Sécurisation des communications

Introduction

Qu'est ce qu'une connexion sécurisée dans ce cours ?

Nous nous intéresserons dans ce cours à la façon dont on sécurise les communications sur un réseau ouvert. Pour simplifier le propos nous considérerons une relation client-serveur dans laquelle Alice est le client et Bob est le serveur. L'attaquant potentiel sera dénommé Oscar (plutôt imaginer Oscar comme une agence de renseignement que comme un hacker en sweat à capuche).

Nous dirons que la *connexion établie est sécurisée* dès lors que Alice et Bob peuvent échanger des messages de sorte que :

- Alice est certaine qu'elle communique avec Bob (on dit que *le serveur Bob est authentifié par le client*),
- personne ne peut lire (*confidentialité*) ou altérer (*intégrité*) les messages échangés entre Alice et Bob (*les messages doivent être chiffrés*).

On ne s'intéressera pas à l'authentification éventuelle d'Alice auprès de Bob (classiquement pour l'accès d'Alice à son compte utilisateur). Tout d'abord parce que dans de nombreux cas, le client ne s'identifie pas auprès du serveur (par exemple lorsque vous effectuez une recherche sur Wikipédia ou que vous consultez le site de votre journal préféré). Et ensuite parce qu'une fois qu'Alice et Bob ont établi une connexion sécurisée, Bob peut demander à Alice son identifiant et son mot de passe sans aucun risque (puisque la connexion est sécurisée).

Un peu de vocabulaire

On dit qu'un message est *chiffré* lorsque la suite initiale de symboles composant le message est rendue incompréhensible grâce à l'utilisation d'une *clef de chiffrement.*
On dit qu'on *déchiffre* un message chiffré lorsqu'on retrouve le message initial *en utilisant la clef de déchiffrement*.
On dit qu'on *décrypte* un message chiffré lorsqu'on retrouve le message initial *sans utiliser la clef de déchiffrement* (en piratant).

Attention : les termes *coder* et *décoder* sont réservés aux cas où on représente de l'information grâce à des symboles prédéfinis. Par exemple on peut représenter les caractères de l'alphabet grâce à des symboles 0 ou 1 en utilisant le code ASCII. Donc dans les termes coder et décoder, il n'y a pas la notion de secret ou de confidentialité.

Quel est le contexte à avoir en tête lorsqu'on parle de sécurisation des communications ?

- *On suppose que les messages transitent par l'attaquant Oscar. Dit autrement, lorsqu'elle envoie un message à Bob, Alice doit se dire qu'il a été potentiellement intercepté par Oscar. Et lorsqu'elle reçoit un message de Bob, Alice doit se dire qu'il a potentiellement été écrit par Oscar.*
- Des milliards de connexions sécurisées sont établies chaque jour et les utilisateurs changent régulièrement d'appareil : cela nécessite d'avoir des procédés de sécurisation non centralisés.
- Les données à chiffrer peuvent être volumineuses (par exemple les vidéos) : le chiffrement doit être efficace.
- On suppose que l'attaquant connait les algorithmes utilisés et peut recueillir de gros volumes de données chiffrées (pour éventuellement faire des attaques basées sur des observations statistiques).

Quel est le plan du cours ?

Nous verrons d'abord comment on peut chiffrer efficacement des données grâce à une méthode *(masque jetable et dérivées)* de *cryptographie symétrique* (dite aussi *cryptographie à clef partagée*) qui garantit la confidentialité et l'intégrité des messages échangés. Néanmoins cette méthode nécessite que Alice et Bob disposent tous les deux d'une *même clef partagée* : comment peuvent-ils en toute sécurité se mettre d'accord sur cette *clef partagée* alors qu'Oscar écoute tout ce qui circule sur le réseau ?

Nous verrons pour cela une méthode *(Diffie-Hellman)* de *cryptographie asymétrique* (dite aussi *cryptographie à clef publique*) qui permet à Alice et Bob de se mettre d'accord sur une même clef partagée sans qu'Oscar ne puisse la deviner. Néanmoins il reste un problème de taille : comment Alice peut-elle être certaine qu'elle est bien en train d'échanger avec Bob ? Et si depuis le début Oscar – qui contrôle le réseau – se faisait passer pour Bob ?

Nous verrons alors une dernière méthode *(RSA)* de *cryptographie asymétrique* qui permet à Alice d'être certaine que c'est bien avec Bob qu'elle communique depuis le début. Au final la connexion se fera en trois phases dans l'ordre suivant :

1) Méthode asymétrique RSA : Bob est authentifié auprès d'Alice.
2) Méthode asymétrique Diffie-Hellman : Alice et Bob se mettent d'accord sur une clef partagée.
3) Méthode symétrique du masque jetable (et dérivées) : Alice et Bob communiquent de façon sécurisée.

Nous allons donc voir ces trois méthodes cryptographiques puis nous ferons une synthèse qui décrira la sécurisation d'une communication https.

A : Méthodes de chiffrement symétrique (à clef partagée)

Dans tout ce qui suit Alice et Bob connaissent tous les deux la clef partagée et peuvent donc chiffrer / déchiffrer leurs messages.

A.1) Chiffrement de César et variantes

Le célèbre et historique chiffrement de César consiste à décaler les lettres du message de N lettres vers la droite de l'alphabet.
Pour N = 7 avec comme message `"Specialite NSI"` on obtient le message chiffré suivant : `"Zwljphspal UZP"`
Pour déchiffrer le message, il suffit alors de décaler les lettres du message chiffré de N lettres vers la gauche de l'alphabet.

On peut – par exemple en Python – créer une fonction qui assure le chiffrement et une autre fonction qui assure le déchiffrement des messages. Les fonctions de chiffrement et de déchiffrement ont besoin de deux paramètres : le message à chiffrer et la valeur du décalage : N. *La clef partagée de cette méthode est le décalage N.* Les deux interlocuteurs ont uniquement besoin de cette information pour pouvoir chiffrer et déchiffrer les messages qu'ils s'envoient.

On remarque que c'est la même clef qui est utilisée pour chiffrer et déchiffrer d'où le nom de chiffrement symétrique.

Remarque 1 : Cette méthode n'est pas très sécurisée car il n'y a que 26 clefs différentes possibles. On peut donc facilement *décrypter* un message *chiffré* en effectuant une *attaque par force brute* (on teste chacune des 26 clefs, les unes après les autres).

Remarque 2 : Si on implémente cette méthode sur machine on peut obtenir une variante en étendant le chiffrement de César aux 189 caractères présents dans l'encodage latin-1 (c'est-à-dire les caractères allant de U+0000 à U+00FF de Unicode) qui ne sont pas des caractères de contrôle.
Dans ce cas, si on appelle C et D les fonctions de chiffrement et déchiffrement correspondantes on peut obtenir les exemples ci-contre qui acceptent les accents.
Cette fois ci il y a 189 clefs différentes possibles (ce qui reste insuffisant).

```
>>> C('Spécialité NSI', 7)
"Zwðjphsp{ð'UZP"

>>> D("Zwðjphsp{ð'UZP", 7)
'Spécialité NSI'
```

Remarque 3 : Dans la remarque 2, on voit qu'on a été embêté par l'encodage utilisé (à cause des caractères de contrôle) alors que cela ne concerne pas le chiffrement. Pour s'éviter ce genre de détails techniques, dans la suite de ce cours les messages à chiffrer et à déchiffrer seront considérés comme déjà encodés. Autrement dit, on cherchera à chiffrer des messages déjà encodés en hexadécimal et non pas des messages donnés sous forme de caractères.

A.2) Masque jetable et méthodes dérivées

La méthode du masque jetable consiste à utiliser une clef partagée aussi grande que le message que l'on souhaite chiffrer et à utiliser l'opérateur booléen XOR (évoqué dans le cours de première) pour chiffrer le message. Voyons cela sur un exemple.

Chiffrement

Prenons comme message à chiffrer en héxadécimal : 4e 53 49 (il s'agit de "NSI" encodé en utf-8).
Prenons comme clef partagée en héxadécimal : 54 4f 50 (il s'agit de "TOP" encodé en utf-8)
Pour obtenir le message chiffré on va utiliser le XOR bit à bit. Pour cela on écrit le message et la clef partagée en binaire puis on applique le XOR :

```
message en binaire               : 01001110 01010011 01001001
clef partagée en binaire         : 01010100 01001111 01010000
-------------------------------------------------------------
XOR bit à bit = message chiffré  : 00011010 00011100 00011001
```

Le message chiffré en héxadécimal est donc : 1a 1c 19.
Remarque : le message chiffré ne peut pas se représenter sous forme d'une chaîne de caractères car les points de code U+001A, U+001C et U+0019 correspondent à des caractères de contrôle.

Déchiffrement

Le déchiffrement va être très facile car l'opérateur XOR est réversible. Il suffit de faire à nouveau le XOR avec le message chiffré et la clef partagée pour obtenir le message initial :

```
message chiffré en binaire         : 00011010 00011100 00011001
clef partagée en binaire           : 01010100 01001111 01010000
-------------------------------------------------------------
XOR bit à bit = message déchiffré  : 01001110 01010011 01001001
```

On retrouve bien le message de départ : 4e 53 49.

Points forts et points faibles du masque jetable

Le premier point fort du masque jetable est le nombre de clefs possibles : avec une clef de 3 octets comme ci-dessus, soit 24 bits, on dispose déjà de 2^{24} clefs différentes soit environ 16 millions de clefs différentes. Ainsi en prenant des clefs partagées de 128 ou 256 bits, les attaques par force brute deviennnent inenvisageables.
Un autre point fort est que l'opérateur XOR est un opérateur qui est implémenté en dur dans les processeurs : il y a des circuits dédiés au calcul du XOR. Le chiffrement et le déchiffrement peuvent donc être réalisés de façon très rapide ce qui est très important lorsqu'on chiffre des données volumineuses (vidéo par exemple).

En revanche le point faible du masque jetable est la taille de la clef qui doit être égale à la taille du message à chiffrer. Si on souhaite chiffrer des données volumineuses c'est très gênant. Par ailleurs un serveur ne pourrait pas utiliser plusieurs fois la même clef au risque de laisser la porte ouverte à des attaques : la clef doit donc être jetée à chaque chiffrement (d'où le nom de masque jetable).

Améliorations du masque jetable

Il existe des méthodes permettant d'étendre la clef. La méthode la plus simple est de répéter la clef autant de fois que nécessaire pour qu'elle atteigne la taille du message. Avec l'exemple ci-dessus on pourrait par exemple utiliser la clef TOPTOPTOPTOPTOPT si on souhaitait chiffrer seize octets de données.

Mais cette méthode d'extension est trop naïve et laisserait également la porte ouverte à des attaques statistiques. Heureusement, il existe des procédés beaucoup plus sophistiqués (DES, AES, RC4, E0 …) permettant d'étendre une clef partagée de taille modeste afin de pouvoir chiffrer un gros volume de données en toute sécurité et de façon efficace (ce qui est impératif lorsqu'on chiffre de gros volumes de données en direct sur un réseau).

Par exemple, à titre de culture générale, avec une seule clef partagée de taille k=256 bits et le chiffrement par blocs AES on peut étendre la clef pour chiffrer en toute sécurité environ $2^{n/2}=2^{64}$ blocs de taille n=128 bits soit 250 milliards de Go sans faire apparaître de motifs ouvrant la porte à une attaque statistique. Qui plus est sur les processeurs Intel Core ou AMD Ryzen supportant le jeu d'instructions AES-NI, ce chiffrement peut être effectué à la vitesse de 10 Go / s (ce qui est très rapide).
(Sources : https://en.wikipedia.org/wiki/Advanced_Encryption_Standard , https://www.di.ens.fr/~fouque/mpri/des-aes.pdf)

Remarque : Une même taille de clef n'apporte pas le même niveau de sécurité s'il s'agit d'une clef partagée utilisée dans un chiffrement symétrique (comme ici où une taille 256 bits est actuellement considérée comme suffisante) ou dans une méthode cryptographique asymétrique (comme nous verrons plus loin où il peut falloir une taille de clef de 2048 bits).

A.3) Garanties apportées par le chiffrement symétrique (à clef partagée)

Si le chiffrement symétrique utilise une clef partagée de taille suffisament importante, on peut considérer qu'une attaque par force brute n'est pas envisageable. Dans ce cas l'attaquant Oscar semble impuissant :
- Oscar ne peut pas lire les messages chiffrés échangés entre Alice et Bob.
- Oscar ne peut pas modifier un message chiffré échangé entre Alice et Bob : les modifications sur le message chiffré se traduiraient par un message incompréhensible une fois déchiffré.
- Oscar ne peut pas se faire passer pour Alice ou Bob *(attaque dite de l'homme du milieu)* car s'il crée un faux message chiffré avec une "fausse" clef, une fois déchiffré avec la "vraie" clef partagée, le message sera incompréhesible et Alice ou Bob se rendra compte de la supercherie.

Finalement, le chiffrement symétrique apporte toutes les garanties nécessaires à condition qu'une nouvelle clef partagée soit utilisée à chaque nouvelle connexion entre Alice et Bob. Mais pour cela il faut qu'Alice et Bob se mettent d'abord d'accord sur une clef partagée en utilisant le réseau alors même que leur connexion n'est pas encore sécurisée. Dit autrement, ils doivent réussir à se mettre d'accord sur une clef partagée alors qu'Oscar peut écouter tous leurs échanges. Cela peut paraître impossible à réaliser. Pourtant c'est bel et bien possible.

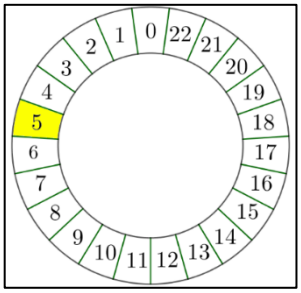
Remarque : Il reste une attaque possible qui n'a pas été évoquée : une attaque par rejeu. Pour cela Oscar enregistre les messages échangés et rejoue les messages de Bob auprès d'Alice un peu plus tard (même si Oscar ne comprend rien à ce qu'il rejoue). Il peut ainsi se faire passer pour Bob auprès d'Alice.
Pour éviter ce type d'attaque il existe des dispositifs mais qui sortent du cadre de ce cours : renouveler la clef partagée à chaque nouvelle connexion, introduire des nombres aléatoires à certaines étapes de la connexion pour garantir l'unicité de chaque connexion (notion de nonce) etc.

B : Méthode asymétrique (à clef publique) Diffie-Hellman d'échange de secret partagé.

Alice et Bob cherchent à obtenir un message qu'eux seuls connaissent (ce sera la clef partagée du A de ce cours) sachant qu'ils communiquent sur un réseau écouté par Oscar. Pour y parvenir ils peuvent utiliser la méthode de Diffie-Hellman qui repose sur des propriété arithmétiques (propriétés liées au calcul modulo p où p est un nombre premier).

Méthode d'échange de clef de Diffie-Hellman

La méthode de Diffie-Hellman a été élaborée par Diffie et Hellman en 1976 (cryptologues américains). Elle repose sur une *clef publique* constituée de deux nombres : un nombre premier P et un nombre graine G plus petit que P.
Puisque tous les calculs se feront modulo P, voici l'image que l'on peut se faire de la *clef publique* lorsque P = 23 et G = 5.

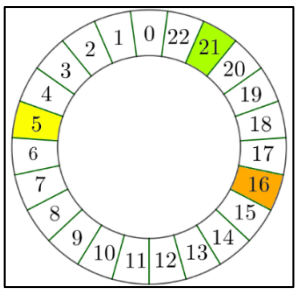


Clef publique : P = 23 et G = 5

Il est à noter que P et G sont connus de tous, d'ailleurs les valeurs qu'il est recommandé d'utiliser sont disponibles dans la RFC5114 mais sont beaucoup plus grandes que 23 et 5 (voir dernière page du cours).

Alice choisit alors un nombre a (qu'elle garde secret, par exemple a=8) puis calcule :
$A = G^a \% P = 5^8 \% 23 = 16$
Bob fait la même chose avec un nombre b (qu'il garde secret, par exemple b = 13) :
$B = G^b \% P = 5^{13} \% 23 = 21$

Alice et Bob s'échangent alors les résultats obtenus en utilisant le réseau : les informations représentées ci-contre sont donc connues d'Oscar. Par contre a = 8 et b = 13 restent secrets.



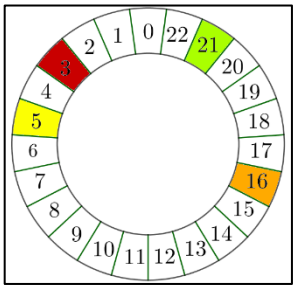
Informations connues d'Oscar

Reste alors à Alice et Bob à faire un dernier calcul pour obtenir leur secret partagé. Pour cela ils refont ce qu'ils ont déjà fait mais avec le résultat de Bob (ou d'Alice) à la place de G :
Alice calcule $Y = B^a \% P = 21^8 \% 23 = 3$
Bob calcule $Z = A^b \% P = 16^8 \% 23 = 3$
Alice et Bob sont certains d'obtenir le même résultat (qui est finalement $G^{a*b} \% P$). Y = Z = 3 est désormais leur secret partagé (en brun ci-dessous).

Intuitivement, on se dit qu'Oscar – qui connait beaucoup d'informations – devrait réussir à retrouver le secret entouré en rouge.

En fait cela n'est pas possible car les algorithmes pour retrouver les valeurs secrètes a et b à partir des quatre valeurs connues d'Oscar (P, G, A et B) sont très coûteux et reviennent (en simplifiant abusivement et en l'état actuel des connaissances) à faire de la force brute (https://fr.wikipedia.org/wiki/Logarithme_discret pour plus de détails).

Et Oscar aura le même problème s'il cherche à retrouver le secret directement à partir de P, G, A et B … le secret d'Alice et Bob est donc bien secret malgré les informations connues par Oscar.



En brun, la clef partagée calculée par Alice et Bob mais inaccessible à Oscar

Remarque 1 : Ici il faut bien noter que Alice et Bob ne contrôlent pas du tout le contenu du secret qu'ils vont partager. Autrement dit, le nombre 3 qui leur servira de clef partagée dans l'exemple ci-dessus est – de leur point de vue – trouvé "au hasard". C'est juste un nombre qu'Alice et Bob sont les seuls à connaître. Ils auraient très bien pu tomber sur 17, sur 9, sur 11 … Ils ne peuvent donc pas se servir de cette méthode pour obtenir comme secret partagé "RDV à 23h00 au parc avec la mallette."

Remarque 2 : Voir en fin de cours pour de vraies valeurs de P et G utilisées en production. Les valeurs conseillées sont par exemple disponibles dans la RFC3526 ou dans la RFC5114. La RFC va jusqu'à 8192 bits (P écrit en base 10 comporte alors 2187 chiffres) sachant que la RFC précise qu'avec P écrit sur 2048 bits, la sécurité apporté est équivalente à celle d'une clef symétrique de 112 bits seulement (voir la remarque à la fin du A.2).

Remarque 3 : Seul le principe de Diffie-Hellman est à connaître : grâce à des propriétés de l'arithmétique sur les nombres premiers, il est possible d'échanger – à travers un réseau potentiellement contrôlé par l'attaquant – une clef partagée (*mais non choisie*). Le reste est de la culture générale.

Bilan

Grâce à la méthode asymétrique de Diffie-Hellman, Alice et Bob peuvent créer une clef partagée qu'eux seuls connaissent. Ils peuvent ensuite utiliser cette clef partagée pour établir une communication chiffrée symétriquement (A de ce cours).
Il reste un problème de taille à régler : au moment où Alice contacte Bob pour l'échange de clefs de Diffie-Hellman, comment est-elle certaine que c'est bien à Bob qu'elle s'adresse et pas à Oscar ? Comment va-t-elle pouvoir authentifier Bob ?

C : Méthode asymétrique (à clef publique) RSA & application à l'authentification d'un serveur

C.1) Méthode de chiffrement asymétrique RSA : principe, possibilités offertes et limitations

La méthode de chiffrement RSA fonctionne avec deux clefs : une clef publique et une clef privée. Ces clefs sont constituées de plusieurs nombres et agissent – comme pour Diffie-Hellman – grâce à des propriétés arithmétiques sur les nombres premiers. En revanche chaque machine qui souhaite utiliser RSA doit créer sa propre paire de clefs (alors qu'avec Diffie-Hellman chacun peut utiliser la même clef publique) : pour pouvoir générer de telles clefs facilement il existe des logiciels spécifiques (par exemple openssl sous Unix). Ainsi on peut facilement générer de très nombreuses paires de clefs RSA (vous pouvez aller voir en fin de cours un exemple de paire de clefs RSA).

Un message chiffré par une clef publique peut être déchiffré uniquement avec la clef privée associée et – réciproquement – un message chiffré par une clef privée peut uniquement être déchiffré par la clef publique associée.
Considérons une machine X équipée d'une paire de clefs publique et privée RSA, cela permet deux choses :

- N'importe qui peut chiffrer des messages avec la clef publique (y compris des attaquants) : seule la machine X pourra les déchiffrer avec sa clef privée.

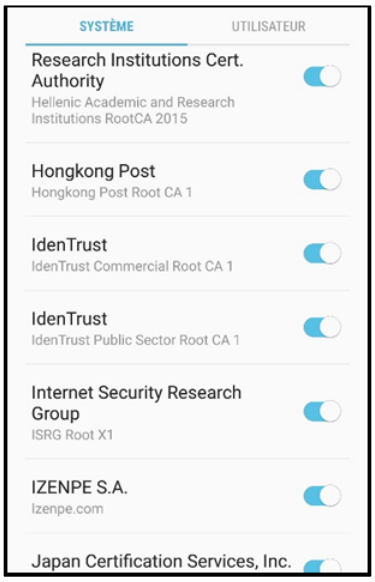
- La machine X peut chiffrer des messages avec sa clef privée : n'importe qui pourra utiliser la clef publique pour déchiffrer le message (y compris des attaquants). Une fois le déchiffrement effectué, il est prouvé que le message a été chiffré par le propriétaire de la clef privée.

Exercice (court mais fondamental) :
Alice cherche à contacter Bob : la machine X distante lui répond en lui remettant un message chiffré par une clef privée et lui donne également la clef publique associée.
Alice utilise la clef publique et déchiffre le message qui lui dit "Bonjour Alice, je suis Bob, comment fait-on pour la suite de la connexion ?".
La machine X est-elle celle de Bob ? Le message est-il celui de Bob ?

Bilan :
RSA permet d'envoyer un message à une machine disposant de la clef privée sans que personne d'autre ne puisse lire le message. RSA permet aussi de lire un message en étant certain qu'il a été *chiffré* par la machine disposant de la clef privée. C'est donc un premier pas vers l'authentification mais pour l'instant insuffisant.

C.2) Authentification grâce à la méthode de chiffrement RSA

Le problème fondamental pour l'authentification est le fait que lorsque Alice envoie ou reçoit un message, il transite par le réseau qui est supposé être contrôlé par Oscar. Tant qu'elle n'utilise que des informations ayant transité par le réseau elle peut être intégralement manipulée par Oscar (voir le film The Truman Show).
La seule possibilité pour s'en sortir est qu'Alice dispose d'informations fiables n'ayant pas transité par le réseau.



Le magasin d'un navigateur sous Android avec quelques noms de clefs publiques d'autorités de certification

Pour cela, chaque système d'exploitation (Windows, MAC, Android …) ou navigateur (Chrome, FireFox, Safari, Edge) est équipé dès son installation des clefs publiques d'une centaine d'organismes mondiaux appelés autorités de certification. On peut donc avoir confiance dans les clefs publiques des autorités de certification puisqu'elles n'ont pas transité par le réseau. On peut également avoir confiance dans les autorités de certification qui sont soumises à des contrôles stricts (certaines se font parfois bannir lorsqu'elles ne sont pas assez sérieuses : https://en.wikipedia.org/wiki/StartCom).

Dès lors, Bob peut facilement se faire authentifier. Pour cela il contacte une autorité de certification AC qui a sa clef publique PUB_{AC} installée sur tous les navigateurs de la planète (dont celui d'Alice). Cette AC va vérifier que Bob est bien propriétaire du site à certifier.
Une fois les vérifications effectuées, cette autorité de certification AC va alors :

- générer une paire de clefs RSA pour Bob (appelons ces clefs PUB_{BOB} et $PRIVE_{BOB}$),

- avec sa clef privée $PRIVE_{AC}$, chiffrer un fichier contenant (la clef PUB_{BOB} ainsi que des informations indiquant qu'il s'agit bien de la clef publique de Bob). Ce fichier chiffré s'appelle également un certificat (voir en annexe pour un exemple de Certificat).

Tout cela est remis à Bob qui dispose : d'une clef publique PUB_{BOB}, d'une clef privée $PRIVE_{BOB}$ et d'un certificat chiffré par $PRIVE_{AC}$ contenant (PUB_{BOB} + des informations disant que cela appartient bien à Bob).

Plaçons nous maintenant du côté d'Alice : elle cherche à établir une connexion avec Bob.
Une machine lui répond en lui donnant une clef publique PUB_{BOB} et un certificat chiffré par la clef $PRIVE_{AC}$.
Avec la clef publique PUB_{AC} installée nativement sur son navigateur, Alice déchiffre le certificat qu'elle vient de recevoir. Elle est certaine que ce certificat vient de AC car la clef PUB_{AC} utilisée pour déchiffrer n'a jamais transité sur le réseau. Elle est donc certaine que le contenu de ce certificat est fiable.
Or ce certificat indique que la clef publique PUB_{BOB} fournie est bien la vraie clef publique du vrai Bob.

Là Alice réfléchit et se dit que certes, c'est le certificat du vrai Bob et la vraie clef publique du vrai Bob, mais la machine avec qui elle vient de communiquer ? Est-ce que ce ne serait pas Oscar qui lui aurait remis le certificat de Bob ?

Pour s'assurer que ce n'est pas le cas il lui suffit pour continuer la communication :

- de chiffrer systématiquement ses messages avec PUB_{BOB} : pour que la machine distante puisse les déchiffrer il faudra nécessairement utiliser $PRIVE_{BOB}$ que seul le vrai Bob possède,
- d'exiger que tous les messages venant de la machine distante soient chiffrés avec $PRIVE_{BOB}$.

Remarque : pour éviter les attaques par rejeu on pourrait rajouter des nonces dans le discours ci-dessus mais cela risquerait de complexifier le propos qui demande déjà une certaine réflexion intellectuelle …

C.3) Bilan : sécurisation d'une communication https

La méthode de chiffrement RSA, en utilisant une autorité de certification, permet d'authentifier le serveur de Bob. Cela requiert deux paires de clefs RSA : une pour l'autorité de certification et une pour Bob.
Cela requiert également une chaîne de confiance : Alice fait confiance à son système d'exploitation ou à son navigateur, qui eux même font confiance à une centaine d'autorités de certification qui sont chargées des vérifications.

Il est important de noter qu'en commençant la communication avec le protocole décrit ci-dessus, Alice s'assure uniquement que la machine avec laquelle elle communique est bien la machine de Bob (authenticité). La confidentialité n'est pas assurée par ce seul protocole puisque les messages envoyés par Bob peuvent être déchiffrés par tout le monde (puisque tout le monde dispose de la clef publique PUB_{BOB}).

C'est pourquoi le protocole de Diffie-Hellman est particulièrement bienvenu puisqu'il permet d'échanger un secret partagé même en se sachant écouté. En effectuant l'échange de clef partagée de Diffie-Hellman tout en chiffrant/déchiffrant les messages avec RSA, Alice s'assure finalement :
- que c'est bien avec Bob qu'elle communique tout au long de l'échange de clef partagée (grâce à RSA),
- que la clef partagée obtenue à la fin de l'échange n'est pas connue d'Oscar (grâce à Diffie-Hellman).

Une fois que l'échange de clef partagée grâce à la méthode de Diffie-Hellman sous authentification RSA a eu lieu, Alice sait que seuls elle et le vrai Bob possèdent la clef partagée de chiffrement symétrique. Ils peuvent alors entamer une communication sécurisée : Bob est authentifié, la confidentialité et l'intégrité des messages sont assurées.

Remarque : On pourrait se demander pourquoi tout ce fatras de méthodes différentes. En effet on pourrait théoriquement n'utiliser que RSA pour sécuriser les communications. Mais RSA est très gourmand et nécessite beaucoup de calculs …

Exercice : Compléter le schéma ci-dessous résumant la sécurisation d'une connexion https.

Annexe 1 :

Nombres premier P et générateur G utilisables pour Diffie-Hellman (de longueur 8192 bits) selon RFC3526 :

```
P=109074813561941592945029492935978450034815512495317221177410110696615016892278563902853247384883681776971216416907643296922469875267467766273999426578543723359615704597092233804069810050786103304731233182398243527947570019986097161273254052879655450286791974677698375939147598714252131587871957751914881183087991942693995848708754096571641916746749932615622652967520917227700137759124814756378288055886108332717415401497513489312511601577631889029596069801161415772128252753946881651931933333750311477719236041228172101895583437761548046847925274886732036238535559660179512280675621771357981987063432156190781325515370395079527123265240489498386949217448165230380349888136621050864726366837651413103110233683748899977574404673365182723939535354034841487285463971929469432345018688418982254454064722698729216069318473465494190693664657613026097219328031717169641897155395416144619175909371952495111670557736207348131929604120128351615426904438925772770028968411946028348045230620413002491387998113590802698386820596931816781968085099864969441690795271290496240493777578969891720735635522745506618381584766913553054975543981948032173292586906913614608532638233462874545639807160305805163420938670870330654590319960852382451372962513665912822110096773545051995240424819826281383109737426165038001727791697532413484657468130733701738083035368062321633694947130619168643824930568641338023104609645095359408937554028503729247092939511402830554745258496207430943815182543790297601289174935519867842060372203490031136489304649576140433393868614003784803091629254327368453364003263763910077450237154247930247369838869289242094647894773380038778274141778648477019010886787977899163321862864053398261932246615488301145229189025233648723608665439609385389862880581317755916207636315443649447750787129411984163786770172216660983120184548407807051804133686980839845462558692120130818563888808269940868653604519264956919811035365994311180230063610650986502394366182943642656300791728205089442938884174888539829070774305297360535927751574961973082377321589475512176146788786532770711557380426451920634921585019519536481338752681174247413154980213024650634120702033579770678070540694527543880626597851620970679570257924407538049023174103086261496878330620786968786810842363997198320907762475808049998827559139278726762718244289280964687422826317243564236858826013916196283612148196609274532548864105423883929513899297933544611009032523095527687052461135912491839274035315429485838335 9
```

G=2

Annexe 2 :

Un exemple de paire de clefs privée et publique pour RSA (clef publique de longueur 2048 bits) obtenue grâce à openssl :

```
$ openssl genrsa | openssl rsa -text -out mykey.txt
```

Mathématiquement modulus et publicExponent constituent la clef publique alors que modulus et privateExponent constituent la clef privée. Voir https://fr.wikipedia.org/wiki/Chiffrement_RSA pour plus d'informations (modulus est le nombre n, publicExponent est le nombre d et privateExponent est le nombre e).
Le texte situé sous les deux colonnes correspond aux deux colonnes encodées au format pem.

```
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:aa:0d:57:33:41:b7:0a:70:7e:a7:9e:de:ce:ea:
    e3:e0:03:a5:69:e6:f9:f5:c0:dc:c6:b1:e5:68:50:
    4b:0e:99:74:9f:bd:85:4f:49:46:ff:fe:f5:3b:5a:
    bc:b7:0f:0d:13:0a:2d:1f:4a:fe:d8:cf:e4:54:88:
    2b:e0:4f:7e:cc:ac:95:5b:db:61:2a:88:54:c3:25:
    b8:bf:99:e9:df:cc:94:d3:14:58:6e:0b:21:15:16:
    12:09:fe:ad:f3:06:ed:0f:81:56:e0:c3:c6:f0:12:
    21:43:87:72:32:59:9a:df:ec:28:9a:34:44:0c:59:
    58:0f:fb:b3:46:7b:36:fe:6e:2c:fc:7b:08:38:33:
    4e:ed:81:e0:9e:20:23:e0:84:4a:32:94:7f:9e:83:
    ae:21:a0:c4:3e:d2:b6:43:36:e5:f5:94:7a:88:64:
    a4:00:99:6e:d8:53:df:35:6f:c6:cf:6e:cb:4f:a4:
    fd:6c:de:ed:18:05:9f:aa:dd:14:12:12:a2:af:f0:
    d5:ee:26:60:0b:d0:07:ba:be:c8:8f:98:c2:1f:5c:
    92:74:bd:c5:0f:31:f7:95:cb:32:9d:dd:e7:1c:f8:
    86:36:c8:5b:ab:89:b2:92:2c:2d:c1:84:4d:58:4d:
    05:44:f6:ce:f2:18:6f:45:79:68:2f:e3:c9:21:80:
    e9:51
publicExponent: 65537 (0x10001)
privateExponent:
    00:9e:78:1a:27:f0:f8:73:69:0c:0e:86:95:99:fb:
    f8:e9:5c:5c:7c:c9:3b:6c:d2:12:ab:b3:42:56:a8:
    64:99:b1:55:a6:3b:06:0d:31:fb:51:3a:b6:2b:5e:
    ce:78:45:35:68:e5:d5:d0:d0:a1:97:48:7c:be:6d:
    bb:7d:a8:77:40:a3:1f:f8:df:02:b8:91:1d:74:52:
    25:cd:9f:cf:fb:b7:f9:84:8e:f4:2d:70:7f:9e:d7:
    d7:6d:ab:01:4d:75:c9:da:e6:2c:20:ff:30:d7:ad:
    fd:83:a1:a8:40:d5:91:e7:54:7b:2d:e0:c0:fd:45:
    11:34:ab:d9:90:37:e8:b2:c6:52:72:61:db:d2:01:
    e8:4e:3a:8f:06:74:56:39:ac:f9:21:f5:fe:39:74:
    1d:25:e6:b9:3a:58:4e:f2:47:33:78:a2:cc:aa:6e:
    8d:46:47:40:a1:ee:98:3d:87:a8:0c:df:a2:1d:f0:
    4a:b3:d8:dc:bc:73:c4:b0:0a:a3:8c:f3:9b:1e:2a:
    6c:c7:19:2b:c3:99:72:b3:94:cf:38:12:c1:65:32:
    cf:93:9d:45:70:9f:5f:48:c6:60:14:73:b2:16:27:
    91:55:d4:a4:bd:56:ee:19:99
prime2:
    00:cb:4e:ee:ac:80:47:c8:7b:1e:64:b1:92:a7:ff:
    c4:37:61:a8:d5:39:d9:32:59:d1:9c:90:4d:57:c4:
    a7:cb:5a:61:14:4a:b9:1d:22:15:7b:68:86:cb:66:
    b3:bd:b1:e1:47:23:fd:df:f3:8c:fa:7f:d8:dc:d2:
    d6:5c:16:f8:38:9f:c5:60:e8:a5:45:f3:c3:e4:7c:
    91:48:80:be:1c:10:79:6e:9a:15:e0:e9:e7:cf:7a:
    05:ba:9f:1b:06:a6:46:75:20:b3:04:bf:87:0c:fb:
    cf:7f:51:4f:5a:d2:d7:f0:a8:51:d9:20:8a:e7:f2:
    fd:e3:b0:ea:8a:1b:0f:50:79
exponent1:
    6b:84:a5:03:b4:23:11:66:e4:a5:4a:fc:62:b9:43:
    76:c2:82:35:5c:5e:ec:b8:e8:9d:0e:1c:e9:06:14:
    40:ea:ad:f2:79:06:b4:7b:f0:40:2c:18:30:be:be:
    5a:69:f2:94:c9:e2:3a:8f:09:5e:08:70:19:87:3f:
    c1:26:4e:22:01:b8:98:4c:e1:e9:74:f1:e6:ae:9e:
    14:28:b4:ce:3c:23:6b:72:2c:25:be:bd:09:2c:ac:
    6a:5d:0d:48:ae:68:96:1c:06:e1:55:29:9f:33:20:
    60:32:fc:87:4c:bf:7f:bf:7c:c7:48:41:08:36:de:
    17:87:2b:21:97:e0:ff:21
exponent2:
    19:69:c4:6b:cb:9d:2d:72:36:5a:5f:d7:f5:28:03:
    5e:e8:d6:31:d1:09:55:41:e4:f1:ad:17:fd:e0:97:
    18:d4:33:4e:56:08:cd:9b:75:13:f7:fc:e1:6f:f5:
    07:c1:34:67:b0:18:0b:e6:65:b8:ea:42:31:58:29:
    73:59:6b:ad:a8:4b:03:d2:10:d6:ad:a1:ce:ed:c3:
    3b:4c:dc:76:a5:98:88:3a:ba:81:62:bb:97:33:a9:
    83:aa:5f:b6:75:12:59:91:bc:ce:db:22:06:7d:73:
```

```
    00:4c:1d:c4:a2:f7:59:09:c8:b9:91:a2:39:be:21:
    66:f0:1f:08:12:55:0a:a1:f0:bf:8d:db:c8:8d:c9:
    54:30:19:e2:21:0d:63:ee:06:09:49:d3:5a:2c:2c:
    ff:bd:fd:1d:88:76:56:ec:ab:3b:06:69:81:03:89:
    36:48:ef:20:76:53:41:7e:c4:13:18:39:0c:33:6f:
    62:6a:45:fc:42:eb:e9:95:15:70:f1:58:19:16:87:
    93:41
prime1:
    00:d6:1f:ee:24:3a:ad:5d:75:b3:5c:32:6c:35:11:
    41:80:67:80:ba:f0:b1:50:3f:e4:c9:b0:da:7f:4a:
    c0:cb:1f:b3:0e:76:21:26:c2:e4:4e:82:d1:a4:62:
    12:ed:9c:b6:95:c5:1e:6f:98:79:15:b3:df:87:72:
    72:76:b7:74:a2:e6:5b:2e:af:0c:75:04:e7:f9:77:
    c2:fe:5a:52:e9:0f:01
coefficient:
    00:b6:01:14:e7:74:01:1b:fd:9e:9d:b8:3f:64:aa:
    77:bb:9a:03:f1:0e:2c:06:58:a1:25:ea:ad:8f:bc:
    4e:8e:f1:a7:ea:92:7a:d2:60:da:99:5e:05:b8:bc:
    8f:49:c3:81:52:ec:25:82:d0:30:4a:d4:a6:15:46:
    ad:51:34:82:90:85:e7:11:b2:4b:37:04:5a:bc:33:
    90:3c:b2:90:97:cb:86:21:92:3f:06:d9:6a:c4:70:
    50:93:51:e2:ff:d7:bc:22:fd:8f:df:f0:b2:50:db:
    84:04:af:2c:3b:14:2a:a7:4f:c2:16:fc:89:4a:cb:
    d0:ed:89:a5:df:9c:8f:1a:3f
```

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAqg1XM0G3CnB+p57ezurj4AOlaeb59cDcxrHlaFBLDpl0n72FT0lG//71O1q8tw8NEwotH0r+2M/kVIgr4E9+zKyVW9thKohUwyW4v5
np38yU0xRYbgshFRYSCf6t8wbtD4FW4MPG8BIhQ4dyMlma3+womjREDFlYD/uzRns2/m4s/HsIODNO7YHgniAj4IRKMpR/noOuIaDEPtK2Qzbl9ZR6iGSk
AJlu2FPfNW/Gz27LT6T9bN7tGAWfqt0UEhKir/DV7iZgC9AHur7Ij5jCH1ySdL3FDzH3lcsynd3nHPiGNshbq4mykiwtwYRNWE0FRPbO8hhvRXloL+PJIY
DpUQIDAQABAoIBAQCeeBon8PhzaQwOhpWZ+/jpXFx8yTts0hKrs0JWqGSZsVWmOwYNMftROrYrXs54RTVo5dXQ0KGXSHy+bbt9qHdAox/43wK4kR10UiXN
n8/7t/mEjvQtcH+e19dtqwFNdcna5iwg/zDXrf2DoahA1ZHnVHst4MD9RRE0q9mQN+iyxlJyYdvSAehOOo8GdFY5rPkh9f45dB0l5rk6WE7yRzN4osyqbg
BMHcSi91kJyLmRojm+IWbwHwgSVQqh8L+N28iNyVQwGeIhDWPuBglJ01osLP+9/R2IdlbsqzsGaYEDiTZI7yB2U0F+xBMYOQwzb2JqRfxC6+mVFXDxWBkW
h5NBAoGBANYf7iQ6rV11s1wybDURQYBngLrwsVA/5Mmw2n9KwMsfsw52ISbC5E6C0aRiEu2ctpXFHm+YeRWz34dyjUZHQKHumD2HqAzfoh3wSrPY3LxzxL
AKo4zzmx4qbMcZK8OZcrOUzzgSwWUyz5OdRXCfX0jGYBRzshYnkVXUpL1W7hmZAoGBAMtO7qyAR8h7HmSxkqf/xDdhqNU52TJZ0ZyQTVfEp8taYRRKuR0i
FXtohstms72x4Ucj/d/zjPp/2NzS1lwW+DifxWDopUXzw+R8kUiAvhwQeW6aFeDp5896BbqfGwamRnUgswS/hwz7z39RT1rS1/CoUdkgiufy/eOw6oobD1
B5AoGAa4SlA7QjEWbkpUr8YrlDdsKCNVxe7LjonQ4c6QYUQOqt8nkGtHvwQCwYML6+WmnylMniOo8JXghwGYc/wSZOIgG4mEzh6XTx5q6eFCi0zjwja3Is
Jb69CSysal0NSK5olhwG4VUpnzMgYDL8h0y/f798x0hBCDbeF4crIZfg/yECfxlpxGvLnS1yNlpf1/UoA17o1jHRCVVB5PGtF/3glxjUM05WCM2bdRP3/O
Fv9QfBNGewGAvmZbjqQjFYKXNZa62oSwPSENatoc7twztM3HalmIg6uoFiu5czqYOqX7Z1ElmRvM7bIgZ9c3J2t3Si5lsurwx1BOf5d8L+WlLpDwECgYEA
tgEU53QBG/2enbg/ZKp3u5oD8Q4sBlihJeqtj7xOjvGn6pJ60mDamV4FuLyPScOBUuwlgtAwStSmFUatUTSCkIXnEbJLNwRavDOQPLKQl8uGIZI/BtlqxH
BQk1Hi/9e8Iv2P3/CyUNuEBK8sOxQqp0/CFvyJSsvQ7Yml35yPGj8=
-----END RSA PRIVATE KEY-----
```

Annexe 3 : exemple de certificat

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:97:fe:ad:62:de:fa:b6:2e:51:da:c0:b1:be:c8:6a:d2:3c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3        <---- AC
        Validity
            Not Before: Oct 30 21:46:33 2020 GMT                <---- période de validité du certificat
            Not After : Jan 28 21:46:33 2021 GMT
        Subject: CN = jupyter.ovh                               <---- site certifié
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption                 <---- début de la clef publique du site
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:be:05:e7:70:d7:5f:54:3b:5c:2d:35:00:49:75:
                    e0:42:67:34:9e:96:6a:53:ff:d5:26:88:e0:bb:8e:
                    53:61:0e:3a:1c:51:54:ec:99:02:1a:0f:1f:aa:89:
                    74:b8:4f:66:f6:09:a0:c1:47:4c:f3:ba:96:be:cb:
                    54:f4:a7:74:e2:75:e6:ed:d1:b0:fa:63:ed:a2:3a:
                    75:f5:54:a6:b8:9a:ed:30:5c:bf:ae:3d:54:af:c0:
                    b2:c1:10:37:03:80:35:ae:83:7b:85:40:cd:1c:06:
                    67:c6:3f:7f:a1:55:a2:71:47:27:5f:82:35:d5:5e:
                    ed:31:e6:df:00:82:d9:91:aa:22:ab:7e:73:f4:bd:
                    0c:5f:bc:2c:b6:29:ff:3d:5d:62:4f:7e:d2:61:2d:
                    43:6d:75:ce:c3:ec:f8:ba:6e:73:75:cc:af:1d:d7:
                    84:26:96:b8:19:05:d4:92:cf:7f:de:67:f7:40:13:
                    30:ed:2b:c8:cc:7a:65:8b:5f:74:22:aa:d5:7b:66:
                    b5:b3:0c:53:6b:29:d6:34:2a:0a:57:e9:1c:c6:ad:
                    63:f6:90:19:8c:9e:97:4b:e5:95:d5:78:75:c8:91:
                    52:dd:f1:d9:dc:9d:05:7a:ef:3e:25:ce:d2:a7:cf:
                    be:a1:f8:c5:74:67:d5:30:6b:c1:5c:ca:41:06:9d:
                    06:b1
                Exponent: 65537 (0x10001)                       <---- fin de la clef publique du site
        X509v3 extensions:
            X509v3 Key Usage: critical                                          |
                Digital Signature, Key Encipherment                             |  détails
            X509v3 Extended Key Usage:                                          |  techniques
```

```
            TLS Web Server Authentication, TLS Web Client Authentication       |
        X509v3 Basic Constraints: critical                                     V
            CA:FALSE
        X509v3 Subject Key Identifier:
            AF:6B:A8:1B:69:1B:4F:1D:48:CE:ED:BE:75:C7:1B:C1:4F:03:A9:B5
        X509v3 Authority Key Identifier:
            keyid:A8:4A:6A:63:04:7D:DD:BA:E6:D1:39:B7:A6:45:65:EF:F3:A8:EC:A1

        Authority Information Access:
            OCSP - URI:http://ocsp.int-x3.letsencrypt.org
            CA Issuers - URI:http://cert.int-x3.letsencrypt.org/

        X509v3 Subject Alternative Name:
            DNS:jupyter.ovh
        X509v3 Certificate Policies:
            Policy: 2.23.140.1.2.1
            Policy: 1.3.6.1.4.1.44947.1.1.1
              CPS: http://cps.letsencrypt.org

        CT Precertificate SCTs:
            Signed Certificate Timestamp:
                Version   : v1 (0x0)
                Log ID    : 44:94:65:2E:B0:EE:CE:AF:C4:40:07:D8:A8:FE:28:C0:
                            DA:E6:82:BE:D8:CB:31:B5:3F:D3:33:96:B5:B6:81:A8
                Timestamp : Oct 30 22:46:33.683 2020 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:46:02:21:00:BB:48:C9:FE:B0:2B:A8:63:5F:FF:6A:
                            D2:49:79:B7:F8:42:F8:D1:05:24:44:20:E1:8E:2D:E4:
                            D1:E4:43:CF:E8:02:21:00:E2:7A:7D:ED:5E:8F:1F:A2:
                            AF:27:BA:76:20:E9:0B:9E:38:CC:EA:07:85:54:C6:B2:
                            FA:13:F7:3A:24:21:17:31
            Signed Certificate Timestamp:
                Version   : v1 (0x0)
                Log ID    : 7D:3E:F2:F8:8F:FF:88:55:68:24:C2:C0:CA:9E:52:89:
                            79:2B:C5:0E:78:09:7F:2E:6A:97:68:99:7E:22:F0:D7
                Timestamp : Oct 30 22:46:33.713 2020 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:45:02:20:17:AF:A7:88:46:0E:AD:32:34:B2:67:E5:
                            52:43:2B:60:23:94:CF:96:29:DF:08:D5:CF:B2:5F:8F:
                            D7:8D:3C:D0:02:21:00:91:1B:D3:1F:61:C6:AB:F8:20:
                            AE:95:BA:AD:CF:5A:45:0E:A6:0D:20:F6:C2:04:75:88:
                            A8:EB:B4:74:1F:48:A9
    Signature Algorithm: sha256WithRSAEncryption
         88:5e:dc:56:18:3a:5f:fb:ca:b0:e3:a4:5f:88:1d:14:ef:14:
         d6:50:b2:11:6e:41:14:e2:47:4b:f0:a3:16:77:17:40:e2:c8:
         c8:0d:a8:e6:26:d0:68:ec:71:9c:78:23:05:89:a9:89:61:2c:
         13:cf:1f:d9:81:4f:62:5a:6b:cc:d7:24:18:1b:59:89:c8:8d:
         15:92:c5:cf:cb:05:bf:72:8c:1b:fb:47:66:cd:08:4a:69:95:
         5f:f3:a8:ff:86:a6:40:11:d0:80:4b:cf:77:d6:32:af:ae:80:
         82:1c:f6:71:55:42:4b:a6:01:a1:80:55:a0:85:5e:7a:2d:2a:
         4d:46:b6:03:6a:6d:7d:47:c5:59:ff:de:4a:5f:10:68:fd:ac:
         ca:11:1b:77:e2:3f:6c:5b:66:f8:35:ac:5e:40:d4:e8:10:98:
         5e:c2:ec:41:cb:ec:a2:12:8a:4c:94:c4:94:9a:2a:fb:ad:01:
         ef:d5:87:db:50:39:ff:0d:53:13:df:05:9b:ad:14:32:76:9e:
         d7:2e:b7:2a:6b:c8:74:1e:46:bd:66:a6:ab:f2:85:f7:fd:8b:
         55:4f:32:5e:1e:a3:bd:86:62:c6:90:bc:e3:fa:e3:84:26:06:
         76:dc:57:9d:52:f7:5e:5f:05:e0:c7:f2:ae:11:1e:be:6a:dd:
         d4:c9:74:c8
```