IKEv2 and Smart Objects

(Tero Kivinen <kivinen@iki.fi>)

1.0 Introduction

This document tells what minimal IKEv2 implementation could look like. Minimal IKEv2 implementation only supports initiator end of the protocol, and only supports the initial IKE_SA_INIT and IKE_AUTH exchanges and does not initiate any other exchanges, and replies with empty (or error) message to all incoming requests.

This means that most optional features of IKEv2 are left out: NAT Traversal, IKE SA rekey, Child SA Rekey, Multiple Child SAs, Deleting Child / IKE SAs, Configuration payloads, EAP authentication etc

Some optimizations can be done because of this selection of supported features. Those optimizations are specifically pointed out below.

2.0 Initial exchange

All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange", and is sometimes called a "request/response pair". Every request requires a response. For every pair of IKE messages, the exchange initiator is responsible for retransmission in the event of a timeout.

IKE is a reliable protocol: the initiator MUST retransmit a request until it either receives a corresponding response or deems the IKE SA to have failed.

The minimal implementation of IKEv2 only uses first 2 exchange of messages called IKE_SA_INIT and IKE_AUTH which create the IKE SA and the first child SA. In addition to those messages minimal IKEv2 implementation need to understand CREATE_CHILD_SA request so it can reply with CREATE_CHILD_SA error response saying NO_ADDITIONAL_SAS to it, and understand INFORMATIONAL request so much, it can reply with empty INFORMATIONAL response to it.

All messages following the IKE_SA_INIT exchange are cryptographically protected using the cryptographic algorithms and keys negotiated in the IKE_SA_INIT exchange.

Every IKE message contains a Message ID as part of its fixed header. This Message ID is used to match up requests and responses, and to identify retransmissions of messages.

Minimal implementation need only support of being initiator, so it does not ever need to send any other request as one IKE_SA_INIT,

and one IKE_AUTH message. As those messages have fixed Message IDs
(0 and 1) it does not need to keep track of its own Message IDs
after that.

As all incoming requests are just repied to, but not processed in
other ways, there is no need to protect against replay attacks.
This means minimal implementation can always answer to request
coming in with same Message ID than what the request had, and then
forget the request/response pair immediately.

In the following descriptions, the payloads contained in the
message are indicated by names as listed below.

Notation	Payload
AUTH	Authentication
CERTREQ	Certificate Request
HDR	IKE header (not a payload)
IDi	Identification - Initiator
IDr	Identification - Responder
KE	Key Exchange
Ni, Nr	Nonce
N	Notify
SA	Security Association
SK	Encrypted and Authenticated
TSi	Traffic Selector - Initiator
TSr	Traffic Selector - Responder

The initial exchanges are as follows:

```
Initiator                       Responder
-------------------------------------------------------------------
HDR(SPIi=xxx, SPIr=0, IKE_SA_INIT,
   Flags: Initiator, Message ID=0),
   SAi1, KEi, Ni  -->

              <--  HDR(SPIi=xxx, SPIr=yyy, IKE_SA_INIT,
                     Flags: Response, Message ID=0),
                    SAr1, KEr, Nr, [CERTREQ]
```

HDR contains the Security Parameter Indexes (SPIs), version
numbers, and flags of various sorts. Each endpoint chooses one of
the two SPIs and MUST choose them so as to be unique identifiers of
an IKE SA. An SPI value of zero is special: it indicates that the
remote SPI value is not yet known by the sender.

The SAi1 payload states the cryptographic algorithms the initiator
supports for the IKE SA. The KEi and KEr payload contain
Diffie-Hellman values and Ni and Nr are the nonces. The SAr1
contains chosen cryptographic suite from initiator's offered

choices. Minimal implementation using shared secrets will ignore
the CERTREQ payload.

Minimal implementation will most likely support exactly one set of cryptographic algorithms, meaning the SAi1 payload will be static. It needs to check that the SAr1 received matches the proposal it sent.

At this point in the negotiation, each party can generate SKEYSEED, from which all keys are derived for that IKE SA.

The keys used for the encryption and integrity protection are derived from SKEYSEED and are known as SK_e (encryption) and SK_a (authentication, a.k.a. integrity protection). A separate SK_e and SK_a is computed for each direction. The keys used to protect messages from the original initiator are SK_ai and SK_ei. The keys used to protect messages in the other direction are SK_ar and SK_er. The notation SK { ... } indicates that these payloads are encrypted and integrity protected using that direction's SK_e and SK_a.

```
HDR(SPIi=xxx, SPIr=yyy, IKE_AUTH,
   Flags: Initiator, Message ID=1),
   SK {IDi, AUTH, SAi2, TSi, TSr,
      N(INITIAL_CONTACT)}  -->

              <--  HDR(SPIi=xxx, SPIr=yyy, IKE_AUTH, Flags:
                     Response, Message ID=1),
                    SK {IDr, AUTH, SAr2, TSi, TSr}
```

The initiator asserts its identity with the IDi payload, proves knowledge of the secret corresponding to IDi and integrity protects the contents of the first message using the AUTH payload. The responder asserts its identity with the IDr payload, authenticates its identity and protects the integrity of the second message with the AUTH payload.

As minimal implementation usually has only one host where it connects, and that means it has only one shared secret. This means it does not need to care about ID payloads that much. If the other end sends AUTH payload which initiator can verify using the shared secret it has, then it knows the other end is the peer it was configured to talk to.

In the IKE_AUTH initiator sends SA offer(s) in the SA payload, and the proposed Traffic Selectors for the proposed Child SA in the TSi and TSr payloads. The responder replies with the accepted offer in an SA payload, and selected Traffic Selectors. The selected Traffic Selectors may be a subset of what the initiator proposed.

In the minimal implementation both SA payloads and TS payloads are going to be mostly static. The SA payload will have the SPI value used in the ESP, but the algorithms are most likely going to be the one and only supported set. The TS payloads on the initiator end will most likely say from any to any, i.e. full wildcard ranges, or from the local IP to the remote IP. In the wildcard case the server quite often narrow the range down to the one IP address pair. Using single IP address pair as a traffic selectors when sending IKE_AUTH will simplify processing as then server will either accept that pair or return error. If wildcard ranges are used, there is possibility that server narrows the range to some other range than what is used by the minimal implementation.

The IKE_AUTH (and IKE_SA_INIT) may contain multiple status notification payloads which can be ignored by minimal implementation. There can also be Vendor ID, Certificate, Certificate Request or Configuration payloads, but those can be ignored by the minimal implementation doing shared secret authentication.

When using shared secret authentication, the peers are authenticated by having each calculating a MAC over a block of data.

The initiator might also get response back having notification payload with error code inside. As that error code will be unauthenticated and make be faked, there is no need to do anything for those. Minimal implementation can simply ignore those and retransmit its request until it times out and if that happens then the IKE SA (and Child SA) creation failed.

Responder might also reply with IKE_AUTH response packet which do not contain payloads needed to set up Child SA (SAr2, TSi and TSr), but contains AUTH payload and an error. As minimal implementation probably do not support multiple SAs nor sending the CREATE_CHILD_SA exchanges the IKE SA is useless for initiator. It can delete the IKE SA and start over from the beginning (which might fail again if this is configuration error, or it might succeed if this was temporal failure).

2.1 Other exchanges

Minimal implementation needs to be able to reply to INFORMATIONAL request by sending empty reply back:

```
Initiator                    Responder
-----------------------------------------------------------------
               <--  HDR(SPIi=xxx, SPIr=yyy, INFORMATIONAL,
                      Flags: none,  Message ID=m),
                    SK {...}
```

```
HDR(SPIi=xxx, SPIr=yyy, INFORMATIONAL,
   Flags: Initiator I Response,
   Message ID=m),
   SK{} -->
```

Minimal implementation also needs to be able to reject CREATE_CHILD_SA exchanges by sending NO_ADDITIONAL_SAS error notify back:

```
Initiator                  Responder
-----------------------------------------------------------------
             <--  HDR(SPIi=xxx, SPIy=yyy, CREATE_CHILD_SA,
                    Flags: none, Message ID=m),
                   SK{...}

HDR(SPIi=xxx, SPIr=yyy, CREATE_CHILD_SA,
   Flags: Initiator I Response, Message ID=m),
   SK{N(NO_ADDITIONAL_SAS)} -->
```

3.0 Conclusions

Minimal IKEv2 implementation can omit lots of complex IKEv2 optional features, and get IKEv2 implementation that only supports initiating one ESP to known end point. Such implementation can be written in very little code, in our tests we wrote two separate implementations of minimal IKEv2 implementations using two different high level languages (perl and python) in less than 1000 lines of source code (both implementations also supported minimal ESP and ICMP stack to be able to send out one ping packet and parse the reply to verify that the key material was generated correctly).

Minimal IKEv2 implementation is usable in settings where device is sending out periodic packets like sensor data, or sending packet based on the user actions (pressing light switch). In such setups the device creates IKEv2 SA, IPsec SA and then sends one ESP packet (or stream of ESP packets) to the known destination. The communication is always initiated from the device, and there is no requirement for it to be able to respond to the any communication attempts initiated to its direction.

Initial configuration of the device can be handled in the same way, i.e. when the device is first time booted up it uses some mechanism to find the local security gateway and then connects to that gateway using fixed shared secret to fetch initial configuration.