration du protocole *Signal*, dans le sens où, si un attaquant arrive à se glisser entre les deux interlocuteurs, il faudra moins de temps pour *MARSHAL* que pour *Signal* pour se "soigner", c'est-à-dire faire en sorte que l'attaquant ne puisse plus se faire passer pour l'un ou pour l'autre. Sans rentrer dans les détails, nous diront ici que ce "soin" est dû aux utilisations des "ratchets" qui jouent, plus ou moins, le rôle de remise à zéro. Il sera donc facile de voir que *MARSHAL* se soigne plus vite que *Signal*, puisqu'il utilise des "ratchets" beaucoup plus souvent que ce dernier.

Voyons donc rapidement comment *MARSHAL* fonctionne.

5.1 Préliminaires

Le protocole *MARSHAL* fonctionne dans l'idée de la même manière que le protocole *Signal*, au détail près que les clefs de chaîne ne sont plus calculées seulement à partir de la clef de chaîne précédente. En effet, en plus de cette clef de chaîne précédente, il nous faudra ajouter un nouveau DH basé sur les clefs de ratchet "same sender". C'est-à-dire que pour chaque message (successif) envoyé, il y aura une étape de "ratchet" en plus, ce qui rallongera certes le temps d'exécution du protocole, mais qui assurera une sécurité plus forte, puisque la clef publique de ratchet "same user" sera partagée à chaque message, empêchant un attaquant de rester trop longtemps entre Alice et Bob. (Une signature intervient aussi à ce niveau-là, mais elle ne change rien au principe, elle est plutôt là pour *certifier* de qui vient le message.)

De plus, nous n'avons plus notre clef racine rk pour changer d'utilisateur, mais deux DH basés sur deux clefs de ratchet "different sender" (*Signal* n'avait qu'un seul DH).

Nous pouvons donc maintenant voir concrètement à quoi ressemble ce protocole, non pas si éloigné de celui de *Signal*, en différenciant toujours les mêmes cas de figure :

- Alice envoie le premier message.
- Alice envoie un message sans avoir reçu de message de Bob après son dernier message envoyé.
- Bob envoie son premier message.
- Bob envoie un message sans avoir reçu de message de Alice après son dernier message envoyé.
- Alice envoie un message en ayant reçu un (ou plusieurs) message(s) de Bob après son dernier message envoyé.
- Bob envoie un message en ayant reçu un (ou plusieurs) message(s) de Alice après son dernier message envoyé.

5.2 Protocole concret

Notons que, comme dans la partie sur *Signal*, il n'y a pas de partie déchiffrement. En effet, le déchiffrement se fait, encore, de la même manière que précdemment. C'est-à-dire qu'il suffit de retracer le protocole de chiffrement de l'autre en ramplaçant le clair pa le chiffré qu'on cherche à déchiffrer, et la fonction $AEAD$ par la fonction $AEAD^{-1}$.

5.2.1 Alice envoie le premier message

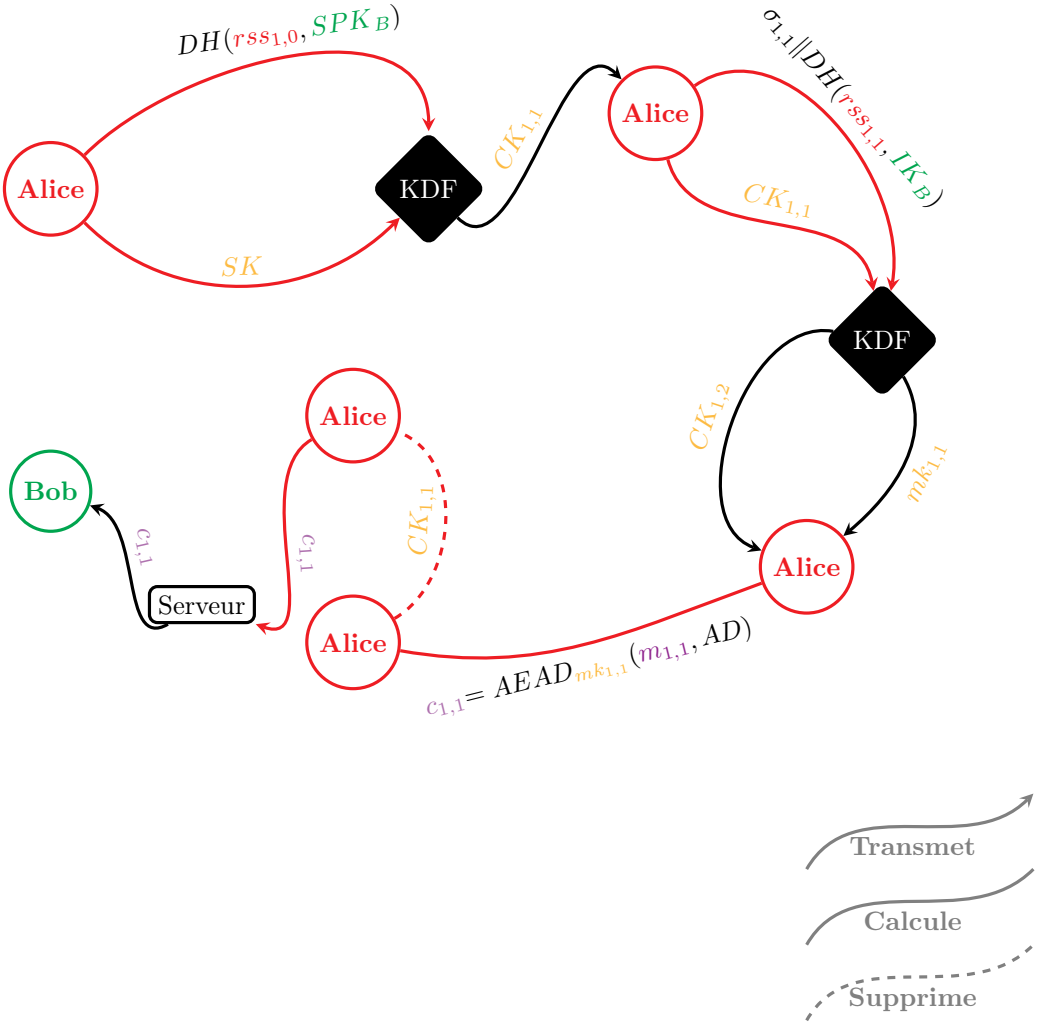
Alice souhaite envoyer son premier message $m_{1,1}$ à Bob. Nous sommes donc à la genèse de leur conversation, rien n'a été envoyé pour l'instant.

Alice va donc suivre le protocole suivant :

- Alice calcule $CK_{1,1} = KDF(DH(rss_{1,0}, SPK_B), SK)$.
- Alice calcule $mk_{1,1}, CK_{1,2} = KDF(CK_{1,1}, \sigma_{1,1} \| DH(rss_{1,1}, IK_B))$.

— Alice calcule $c_{1,1} = AEAD_{mk_{1,1}}(m_{1,1}, AD)$.
— Alice supprime $CK_{1,1}$.
— Alice envoie $c_{1,1}$ au serveur qui l'enverra à Bob.

Le schéma suivant résume le protocole décrit précédemment :



5.2.2 Alice envoie un message sans avoir reçu de message de Bob après son dernier message envoyé

Alice souhaite envoyer un $x+1^{eme}$ message d'affilé $m_{y,x+1}$ à Bob qui ne lui a donc rien envoyé juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Alice va donc suivre le protocole suivant :

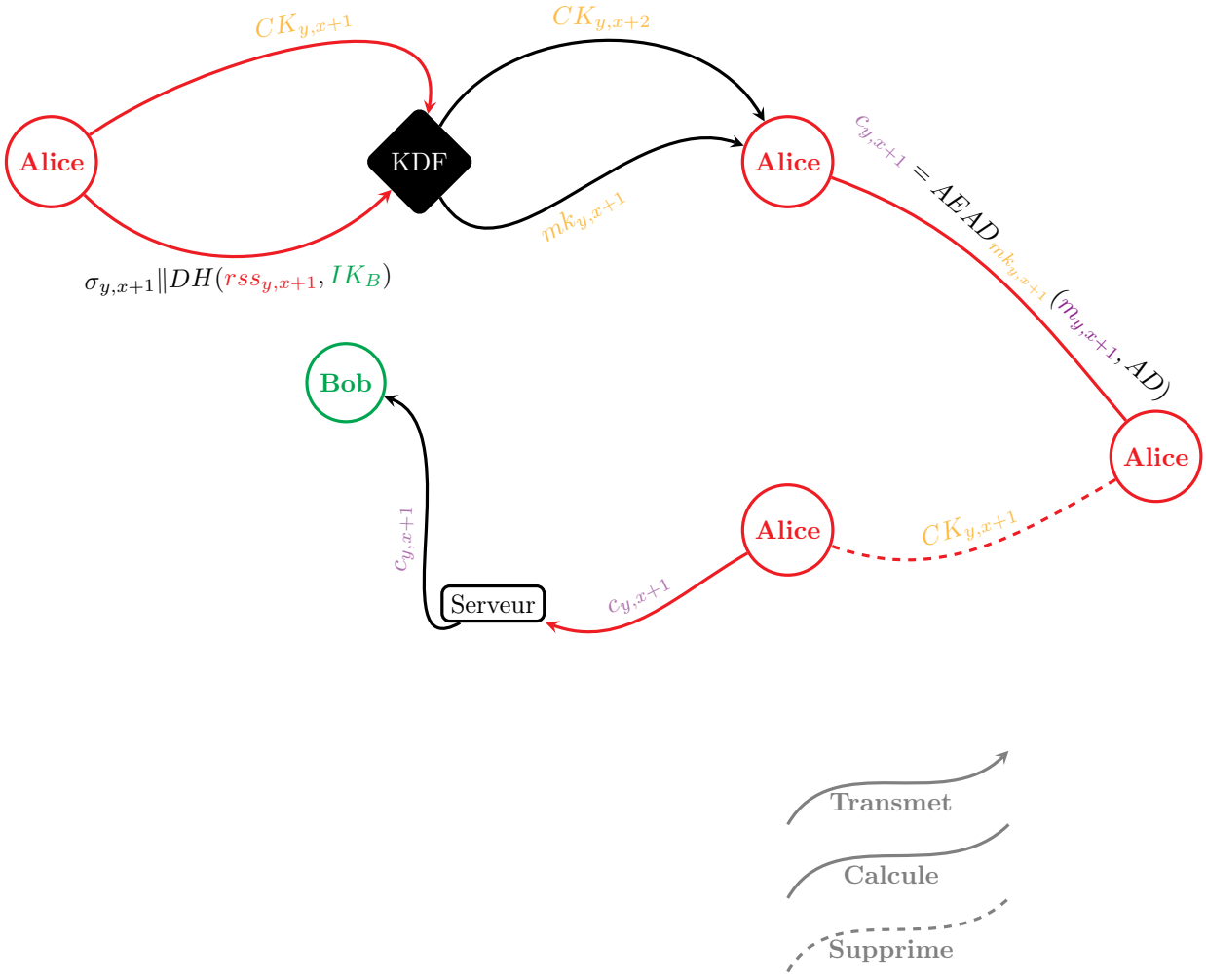
— Alice calcule $mk_{y,x+1}$,$CK_{y,x+2}$= $KDF(CK_{y,x+1}, \sigma_{y,x+1} \| DH(rss_{y,x+1}, IK_B))$.

— Alice calcule $c_{y,x+1} = AEAD_{mk_{y,x+1}}(m_{y,x+1}, AD)$.

— Alice supprime $CK_{y,x+1}$.

— Alice envoie $c_{y,x+1}$ au serveur qui l'enverra à Bob.

Le schéma suivant résume le protocole décrit précédemment :



5.2.3 Bob envoie son premier message

Bob souhaite envoyer son premier message $m_{2,1}$ à Alice. C'est donc la première fois que Bob envoie un message à Alice durant cette conversation.

Bob va donc suivre le protocole suivant :

— Bob calcule $CK_{2,1} = KDF(DH(rds_1, IK_B), DH(rds_0, IK_A))$.

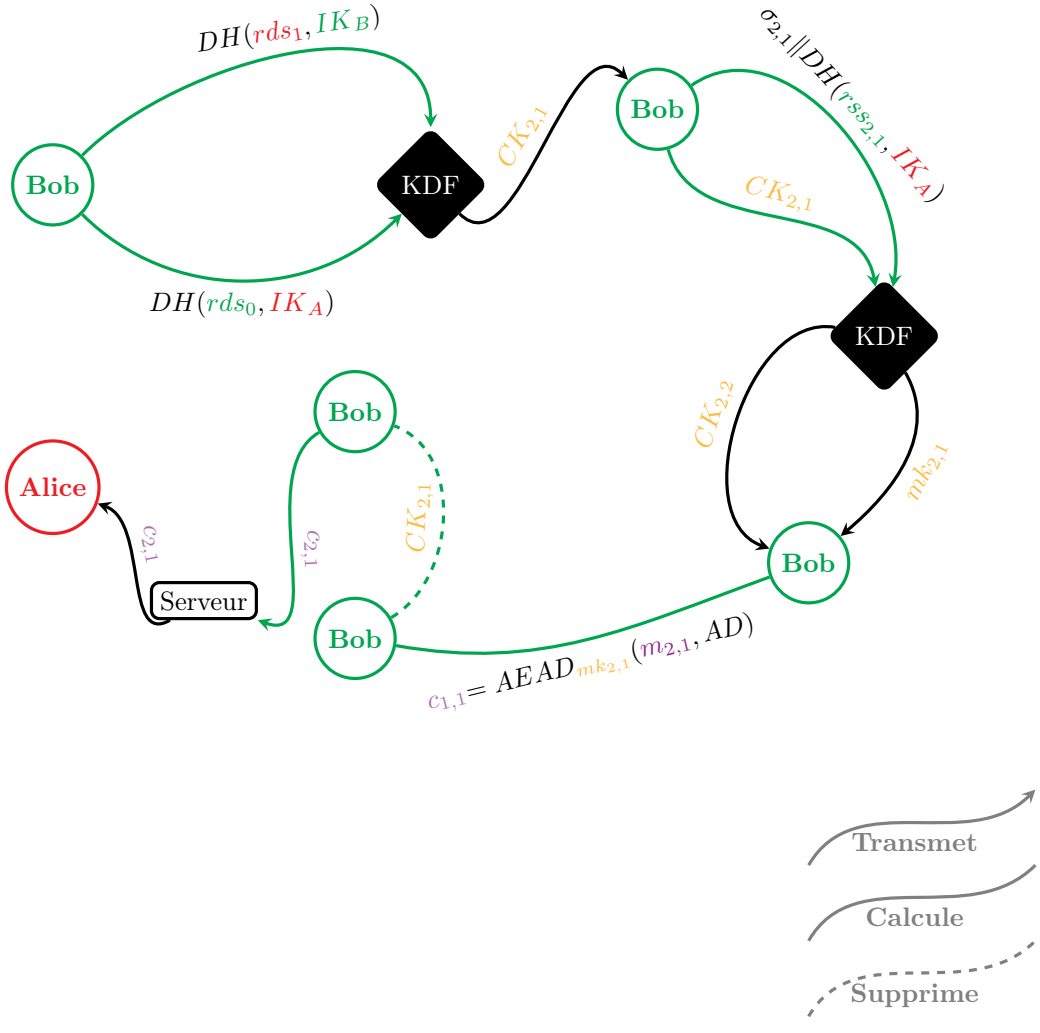
— Bob calcule $mk_{2,1}, CK_{2,2} = KDF(CK_{2,1}, \sigma_{2,1} \| DH(rss_{2,1}, IK_A))$.

— Bob calcule $c_{2,1} = AEAD_{mk_{2,1}}(m_{2,1}, AD)$.

— Bob supprime $CK_{2,1}$.

— Bob envoie $c_{2,1}$ au serveur qui l'enverra à Alice.

Le schéma suivant résume le protocole décrit précédemment :



5.2.4 Bob envoie un message sans avoir reçu de message d'Alice après son dernier message envoyé

Bob souhaite envoyer un $x+1^{eme}$ message d'affilé $m_{y,x+1}$ à Alice qui ne lui a donc rien envoyé juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Bob va donc suivre le protocole suivant :

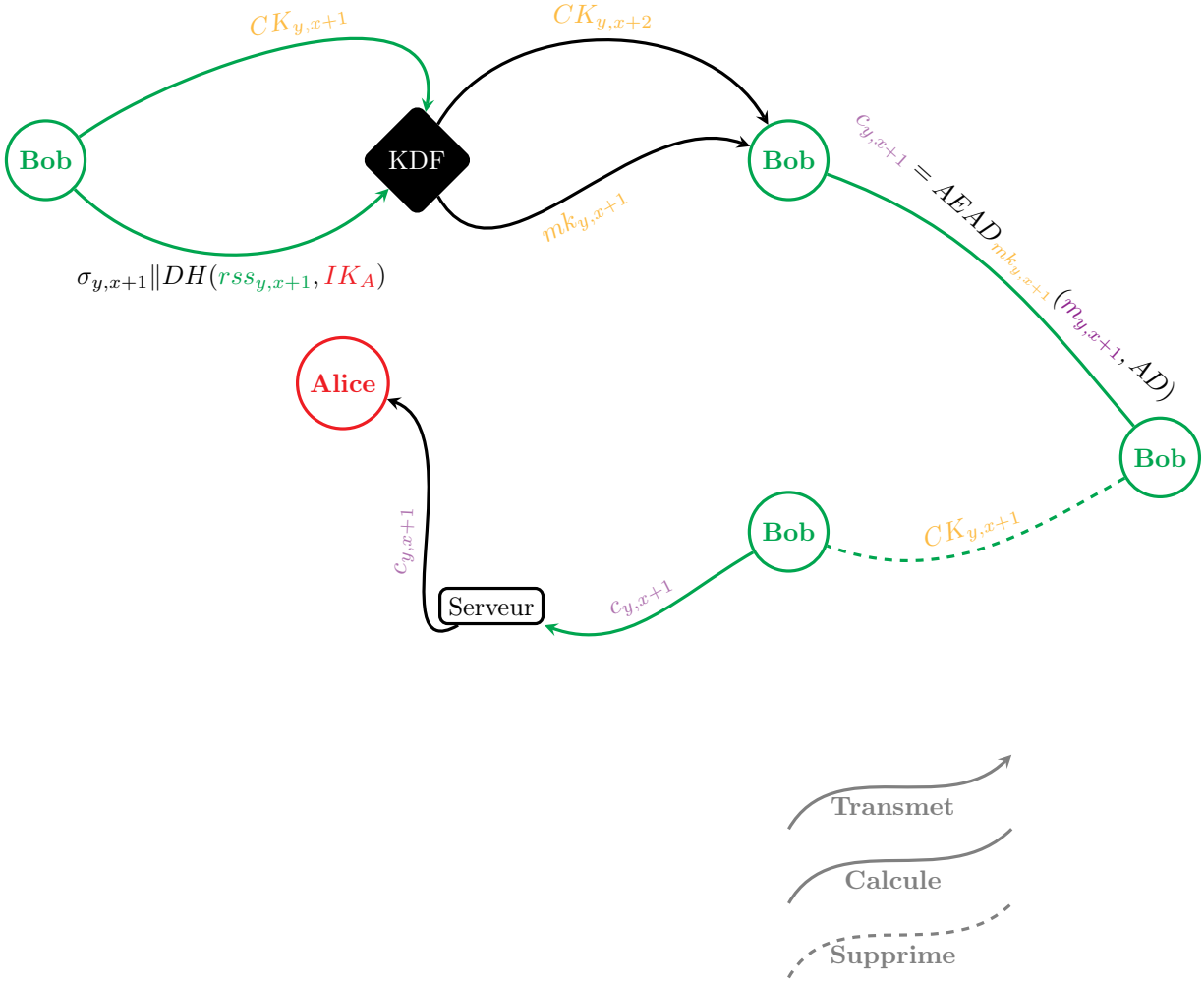
— Bob calcule $mk_{y,x+1}$,$CK_{y,x+2}$= $KDF(CK_{y,x+1}, \sigma_{y,x+1} \| DH(rss_{y,x+1}, IK_A))$.

— Bob calcule $c_{y,x+1} = AEAD_{mk_{y,x+1}}(m_{y,x+1}, AD)$.

— Bob supprime $CK_{y,x+1}$.

— Bob envoie $c_{y,x+1}$ au serveur qui l'enverra à Alice.

Le schéma suivant résume le protocole décrit précédemment :



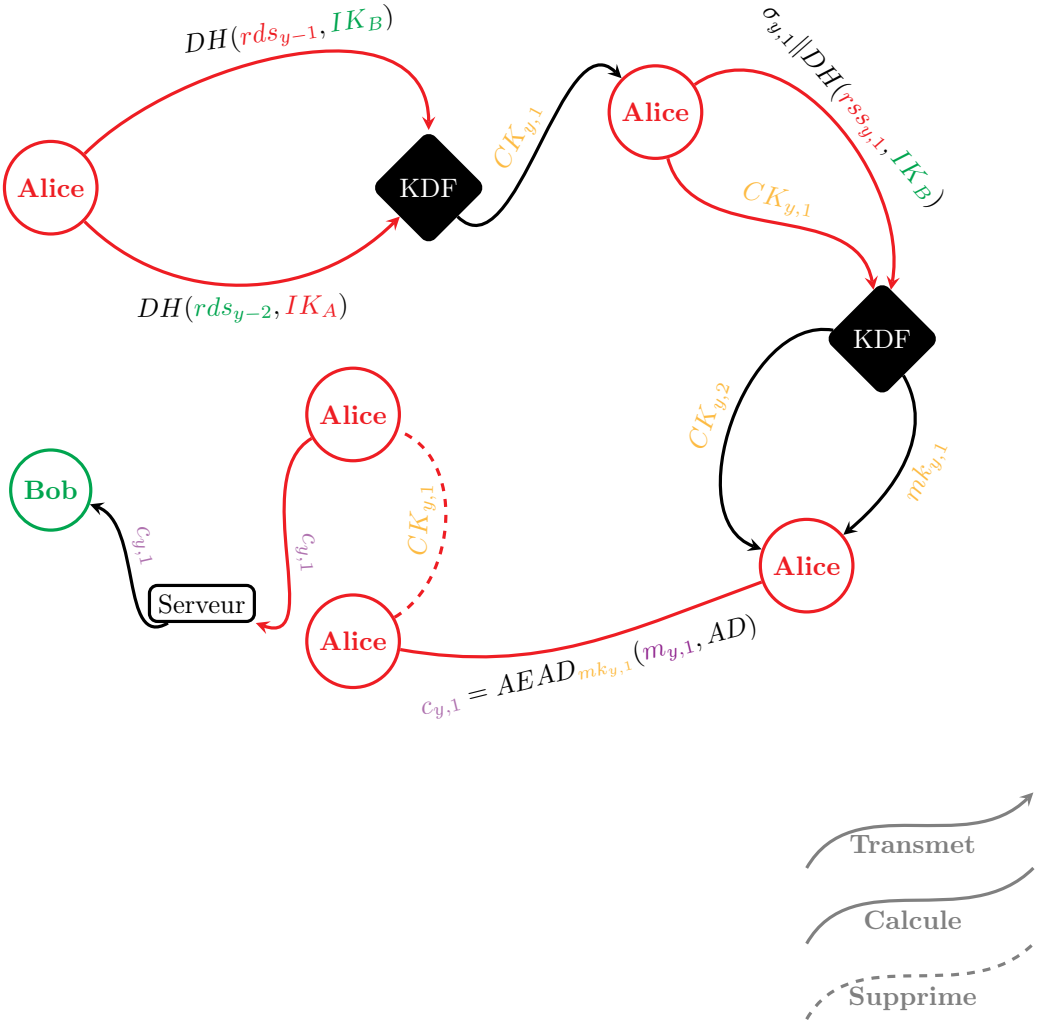
5.2.5 Alice envoie un message en ayant reçu un (ou plusieurs) message(s) de Bob après son dernier message envoyé

Alice souhaite envoyer un message $m_{y,1}$ à Bob qui lui a envoyé un (ou plusieurs) message(s) juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Alice va donc suivre le protocole suivant :

— Alice calcule $CK_{y,1} = KDF(DH(rds_{y-1}, IK_B), DH(rds_{y-2}, IK_A))$.
— Alice calcule $mk_{y,1}, CK_{y,2} = KDF(CK_{y,1}, \sigma_{y,1} \| DH(rss_{y,1}, IK_B))$.
— Alice calcule $c_{y,1} = AEAD_{mk_{y,1}}(m_{y,1}, AD)$.
— Alice supprime $CK_{y,1}$.
— Alice envoie $c_{y,1}$ au serveur qui l'enverra à Bob.

Le schéma suivant résume le protocole décrit précédemment :



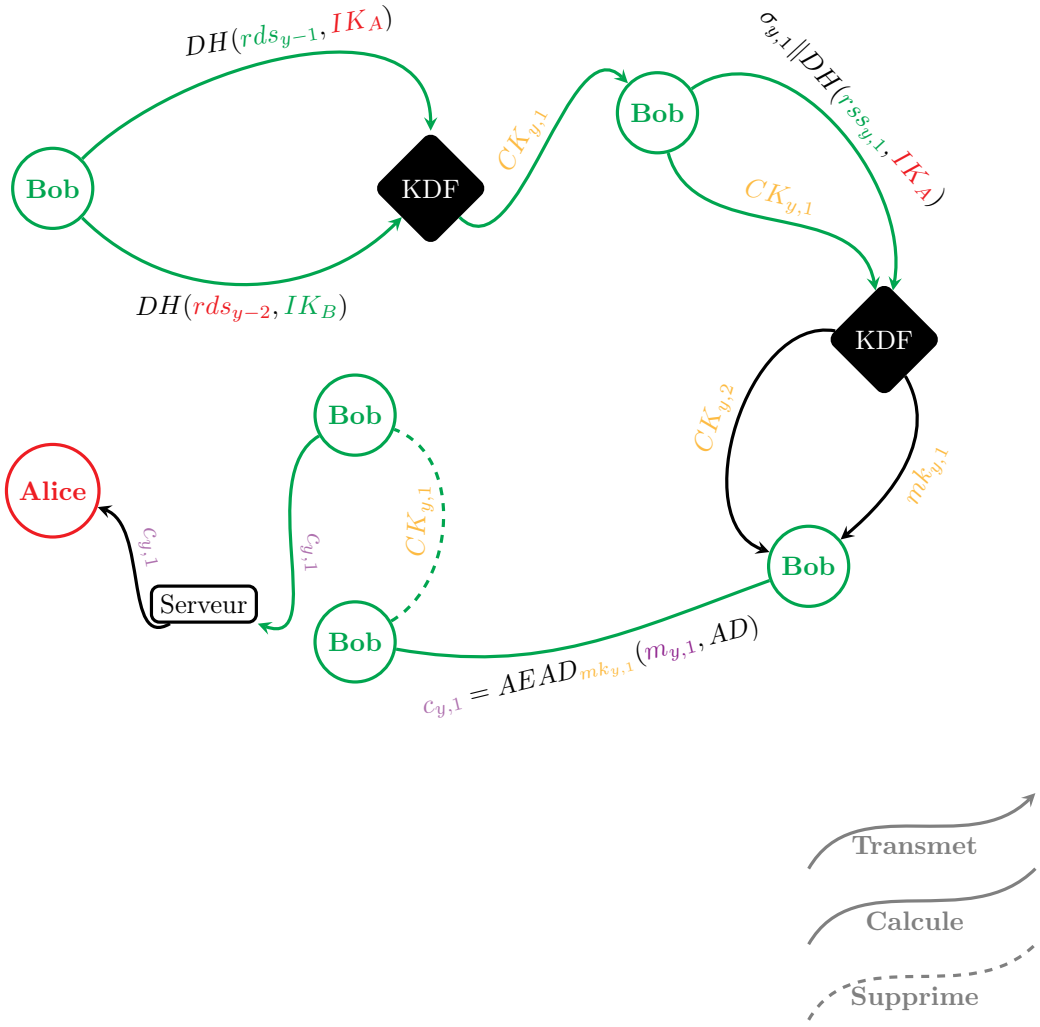
5.2.6 Bob envoie un message en ayant reçu un (ou plusieurs) message(s) d'Alice après son dernier message envoyé

Bob souhaite envoyer un message $m_{y,1}$ à Alice qui lui a envoyé un (ou plusieurs) message(s) juste avant. La conversation a donc déjà commencé, et plusieurs échanges ont pu avoir lieu.

Alice va donc suivre le protocole suivant :

— Bob calcule $CK_{y,1} = KDF(DH(rds_{y-1}, IK_A), DH(rds_{y-2}, IK_B))$.
— Bob calcule $mk_{y,1}, CK_{y,2} = KDF(CK_{y,1}, \sigma_{y,1} \| DH(rss_{y,1}, IK_A))$.
— Bob calcule $c_{y,1} = AEAD_{mk_{y,1}}(m_{y,1}, AD)$.
— Bob supprime $CK_{y,1}$.
— Bob envoie $c_{y,1}$ au serveur qui l'enverra à Alice.

Le schéma suivant résume le protocole décrit précédemment :



Références

[1] Olivier Blazy, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Cristina Onete, and Léo Robert. Marshal : Messaging with asynchronous ratchets and signatures for faster healing. 2022.

[2] Guilhem Castagnos. https://www.math.u-bordeaux.fr/ gcastagn/.

[3] Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm. 2016. https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[4] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. signal. 2016. https://www.whispersystems.org/docs/specifications/x3dh/.