CAA 2020

Lab #2

Rémi Poulard, le 23.05.2020

Choix d'implémentation

Courbe elliptique

En ce qui concerne la courbe elliptique j'ai choisi de prendre la courbe `secp256r1`. J'ai choisi ce que recommandait l'ANSSI (sur le site keylength.com), c'est à dire 256bits. De plus, j'ai pensé que dans le cadre de ce laboratoire nous pouvions ne pas faire attention à la durée de vie des courbes (jusqu'à 2030 pour `secp256r1` apparemment), en production il serait peut-être plus "futur proof" de prendre la courbe de 384bits.

Le fait de choisir cette courbe m'a donc donné un paramètre τ d'une longueur de 128bit car le nombre de bit de q doit être égal à 2τ.

Authenc/AuthDec

Pour cette partie nous devions faire en sorte que ce soit "random-key robust". D'après ce qui est noté dans le draft du protocole cette propriété peut être remplie en faisant un `encrypt-then-mac` ou en modifiant GCM. Dans mon cas j'ai choisi d'utiliser `encrypt-then-mac` avec AES et HMAC256.

Pour le mode de AES j'ai choisi CTR car, comme expliqué ici, CTR est très efficace lorsque l'on veux simplement chiffrer les données, ce qui est notre cas. Pour AES et HMAC la taille de la clé utilisé a été de 256 bits. La données nous dit que si l'on choisit AES nous pouvons mettre le nonce à 0 car il est utilisé une seule fois de toute manière.

Fonction de hashage H

Pour la fonction de hashage j'ai choisi d'utiliser BLAKE2b car d'après le cours c'est un algorithme qui est valide dans le futur. De plus, il semble plus rapide que MD5, SHA1, SHA2 et SHA3 et aussi sûr que SHA3.

Il a été possible de définir la taille de l'output de la fonction. D'après la figure donnée la taille de notre sortie devait être de 2τ, ce qui nous fait une sortie de 256 bits.

PRF

Pour la PRF j'ai aussi choisi HMAC256 car, à nouveau, d'après l'ANSSI il est recommandé d'utiliser SHA-256 pour les fonctions de hashage et du coup le fait de choisir HMAC256 me permettait d'avoir SHA256 dans mon MAC.

KDF

Etant donné que nous utilisons la méthode "encrypt-then-mac" nous avions besoin de deux clés différentes. Nous avions a disposition rw. Nous avons donc utilisé une KDF pour dériver nos deux clés. Nous avons choisi la méthode `scrypt`. En ce qui concerne les paramètre de cette méthode nous avons choisi ce qui était conseillé par "Colin Percival" dans sa présentation de 2009, c'est à dire N=14, r=8 et p=1. Pour le sel étant donné que le client

et le serveur doivent avoir le même et qu'il est conseillé d'être de longueur 16 bytes nous avons repris le SSID.

Fonctionnement de OPAQUE

Note: nous sommes dans un corps additif cela veut dire que lorsque nous avons un exposant nous faisons une multiplication et que lorsque nous avons une multiplication nous faisons une addition. Dans la description suivante nous avons adapté les formules au corps additif.

La première phase de OPAQUE est une phase d'enregistrement. On suppose que le client et le serveur communique à travers un canal sécurisé. Lors de cette phase il va se passer les choses suivante:

- Le serveur va tirer des nombres aléatoires $k_s, p_s, p_u \leftarrow_{\mathbb{R}} \mathbb{Z}_q$ ainsi que des points sur une courbe elliptique (clés public du client et du serveur) où $P_u = g * p_u$ et $P_s = g * p_s$.
- Le serveur va calculer une clé rw qui a été généré comme ceci: $rw = H(password, g * H(password) * k_s)$.
- Il va ensuite utiliser rw pour chiffrer la clé privé du client, la clé public du client et la clé public du serveur: $c = AuthEnc_{rw}(p_u, P_u, P_s)$. Comme AuthEnc est composé de AES et HMAC nous utilisons KDF pour dériver deux clés: $aes\_key = KDF(0, rw)$ et $hmac\_key = KDF(1, rw)$ Une fois ces deux clés obtenus nous pouvons utiliser AES comme ceci: $ciphertext = AES_{aes\_key}(p_u, P_u, P_s)$ puis HMAC comme ceci $mac = HMAC_{hmac\_key}(ciphertext)$.
- Puis une fois tout ces éléments calculés, il va les stocker dans un fichier avec comme identificateur l'id de l'utilisateur.

En production cette phase doit se faire pour chaque utilisateur. Dans notre implémentation nous la faisons automatiquement étant donné que nous n'avons qu'un seul utilisateur

Par la suite nous avons la phase de login, cette phase pourrait être représenté par le login sur une page web:

1. Client:
   - Le client va tirer aléatoirement deux nombre aléatoire dans $\mathbb{Z}_q$: $r, x_u \leftarrow_{\mathbb{R}} \mathbb{Z}_q$. Puis il va généré le point X_u sur la courbe comme ceci: $X_u = g * x_u$
   - Il va ensuite calculer α comme ceci $\alpha = (H'(password))^r$. Dans notre cas nous travaillons avec des courbes elliptiques et il nous a été conseillé, dans un premier temps, d'utiliser H' comme ceci: $H'(password) = g * H(password)$. Et donc α se calcule comme ceci : $\alpha = g * H(password) * r$
   - Une fois ces éléments calculé, le client va les envoyer au serveur
2. Serveur:
   - Lorsque le serveur reçoit α et X_u il va d'abord vérifier que $\alpha \in G^*$. Pour faire cela nous allons essayer de reconstruire le point α dans notre courbe, puis, une fois le point reconstruit nous vérifions que ce n'est pas le point à l'infini et que $q * \alpha = O$. Si c'est le cas notre point est correct.
   - Le serveur va donc aller rechercher les informations qu'il a stocké dans le fichier pour cet utilisateur. Il va s'aider du sid (user id) pour récupérer les bonnes informations.
   - Il va ensuite généré $x_s \leftarrow_{\mathbb{R}} \mathbb{Z}_q$. Puis à l'aide de x_s et k_s(récupéré dans le fichier) il va calculer β et X_s comme ceci: $\beta = \alpha^{k_s}$ et $X_s = g * x_s$
   - La prochaine étape consiste au calcule de K. Mais afin de calculer cet élément il va falloir calculer: $ssid', e_u$ et e_s. Pour $ssid'$ le calcul est le suivant : $ssid' = H(sid, ssid, \alpha)$. En ce qui concerne isd j'ai fixé sa valeur à 1 pour le POC, puis je fais en sorte d'avoir une valeur de 128 bits. Je vais donc ajouter 127 fois 0 devant pour avoir ma valeur sur 128 bits. Cela nous laisse la possibilité d'avoir 2^{128} utilisateurs sur notre application. Pour le $ssid$ qui doit représenter l'ide de session j'ai fixé une valeur de 16 charactère. En production nous pouvons imagine prendre des informations fournis par HTTP par exemple pour avoir un vrai id de session. Ce ssid va donc être padé pour avoir une longueur de 128bits. Pour e_u le calcul est le suivant : $e_u = H(X_u, ssid') \bmod q$. Puis e_s est calculé comme ceci: $e_s = H(X_s, ssid') \bmod q$. Maintenant que nous avons tous les éléments nous pouvons calculer K: $K = H((X_u + P_u * e_u) * (x_s + e_s * p_s))$.
   - K va nous permettre de calculer SK, A_s et A_u car ce sera la clé de notre PRF (HMAC256 ici). Donc nous calculons $SK = f_K(0, ssid')$ et $A_s = f_K(1, ssid')$
   - Une fois ces valeurs calculées le serveur va envoyer β, X_s, c et A_s au client

3. Client:

- Lorsque le client reçoit β, X_s, c et A_s il va d'abord vérifier que $\beta \in G^*$. Pour faire cela nous allons essayer de reconstruire le point β dans notre courbe, puis, une fois le point reconstruit nous vérifions que ce n'est pas le point à l'infini et que $q * \beta = O$. Si c'est le cas notre point est correct.
- Le client va ensuite reconstruire rw de la manière suivante: $rw = H(password, \beta * 1/r)$. Etant donné que rw a été utilisé pour dériver les clés de AES et HMAC le client va aussi pouvoir dériver ces clés et il va par la suite pouvoir déchiffrer le paramètre c. Donc nos clés sont dérivé comme ceci: $aes\_key = KDF(0, rw)$ et $hmac\_key = KDF(1, rw)$. Puis une fois que nous avons ces deux clé nous pouvons faire $AuthDec_{rw}(c)$. Si le client arrive a déchiffrer il va pouvoir récupérer p_u, P_u et P_s, s'il n'arrive pas à déchiffrer la transaction est interrompue.
- Le client va à son tour pouvoir calculer $K, ssid', e_u, e_s, SK, A_s$ et A_u. Le calcul de e_u, e_s, A_s, A_u, SK et $ssid'$ ne change pas. Lorsque le client va calculer K il va dans un premier temps vérifier que $X_s, P_s \in G^*$, si c'est le cas il va calculer K et il va le faire de cette manière: $K = H((X_s + P_s * e_s) * (x_u + e_u * p_u))$.
- Maintenant que le client a aussi la clé K il va pouvoir calculer $SK = f_K(0, ssid'),\ A_s = f_K(1, ssid')$ et $A_u = f_K(2, ssid')$
- Le client va ensuite vérifier que le A_s qu'il a calculer et le même que celui qu'il a reçu. Si c'est le cas il va envoyer A_u au serveur et écrire à l'utilisateur que c'est OK, si ce n'est pas le cas il va interrompre la transaction et écrire "Error" dans la console.

4. Server

- Le serveur va vérifier que A_u qu'il a reçu du client est bien égal à $f_K(2, ssid')$. Si c'est le cas il écrit "Ok" sino il écrit "Error".