

IKEV2 TESTING

DAVIDE DE ZUANE & RAHMI EL MECHRI

CONTENTS

- 1 Introduction 3
- 2 Setup 4
 - 2.1 Environment 4
 - 2.2 Configuration 4
- 3 Testing 7
 - 3.1 Analisi traffico 7
 - 3.2 Utilizzo risorse CPU 8
 - 3.3 Occupazione di memoria 8
- 4 Conclusioni 9
- A Configuration File 10
 - A.1 Initiator 10
 - A.2 Responder 11
- B Tools 12
- C Certificati 16
 - C.1 RSA Certificate 16
 - C.2 ECDSA 17

ABSTRACT

1 INTRODUCTION

Tutte le informazioni su IKE sono state recuperate tramite la RFC[1].

2 SETUP

Andiamo a vedere nel dettaglio l'ambiente e la configurazione che abbiamo utilizzato per realizzare i test. Per verificare le capacità di IKE abbiamo previsto:

- 3 modalità di autenticazione;
- 2 chiper suite differenti da utilizzare.

Nella fase di sperimentazione abbiamo utilizzato le seguenti convenzioni:

- **Initiator:** l'host che invia la richiesta di stabilire una SA;
- **Responder:** l'host che risponde alle richieste.

2.1 Environment

Per simulare i due host della comunicazione abbiamo creato due macchine virtuali tramite l'utilizzo di qemu/kvm, questo per avere delle performance il più possibile simili a quelle reali. Le due macchine virtuali sono state create in modalità bridge, questo per evitare problemi con la modalità NAT.

Le macchine virtuali utilizzate hanno le seguenti specifiche:

- *Processore:* 2 core (flag -smp)
- *Memoria:* 2048MB (flag -m)
- *OS:* Debian 11
- *Network:* Bridge

Le macchine virtuali sono state create utilizzando `qemu/kvm` tramite i seguenti comandi è possibile creare la macchina virtuale.

Per prima cosa è necessario creare un disco immagine.

```
$ qemu-img create -f qcow2 disk.img 10G
```

Ora avviamo la macchina virtuale utilizzando il seguente comando.

```
$ qemu-system-x86_64 -smp 2 -m 2G -hda disk.img -cdrom <debian_iso> \
-net bridge,br=virbr0 -enable-kvm & disown
```

Un procedimento simile si applica per l'altra macchina virtuale. Se non si vuole proseguire in questo modo si può utilizzare l'interfaccia grafica fornita da virt-manager.

2.2 Configuration

I file e le directory coinvolte nel processo di configurazione sono i seguenti. Dato che una delle principali modifiche di IKEv2 rispetto alla versione precedente è la possibilità di autenticazione tramite certificati.

```
/etc
├─ ipsec.conf
├─ ipsec.secrets
├─ ipsec.d
│   └─ cacerts
│       └─ certs
│           └─ private
```

- Il file `ipsec.conf`¹ specifica la maggior parte delle configurazioni e le informazioni di controllo per il sottosistema IPsec (ulteriori specifiche e sintassi sono disponibili al seguente [link](#)).
- Il file `ipsec.secrets`¹ contiene i segreti che poi verranno utilizzati nella fase di autenticazione (ulteriori specifiche al seguente [link](#)).

Certificati

Una delle principali novità che introduce IKEv2 è la possibilità di eseguire l'autenticazione tra certificati X.509. In fase di testing abbiamo preso in considerazione due tipi di certificati:

- Certificati RSA
- Certificati ECDSA

A partire da una chiave pubblica è necessario realizzare un certificato di chiave pubblica e questo richiede la chiave privata di una CA. Nel nostro caso ci siamo creati dei certificati da CA e li abbiamo distribuiti manualmente tra i due host.

Per la generazione abbiamo utilizzato il tool `pki`

CA Certificate

Partiamo con la generazione dei certificati da Certification Authority, di seguito sono riportati i due comandi da utilizzare. Ne occorrono due poichè per firmare i certificati ECDSA occorre una chiave con lo stesso schema.

```
$ pki --gen --type rsa --size 2048 --outform pem > 'ca.rsa.key.pem'
$ pki --gen --type ecdsa --size 256 --outform pem > 'ca.ecdsa.key.pem'
```

Ora utilizziamo la chiave privata per firmare il certificato di chiave pubblica.

```
$ pki --self --ca --lifetime 3650 --in 'ca.<type>.key.pem' --type <type> \
  --dn "CN=CA" --outform pem > ca.<type>.cert.pem
```

Occorre poi distribuire questi due certificati ai due host, vanno messi all'interno della directory `cacerts`.

Host Certificate

Passiamo ora a generare i certificati che gli host andranno ad utilizzare nella fase di autenticazione, occorre generare la coppia chiave privata, chiave pubblica.

```
$ pki --gen --type ecdsa --size 256 --outform pem > 'host.ecdsa.key'
$ pki --gen --type rsa --size 2048 --outform pem > 'host.rsa.key'
```

E' buona norma salvare le chiavi all'interno della directory private. Ora andiamo ad estrarre la chiave pubblica da quella appena generata e la firmiamo con la chiave delle CA del passo precedente.

```
$ pki --pub --in 'host.rsa.key' --type rsa | pki --issue --lifetime 1825 \
  --cacert 'ca.rsa.cert.pem' --cakey 'ca.rsa.key.pem' \
  --dn "CN=<Host_IP>" --san @<Host_IP> --san <Host_IP> \
  -- flag serverAuth --outform pem > 'host.rsa.cert.pem'
```

¹ Le configurazioni utilizzate si trovano in [appendice](#).

Si procede in maniera analoga con le opportune modifiche anche per il certificato ECDSA. Questi vanno poi posizionati all'interno della directory certs.

2.2.1 Mschap

Il riassunto della configurazione è mostrato in tabella, per l'initiator e il responder sono riportate le modalità della loro autenticazione.

Configuration	
<i>Initiator</i>	EAP-Mschapv2
<i>Responder</i>	RSA Certificate 2048
<i>Chipset Suite</i>	AES_CBC_128_HMAC_SHA2_256_128_DH_ECP_256

Esaminando gli scambi di IKE AUTH osserviamo che questa modalità richiede in totale 4 exchange.

2.2.2 RSA

Configuration	
<i>Initiator</i>	RSA Certificate 2048
<i>Responder</i>	RSA Certificate 2048
<i>Chipset Suite</i>	AES_CBC_128_HMAC_SHA2_256_128_DH_ECP_256

Utilizzando certificati RSA si osserva che la dimensione di un certificato eccede la dimensione massima di un pacchetto IP per tali motivi si ha la frammentazione: ovvero il contenuto, poichè eccede la dimensione massima del campo *data* viene spezzato in più pacchetti.

Anche ad esaminare il certificato, si osserva che ha una dimensione pari a 1032 byte, di cui abbiamo:

- 256 byte per la rappresentazione del modulo;
- 1 byte per la rappresentazione dell'esponente di cifratura
- 384 byte per la firma
- i restanti byte sono esaminati in [appendice](#).

Idealmente gli scambi durante IKE AUTH dovrebbero essere 2 ovvero i due si scambiano reciprocamente i certificati. Tuttavia, data la dimensione di quest'ultimi, gli scambi effettivi risultano essere in totale 4.

2.2.3 ECDSA

Configuration	
<i>Initiator</i>	ECDSA Certificate 256
<i>Responder</i>	ECDSA Certificate 256
<i>Chipset Suite</i>	AES_CBC_128_HMAC_SHA2_256_128_DH_ECP_256

Si osserva che i certificati ECDSA hanno una dimensione ridotta rispetto a quella dei certificati RSA, infatti quello utilizzato nel nostro caso ha una dimensione pari a 619 byte. Questo fa sì che non si ecceda la dimensione del payload del pacchetto IP, in questo modo la fase di IKE AUTH effettua solamente uno scambio.

3 TESTING

Il nostro lavoro si concentra sullo stimare le risorse necessarie per l'instaurazione di Security Associations (SA). In ogni sottosezione:

1. analizzeremo uno degli aspetti critici considerati;
2. descrivere come abbiamo effettuato le misurazioni;
3. i risultati ottenuti.

3.1 Analisi traffico

3.1.1 Misurazioni

Per effettuare tra i due nodi abbiamo realizzato uno shell script. Il tool principale per la realizzazione di quest'ultimo è `tcpdump`, un programma per il monitoraggio delle interfacce e la cattura dei pacchetti in entrata ed in uscita da un calcolatore. La scelta è dovuta alla precisione dei timestamp che utilizza, e alla sua diffusione (è già presente di default in molte distro linux).

Lo script iterativamente *instaura* ed *elimina* una security association, e ne cattura il traffico. Tramite questo possiamo misurare i tempi necessari per le diverse fasi del protocollo, e le dimensioni dei pacchetti. Il tool permette di specificare la suite crittografica che si vuole prendere in esame, ed il numero di tentativi da effettuare. Al termine delle iterazioni lo script fornisce delle medie dei valori che si vogliono misurare.

3.1.2 Risultati

Come specificato nella fase di configurazione si nota che il numero di scambi dipende fortemente dalla suite crittografica considerata. In particolare, il **metodo di autenticazione** è la componente che incide maggiormente. Ad esempio l'autenticazione tramite MSCHAPv2 prevede 4 scambi, mentre l'utilizzo di certificati prevede 2 scambi. Tuttavia è importante osservare che qualora il certificato ecceda la dimensione massima per il payload dei pacchetti ip, si ha frammentazione. Questo fenomeno si può osservare ad esempio con l'utilizzo di certificati generati con RSA 2048.

Di seguito riportiamo i risultati ottenuti utilizzando certificati generati rispettivamente con RSA 2048 e con ECDSA 256, data la seguente suite crittografica:

AES_128_CBC_HMAC_256_AES_128_XCBC_ECP_256.

Listing 1: Risultati ECDSA

Average time per attempt	0.0587
-----	-----
Total attempts time	02.9371
-----	-----
Average packets	4
-----	-----
Average init bytes	1365
-----	-----
Average auth bytes	1772
-----	-----
Average exchanged bytes	3137
-----	-----
Total exchanged bytes	156850

Listing 2: Risultati RSA

Average time per attempt	0.0839
-----	-----
Total attempts time	04.1999
-----	-----
Average packets	6
-----	-----
Average init bytes	1365
-----	-----
Average auth bytes	4184
-----	-----
Average exchanged bytes	5549
-----	-----
Total exchanged bytes	277450

3.2 Utilizzo risorse CPU

3.2.1 Misurazioni

Per misurare le risorse computazionali richieste da Strongswan, abbiamo utilizzato il tool `perf`. Questo permette di effettuare CPU profiling, esaminando diversi parametri della CPU per l'esecuzione di un certo comando.

In questo caso ci siamo soffermati sul numero di cicli di clock ed istruzioni per SA instaurata. Inoltre `perf` permette di misurare n esecuzioni del comando, fornendo media e varianza dei valori.

3.2.2 Risultati

Di seguito riportiamo l'esecuzione del comando `perf` per l'instaurazione di SA che utilizzano rispettivamente `RSA` e `ECDSA` considerando 3 esecuzioni.

```
Performance counter stats for '/usr/sbin/ipsec_up_base-rsa' (3 runs):
```

```
12.685.614      cycles          ( +-  6,59% )
12.383.771      instructions     ( +-  0,61% )
41.040.697 ns    duration_time    ( +- 58,59% )

0,0410 +- 0,0240 seconds time elapsed ( +- 58,59% )
```

```
Performance counter stats for '/usr/sbin/ipsec_up_base-ecdsa' (3 runs):
```

```
12.461.099      cycles          ( +-  5,87% )
12.146.504      instructions     ( +-  1,52% )
25.065.990 ns    duration_time    ( +- 71,83% )

0,0251 +- 0,0180 seconds time elapsed ( +- 71,83% )
```

3.3 Occupazione di memoria

3.3.1 Misurazioni

Per misurare la quantità di RAM necessaria per Strongswan abbiamo utilizzato il tool `pmap`, che permette di avere informazioni sulla memoria allocata per un determinato processo.

3.3.2 Risultati

In seguito a diverse misurazioni possiamo constatare che l'utilizzo di memoria per il demone **charon** si attesta tipicamente a 10MB, più in generale nell'ordine delle decine di MB, mentre per ogni SA instaurata si ha un aumento in memoria di circa 15KB, più in generale nell'ordine delle decine di KiloByte.

	Kbytes	RSS	Dirty
total kB	1128980	10692	2772

4 CONCLUSIONI

A CONFIGURATION FILE

Di seguito riportiamo i file di configurazione `ipsec.conf` e `ipsec.secrets` rispettivamente di initiator e di responder. Una possibile modifica ai file potrebbe essere quella di rendere il tutto simmetrico, allo stato attuale i due non possono scambiarsi di ruolo. Alcune note:

- la connessione **default** definisce la configurazione comune a tutte le altre.
- la connessione **secure** è quella con cui specifichiamo la chiper_suite sicura.
- **also** permette di realizzare l'ereditarietà multipla tra le connessioni.
- il parametro **auto** specifica quale operazione effettuare con la connessioni all'avvio di IPsec; il valore *add* la aggiunge alle possibile connessioni ma non cerca di stabilirla

A.1 Initiator

`ipsec.conf`

```
#####
# ipsec.conf - strongSwan IPsec configuration file
#####
conn %default
    leftsourceip=%config
    right=<ip_responder>
    rightsubnet=0.0.0.0/0
    auto=add

conn secure
    ike=aes256-sha384-ecp384!

conn base-mschap
    leftauth=eap-mschapv2
    eap_identity="<identity>"
    rightauth=pubkey

conn base-rsa
    rightauth=pubkey-rsa-2048
    leftauth=pubkey-rsa-2048
    leftcert=<path_to_cert>

conn base-ecdsa
    rightauth=pubkey-ecdsa-256
    leftauth=pubkey-ecdsa-2048
    leftcert=<path_to_cert>

conn secure-rsa
    also=base-rsa
    also=secure

conn secure-ecdsa
    also=base-ecdsa
    also=secure

conn ipsec-ike
    also=secure
    also=base-mschap
```

ipsec.secrets

```
#####
# ipsec.secrets - strongSwan IPsec configuration file
#####
<identity> : EAP "<password>"

: ECDSA "/etc/ipsec.d/private/<key>.pem"
: RSA  "/etc/ipsec.d/private/<key>.pem"
```

A.2 Responder**ipsec.conf**

```
#####
# ipsec.conf - strongSwan IPsec configuration file
#####
conn %default
    keyexchange=ikev2
    left=<ip_host>
    leftsubnet=0.0.0.0/0
    forceencaps=yes
    compress=no
    type=tunnel
    fragmentation=yes
    rekey=no
    right=<ip_initiator>
    rightid=%any
    rightsourceip=0.0.0.0/0
    rightdns=8.8.8.8,4.4.4.4
    auto=add

conn mschap
    rightauth=eap-mschapv2
    eap_identity=%identity
    leftcert=<path_to_cert>
    leftsendcert=always

conn rsa
    leftcert=<path_to_cert>
    leftauth=pubkey-rsa-2048
    rightauth=pubkey-rsa-2048

conn ecdsa
    leftcert=<path_to_cert>
    leftauth=ecdsa-256
    rightauth=ecdsa-256
```

ipsec.secrets

```
<identity> : EAP "<password>"

: RSA  "/etc/ipsec.d/private/<key>.pem"
: ECDSA "/etc/ipsec.d/private/<key>.pem"
```

B TOOLS

Per instaurare la connessione IPsec si utilizza il seguente comando.

```
$ ipsec up <conn_name>
```

Per verificare che la SA sia stata correttamente instaurata è possibile utilizzare il seguente tool `ip xfrm`, il quale consente di effettuare la trasformazione dei pacchetti. Questo fornisce un'interfaccia ai due database:

- SAD: Security Association Database, tramite l'oggetto `state`.
- SPD: Security Policy Database, tramite l'oggetto `policy`.

L'esecuzione del seguente comando fornisce una vista delle entry presenti nel SAD, possiamo poi utilizzare queste informazioni in `wireshark` per poter vedere il traffico tra i due host in chiaro.


```
$ ip xfrm state list
src <initiator> dst <responder>
  proto esp spi 0xc49d3a6d reqid 1 mode tunnel
  replay-window 0 flag af-unspec
  auth-trunc hmac(sha256) <skey> 128
  enc cbc(aes) <skey>
  anti-replay context: seq 0x0, oseq 0xc, bitmap 0x00000000
src <responder> dst <initiator>
  proto esp spi 0xca382e6d reqid 1 mode tunnel
  replay-window 32 flag af-unspec
  auth-trunc hmac(sha256) <skey> 128
  enc cbc(aes) <skey>
  anti-replay context: seq 0x0, oseq 0x0, bitmap 0x00000000
```

Wireshark

Per vedere il traffico sniffato in chiaro occorre configurare il protocollo ISAKMP all'interno di `wireshark`, andiamo a specificare quelle che sono le chiavi negoziate per l'autenticazione di messaggio e di cifratura.

- Andare su Edit->Preferences->Protocols->ISAKMP.
- Aggiungere all'interno della tabella le varie entry riportate tramite `ip xfrm`

Shell script

Di seguito è riportato lo shell script per l'analisi del traffico dei pacchetti. Non sono stati riportati gli output, lo script completo, così come tutto il materiale della tesina, sono contenuti all'interno della  [repository](#). Prima di eseguire lo script è vanno cambiati i permessi.

```
chmod +x ikev2-tester.sh
```

I **flag** da specificare al momento dell'esecuzione dello script sono i seguenti:

- `-i`: per specificare l'interfaccia su cui catturare i pacchetti con `tcpdump`
- `-s`: per specificare la suite crittografica da usare
- `-n`: per specificare il numero di tentativi da effettuare

```

#!/usr/bin/env bash

while getopts ":n:s:i:" option; do
    case $option in
        n)
            attempts="$OPTARG"
            ;;
        s)
            suite="$OPTARG"
            if [ $suite == 1 ];
            then
                conn="base-mschap"
                enc_alg="AES_128_CBC"
                auth_alg="HMAC_256"
                prf_alg="AES_128_XCBC"
                dh_alg="ECP_256"
                auth_method="EAP_MSCHAPv2"
            elif [ $suite == 2 ]
            then
                conn="base-rsa"
                enc_alg="AES_128_CBC"
                auth_alg="HMAC_256"
                prf_alg="AES_128_XCBC"
                dh_alg="ECP_256"
                auth_method="X.509_RSA_2048_Certificate"
            elif [ $suite == 3 ]
            then
                conn="base-ecdsa"
                enc_alg="AES_128_CBC"
                auth_alg="HMAC_256"
                prf_alg="AES_128_XCBC"
                dh_alg="ECP_256"
                auth_method="X.509_ECDSA_256_Certificate"
            else
                $(echo "Suite_must_be_1_or_2...")
                exit 1
            fi
            ;;
        i)
            interface="$OPTARG"
            $(sudo timeout --preserve-status 0.5 tcpdump -ni $interface > /dev/
                null 2>&1 || $(echo "Supplied_interface_doesn't_exist!" && exit)
                )
            ;;
        *)
            echo "Usage:_$0_[-n_number_of_attempts]_[-s_suite]_[-i_interface]"
            exit 1
            ;;
    esac
done

if [ -z $suite ] || [ -z $interface ]
then
    echo "Set_all_flags!"
    echo "Usage:_$0_[-n_number_of_attempts]_[-s_suite]_[-i_interface]"
    exit 1
fi

cumulative_time=0
cumulative_packets=0

```

```

cumulative_size=0
cumulative_init_size=0
cumulative_auth_size=0
sudo ipsec down $conn > /dev/null
sleep 3

for (( att=1; att<=$attempts; att++ ))
do
    echo "Starting_attempt_$att"
    echo "Waiting_to_be_sure_connection_is_closed!"
    sleep 4
    echo "Wait_is_over!"
    echo "Dumping_on_port_500" && sudo timeout --preserve-status 5 tcpdump --
        immediate-mode -ni $interface -tt -l -w inittmpbuffer1 -Z root port 500
        >/dev/null 2>&1 && echo "Dump_on_500_over" &
    echo "Dumping_on_port_4500" && sudo timeout --preserve-status 5 tcpdump --
        immediate-mode -ni $interface -tt -l -w authtmpbuffer1 -Z root port 4500
        >/dev/null && echo "Dump_on_4500_over" && sleep 1 && echo "Closing_SA!"
        && sudo ipsec down $conn > /dev/null &
    sleep 2 && echo "Establishing_SA!" && sudo ipsec up $conn > /dev/null
    echo "Waiting_now!"
    sleep 10
    echo "Waiting_is_over!"
    sudo tcpdump -tt -Z root -n -vvv -e -r inittmpbuffer1 > inittmpbuffer2 2>&1
    sudo tcpdump -tt -Z root -n -vvv -e -r authtmpbuffer1 > authtmpbuffer2 2>&1
    grep -B 1 "isakmp" authtmpbuffer2 > authtmpbuffer3
    grep -B 1 "isakmp" inittmpbuffer2 > inittmpbuffer3
    init_packets=$(grep -Eo "[0-9]+" inittmpbuffer2 | wc -l | grep -Eo "[0-9]+")
    auth_packets=$(grep -Eo "[0-9]+" authtmpbuffer3 | wc -l | grep -Eo "[0-9]+")

    init_start_time=$(grep -Eo "[0-9]+.[0-9]+" inittmpbuffer3 | head -n 1)
    init_end_time=$(grep -Eo "[0-9]+.[0-9]+" inittmpbuffer3 | tail -n 1)
    auth_start_time=$(grep -Eo "[0-9]+.[0-9]+" authtmpbuffer3 | head -n 1)
    auth_end_time=$(grep -Eo "[0-9]+.[0-9]+" authtmpbuffer3 | tail -n 1)
    init_time=$(echo "scale=6;_${init_end_time}_${init_start_time}" | bc)
    auth_time=$(echo "scale=6;_${auth_end_time}_${auth_start_time}" | bc)
    attempt_time=$(echo "scale=6;_${auth_end_time}_${init_start_time}" | bc)
    attempt_packets=$(( init_packets + auth_packets ))
    cumulative_packets=$(( cumulative_packets + attempt_packets ))
    cumulative_time=$(echo "scale=6;_${cumulative_time}_${attempt_time}" | bc)
    init_size=$(grep -Eo "length_[0-9]:" inittmpbuffer3 | grep -Eo "[0-9]+" |
        awk '{SUM+=_$_}END{print_SUM}')
    auth_size=$(grep -Eo "length_[0-9]:" authtmpbuffer3 | grep -Eo "[0-9]+" |
        awk '{SUM+=_$_}END{print_SUM}')
    attempt_size=$(( init_size + auth_size ))
    cumulative_size=$(( cumulative_size + attempt_size ))
    cumulative_init_size=$(( cumulative_init_size + init_size ))
    cumulative_auth_size=$(( cumulative_auth_size + auth_size ))
    sudo rm inittmpbuffer1 inittmpbuffer2 inittmpbuffer3 authtmpbuffer1
        authtmpbuffer2 authtmpbuffer3
done

average_time=$(echo "scale=6;_${cumulative_time}/${attempts}" | bc)
average_init_size=$(( cumulative_init_size / attempts ))
average_auth_size=$(( cumulative_auth_size / attempts ))
average_size=$(( cumulative_size / attempts ))
average_packets=$(( cumulative_packets / attempts ))

```

perf

Per installare `perf` sulle macchine debian create eseguire il seguente comando.

```
$ sudo apt-get install linux-perf
```

Il comando potrebbe dare errore a causa dei permessi, per risolvere questo problema basta sovrascrivere il contenuto del file `/proc/sys/kernel/perf_event_paranoid` con il valore 3.

C CERTIFICATI

Andiamo a vedere a cosa è dovuta la dimensione dei certificati, per vedere il contenuto del certificato sotto forma di output testuale utilizzare il seguente comando.

```
$ openssl x509 --in <cert> -text
```

Andiamo a vedere nello specifico il contenuto delle due tipologie di certificati utilizzate per la sperimentazione:

- ECDSA
- RSA

La differenza principali tra i due sta nella dimensione della chiave che nel nostro caso è di fondamentale importanza, in quanto evita la frammentazione del pacchetto. Anche se fa uso di chiavi da 256 bit ECDSA garantisce un livello di sicurezza pari a 2^{256} .

c.1 RSA Certificate

```
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 3952640834610742420 (0x36da99e5a7ad4494)
Signature Algorithm: sha384WithRSAEncryption
Issuer: CN = <info>
Validity
    Not Before: May 29 08:42:06 2023 GMT
    Not After : May 27 08:42:06 2028 GMT
Subject: CN = <info>
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (4096 bit)
        Modulus:
            00:c5:7d:50:95:2c:c3:42:32:b1:b8:1f:55:00:94:
            ---
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Authority Key Identifier:
        keyid:99:C3:D7:54:F4:40:EC:DE:9C:7C:60:DC:ED:29:60:BF:75:B6:94:30

    X509v3 Subject Alternative Name:
        DNS:192.168.122.145, IP Address:192.168.122.145
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, 1.3.6.1.5.5.8.2.2
Signature Algorithm: sha384WithRSAEncryption
18:e9:7c:2b:ea:2f:2c:2b:a6:d4:bd:6c:94:63:41:29:f9:45:
---
```

Come possiamo osservare dall'output sono contenute numerose informazioni che dunque aumentano notevolmente la dimensione del certificato e quindi che portano alla frammentazione di quest'ultimo durante la fase di IKE AUTH.

c.2 ECDSA

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 6875679331162392113 (0x5f6b51ec3b6e0631)

Signature Algorithm: ecdsa-with-SHA256

Issuer: CN = CA ECDSA

Validity

Not Before: May 29 14:17:10 2023 GMT

Not After : May 27 14:17:10 2028 GMT

Subject: CN = 192.168.122.171 ECDSA

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:21:d7:c7:a0:6f:fd:13:1a:1e:f4:c6:5b:5c:88:

5c:99:3e:bf:92:89:7c:b2:0d:44:d0:9a:c7:aa:c3:

0b:fe:4a:75:3a:ca:7b:91:ee:1b:69:e7:4f:40:06:

e1:27:ee:62:72:eb:f7:06:30:c6:47:ae:db:01:e4:

36:62:12:3e:92

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Authority Key Identifier:

keyid:1A:12:82:AD:18:CF:85:0A:24:03:32:DC:D7:10:26:92:15:14:00:F9

X509v3 Subject Alternative Name:

DNS:192.168.122.171, IP Address:192.168.122.171

X509v3 Extended Key Usage:

TLS Web Server Authentication

Signature Algorithm: ecdsa-with-SHA256

30:44:02:20:62:aa:81:67:fe:b7:2e:2f:13:f9:69:d4:6c:72:

7e:a9:62:6a:db:7a:1b:af:35:b7:42:dc:42:fc:11:95:fa:d7:

02:20:33:6f:7f:6b:a8:c4:c1:33:0e:04:7b:2f:99:14:85:ff:

93:78:9c:ed:5d:84:58:61:76:d8:4d:b7:24:07:bd:b2

REFERENCES

- [1] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, October 2014.