State Machines in an IKEv2 implementation

Stjepan Groš, Domagoj Jakobović, Leonardo Jelenković
Faculty of Electrical and Computing Engineering
University of Zagreb
Unska bb, 10000 Zagreb, Croatia
E-Mail: {stjepan.gros, domagoj.jakobovic, leonardo.jelenkovic}@fer.hr

***Abstract*—State machines are very important aspect of network protocol description, implementation and verification. In this paper we describe state machines used to implementat behavior of IKEv2 protocol.**

***Index Terms*—IKEv2, state machines**

I. Introduction

The essence of any protocol of even moderate complexity is it's behavior. There are multiple ways that the behavior of an protocol can be specified, but the most precise one is via state machines. State machine description, while very hard to understand by humans, allows formal reasoning about the properties of the protocol as well as fast implementation by mapping description into appropriate programming language.

In this paper we describe state machines as implemented in ikev2[1], an IKEv2[2] protocol implementation. It should be noted that this particular implementation has the following state machines:

- IKE SA state machine
- CHILD SA state machine
- DHCP client behaviour
- EAP state machines

Of all of them, we do not describe EAP state machines since our implementation of IKEv2 protocol doesn't implement them directly but takes the WPA_supplicant implementation, which in turn is described in a [3].

The purpose of descriptions in this paper are twofold. The first one is to describe how is our ikev2 implementation structured and how it works. The other purpose is to allow more implementations to be built on experiences we learned during development.

The paper is structured as follows. Fist, in the Section II we describe preliminaries necessary for the rest of the paper, i.e. basic terminology from the IKEv2 specification, operating environment of ikev2 implementation, and notation used to describe state machines and all the transitions. Then, in Section III we describe state machines. Section IV outlines implementation of state machines in ikev2. The paper finishes with conclusions and future work in Section V and bibliography.

II. Preliminaries

A. Basic terminology

IKE SA, CHILD SA, Initiator, Responder
IRAC
IRAS
SAD
SPD
acquire
pfkey
message
udp, port 500, port 4500
request
response
exchage
payload
notify payloads

B. Operating environment

Ovdje treba mozda staviti opis ike-a i okoline, tj. odakle dolaze dogadaji

C. Notation for describing state machines

This section describes notation we use throughout the paper to depict and describe state machines.

States are shown in ellipses with the name of the state written inside the ellipse. Transitions between the states are shown with arrows with attached label. Labels are of the form <event>/<action> meaning that this particular transition happens when *event* occures and that, during transition, *action* is performed. The event will usually be some packet received from the network, or notification from the kernel, while the action will usually be a message sent to a network, or to the kernel.

Sometimes <event> will be of the form N(*something*) and simetimes simply as *something*. For example,

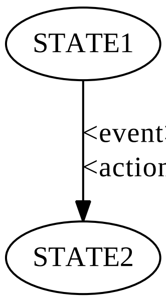

Fig. 1. Example states and transition

N(AUTH_FAILED) and AUTH_FAILED. The difference is that N() denotes that we received notify, and without N() it means that the error condition occured localy. To take given example further, N(AUTH_FAILED) denotes that we received notify about failed notification on the peer, while AUTH_FAILED means authentication failed localy.

III. State machines description

As we already mentioned in the introduction section, the ikev2 implements three state machines, each of which is described in the following subsections. The first one is IKE SA state machine which governs establishment, maintenance and termination of IKE SA. For the clarity, the state machine that governs IKE SA behavior is divided into four parts, i.e. the Initiator role, Responder and two common parts, one for sending requests and one for sending replies. Each of those parts is described separately in the following subsesctions.

Then, we describe CHILD SA state machines that implement behavior of CHILD SAs within IKE SA. Finally, when ikev2 provides configuration data to VPN clients, it can use DHCPv4 provider and, in order to do that, has to implement specific state machine, different than client's or relay's usually found in DHCPv4 implementations.

In all the descriptions we use the same pattern. First, we describe successful exchange, and at the end we enumerate and describe error conditions that can occur.

A. IKE SA Initiator's state machine

Initiator's state machine, without error transitions, is shown in Figure 2. The state machine is instantiated for each IKE SA, i.e. when new security associtation has to be established or when reauthentication is performed.

Immediately after instantiating state machine it is placed into SMI_INIT state. In this state memory for the necessary structures is reserved and initialized, and configuration data for the given peer is found. If the NAT traversal is enabled, then hash of all the source IP addresses is calculated. Finally, IKE_SA_INIT request is constructed and sent to the peer and the state machine advances into SMI_AUTH state.

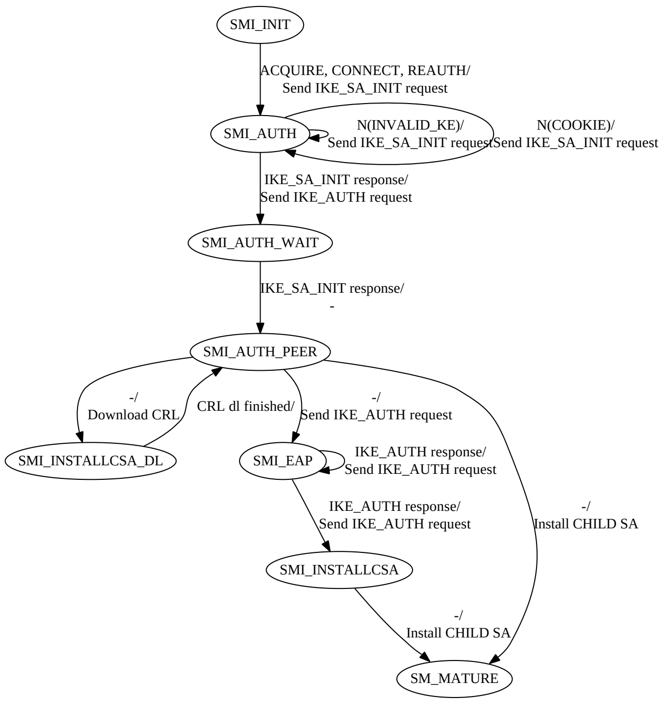

Fig. 2. Initiator state machine

In SMI_AUTH state we are waiting for IKE_SA_INIT response that will secure the channel communication using encription and integrity protection with keys negotiated in the IKE_SA_INIT exchange. After that we are certain that we are securily communicate with the peer, though, we are do not know yet if the peer is the intended one.

Responder can send three different responses. The first one is COOKIE notify meaning that it's in DoS protection mode and tries to validate our IP address. In that case we repeat IKE_SA_INIT request with COOKIE included. The second response that we can receive is INVALID_KE notify. In that case, we receive proposed DH group by responder, and if it's allowed by the local policy, we repeat the request. Finally, the third response we can receive is *regular* IKE_SA_INIT response.

When we receive regular IKE_SA_INIT response we know what cryptographyc algorithms peer selected and we are able to generate keys necessary for IKE SA protection and encryption. We also check NAT-T payload to determine if there is a NAT between the two peers. If there is, then all the further communication is perfomed over port 4500 instead of the default port 500. Additionally, if we are behind NAT, then we periodically send keepalive packets in order for NAT to keep it's bindings alive.

The next step is to prepare IKE_AUTH message. We search through configuration file to find appropriate sections that define our behavior with respect to a given peer. From the found sections we take traffic selectors and proposals for the CHILD SA. The key part in the process is to generate AUTH payload, but it is only performed in the case shared keys or certificates are used for the authentication. After sending IKE_AUTH request state machine transitions into SMI_AUTH_WAIT state. In that state we are waiting for the IKE_AUTH response.

After receiving IKE_AUTH response, we perform peer authentication. In case peer uses certificates to authenticate there is possibility that we need either CRL, or certificate itself if we were given URL where to obtain certificate along with it's hash value. Until those are obtained, state machine transitions into SMI_INSTALLCSA_DL and returns when appropriate certificate material is successfuly obtained.

Upon successful authentication, if initiator used shared keys or certificates, then CHILD SA is installed into the SAD and state machine transitions into SM_MATURE state.

When EAP is used to authenticate to a peer we go through few more states. The EAP authentication is based on a series of EAP requests and responses. RADIUS server sends EAP requests to responder. Responder places EAP request into IKE_AUTH response. Initiator receives this message, unpacks EAP request and based on it constructs EAP response which is sent to the responder in a new IKE_AUTH request. During this circular behavior of sending EAP responses and EAP requests Initiator's state machine is constantly in SMI_EAP state.

The process of authentication finishes when RADIUS server sends *EAP Success* message. At that point state machine makes transition into SMI_INSTALLCSA state and simultaniously sends final IKE_AUTH request. This final request is used to prove knowledge of MSK by both the initiator and the responder in order to prevent MITM attack.

It should be noted that in cases when EAP supplicant is asynchronous it might be necessary to introduce additional state that will be used to wait for responses from supplicant.

When Responder sends it's reponse, it contains AUTH payload as well as selected proposal and traffic selectors for the CHILD SA. Based on the received data Initiator install CHILD SA into the SAD and makes transition into MATURE state.

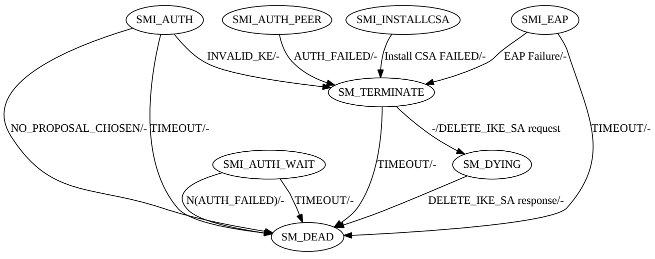

Fig. 3. Error transitions in the Initiator state machine

Transitions caused by errors are shown in Figure 3. There are two major reactions to all the possible errors. The first one is simple removal of state machine, i.e. state SM_DEAD. This state is used when initiator doesn't need to send error notify to responder, e.g. if timeout has occured so responder is unavailable, or when we conclude that responder tried to do something illegal and in that case we assume we are attacked and thus, we ignore responder.

The second reaction is termination of session. It involves sending DELETE notify to responder (from state SM_TERMINATE) and waiting for the reponse in the state SM_DYING. When we receive the response, or if timeout occured, we transition into state SM_DEAD and state machine should be removed.

B. IKE SA Responder's state machine

Responder has more complex functionality than Initiator, and thus it has larger state machine to govern it's behavior. It is shown in the Figure 4. Again, this state machine is initiated for each Initiator that Responder communicates with. What is not shown is the DoS protection mode. This mode is activated *before* state machine is created since, by definition, it's purpose is *not* to create any state before validity of the request is confirmed.

Everything starts with the reception of valid IKE_SA_INIT request. At that moment the state machine is created and placed into SMR_INIT state. Then, based on the received request all the necessary crypto algorithms are selected based on the set proposed by the Initiator. Additionaly necessary crypto material is generated (e.g. DH value, noces) that allows crypto keys to be generated. Then, IKE_SA_INIT response is generated and sent to the initiator. The state machine advances into SMR_AUTH state where IKE_AUTH request is expected.

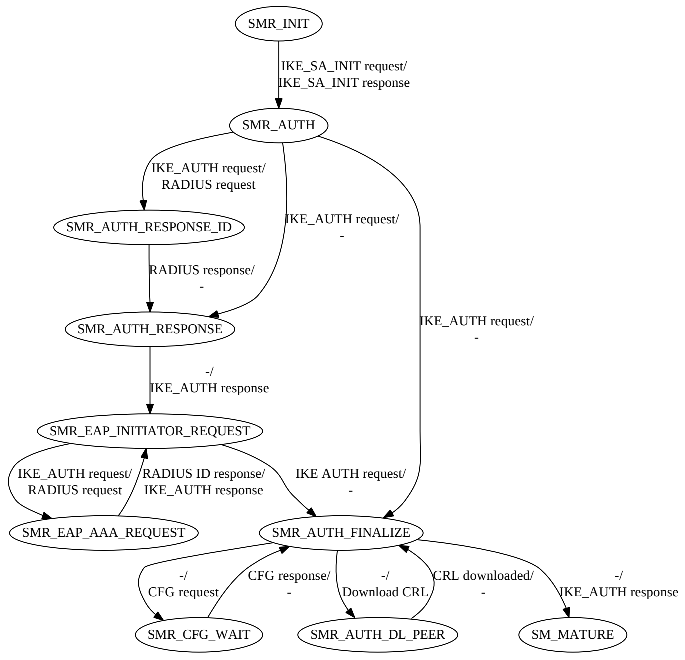

Fig. 4. Responder state machine

After IKE_AUTH request is received we have three transitions that can be followed, based primary wether Initiator requested EAP authentication or not. In a case a certificate or a shared key authentication has been requested, the received authentication material is checked and if it's valid then responder transitions into SMR_AUTH_FINALIZE state where further processing is performed and response is generated.

In the case of the EAP authentication by the Initiator we have to send EAP identity to the backend server as the first EAP response. What ikev2 does is that it checks received IKEv2 ID payload. If it's mail address then this is used as EAP ID and request is sent to the RADIUS server. The state machine simultaniously transitions into the state SMR_AUTH_RESPONSE_ID. There, we wait for RADIUS's response and, after receiving it, we go into SMR_AUTH_RESPONSE state. Otherwise, if we do not know EAP identity of the Initiator, we internally generate EAP Identity request and make transition into SMR_AUTH_RESPONSE state.

In the SMR_AUTH_RESPONSE state we prepare Responder's authentication material to be sent to the Initiator in order to authenticate Responder. IKE_AUTH response is then sent with the embedded authentication material (and appropriate certificate if necessary) as well as the first EAP request. At the same time transition is made into SMR_EAP_INITATOR_REQUEST state.

At that point state SMR_EAP_AAA_REQUEST and SMR_EAP_INITATOR_REQUEST are traversed as long as EAP authentication doesn't finish by receiving EAP Success message from the RADIUS server, and finally, sending that message to the Initiator. This positive response is received when state machine is in the SMR_EAP_AAA_REQUEST state. At that moment we have to prepare MSK, or it's replacement in case MSK is not generated as part of the EAP authentication, and then make final transition into SMR_EAP_INITATOR_REQUEST state while at the same time sending IKE_AUTH response with the final EAP message. After receiving new IKE_AUTH request from the Initiator, this time the message wan't contain embedded EAP payload, but only the final AUTH payload used to prove knowledge of MSK by the Responder. At the moment of the reception of this last AUTH payload, the state machine makes transition into SMR_AUTH_FINALIZE state.

In SMR_AUTH_FINALIZE state authentication data of the Initiator has to be verified, no matter what authentication method was used. The difference is only in the parameters, e.g. certificates, shared keys, or MSK in case of EAP, but the authentication method is the same in all three cases. There are a number of possible transitions in this state. First, we might transition into SMR_AUTH_DL_PEER state where we wait for the CRL or certificate to be downloaded. Certificates are not necessary in case of the EAP authentication and shared key authentication by the Initiator, so this state won't be used in those cases.

Next, if the client requested configuration data, then we request it from the configuration provider and wait for the response in the state SMR_CFG_WAIT. In case configuration provider works synchronously with the ikev2, i.e. it is embedded into the ikev2 itself, then no additional state is necessary and this transition doesn't happen.

Final transition that will happen is when we send final IKE_AUTH response message to the Initiator. In that case we also make transition into SM_MATURE state meaning that IKE SA and first CHILD SA are fully established. Still, in order for IKE_AUTH to be sent to Initiator parameters for the CHILD SA have to be selected and generated, i.e. traffic selectors, crypto algorithms, and crypto keys.

Error transitions in Responder's state machine are shown on Figure 3. We distinguish two types of errors. One group prevents establishment of IKE SA, and other prevent CHILD SA establishment. All the errors that prevent IKE SA from beeing established lead

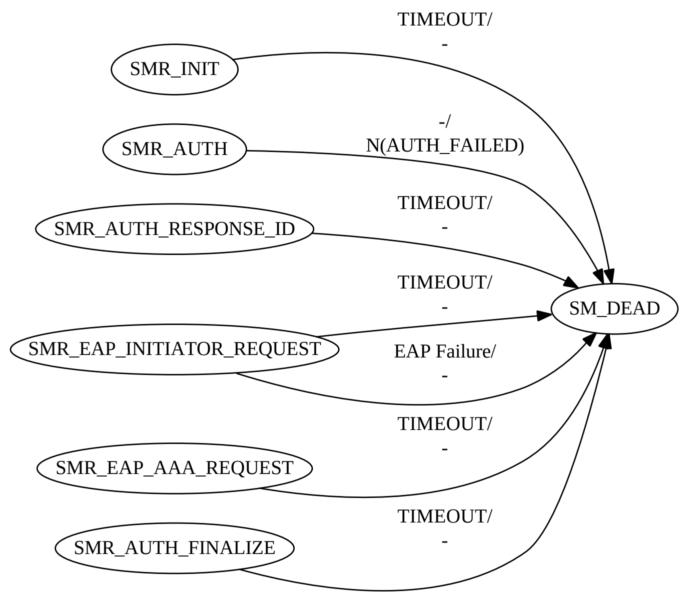

Fig. 5. Error transitions in the Responder state machine

to SRM_DEAD state, while those preventing CHILD SA establishment lead to the SMR_AUTH_FINALIZE state. On the given figure, only those errors that prevent IKE SA establishment are shown because in the second case the only difference is in the logic of SMR_AUTH_FINALIZE state that should skip preparation of CHILD SA parameters to be sent back to Initatiator, and also, doesn't have to install anything into SAD.

TIMEOUT condition occurs when Initiator doesn't issue new request within a specified time. In that case, state machine is advanced into SM_DEAD state and removed from the memory. Authentication failure can occur beacuse of the number of reasons:

1) no local configuration data
2) unsupported authentication method requested by the Initiator
3) no authentication material for the Responder
4) authentication of the Initiator failed
5) CRL/Certificate download failed

C. Common IKE SA state machine

As we already mentioned in the introduction of this section, common part are the states that Initiator and Responder enter upon successful IKE SA establishment. For the clarity this part of the state machine is broken down into two parts, the requestor and responder's part. The more complex part, Figure 6, shows transitions for all the requests that can be made by the implementation.

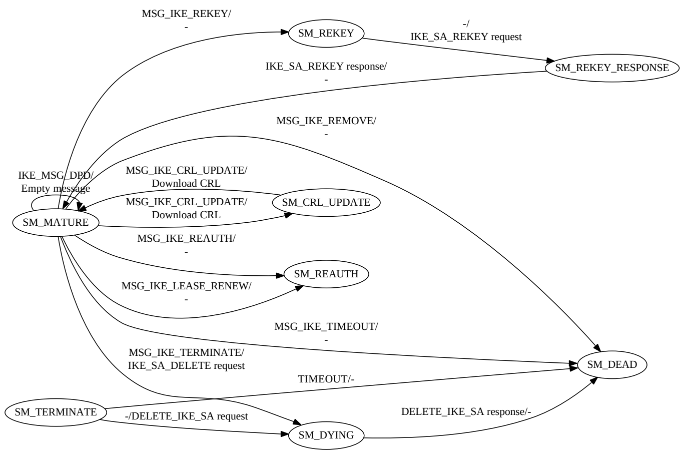

Fig. 6. IKE SA common state machine for requests

TABLE I
INTERNAL MESSAGES

MSG_IKE_REKEY	Rekey IKE SA
MSG_IKE_DPD	Dead peer detection process
MSG_IKE_CRL_UPDATE	Download CRL
MSG_IKE_REAUTH	Start IKE SA reauthentication
MSG_IKE_TERMINATE	Delete IKE SA by sending DELETE notify
MSG_IKE_LEASE_RENEW	Start renewal of lease time

For all the requests that the ikev2 issues, first internal message is sent that is processed inside the thread pool. This message causes the state machine to transition from the SM_MATURE state. From there, the request is sent and state machine makes transition into another state where it waits for the response. Note that this design makes state machine able to process only one request at a time that acts upon the IKE SA state machine.

Messages and their purpose are listed in the Table I.

The response part is much simpler because the protocol is based on simple requests and responses so it is not necessary to keep any additional state. In other words, when we receive request we immediatelly act upon it, send response, and we are ready for a new request. This design allows us to receive and process multiple requests in parallel. Still, there is one transition involved, as shown in the Figure 7. It's the case when we receive request to delete IKE SA. In that case, after sending response, the state machine makes transition into SM_DEAD and it's removed from the memory.

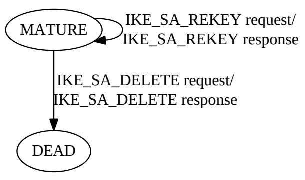

Fig. 7. IKE SA common state machine for responses

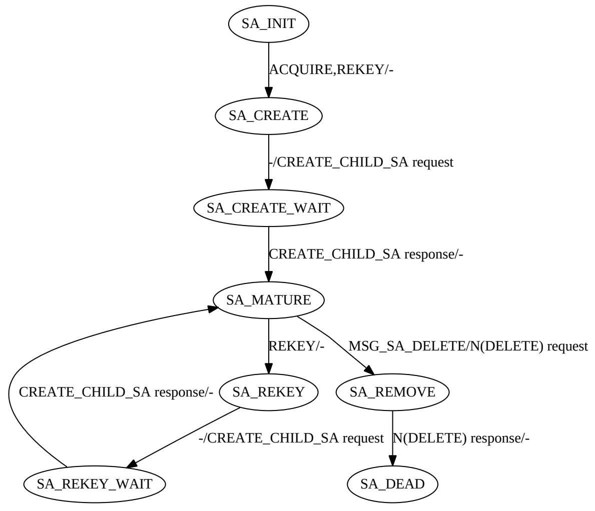
SA_INIT
ACQUIRE,REKEY/-
SA_CREATE
-/CREATE_CHILD_SA request
SA_CREATE_WAIT
CREATE_CHILD_SA response/-
SA_MATURE
REKEY/-
MSG_SA_DELETE/N(DELETE) request
CREATE_CHILD_SA response/-
SA_REKEY
SA_REMOVE
-/CREATE_CHILD_SA request
N(DELETE) response/-
SA_REKEY_WAIT
SA_DEAD

Fig. 8. CHILD SA state machine

D. CHILD SA state machine

All the CHILD SAs established during IKE SA have also their state machine, though it is much simpler than the state machine for IKE SA. It is shown in Figure 8.

E. DHCP client state machine

ikev2, in the role of the Responder, uses DHCPv4 server as one of the providers for the configuration data to be sent to the Initiator. The state machine that DHCPv4 client uses is proposed in [4]. The problem with the given state machine is that ikev2 is not actually a regular DHCPv4 client, but it's also not a regular relay. So, the modifications made are related to the way ikev2 works. It behaves as a client since it has to obtain IP configuration parameters and to take care of lease times. On the other hand, it obtains IP addresses for IRAC, so it bahaves as a relay in order for DHCPv4 server to be aware of a real client. As a result, the state machine from [4] is extended with some retransmission mechanisms and two additional states, while states INIT-REBOOT, REBOOTING and REBINDING are removed. This is shown in the Figure 9.

States ERROR and FINISHED are besically the same, but they give the information on how the state machine finished it's operation, with error or orderly.

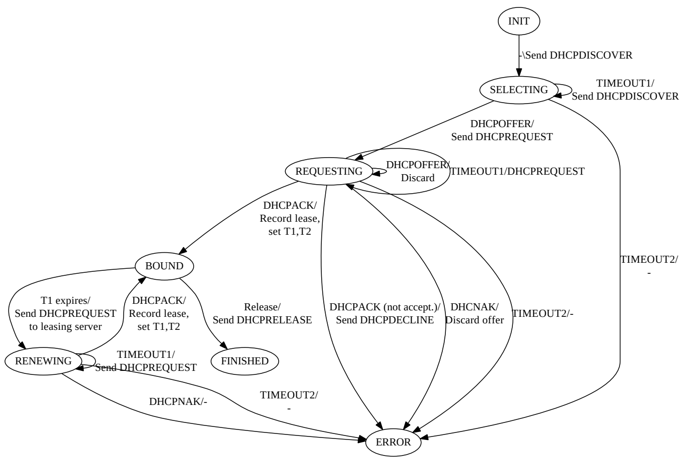

Fig. 9. Modified DHCPv4 state machine

When created, state machine is initially in the state INIT. Immediately from that state it transitions to state *SELECTING* while in the same time *DHCPDISCOVER* is sent to the DHCP server.

During the design phase, there was a question what to do in case timeouts occur because DHCP server isn't responding. There are three possibilities, immediately report an error, try few more times and then report an error, or try another DHCP server. The first variant, i.e. when we immediately report an error, is problematic because DHCP uses UDP that can loose packets. Thus we have to count on that problem and this option is not acceptable. That leaves us with the two other choices. We argue that the behavior by which we try several times more and that report an error is an optimal one. The argument is that it's unnecessary to implement DHCP fail over logic in ikev2 when DHCP itself has fail over capabilities. The consequence of this decision is that state REBINDING is removed from the original state machine. Namely, the purpose of REBINDING state is to find other DHCP servers capable of providing configuration data but, which is not necessary any more.

Upon receiving DHCPOFFER message, the state machine makes transition into REQUESTING state while in the same time sending DHCPREQUEST to the DHCP server. In REQUESTING state ikev2 is waiting for acknowledgment from the DHCP server. While in the REQUESTING state any additional DHCPOFFER messages are discarded. When DHCPACK is received, state machine makes transition into BOUND state and all the lease times are recorded. When T1 expires we send DHCPREQUEST to try to renew a lease, and transition into RENEWING state where we wait for DHCPACK after which, we go again into state BOUND. Note that, in RENEWING state, we immediatelly make transition into ERROR state in case no response is received, or, negative acknowledgment is received.

It's obvious from the preceding description that we do not rely on T2 since we don't support backup DHCPv4 server. It's also worth noting that primary purpose of the lease time is to count for the clients that simply go away and don't release their resources back to DHCPv4 server. If there whould be no lease time, it could happend that there are more available resporces, though there's no client connected to the network. This is not such a problem for the ikev2, as it's always known when the client disconnects. Still, because of the unreliability of UDP and the way DHCPv4 protocol works, it might happen that resources are not released.

Finally, when the client disconnects, ikev2 releases it's resources by sending DHCPRELEASE message and makes transition into FINISHED state. At that moment, the state machine can be removed from the memory by some garbage collection process.

Transitions marked with TIMEOUT1 are activated after timeout expires and new request is sent. TIMEOUT2 occurs after predefined number of times TIMEOUT1 exceeds.

IV. Implementation

Additional states IKE_SMI_COOKIE, IKE_SMI_INVALIDKE

Linux differences from the IPsec architecture spec

In state machines that handle IKE SAs and CHILD SAs there are no states that handle illegal messages, e.g. those that have illegal checksum, or some field in the header. This is because such cases are handled before they reach state machines and they are usually dropped without any further notice. Also, state machines do not have any provision for retransmissions and repeating old responses. This is also handled by the lower level code and thus, such problems do not come to the ikev2 state machines. Only in the case that there is no response after a number of retries, state machines are given TIMEOUT message.

V. Conclusions and Future Work

References

[1] "ikev2," Dec. 2007. [Online]. Available: http://sourceforge.net/projects/ikev2

[2] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol," RFC 4306 (Proposed Standard), Dec. 2005, updated by RFC 5282. [Online]. Available: http://www.ietf.org/rfc/rfc4306.txt

[3] J. Vollbrecht, P. Eronen, N. Petroni, and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator," RFC 4137 (Informational), Aug. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4137.txt

[4] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131 (Draft Standard), Mar. 1997, updated by RFCs 3396, 4361. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt