Introduction à IPsec/ESP

Approche pour construire un *baby* ESP avec Python3

Daniel Migault

October 9, 2023

Contents

List of Figures

Listings

1 Vue de haut niveau du laboratoire

Ce laboratoire considère le cas d'un réseau privé virtuel (VPN) IPsec.

Les cas d'utilisation typiques, représentés dans la figure 1, incluent le scénario roadwarrior représenté sur la figure 1a également en tant que passerelle de sécurité vers le scénario de passerelle de sécurité représenté dans figure 1b.

Dans le scénario de type *road warior*, un appareil, tel qu'un ordinateur portable ou un téléphone mobile —comme représenté sur la figure 1a —accède à distance à un réseau de confiance au travers un réseau non fiable réseau. Dans le cas d'un réseau d'entreprise, la construction d'un VPN permet à un PC mobile ou distant de se connecter au réseau de l'entreprise tout en étant connecté à Internet via un point d'accès WiFi public dans un Internet Café par exemple. Le réseau d'entreprise est considéré comme étant le domaine de confiance, alors que tout ce qui se trouve en dehors du réseau de l'entreprise est considéré comme non fiable.

Pour inclure l'ordinateur portable distant dans le réseau d'entreprise, l'ordinateur portable définit un VPN représenté en rouge. L'ordinateur portable connecté à l'accès WiFi reçoit de l'opérateur une adresse IP. Cette adresse IP est une adresse IP publique et ne permettra pas à l'ordinateur portable d'accéder au réseau privé interne de l'entreprise, par exemple pour envoyer un fichier à imprimer. Afin d'atteindre cette imprimante, l'ordinateur portable doit appartenir au réseau de l'entreprise et s'être vu attribuer une adresse IP privée interne a ce réseau d'entreprise. L'IP publique, acquise dans le cybercafé, sera utile pour joindre la passerelle de sécurité uniquement. La passerelle de sécurité sera le point d'entrée de l'ordinateur portable au réseau d'entreprise. Plus précisément, l'ordinateur portable définit un tunnel chiffré avec la passerelle de sécurité. De plus, la passerelle de sécurité fournira à l'ordinateur portable une adresse IP interne qui appartient au réseau de l'entreprise. Avec cette adresse IP interne, l'ordinateur portable sera en mesure d'établir des communications internes. Typiquement, il pourra contacter l'imprimante et ainsi établir une communication entre les deux adresses IP internes. Ces paquets IP internes sont encapsulés dans le tunnel crypté entre l'ordinateur portable et la passerelle de sécurité, qui est piloté par le VPN.

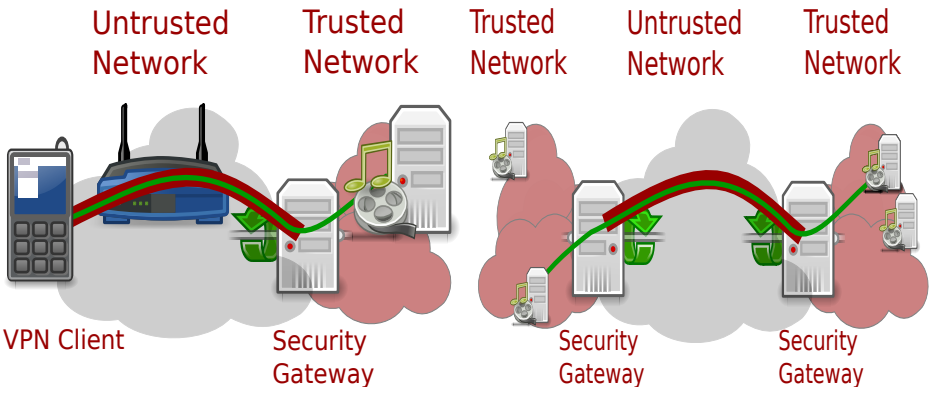
L'ordinateur portable négocie à l'aide du protocole IKEv2 un canal de contrôle sécurisé. L'établissement de ce canal permettra à l'ordinateur portable et à la passerelle de sécurité de s'authentifier mutuellement et de définir un canal authentifié et crypté. Une fois défini, ce canal sera utilisé pour négocier les paramètres nécessaires pour définir le VPN lui-même. Il comprend entre autres paramètres le matériel cryptographique ainsi que l'adresse IP que l'ordinateur portable doit utiliser au sein du réseau de l'entreprise. Cette adresse IP sera généralement utilisée pour atteindre l'imprimante. Les paquets IP entre l'ordinateur portable et l'imprimante auront des adresses IP source et destination qui appartiennent au réseau de l'entreprise.

Ce dernier paquet sera alors encapsulé par le VPN entre l'ordinateur portable et la passerelle de sécurité. Pour les paquets envoyés par le portable vers l'imprimante, la passerelle de sécurité procède au décapsulage et à l'acheminement

du paquet vers l'imprimante. Pour les paquets envoyés de l'imprimante à l'ordinateur portable, la passerelle de sécurité encapsule le paquet et le dirige vers l'ordinateur portable. Le VPN est un tunnel crypté qui protège le trafic entre l'ordinateur portable et la passerelle de sécurité.

Les communications au sein du réseau de l'entreprise sont représentées en vert dans la figure 1. Le réseau d'entreprise utilise généralement des adresses IP de portée locale qui ne sont pas routables sur Internet global. Le client VPN encapsulera alors la communication de l'entreprise à l'intérieur d'un VPN représenté en rouge sur la figure 1. Le tunnel utilise des adresses IP routables sur Internet comme adresses externes.

Les VPNs sont également largement utilisés pour interconnecter les centres de données. Ce cas d'utilisation est représenté dans la figure 1b. La procédure reste similaire, sauf que l'utilisateur n'a pas besoin de se voir attribuer une adresse IP. De même, l'administrateur du centre de données ne souhaite pas exposer à l'Internet sauvage les communications internes de son réseau. En conséquence, la communication interne représentée en vert sur la figure 1b sera encapsulée et cryptée entre les deux passerelles de sécurité.



(a) Road warrior (b) Security Gateway-to-Security Gateway

Figure 1: IPsec VPN use cases

L'architecture IPsec est détaillée dans [7] et définit, entre autres, deux bases de données:

- la base de données de politique de sécurité (Security Policy Database (SPD)) qui définit si un paquet est rejeté (DISCARDed), non protégé (BYPASSed), protégé (PROTECTed). Lorsqu'un nœud envoie un paquet IP, le nœud vérifie les politiques de sécurité (Security Policy (SP)) correspondantes.
- la base de données des associations de sécurité (Security Association Database (SAD)) est une liste ordonnée d'associations de sécurité (Security Asso-

ciation (SA)), et le paquets à PROTÉGER (PROTECT) est testé avec chaque SA jusqu'à ce qu'une correspondance se produise (match). La correspondance est définie grâce aux Traffic Selectors (TS), c'est-à-dire l'adresse IP source, l'adresse IP destination, le protocole de transport (TCP ou UDP), le port source, le port destination. Chaque SA contient des TS et ceux-ci sont comparés avec les TS extraits du paquet. Lorsqu'une correspondance se produit, la SA est sélectionnée et les informations nécessaires pour appliquer la politique de sécurité sont lues. Ces informations contiennent entre autres, l'algorithme utilisé pour le chiffrement et l'authentification, ainsi que d'autres paramètres nécessaires pour construire le paquet IPsec comme l'index des paramètres de sécurité (Security Parameter Index (SPI)), le compteur de numéro de séquence, ainsi que le protocole IPsec utilisé (ESP ou AH), le mode (Tunnel ou Transport) et lorsque le mode Tunnel est défini, l'en-tête Tunnel associée.

Lorsqu'un nœud reçoit un paquet, le SPI indique la SA, qui fournit les paramètres nécessaires pour déchiffrer le paquet. Par souci de simplicité, le laboratoire ne considère pas un paquet en cours de réception et se limite à un nœud envoyant un paquet.

Le but de ce laboratoire est d'implémenter le pseudo code décrit dans la figure 1. Bien que la mise en œuvre d'un ESP interopérable n'entre pas dans le cadre du laboratoire, celui-ci procède à certaines simplifications. Cependant, le laboratoire a l'intention de mettre en œuvre:

- chiffrement authentifié effectué de la même manière qu'avec ESP
- une classe SA qui contient les informations nécessaires pour terminer le processus IPsec
- une classe ESP qui permet de générer la charge utile ESP

```
1  ## Instantiation of the SA object
2  sa = SA()
3
4  ## Processing of the ESP object
5  esp = ESP(sa)
6
7  ## Inner packet to be processed
8  inner_ip_packet
9
10 ## Processing the ESP packet
11 esp_pkt = esp.process(ip_packet)
12
13 ## Buiding the IP packet
14 outbound_pkt
```

Listing 1: Lab Pseudo Code

Bien qu'aucune connaissance ne soit requise pour effectuer le laboratoire, on s'attend à ce qu'après le laboratoire, l'étudiant se familiarise avec :

- la lecture de spécifications techniques comme les RFC
- la programmation Python3
- l'implémentation de protocole réseau

2 Installation

Ce laboratoire sera fortement basé sur Python3 et des modules spécifiques. Ce laboratoire suppose également que python est exécuté sur Ubuntu, une distribution Linux. Si d'autres environnements sont utilisés, une procédure d'installation spécifique peut être impliquée mais resterait proche de celle décrite dans cette section. La procédure d'installation est détaillée dans Listing 2.

```
## Installation on linux Ubuntu of the python3 as well
## as pip, the python package manager:

sudo apt-get install python3 python3-pip

## Installation of construct to package to handle packet
## structure as well as cryptodome the package to perform
## cryptographic operations.

pip3 install pycryptodomex construct==2.9.52
```

Listing 2: Installation des modules Python dans Linux Ubuntu

3 Construire une classe SA

Dans notre cas, l'objectif principal d'une SA est de permettre le chiffrement de la charge utile ainsi que les paramètres nécessaires pour permettre la génération d'un paquet ESP.

Dans ce laboratoire, le chiffrement utilisé est ENCR_AES_GCM_16_IIV [6]. Cette transformation est définie à partir d'AES-GCM [8] et définit une manière spécifique de générer le nonce.

3.1 Introduction à AES_GCM

Avant de considérer le format défini par *baby* ESP, cette section présente le chiffrement cryptographique d'AES-GCM. *cryptodome* [3] est un module qui permet de chiffrer/déchiffrer en utilisant AES en mode GCM (AES_GCM). Il est recommandé de lire la documentation associée [2] qui fournit des exemples

ainsi qu'une description des différents objets. Le code, dans Listing 3 répertorie l'exemple fourni dans le fichier `aes_gcm.py`.

Supposons qu'Alice veuille chiffrer le message secret "secret" à Bob (texte clair). Alice et Bob partagent une clé ainsi qu'un algorithme de chiffrement (AES_GCM) à utiliser.

AES_GCM [4] est un mode qui permet le chiffrement authentifié (Authenticated Encryption (AE)) et le chiffrement authentifié avec les données associées (AEAD) [9]. AEAD est une forme de cryptage qui garantit la confidentialité et l'authenticité. De plus, il permet d'effectuer l'authentification en fournissant des données supplémentaires pour s'assurer que l'authentification est effectuée dans un contexte donné. Un contexte typique peut être lié à une session établie entre Alice et Bob. Le besoin de ce contexte pour authentifier le paquet rend le paquet inutilisable en dehors de ce contexte.

Notez que tandis que AES_GCM peut être utilisé avec des données associées. Ce laboratoire fournit une forme simpliste de l'utilisation d'AES_GCM qui ne tient pas compte des données associées.

Le code présenté dans Listing 3 est essentiellement composé de deux phases :

1 Instanciation d'un objet chiffré : `AES.new(...)`

2 Utilisation de l'objet de chiffrement pour effectuer les opérations cryptographiques

   a) Cryptage AE pour Alice : `encrypt_and_digest(..)`

   b) Décryptage AE pour Bob : `decrypt_and_verify(...)`

Comme les objets chiffrés respectivement instanciés par Alice et Bob doivent fonctionner de manière coordonnée, ces deux objets doivent être instanciés avec exactement les mêmes paramètres. Plus explicitement, les paramètres des objets chiffrés `key`, `mode`, `nonce` et `mac_len` doivent être les mêmes. Comme mentionné dans la documentation de cryptodome [2], le `nonce` peut être généré aléatoirement par Alice, et en tant que tel n'a pas besoin d'être fourni. Cependant, il doit être fourni par Bob lorsqu'il instancie son objet de chiffrement. Dans notre exemple, la valeur nonce est extraite de l'attribut 'nonce' de l'objet de chiffrement d'Alice (voir la documentation du cryptodome [2]).

Le nonce est un paramètre essentiel généré pour chaque texte brut chiffré qui doit être unique pour une clé donnée.

Notez que le nonce est également désigné comme vecteur d'initialisation (IV) dans certaines spécifications comme [4]. Dans IPsec, IV et nonce ne sont pas équivalents car l'IV est utilisé pour générer le nonce [8] et ils ont des tailles différentes. Dans ce laboratoire, nous faisons une distinction entre nonce et IV.

IPsec utilise également la valeur de contrôle d'intégrité (ICV) tandis que NIST [4] ou cryptodome utilisent les termes tag d'authentification ou tag. Ces termes sont équivalents.

Voici le code qui chiffre/déchiffre un message en clair. Le code se trouve dans le fichier `aes_gcm.py`.

```
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes

from binascii import hexlify
from typing import Tuple

def nonce_info( cipher )-> Tuple[ bytes, int ]:
  """returns nonce value and nonce length """
  ## BEGIN CODE TO BE CHANGED
  nonce = b''
  nonce_len = 0
  ## END CODE TO BE CHANGED
  return nonce, nonce_len

print( "================================================"
                              )
print( "====== Alice and Bob Secure Communication ======"
                              )
print( "================================================"
                              )
## secret key shared by Alice and Bob
key = get_random_bytes(16)

## Alice
alice_plaintext = b"secret"
print("Alice plaintext is: %s"%alice_plaintext)
##    - 1. initilaization of the cipher
alice_cipher = AES.new(key, AES.MODE_GCM,\
                       nonce=None, mac_len=16)
##    - 2. encryption, authentication
ciphertext, icv = \
  alice_cipher.encrypt_and_digest(alice_plaintext)

print( "The encrypted message sent by Alice to Bob is:" )
print( f"    - ciphertext: {hexlify( ciphertext, sep=' '
                            )}" )
print( f"    - icv: {hexlify(icv, sep=' ' )}" )
nonce, nonce_len = nonce_info( alice_cipher )
print( f"    - nonce [{nonce_len} bytes ]: {hexlify(
                            nonce, sep=' ' )}" )

## Bob
bob_cipher = AES.new(key, AES.MODE_GCM,\
                     nonce=nonce,\
                     mac_len=16)

bob_plaintext = \
  bob_cipher.decrypt_and_verify(ciphertext, icv)
print("Bob plaintext is: %s"%bob_plaintext)
```

```
45
46 ## secret key shared by Alice and Bob
47 key = b''
48 nonce_dict = {
49   'nonce_0': b'', \
50   'nonce_1': b'', \
51   'nonce_2': b'', \
52   'nonce_3': b'' }
53 clear_text = "secret"
54
55 def encrypt_and_digest( key:bytes, nonce_dict:dict,
                            clear_text:bytes ) -> dict:
56   """ return a dictionary that associates to any nonce
                              label a uplet
57       ( nonce, cipher_text, icv ) corresponding to the
                                  cipher_text and icv
58       of the encrypted clear_text
59   """
60   output_dict = {}
61   ## BEGIN CODE TO BE CHANGED
62
63
64   ## END CODE TO BE CHANGED
65   return output_dict
```

Listing 3: Code source de `aes_gcm.py`

L'exécution du code `aes_gcm.py` donne les résultats suivants :

```
$ python3 aes-gcm-lab.py
Alice plaintext is: b'secret'
The encrypted message sent by Alice to Bob is:
    - ciphertext: b'23505b34f096'
    - icv: b'7269aa184fefc74b55d2f90d26d12b55'
    - nonce: b'3fbe88d0b2891fc5efc9f132c1be67eb'
Bob plaintext is: b'secret'

$ python3 aes-gcm-lab.py
Alice plaintext is: b'secret'
The encrypted message sent by Alice to Bob is:
    - ciphertext: b'f7ca79fade5b'
    - icv: b'0889d696d280419ae844b271380128d4'
    - nonce: b'a76fc972816e3a03c1d174adb9b880e5'
Bob plaintext is: b'secret'
```

Listing 4: Exécution de `aes_gcm.py`

QUESTION 1

Mettez à jour le code `aes_gcm.py` afin qu'il puisse s'exécuter correctement.
Mettre a jour la fonction `nonce_info`, qui à partir d'un object cipher retourne la valeur du nonce ainsi que sa taille. (2 points)

QUESTION 2

Le listing 4 représente la sortie du code `aes_gcm.py` exécuté deux fois. En regardant les paramètres fournis pour instancier les objets chiffrés d'Alice et Bob, expliquez pourquoi les textes chiffrés (ciphertext) sont différents à chaque fois que `aes_gcm.py` est exécuté.
Seule une réponse textuelle dans le rapport de laboratoire est attendue pour cette question, aucun script python n'est nécessaire pour cette question. (1 point)

QUESTION 3

Montrez que pour une valeur de clé donnée, l'utilisation de nonce de valeur différentes donne des chiffrés (ciphertext) et des signatures (icv) de valeur différentes.
Pour ce faire, on considèrera la clé suivante:

```
\xf1\x6a\x93\x0f\x52\xa1\x9b\xbe\x07\x1c\x6d\x44\xb4\x24\xf3\x03
```

Les nonces suivants associes à leur label respectif.

- $nonce_0$ est un nonce de 16 octets nuls.
- $nonce_1$ est un nonce de 12 octets nuls.
- $nonce_2$ est un nonce de 16 octets nuls avec son bit de moindre poids mis a 1.
- $nonce_3$ est un nonce de 12 octets nuls avec son bit de moindre poids mis a 1.

Le message clear_text est b'secret'.
Les valeurs seront calculées a l'aide de la fonction `encrypt_and_digest` et les valeurs spécifiques seront reportées dans le rapport. L'évaluation de la fonction `encrypt_and_digest` ferra l'objet d'une évaluation qui prendra également d'autres valeurs que celles fournies par l'exemple. (4 points)

3.2 Mise en œuvre de l'ENCR_AES_GCM_16_IIV

Le chiffrement défini par ENCR_AES_GCM_16_IIV est décrit dans [6] et est basé sur le chiffrement ENCR_AES_GCM_16 décrit [8].

Pour ENCR_AES_GCM_16, selon [8], le nonce est constitué de 12 octets dérivé d'un sel de 4 octets et d'un IV de 8 octets. Le sel devrait être fourni par la SA (c'est-à-dire à partir d'un contexte local), tandis que l'IV est fourni explicitement dans le paquet ESP. Selon [6], avec ENCR_AES_GCM_16_IIV, l'IV n'est plus fourni explicitement, mais est plutôt dérivé du champ de numéro de séquence d'un paquet ESP. Voir Figure 2 ou [5] pour une description d'ESP. Cela permet d'économiser 8 octets par paquet. Une fois le nonce généré, tout est traité comme pour ENCR_AES_GCM_16.

Le numéro de séquence peut être codé sur 4 octets ou 8 octets lorsqu'un numéro de séquence étendu est utilisé. Par conséquent, lorsqu'un numéro de séquence étendu est utilisé, l'IV peut prendre la valeur du numéro de séquence. D'autre part, lorsque le numéro de séquence est codé sur 4 octets, il est nécessaire de définir comment l'IV de 8 octets est dérivé du numéro de séquence de 4 octets. L'information sur le type de numéro de séquence utilisé (régulier ou étendu) est fournie par le SA. Dans notre cas, la SA met `ext_seq_num_flag` à `True` lorsque les numéros de séquence étendus sont utilisés.

Pour générer le nonce à partir du numéro de séquence, nous utilisons un module spécifique *construct* [1], qui permet de définir des structures et de les convertir dans un format binaire. Dans notre cas, il permet de spécifier un numéro de séquence (à l'aide d'un *int*) et de générer l'IV et le nonce associés (au format binaire). A l'inverse, il permet de convertir le format binaire approprié en int approprié. Typiquement, selon le`ext_seq_num_flag`, la structure considérera automatiquement le numéro de séquence comme codé sur 32 bits ou 64 bits et le convertira dans un format binaire. Le nonce peut avoir deux représentations : une structure (un dictionnaire) et une représentation binaire. *construct* permet de traduire la structure en représentation binaire à l'aide de la fonction *build*. Inversement, il permet de traduire la représentation binaire en structure à l'aide de la fonction *parse*. Il s'agit d'une application directe du module de construction, veuillez donc consulter [1] pour plus d'informations. Notez qu'il s'agit d'un cas extrêmement simple ou l'on pourrait se passer d'un parseur de type *construct* et utiliser des fonctions spécifiques Python3. Le code ci-dessus se trouve dans `nonce.py`.

```
1  from construct.core import *
2  from construct.lib import *
3  from binascii import hexlify
4  from typing import Tuple
5
6  ## The section of the code that need to be updated are
7  ## indicated with XXXX
8
9
10
```

```
11 def show_nonce(salt:bytes, seq_num:int, ext_seq:bool) ->
                              Tuple[ bytes, dict ] :
12   """shows the nonce in a binary and structure format """
13
14   IIV_Nonce = Struct(
15     ## Replace XXXX by the appropriated value which
16     ## indicates the length of the salt as
17     ## a number of bytes
18     "salt" / Bytes(XXXX),
19     ## Replace the byte value taken by Const. The
20     ## binary value is not correct and needs to be
21     ## replaced completely. The first  bytes have
22     ## only been indicated as an example
23     ## on how to write bytes and may not be correct.
24     "iv" / IfThenElse(this._.ext_seq_num_flag,
25       Struct( "seq_num_counter" / Int64ub),
26       Struct( "zero" / Const(b'\x01\x02\x03XXXXX'),
27               "seq_num_counter" / Int32ub)
28       )
29   )
30
31   ## defining the structure
32   nonce = { 'salt' : salt, \
33             'iv' : {'seq_num_counter' : seq_num } }
34   try:
35     ## converting structure to binary
36     byte_nonce = IIV_Nonce.build(\
37                     nonce,
38                     ext_seq_num_flag=ext_seq)
39     ## parsing binary to structure
40     struct_nonce = IIV_Nonce.parse(\
41                      byte_nonce,
42                      ext_seq_num_flag=ext_seq)
43     return byte_nonce, struct_nonce
44   except:
45     print("\n---")
46     print("> ERROR : Unable to generate the nonce")
47     print("> Inputs:")
48     print(">    - salt: %s"%salt)
49     print(">    - sec_num: %s"%seq_num)
50     print(">    - ext_seq_flag: %s"%ext_seq)
51     print("-----\n")
52     return None, None
53
54
55   ## printing the different representations
56   print("\n---")
57   print("Inputs:")
58   print("    - salt: %s"%salt)
59   print("    - sec_num: %s"%seq_num)
```

```
60    print("    - ext_seq_flag: %s"%ext_seq)
61    print("Nonce (binary)")
62    print("    - nonce [%s bytes]: %s"%(len(byte_nonce),
63                                        byte_nonce))
64    print("Nonce (structure)")
65    print("    - nonce: %s"%struct_nonce)
66    print("---\n")
```

Listing 5: Code source de `nonce.py`

? QUESTION 4

Assurez-vous que `nonce.py` fonctionne correctement. Mettez à jour le code afin que la structure reflète la description fournie dans [5] et [6]. Une réponse textuelle est attendue dans le rapport de laboratoire ainsi que le code correspondant utilisé. Le nom de fichier pour le code sera `nonce.py` (4 points)

? QUESTION 5

Mettez à jour le code `nonce.py` et fournissez la structure et la représentation binaire des nonces suivants.

- a) un numéro de séquence de 5 avec un numéro de séquence étendu activé
- b) un numéro de séquence de 5 avec un numéro de séquence étendu désactivé (cas standard)
- c) un numéro de séquence de 4294967295 avec un numéro de séquence étendu activé
- d) un numéro de séquence de 4294967295 avec un numéro de séquence étendu désactivé (cas standard).
- e) un numéro de séquence de 4294967296 avec un numéro de séquence étendu activé
- f) un numéro de séquence de sur 4294967296 avec un numéro de séquence étendu désactivé (cas standard). Si une erreur se produit, expliquez en la raison ainsi que ce que représente 4294967296.

Dans tous les cas, nous prendrons `\xf7\xca\x79\xfa` pour la valeur du sel.
Une réponse textuelle est attendue dans le rapport de laboratoire. Aucun code n'est attendu (2 points)

La transformation qui nous intéresse est ENCR_AES_GCM_16_IIV. Jusqu'à présent, nous avons examiné AES_GCM ainsi que la formation du Nonce. AES_GCM est un algorithme de chiffrement qui fournit également une authentification. L'authentification est effectuée via un icv qui peut prendre différentes longueurs.

? QUESTION 6

D'après la spécification [6], déterminez la longueur de l'icv dans le cas de ENCR_AES_GCM_16_IIV.
Une réponse textuelle est attendue dans le rapport de laboratoire. Aucun script spécifique n'est attendu. (1 point)

? QUESTION 7

Sur la base de l'achèvement de `nonce.py` à la question 5, complétez le code `cipher_obj.py` pour que la fonction `ciphers_obj` renvoie un objet de chiffrement. L'objet chiffré est généré par la fonction `AES.new(...)` (voir section 3.1). Les paramètres pour instancier l'objet de chiffrement sont définis dans le fichier. Seule la fonction `cipher_obj` doit être mise à jour. Mettez à jour le code et fournissez le code résultant, ainsi que les valeurs pour le nonce, le texte chiffré et l'icv.
Le nom de fichier du code sera `cipher_obj.py`. (2 points)

```
1  from construct.core import *
2  from construct.lib import *
3  from binascii import hexlify
4  from Cryptodome.Cipher import AES
5  from Cryptodome.Random import get_random_bytes
6
7  ## The section to complete are indicated
8  ## with XXXX or tagged with BEGIN_CODE and
9  ## END_CODE
10
11 ## AES_GCM_16_IIV related shared context between
12 ## Alice and Bob. Please do not update this section
13 key = b'\xf1\x6a\x93\x0f\x52\xa1\x9b\xbe\x07\x1c\x6d\x44\
                            xb4\x24\xf3\x03'
14 mac_len=16
15 ext_seq_num_flag = False
16 seq_num_counter = 5
17 salt = b'\xf7\xca\x79\xfa'
18
19
20
21 def ciphers_obj( key:bytes, mac_len:int, ext_seq_num_flag
                            :bool, seq_num_counter:int,
                            salt:bytes ):
```

```
22    """ returns the cipher object for AES_GCM """
23    ## Complete the code so the function returns the
24    ## appropriated cipher object. The purpose is to
25    ## fill XXXX with the appropriated parameters
26
27    ## BEGIN_CODE
28    IIV_Nonce = Struct(
29      ## Replace XXXX by the appropriated value which
30      ## indicates the length of the salt as
31      ## a number of bytes
32      "salt" / Bytes(XXXX),
33      "iv" / IfThenElse(this._.ext_seq_num_flag,
34      ## Replace the byte value taken by Const. The
35      ## binary value is not correct and needs to be
36      ## replaced completely. The first  bytes have
37      ## only been indicated as an example
38      ## on how to write bytes and may not be correct.
39        Struct( "seq_num_counter" / Int64ub),
40        Struct( "zero" / Const(b'XXXXXXXXX'),
41                "seq_num_counter" / Int32ub)
42        )
43    )
44
45    return AES.new(XXX)
46    ## END_CODE
47
48
49  ## Alice
50  alice_plaintext = b"yet another secret"
51  print("Alice plaintext is: %s"%alice_plaintext)
52  alice_cipher = ciphers_obj()
53  ## encryption, authentication
54  ciphertext, icv =\
55    alice_cipher.encrypt_and_digest(alice_plaintext)
56  nonce = alice_cipher.nonce
57  print("The encrypted message sent by Alice to Bob is:")
58  print(" - (nonce [%s]: %s,"%(len(nonce),
59                               hexlify(nonce)))
60  print(" - ciphertext: %s"%hexlify(ciphertext))
61  print("icv[%s]: %s"%(len(icv), hexlify(icv)))
62
63
64  ## Bob
65  bob_cipher = AES.new(key, AES.MODE_GCM,\
66                       nonce=nonce, mac_len=mac_len)
67  ### verification, decryption
68  bob_plaintext = \
69    bob_cipher.decrypt_and_verify(ciphertext, icv)
70  print("Bob plaintext is: %s"%bob_plaintext)
```

Listing 6: Code source de `cipher_obj.py` à compléter par l'étudiant.

3.3 Utiliser SA

Dans notre cas, l'association de sécurité (SA) héberge les informations partagées entre Alice et Bob et est chargée de fournir l'objet de chiffrement approprié pour tout paquet. N'oubliez pas que SA est recherché pour tout paquet, tout paquet a un numéro de séquence différent, l'objet de chiffrement est généré pour n'importe quel numéro de séquence.

? QUESTION 8

Mettez à jour le module `sa.py` afin que `test-sa-lab.py` puisse être exécuté de manière appropriée. Le module `test-sa-lab.py` importe le module `sa.py`. La mise à jour de `sa.py` est limitée à la fonction `cipher_obj`. La fonction est très similaire à celle écrite dans le `cipher_obj.py`. Cependant, les deux fonctions diffèrent sur plusieurs points. Dans `sa.py`, la fonction `ciphers_obj` renvoie une liste d'objets chiffrés et pas seulement un seul objet chiffré. De plus, la fonction doit initialiser les paramètres à partir des attributs de la fonction SA. Ces attributs sont supposés être trouvés dans la fonction `SA.__init__(...)`, ainsi que par certaines fonctions au sein de l'objet SA. La structure VII_Nonce doit également être mise à jour. La structure est la même que celle utilisée dans `cipher_obj.py`.
Avec la classe SA, Alice peut chiffrer et Bob peut déchiffrer. Alice et Bob n'utilisent pas actuellement le format ESP. La section suivante est consacrée à la construction d'un module ESP afin qu'Alice et Bob puissent réellement établir une communication à l'aide d'ESP. Le module ESP sera chargé de formater et de chiffrer le message à l'aide de l'objet de chiffrement fourni par le SA.
Une réponse textuelle est attendue dans le rapport de laboratoire ainsi que le code correspondant utilisé. Le nom de fichier du code sera `sa.py`. (2 points)

```
1 from sa import SA
2 from binascii import hexlify
3
4 ## The SA is instantiated both Alice and Bob
5 alice_sa = SA()
6 ## AES_GCM_16_IIV related shared context between Alice
                            and Bob
7 alice_sa.esp_enc_alg = "ENCR_AES_GCM_16_IIV"
```

```
8  alice_sa.esp_enc_key = b'\xf1\x6a\x93\x0f\x52\xa1\x9b\xbe
                           \x07\x1c\x6d\x44\xb4\x24\xf3\
                           x03'
9  alice_sa.ext_seq_num_flag = False
10 alice_sa.seq_num_counter = 5
11
12 bob_sa = SA()
13 ## AES_GCM_16_IIV related shared context between Alice
                           and Bob
14 bob_sa.esp_enc_alg = "ENCR_AES_GCM_16_IIV"
15 bob_sa.esp_enc_key = b'\xf1\x6a\x93\x0f\x52\xa1\x9b\xbe\
                           x07\x1c\x6d\x44\xb4\x24\xf3\x03
                           '
16 bob_sa.ext_seq_num_flag = False
17 bob_sa.seq_num_counter = 5
18
19
20 ## Alice
21 alice_plaintext = b"yet another secret"
22 print("Alice plaintext is: %s"%alice_plaintext)
23 alice_cipher = alice_sa.ciphers_obj()[0]
24 ## encryption, authentication
25 ciphertext, icv = \
26   alice_cipher.encrypt_and_digest(alice_plaintext)
27 nonce = alice_cipher.nonce
28 print("The encrypted message sent by Alice to Bob is:")
29 print(" - (nonce [%s]: %s,"%(len(nonce), hexlify(nonce)))
30 print(" - ciphertext: %s"%hexlify(ciphertext))
31 print("icv[%s]: %s"%(len(icv), hexlify(icv)))
32
33
34 ## Bob
35 bob_cipher = bob_sa.ciphers_obj()[0]
36 ## encryption, authentication
37 ### verification, decryption
38 bob_plaintext = \
39   bob_cipher.decrypt_and_verify(ciphertext, icv)
40 print("Bob plaintext is: %s"%bob_plaintext)
```

Listing 7: Code source de `test-sa-lab.py` qui teste `sa.py`

Cette section implémente *baby* ESP. Cela comprend le cryptage, l'authentification des données à envoyer ainsi que le formatage approprié du paquet. La description complète d'ESP est définie dans [5]. ESP contient plusieurs mécanismes qui peuvent ne pas nécessairement être mis en œuvre dans une mise en œuvre minimale. La description d'une implémentation minimale pour ESP est définie dans [**?**].

Le format du paquet ESP est défini dans [5] comme ci-dessous :

Si Payload Data représentent les données qui doivent être chiffrées, ESP

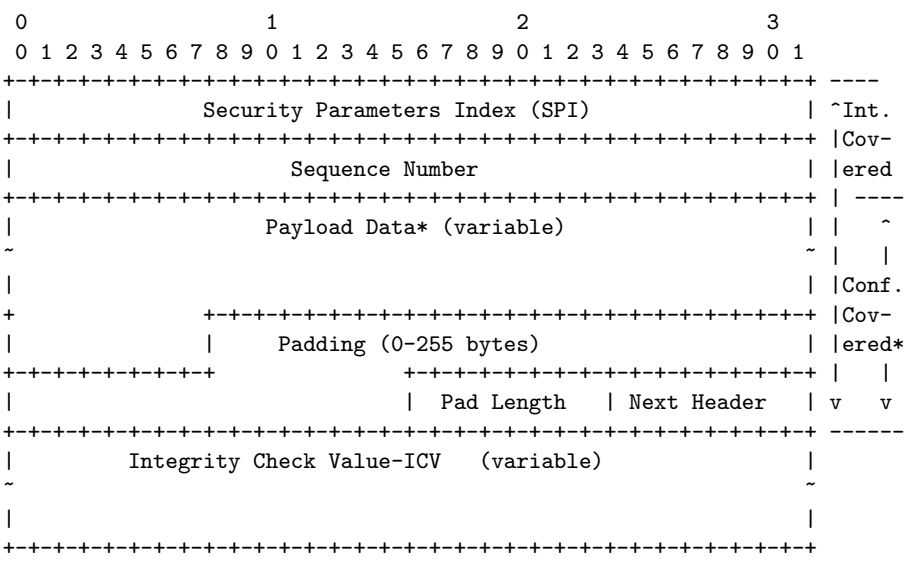


Figure 2: Format d'un packet ESP

procède comme suit :

1. Construisez le clear text ESP désigné par la structure ClearTextESPPayload, qui comprend Payload Data, Padding, Pad Length, Next Header. formate les données en une charge utile en texte clair qui inclut les données de charge utile, le rembourrage, la longueur du pad et l'en-tête suivant.

2. Construisez le paquet ESP chiffrée désigné par la structure EncryptedESPayload, c'est-à-dire la concaténation du texte clair chiffrée dans un texte chiffré et l'icv.

3. Construisez le paquet ESP en concatenant le chiffré ESP chiffrée et l'entête ESP. L'en-tête ESP est constitué du SPI et du numéro de séquence.

Lorsqu'un paquet ESP est reçu, seuls le SPI et le numéro de séquence sont en clair. Le SPI est utilisé pour lier la SA appropriée qui contient les clés et les algorithmes cryptographiques impliqués. Ces algorithmes cryptographiques spécifient la longueur de l'ICV. Le SPI et le numéro de séquence ont une longueur fixe de 4 octets. Compte tenu de la longueur du SPI, du numéro de séquence, de l'ICV ainsi que de la taille de l'ensemble du paquet ESP, le récepteur peut extraire le texte chiffré ainsi que l'ICV et procéder au déchiffrement authentifié. Une fois cette étape réalisée, le récepteur lit la nature des données utiles en lisant le champ Next Header et supprime les octets de padding pour extraire les données utiles. Le noyau peut continuer à traiter les données en claire du Payload Data en conséquence.

QUESTION 9

Dans le cas d'un VPN, le Payload Data représenté dans la Figure 2, est un paquet IP (avec des adresses internes). Supposons qu'Alice tunnelise le paquet IPv6. Déterminez la valeur du champ Next Header dans son paquet ESP. La valeur est une valeur standard définie d'un numéro de protocole IP défini sur la page Web IANA.
Conseils : Il peut être plus facile de regarder [5].
N'oubliez pas que lors du décryptage du paquet ESP, le récepteur utilisera le Next Header pour déterminer la nature du contenu attendue.
Une réponse textuelle est attendue dans le rapport. (1 point)

Dans le cas d'AES_GCM, le champ Padding permet un alignement sur 4 octets. Sa longueur est définie par le champ Pad Length et son contenu est défini par [7] comme suit :

> The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: 1, 2, 3, ...

```
1  from construct.core import *
2  from construct.lib import *
3
4  def pad( data_len ):
5    """ return the Padding field
6
7    Args:
8      data_len: the length of the Data Payload
9    """
10   ### Complete the code so it returns the necessary
11   ### padding bytes for an ESP packet. The padding
12   ### bytes are derived from data_len the length
13   ### expressed in number of bytes of the Data
14   ### Payload
15
16   ##BEGIN_CODE
17
18   return padding_bytes
19   ##END_CODE
```

Listing 8: Source code de `pad`

QUESTION 10

Écrivez une fonction de padding comme décrit dans [7]. Cette fonction sera appelée `pad` dans `pad.py` comme illustré dans Listing 8. Il renverra le padding approprié (voir Figure 2) pour un Payload Data de *data_len*. Notez que le padding, dans notre cas, n'incluent pas Pad Length et est limite au champs Padding de la Figure 2.
Notez que le padding peut prendre plusieurs valeurs pour sa longueur. Typiquement, si un bourrage de longueur 2 ($pad\_len = 2$) réalise un alignement de 4 octets, $pad\_len = 2 + 4$ réalise également un tel alignement et plus généralement $pad\_len = 2 + n * 4$ avec $pad\_len < 255$. Dans notre cas, la fonction pad retournera la plus petite valeur possible, c'est-à-dire celle qui correspond à $n = 0$.
Le nom de fichier pour le code sera `pad.py`. (4 points)

QUESTION 11

Le code `esp.py` implémente un "baby esp". Veuillez compléter le code afin qu'Alice et Bob puissent échanger des charges utiles ESP.
Une réponse textuelle est attendue dans le rapport de laboratoire ainsi que le code correspondant utilisé. Le nom de fichier du code sera `esp.py`. (2 points)

4 Instruction de soumission

Veuillez soumettre les fichiers suivants en respectant très exactement leur nom. Veuillez ne pas les soumettre dans une archive, mais bien en tant que fichier individuel au sein de votre répertoire de soumission. Les fichiers n'ayant pas le nom attendu peuvent ne pas être évalués.

- report.pdf
- aes_gcm.py
- nonce.py
- cipher_obj.py
- sa.py
- pad.py
- esp.py

References

[1] "construct," 2018. [Online]. Available: https://construct.readthedocs.io/en/latest/

[2] "Modern modes of operation for symmetric block ciphers: AES GCM," 2020. [Online]. Available: https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html#gcm-mode

[3] "PyCryptodome: a self-contained Python package of low-level cryptographic primitives," 2020. [Online]. Available: https://pycryptodome.readthedocs.io

[4] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST SP 800-38D, Nov. 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[5] S. Kent, "IP Encapsulating Security Payload (ESP)," RFC 4303, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4303

[6] D. Migault, T. Guggemos, and Y. Nir, "Implicit Initialization Vector (IV) for Counter-Based Ciphers in Encapsulating Security Payload (ESP)," RFC 8750, Mar. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8750

[7] K. Seo and S. Kent, "Security Architecture for the Internet Protocol," RFC 4301, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4301

[8] J. Viega and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)," RFC 4106, Jun. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4106

[9] "Authenticated encryption," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Authenticated_encryption