Note Crypto Référentiel IPsec DR

{LRP,LCR} @ANSSI

23 mars 2021

Résumé

De manière à favoriser la sécurité et l'intéropérabilité des chiffreurs IPsec visant agrément DR, un référentiel a été élaboré. Celui-ci, par une limitation et une clarification des modes, algorithmes et fonctionnalités autorisés pour ces produits, vise à simplifier à la fois leur évaluation et leur mise en oeuvre dans le cadre d'une utilisation pour protéger des réseaux à un niveau DR.

Table des matières

1 Introduction **4**

2 Courbes elliptiques **6**
2.1 `BrainpoolP256r1` 6
2.2 `secp256r1` 7

3 Algorithmes de signature sur courbe elliptique **8**
3.1 Documents de référence 8
3.2 Identifiants pour IKEv2 9
3.3 ECSDSA 9
3.3.1 Notations 10
3.3.2 Génération 10
3.3.3 Signature 10
3.3.4 Vérification 11
3.4 ECDSA 12
3.4.1 Notations 12
3.4.2 Génération 12
3.4.3 Signature 12
3.4.4 Vérification 14
3.5 Vecteurs de test 14
3.5.1 ECSDSA avec SHA256 sur `BrainpoolP256r1` 15
3.5.2 ECSDSA avec SHA256 sur `secp256r1` 16
3.5.3 ECDSA avec SHA256 sur `BrainpoolP256r1` 17
3.5.4 ECDSA avec SHA256 sur `secp256r1` 18

4 Algorithmes d'échange de clé sur courbe elliptique **20**
4.1 Contexte 20
4.2 Utilisation des payload KE{} 21
4.3 Génération des payload 22
4.4 Traitement des payload KE{} reçus 23
4.5 Calcul du secret partagé 24
4.6 Note concernant la réutilisation et l'effacement 24
4.7 Vecteurs de test 25
4.7.1 `secp256r1` 25
4.7.2 `BrainpoolP256r1` 26

5 Génération de secrets et authentification symétrique **28**
5.1 Génération de secrets pour la IKE_SA 28
5.2 Génération du matériel de clé pour SA IPsec 29
5.3 Authentification symétrique 30
5.4 Vecteurs de test 30
5.4.1 `PRF_HMAC_SHA2_256` test 1 30
5.4.2 `PRF_HMAC_SHA2_256` test 6 30

6 Algorithmes de chiffrement intègre **32**
6.1 AES-GCM avec ICV à 16 octets 34
6.1.1 Utilisation dans IKEv2 34
6.1.2 Utilisation dans ESP 35
6.2 AES-CTR avec `AUTH_HMAC_SHA2_256_128` 36
6.2.1 Utilisation dans IKEv2 36
6.2.2 Utilisation dans ESP 37

7 Remarques diverses **38**
7.1 Génération d'aléa 38
7.2 Utilisation de coprocesseur cryptographique 39
7.3 Certificats 39
7.4 Anti-rejeu 40

A Exemples de payload SA **42**

B Scénario de tests **53**

1 Introduction

Dans le cadre du référentiel IPsec/IKEv2 pour les chiffreurs DR, ce document vise à centraliser en un point unique les informations utiles concernant l'ensemble des aspects liés à la cryptographie. Il s'adresse à deux types de public distincts :

1. aux développeurs du démon IKEv2 et de la pile IPsec (et notamment des bibliothèques cryptographiques associées) intégrés dans un produit. Dans ce cadre, le document doit permettre l'implémentation sans ambiguïté des mécanismes cryptographiques (au sens large, les aspects protocolaires relatifs à IKEv2 et ESP ayant trait à la cryptographie en faisant partie).
2. aux évaluateurs de produits de manière à fournir des éléments devant être vérifiés sur les aspects cryptographiques lors de l'évaluation.

Il a pour rôle de définir un périmètre restreint d'algorithmes imposés sur les équipements concernés visant agrément DR et de clarifier leur utilisation, dans l'objectif de simplifier les développements. Il vise aussi à garantir l'interopérabilité entre produits et l'évaluation. À ce titre, les algorithmes et fonctionnalités imposées sont les suivantes :

- Confidentialité et intégrité dans IKEv2 et ESP :
  - AES-GCM avec clé de 256 bits et ICV de 16 octets
  - AES-CTR avec clé de 256 bits et `AUTH_HMAC_SHA2_256_128`
- PRF dans IKEv2 : `PRF_HMAC_SHA2_256`
- Authentification dans IKEv2 :
  - Asymétrique :
    - ECSDSA sur `BrainpoolP256r1` avec SHA256
    - ECSDSA sur `secp256r1` avec SHA256
    - ECDSA sur `BrainpoolP256r1` avec SHA256
    - ECDSA sur `secp256r1` avec SHA256
  - Symétrique :
    - `PRF_HMAC_SHA2_256`
- Anti-rejeu pour ESP : ESN non optionnel.

Il a également vocation à interdire explicitement l'utilisation de certains algorithmes (MD5, SHA1, RSA, DH sur groupe d'entiers) ou de fonctionnalités comme EAP. Les écarts avec le standard IKEv2 résident principalement dans le support de certaines extensions au protocole, rendues obligatoire, comme l'établissement de *childless* IKE_SA (défini dans [RFC6023]), et le

support de nouveaux algorithmes comme ECSDSA (défini dans ce document).

Dans le cadre du référentiel, pour limiter le nombre d'algorithmes devant être supportés par l'ensemble des produits, des choix ont dû être réalisés, prenant notamment en compte la sécurité, la simplicité d'implémentation et d'audit mais également les performances. Même si du simple point de vue de la sécurité, il aurait été possible d'étendre la liste d'algorithmes supportés pour y ajouter par exemple ECKCDSA sur `BrainpoolP512r1` avec SHA512 ou encore AES-CBC avec clés de 256 bits utilisé avec `AUTH_HMAC_SHA2_512_256`, cette extension se serait faite au détriment de la simplicité et de l'auditabilité.

Le document couvre au travers de ses différentes sections les aspects suivants :

- Section 2 : définitions des deux courbes supportées dans le référentiel (`secp256r1` et `brainpoolP256r1`) pour une utilisation pour la signature (ECDSA et ECSDSA) et l'échange de clé (ECDH) dans IKEv2.
- Section 3 : définition des algorithmes de signature sur courbes utilisés dans IKEv2 (ECDSA et ECSDSA).
- Section 4 : définition de l'algorithme d'échange de clé sur courbes (ECDH) utilisés dans IKEv2.
- Section 5 : Génération de secrets et authentification symétrique. L'objet de cette section est principalement la fonction pseudo-aléatoire (PRF).
- Section 6 : Algorithmes de chiffrement intègre pour IKEv2 et ESP. Cette section couvre donc à la fois les modes combinés (AES-GCM) mais également les constructions basées sur un algorithme de chiffrement (et son mode) et un algorithme d'intégrité. Une des préoccupations importantes de cette section concerne la génération d'IV.
- Section 7 : remarques diverses.

Lorsque ceci est possible le document renvoie vers les documents applicables existants plutôt que de paraphraser ceux-ci. Lorsque ces références nécessitent ou peuvent bénéficier de clarifications, celles-ci sont données.

2 Courbes elliptiques

Les 2 courbes elliptiques dont le support est imposé par le référentiel sont décrites dans cette section. Celles-ci sont définies sur un corps de base $\mathbb{F}_p$ avec p premier. Les notations suivantes sont utilisées pour décrire pour chaque courbe ses paramètres :

- nom : le nom de la courbe, sous forme de chaine ASCII.
- OID : l'identifiant d'objet de la courbe.
- p : le nombre premier spécifiant le corps de base, en notation hexadécimale.
- A et B : les coefficients, en notation hexadécimale, de l'équation de Weierstrass sous sa forme réduite $y^2 = x^3 + Ax + B \mod p$.
- $G = (G_x, G_y)$: le point de base de la courbe, d'ordre premier. G_x et G_y sont respectivement son abscisse et son ordonnée en représentation affine. Celles-ci sont données en notation hexadécimale.
- q : l'ordre premier du groupe engendré par G, en notation hexadécimale.
- i : le cofacteur [1] de q, en notation hexadécimale.

Le reste de la section, au travers de ses sous-sections, donne dans le formalisme précédent les paramétrages des 2 courbes dont le support est imposé par le référentiel.

2.1 `BrainpoolP256r1`

nom :	`BrainpoolP256r1`
OID :	1.3.36.3.3.2.8.1.1.7
p :	`A9FB57DBA1EEA9BC3E660A909D838D72` `6E3BF623D52620282013481D1F6E5377`
A :	`7D5A0975FC2C3057EEF67530417AFFE7` `FB8055C126DC5C6CE94A4B44F330B5D9`
B :	`26DC5C6CE94A4B44F330B5D9BBD77CBF` `958416295CF7E1CE6BCCDC18FF8C07B6`
G_x :	`8BD2AEB9CB7E57CB2C4B482FFC81B7AF`

G_y :	`B9DE27E1E3BD23C23A4453BD9ACE3262`
	`547EF835C3DAC4FD97F8461A14611DC9`
	`C27745132DED8E545C1D54C72F046997`

1. $q * i$ donne le nombre de points de la courbe

q :	A9FB57DBA1EEA9BC3E660A909D838D71
	8C397AA3B561A6F7901E0E82974856A7
i :	1

Cette courbe `BrainpoolP256r1` est notamment définie en section 3.4 de [RFC5639] et 2.2 de [RFC6932].

2.2 `secp256r1`

nom :	`secp256r1` [2]
OID :	1.2.840.10045.3.1.7
p :	FFFFFFFF00000001000000000000000
	00000000FFFFFFFFFFFFFFFFFFFFFFFF
A :	FFFFFFFF00000001000000000000000
	00000000FFFFFFFFFFFFFFFFFFFFFFFC
B :	5AC635D8AA3A93E7B3EBBD55769886BC
	651D06B0CC53B0F63BCE3C3E27D2604B
G_x :	6B17D1F2E12C4247F8BCE6E563A440F2
	77037D812DEB33A0F4A13945D898C296
G_y :	4FE342E2FE1A7F9B8EE7EB4A7C0F9E16
	2BCE33576B315ECECBB6406837BF51F5
q :	FFFFFFFF00000000FFFFFFFFFFFFFFFF
	BCE6FAADA7179E84F3B9CAC2FC632551
i :	1

Cette courbe `secp256r1` est également décrite en section 3.1 de [RFC5903]. La section 5 de ce même document décrit l'apparition de cette courbe dans d'autres standards principaux (NIST, ISO/IEC, ANSI, SECG).

2. Le nom P-256 est également souvent utilisé.

3 Algorithmes de signature sur courbe elliptique

Les différents algorithmes de signature sur courbes elliptiques suivants sont spécifiés dans le référentiel :

- ECSDSA sur `BrainpoolP256r1` avec SHA256,
- ECSDSA sur `secp256r1` avec SHA256,
- ECDSA sur `BrainpoolP256r1` avec SHA256,
- ECDSA sur `secp256r1` avec SHA256.

3.1 Documents de référence

Dans le cadre de la normalisation réalisée par l'IETF pour IKEv2 et IPsec, [RFC4754] définit l'authentification par ECDSA pour IKEv2. En pratique, le document se limite aux instanciations spécifiques suivantes, **dont seule la première est reprise dans le référentiel** :

- **ECDSA-256 : ECDSA sur `secp256r1` avec SHA256**,
- ECDSA-384 : ECDSA sur `secp384r1` avec SHA384,
- ECDSA-512 : ECDSA sur `secp521r1` avec SHA512.

Il est donc nécessaire dans le cadre du référentiel d'étendre ce travail de normalisation pour permettre l'utilisation d'ECDSA sur la courbe `BrainpoolP256r1`. Cet effort doit également être étendu pour le support d'ECSDSA sur `BrainpoolP256r1` et `secp256r1`.

Concernant la définition des mécanismes de signature, le standard associé à ECDSA ([X9.62:2005]) ne définit que cet algorithme spécifique. Un autre standard plus récent ([ISO14888-3:2016]) a l'avantage de recenser et de spécifier différents algorithmes de signature, dont ECDSA [3] et ECSDSA. Il sert donc de base commune dans le référentiel pour la définition des deux mécanismes de signature ECDSA et ECSDSA.

L'algorithme de hachage utilisé (SHA256) pour les quatre algorithmes de signature sélectionnés dans le référentiel est défini dans [FIPS180-4].

Les deux courbes utilisées par les instances supportées par le référentiel sont décrites à la section 2.

3. Les deux documents [X9.62:2005] et [ISO14888-3:2016] décrivent ECDSA comme un même mécanisme, associé aux mêmes vecteurs de test.

3.2 Identifiants pour IKEv2

Pour être utilisable dans IKEv2, chacune des quatre instances d'algorithme de signature doit disposer d'un identifiant de payload AUTH{} (*Authentication Method*). Seule *"ECDSA with SHA256 on the P-256 curve"* dispose déjà d'un tel identifiant via [RFC4754]. Pour les trois autres, un numéro de méthode d'authentification a donc été sélectionné dans la plage 201-255 (*Private Use*) :

- pour `ECDSA` :
  - 9 : `ECDSA` sur `secp256r1` avec `SHA256`
  - 214 : `ECDSA` sur `BrainpoolP256r1` avec `SHA256`
- pour `ECSDSA` :
  - 225 : `ECSDSA` sur `secp256r1` avec `SHA256`
  - 228 : `ECSDSA` sur `BrainpoolP256r1` avec `SHA256`

Seuls ces identifiants sont supportés et doivent être implémentés dans le cadre du référentiel.

Vérification №1

Seules les quatre méthodes d'authentification définies dans le référentiel (associées aux identifiants de payload AUTH{} 9, 214, 225 et 228) sont proposées et acceptées.

3.3 ECSDSA

Le mécanisme de signature ECSDSA est défini dans [ISO14888-3:2016] au travers de trois primitives principales : génération de clé, signature et vérification. Ce document de référence décrit une version standard et une version optimisée de l'algorithme. La version utilisée dans le cadre du référentiel est la version standard.

Les primitives de signature et de vérification ECSDSA sont paramétrées par un algorithme de hachage et une courbe (associée à la clé utilisée).

De manière à discuter les détails d'implémentations, ces trois primitives sont données ci-dessous dans un formalisme simple mais compatible du document de référence (dans le cadre d'une utilisation avec les combinaisons de courbes et algorithmes de hachage utilisés dans le référentiel).

3.3.1 Notations

Les notations suivantes sont utilisées dans les définitions des algorithmes de génération, signature et vérification. Elles viennent en complément des notations données dans la section 2 pour la définition des courbes (e.g. p, q, G) :

- x et Y représentent respectivement la clé privée (un élément dans $]0, q[$) et la clé publique associée (un point sur la courbe considérée).
- H est la fonction de hachage dont les condensats produits ont une taille de γ bits. Dans le cadre du référentiel, γ est fixé à 256, la taille de sortie de SHA256. Les condensats produits peuvent donc être considérés au choix comme des chaines d'octets *(Octet String)* ou chaines de bits *(Bit String)*.
- $|r|$ désigne la taille en nombre de bits de la chaine r.
- $FE2OS(x)$ convertit l'élément $x \in \mathbb{F}_p$ en une chaine d'octet de longueur $\lceil p/8 \rceil$, en notation *big endian*. Dans le cadre du référentiel, les deux courbes supportées sont telles que $\lceil p/8 \rceil$ vaut 32.
- $OS2I(r)$ convertit la chaine d'octet r (en notation *big endian*) en entier.

3.3.2 Génération

La génération d'un couple de clé privée et clé publique ECSDSA (x, Y) est réalisée de la manière suivante :

1. Tirer x aléatoirement dans $]0, q[$ (voir 7.1)
2. Calculer $Y = xG$

Les problématiques liées à la génération d'aléa son détaillées en sous-section 7.1.

3.3.3 Signature

La génération d'une signature ECSDSA (r, s) pour le message M avec la clé privée x est réalisée de la manière suivante. Celle-ci présuppose la connaissance de l'algorithme de hachage utilisé (SHA256 dans le cadre du référentiel) et des paramètres de la courbe spécifique utilisée (`BrainpoolP256r1` ou `secp256r1`) :

1. Tirer k aléatoirement dans $]0, q[$ (voir 7.1)
2. Calculer $W = (W_x, W_y) = kG$
3. Calculer $r = H(W||M) = H(FE2OS(W_x)||FE2OS(W_y)||M)$

4. Calculer $e = OS2I(r) \mod q$
5. Si $e == 0$, redémarrer à l'étape 1.
6. Calculer $s = (k + ex) \mod q$
7. Si $s == 0$, redémarrer à l'étape 1.
8. Retourner (r, s)

Vérification №2

En plus de satisfaire les vecteurs de tests donnés en section 3.5, l'implémentation de l'algorithme de signature ECSDSA devra inclure les vérifications complémentaires des étapes 5 et 7 de l'algorithme présenté en section 3.3.3.

Les problématiques de génération d'aléa son traitées en sous-section 7.1. Les aspects liés à l'encodage de la signature dans le cadre de son utilisation dans IKEv2 sont traités en section 3.5. En effet, l'algorithme précédent produit un couple (r, s) composé d'une chaine d'octets et d'un entier.

3.3.4 Vérification

La procédure de vérification de signature ECSDSA (r, s) pour le message M avec la clé publique Y est réalisée de la manière suivante. Celle-ci présuppose la connaissance de l'algorithme de hachage utilisé (SHA256 dans le cadre du référentiel) et des paramètres de la courbe spécifique utilisée (`BrainpoolP256r1` ou `secp256r1`) :

1. Si $|r| \neq \gamma$, rejeter la signature.
2. Si $s \notin]0, q[$, rejeter la signature.
3. Calculer $e = OS2I(r) \mod q$
4. Si $e == 0$, rejeter la signature.
5. Calculer $W' = (W'_x, W'_y) = sG - eY$
6. Calculer $r' = H(W'||M) = H(FE2OS(W'_x)||FE2OS(W'_y)||M)$
7. Si $r == r'$, accepter la signature, sinon la rejeter.

Comme évoqué précédemment les aspects liés au décodage de la signature dans le cadre de son utilisation dans IKEv2 sont traités en section 3.5. En effet, l'algorithme précédent prend en entrée un couple (r, s) composé d'une chaine d'octets et d'un entier.

3.4 ECDSA

Le mécanisme de signature ECDSA est défini initialement dans [X9.62:2005] et repris dans [ISO14888-3:2016] de manière équivalente au travers de trois primitives principales : génération de clé, signature et vérification.

Les primitives de signature et de vérification ECDSA sont paramétrées par un algorithme de hachage et une courbe (associée à la clé utilisée).

De manière à discuter les détails d'implémentation, ces trois primitives sont données ci-dessous dans un formalisme simple mais compatible des documents de référence (dans le cadre d'une utilisation avec les combinaisons de courbes et algorithmes de hachage utilisés dans le référentiel).

3.4.1 Notations

Les notations utilisées sont celles données précédemment en sous-section 3.3.1.

3.4.2 Génération

La génération d'un couple de clé privée et clé publique ECDSA (x, Y) est réalisée de la manière suivante :

1. Tirer x aléatoirement dans $]0, q[$ (voir 7.1)
2. Calculer $Y = xG$

Les problématiques de génération d'aléa sont traitées en sous-section 7.1.

3.4.3 Signature

La génération d'une signature ECDSA (r, s) pour le message M avec la clé privée x est réalisée de la manière suivante. Celle-ci présuppose la connaissance de l'algorithme de hachage utilisé (SHA256 dans le cadre du référentiel) et des paramètres de la courbe spécifique utilisée (`BrainpoolP256r1` ou `secp256r1`). Le mécanisme de signature ECDSA est spécifié ci-dessous :

1. Tirer k aléatoirement dans $]0, q[$ (voir 7.1)
2. Calculer $W = (W_x, W_y) = kG$

3. Calculer $r = W_x \mod q$
4. Si $r == 0$, redémarrer à l'étape 1.
5. Calculer $h = H(m)$
6. Calculer $e = OS2I(h) \mod q$
7. Si $e == rx \mod q$, redémarrer à l'étape 1.
8. Calculer $s = k^{-1}(e + xr) \mod q$
9. Si $s == 0$, redémarrer à l'étape 1.
10. Retourner (r, s)

Vérification №3

En plus de satisfaire les vecteurs de tests donnés en section 3.5, l'implémentation de l'algorithme de signature ECDSA devra inclure les vérifications complémentaires des étapes 4, 7 et 9 de l'algorithme présenté en section 3.4.3.

Dans le cadre du référentiel, l'algorithme décrit ci-dessus est compatible avec celui décrit dans [ISO14888-3:2016] et [X9.62:2005]. Il est donc possible d'implémenter ECDSA sur la base d'un de ces deux documents de référence en ajoutant les étapes supplémentaires imposées décrites ci-dessus. En pratique, la description ci-dessus diffère sur les deux points suivants de l'algorithme (générique) décrit dans ces deux documents de référence :

1. Les deux documents de référence contiennent des étapes supplémentaires de traitement dans le cas où la taille de sortie de la fonction de hachage γ est supérieure à la taille de l'ordre de la courbe β (noté q dans ce document). Dans le cadre du référentiel, l'ensemble des instanciations d'ECDSA sont réalisées sur des couples (courbes, algorithme de hachage) tels que $\gamma \leq \beta$. Les étapes supplémentaires associées aux cas $\gamma > \beta$ ont donc été omises ci-dessus.
2. Les étapes 7 et 9 dans l'algorithme ci-dessus ne sont pas présentes dans les documents de référence mais n'affectent pas le mécanisme de signature. Leurs raisons d'être sont décrites dans [MBAK].

Les problématiques de génération d'aléa son traitées en sous-section 7.1. Les aspects liés à l'encodage de la signature dans le cadre de son utilisation dans IKEv2 sont traités en section 3.5. En effet, l'algorithme précédent produit un couple (r, s) composé de deux entiers.

3.4.4 Vérification

Le mécanisme de vérification de signature ECDSA est spécifié ci-dessous :

1. Si $r \notin]0, q[$ ou $s \notin]0, q[$, rejeter la signature
2. Calculer $h = H(m)$
3. Calculer $e = OS2I(h) \mod q$
4. Calculer $u = s^{-1}e \mod q$
5. Calculer $v = s^{-1}r \mod q$
6. Calculer $W' = (W'_x, W'_y) = uG + vY$
7. Si W' est le point à l'infini, rejeter la signature
8. Calculer $r' = W'_x \mod q$
9. Si $r == r'$, accepter la signature, sinon la rejeter.

Comme évoqué en fin de section 3.4.3, les étapes associées aux cas $\gamma > \beta$ ont été omises pour plus de lisibilité puisqu'elles ne sont jamais utilisées dans le cadre des instanciations de l'algorithme mises en œuvre dans le référentiel.

Comme évoqué précédemment, les aspects liés au décodage de la signature dans le cadre de son utilisation dans IKEv2 sont traités en section 3.5. En effet, l'algorithme précédent prend en entrée un couple (r, s) composé de deux entiers.

3.5 Vecteurs de test

Cette sous-section donne pour chacune des instances d'algorithme de signature supportées un vecteur de test permettant de recaler (pour une clé privée x donnée, une valeur éphémère k donnée et un message donné) les différentes étapes de calcul liées à la génération de clé, la signature et la vérification.

Les notations associées aux différentes valeurs intermédiaires sont celles des sections précédentes décrivant les différents algorithmes de signature.

Chaque vecteur de test est complété du contenu attendu pour un payload d'authentification associé à l'instance spécifique du vecteur de test (i.e. si les données à signer dans le cadre de l'échange étaient le message m donné dans le vecteur de test). Ceci permet de faire le lien entre la sortie haut niveau des algorithmes de signature (un couple (r, s) sans encodage précis) et le format précis à produire par les implémentations. En pratique, ceci revient à donner pour chaque instance l'encodage en octets des sorties r et s.

Pour rappel, le format du payload AUTH{} est le suivant.

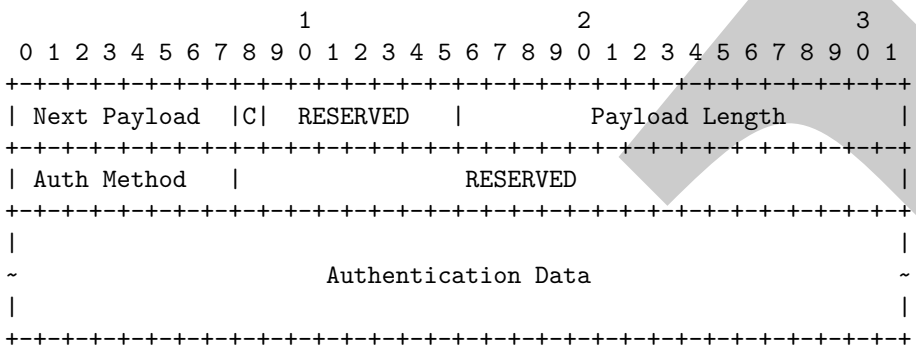


FIGURE 1 – Authentication Payload [RFC7296] Section 3.8

Les exemples de payload AUTH{} donnés ci-dessous le sont avec les champs `Next Payload`, `C` et `RESERVED` à 0. Il est important de préciser ici que les exemples donnés dans [RFC4754] pour ECDSA sur les courbes `secp256r1`, `secp384r1` et `secp521r1`, sont faux : la position du champs `Auth Method` est invalide (décalage de un octet vers la droite). Des versions corrigées sont données ici.

3.5.1 ECSDSA avec SHA256 sur `BrainpoolP256r1`

x :	A9357133 4AC32B50 268DDCA0 9523893A
	8F2989A9 4F9F44A9 1B7743F7 E145AEB7
Y_x :	A8016E47 23C89C6F D6E4A1E2 F3B467B1
	F54C4506 28361BDD C2C5F04D 5542515F
Y_y :	291C8A6A F7A72BA8 A4242631 1E178521
	CA84C760 06BE42C7 CCCE870D AC851243
k :	29A5C264 BA76379D 86498A64 16FC7FBA
	9D4F6275 64C698AB 4D95D190 6C8C61E4
W_x :	77D4F966 1DBC607B 159F23E6 BBD54A38
	BBE2D63D 3B36833F 316E195A E132D8AE
W_y :	34A3B4EB 74E1FFAD 0E35DD52 52EB60E5
	7410E894 EEF1DF3E 2EEE5F96 0834C71A
m :	616263
r :	0E7AF50B F4E08BF8 51004424 EE9D6502
	FCD1164E E3D99A00 A84FD5DB 814800EB

s :	647A24E6 07B6FC09 D88B1B57 2CFC4CE2
	9E25FAE1 431F0DFA 586BD16D EEFF92B6
t :	9B8062CF AD0E1DC3 ED65C66B AEE6286E
	8F686454 D1880CF6 E7CE38A7 160055BC
W'_x :	77D4F966 1DBC607B 159F23E6 BBD54A38
	BBE2D63D 3B36833F 316E195A E132D8AE
W'_y :	34A3B4EB 74E1FFAD 0E35DD52 52EB60E5
	7410E894 EEF1DF3E 2EEE5F96 0834C71A
AUTH{}	00000048 E4000000 0E7AF50B F4E08BF8
	51004424 EE9D6502 FCD1164E E3D99A00
	A84FD5DB 814800EB 647A24E6 07B6FC09
	D88B1B57 2CFC4CE2 9E25FAE1 431F0DFA
	586BD16D EEFF92B6

3.5.2 ECSDSA avec SHA256 sur `secp256r1`

x :	5202A3D8 ACAF6909 D12C9A77 4CD886F9
	FBA61137 FFD3E8E7 6AED363F B47AC492
Y_x :	09B58B88 323C52D1 080AA525 C89E8E12
	C6F40FCB 014640FA 88081ED9 E9352DE7
Y_y :	5CCBBD18 95385162 38B0B0B2 8ACB5F0B
	5E27217C 3A987242 1219DE0A EEBF1080
k :	DE7E0E5E 663F2418 3414B7C7 2F24546B
	81E9E5F4 10BEBF26 F3CA5FA8 2F5192C8
W_x :	847CE3CD 474FEC19 722AA9BA 81AFBF34
	7EE2D70E D067413F 1F716783 27A758CA
W_y :	DBFAD4AF 8C1D93AB 9C16467E 96BD11B5
	33643AA6 63498D8F 95919C6C A1AD91FC
m :	616263
r :	5A79A0AA 9B241E38 1A594B22 0554D096
	A5F09FA6 28AD9A33 C3CE4393 ADE1DEF7
s :	5C0EB78B 67A513C3 E53B2619 F96855E2
	91D5141C 7CD0915E 1D04B347 457C9601
t :	A5865F54 64DBE1C8 E5A6B4DD FAAB2F69
	16F65B07 7E6A0451 2FEB872F 4E81465A
W'_x :	847CE3CD 474FEC19 722AA9BA 81AFBF34
	7EE2D70E D067413F 1F716783 27A758CA
W'_y :	DBFAD4AF 8C1D93AB 9C16467E 96BD11B5

	33643AA6 63498D8F 95919C6C A1AD91FC
AUTH{}	00000048 E1000000 5A79A0AA 9B241E38
	1A594B22 0554D096 A5F09FA6 28AD9A33
	C3CE4393 ADE1DEF7 5C0EB78B 67A513C3
	E53B2619 F96855E2 91D5141C 7CD0915E
	1D04B347 457C9601

3.5.3 ECDSA avec SHA256 sur BrainpoolP256r1

x :	0051D386 6A15BACD E33D96F9 92FCA99D
	A7E6EF09 34E70975 59C27F16 14C88A7F
Y_x :	8ECB57AA E85AEF65 4714190B 8BE11E28
	90863E2E 286B6AEC 37506BDB 67BDDD25
Y_y :	0E4ED4D8 28A303B0 FFFA35F8 E1A98707
	CC0A28AA 83299509 A516E61D 5BC3D4E4
k :	9E56F509 196784D9 63D1C0A4 01510EE7
	ADA3DCC5 DEE04B15 4BF61AF1 D5A6DECE
W_x :	A3FA539A C2CFFBD5 C5ADB664 8CB3B5E3
	6A087DCC D5DAAE8A 0587AC37 887879B5
W_y :	073AD166 5BD32E91 12257A3F 79778BFC
	9F27DFDF 22195E59 E6A11505 3AEE2E19
m :	616263
h :	BA7816BF 8F01CFEA 414140DE 5DAE2223
	B00361A3 96177A9C B410FF61 F20015AD
$kinv$:	458D78C8 4535C431 5BED31D1 2F00FA1B
	9247FE9F B0C214C6 33033649 51B71E87
r :	A3FA539A C2CFFBD5 C5ADB664 8CB3B5E3
	6A087DCC D5DAAE8A 0587AC37 887879B5
s :	A7FF72A9 D85C6EDD 48562E8C D8F76DAB
	E3DBC396 0569DF5D 13F9835C F4CA723B
$sinv$:	5CEC40F9 E993A512 4D14B4F1 2361017D
	F28EE865 14E908DD 1C2BE914 77219510
u :	7DFC29A7 0D74CB3D 07B95B3B 7E20F95C
	45C7D112 B3A629CE 8B989F8F 0A3C0A8C
v :	56B7F800 B076DE92 C08F5E66 0DD2D810
	724E58AA 055289FE 92894A3C 4C5CC7B8
W'_x :	A3FA539A C2CFFBD5 C5ADB664 8CB3B5E3
	6A087DCC D5DAAE8A 0587AC37 887879B5
W'_y :	073AD166 5BD32E91 12257A3F 79778BFC

	9F27DFDF	22195E59	E6A11505	3AEE2E19
AUTH{}	00000048	D6000000	A3FA539A	C2CFFBD5
	C5ADB664	8CB3B5E3	6A087DCC	D5DAAE8A
	0587AC37	887879B5	A7FF72A9	D85C6EDD
	48562E8C	D8F76DAB	E3DBC396	0569DF5D
	13F9835C	F4CA723B		

3.5.4 ECDSA avec SHA256 sur secp256r1

x :	DC51D386	6A15BACD	E33D96F9	92FCA99D
	A7E6EF09	34E70975	59C27F16	14C88A7F
Y_x :	2442A5CC	0ECD015F	A3CA31DC	8E2BBC70
	BF42D60C	BCA20085	E0822CB0	4235E970
Y_y :	6FC98BD7	E50211A4	A27102FA	3549DF79
	EBCB4BF2	46B80945	CDDFE7D5	09BBFD7D
k :	9E56F509	196784D9	63D1C0A4	01510EE7
	ADA3DCC5	DEE04B15	4BF61AF1	D5A6DECE
W_x :	CB28E099	9B9C7715	FD0A80D8	E47A7707
	9716CBBF	917DD72E	97566EA1	C066957C
W_y :	2B57C023	5FB74897	68D058FF	4911C20F
	DBE71E36	99D91339	AFBB903E	E17255DC
m :	616263			
h :	BA7816BF	8F01CFEA	414140DE	5DAE2223
	B00361A3	96177A9C	B410FF61	F20015AD
$kinv$:	AFA27894	5AF74B1E	295008E0	3A8984E2
	E1C69D9B	BBC74AF1	4E3AC4E4	21ABFA61
r :	CB28E099	9B9C7715	FD0A80D8	E47A7707
	9716CBBF	917DD72E	97566EA1	C066957C
s :	86FA3BB4	E26CAD5B	F90B7F81	899256CE
	7594BB1E	A0C89212	748BFF3B	3D5B0315
$sinv$:	33BDC294	E90CFAD6	2A9F2FD1	F8741DA7
	7C02A573	E1B53BA1	7A60BA90	4F491952
u :	C3875E57	C85038A0	D60370A8	7505200D
	C8317C8C	534948BE	A6559C7C	18E6D4CE
v :	3B4E49C4	FDBFC006	FF993C81	A50EAE22
	1149076D	6EC09DDD	9FB3B787	F85B6483
W'_x :	CB28E099	9B9C7715	FD0A80D8	E47A7707
	9716CBBF	917DD72E	97566EA1	C066957C
W'_y :	2B57C023	5FB74897	68D058FF	4911C20F

	DBE71E36	99D91339	AFBB903E	E17255DC
AUTH{}	00000048	09000000	CB28E099	9B9C7715
	FD0A80D8	E47A7707	9716CBBF	917DD72E
	97566EA1	C066957C	86FA3BB4	E26CAD5B
	F90B7F81	899256CE	7594BB1E	A0C89212
	748BFF3B	3D5B0315		

Vérification №4

L'implémentation des algorithmes de signature sur courbe vérifie les vecteurs de test.

4 Algorithmes d'échange de clé sur courbe elliptique

4.1 Contexte

Dans IKEv2 ([RFC7296]), la mise en place d'un secret partagé est réalisée via l'échange de payload KE{} transportant les valeurs publiques de chacun des pairs :

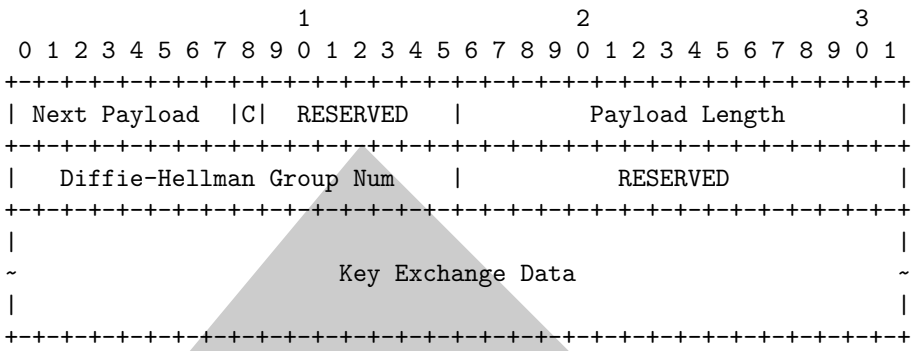


FIGURE 2 – Key Exchange Payload [RFC7296] Section 3.4

Dans le cadre du référentiel, un échange de payload KE{} est réalisé lors de la montée de la IKE_SA mais est également imposé lors de chaque montée de SA IPsec.

En pratique, [RFC7296] ne supporte nativement que des groupes DH classiques (les MODP, définis sur $\mathbb{Z}_p^*$, qui ne sont pas autorisés dans le cadre du référentiel) mais aucun groupe sur courbe elliptique.

[RFC5903] corrige en partie ce manque en définissant un ensemble de groupe DH sur courbes et leur intégration dans IKEv2 :

- Les seules courbes supportées sont `secp256r1`, `secp384r1` et `secp521r1`.
- Un numéro de groupe spécifique leur est associé (respectivement 19, 20 et 21).
- Un encodage du contenu du payload KE{}, notamment le champ `Key Exchange Data`, correspondant à la concaténation des coordonnées x et y (encodées sur un nombre entier d'octets) de la valeur publique.
- La manière de calculer le secret partagé sur la base de l'échange ECDH associé aux payload KE{}, la coordonnée x de la valeur ECDH com-

mune.
– Des vecteurs de test.

Il convient de préciser ici un point historique notamment repris dans la [RFC5903]. Lors d'une première définition par l'IETF de groupes ECDH pour IKEv2 via [RFC4753], le secret partagé était défini comme la concaténation des coordonnées x et y de la valeur ECDH commune. Un erratum ([Err9]) fut ensuite publié pour ne considérer que la coordonnée x comme le secret partagé produit. Cette modification est alignée avec ce que décrit [RFC5903]. En pratique, certaines implémentations basées sur la seule [RFC4753] ne sont donc pas interopérables [4].

Cette situation a évolué avec la publication de [RFC6954] [5] qui rend les courbes Brainpool utilisables pour l'échange de secret dans IKEv2. Au final, parmi les groupes intégrés au référentiel, les documents de référence de l'IETF couvrent toutes les courbes utiles.

Les numéros des groupes supportés par ce référentiel sont enregistrés auprès de l'IANA dans [IKEv2-PARAMS].

On notera par ailleurs que [RFC6989] liste un ensemble de tests additionnels à intégrer lors de l'implémentation de DH ou ECDH. Ceux-ci sont intégrés ci-dessous.

Le reste de cette section reprend l'ensemble des détails d'implémentation pour le support d'ECDH dans IKEv2 en utilisant les différentes courbes supportées par le référentiel. Les différents aspects couverts sont les suivants :
– Utilisation des payload KE{} dans IKEv2
– Génération des payload (calcul de la valeur publique, encodage)
– Traitement des payload KE{} reçus (décodage correct, vérification de la valeur publique reçue, etc)
– Calcul du secret partagé
– Note concernant la réutilisation et l'effacement.

4.2 Utilisation des payload KE{}

Dans le cadre du référentiel, pour bénéficier de secrets frais à chaque étape importante, les modifications suivantes sont opérées :
– La mise en place d'une IKE_SA débute par l'échange de messages `IKE_SA_INIT` comme le décrit la section 1.2 de [RFC7296]. Ceux-ci doivent inclure chacun un payload KE{} comme l'impose [RFC7296] .

4. On peut actuellement citer le cas d'OpenIKED.
5. [RFC6932] n'apporte rien à la discussion.

Dans le cadre du référentiel, les groupes DH à supporter sont imposés (19 et 28).

- Dans le cadre de la mise en place d'une IKE_SA, un premier couple de SA IPsec est normalement mis en place entre les deux pairs dès l'échange de messages `IKE_AUTH`. **Dans le cadre du référentiel, la montée de la IKE_SA est réalisée conformément à [RFC6023]**, i.e. sans résulter en une mise en oeuvre de SA IPsec (i.e. Childless IKE SA). C'est à dire que les payload SA, TS et notifications relatives à l'établissement de SA IPsec sont absents de l'échange `IKE_AUTH`. Ainsi, le matériel de clé du premier couple de SA IPsec n'est pas directement déduit du matériel de clé généré pour la IKE_SA.
- [RFC7296] permet de manière optionnelle l'échange de payload KE lors des `CREATE_CHILD_SA`. Dans le cadre du référentiel, la présence de ces payload KE{} est rendue obligatoire pour bénéficier d'un matériel de clé frais pour chaque couple de SA IPsec (ceci incluant le premier couple de SA évoqué au point précédent). Les groupes DH imposés sont les mêmes que pour l'échange `IKE_SA_INIT`.

Vérification №5

Il doit être vérifié que l'implémentation IKEv2 ne supporte ni en tant qu'initiator, ni en tant que responder la négociation de SA IPsec lors de la montée de IKE_SA, i.e. que les montées de IKE_SA sont *childless*.

4.3 Génération des payload

Dans le cadre du référentiel, la génération d'un payload KE{} pour un groupe (i.e. une courbe elliptique parmi celles supportées) est réalisée en calculant une valeur publique puis en encodant celle-ci pour envoi au travers d'un payload KE{}. Les étapes sont les suivantes :

- Tirer aléatoirement k, la valeur privée ECDH, dans $]0, q[$ (voir 7.1)
- Calculer $Y = kG$, la valeur publique
- Remplir les champs du payload KE{} (cf Figure 2) :
  - `Diffie-Hellman Group Num` : la valeur est celle du groupe sélectionné parmi ceux autorisés par le référentiel :

Courbe	*Numéro de groupe DH*
`secp256r1`	19
`BrainpoolP256r1`	28

– `Key Exchange Data` : ce champ est rempli avec la valeur publique Y produite précédemment. L'encodage utilisé, applicable aux 2 groupes supportés, est le suivant : les coordonnées x et y ($\in \mathbb{F}_p$) du point Y sont chacune exportées sur un nombre d'octets entiers ($\lceil p/8 \rceil$) et concaténées l'une à la suite de l'autre dans cet ordre. Le tableau ci-dessous récapitule les longueurs associées aux différents groupes :

Courbe	*Num grp*	|p| *(bits)*	*Longueur du champ Key Exchange Data* *(bits)*	*(octets)*
`secp256r1`	19	256	512 (2×256)	64
`BrainpoolP256r1`	28	256	512 (2×256)	64

– `Payload Length` : la valeur du champ `Payload Length` se déduit directement de la taille en octets du champ `Key Exchange Data` donné en dernière colonne du tableau précédent en lui ajoutant 8.

Courbe	*Num grp*	*Valeur du champ Payload Length*
`secp256r1`	19	72 ($64 + 8$)
`BrainpoolP256r1`	28	72 ($64 + 8$)

Vérification №6

Les payload KE{} produits doivent uniquement présenter une valeur dans le champ `Diffie-Hellman Group Num` égale à 19 ou 28 (associée à l'une des deux courbes supportées par le référentiel).

4.4 Traitement des payload KE{} reçus

Le traitement des payload KE{} reçus dans le cadre des différents échanges (`IKE_SA_INIT` puis `CREATE_CHILD_SA` suivants) est réalisé conformément aux descriptions de [RFC7296] complétées par les tests additionnels décrits dans [RFC6989].

Après les vérifications sur le format du payload (champs de longueur en adéquation[6] avec le contenu du champ `DH Group Num` et la longueur réelle restante dans le paquet en cours de parsing), le contenu du champ `Key Exchange Data` est validé via les tests suivants (en partie repris de la section 2.3 de [RFC6989]) :

- vérifier que les coordonnées x et y importées pour la valeur publique reçue Y sont bien strictement inférieures à p.
- vérifier que Y n'est pas le point à l'infini. En pratique, comme évoqué en fin de section 2.3 de [RFC6989], l'encodage du champ `Key Exchange Data` ne permet pas d'y passer le point à l'infini.
- vérifier que Y est bien sur la courbe en vérifiant que ses coordonnées x et y satisfont bien l'équation de la courbe ($y^2 = x^3 + Ax + B \mod p$)

Vérification №7

Le champ `Diffie-Hellman Group Num` doit contenir une valeur égale à 19 ou 28 (associé à l'une des deux courbes supportées par le référentiel).

4.5 Calcul du secret partagé

Comme décrit dans [RFC5903], le secret partagé est produit en calculant (à partir de la valeur privée k générée localement et la valeur Y reçue du pair) la valeur commune $Z = (Z_x, Z_y) = kY$ puis en conservant uniquement sa coordonnée Z_x ($\in \mathbb{F}_p$) encodée sous forme de chaine d'octets (de taille entière $\lceil p/8 \rceil$, i.e. 32).

4.6 Note concernant la réutilisation et l'effacement

Afin d'assurer la propriété de *Perfect Forward Secrecy* (PFS) souhaitée dans le cadre de ce référentiel, les valeurs privées utilisées lors de l'ECDH pour l'établissement de la IKE_SA et des SA IPSec ne doivent être utilisées qu'une seule fois. C'est à dire qu'elles doivent être effacées de la mémoire dès lors qu'un secret partagé a été généré.

Il est important de noter que l'effacement des clés éphémères et du secret partagé peut, lorsque le programme est compilé avec des optimisations, être rendu inopérant par le compilateur. Il est donc fortement recommandé d'avoir recours à une fonction d'effacement sécurisée.

6. voir les tableaux de la section précédente

Une paire de clés éphémères n'est pas réutilisée pour plusieurs échanges ni avec plusieurs pairs.

4.7 Vecteurs de test

Cette sous-section fournit des vecteurs de test pour chacune des courbes définies dans le référentiel sous la forme suivante :

- k_i : clé secrète (éphémère) de l'*initiator*
- $Y_i = (Y_{ix}, Y_{iy})$: clé publique (éphémère) de l'*initiator*
- KE_i : payload KE{} de l'*initiator* contenant la valeur publique ECDH. Ce payload inclut le header générique ; son champ `Next Payload` est fixé à 0 et sont bit critical également.
- k_r : clé secrète (éphémère) du *responder*
- $Y_r = (Y_{rx}, Y_{ry})$: clé publique (éphémère) du *responder*
- KE_r : payload KE{} du *responder* contenant la valeur publique ECDH. Ce payload inclut le header générique ; son champ `Next Payload` est fixé à 0 et sont bit critical également.
- $Z = (Z_x, Z_y) = kY$: la valeur commune ECDH
- $S = Z_x$: le secret partagé (l'abscisse de la valeur commune ECDH encodée sur $\lceil p/8 \rceil$ octets)

4.7.1 `secp256r1`

k_i :	C88F01F5 10D9AC3F 70A292DA A2316DE5
	44E9AAB8 AFE84049 C62A9C57 862D1433
Y_{ix} :	DAD0B653 94221CF9 B051E1FE CA5787D0
	98DFE637 FC90B9EF 945D0C37 72581180
Y_{iy} :	5271A046 1CDB8252 D61F1C45 6FA3E59A
	B1F45B33 ACCF5F58 389E0577 B8990BB3
KE_i :	00000048 00130000 DAD0B653 94221CF9
	B051E1FE CA5787D0 98DFE637 FC90B9EF
	945D0C37 72581180 5271A046 1CDB8252
	D61F1C45 6FA3E59A B1F45B33 ACCF5F58
	389E0577 B8990BB3
k_r :	C6EF9C5D 78AE012A 011164AC B397CE20
	88685D8F 06BF9BE0 B283AB46 476BEE53
Y_{rx} :	D12DFB52 89C8D4F8 1208B702 70398C34
	2296970A 0BCCB74C 736FC755 4494BF63

Y_{ry} :	56FBF3CA 366CC23E 8157854C 13C58D6A
	AC23F046 ADA30F83 53E74F33 039872AB
KE_r :	00000048 00130000 D12DFB52 89C8D4F8
	1208B702 70398C34 2296970A 0BCCB74C
	736FC755 4494BF63 56FBF3CA 366CC23E
	8157854C 13C58D6A AC23F046 ADA30F83
	53E74F33039872AB
Z_x :	D6840F6B 42F6EDAF D13116E0 E1256520
	2FEF8E9E CE7DCE03 812464D0 4B9442DE
Z_y :	522BDE0A F0D8585B 8DEF9C18 3B5AE38F
	50235206 A8674ECB 5D98EDB2 0EB153A2
S :	Z_x

Ce vecteur de test est aussi présent en section 8.1 de [RFC5903].

4.7.2 BrainpoolP256r1

k_i :	81DB1EE1 00150FF2 EA338D70 8271BE38
	300CB542 41D79950 F77B0630 39804F1D
Y_{ix} :	44106E91 3F92BC02 A1705D99 53A8414D
	B95E1AAA 49E81D9E 85F929A8 E3100BE5
Y_{iy} :	8AB4846F 11CACCB7 3CE49CBD D120F5A9
	00A69FD3 2C272223 F789EF10 EB089BDC
KE_i :	00000048 001C0000 44106E91 3F92BC02
	A1705D99 53A8414D B95E1AAA 49E81D9E
	85F929A8 E3100BE5 8AB4846F 11CACCB7
	3CE49CBD D120F5A9 00A69FD3 2C272223
	F789EF10 EB089BDC
k_r :	55E40BC4 1E37E3E2 AD25C3C6 654511FF
	A8474A91 A0032087 593852D3 E7D76BD3
Y_{rx} :	8D2D688C 6CF93E11 60AD04CC 4429117D
	C2C41825 E1E9FCA0 ADDD34E6 F1B39F7B
Y_{ry} :	990C5752 0812BE51 2641E470 34832106
	BC7D3E8D D0E4C7F1 136D7006 547CEC6A
KE_r :	00000048 001C0000 8D2D688C 6CF93E11
	60AD04CC 4429117D C2C41825 E1E9FCA0
	ADDD34E6 F1B39F7B 990C5752 0812BE51
	2641E470 34832106 BC7D3E8D D0E4C7F1

	136D7006	547CEC6A		
Z_x :	89AFC39D	41D3B327	814B8094	0B042590
	F96556EC	91E6AE79	39BCE31F	3A18BF2B
Z_y :	49C27868	F4ECA217	9BFD7D59	B1E3BF34
	C1DBDE61	AE129316	48F43E59	632504DE
S :	Z_x			

Ce vecteur de test est aussi présent en annexe A.2 de [RFC6954].

5 Génération de secrets et authentification symétrique

La fonction pseudo-aléatoire, ou PRF, négociée lors de la montée de la IKE_SA (ou d'une SA IPsec) a deux usages distincts :

- comme mécanisme de dérivation de clés lors des étapes de génération de secrets communs à l'*initiator* et au *responder*.
- comme mécanisme d'authentification symétrique dans le cadre de la montée de la IKE_SA.

Dans le cadre du référentiel, seule la fonction `PRF_HMAC_SHA2_256` définie pour IKEv2 dans [RFC4868] est autorisée avec l'identifiant suivant :

Fonction	***Num transform***
PRF_HMAC_SHA2_256	5

Les secrets de la IKE_SA et des SA IPsec sont dérivés via une construction dénommée PRF+ basée sur la fonction PRF. Elle retourne un aléa de la taille des secrets à générer (habituellement plus longs que le résultat d'un seul appel à la PRF). La PRF+ est définie dans la section 2.13 de [RFC7296].

5.1 Génération de secrets pour la IKE_SA

Les secrets de la IKE_SA sont dérivés tel que décrit dans la section 2.14 de [RFC7296] :

1. Calcul de `SKEYSEED` à partir du secret partagé ECDH (noté Z et décrit dans la section 4) et du contenu des payload NONCE{} (NONCE_i pour l'*initiator* et NONCE_r pour le *responder*). Les nonces mesurent la moitié de la taille du résultat de la PRF, soit 128 bits (16 octets) dans le cadre du référentiel.
2. Calcul des secrets SK à partir de `SKEYSEED`, des nonces et des SPI de la IKE_SA, pour produire les sous clés suivantes :
   - SK_d : clé de 256 bits (32 octets) utilisée pour dériver de nouvelles clés pendant la durée de vie de la IKE_SA.

- SK_{ai} : clé de protection en intégrité dans le sens *initiator* vers *responder*. Sa taille dépend de la suite de chiffrement intègre négociée (voir 6).
- SK_{ar} : clé de protection en intégrité dans le sens *responder* vers *initiator*. Sa taille dépend de la suite de chiffrement intègre négociée (voir 6).
- SK_{ei} : secret de 36 octets, concaténation d'une clé $ENCKEY_i$ de 32 octets et d'un sel $SALT_i$ de 4 octets, pour la protection en confidentialité (ou confidentialité et intégrité avec un mode combiné) dans le sens *initiator* vers *responder*.
- SK_{er} : secret de 36 octets, concaténation d'une clé $ENCKEY_r$ de 32 octets et d'un sel $SALT_r$ de 4 octets, pour la protection en confidentialité (ou confidentialité et intégrité avec un mode combiné) dans le sens *responder* vers *initiator*.
- SK_{pi} : clé de 256 bits (32 octets) utilisée pour la construction du payload AUTH{} de l'*initiator*.
- SK_{pr} : clé de 256 bits (32 octets) utilisée pour la construction du payload AUTH{} du *responder*.

Le mécanisme de génération est repris ci-dessous :

- `SKEYSEED` = PRF($NONCE_i$ || $NONCE_r$, Z)
- SK_d || SK_{ai} || SK_{ar} || SK_{ei} || SK_{er} || SK_{pi} || SK_{pr} = PRF+(`SKEYSEED`, $NONCE_i$ || $NONCE_r$ || SPI_i || SPI_r)
- SK_{ei} = $ENCKEY_i$ || $SALT_i$
- SK_{er} = $ENCKEY_r$ || $SALT_r$

5.2 Génération du matériel de clé pour SA IPsec

Seul l'échange de message `CREATE_CHILD_SA` avec génération d'un nouveau secret partagé engendre la création de SA IPsec. Cela signifie que la présence des *payload* KE{} et NONCE{} est obligatoire. Les nonces mesurent la moitié de la taille du résultat de la PRF, soit 128 bits (16 octets), dans le cadre du référentiel.

Le matériel de clé utilisé pour protéger les SA IPsec est dérivé de la façon suivante (décrit dans la section 2.17 de [RFC7296]) :

1. Calcul de `KEYMAT` à partir du secret SK_d de la IKE_SA, du secret partagé ECDH (noté Z et décrit dans la section 4) et des nonces

NONCE$_i$ et NONCE$_r$.

2. Découpage du `KEYMAT` pour avoir un matériel de clé par sens de communication. D'abord *initiator* vers *responder*, puis *responder* vers *initiator*.
3. Pour chaque sens, extraction des secrets KEYMAT$_i$ ou KEYMAT$_r$ pour les algorithmes de protection en confidentialité et en intégrité. Les algorithmes négociés pour la protection des SA IPsec déterminent la taille de chaque élément (voir 6).

Le mécanisme est repris ci-dessous :
- KEYMAT = PRF+(SK$_d$, Z || NONCE$_i$ || NONCE$_r$)
- KEYMAT$_i$ || KEYMAT$_r$ = KEYMAT
- ENCKEY$_i$ || SALT$_i$ || INTEGKEY$_i$ = KEYMAT$_i$
- ENCKEY$_r$ || SALT$_r$ || INTEGKEY$_r$ = KEYMAT$_r$

5.3 Authentification symétrique

Un secret partagé fort entre deux pairs peut être utilisé comme méthode d'authentification. La PRF est alors utilisée pour fournir un mécanisme d'authentification symétrique, comme spécifié à la section 2.15 de [RFC7296].

5.4 Vecteurs de test

Ces tests sont repris de la section 2.7.1 de [RFC4868].

5.4.1 `PRF_HMAC_SHA2_256` test 1

K :	`0B0B0B0B 0B0B0B0B 0B0B0B0B 0B0B0B0B 0B0B0B0B`
S :	`48692054 68657265`
PRF(K, S) :	`B0344C61 D8DB3853 5CA8AFCE AF0BF12B 881DC200 C9833DA7 26E9376C 2E32CFF7`

5.4.2 `PRF_HMAC_SHA2_256` test 6

K :	`AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA`

	AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAA
S :	54686973 20697320 61207465 73742075 73696E67 2061206C 61726765 72207468 616E2062 6C6F636B 2D73697A 65206B65 7920616E 64206120 6C617267 65722074 68616E20 626C6F63 6B2D7369 7A652064 6174612E 20546865 206B6579 206E6565 64732074 6F206265 20686173 68656420 6265666F 72652062 65696E67 20757365 64206279 20746865 20484D41 4320616C 676F7269 74686D2E
PRF(K, S) :	9B09FFA7 1B942FCB 27635FBC D5B0E944 BFDC6364 4F071393 8A7F5153 5C3A35E2

6 Algorithmes de chiffrement intègre

Les deux types de SA IKEv2 et IPsec sont concernés par cette section qui définit les algorithmes de chiffrement symétrique, les modes, et les contraintes de génération des IV.

Les deux suites de chiffrement intègre (C+I) suivantes basées sur AES [FIPS197] avec une taille de clé de 256 bits sont imposées par le référentiel :

- AES-GCM (clé de 256 bits) avec ICV à 16 octets
- `ENCR_AES_CTR` (clé de 256 bits) avec `AUTH_HMAC_SHA2_256_128` (clé de 256 bits)

Lors de la construction de payload SA{}, tel que décrit dans la section 3.3 de [RFC7296], les identifiants associés à ces deux suites sont les suivants :

Suite	***Num transform ENCR***	***Key bitlen***	***Num transform INTEG***
AES-GCM	20	256	*absent*
AES-CTR avec HMAC	13	256	5

Dans le cadre du référentiel, une *proposal* ne peut contenir qu'une *transform* de chaque type. De plus, la *proposal* AES-GCM ne doit pas contenir de *transform INTEG*, même si celle-ci prend la valeur *NONE* (0).

Les deux suites reposent, pour fournir la confidentialité des données, sur l'utilisation d'AES en mode compteur[7]. L'utilisation de ce mode pour le chiffrement permet de limiter le nombre d'octets additionnels dans les paquets produits pour la gestion du padding (1 pour IKEv2 et au maximum 4 pour ESP). Par ailleurs, le mécanisme de génération d'IV fonctionne sur une base commune ne nécessitant pas - comme précisé dans la suite du document - de recours à un générateur d'aléa par l'utilisation d'une logique incrémentale.

Les détails des modes sont donnés dans la suite du référentiel en se basant notamment sur les documents de référence. Il convient de préciser que ces documents ont tendance à utiliser des termes identiques (nonce, salt, block, block counter) pour désigner des éléments différents concernant la création des vecteurs d'initialisation (IV). De la même manière, les descriptions fournies par les différents documents incluent ou non le motif d'intégrité au chiffré.

7. GCM combine CTR pour la confidentialité et GHASH pour l'intégrité.

Dans le cadre du référentiel, les conventions suivantes sont utilisées par la suite. Tout payload ESP ou payload IKEv2 SK{} est constitué de 3 éléments :

- Un vecteur d'initialisation (IV) de 8 octets
- Un chiffré (Ciphertext)
- Un motif d'intégrité (ICV) de 16 octets

Pour IKEv2 et ESP, pour un sens de communication donné, l'IV sur 8 octets (dans les deux cas d'algorithmes de chiffrement du référentiel) est une valeur incrémentale en notation *big endian* mise à 1 pour le premier paquet. L'IV est un des constituant du Nonce utilisé par les modes CTR et GCM :

$$\text{Nonce} = \text{SALT} \,||\, \text{IV}$$

Pour les deux suites supportées par le référentiel, aussi bien pour la protection des paquets par ESP que pour la protection du trafic IKEv2, le SALT est une valeur de 4 octets.

La clé AES ENCKEY et le SALT sont fournis par les éléments SK_{ei}/SK_{er} de la IKE_SA, ou directement à partir de l'élément `KEYMAT` issu de l'échange `CREATE_CHILD_SA`, tel que décrit respectivement dans 5.1 et 5.2. Pour rappel, les éléments sont donc extraits dans cet ordre pour IKEv2 :

$$\begin{gathered}\text{ENCKEY}_i \text{ (32 octets)} \,||\, \text{SALT}_i \text{ (4 octets)} = \text{SK}_{ei} \\ \text{ENCKEY}_r \text{ (32 octets)} \,||\, \text{SALT}_r \text{ (4 octets)} = \text{SK}_{er} \\ \text{INTEGKEY}_i = \text{SK}_{ai} \\ \text{INTEGKEY}_r = \text{SK}_{ar}\end{gathered}$$

Et dans l'ordre suivant pour ESP :

$$\begin{gathered}KEYMAT_i = ENCKEY_i(32\ octets)||SALT_i(4\ octets)||INTEGKEY_i \\ KEYMAT_r = ENCKEY_r(32\ octets)||SALT_r(4\ octets)||INTEGKEY_r\end{gathered}$$

La taille de clé INTEGKEY dépend de la suite choisie, elle est nulle pour AES-GCM et de 32 octets (256 bits) pour AES-CTR avec `AUTH_HMAC_SHA2_256_128`.

Pour ce qui concerne l'ICV de 16 octets, celui-ci est constitué par le tag AES-GCM pour la première suite du référentiel ou par le HMAC tronqué pour la seconde suite. Dans les deux cas, les données sur lesquelles porte le calcul du motif d'intégrité sont les suivantes :

– ESP : SPI, ESN, IV, Ciphertext
– IKEv2 SK{} : du premier octet de header IKEv2 jusqu'à la fin du chiffré. Les 4 premiers octets à zéro ajoutés lors du transport sur UDP ne font pas parties du *header* IKEv2 et ne doivent pas être pris en compte dans le calcul de l'ICV. Dans le cadre du référentiel, le payload SK{}, lorsqu'il est présent, est le seul et unique payload du paquet (tous les autres étant protégés).

La spécification dans les standards et l'usage des deux suites dans le cadre d'IKEv2 et d'IPsec sont détaillées dans les sous-sections suivantes.

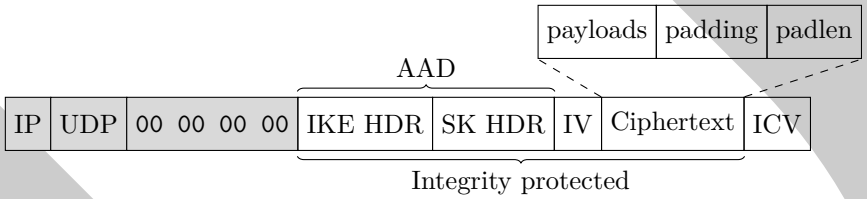
6.1 AES-GCM avec ICV à 16 octets

Les différents documents suivants sont les références principales à considérer :

– [SP800-38D] définit le mode GCM pour AES.
– [RFC5282] définit l'utilisation d'AEAD (dont notamment GCM) dans IKEv2.
– [RFC4106] définit l'utilisation du mode GCM dans ESP.

6.1.1 Utilisation dans IKEv2

[RFC5282] définit l'usage des AEAD dans IKEv2, mais se contente essentiellement d'exposer ses différences avec [RFC4106], qui définit l'usage des AEAD dans ESP. Un résumé de l'utilisation d'AES-GCM dans IKEv2 est présenté ici.



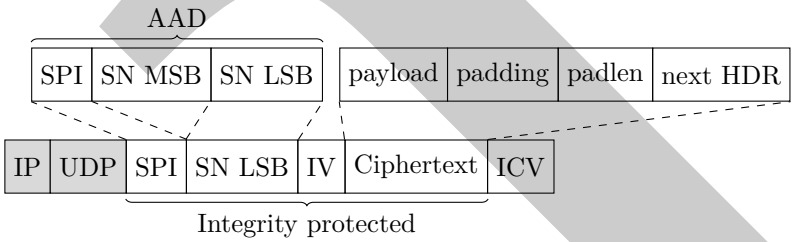
AES-GCM étant un mode combiné, les données chiffrées sont de base protégées en intégrité, le reste du paquet IKEv2 à protéger en intégrité seule est fourni en tant que données associées (AAD) à partir du *header* fixe IKEv2 (sans les 4 octets nuls lié à l'encapsulation UDP) jusqu'à la fin du *header* générique du payload SK{} tel que décrit dans la section 5.1 de [RFC5282].

Le premier message d'une session IKEv2 sera ainsi généré avec les paramètres suivants :

- IV = `00 00 00 00 00 00 00 01`
- Nonce = SALT || IV
- AAD = HeaderIKEv2 || HeaderPayloadSK
- Padlen = `00`
- Ciphertext, ICV = AES_GCM(ENCKEY, Nonce, payloads || Padlen, AAD)

6.1.2 Utilisation dans ESP

Le support de l'*Extended Sequence Number* (ESN) tel que décrit par [RFC4303] en section 2.2.1 est obligatoire dans ce référentiel. Il est par ailleurs indiqué que seule la partie basse des 64 bits du compteur représenté en *big endian* doit être transmise en ESP mais que la partie haute est tout de même prise en compte dans le calcul de l'ICV. Le schéma suivant résume l'utilisation d'AES-GCM avec ESP et ESN activé.



Les données associées (AAD) comprennent les champs SPI et ESN (complet, 64 bits) tel que décrit dans la section 5 de [RFC4106].

Le premier paquet transmis sur une SA IPsec sera ainsi généré avec les paramètres suivants :

- IV = `00 00 00 00 00 00 00 01`
- N = SALT || IV
- ESN = `00 00 00 00 00 00 00 01`
- AAD = `SPI` || ESN
- Ciphertext, ICV = AES_GCM(ENCKEY, Nonce, *payload* || *padding* || *padlen* || *nextHDR*, AAD)

6.2 AES-CTR avec `AUTH_HMAC_SHA2_256_128`

Les différents documents suivants sont les références principales à considérer :

- [SP800-38A] définit le mode CTR pour AES
- [RFC5930] définit l'utilisation d'AES en mode compteur (CTR) avec IKEv2
- [RFC3686] définit l'utilisation d'AES en mode compteur (CTR) avec ESP
- [RFC4868] définit l'utilisation de `HMAC-SHA-256-128` avec IPsec (ESP et IKEv2)

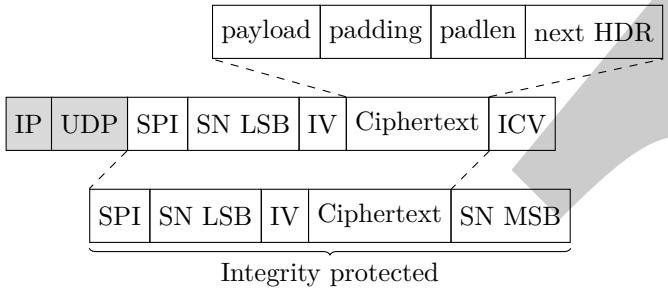
6.2.1 Utilisation dans IKEv2

Le premier message de la session IKEv2 sera généré avec les paramètres suivants :

- IV = `00 00 00 00 00 00 00 01`
- Nonce = SALT || IV
- Padlen = `00`
- Ciphertext = AES_CTR(ENCKEY, payloads || Padlen)
- Protected = HeaderIKEv2 || SK{}
- ICV = HMAC_SHA2(INTEGKEY, Protected)[0..15]

L'ICV correspond au 128 premiers bits (16 octets) de la chaine de bits produite par la fonction HMAC_SHA256.

6.2.2 Utilisation dans ESP



Lors de l'utilisation d'un mode séparant protection en confidentialité et en intégrité avec deux algorithmes (comme c'est le cas avec AES-CTR et HMAC) et que l'ESN est activé, [RFC4303] en section 3.3.2.1 indique que les 32 bits de poids fort sont inclus dans le calcul de l'ICV mais pas dans le paquet ESP final, comme présenté ci-dessus.

Le premier paquet transmis sur une SA IPsec sera généré avec les paramètres suivants :

- IV = `00 00 00 00 00 00 00 01`
- Nonce = SALT || IV
- Ciphertext = AES_CTR(ENCKEY, $payload$ || $padding$ || $padlen$ || $nextHDR$)
- Protected = SPI || lowSN || IV || Ciphertext || highSN
- ICV = HMAC_SHA2(INTEGKEY, Protected)[0..15]

L'ICV correspond au 128 premiers bits (16 octets) de la chaine de bits produite par la fonction HMAC_SHA256.

7 Remarques diverses

7.1 Génération d'aléa

Vérification №8

Les sources d'aléa utilisées doivent être conformes au RGS.

Dans le cadre du référentiel, les besoins de génération d'aléa sont exclusivement associés aux trois opérations suivantes :

1. Calcul de signature ECDSA ou ECSDSA dans le cadre de la génération du payload AUTH{} [8]
2. Calcul de la clé privée éphémère ECDH lors de la génération d'un payload KE{}
3. Génération des payload NONCE{}

Il convient de noter ici que dans le cadre du référentiel, les IV nécessaires au fonctionnement des modes de chiffrement symétriques (GCM et CTR) sont générés de manière incrémentale et ne nécessitent donc pas d'aléa.

Dans le cadre du calcul de signatures ECDSA ou ECSDSA (décrit respectivement aux sections 3.4 et 3.3) mais également dans le calcul de clé privée ECDH (décrit en section 4.3), une valeur k doit être tirée aléatoirement dans $]0, q[$. Si la génération de k est réalisée par tirage d'une valeur aléatoire puis réduction de celle-ci modulo q, il est nécessaire que la valeur tirée initialement soit au moins de taille $3|q|/2$. Une autre approche permettant de réaliser un tirage aléatoire cette fois uniforme dans $]0, q[$ consiste à tirer autant de fois que nécessaire une valeur aléatoire sur $|q|$ bits jusqu'à obtenir un candidat soit dans $]0, q[$ (cette méthode est décrite en annexe B.2.2 de [FIPS186-4]).

Vérification №9

Toute génération de valeur k devant être tirée aléatoirement dans $]0, q[$ doit être réalisé en utilisant l'une des deux méthodes décrites en section 7.1.

8. L'opération de génération de clé ECDSA ou ECSDSA n'est pas considérée dans le périmètre du référentiel.

La [RFC7296] définie dans les sections 2.10 et 3.9 la taille des payload NONCE{} à utiliser pour établir une session IKE ou un SA IPsec sur la base de la PRF utilisée. Comme indiqué dans le cadre du référentiel en section 5, l'unique fonction de PRF retenue étant `PRF_HMAC_SHA2_256` les payload NONCE{} doivent avoir une longueur d'au moins 128 bits (16 octets). Dans le cadre du référentiel, la taille imposée à 16 octets.

Vérification №10

La taille des nonces produits et acceptés doit être de 16 octets.

7.2 Utilisation de coprocesseur cryptographique

Les éléments concernant l'utilisation de coprocesseurs cryptogaphiques et plus généralement les modalités concernant la réalisation des analyses cryptographiques sont donnés dans [ANSSI-CC-CY-P-01].

7.3 Certificats

Lors d'une authentification IKEv2 par certificat, la validation de la chaine de certificats doit mettre en œuvre uniquement des algorithmes de vérification sur courbe définis en section 3.

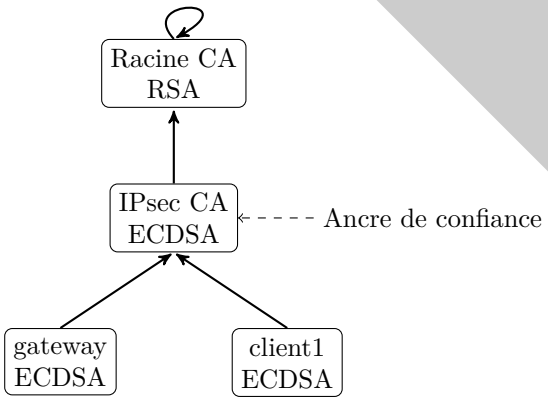


FIGURE 3 – Exemple de chaines de certification

Bien que l'usage de RSA soit prohibé dans ce référentiel, un certificat racine RSA peut être utilisé pour signer un certificat intermédiaire, dédié à IPsec par exemple, à partir du moment où l'autorité de certification utilisée

comme ancre de confiance sur l'équipement est le certificat intermédiaire.

7.4 Anti-rejeu

Les protocoles IKEv2 et ESP fournissent tous les deux une protection contre le rejeu, i.e. un mécanisme d'anti-rejeu.

Dans IKEv2, le champs `Message ID` présent dans le *header* IKE est protégé en intégrité suite à la mise en place de la IKE_SA, comme décrit en section 2.2 de [RFC7296]. Dans le cadre du référentiel, la *window size* est imposée à 1 non négociable pour éviter une complexification du code de l'implémentation et notamment de son automate d'état.

Vérification №11

L'implémentation IKEv2 doit supporter une *window size* de 1 non configurable, i.e. ne doit ni émettre, ni accepter de notification `SET_WINDOW_SIZE` permettant de changer cette valeur. Toute notification `SET_WINDOW_SIZE` doit être ignorée.

Dans ESP, l'anti-rejeu est garanti par l'intégration de la valeur de *Sequence Number* dans le calcul de l'ICV. Dans le cadre du référentiel, le mode ESN doit être proposé lors de la négociation des SA IPsec et accepté. Il en résulte une utilisation d'un compteur sur 64 bits (pour les calculs d'intégrité en émission et réception) dont seule la partie basse est intégrée dans les paquets échangés (tel que le détaille les sous sections 6.1.2 et 6.2.2).

Vérification №12

Le support correct de l'ESN, en émission et réception, doit être vérifié.

Dans le cadre du référentiel, pour garantir une interopérabilité entre équipements, il est nécessaire d'imposer quelques règles concernant la gestion en émission et réception des numéros de séquence pour ESP.

Vérification №13

Un produit compatible du référentiel ne produira pas en émission de paquets déséquencés avec une distance supérieure à 64.

Vérification №14

Un produit compatible du référentiel supportera une fenêtre anti-rejeu d'au moins 1024 paquets.

A Exemples de payload SA

Dans le cadre du référentiel, une implémentation présentera donc le payload SA suivant pour la montée d'une SA IPsec lors d'un échange `CREATE_CHILD_SA` :

```
SA Payload
   |
   +--- Proposal #1 ( Proto ID = ESP(3), SPI size = 4,
   |     |            3 transforms,      SPI = 0x052357bb )
   |     +-- Transform ENCR ( Name = AES-GCM with a 16 octet ICV )
   |     |     +-- Attribute ( Key Length = 256 )
   |     +-- Transform D-H ( Name = ECDH on BrainpoolP256r1 (28) )
   |     +-- Transform ESN ( Name = ESNs )
   |
   +--- Proposal #2 ( Proto ID = ESP(3), SPI size = 4,
   |     |            3 transforms,      SPI = 0x35a1d6f2 )
   |     |
   |     +-- Transform ENCR ( Name = AES-GCM with a 16 octet ICV )
   |     |     +-- Attribute ( Key Length = 256 )
   |     +-- Transform D-H ( Name = ECDH on secp256r1 (19) )
   |     +-- Transform ESN ( Name = ESNs )
   |
   +--- Proposal #3 ( Proto ID = ESP(3), SPI size = 4,
   |     |            4 transforms,      SPI = 0x1c97c41c )
   |     |
   |     +-- Transform ENCR ( Name = ENCR_AES_CTR )
   |     |     +-- Attribute ( Key Length = 256 )
   |     +-- Transform INTEG ( Name = AUTH_HMAC_SHA256_128 )
   |     +-- Transform D-H ( Name = ECDH on BrainpoolP256r1 (28) )
   |     +-- Transform ESN ( Name = ESNs )
   |
   +--- Proposal #4 ( Proto ID = ESP(3), SPI size = 4,
         |            4 transforms,      SPI = 0x43ca0db1 )
         |
         +-- Transform ENCR ( Name = ENCR_AES_CTR )
         |     +-- Attribute ( Key Length = 256 )
         +-- Transform INTEG ( Name = AUTH_HMAC_SHA256_128 )
         +-- Transform D-H ( Name = ECDH on secp256r1 (19) )
         +-- Transform ESN ( Name = ESNs )
```

Présenté sous forme héxadécimale brute, ce payload SA prend alors la forme suivante :

```
220000b4 02000028 01030403 052357bb 0300000c 01000014 800e0100 03000008
0400001c 00000008 05000001 02000028 02030403 35a1d6f2 0300000c 01000014
800e0100 03000008 04000013 00000008 05000001 02000030 03030404 1c97c41c
0300000c 0100000d 800e0100 03000008 0300000c 03000008 0400001c 00000008
05000001 00000030 04030404 43ca0db1 0300000c 0100000d 800e0100 03000008
0300000c 03000008 04000013 00000008 05000001
```

Une version décomposée du même payload dest également présentée ci-dessous :

22	Next Payload (KE)
00	Critical flag + reserved
00b4	Payload Length (0x04 + 0x28 + 0x28 + 0x30 + 0x30)
	Proposals
	Proposal #1 :
02	Last Substruc (more propososal)
00	RESERVED
0028	Proposal Length
01	Proposal Num
03	Protocol ID (ESP)
04	SPI Size
03	Num Transforms
052357bb	SPI (size is 4 here)
	Transform #1
03	Last Substruc (more transform)
00	RESERVED
000c	Transform Length
01	Transform Type (ENCR)
00	RESERVED
0014	Transform ID (AES-GCM with a 16 octet ICV)
	Transform Attributes
800e	Attribute Format (TV) + Type (14, i.e. Key Length)
0100	Attribute Value (256 bit key length)
	Transform #2
03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
04	Transform Type (DH)

```
00          RESERVED
001c        Transform ID (brainpoolP256r1)
          Transform #3
00          Last Substruc (last transform)
00          RESERVED
0008        Transform Length
05          Transform Type (ESN)
00          RESERVED
0001        Transform ID (Use ESN)
        Proposal #2
02      Last Substruc (more propososal)
00      RESERVED
0028    Proposal Length
02      Proposal Num
03      Protocol ID (ESP)
04      SPI Size
03      Num Transforms
35a1d6f2    SPI (size is 4 here)
          Transform #1
03          Last Substruc (more transform)
00          RESERVED
000c        Transform Length
01          Transform Type (ENCR)
00          RESERVED
0014        Transform ID (AES-GCM with a 16 octet ICV)
            Transform Attributes
800e          Attribute Format (TV) + Type (14, i.e. Key Length)
0100          Attribute Value (256 bit key length)
          Transform #2
03          Last Substruc (more transform)
00          RESERVED
0008        Transform Length
04          Transform Type (DH)
00          RESERVED
0013        Transform ID (256-bit random ECP group)
          Transform #3
00          Last Substruc (last transform)
00          RESERVED
0008        Transform Length
```

05	Transform Type (ESN)
00	RESERVED
0001	Transform ID (Use ESN)
	Proposal #3
02	Last Substruc (more propososal)
00	RESERVED
0030	Proposal Length
03	Proposal Num
03	Protocol ID (ESP)
04	SPI Size
04	Num Transforms
1c97c41c	SPI (size is 4 here)
	Transform #1
03	Last Substruc (more transform)
00	RESERVED
000c	Transform Length
01	Transform Type (ENCR)
00	RESERVED
000d	Transform ID (ENCR_AES_CTR)
	Transform Attributes
800e	Attribute Format (TV) + Type (14, i.e. Key Length)
0100	Attribute Value (256 bit key length)
	Transform #2
03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
03	Transform Type (INTEG)
00	RESERVED
000c	Transform ID (AUTH_HMAC_SHA2_256_128)
	Transform #3
03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
04	Transform Type (DH)
00	RESERVED
001c	Transform ID (brainpoolP256r1)
	Transform #4
00	Last Substruc (last transform)
00	RESERVED

0008	Transform Length
05	Transform Type (ESN)
00	RESERVED
0001	Transform ID (Use ESN)
	Proposal #4
00	Last Substruc (last proposal)
00	RESERVED
0030	Proposal Length
04	Proposal Num
03	Protocol ID (ESP)
04	SPI Size
04	Num Transforms
43ca0db1	SPI (size is 4 here)
	Transform #1
03	Last Substruc (more transform)
00	RESERVED
000c	Transform Length
01	Transform Type (ENCR)
00	RESERVED
000d	Transform ID (ENCR_AES_CTR)
	Transform Attributes
800e	Attribute Format (TV) + Type (14, i.e. Key Length)
0100	Attribute Value (256 bit key length)
	Transform #2
03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
03	Transform Type (INTEG)
00	RESERVED
000c	Transform ID (AUTH_HMAC_SHA2_256_128)
	Transform #3
03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
04	Transform Type (DH)
00	RESERVED
0013	Transform ID (256-bit random ECP group)
	Transform #4
00	Last Substruc (last transform)

```
00          |       RESERVED
0008        |       Transform Length
05          |       Transform Type (ESN)
00          |       RESERVED
0001        |       Transform ID (Use ESN)
```

Dans le cadre de la montée d'une IKE_SA (le cas d'une renégociation serait identique, à la taille et la valeur des SPI près), le payload SA contiendra les éléments suivants :

```
SA Payload
   |
   +--- Proposal #1 ( Proto ID = IKE(1), SPI size = 0,
   |     |            3 transforms )
   |     |
   |     +-- Transform ENCR ( Name = AES-GCM with a 16 octet ICV )
   |     |     +-- Attribute ( Key Length = 256 )
   |     |
   |     +-- Transform PRF ( Name = PRF_HMAC_SHA2_256 )
   |     |
   |     +-- Transform D-H ( Name = ECDH on BrainpoolP256r1 (28) )
   |
   +--- Proposal #2 ( Proto ID = IKE(1), SPI size = 0,
   |     |            3 transforms )
   |     |
   |     +-- Transform ENCR ( Name = AES-GCM with a 16 octet ICV )
   |     |     +-- Attribute ( Key Length = 256 )
   |     |
   |     +-- Transform PRF ( Name = PRF_HMAC_SHA2_256 )
   |     |
   |     +-- Transform D-H ( Name = ECDH on secp256r1 (19) )
   |
   +--- Proposal #3 ( Proto ID = IKE(1), SPI size = 0,
   |     |            4 transforms )
   |     |
   |     +-- Transform ENCR ( Name = ENCR_AES_CTR )
   |     |     +-- Attribute ( Key Length = 256 )
   |     |
   |     +-- Transform INTEG ( Name = AUTH_HMAC_SHA256_128 )
   |     |
```

```
|     +-- Transform PRF ( Name = PRF_HMAC_SHA2_256 )
|     |
|     +-- Transform D-H ( Name = ECDH on BrainpoolP256r1 (28) )
|
+--- Proposal #4 ( Proto ID = IKE(1), SPI size = 0,
      |            4 transforms )
      |
      +-- Transform ENCR ( Name = ENCR_AES_CTR )
      |     +-- Attribute ( Key Length = 256 )
      |
      +-- Transform INTEG ( Name = AUTH_HMAC_SHA256_128 )
      |
      +-- Transform PRF ( Name = PRF_HMAC_SHA2_256 )
      |
      +-- Transform D-H ( Name = ECDH on secp256r1 (19) )
```

Présenté sous forme héxadécimale brute, ce payload SA prend alors la forme suivante :

```
220000a4 02000024 01010003 0300000c 01000014 800e0100 03000008 02000005
00000008 0400001c 02000024 02010003 0300000c 01000014 800e0100 03000008
02000005 00000008 04000013 0200002c 03010004 0300000c 0100000d 800e0100
03000008 0300000c 03000008 02000005 00000008 0400001c 0000002c 04010004
0300000c 0100000d 800e0100 03000008 0300000c 03000008 02000005 00000008
04000013
```

Une version décomposée du même payload est également présentée ci-dessous :

22	Next Payload (KE)
00	Critical flag + reserved
00a4	Payload Length (0x04 + 0x24 + 0x24 + 0x2c + 0x2c)
	Proposals
	Proposal #1
02	Last Substruc (more prososal)
00	RESERVED
0024	Proposal Length
01	Proposal Num
01	Protocol ID (IKE)
00	SPI Size
03	Num Transforms
	SPI (size is 0 here)

```
          |     Transform #1
03        |       Last Substruc (more transform)
00        |       RESERVED
000c      |       Transform Length
01        |       Transform Type (ENCR)
00        |       RESERVED
0014      |       Transform ID (AES-GCM with a 16 octet ICV)
          |       Transform Attributes
800e      |         Attribute Format (TV) + Type (14, i.e. Key Length)
0100      |         Attribute Value (256 bit key length)
----------+-----------------------------------------------------------
          |     Transform #2
03        |       Last Substruc (more transform)
00        |       RESERVED
0008      |       Transform Length
02        |       Transform Type (PRF)
00        |       RESERVED
0005      |       Transform ID (PRF_HMAC_SHA2_256)
----------+-----------------------------------------------------------
          |     Transform #3
00        |       Last Substruc (last transform)
00        |       RESERVED
0008      |       Transform Length
04        |       Transform Type (DH)
00        |       RESERVED
001c      |       Transform ID (brainpoolP256r1)
----------+-----------------------------------------------------------
          |   Proposal #2
02        |     Last Substruc (more propososal)
00        |     RESERVED
0024      |     Proposal Length
02        |     Proposal Num
01        |     Protocol ID (IKE)
00        |     SPI Size
03        |     Num Transforms
          |     SPI (size is 0 here)
----------+-----------------------------------------------------------
          |     Transform #1
03        |       Last Substruc (more transform)
00        |       RESERVED
000c      |       Transform Length
01        |       Transform Type (ENCR)
00        |       RESERVED
```

```
0014        Transform ID (AES-GCM with a 16 octet ICV)
            Transform Attributes
800e          Attribute Format (TV) + Type (14, i.e. Key Length)
0100          Attribute Value (256 bit key length)
-------------------------------------------------------------------
          Transform #2
03          Last Substruc (more transform)
00          RESERVED
0008        Transform Length
02          Transform Type (INTEG)
00          RESERVED
0005        Transform ID (PRF_HMAC_SHA2_256)
-------------------------------------------------------------------
          Transform #3
00          Last Substruc (last transform)
00          RESERVED
0008        Transform Length
04          Transform Type (DH)
00          RESERVED
0013        Transform ID (256-bit random ECP group)
-------------------------------------------------------------------
        Proposal #3
02        Last Substruc (more propososal)
00        RESERVED
002c      Proposal Length
03        Proposal Num
01        Protocol ID (IKE)
00        SPI Size
04        Num Transforms
          SPI (size is 0 here)
-------------------------------------------------------------------
          Transform #1
03          Last Substruc (more transform)
00          RESERVED
000c        Transform Length
01          Transform Type (ENCR)
00          RESERVED
000d        Transform ID (ENCR_AES_CTR)
            Transform Attributes
800e          Attribute Format (TV) + Type (14, i.e. Key Length)
0100          Attribute Value (256 bit key length)
-------------------------------------------------------------------
          Transform #2
03          Last Substruc (more transform)
```

```
00          RESERVED
0008        Transform Length
03          Transform Type (INTEG)
00          RESERVED
000c        Transform ID (AUTH_HMAC_SHA2_256_128)
-----------------------------------------------------
          Transform #3
03          Last Substruc (more transform)
00          RESERVED
0008        Transform Length
02          Transform Type (PRF)
00          RESERVED
0005        Transform ID (PRF_HMAC_SHA2_256)
-----------------------------------------------------
          Transform #4
00          Last Substruc (last transform)
00          RESERVED
0008        Transform Length
04          Transform Type (DH)
00          RESERVED
001c        Transform ID (brainpoolP256r1)
-----------------------------------------------------
        Proposal #4
00        Last Substruc (last proposal)
00        RESERVED
002c      Proposal Length
04        Proposal Num
01        Protocol ID (IKE)
00        SPI Size
04        Num Transforms
          SPI (size is 0 here)
-----------------------------------------------------
          Transform #1
03          Last Substruc (more transform)
00          RESERVED
000c        Transform Length
01          Transform Type (ENCR)
00          RESERVED
000d        Transform ID (ENCR_AES_CTR)
            Transform Attributes
800e          Attribute Format (TV) + Type (14, i.e. Key Length)
0100          Attribute Value (256 bit key length)
-----------------------------------------------------
          Transform #2
```

03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
03	Transform Type (INTEG)
00	RESERVED
000c	Transform ID (AUTH_HMAC_SHA2_256_128)
	Transform #3
03	Last Substruc (more transform)
00	RESERVED
0008	Transform Length
02	Transform Type (PRF)
00	RESERVED
0005	Transform ID (PRF_HMAC_SHA2_256)
	Transform #4
00	Last Substruc (last transform)
00	RESERVED
0008	Transform Length
04	Transform Type (DH)
00	RESERVED
0013	Transform ID (256-bit random ECP group)

B Scénario de tests

Liste reprenant des vérifications déjà inclues dans le documents.

- L'échange de cookie est requis pour toutes nouvelles ouverture de session.
- L'encapsulation UDP est activée et toujours effective.
- La *window size* doit valoir 1 ; les notifications `SET_WINDOW_SIZE` reçues sont ignorées.
- Les nonces envoyés sont de la bonne taille et les implémentations refusent des nonces de taille inférieur.
- Les payloads ID ne contiennent pas d'informations topologiques.
- Seules les suites, courbes et algorithmes du référentiel sont proposés et acceptés.
- Les vérifications complémentaires des algorithmes de signature sont implémentés et effectives.
- Les sources d'aléa utilisées sont conformes au RGS.
- La courbe Brainpool est proposée en priorité par rapport à la courbe secp.
- Une proposal ne contient qu'un seul payload transform de chaque type.
- Les clés ECDH sont éphémères et renouvelées à chaque nouvel échange.
- Les clés publiques ECDH générées sont valides pour les courbes supportées par le référentiel et l'implémentation refuse des clés publiques invalides.
- L'établissement d'une IKE_SA n'engendre pas la création de CHILD_SA. Seul un échange CREATE_CHILD_SA permet la création d'une SA IPsec.
- Un ECDH est effectué à chaque nouvel échange CREATE_CHILD_SA.
- La compression IPComp n'est pas proposée et est refusée.
- L'authentification EAP n'est pas proposée et est refusée.
- Seul le mode tunnel protégé par ESP est proposé. Le mode transport est refusé.
- Lors d'une authentification par clé publique, le pair valide que le certificat est rattaché à une racine de confiance.
- L'ESN est systématiquement proposé et accepté.
- Les paquets ESP recus invalides cryptographiquement sont rejetés silencieusement sur le réseau.

Références

[RFC2104] **"HMAC : Keyed-Hashing for Message Authentication"**, Informational, disponible à https://www.ietf.org/rfc/rfc2104.txt

[RFC3686] **"Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc3686.txt

[RFC3713] **"A Description of the Camellia Encryption Algorithm"**, Informational, disponible à https://www.ietf.org/rfc/rfc3713.txt

[RFC4106] **"The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc4106.txt

[RFC4303] **"IP Encapsulating Security Payload (ESP)"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc4303.txt

[RFC4309] **"Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc4309.txt

[RFC4753] **"ECP Groups for IKE and IKEv2"**, Informational, obsoletée par [RFC5903], voir également [Err9], disponible à https://www.ietf.org/rfc/rfc4753.txt

[Err9] **"RFC Errata, Errata ID 9, RFC 4753"**, disponible à http://www.rfc-editor.org

[RFC4754] **"IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc4754.txt

[RFC4868] **"Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc4868.txt

[RFC5282] **"Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol"**, Standards Track, disponible à https://www.ietf.org/rfc/rfc5282.txt

[RFC5639] **"Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation"**, Informational, disponible à https://www.ietf.org/rfc/5639.txt

[RFC5903] **"Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2"**, Informational, disponible à https://www.ietf.org/rfc/rfc5903.txt

[RFC5930] **"Using Advanced Encryption Standard Counter Mode (AES-CTR) with the Internet Key Exchange version 02 (IKEv2) Protocol"**, Informational, disponible à https://www.ietf.org/rfc/rfc5930.txt

[RFC6023] **"A Childless Initiation of the Internet Key Exchange Version 2 (IKEv2) Security Association (SA)"**, Experimental, disponible à https://www.ietf.org/rfc/rfc6023.txt

[RFC6234] **"US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)"**, Informational, disponible à https://www.ietf.org/rfc/rfc6234.txt Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2)

[RFC6932] **"Brainpool Elliptic Curves for the Internet Key Exchange (IKE) Group Description Registry"**, Informational, disponible à https://www.ietf.org/rfc/rfc6932.txt

[RFC6954] **"Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2)"**, Informational, disponible à https://www.ietf.org/rfc/rfc6954.txt

[RFC6989] **"Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)"**, Standard Tracks, disponible à https://www.ietf.org/rfc/rfc6989.txt

[RFC7296] **"Internet Key Exchange Protocol Version 2 (IKEv2)"**, Standards Tracks, disponible à https://www.ietf.org/rfc/rfc7296.txt

[RFC7670] **"Generic Raw Public-Key Support for IKEv2"**, Standards Tracks, disponible à https://www.ietf.org/rfc/rfc7670.txt

[IKEv2-PARAMS] **"Internet Key Exchange Version 2 (IKEv2) Parameters"**, disponible à http://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml

[ISO14888-32016] **'ISO/IEC 14888-3 :2016 - Information technology – Security techniques – Digital signatures with appendix – Part 3 : Discrete logarithm based mechanisms"** disponible (de manière payante) à http://www.iso.org/iso/catalogue_detail.htm?csnumber=64267

[X9.622005] **"Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)- version 2005"** disponible (de manière payante) à http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.62%3A2005

[FIPS197] **"FIPS PUB 197 : Advanced Encryption Standard (AES)"**, disponible à http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[FIPS180-4] **"FIPS PUB 180-4 - FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION - Secure Hash Standard (SHS)"**, disponible à http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[FIPS186-4] **"FIPS PUB 186-4 - FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION - Digital Signature Standard (DSS)"**, disponible à http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[SP800-38A] **"Recommendation for Block Cipher Modes of Operation : Methods and Techniques - Special Publication 800-38A"**, disponible à http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[SP800-38B] **"Recommendation for Block Cipher Modes of Operation : The CMAC Mode for Authentication - Special Publication 800-38B"**, disponible à http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[SP800-38D] **"Recommendation for Block Cipher Modes of Operation : Galois/Counter Mode (GCM) for Confidentiality and Authentication - Special Publication 800-38D"**, disponible à http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[MBAK] **"A Note on Signature Standards"**, disponible à http://eprint.iacr.org/2007/357

[ANSSI-CC-CY-P-01] **"MODALITES POUR LA REALISATION DES ANALYSES CRYPTOGRAPHIQUES ET DES EVALUATIONS DE NOMBRES ALEATOIRES"**, Procédure ANSSI-CC-CRY-P-01/3.0 du 05 mai 2015, disponible à https://www.ssi.gouv.fr/uploads/2014/11/ANSSI-CC-CRY-P-01-Modalit%C3%A9s-pour-la-r%C3%A9alisation-des-analyses-cryptographiques_v3.0.pdf